University of Ferrara
Department of Engineering

Error Control Algorithms for Wireless Communication Networks: Analysis and Performance Evaluation

Michele Rossi

Ph.D. Thesis

March 2004

to Rita

Acknowledgments

Usually, I am not very good at saying thank you. But, given the great effort and sacrifice that I have put in the realization of the research presented in this thesis and the great support that I have received by the many people surrounding me, I think it is my duty to say thanks here to all of them without saving any word.

First and foremost, I would like to thank my family, whose moral support has helped me in facing a lot of difficulties and deflecting my negativity during hard working periods. I am glad to have had all of you so close to me, you all have contributed to the success of my research activity. Among the all of you, I want to say a special thank to Rita, thank you for all your patience and understanding, thank for the many times in which I have had to work during the week-ends and you were always there to support me. Thank you for your love that has always been my main source of strength. Thank to all my friends that have helped me in many manners, maybe not in a technical sense, but surely of a pivotal importance.

Moreover, I would like to thank my Ph.D. supervisor, Prof. Michele Zorzi. Throughout all these years he has provided me with a free exchange of ideas, constructive criticism, encouragement, advise, and moral support, as well as many opportunities for professional growth. I am indebted with him for all of this and for having made of me a better researcher, continuously stimulating my creativity and sense of critic.

I am also most thankful for the opportunity that Prof. Ramesh R. Rao has offered to me of doing research at the University of California, were I have been working for a period of six months during the year 2003. This has been a priceless and fruitful experience that has contributed to improve myself from both a professional and human point of view.

Finally, I would like to thank the many colleagues (and friends) of mine I have shared stimulating technical conversations with, and, in some circumstances, with whom I have spent some of the most beautiful moments of my life. I do not want to mention any name here; this decision is not due to the lack of space but to be fair and do not exclude anyone. The ones of you that will read this paragraph will recognize them self without any doubt in this long unmentioned list.

MICHELE ROSSI

University of Ferrara, Italy
March 2004

Error Control Algorithms for Wireless Communication Networks: Analysis and Performance Evaluation

Michele Rossi, Ph.D.
University of Ferrara, Italy, March 2004

Supervisor: Prof. Michele Zorzi

Abstract

This thesis contains an accurate selection of the research activities I performed as a Ph. D. candidate. The main focus of my work was on higher layer protocols such as TCP/IP [84] and the characterization of link layer techniques that are used in modern wireless communication systems to counteract the residual errors affecting wireless channels that are still present even after physical layer processing. In modern telecommunications systems, such as the UMTS [1], link layer techniques are coupled with higher layer TCP/IP transmissions to improve performance and to solve some problems affecting protocols as TCP when they are operating over a wireless medium. These techniques are also gaining interest in modern mobile satellite architectures (see Chapter 3 for a better introduction and a detailed list of references), where the wireless channel is impaired by errors due both to the user mobility and to atmospheric phenomena such as rain and fog. For these reasons, it is important to gain precise indications on the way in which link layer and TCP interact and on the correct setting to achieve optimal performance depending on the underlying channel error process. The thesis is dedicated to this understanding as well as to the application of these two techniques to different scenarios and problems. The results presented in the following chapters have been obtained by a mixture of simulation and analytical techniques.

The thesis is organized in such a manner that it presents the results to the reader following an increasing level of complexity. The study is started by considering high level retransmission policies (i.e., TCP), and by pointing out their inefficiencies over the wireless links. Subsequently, lower layer recovery schemes (algorithms operating at the link layer) are considered in order to cope with these inefficiencies, and the performance is characterized considering both approaches working together. In the last part of the thesis, the focus is shifted towards the characterization and optimization of link layer algorithms only.

The document is subdivided in two main parts. In the first part (Chapters 1–4), the TCP protocol is addressed by studying its performance over several networks both by simulation and analysis. In the first Chapter, the TCP algorithm is investigated in a W–CDMA cellular environment, where the achievable performance is derived as a function of the system load and other user parameters (such as mobility and spreading factor, i.e., user bit rate). Here, the TCP protocol is the only mechanism which is responsible for the retransmission of corrupted data. In Chapter 2, a novel TCP header compression algorithm is proposed to save bandwidth, thereby improving the transmission efficiency of the TCP flows in a W–CDMA cellular system. However, as it will be highlighted in Chapter 2, this mechanism alone is not able to completely cope with the inefficiencies of TCP over wireless links. Further solutions to solve these problems are presented in Chapter 5. Later on, the focus is moved to the satellite environment (Chapter 3), where TCP is used together with other protocols to achieve optimal performance. In this scenario, TCP is considered together with other recovery algorithms. For instance, a special retransmission scheme was designed for the satellite link. However, only one retransmission protocol is considered at one time, i.e., only one retransmission protocol is operating over a single link. Next, in Chapter 4 the TCP algorithm is studied analytically following a semi–Markov approach. The aim of this Chapter is to provide an accurate understanding of the mechanisms involved in various versions of TCP and to point out how the TCP performance depends on the underlying channel statistics.

In the second part of the thesis, the focus is being put on lower layer retransmission mechanisms (mechanisms operating at the link layer) and on their interaction with higher layer protocols (i.e., TCP). This second part starts with Chapter 5, with the characterization of the performance of TCP operating in a W–CDMA cellular environment, but considering a link layer retransmission scheme to cope with the inefficiencies of TCP over the wireless link. In this Chapter several useful guidelines for an optimal design of this coupled system (TCP plus link layer algorithm) are derived. In Chapter 6, the focus has been put on retransmission schemes only, by providing a complete theoretical framework in order to derive link layer delay statistics. These delay statistics are useful since they can be used to estimate several performance metrics at higher layers (e.g., the TCP timeout event probability). Finally, in Chapter 7 we exploit link layer algorithms to improve the performance of the multicast data transmission in W–CDMA cellular systems. In this case, particular attention has been paid to the video streaming case. In the following, a more detailed summary of the results contained in every Chapter is given.

Overview

The thesis starts with Chapter 1 with the performance evaluation of the TCP/IP protocol suite operating in a multi-cellular system, where W–CDMA (Wideband CDMA) [83] is considered as the access technique used over the air interface. In this Chapter, the main focus is on the characterization of how the user interference affects the TCP/IP performance. In particular, this investigation is carried out as a function of various system parameters such as the user Doppler Frequency (a parameter directly related to the user speed, i.e., to the fast fading propagation phenomenon) the system load and the TTI (Time Transmission Interval), i.e., the reference time period over which the coding and interleaving processes are performed at the physical layer. Heuristics are derived to approximate the TCP/IP throughput as a

function of the above cited parameters.

In Chapter 2, Header Compression (HC) techniques are introduced to improve the TCP/IP flows performance over wireless channels. In more detail, HC is employed to reduce the size of the transmitted TCP/IP headers. This size reduction, results in a saving of the bandwidth required to transmit the TCP traffic and in an improvement of the response time for interactive sessions. Here, new algorithms are proposed to improve the HC robustness against wireless channel errors. The performance of these algorithms is tested again in the cellular scenario considered in Chapter 1.

In Chapter 3, the TCP/IP protocol performance is investigated in a satellite environment. In particular, a scenario including geostationary satellite links is considered, and a set of techniques is introduced to improve the performance of end-to-end TCP/IP sessions in such cases. These techniques include some architectural modifications of the existing architecture as well as the introduction of a new protocol operating over the wireless link. This protocol is based on a Selective Repeat [82] algorithm to which some new features have been added to allow for flow control between the wired and the wireless portions of the network (the Internet backbone and the satellite link). Performance evaluation is carried out by both simulation and analysis. The proposed solution is shown to highly improve the throughput efficiency as well as the end-to-end delay in case of error prone satellite links.

The throughput of TCP/IP running over a wireless link is derived in Chapter 4 where a Markov analytical technique is used to describe the behavior of the protocol and derive the throughput curves as a function of the wireless channel characteristics (Doppler Frequency and round trip delay). The analysis presented in this Chapter is quite general and applicable to any wireless channel since the error statistics is modeled by means of a Two-State Markov model [52].

A first characterization of link layer algorithms is carried out in Chapter 5, where the link layer (RLC, Radio Link Control [4]) of a UMTS system is considered. In this Chapter, all the RLC parameters are considered as specified in the standard [1]. The RLC layer is a Selective Repeat based algorithm that is used at the second layer of the UMTS protocol stack to counteract the residual errors that are still present after physical layer processing. The role of RLC is to shield higher layers from these errors so as to avoid the triggering of some error recovery mechanisms that, in some cases, could lead to a substantial decrease in performance (i.e., in the TCP case to avoid a spurious occurrence of the timeout mechanism). The performance evaluation is carried out considering higher layer metrics such as TCP/IP throughput and delay. Indications on the correct RLC parameter settings are derived as a function of the size and the typology of traffic considered at higher layers.

In Chapter 6, the delay of Selective Repeat algorithms is investigated in great detail by deriving delay statistics over independent (iid) and correlated channels. The results obtained in this Chapter are analytical and applicable to a large set of systems. In the last part of the Chapter, the delay statistics are derived modeling the wireless channel as an N–State Markov model so as to obtain a good level of accuracy with respect to the simulation results obtained over a Rayleigh fading channel. In fact, this channel is often

not very well approximated by the Two-State Markov Model introduced and used in Chapter 4.

Finally, in Chapter 7 some solutions to improve the performance of multicast streaming in 3G cellular systems are proposed. In UMTS, for instance, this service is referred to as MBMS [1] (Multimedia Broadcast/Multicast Service). In that Chapter, the best way to manage the available resource to handle the multicast traffic is investigated first; the focus here is on the best way to allocate channel resources to provide such a traffic. This point is discussed both qualitatively and by means of simulation results. Subsequently, once the correct channel has been established, some new Hybrid ARQ (HARQ) schemes are presented to improve the performance of the multicast sessions over standard ARQ mechanisms. These new schemes and their advantages are then investigated in detail.

Contents

Chapter 1

Performance Evaluation of TCP/IP over a W–CDMA Wireless Air Interface

In this Chapter, a study on the performance of TCP in terms of both throughput and energy consumption in the presence of a Wideband CDMA radio interface typical of third generation wireless systems is presented. The results show that the relationship between throughput and average error rate is largely independent of the network load, making it possible to introduce a universal throughput curve, empirically characterized, with gives throughput predictions for each value of the user error probability. Furthermore, the study of the energy efficiency shows the possibility to select an optimal power control threshold to maximize the tradeoff between throughput and energy, thereby potentially achieving very significant energy gains.

1.1 Introduction

In the past decade, the introduction of mobile phones and the rate of subscription to wireless communication systems have been spectacular. The access to the Internet is becoming part of everyday's life for a large number of people throughout the world. Even though voice applications will still dominate the wireless market in the immediate future, everybody expects that the real future of wireless is in IP-based data applications, which will deliver the promise of easy access to large amounts of information for anyone at any time and from anywhere [62].

Based on this evolutionary trend of the telecommunications market, a significant step has been taken by the wireless industry, by developing a new generation of systems whose capabilities are intended to considerably exceed the very limited data rates and packet handling mechanisms provided by current second generation systems, such as GSM, IS-136, IS-95 and PDC. Third generation systems are already at the advanced stage of standardization [55]. For example, a self-contained set of specifications for the Universal Mobile Telecommunications System (UMTS, the European/Japanese version of third generation wireless) has already been published last year, and a new release is expected soon, based on an all-IP Core Network architecture [1, 87, 95].

The goal of providing multimedia services to the wireless terminals and of offering transport capabilities based on an IP backbone and on the Internet paradigm calls for the extension of widespread

Internet protocols to the wireless domain. In particular, extension of the Transmission Control Protocol (TCP) [97] has received considerable attention in recent years, and many studies have been published in the open literature, which address the possible performance problems that TCP has when operated over a connection comprising wireless links, and propose solutions to those problems (see for example [17, 25, 28, 30, 68, 70, 106]).

When TCP is run in a wireless environment, two major considerations must be made regarding its performance characterization. First of all, wireless links exhibit much poorer performance than their wireline counterparts and, even more importantly, the effects of this behavior are erroneously interpreted by TCP, which reacts to network congestion every time it detects packet loss, even though the loss itself may have occurred for other reasons (e.g., channel errors). Furthermore, it has been shown that the statistical behavior of packet errors has a significant effect on the overall throughput performance of TCP, and that different higher-order statistical properties of the packet error process may lead to vastly different performance even for the same *average* packet error rates [113]. It is therefore important to be able to accurately characterize the actual error process as arising in the specific environment under study, as simplistic error models may just not work.

Another critical factor to be considered when wireless devices are used is the scarce amount of energy available, which leads to issues such as battery life and battery recharge, and which affects the capabilities of the terminal as well as its size, weight and cost. Since dramatic improvements on the front of battery technology do not seem likely, an approach which has gained popularity in the past few years consists in using the available energy in the best possible way, trying to avoid wasting power and to tune protocols and their parameters in such a way as to optimize the energy use [2]. It should be noted that this way of thinking may lead to completely different design objectives or to scenarios in which the performance metrics which have traditionally been used when evaluating communications schemes become less important than energy-related metrics. This energy-centric approach therefore gives a different spin to performance evaluation and protocol design, and calls for new results to shed some light on the energy performance of protocols.

Studies on the energy efficiency of TCP have been very limited so far [100, 101, 111, 114]. In addition, they do not address specifically the Wideband CDMA environment typical of third generation systems, and therefore do not necessarily provide the correct insight for our scenario. The purpose of the results reported in the next is to provide a detailed study of the performance of TCP, in terms of both throughput and energy, when a Wideband CDMA radio interface is used. In particular, the parameters of the UMTS physical layer will be used in the study. As a first step, in this Chapter we present results in the absence of link-layer retransmissions. This is done in order to more clearly understand the interactions between TCP's energy behavior and the details of the radio interface. Results on the throughput performance of TCP in the presence of link-layer retransmissions will be presented in details later on in Chapter 5 (see [28, 30, 63] for some contributions on this topic).

The Chapter is organized as follows. In Section 1.2, the main system assumptions and some details of the simulation tool are described. In Section 1.3, the throughput performance of TCP in various scenarios is discussed, and an analytical approximation for the TCP throughput performance curves is proposed. In Section 1.4, the energy performance of TCP is directly addressed. In particular, the effect of the power control threshold, which affects both the error statistics and the transmit power consumption, is examined,

and the tradeoff between QoS and energy efficiency is explored. A main conclusion is that an appropriate choice of the power control threshold, while leading to incremental (though non-negligible) throughput improvements, has the potential to produce multiple-fold energy savings.

1.2 System Model

1.2.1 System–Level Simulation

In order to carry out the proposed research, an accurate simulation tool has been developed. It simulates the operation of a multicellular wideband CDMA environment, where user signals are subject to propagation effects and transmitted powers are adjusted according to the power control algorithm detailed in the specifications [1].

We consider a hexagonal cell layout with 25 cells total. This structure is wrapped onto itself to avoid border effects. Each simulation is a snapshot in time, so that no explicit mobility is considered while running the simulation. However, the fact that users may be mobile is taken into account in the specification of the Doppler frequency, which characterizes the speed of variation of the Rayleigh fading process. For the same reason, long-term propagation effects, namely path loss and shadowing, are kept constant throughout each simulation. Path loss relates the average received power to the distance between the transmitter and the receiver, according to the general inverse-power propagation law $P_r(r) = Ar^{-\beta}$, where in our results we chose $\beta = 3.5$. Shadowing is modeled by a multiplicative log-normal random variable with dB-spread $\sigma = 4$ dB, i.e., a random variable whose value expressed in dB has Gaussian distribution with zero mean and standard deviation equal to 4 [61].

At the beginning of each simulation, all user locations are randomly drawn with uniform distribution within the service area, and the radio path gains towards all base stations are computed for each. Users are then assigned to the base station which provides the best received power. Such assignment does not change throughout the simulation. Also, note that each user is assigned to a single base station, i.e., soft handover is not explicitly considered here. Extension of the program to include soft handover is currently in progress, although qualitatively similar results are expected.

During a simulation run, the fading process for each user is dynamically changed, according to the simulator proposed by Jakes [61] and to the selected value of the Doppler frequency. Note that the instantaneous value of the Rayleigh fading does not affect the base station assignment. In order to take into account the wideband character of the transmitted signals, a frequency-selective fading model is used, with five rays whose relative strengths are taken from [36]. Maximal ratio combining through RAKE receiver is assumed at the base station. Only the uplink is considered here, although similar results have been obtained for the downlink as well.

Each connection runs its own power control algorithm as detailed in the specifications. The resulting transmitter power levels of all user, along with the instantaneous propagation conditions, determine the received signal level at each receiver, and therefore the level of interference suffered by each signal. We assume here perfect knowledge of the Signal-to-Interference Ratio (SIR) which is used to make the decision about whether the transmitted power should be increased or decreased by the amount Δ. A finite dynamic range is assumed for the power control algorithm, so that under no circumstances can the transmitted power be above a maximum or below a minimum value. The delay incurred in this update is

PARAMETER	VALUE
Cell Side	200 m
β (path loss model)	3.5
A (path loss model)	-30 dB
Max. TX Power	-16 dBW
Power Range	80 dB
σ (shadowing)	4 dB
Time Unit (physical slot duration)	0.667 ms
Number of Oscillators (Jakes)	8
n_rays (Selective Channel)	5
Chip Rate	3.84 Mcps
Data Rate	240 kbps
SF (Spreading Factor)	16
Δ (Power Control Step)	0.5 dB
Noise	-132 dBW

Table 1.1: System-level simulation parameters

assumed to be one time unit (given by the power control frequency of update), i.e., the transmitted power is updated according to the SIR resulting from the previous update. The effect of late updates on the overall performance has been studied in [47].

Table 1.1 summarizes the various parameters used in the simulation.

1.2.2 Block Error Process Generation

The output of the system level simulations is a log of the values of the SIR, transmitted power and fading attenuation for all users, which allows us to gain some understanding on the time evolution of these quantities, as well as on the behavior of the network at the system level. A post-processing package translates the SIR traces into sequences of block error probabilities (BEP). This is done while taking into account how the radio frames are deinterleaved and decoded. The interleaving schemes have been taken from the specifications, and a convolutional code with rate $1/2$ and constraint length 8 with Viterbi decoding has been considered. An analytical approximation has been used to relate a string of SIR values (one per time unit) to the probability that the corresponding block (transmitted within one or more radio frames) is in error. The resulting trace of the block error probabilities can then be used in the simulation of higher-layer protocols, as is done here, or to perform some statistical analysis of block errors. The latter approach is explored in [107], where the burstiness of the error process is investigated.

1.2.3 TCP Simulation

For each simulation run (which corresponds to a given set of parameters), the SIR traces of all users are produced, which are then mapped into BEP traces as explained above. The latter are then used to

randomly generate a block error sequence (BES). The BES is generated from the BEP traces by just flipping a coin with the appropriate probability in each Time Transmission Interval (TTI), which is the time used to transmit a block. The BES obtained is then fed to the TCP simulator that uses it to specify the channel status in each TTI. By doing so, the SIR traces generated for all users in a simulation run are used by the TCP simulator to compute the throughput.

The average throughput is defined as the fraction of the channel transmission rate (considered at the IP level output) which provides correct information bits at the receiver, i.e., not counting erroneous transmissions, spurious retransmissions, idle time and overhead. In addition to the throughput value, the TCP simulator computes other metrics of interest. In particular, we are interested in the average block error probability, P_e, which is obtained for every user simply by averaging all the values of its BEP sequence, and which will be used later to report the obtained results. Another metric of interest is the average energy spent in transmitting data. For this purpose, besides BEP, we have also considered the *transmit power* traces generated by the system-level simulator. These traces have been generated assuming continuous channel transmission (transmitter is always active) and by updating the transmitted power according to the power control algorithm. However, when TCP is considered, transmission is bursty, due to the window adaptation mechanism implemented by the TCP algorithms, i.e., idle times occur when the window is full or the system is waiting for a timeout. To account for idle times, we have considered the *actual* transmitted power, equal to the one obtained from power traces when TCP transmits, and to zero otherwise. The average transmitted power is then computed by summing the actual transmitted power throughout the simulation (all slots) and dividing it by the total simulation time (in number of slots). Moreover, to obtain the average consumed energy per correctly received bit, we simply divide the average transmit power by the correct information bit delivery rate. In the following, we report the throughput and average consumed energy expressions

$$S = \frac{CIB}{CBR} \tag{1.1}$$

$$ACE = \frac{ATP}{CIB} = \frac{ATP}{S \cdot CBR} \tag{1.2}$$

where S=average throughput, CIB=correct information bits per second, CBR=channel bit rate at the IP level output, ACE=average consumed energy per bit, and ATP=average transmitted power. We remark that, with our definitions, ACE is the average consumed energy per *correctly received bit*, i.e., the energy cost of delivering a single bit to the destination. Notice that this quantity is equal to the inverse of the *energy efficiency* of the protocol as defined in [108].

The TCP simulator implements fragmentation of TCP segments, window adaptation and error recovery. It simulates a simple unidirectional FTP (File Transfer Protocol) session, where the direct link packet generation is assumed continuous as in a long FTP transfer. The TCP algorithm considered is *New Reno* [44]. Data flow is unidirectional, i.e., data packets flow only from sender to receiver, while ACKs flow in the reverse direction. Receiver generates non-delayed ACKs, i.e., one ACK is sent for each packet received. The TCP/IP stack is version 4, with a total of 40 bytes (including both TCP and IP overhead) for each header (compression is not taken into account in the presented results, its effect will be investigated in Chapter 2) and MTU size of 512 bytes. In the results presented in this Chapter, RLC and MAC levels

are assumed to operate in transparent mode. Detailed results on the RLC impact will be presented later on in Chapter 5.

To compute the bit rate at the output of the IP level, we have to account for overhead added by the physical layer as well as possibly due to multiplexing of other channels. For the purpose of discussion, we assume the following figures: a transport block is 1050 bits, including 16 bits of overhead due to transparent RLC/MAC operation; a CRC and tail bits for code termination are added to this block, and the result is convolutionally encoded at rate $1/2$. The resulting encoded block is then brought to 2400 encoded symbols by the rate matching algorithm, so that the raw physical layer symbol rate is 240 kbps. The application of a spreading factor $SF = 16$ makes it 3.84 Mcps, which is the standard channel transmission rate [1]. At the IP level output, we then have a block of 1034 bits of data every 10 ms, thereby yielding a net bit rate of 103.4Kbps, which is the bit rate used in the TCP simulator.

The use of TCP New Reno algorithm has been motivated by its implementation of fast recovery and fast retransmit algorithms [14], as recommended in [32], especially for wireless environments. This is an optimization over previous TCP versions.

1.3 TCP Throughput Performance

In this Section, we present some numerical results which illustrate the behavior of TCP throughput in the considered environment.

In the graphs presented we will indicate with N_u the number of users in the simulation, with TTI the number of radio frames over which interleaving is performed, $SIR_{th} = t + \Delta$ dB indicates that the threshold used in the power control algorithm is t, while Δ is its increment as described in Section 1.2. Finally, with the term f_d we refer to the Doppler frequency used in the Rayleigh fading simulator. In the following graphs, the results will be represented as average TCP throughput, S, vs. average block error probability, P_e, thereby assigning a single point in the graph to each user.

1.3.1 Sensitivity to the Doppler Frequency

Fig. 1.1 shows the TCP throughput performance for different values of the Doppler frequency. The graph is plotted by reporting throughput vs. P_e for each of the 90 users involved in the simulation; each user is identified by a marker. The case of independent errors is also reported for comparison purpose (here the markers are used only to identify the curve and are not related to the users). The first interesting observation concerns the fact that, for a given value of the Doppler frequency, f_d, the points representing the various users of a simulation appear to lie along a fairly well-defined curve. It is worth stressing that this was not obvious a priori since different users are placed in different locations and are subject to different propagation conditions, both in terms of slow impairments (log-normal shadowing) and in terms of fast fading. This allows us to introduce the concept of "universal throughput curve" for a given situation, in the sense that users which suffer similar values of P_e will enjoy about the same throughput. Again, this is not obvious since different users in a simulation may see different statistical behaviors of the errors, which could in principle lead to different performance even in the presence of the same average error rate [113]. An explanation can be drawn from the results in [107], where it was found that for a given value of f_d there is a strong correlation between P_e and the error burstiness. In this situation, P_e

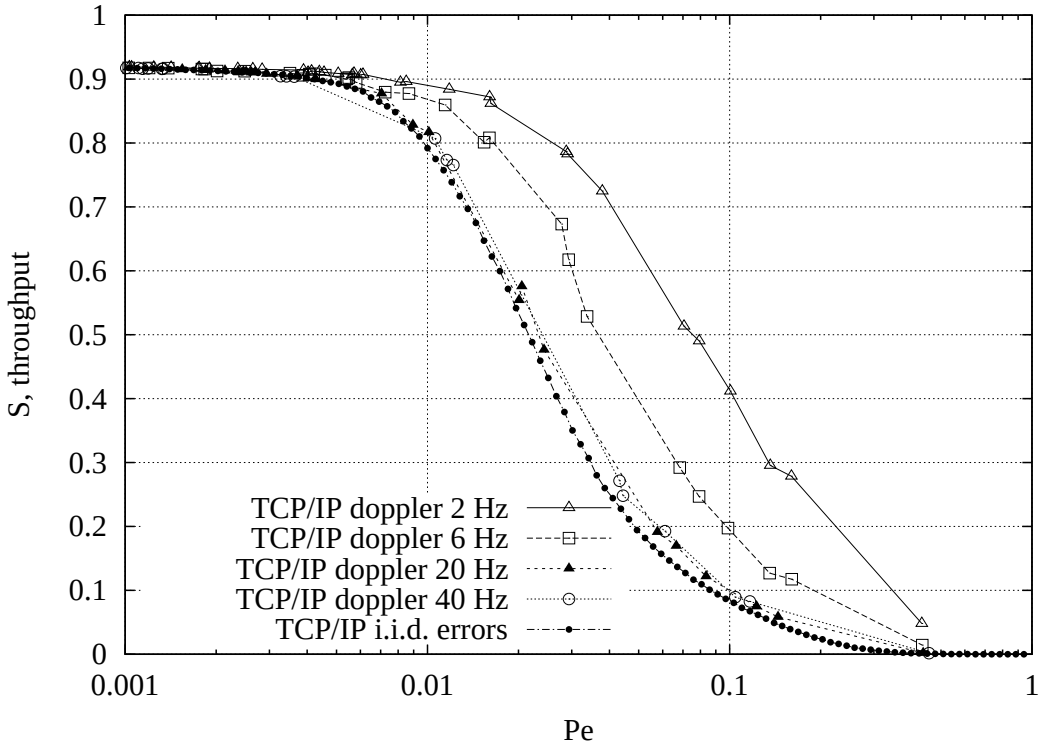


Figure 1.1: TCP/IP Sensitivity to Doppler, ($N_u = 90$, $TTI = 1$, $SIR_{th} = 5.5 + 0.5$dB, $f_d = 2, 6, 20, 40$Hz).

essentially determines the full second-order characterization of the error process, which in turn almost fully specifies the value of the TCP throughput. On the other hand, for different values of f_d we observe different curves. In fact, even in the presence of the same P_e the different extent of the channel memory results in different performance.

Another interesting observation from Fig. 1.1 is that as the Doppler frequency increases the performance degrades, i.e., slower channels correspond to better performance, as already observed in [106]. As expected, for sufficiently high values of the Doppler frequency, the behavior of the system is close to the iid case. Finally, we note that the shape of the curves appears to be fairly regular, with a smooth transition from highest throughput values (essentially limited only by the percentage of overhead in the TCP packets, far left of the graph), to essentially zero throughput when errors are very likely (right end of the graph). This shape, which has been observed by other authors, lends itself nicely to numerical fitting, as detailed later.

1.3.2 Sensitivity to Interleaving Depth

In UMTS, besides the so-called *intra-frame* interleaving, which is always used to scramble the bits within a radio frame (10 ms) before encoding, it is also possible to use a second, *interframe*, interleaving,

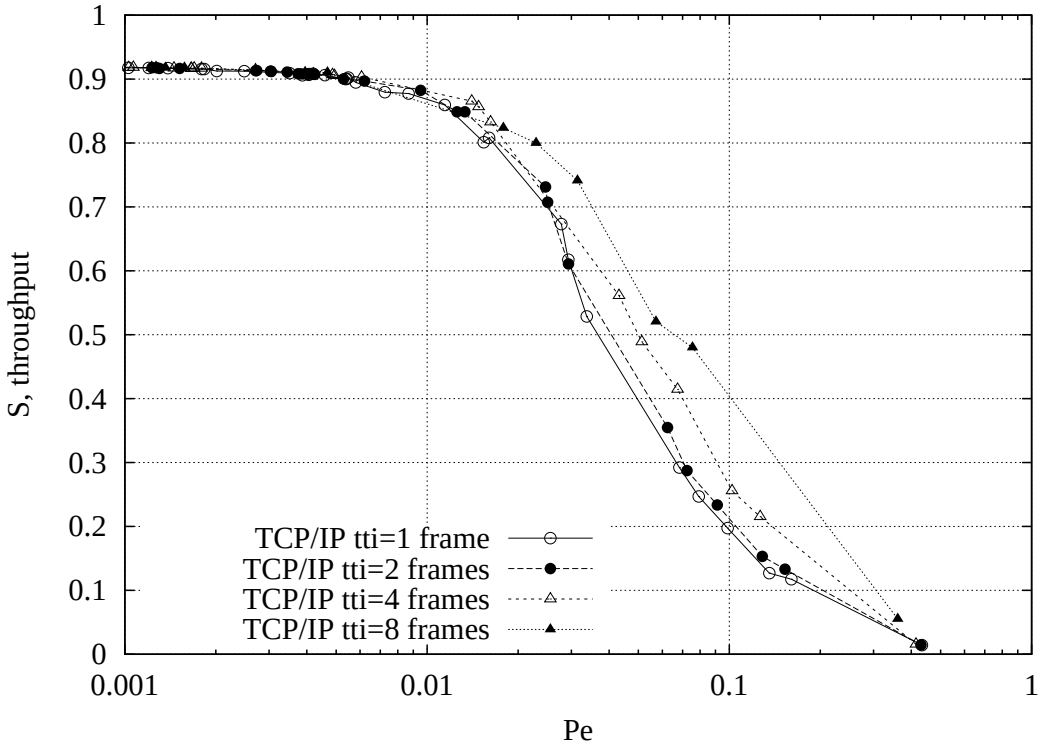
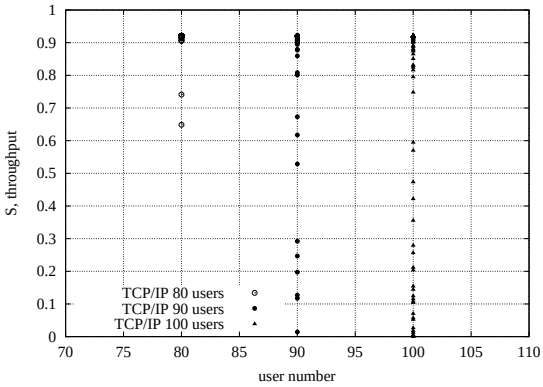


Figure 1.2: Average throughput vs TTI, ($N_u = 90$, $TTI = 1, 2, 4, 8$, $SIR_{th} = 5.5 + 0.5$dB, $f_d = 6$Hz).

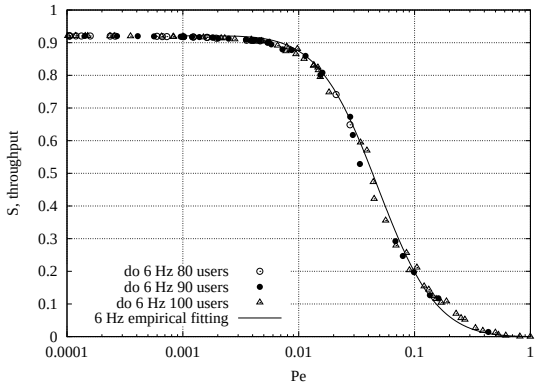
which mixes bits *across* frames. By doing so, the performance of the decoder is of course improved also in the presence of burst errors, but a larger interleaving delay is introduced. Therefore, another interesting sensitivity analysis regards the interleaving span allowed by the application. While keeping in mind the price to be paid in terms of delay (an increase of the TTI corresponds to a larger delay), we can note from Fig. 1.2 how a deeper interleaving gives a beneficial effect. Notice also that the shape of the curves is similar to what observed in the previous case.

1.3.3 Effect of the Network Load

The effect of the network load is shown in Figs. 1.3(a) and 1.3(b). In Fig. 1.3(a), results from three simulations are shown, with 80, 90 and 100 users in the network, respectively. For each simulation, all users are assigned a point with the same abscissa (which is given by the number of users in the system) and with vertical coordinate given by their average TCP throughput. We can see how for increasing load the presence of disadvantaged users becomes more noticeable, as expected, and the system is more and more unfair. The same results are represented in Fig. 1.3(b) by reporting the throughput as a function of P_e where the curve along which the various points are aligned is *relatively insensitive to the system load.* In Fig. 1.1, it was observed that, in a given scenario, two users whose average block error probability is the same will have essentially the same throughput, regardless of the specific situation of each. What we see here is that this behavior still holds *across* simulations in which different levels of network load are considered (but for the same Doppler frequency). For higher network load, each user will certainly see worse performance due to the increased interference, but the relationship between average throughput and



(a) TCP/IP average throughput vs network load, ($N_u = 80, 90, 100, TTI = 1$,$SIR_{th} = 5.5$+0.5dB, $f_d = 6$Hz).



(b) Throughput curve, independence from network load with Doppler 6Hz, ($N_u = 80, 90, 100$, $TTI = 1$, $SIR_{th} = 5.5 + 0.5$dB, $f_d = 6$Hz).

Figure 1.3: Impact of the network load.

average error rate is essentially unaffected. This highlights the power of the concept of "universal curve" which can be used to study typical cases and to infer TCP throughput performance based only on easily measurable physical layer parameters. In Fig. 1.3(b) a possible form for the universal curve is given for comparison with the simulator output points; more details about this fitting function will be presented below. Similar behavior has been observed for other values of the Doppler frequency.

1.3.4 Analytical Throughput Prediction

The observed shape of the throughput curves, which tend to a constant equal to one minus the percentage of overhead for $P_e \to 0$ and to zero for $P_e \to 1$, suggests a numerical fit involving the logarithm of P_e. Also, the shape of the transition is seen to depend on the value of the Doppler frequency. In Fig. 1.4 we propose a function for the modeling of the TCP throughput behavior through a heuristic function $f(x)$, independent of the network load and parameterized only by the Doppler frequency, as suggested by the curves of Fig. 1.3(b) as well as by other results not reported in the thesis. The proposed expression for $f(x)$ is as follows

$$f(x) = S(0) \cdot \frac{10^{\alpha \cdot \ln(\frac{1}{\tilde{x}-x_s}-1)}}{10^{\alpha \cdot \ln(\frac{1}{\tilde{x}-x_s}-1)} + 1} \tag{1.3}$$

where

$$\tilde{x} = 1 + \frac{\log_{10}(x)}{3} \tag{1.4}$$

$$\alpha = 1.3 \tag{1.5}$$

$$x_s = \frac{1}{Af_d + B} - k \tag{1.6}$$

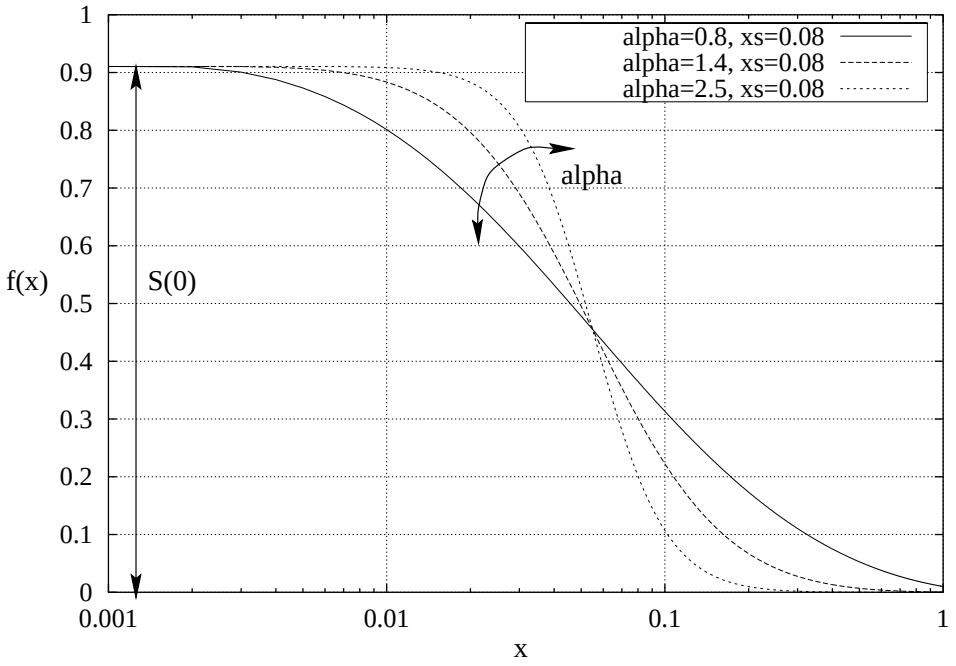


Figure 1.4: Throughput heuristic function.

and where $A = 1.39$, $B = 2.78$, $k = 0.03$, and $S(0)$ is the average throughput for $P_e = 0$ and f_d is the Doppler frequency in Hz.

The accuracy of the proposed fit has been tested for various values of the parameters involved. Examples of these tests are given in Figs. 1.3(b) and 1.5, in which the fitting expression is compared against the simulation results for two values of the Doppler frequency. These graphs show that the analytical expression is reasonably close to the actual points obtained by simulation.

1.4 TCP Energy Performance

All previous results were obtained for a given value of the power control threshold, SIR_{th}, which is used to drive the transmit power dynamics at each user and which directly affects the error performance. In fact, choosing a higher value of this threshold has the double effect of forcing the users to transmit more power in order to achieve a higher SIR (thereby consuming more energy) but also of causing the SIR experienced by the typical user to be higher (thereby improving the error rates and therefore the TCP throughput). It is therefore of interest to study how varying the power control SIR threshold makes it possible to cut a tradeoff between QoS and energy consumption.

Figs. 1.6(a) and 1.6(b) show the trade-off between TCP throughput and consumed energy. Each curve corresponds to a given user for a given set of parameters, whereas different points on the same curve refer to different values of the power control threshold. Fig. 1.6(a) shows the effect of using different threshold values in terms of S as a function of ATP. In this figure only the behavior of some selected users has been reported. These users can be seen as representative of all users in the network, in the sense that they illustrate typical behaviors as they arise in the system. In Fig. 1.6(b) the same results are shown, by

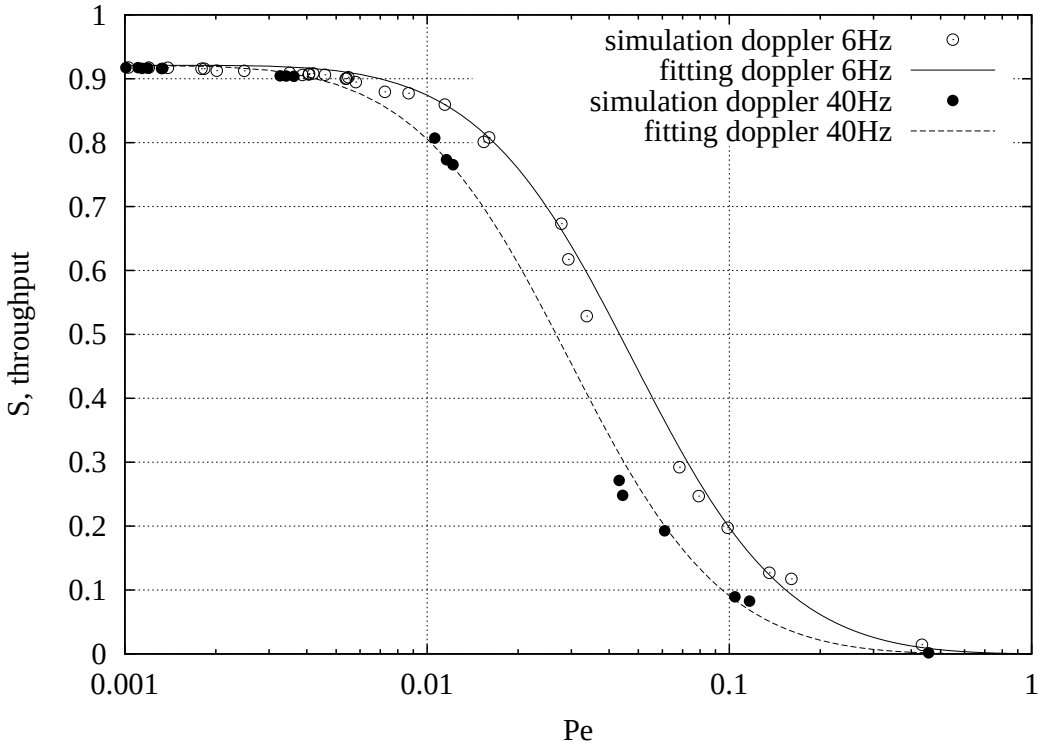
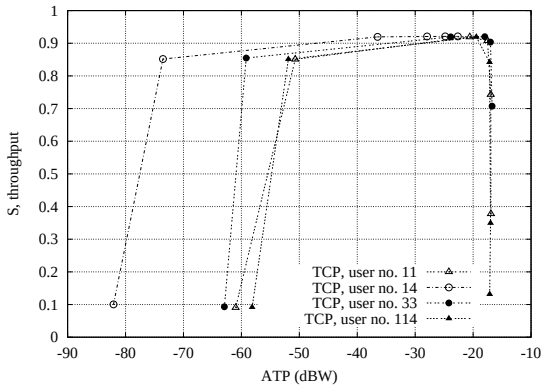
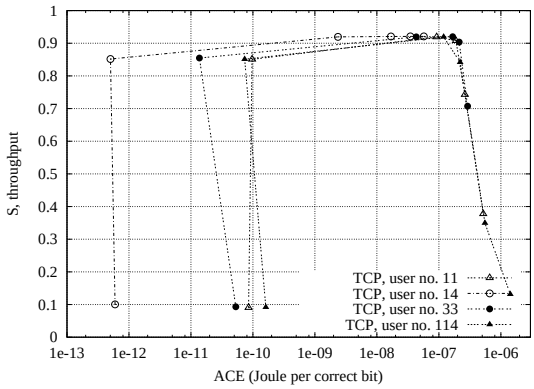


Figure 1.5: Throughput heuristic function: approximation of simulator outputs, ($N_u = 90$, $TTI = 1$, $SIR_{th} = 5.5 + 0.5$, $f_d = 6, 40$Hz).



(a) Throughput vs average transmitted power for different threshold values, ($N_u = 120$, $TTI = 1$, $SIR_{th} = \{1.5, 2.5, 3.5, 4, 4.5, 5\} + 0.5$dB, $f_d = 6$Hz).



(b) Throughput vs average consumed energy for different threshold values, ($N_u = 120$, $TTI = 1$, $SIR_{th} = \{1.5, 2.5, 3.5, 4, 4.5, 5\} + 0.5$dB, $f_d = 6$Hz).

Figure 1.6: Throughput vs power and energy.

considering ACE instead of ATP; as expected, for low throughput the obtained curves are shifted to the right, since ACE is obtained by dividing ATP by the throughput S (except for an inessential constant scaling factor, see equation (1.2)). This is intuitively explained by the fact that, for a given average consumed power, the energy per *correct information bit* is greater for low throughput values, i.e., when it is hard to deliver bits correctly, the cost associated to each one of them is higher. Notice that TCP already does the right thing by stopping transmission when the channel is very bad (the timeout event), whereas it tries to recover from errors whenever possible through retransmissions, which may waste some power.

In general, increasing the threshold should lead to better performance for many users, since the SIR experienced is expected to be higher; this is not necessarily true, however, since, in order to achieve a higher SIR threshold, many users will transmit more power, thereby causing more interference in the system. If the SIR objective is not achievable for all users in the system, some users will actually see degraded performance for higher values of the threshold since, although the threshold value would correspond to better performance, they cannot achieve its value.

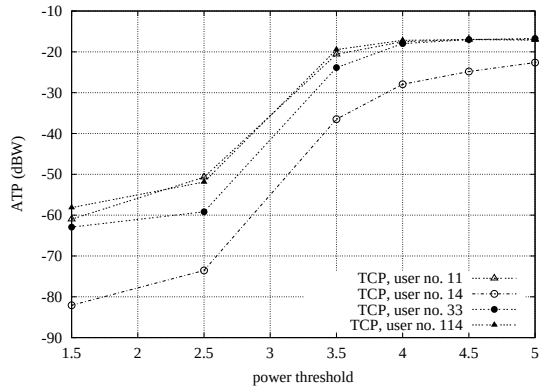
From the obtained results we have noted different users behavior. For some users an increasing power threshold always corresponds to a greater throughput: for these users the throughput as a function of the power threshold is a monotonic curve. For others, the throughput vs. consumed power curve increases up to a breakpoint, after which an increment of the power threshold (and thereby of the transmitted power) actually leads to worse performance, due to greater interference as discussed above. In our results, user 14 is the one showing monotonic behavior, since it experiences favourable propagation conditions, and therefore is not significantly affected by the increased interference level in the system. For users 11, 33 and 114, a different situation can be observed. In particular, user 11 is the one with the worst behavior as the power threshold increases.

In any event, from these results we can conclude that increasing the target value of the SIR in the system does not necessarily translate into improved quality, but there exists an optimal value of the threshold, beyond which some users will experience negligible throughput improvements, whereas others will even see degraded performance. In the cases considered here, this optimal value is seen to be close to 3.5 dB.

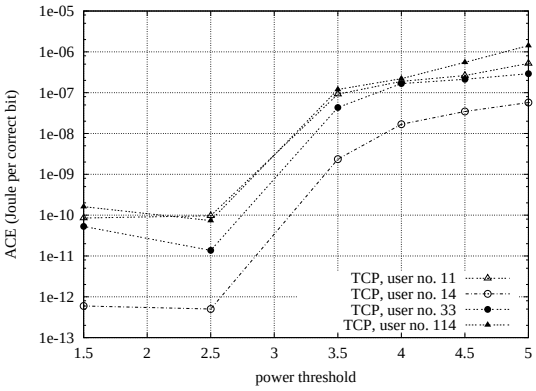
Another important remark regards the numerical values shown in Figs. 1.6(a) and 1.6(b). It can be clearly seen that unlike for throughput, which except for badly chosen values of the threshold exhibits relatively small variations, the range spanned by the energy performance extends over multiple orders of magnitude. This indicates that the choice of the proper power control threshold, while certainly important for error and throughput considerations, becomes critical when energy performance is considered.

Fig. 1.7(a) shows the average consumed power as a function of the power threshold. As expected, a greater threshold value always corresponds to a higher consumed power, i.e., all curves have a strictly monotonic behavior. This is due to the fact that for higher threshold values, more power is necessary in order to obtain the required SIR target. A different behavior is observed in Fig. 1.7(b) in which the consumed energy per bit is reported instead. In particular, notice that in the far left of the graph, a decrease of the threshold, although corresponding to smaller average power (see Fig. 1.7(a)), results in a higher energy cost per bit. This behavior corresponds to users suffering from low throughput performance, where the consumed energy per correct information bit grows, as explained before. As a last observation, we note from Fig. 1.7(a) that users 11, 33 and 114, i.e., the ones that suffer from system interference as the threshold grows, show a transmitted power which is essentially constant for values of the threshold

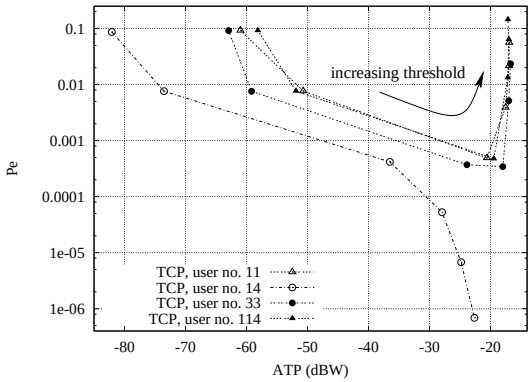
beyond 4 dB, This is due to the fact that these users, in trying to achieve the required SIR and to make up for the increased interference, have reached the maximum allowed value for the transmit power, and therefore their power can not be increased any more.



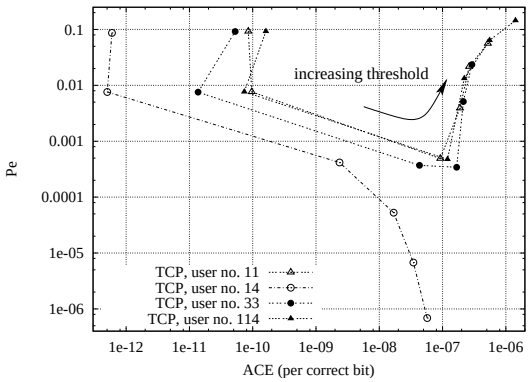
(a) Average consumed power vs power threshold, ($N_u = 120$, $TTI = 1$, $SIR_{th} = \{1.5, 2.5, 3.5, 4, 4.5, 5\} + 0.5$dB, $f_d = 6$Hz).



(b) Average consumed energy vs power threshold, ($N_u = 120$, $TTI = 1$, $SIR_{th} = \{1.5, 2.5, 3.5, 4, 4.5, 5\} + 0.5$dB, $f_d = 6$Hz).



(c) P_e vs average transmitted power for different threshold values, ($N_u = 120$, $TTI = 1$, $SIR_{th} = \{1.5, 2.5, 3.5, 4, 4.5, 5\} + 0.5$dB, $f_d = 6$Hz).



(d) P_e vs average consumed energy for different threshold values, ($N_u = 120$, $TTI = 1$, $SIR_{th} = \{1.5, 2.5, 3.5, 4, 4.5, 5\} + 0.5$dB, $f_d = 6$Hz).

Figure 1.7: Performance as a function of the power control threshold.

A similar QoS-energy trade-off relates to the error rate performance instead of the TCP throughput. In Fig. 1.7(c) the block error probability P_e has been reported against the average transmitted power ATP by using different values of the threshold. From the graph, we note that P_e decreases as the threshold grows until a minimum is reached. After this point P_e starts to grow, again due to the increased interference in the system. The points in which P_e has a minimum are the same on which the throughput of the system is maximized (see Fig. 1.6(a)). As before, user 14 is the only user considered for which P_e never

grows, i.e., increasing the threshold always leads to better performance. In Fig. 1.7(d) ACE is reported instead of ATP and, as in the previous cases, some points are shifted to the right due to poor throughput performance.

From the above results, we may conclude that the selection of the power control SIR threshold is critical in cutting the right trade-offs between QoS and energy performance. In particular, we observe that the potential for energy gains is very significant compared to similar effects on throughput, being measured over multiple orders of magnitude. Therefore, it seems that more attention should be given to energy consumption issues at the Radio Resource Management level, which is responsible for the power control parameter selection.

1.5 Conclusions

In this Chapter, some results on the behavior of TCP over a Wideband CDMA air interface have been reported. In particular, TCP throughput curves have been obtained, and their dependence on various parameters, such as the number of users, the interleaving depth, and the Doppler frequency, has been investigated. From this study, we have found that the relationship between average TCP throughput and average block error rate is largely independent of the number of users in the system. For this reason, it is possible to empirically characterize such a curve with a matching function that only depends on the Doppler frequency. A study of the energy consumption has also been performed, showing that a trade-off between the throughput and the power control threshold exists. It is therefore possible to trade-off QoS of the data transfer for increased energy efficiency. For many users, it has been shown that an optimal value of the threshold exists, potentially leading to very significant energy savings in return for very small throughput degradation. In the considered case, values of the power control threshold close to $3.5 - -4$ dB cut the best tradeoff.

In order to focus exclusively on the interactions of TCP with the WCDMA radio technology and as a preliminary step in characterizing TCP's energy performance in this scenario, no link-layer retransmissions have been considered in the study presented in this Chapter. The impact of a radio link layer on the TCP/IP performance will be persented in detail later on in Chapter 5.

Chapter 2

Enhanced Header Compression algorithms for TCP/IP operating over wireless links

In this Chapter, the existing TCP Header Compression algorithms are first reviewed and then a new proposal, specifically tailored for the wireless channel, is presented.

Header Compression consists in compressing the TCP/IP headers prior to their actual transmission over the channel. These techniques have been studied since 1990 [59] with the original aim of improving performance over slow serial IP links, i.e., by the need for good interactive response time over such lines. Human factor studies have found that interactive response is perceived as "bad" when low-level feedback (character echo) takes longer than 100 to 200 ms. The reduction of header sizes helps, over such slow links, to keep the response time to a reasonable level and, this is achieved by limiting, as much as possible, the amount of redundancy to be sent over the transmission medium. The reader is referred to [59] and to the references therein for a detailed discussion on the implications of transmitting full IP headers over slow serial links.

Recently, these techniques have gained new attention in the wireless communication world. Taking as an example 3G cellular systems [1], these algorithms would indeed allow a more efficient usage of the limited available bandwidth. Bandwidth saving, in such scenarios, turns out in an increased system capacity as well as improved performance perceived by the users that, due to header compression, can experience lower interactive delays as well as higher throughputs.

In the past few years, different compression techniques have been proposed and tested. These algorithms have been mainly developed to operate over the wired Internet, where they present good performance. However, over wireless links, which are usually error prone, such techniques often fail. In more details, due to the error proneness of the communication medium, such techniques could experience a loss of synchronization (**out-of-synch**) between the transmitter and the receiver. These *out-of-synch* periods cause the receiver to temporarily drop packets, since it is unable to correctly decode their compressed headers. As a consequence, the throughput in the presence of errors can be lower than the one characterizing a plain TCP/IP flow, i.e., a flow to which header compression is not applied. For these reasons, it is

clear that new solutions have to be found for a successful integration of compression algorithms in systems including wireless links or, in general, over error prone channels. The aim of the research presented in this chapter is indeed to propose HC schemes able to deal with the errors occurring over a wireless link and to avoid the out-of-synch problem.

In this chapter, a discussion and a brief explanation of the mechanisms governing the existing compression algorithms is given first. Then, a new TCP/IP header compression scheme for wireless networks is proposed and tested. Header compression schemes are investigated here over a WCDMA air interface using the error traces discussed in Chapter 1. The results are encouraging. The proposed algorithm is able to trade-off some bandwidth saving for a greater robustness against the out-of-synch. This obviously comes at the expense of some additional complexity.

2.1 Introduction

The wide diffusion of Internet and wireless networks leads to the integration of these two technologies to provide wide-area wireless Internet access. TCP/IP [97] is probably the most important protocol stack used over a wide range of different physical networks.

First of all, note that compared with fixed and wired networks, wireless networks have higher bit-error rate (BER) and, due to the limitations characterizing most of the modern wireless systems, offer less available bandwidth with respect to their wired counterpart. Furthermore, in the wireless case host mobility can also result in packet loss or delay during handoff procedures. Hence, the performance of TCP/IP over such networks without any modification would suffer from significant throughput degradations as well as large delays.

One way to efficiently use the available resources is to reduce the size of the protocol headers through TCP/IP Header Compression [59] [34] [33] [89]. These techniques are advantageous both in interactive, i.e., *telnet sessions*, and in continuous applications, i.e., *FTP sessions*. In the former case the packet payload is usually smaller with respect to the latter, but in both cases the header overhead leads to a degradation in terms of throughput. Furthermore, compressing the TCP/IP header sizes reduces packet losses in force of the reduced packet sizes.

In this chapter, we investigate the performance of classic TCP/IP (IPv4) header compression schemes on a WCDMA wireless channel and propose a new compression technique [46] to reduce the **header synchronization loss** problem [59] [34] [33] [89]. The performance has been evaluated by means of the TCP/IP New Reno simulator, taking as input WCDMA channel traces generated according to UMTS standard requirements [1] as explained in Chapter 1. The Radio Link Control (RLC) layer [1] is considered here as operating in transparent mode, while a fixed overhead is added for each packet to simulate the presence of the Medium Access Control (MAC).

The chapter is structured as follows: in Section 2.2 the existing and new proposed header compression schemes are discussed; in Section 2.3 the WCDMA channel and TCP/IP simulator are described; in Section 2.4 simulation results, comparing several compression techniques and varying some significant WCDMA channel parameters, are reported. Finally, in Section 2.5 some conclusions are given.

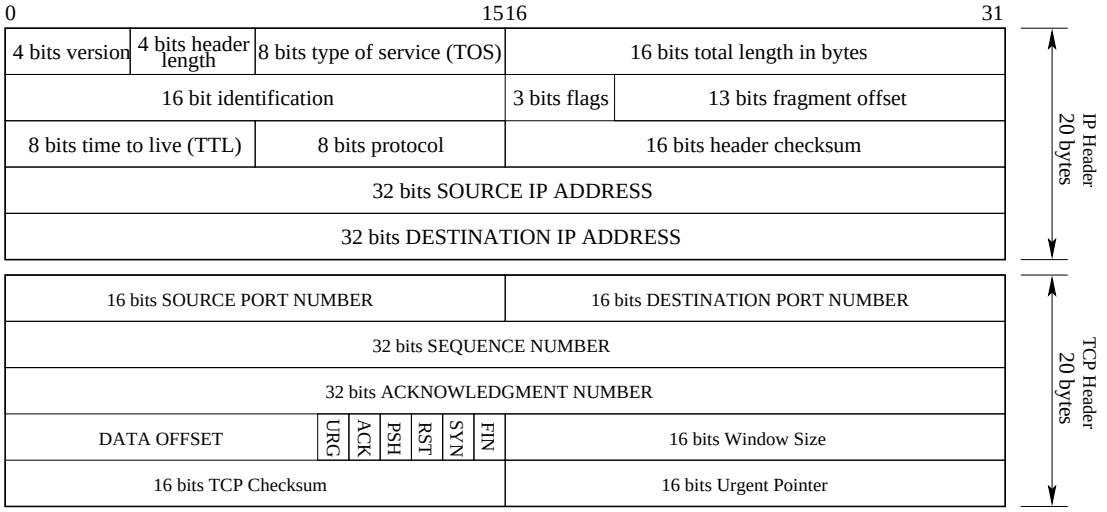


Figure 2.1: The header of a TCP/IP datagram.

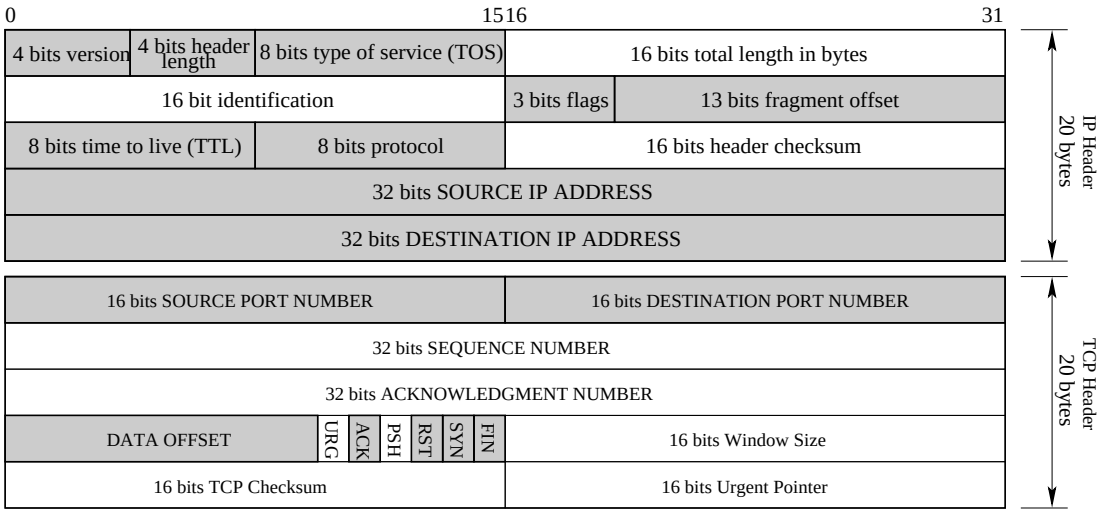


Figure 2.2: Changing fields during a TCP connection.

2.2 Header Compression Schemes

The existing TCP/IP header compression algorithms considered here are: Van Jacobson Header Compression (VJHC) [59], DEgermark Header Compression (DEHC) [34] [33] and PErkins Header Compression (PEHC) [89]. In the next section, we first give a brief review of these schemes to understand in some details the way in which standard compression techniques operate.

2.2.1 Previously Proposed Header Compression Algorithms

VJHC is the first header compression scheme proposed and tested for the TCP/IP protocol stack over wire-line networks. In such environments, this algorithm gives very good performance, allowing a reduction of up to 90 % on the overall header sizes. In the next the VJHC algorithm is first described in its main functionalities without going into the implementation details, but giving an introductory explanation of the main concepts and techniques involved in this header compression scheme. More details, together

with implementation issues, can be found in the standard IETF document [59].

Fig. 2.1 shows a typical (and minimum length) TCP/IP datagram header[1]. The header size is 40 bytes: 20 of IP and 20 of TCP. TCP establishes connections, and roughly half of the TCP/IP header fields are likely to remain unchanged over a connection lifetime. These constant fields are the shaded ones in Fig. 2.2. Hence, if the sender and the receiver keep track of active connections and the receiver gets a copy of the last packet it saw from each connection, the sender gets a factor-of-two compression by sending only a small (≤ 8 bits) *connection identifier* together with the 20 bytes that change and letting the receiver fill the 20 fixed bytes from the saved header. Moreover, one can save a few more bytes by noting that any reasonable link-level framing protocol will tell the receiver the length of a received message, so the *total length* field is redundant and could be neglected as well. In addition, the transmission of the IP *header checksum* field can be avoided as well, since it is useless to protect the transmission of information regarding fields that are not actually transmitted[2]. For this reason, the IP checksum can be neglected and regenerated at the receiver side when the entire header is being reconstructed. This leaves 16 bytes of header information to be sent. All of these bytes are likely to change over the life of the connection, but they do not change all at the same time. For instance, during an FTP file transfer only the *packet ID* (IP identification field), TCP *sequence number* and TCP *checksum* change in the sender $\rightarrow$ receiver direction and only the *packet ID*, TCP *acknowledgment number*, TCP *checksum* and possibly *window size* change in the receiver $\rightarrow$ sender direction. Hence, with a copy of the last packet sent for each connection, the sender can figure out which fields have changed in the current packet, sending a bit-mask indicating the changes followed by the changing fields. Following this procedure, the average header size is decreased to about 10 bytes. However, by looking at how the fields change it is possible to further improve the compression ratio. For instance, the *packet ID* typically comes from a counter which is incremented by one for each packet sent, i.e., the difference between the previous and the current packet ID should be small, usually smaller than 256 and frequently equal to one. For packets belonging to a data transfer, the sequence number in the current packet will be the sequence number of the previous packet plus the amount of data contained in the previous packet (assuming that packets arrive in order). Since IP packets can be at most 64 Kbytes long, the sequence number change must be smaller than 2^{16} (two bytes). So, if the *differences* in the changing fields are sent rather than the changing fields themselves, it is possible to save another three or four bytes per packet. In conclusion, by sending only the information regarding the changing fields in a TCP/IP header and by carefully encoding the *differences* of these changing fields with respect to their previous values, the full header can be substituted by a 5 byte long compressed header, without loss of information. Moreover, recognizing a couple of special configurations it is possible to save, in some cases, two additional bytes, leading to three byte headers. In Fig. 2.3 the above introduced *delta coding* approach is depicted: at the sender side the current packet header is compared with the previously transmitted header and the following data is sent in a compressed header format:

- A byte (byte 0) containing a **bit-mask** indicating the *changing fields* that have modified their value in the current header.
- A **connection number**, so that the receiver can infer the constant fields for the current header.

[1] See [84] and [97] for further details.

[2] This is not an end-to-end checksum. End-to-end checksums as the *TCP checksum field* **must be transmitted** without any compression.

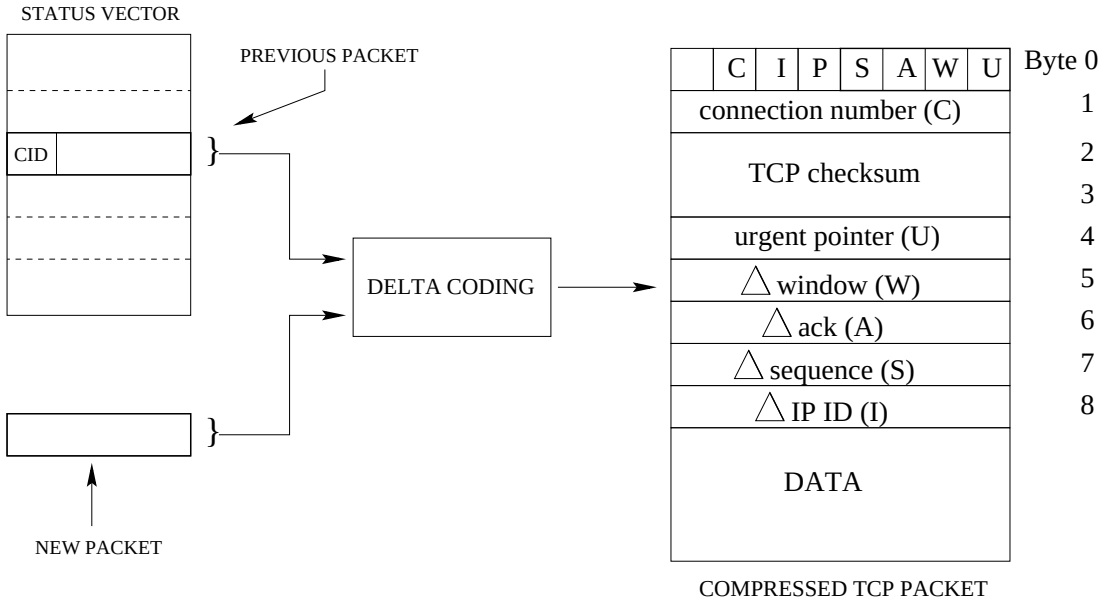


Figure 2.3: Delta coding approach.

- The *unmodified* **TCP checksum field**.
- The **differences** between the current and the previous value for every changing field. These differences will be referred in the sequel as *delta fields*.

At the receiver side, the uncompressed version of the last received header (referred here as *header base*) is stored and used together with the received delta fields to reconstruct the current TCP/IP header. In more detail, the received *header delta fields* are summed to the correspondent *header base* fields to obtain the values for the *changing fields* in the current header. Subsequently, these fields are used together with the constant fields to derive the new uncompressed header. The connection identifier is used here to identify the header base to be considered to decompress the current header.

It is worth noting that, since VJHC uses a differential encoding scheme, the loss of a packet always leads to a loss of synchronization i.e., the event in which the last transmitted *header delta fields* are not correctly received by the decompressor and the decompression fails [34] [89] [26]. In fact, once the delta fields have been received, they are used to construct the uncompressed version of the new header which is also stored in a status vector to decompress the next incoming packet. Suppose now that a packet (say packet $n-1$) is lost, while the following packet n is received correctly. In this case, the decompressor will try to obtain the uncompressed header for packet n using the last stored uncompressed header for the connection, i.e., the header relative to packet $n-2$. However, the decompressor will fail since the correct header base to be summed to the delta fields contained in packet n is the one of packet $n-1$. This failure occurs every time either a single packet or a sequence of subsequent packets is lost. To recover from this situation, the compressor sends uncompressed packets at each TCP retransmission. These uncompressed packets are used by the decompressor to resynchronize the *header base* and restart the decompression process. VJHC works fine over wired links, where the packet error rate is usually low, while in a wireless environment the high packet loss probability leads to frequent out-of-synch events followed by a decrease

of the performance. In the sequel, the Header Compressor and Header Decompressor units will be referred as HC and HD, respectively.

A second Header Compression scheme, proposed by Degemark et al. (DEHC) [34], improves VJHC by enhancing the synchronization robustness. In more detail, when the synchronism is lost, HD assumes that this is due to the loss of a single packet (say packet $n-1$) and that its *header delta fields* are equal to those of the current packet (i.e., n). The correct *header base* is then computed by adding twice the *header delta fields* of packet n to the last reference header base, while ignoring the contents of packet $n-1$. If the assumption is correct, the synchronism is preserved, otherwise a resynchronization request is sent to the HC by exploiting the TCP ACK flow from HD to HC. Note that the "twice" mechanism avoids loss of synchronization *only if exactly one packet is lost or corrupted* and only if its *delta fields* are equal to those of the following received packet. For these reasons DEHC has poor performance in a bursty error environment or when *delta fields* are not stationary.

A third and last algorithm which aims at reducing the synchronization loss probability is PEHC [89]. Here, the differences are computed with respect to a slowly changing base, i.e., the *header base* is updated less frequently, every $K > 1$ packets (in VJHC K is equal to one). Furthermore, in PEHC each update is performed by sending an uncompressed header. Thus, in PEHC a packet loss does not necessarily lead to a synchronization loss. K is set as large as possible, while respecting the constraint on the *header delta fields* sizes [59]. Obviously, since the delta fields are computed with respect to a slowly-varying header, this last method has the drawback that the header delta fields can grow larger than in VJHC. Hence PEHC is more robust to synchronism loss with respect to VJHC but it is less efficient in terms of compression ratio.

2.2.2 A New Header Compression Algorithm

In the following, the new header compression algorithm is presented in detail. This scheme avoids the synchronization loss problem while keeping the compression ratio to a reasonably good value. The algorithm works as follows:

- As in previously proposed header compression schemes a *header base* is maintained at both sides of the link, i.e., at the compressor (sender side) and at the decompressor (receiver side). These two copies of the base **MUST** be equal for the decompression algorithm to succeed; for this reason in the sequel we will talk of "common base". At the compressor/decompressor, the base corresponds to the uncompressed version of the last transmitted/received header. At the compressor the base is used to compute the differences (delta fields) with respect to the current header to be transmitted, whereas at the decompressor these differences are summed back to the base to obtain the full version of the changing fields. The compression method relies on sending differences, as in VJHC. The differences with respect to the VJHC algorithm are in the way in which the base is updated.
- As the compressor (HC) decides to update the common base, it sends to the decompressor (HD) a compressed header[3] having set a suitable flag, called *Request Flag* (RF). This flag is used to

[3] Which is compressed using the old base.

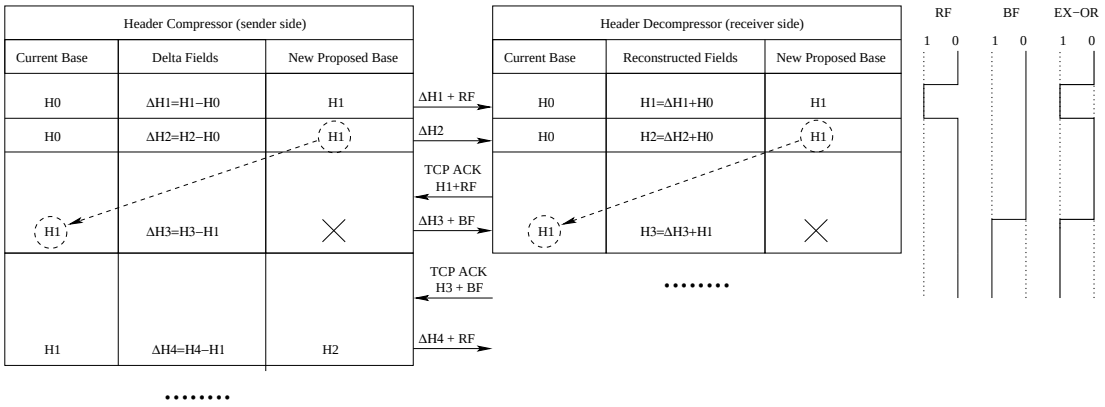


Figure 2.4: New Header Compression algorithm.

indicate that HC is willing to change the base and that the new base has to be derived using the uncompressed version of the current packet. HC also keeps track of the TCP sequence number relative to this *request packet* and stores the new proposed base in its internal memory as the *new proposed base*.

- If the packet containing the RF flag is correctly received at HD, the decompressor performs the following actions: 1) The packet is uncompressed and its uncompressed version is stored in memory as *new proposed base* 2) The decompression algorithm temporarily continues to use the old base.

- In the case the request packet is correctly received at HD and the decompression of its header is successful, a TCP ACK, containing the sequence number for that packet, is generated and sent back to the sender. At the sender side, HC, which is monitoring the backward TCP ACK flow, can then infer whether the *request* message has been correctly received or not.

- If HC receives the TCP ACK indicating the correct reception of the previously transmitted *request* message it starts using the *new proposed base* and notifies this to HD by changing the state of a Bistable Flag (BF) which is included in every compressed header. Only at this point can the new proposed base be used as the *new current base* at the compressor.

- As HD detects a state change for the BF flag it starts using the previously stored proposed based. Only at this point can the new proposed base be used as the *new current base* at the decompressor.

The procedure presented above is the core of the proposed compression algorithm. Nevertheless, some optimizations are still possible for the encoding of RF and BF flags. In Fig. 2.4, a typical behavior for the RF and BF flags is reported. It is easy to observe that, since their transitions always occur at different times, the two flags can be XOR-ed in a single bit without loss of information. The resulting flag can be sent from HC to HD by means of the unused flag in the bit-mask sent along with VJHC compressed headers (byte 0 in Fig. 2.3). Using this technique, RF and BF messages are both encoded by means of changes in the resulting flag, i.e., HD only needs to detect changes in the flag trace rather then absolute values.

In the new scheme, the base update is requested by HC every time a new base has been successfully established between HC and HD[4]. Hence, the base update frequency depends on factors such as packet errors and round trip delay experienced by IP packets. For this reason, the K introduced in the previous section as the number of subsequent headers between two header base updates is here variable. It is worth noting, however, that depending on the channel condition and on the IP round trip time, the base update is always proposed with the minimum possible delay once the new base has been recognized as successfully established. This is done to minimize the *header delta fields* resource occupancy. Moreover, let us note that in our proposal, as depicted in Fig. 2.4, each *header base* is used for at least the number of packets sent in one IP round trip time, i.e., for the number of packets sent between the transmission of a base change request and the reception of the corresponding TCP ACK. Furthermore, the *update confirm mechanism*, which is based on TCP ACKs, makes our update method more efficient if the TCP receiver operates in the non delayed ACK mode, i.e., one ACK is sent for each packet received. In fact, if either delayed ACKs are adopted or some ACK is lost, the IP round trip time is increased with a consequent decrease of both the base update frequency and the compression ratio.

To sum up, the new algorithm is able to completely avoid the loss-of-synch problem by proposing a base first and starting to use it only when a message arrives, ensuring that the new base has been successfully established at both sides. The procedure implies some additional steps, namely, the *base proposal* and the *base confirmation* that have to be accomplished to use a new base. These procedures introduce some additional delay on the base change updates (with respect to VJHC) but ensure robustness for the loss-of-sych problem.

As a last observation, note that if a transmission error occurs, i.e., HC does not receive a TCP ACK for a given packet, then that packet will be retransmitted without compression as in [59]. HC in this case restarts the compression mechanism with the first new packet after the last retransmission, without waiting for the corresponding TCP ACK.

2.3 General Simulator Description

In order to test and compare the classic and proposed Header Compression schemes on a WCDMA air interface, a UMTS simulator which generates WCDMA *channel traces*[5] and a TCP/IP HC algorithm simulator, taking these traces as input, have been developed.

The UMTS simulator is based on two steps. The first step simulates multi-user communications in a multi-cell system, accounting for user placements, signal propagation, power control, intra and inter-cell interference, Doppler frequency (which in our simulation accounts for user mobility) and other factors. In this first step Signal to Interference Ratio (SIR) traces are computed for each user. The second step is based on a post-processing of the SIR traces accounting for channel coding and interleaving. From this we obtain sequences of block error probabilities, one for each user, called *channel traces*.

[4]This is detected by monitoring the TCP ACK flow at HC. Referring to Fig. 2.4, the second base request occurs only when the TCP ACK for the new adopted base has been received at HC.

[5]The same procedure considered in Chapter 1 has been used here to obtain channel traces. This procedure in the next is summarized for the sake of clarity.

These *channel traces* are used to randomly generate sequences of block error, BESs, one for each user, that are then fed to the TCP/IP simulator. This simulator implements the VJHC, DEHC and PEHC schemes as well as the new HC scheme. From that simulator some metrics are obtained. The throughput is defined as the fraction of the channel transmission rate (considered at the IP level) which provides correct information bits at the receiver (not counting erroneous transmissions, spurious transmissions, idle time and overhead). The final throughput, S, of each user is averaged over the entire simulation. The HC simulator also returns the average *header base* update rate, f_H, defined as the inverse of the average number of packets between two successive *header base* changes. This performance figure is an index of the header compression gain; in fact, high f_H implies lower differences on the *header delta fields* and so higher degree of compression. Another output of the simulator is the average compressed header size, H_s, expressed in bytes, not considering the effect of window, ACK and urgent pointer fields.

2.3.1 UMTS Simulator

The UMTS simulator considered here to obtain WCDMA error traces is the same that has been used to derive the results in Chapter 1. The reader is referred to Section 1.2 of that chapter for a detailed discussion on the channel simulation environment and related parameters.

2.3.2 Header Compression Simulator

In this simulator, TCP/IP New Reno [44] version 4 is considered and all HC algorithms described above have been implemented. The data transmission is unidirectional, i.e., packets flow from HC (sender side) to HD (receiver side), whereas in the reverse direction only ACKs flow. The TCP receiver generates non delayed ACKs. The direct link packet generation is assumed continuous, as in a long FTP session, the bit rate at the IP level output is 103.5 Kbit/s, the full TCP/IP header is 40 bytes, whereas the MTU size is considered to be 512 bytes. In VJHC, the compressed header size is fixed at 4 bytes. In fact, as observed in [59], in VJHC the compressed header size spans typically from 3 to 5 bytes. Given its very limited oscillation range, it seems to be reasonable to consider the average value of 4 bytes for the results presented here. Since DEHC uses the same header compression algorithm as VJHC, we consider the same header size. On the other hand, in all other HC schemes the header size has been derived by keeping track of the header fields variations.

For the sake of comparison, in addition to the HC schemes previously described, we have defined and simulated a special Header Compression algorithm, named as IDEAL. In this scheme, VJHC is considered, under the assumption that packet losses due to channel errors affect TCP operation but never lead to loss of synchronization. Note that VJHC is the scheme which provides the best compression performance. Hence, our assumption for the ideal case is expected to provide an optimistic bound for both throughput and compression ratio performance, as discussed in [34]. RLC and MAC levels are configured to operate in transparent mode [1]. The bit rate used in the TCP/IP HC algorithm simulator, i.e., the bit rate at the output of the IP level, has been fixed to 103.5 Kbit/s, which has been derived considering that the raw channel transmission bit rate is 240 Kbit/s, the rate reduction due to the convolutional coding is $1/2$, and accounting for some overhead due to a possible control channel and to RLC/MAC headers.

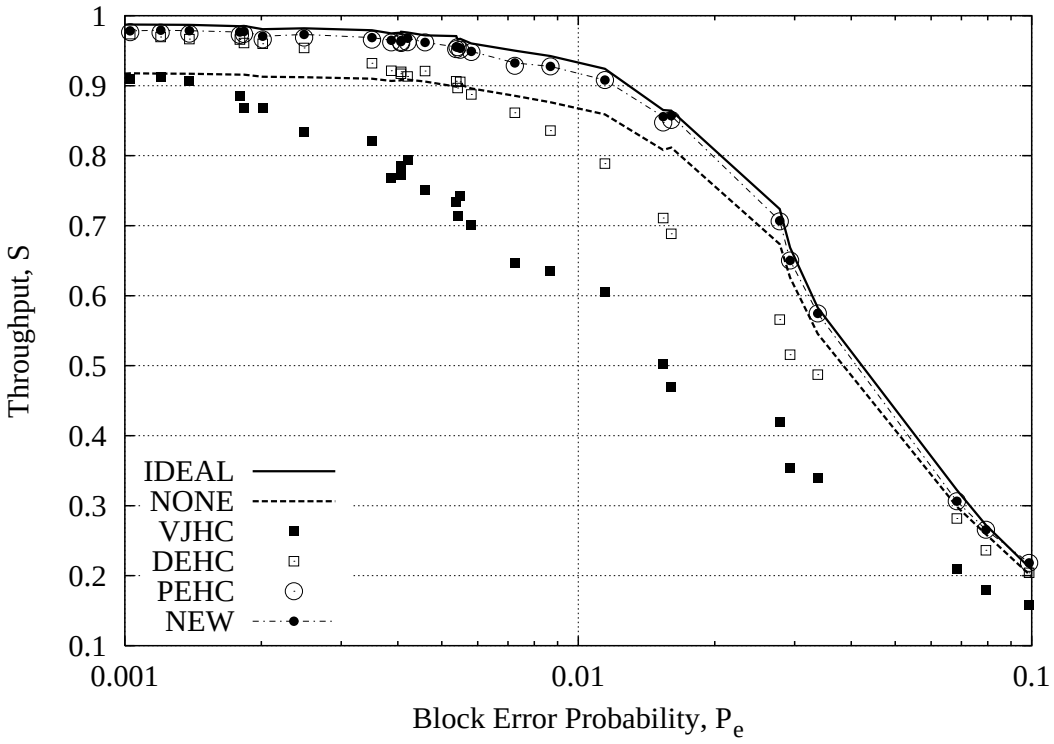


Figure 2.5: Throughput performance for a FTP long file transfer as a function of the Block Error Probability (P_e). 90 users, $TTI = 1$, $MTU = 512$ bytes.

2.4 Numerical Results

In this section some numerical results are reported to highlight the pros and cons of the Header Compression algorithms discussed above. Performance is investigated considering the throughput, the header base update frequency and the compressed header size metrics.

In Fig. 2.5 the TCP/IP throughput is plotted as a function of the Block Error Probability[6] (P_e) considering a system load of 90 users, markers are used to plot the performance of the users in the system. In the figure, the case where no Header Compression is applied to the TCP flow is labelled as **NONE**, whereas the new proposed scheme is referred as **NEW**. As explained above in Section 2.3.2, the name **IDEAL** is assigned to the VJHC scheme without synchronization losses. This is an ideal scheme which is considered as an upper bound for the performance. As a first observation, it is worth noting that the VJHC scheme presents very poor performance when P_e is not negligible and that, in such cases, it is better to consider a plain TCP/IP transmission (labelled as NONE), i.e., without header compression. In fact, in VJHC every out-of-synch event [7] is detected and repaired by timeout and subsequent retransmission of uncompressed packets. Thus, this procedure is very inefficient over error prone channels. Note also that the only reason for the poor performance of the VJHC scheme is the out-of-synch problem (see differences between IDEAL and VJHC). DEHC, despite its very simple algorithm, presents a much better throughput. However, its performance falls below the plain TCP case as well, as P_e increases. Finally,

[6] A block corresponds to a UMTS radio frame.

[7] Rememeber that there is a correspondence one-to-one between packets in error and out-of-synch events.

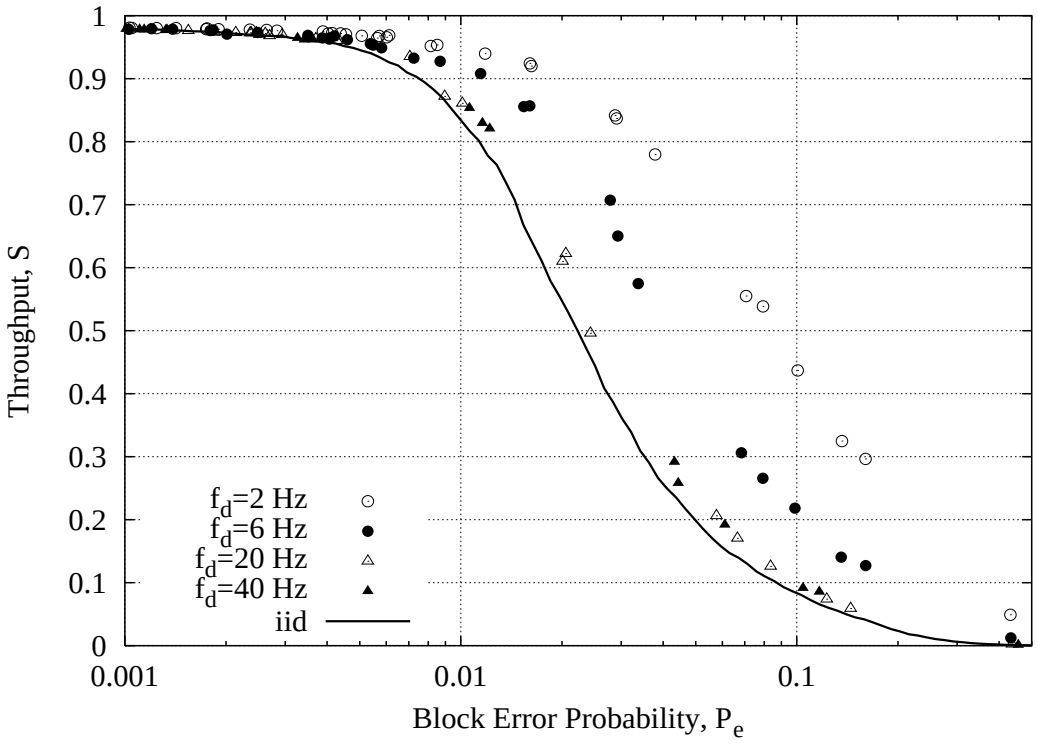


Figure 2.6: Throughput performance for a FTP long file transfer as a function of the Block Error Probability (P_e) by varying the Doppler frequency (f_d). 90 users, $TTI = 1$, $MTU = 512$ bytes.

we note that DEHC and the new scheme perform similarly and that their performance is very close to the IDEAL algorithm. Moreover, their throughput is always larger than the one in the plain TCP case and this makes these schemes suitable to be used over a wireless link.

In Fig. 2.6 the throughput sensitivity to the Doppler frequency of the new proposed scheme is reported. The Doppler frequency is varied in the set $f_d \in \{2, 6, 20, 40\}$ to investigate the effect of the user mobility on throughput performance. The case with independent and identically distributed (*iid*) packet errors is also reported for comparison. As expected, the throughput decreases as f_d increases. Furthermore, as $f_d > 20$ Hz the curves do not significantly differ. This indicates that, for such values of the Doppler frequency, the IP packet error statistics at the application (IP) layer is approaching the independent assumption (which is the worst case for TCP/IP performance, regardless of the considered header compression scheme).

In Fig. 2.7, the average *header base* update rate is reported as a function of P_e. Here, the advantage of using the new proposed algorithm can be clearly understood. In fact, the header update frequency for the new algorithm is almost two orders of magnitude lower than the one in PEHC. The header update period, i.e., the number of headers transmitted between two subsequent base updates is directly related to the size of the compressed header. In fact, since the compression is achieved by means of differences with respect to a header base, the longer the base will remain constant, the larger will be the size of the delta fields sent along with compressed headers. Therefore, a higher base update frequency also implies

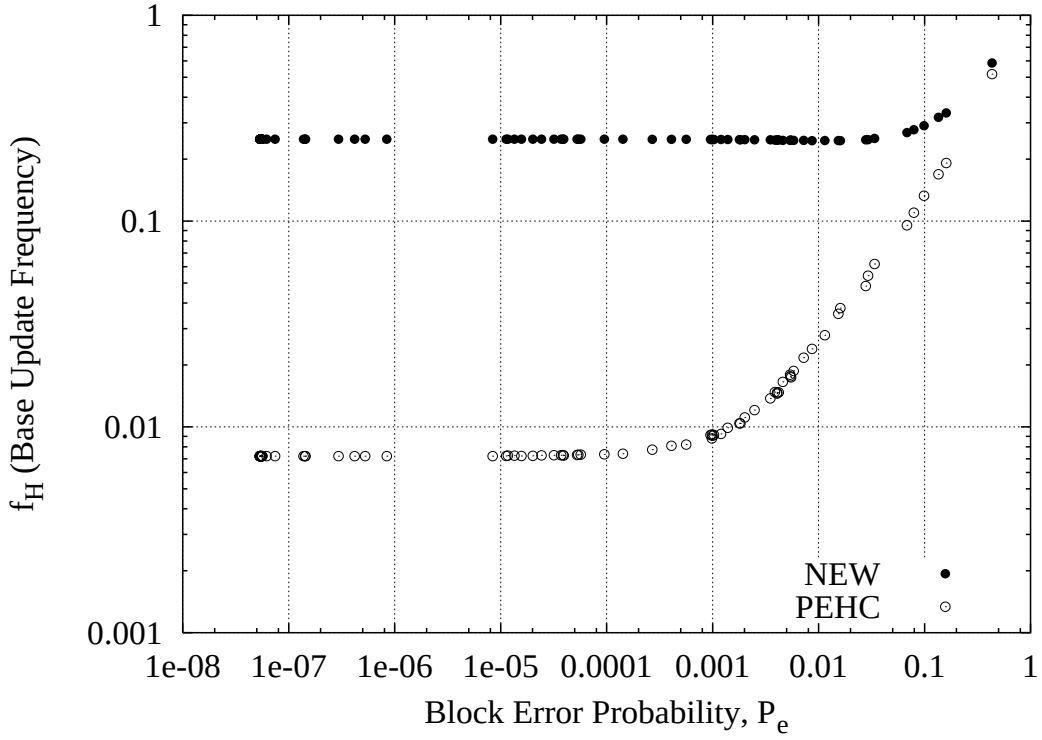


Figure 2.7: Header base update rate for $f_d = 6$ Hz, 90 users, $TTI = 1$, $MTU = 512$ bytes.

a higher degree of compression. From the figure it is also evident that f_H increases with P_e. This fact is promptly explained as follows: by increasing the error probability, the rate of TCP retransmissions grows due to the TCP timeout and Fast Retransmit algorithms. In addition, for both the new and the PEHC schemes, an uncompressed header is sent along with each retransmission[8] and every retransmission also contributes to re-synchronize compressor and decompressor by updating the header base at both sides. The new compression scheme, thanks to the new base request/confirm mechanisms, is able to adapt its base update period to the channel conditions, i.e., it keeps a high base update frequency over error free channels, whereas the frequency decreases as the error rate affecting the IP flow increases. For highly error prone channels, the base update in the new compression algorithm is approaching that of PEHC[9], whereas as the error rate decreases, the base update frequency is kept at a very high value. This self-adaptation is not possible with the PEHC scheme that, instead, can only work with a fixed base update period, usually fixed at the maximum admitted value (that in PEHC is the only way of diminishing the loss-of-synch problem). For this reason, our new scheme is suitable for use over wireless links, where the bit error rate is variable, due to user mobility, channel conditions (i.e., obstacles and fading). In such a scenario, it is more convenient to use a scheme that is self-adapting and able to achieve optimal performance.

Finally, in Fig. 2.8 the average and the standard deviation of the compressed header size, H_s are reported as a function of P_e. The new compression scheme is characterized by a higher compression ratio

[8]This behavior is crucial to recover from decompressor failures of any kind, see [59] for further reading on this feature.

[9]Here, TCP reacts to the high IP error rate by retransmitting. The base update period is then decreased since every retransmission also corresponds to a base update. This feature is the same in both compression schemes and it is necessary to recover from fault conditions at the compressor/decompressor.

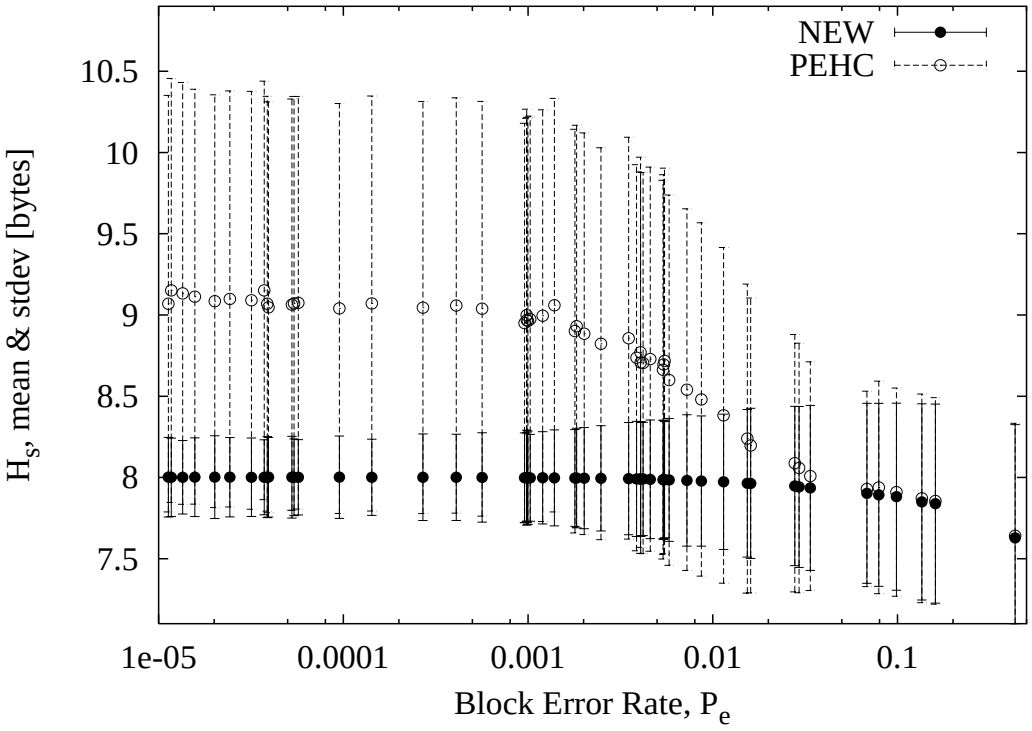


Figure 2.8: Header compression efficiency. $f_d = 6$ Hz, 90 users, $TTI = 1$, $MTU = 512$ bytes.

together with lower standard deviations on the compressed header sizes. This means that, especially for interactive applications, the new scheme can guarantee a more stable flow also characterized by a shorter response time. Moreover, it is worth reminding that, in our study, the impact of some header fields has been neglected. Even higher gains are expected in a real scenario, where all the protocol details can be taken into account. In this case, the impact of header fields such as *urgent pointer*, *window* and *ACK* can be evaluated. Our expectation is that, thanks to a more frequent base update[10] and also depending on the kind of traffic, the additional header occupancy will grow and several additional bytes could be saved using the new approach.

2.5 Conclusions

In this chapter, after a brief review of existing TCP/IP Header Compression algorithms we have described a novel TCP/IP Header Compression scheme for wireless networks which is able to avoid the synchronization loss problem. The performance of these schemes has been investigated by considering a WCDMA air interface. The proposed solution presents some advantages over existing algorithms: first of all, it is self tunable, i.e., due to the new header base request/confirm mechanism, the compression base update frequency is self adapted to the error/delay characteristics of the channel so as to achieve optimal performance. Moreover, higher compression ratios are reached together with a lower base update

[10] In fact, a single change in one of these fields is then affecting every single header up to the next header base change. In the new algorithm this change will be absorbed in a few packets, whereas in the PEHC scheme even more than 100 packets would have to be waited for before a base refresh occurs.

frequency. Both these advantages contribute to making the new scheme suitable for interactive applications over bandwidth limited wireless channels. For what concerns file transfer applications, the new scheme only slightly outperforms the existing ones. Test implementations of both solutions are important to evaluate additional expected gains when all the protocol header details are taken into account.

Chapter 3

PETRA: Performance Enhancing TRansport Architecture for Satellite Communications

In this Chapter a performance enhancing transport architecture for the satellite environment is presented. This solution is aimed at improving the network transport performance by overcoming the limits imposed by a TCP/IP based protocol stack, while maintaining the interfaces offered by it. This is an important issue since TCP/IP is widely used and most of the applications are based on it. The Chapter starts from the state of the art about the transport layer over satellite by distinguing two alternative frameworks: the Black Box (BB) and the Complete Knowledge (CK) approach. In the former, the network is considered as a "black box" and modifications in the terminal tools are only permitted. In the latter, the complete control of any network element is allowed so that a performance optimization procedure is possible. The proposed architecture (called **Performance Enhancing TRansport Architecture - PETRA**) is designed in all details using the second approach. PETRA uses network elements, called Relay Entities, to isolate the satellite portions in case of heterogeneous networks, while a transport layer protocol stack is used to optimize the transport of information over satellite links. A special Satellite Transport Protocol, that is part of the transport layer protocol stack, is used over such links to perform error recovery. Simulation results show that the proposed framework significantly enhances throughput performance.

3.1 Introduction

The Transmission Control Protocol (TCP) is the connection-oriented, end-to-end reliable transport protocol utilized in the Internet. The motivations, the philosophy and the functional specification of the protocol are contained in [84]. TCP assumes that the service offered by lower layer protocols is unreliable; various features such as timeout timers, packet reordering and retransmissions are used in TCP with the aim of providing a reliable channel to higher layer applications. Unfortunately, these features have been designed to be effective over wired networks, but it is well known that they often fail when the underlying channel is characterized by both a large *bandwidth delay* product and *high error rates*. This is the case of heterogeneous networks containing satellite links, where special countermeasures have to be taken to

correct the inefficiencies of the TCP protocol.

Satellite networks are attractive since they offer clear advantages with respect to cable networks [66]: their architecture is more scalable, the diffusion throughout the land is wide and the multicast service is very simple. Given the inefficiencies of standard TCP over such networks, one could design a new protocol, specifically tailored for these environments. However, given the widespread diffusion of TCP/IP applications, it is very difficult to think of a protocol different from TCP, that is still transparent to the user, to be used over satellite links. For these reasons, it is more appealing to provide new solutions, where the TCP/IP can still be used at the end terminals and its inefficiencies are accounted by algorithms and protocols running at the end-terminals and in the satellite portion of the network. The aim of the following work is to propose and validate a transport architecture to deal with this problem.

In the sequel, a transport layer architecture that allows transporting TCP/IP flows efficiently and transparently through satellite networks to the final user is presented. The proposed architecture, which we call *Performance Enhancing Transport Architecture* (PETRA), divides the end-to-end connection into different segments. The bridging between different network segments is performed by elements named *Relay Entities*. The objective of PETRA is the optimization of both the throughput performance for the satellite network environment and the efficient utilization of network resources. This is achieved without re-designing the protocol interfaces, so that they maintain the same characteristics of the interfaces currently used. As a consequence, system performance is optimized and at the same time, utilization of standard applications is still possible, thus reaching a high degree of portability. Moreover, the proposed solution preserves the end-to-end reliability and semantic of the transport layer by introducing, at the transport layer, a new sublayer named *Upper Transport Layer*. The transport layer is divided into two sublayers:

- *Lower Transport Layer* (LTL): A different LTL instance is utilized in each segment of the end-to-end path. The LTL is responsible for the transport in each of these segments. The LTL utilized in the satellite segment is called *Satellite Transport Protocol Plus* (STPP). It is similar to the STP proposed in [51] with the following differences:
  - STPP introduces a mechanism that avoids deadlock situation.
  - STPP introduces a mechanism which stops the flow over the satellite link to avoid buffer overflows in the Relay Entities.
- *Upper Transport Layer* (UTL): Its algorithms are executed end-to-end and are responsible for maintaining the end-to-end reliability and semantics of the transport layer.

PETRA is targeted towards a GEO (Geostationary Orbit) satellite environments, where the long round-trip delay heavily affects the system performance, but radio-mobile and LEO (Low Earth Orbit) satellite systems, characterized by channels with fading and high bit-error rate, are not excluded. PETRA has a high degree of flexibility that allows an efficient adaptation to these peculiarities. The idea of splitting the connection into different segments has been already proposed in the past [22]. The major contributions of the paper are the following:

- A new transport layer protocol architecture is proposed where two sublayers (LTL and UTL) coexist as explained above.

- Algorithms and their interactions are considered in all details. Moreover, the optimal values of several algorithm parameters are calculated analytically or empirically.
- New flow control algorithms are proposed for enabling effective integration of satellite systems in the Internet.

The rest of the Chapter is organized as follows. Section 3.2 presents the state-of-the-art regarding transport layer issues in satellite networks. Section 3.3 contains the description of new protocol architecture (PETRA) and a list of requirements oriented to the implementation of the architecture. Section 3.4 concerns the parameters setting within the PETRA architecture. Performance evaluation is provided in Section 3.6, whereas our conclusions are drawn in Section 3.7.

3.2 State of the Art

The problem of improving the performance of TCP over satellite has been investigated in the literature for some years. Many national and international programs and projects (extensively listed in [77]) in Europe, Japan and USA have been devoted to satellite networks and applications. In particular, some of them, or part of them, are aimed at improving performance at the transport level. NASA ACTS (in [23] and [57]), ESA ARTES-3 (in [37]) and CNIT-ASI (in [6]) deserve a particular attention, among many others.

Also International Standardization Groups as the Consultative Committee for Space Data Systems - CCSDS, which has already emitted a recommendation (see [3]) and the European Telecommunications Standards Institute - ETSI [36], which is running its activity within the framework of the SES BSM (Satellite Earth Station - Broadband Satellite Multimedia) working group, are active on the subject.

Accordingly, a large literature exists on this topic, see [86] for a first overview and [70] for a more specific study in TCP/IP networks with high delay per bandwidth product and random loss. More recently, [45] provides summaries of improved TCP versions as well as issues and challenges in satellite TCP; [51] highlights the ways in which latency and asymmetry impair TCP performance; [12] lists the main limitations of the TCP over satellite and proposes many possible methods to act. References [18] and [9] represent, at the best of the authors' knowledge, the more recent tutorial papers on the topic: various possible improvements both at the transport level and at the application and network level are summarized and referenced.

A recent issue of International Journal of Satellite Communications is entirely dedicated to IP over satellite [35]. In more details, reference [21] proposes a TCP splitting architecture for hybrid environments (see also [105]); reference [67] analyses the performance of web retrievals over satellite and references [77] [75] and [76] show an extensive analysis of the TCP behavior by varying parameters as the buffer size and the initial congestion window. Reference [49] focuses also on the buffer management but in an ATM environment. The proposed approaches can be divided into two classes[1] [78]:

- **Black Box (BB) Approaches**: only the end terminals are modified. The rest of the network is considered non-accessible (i.e., a black box).

[1] The classification proposed is not the only possible one and, probably, it is not exhaustive (i.e., not all the algorithms and methods in the literature can be classified within one of the two classes).

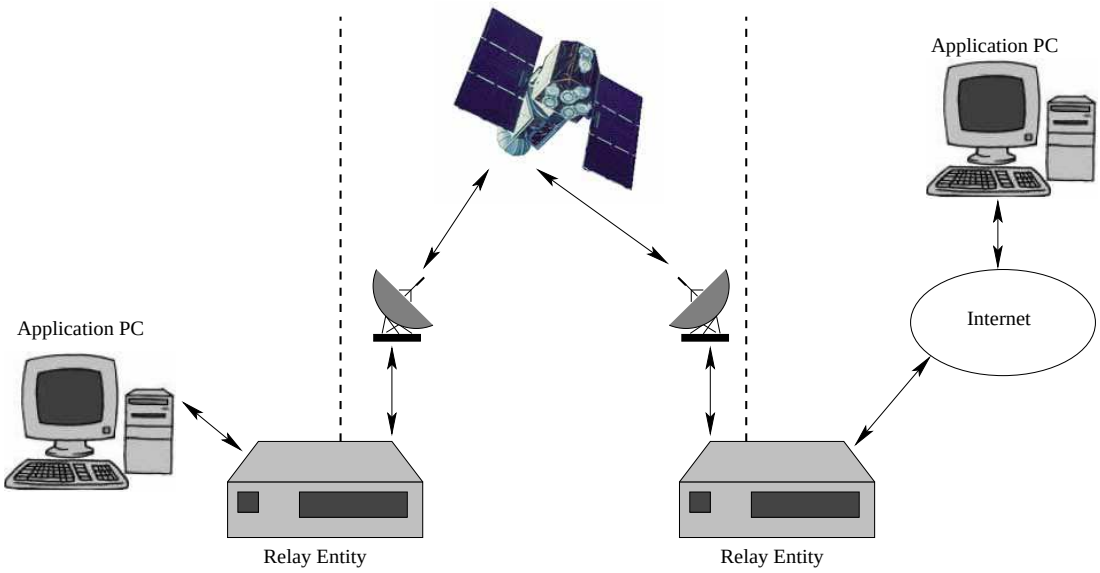


Figure 3.1: PETRA architecture.

- **Complete Knowledge (CK) Approaches**: tuning of the parameters and algorithms of all the network components is allowed.

Many solutions based on the BB approach have been proposed so far, e.g., [10, 11, 13, 27, 60]. These solutions are general and do not violate the end-to-end semantic of TCP. Also considerable work has been carried out according to the CK approach. Such methodologies, e.g., TCP splitting [86] [45] [21] [105] and TCP spoofing [86] [105], bypass the concept of end-to-end service by either dividing the TCP connection into segments or introducing intermediate gateways, with the aim of isolating the satellite link. The recent RFC 3135 [22] is dedicated to extend this concept by introducing Performance Enhancing Proxies (PEPs) intended to mitigate link-related degradations. RFC 3135 is a survey of PEP techniques, not specifically dedicated to the transport layer, even if the emphasis is put on it. Motivations for their development are described as well as consequences and drawbacks. The solutions based on the CK approach provide higher throughput performance with respect to BB but violate the end-to-end characteristics of the transport layer [86] [9].

3.3 Performance Enhancing Transport Architecture (PETRA)

3.3.1 Operative Environment and Proposed Approach

The operative environment is shown in Fig. 3.1. The two Application PCs are the end systems and are connected through *terrestrial segments* to the *Relay Entities* (REs). Observe that the terrestrial segments can be the access LAN (Local Area Network) as well as the Internet. The REs are connected with each other by means of the satellite communication system. In Fig. 3.1 only one satellite segment is considered, however the extension to the case with multiple satellite segments is straightforward.

Also note that in Fig. 3.1 the satellite network is within the backbone. If the satellite network is, instead, the access network, then the RE is a software module directly connected to the Application PC

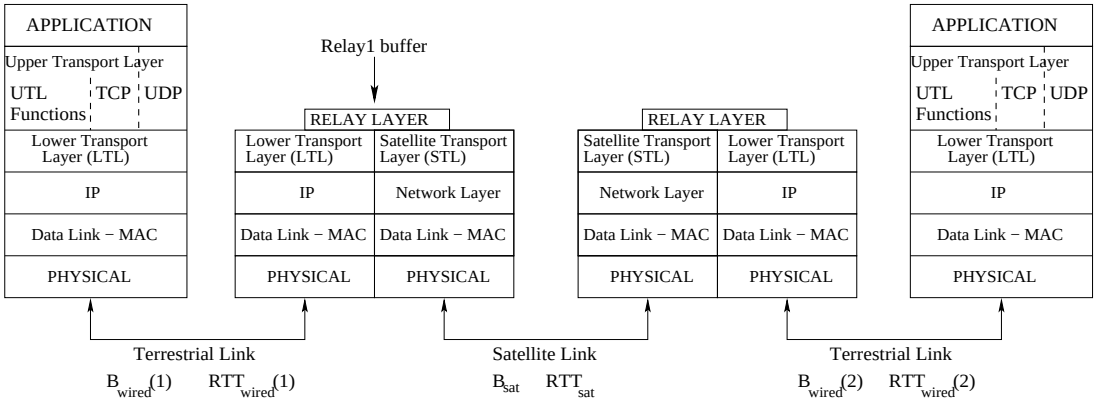


Figure 3.2: End-to-end PETRA architecture.

or a hardware card inside the Application PC. PETRA algorithms are deployed in the Application PCs and Relay Entities. A different transport protocol is run in each network segment. This allows to adapt the protocol to the specific characteristics of the segment. Moreover, an end-to-end algorithm is used between the two Application PCs in order to maintain the end-to-end semantics and reliability of the transport protocol.

3.3.2 Transport Layer Protocol Stack

The proposed protocol architecture is called *Protocol Enhancing Transport Architecture* (PETRA) and is shown in Fig. 3.2. The transport layer is divided into two sub-layers:

- *Upper Transport Layer (UTL)* which guarantees the end-to-end semantics.
- *Lower Transport Layer (LTL)* which is responsible for the transport within a single communication segment. The LTL in the satellite segment is denoted as *Satellite Transport Layer* (STL).

A *Relay Module* is defined in the Relay Entities which bridges the different LTLs. Note that the Relay Module is a part of the transport layer.

3.3.3 Upper Transport Layer (UTL)

The Upper Transport Layer (UTL) is responsible of preserving the end-to-end semantic of the transport layer. UTL segments data coming from the user application into ULT data units (called *batches*) and transmits them according to a window-based flow control characterized by a window size W_{UTL}. The value of W_{UTL} must be large enough to ensure a continuous transmission flow over the satellite link for an entire end-to-end round trip time[2]. Optimal choice of the W_{UTL} is given in Section 3.4.1. UTL is also responsible for failure recovery. To this purpose a timeout $T_0(UTL)$ is defined. If an UTL batch is not acknowledged within a time period equal to $T_0(UTL)$, then the connection is terminated. The timeout value depends on the end-to-end round trip time, whose accurate estimate is, therefore, critical.

[2]It is the time elapsed between the departure of the first bit of a UTL batch and the instant in which the relative (end-to-end) UTL ACK is received.

3.3.4 Lower Transport Layer (LTL)

The Lower Transport Layer (LTL) is responsible of the data integrity over the wired portions of the path. It segments the batches coming from the UTL layer into segments of (usually) smaller size (equal to the LTL MSS). In the terrestrial segments, LTL runs a TCP-like algorithm, with the only exception that the LTL ACK policy is slightly different from the one adopted in a standard TCP receiver. In particular, the LTL ACK flow can be stopped to realize the flow control at Relay Entity buffers. The ACK stopping technique is explained in more detail in the next section. Instead, the protocol executed in the STL is called Satellite Transport Protocol Plus (STPP) and will be presented in Section 3.3.6.

Moreover, UTL batches are considered to be composed of n LTL segments, where n is an integer number such that $n \geq 1$. The choice of n is a design parameter that must be optimized jointly considering user and network requirements. In fact, a large n leads to a long end-to-end round tip time (that translates in a long UTL timeout value) and, as a consequence, the connection will be characterized by a long system failures reaction time. On the other hand, when n is small, we have a short network failure reaction time, but the number of UTL ACKs flowing on the backward path increases (leading to a waste of system resources in the reverse path). The choice of n is discussed in more detail in Section 3.4.1.

3.3.5 Relay Module

At the Relay Module some buffer space is reserved for each connection. Since the terrestrial and satellite segments are characterized by very different bandwidth, this buffer is utilized to harmonize the entering and the outgoing flows. Substantially, to control the Relay buffer occupancy a *stop-and-wait* strategy is used. For instance, focusing on a Relay Entity placed between the terrestrial segment and the satellite channel (i.e., *Relay1 buffer* in Fig. 3.2), the Relay Buffer input flow is due to LTL segments, whereas the output flow is composed by STPP packets.

To avoid buffer overflow problems, as the Relay buffer size reaches a given threshold, B_{th}, the LTL ACK transmission is inhibited and, as a consequence, the forward LTL flow is also stopped. Moreover, as the buffer occupancy becomes lower than B_{th} the LTL ACK flow is resumed. The LTL forward packet flow will be also resumed after a half wired round trip time ($RTT_{wired}/2$), i.e., the time needed for the resumed ACK flow to reach the LTL sending entity.

3.3.6 Satellite Transport Protocol (STPP)

The STPP is the protocol responsible of the data integrity over the satellite channel. It is similar to the STP proposed in [51] with the following differences:

- STPP introduces a mechanism that avoids deadlock situations.
- STPP introduces a mechanism which stops the flow to avoid buffer overflows in the Relay Entities.

STPP must then counteract for channel errors due to wireless propagation phenomena allowing retransmission for lost packets. The STPP is a sliding window based protocol. At the sender side, incoming packets from higher layers (STPP SDUs) are segmented into STPP packets (STPP PDUs) and transmitted over the satellite channel. At the receiver side, these packets are ordered and the original higher level

packets are re-constructed. Retransmissions follow a NACK based Selective-Repeat ARQ mechanism as explained in the following:

1. At the STPP sender packets are taken from the STPP input queue (LTL SDUs) and segmented into STPP PDUs (the packet units handled at the STPP level). Then, a CRC (Cyclic Redundancy Check) field is attached to each STPP PDU before its insertion into the STPP transmission queue.

2. At the sender, a retransmission buffer is accounted for. Retransmissions are requested by the receiver by means of special status messages, as described below. All the packets indicated in these status messages are inserted in the retransmission queue. At a given time, if no retransmission requests are pending (the sender retransmission queue is empty) then a new PDU is taken from the transmission queue and is transmitted over the channel. Instead, if the sender retransmission queue is not empty then the first PDU is taken from that queue and transmitted over the channel. The retransmission of each packet is inhibited for a full satellite round trip time from the instant of its transmission, i.e., all retransmission requests for that packet during this time period are ignored.

3. At the STPP sender, associated with each transmitted PDU, there is a timeout timer, which is initially set to $T_o(STPP)$ seconds. If this timer expires for a given PDU, then such PDU is scheduled for retransmission by inserting it into the STPP retransmission queue.

4. At the STPP receiver, correctness of received packets is determined by using the CRC field. In addition, the STPP receiver identifies the gaps in the received PDUs sequence numbers to infer *lost* packets.

5. If erroneous or lost PDUs are detected, the STPP receiver periodically (every T_{stat} seconds) sends back to the transmitter one *Unsolicited STATUS message* (USTAT) containing the highest PDU sequence number correctly received in order (LAST_IO), the highest sequence number correctly received (LAST_SEQ_NO) and a sequence number mask in which all the sequence numbers of the *erroneous* and *lost* PDUs are listed. USTAT messages are used by the STPP receiver to request the retransmission for lost PDUs. In the case where no errors are detected, USTAT messages containing the LAST_IO identifier are sent back every T_{stat} seconds to advance the sender window.

6. At the STPP sender when an USTAT is received, all the requested PDUs not present in the retransmission queue are scheduled for retransmission by inserting them into the STPP sender retransmission queue. Moreover, all PDUs with sequence number lower than or equal to LAST_IO are deleted from the STPP memory.

7. When the UTL window is utilized for more than the 90% of its maximum size, the STPP sender generates a POLL message triggering an advance of the UTL window in order to avoid deadlock situations. The POLL message is sent to the receiver by setting a special POLL bit in the STPP PDU headers. When the POLL message arrives at the receiver, a Solicited STAT message (SSTAT) is immediately sent back to the sender. This message is identical to the USTAT, with the only difference that it is sent *on-demand* (upon the reception of a POLL) The POLL message is re-sent repeatedly for every following satellite z round trip time until reception of a SSTAT.

Note that the STPP receiver uses STATUS messages in order to save bandwidth in the feedback channel. In the following we show that T_{stat} is a crucial parameter because STPP performance is highly dependent on its choice. Moreover, in order to fully utilize the channel resources over the wireless satellite link it is pivotal to correctly set the STPP window size. Specifically, it must be chosen to ensure that the *satellite link is entirely filled by transmitted packets*. Referring to the STPP window as W_{STPP} (expressed in units of STPP PDUs), to the STPP PDU size as PDU_{len} (expressed in bit), to the satellite channel bandwidth as B_{sat} (expressed in bps) and to the satellite round trip time as RTT_{sat} (expressed in seconds), the constraint that the STPP window must meet in order to be able to entirely fill the satellite channel with transmitted packets can be written as

$$W_{STPP} \geq \left\lceil \frac{B_{sat} RTT_{sat}}{PDU_{len}} \right\rceil \tag{3.1}$$

The bandwidth available in the forward satellite channel is a known quantity. Therefore, the STPP protocol does not need to incrementally probe the channel for available bandwidth. Accordingly, the STPP level does not rely on algorithms such slow-start and congestion avoidance. Instead, it simply uses a sliding window approach by sending, at any time, the maximum amount of data allowed by the current value of W_{STPP}. In addition to the normal operative mode presented above, to avoid buffer overflow at the Relay Node buffer at the receiving STPP entity, the following *flow-control* policy is implemented:

1. When the number of packets in the Relay Node buffer becomes larger than a given buffer threshold (B_{th}), the STPP ACK flow is stopped[3] by sending a *stop transmission* (STOP_TX) message.

2. When the number of packets in the receiver Relay Buffer becomes lower than B_{th}, the transmission is resumed by sending back a *resume transmission* (RESUME_TX) message. This message in inserted in a USTAT packet.

3. Upon receiving the RESUME_TX message, the sender re-starts transmitting following the rules presented above.

3.4 Protocol Parameter Design

The PETRA architecture introduced in the previous sections utilizes several parameters. In the following it is shown how these parameters should be set based both on simulation and analytical results.

3.4.1 Dimensioning the UTL Timeout

Referring to a generic UTL packet, we define the UTL round trip time (RTT_{UTL}) as the time elapsed between the transmission time of the first bit of that packet and the instant at which its UTL ACK is received. The UTL timeout ($T_o(UTL)$) must be set such that $T_o(UTL) \geq RTT_{UTL}$. In this Section, we compute an upper bound for RTT_{UTL}, i.e., where the LTL level has already reached its maximum allowed window size, W_{LTL}^{max} (measured in units of LTL segments).

[3]Note that, due to the PDUs in flight over the channel, after this action, at most one further satellite round trip time of STPP PDUs is received. This must be considered in the dimensioning of B_{th}.

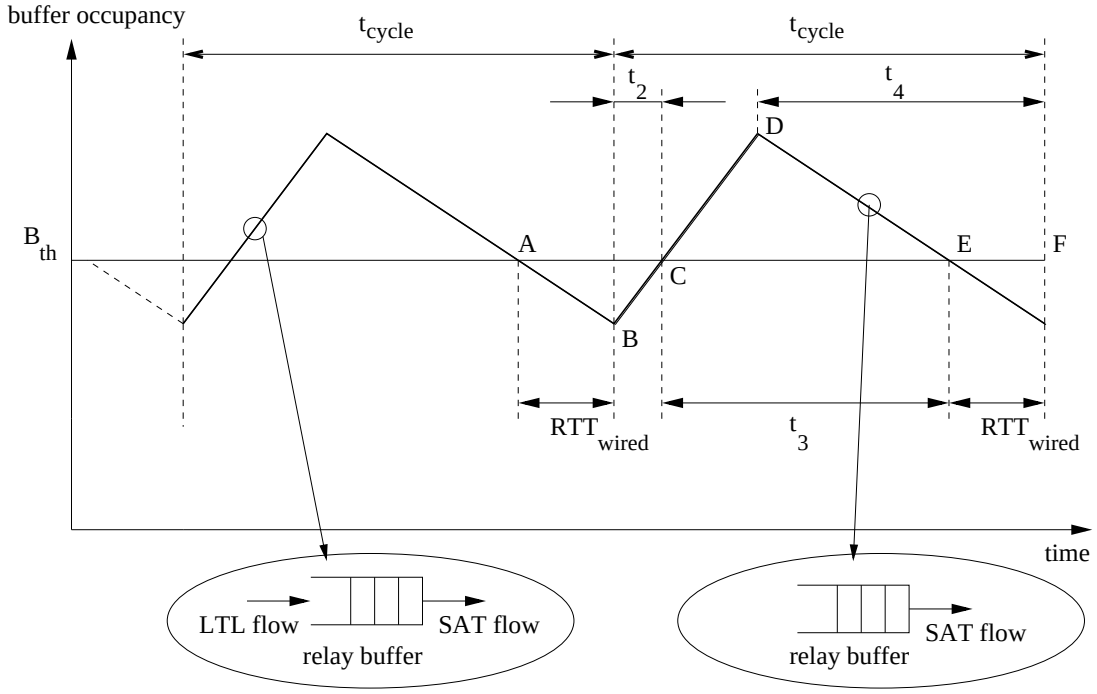


Figure 3.3: Relay buffer behavior.

According to the scenario depicted in Fig. 3.2 let us consider first the behavior of the Relay Buffer placed in the first Relay Entity encountered along the forward path (the leftmost Relay Entity in Fig. 3.2). We assume that the available bandwidth over terrestrial paths[4] is larger than the one characterizing the satellite link (B_{sat}), i.e., $B_{wired}(i) > B_{sat}, \forall i$. In this case, the Relay node buffer is a bottleneck for the incoming LTL flow and the ACK stopping feature at the receiving LTL Entity is used to harmonize the incoming and the outgoing flows.

In Fig. 3.3, we show an example behavior for the Relay buffer occupancy as a function of time. Let us start our description from point A of this figure, where the LTL ACK flow is restored because the number of segments in the buffer is below the threshold B_{th}. Due to the finite round trip time characterizing the first terrestrial link ($RTT_{wired}(1)$) no LTL segments are received at the Relay node up to point[5] B. Hence, between time A and time B at the Relay buffer, we only have the presence of the outgoing satellite link flow (B_{sat}). The LTL flow is restored at time B and is stopped again at time C, when the buffer occupancy grows beyond B_{th}. Note that after point C, under the steady-state condition, an additional number of W_{LTL}^{max} packet is received. This is the number of segments that the LTL protocol (that is a TCP-link protocol) can send without receiving any ACK. Hence, between point C and D, W_{LTL}^{max} LTL segments are received (also in this interval of time we have the presence of both flows). After time D no more LTL segments are received. The ACK flow is restored again at time[6] E, where B decreases below B_{th}. The behavior of the Relay buffer occupancy continues in this way cyclically until all data has been transferred.

[4]We refer to the bandwidth of the i-th terrestrial link encountered along the path, $\mathcal{P}$, as $B_{wired}(i)$ and to the round trip time of the i-th terrestrial link as $RTT_{wired}(i)$.

[5]The time difference between point B and point A is given by $RTT_{wired}(1)$, i.e., the first terrestrial link round trip time.

[6]The first LTL packet is received at the Relay node $RTT_{wired}(1)$ seconds later.

We define t_{cycle} as the time elapsed between B and F (the buffer occupancy function period), where F is the end point of the cycle starting in B (see Fig. 3.3). To compute RTT_{UTL} we need to find t_{cycle} and the number of LTL segments transmitted during this period of time.

First of all we focus our attention on the time interval $[A, C]$. During this time interval the maximum input rate is given by

$$f_{in}(t) = \begin{cases} 0 & A \leq t < B \\ B_{wired}/LTL_{len} & B \leq t \leq C \end{cases} \tag{3.2}$$

where LTL_{len} is the length of a full LTL packet. The output flow is constant and it is due to the satellite link

$$f_{out}(t) = -B_{sat}/LTL'_{len} \tag{3.3}$$

where

$$LTL'_{len} = \frac{(LTL_{len} + RLH_{len})PDU_{len}}{PDU_{len}(payload)} \tag{3.4}$$

The parameter RLH_{len} represents the size of the header added by the Relay Node to each LTL segments to store connection information[7], whereas PDU_{len} and $PDU_{len}(payload)$ are the size of a full STPP packet and of its payload, respectively. In Eq. (3.4), we compute the number of STPP PDUs needed to transmit a full LTL packet ($(LTL_{len}+RLH_{len})/PDU_{len}(payload)$) and then we multiply it by the STPP PDU size. So, the number of transmitted bit per LTL packet at the STPP level can be easily obtained. Since the buffer in points A and C contains the same number of segments we can solve $\int_A^C (f_{in}(t) + f_{out}(t))dt = 0$. This integral can be easily solved by noting that $C - B = t_2$ and that $C - A = RTT_{wired}(1) + t_2$, hence

$$t_2 = \frac{B_{sat} RTT_{wired}(1)}{LTL'_{len}\left(\frac{B_{wired}}{LTL_{len}} - \frac{B_{sat}}{LTL'_{len}}\right)} \tag{3.5}$$

Moreover, the number of LTL segments received during time t_2 (interval $[B, C]$) is given by

$$r(t_2) = \frac{B_{wired}}{LTL_{len}} t_2 \tag{3.6}$$

The time interval t_3 can be easily obtained by observing in Fig. 3.3 that it is the time needed to the satellite link to free the buffer from W_{LTL}^{max} segments[8]

$$t_3 = \frac{LTL'_{len}}{B_{sat}} W_{LTL}^{max} \tag{3.7}$$

So, t_{cycle} is computed as $t_{cycle} = RTT_{wired}(1) + t_2 + t_3$.

In the following we compute the maximum time elapsed between the transmission of the first bit of a UTL packet and the instant at which the last LTL packet composing it is inserted into the Relay buffer. We refer to this time as $t(n)$. The worst situation is the one where the first LTL packet is inserted into the buffer at time D, i.e., it is the last of the W_{LTL}^{max} segments received after point C. In this case a time equal to $t_4 = F - D$ has to be waited before the second LTL packet will be inserted into the relay buffer.

[7] To reserve the bandwidth pipe and to keep other flow specific information (flow identifier, priority).

[8] These segments are the ones received after the ACK stopping time, i.e., time C.

$D - C$ can be easily computed by noting that it is the time needed to transmit W_{LTL}^{max} segments over the wired link. Hence $D - C = LTL_{len} W_{LTL}^{max} / B_{wired}$ and t_4 is obtained as

$$t_4 = t_3 + RTT_{wired}(1) - (D - C) \tag{3.8}$$

Furthermore, by noting that $r(t_{cycle}) = W_{LTL}^{max} + r(t_2)$, LTL segments are inserted into the Relay buffer for each following cycle and accounting for the fixed propagation delay ($t_{prop} = B_{wired}/LTL_{len} + RTT_{wired}(1)/2$) for the reception of the first LTL packet (of n) we can upper bound $t(n)$ as follows

$$\begin{aligned} t(n) &= t_{prop} + t_4 \\ &+ \left\lfloor \frac{n-1}{r(t_{cycle})} \right\rfloor t_{cycle} \\ &+ \left(n - 1 - \left\lfloor \frac{n-1}{r(t_{cycle})} \right\rfloor t_{cycle} \right) \frac{B_{wired}}{LTL_{len}} \end{aligned} \tag{3.9}$$

At this point, a worst case estimate of RTT_{UTL} can be computed as

$$\begin{aligned} RTT_{UTL} &\leq t_{init} + t(n) + t_{buffer} + t_{ACK} + RTT_{sat} \\ &+ \frac{RTT_{wired}(1)}{2} + RTT_{wired}(2) \end{aligned} \tag{3.10}$$

where

- t_{init} is the time elapsed between the beginning of the LTL transmission and the instant at which the buffer occupancy grows to B_{th} for the first time. t_{init} is computed as the lowest value of t that verifies the following inequality

  $$\left\lceil TX(t - RTT_{wired}(1)/2) - \frac{LTL'_{len}}{B_{sat}}(t - RTT_{wired}(1)/2) \right\rceil \geq B_{th} \tag{3.11}$$

  where the function $TX(t)$ (derived in the following Section 3.5) represents the number of segments transmitted by the LTL sender at time t, whereas $\frac{LTL'_{len}}{B_{sat}} t$ is the number of LTL segments transmitted over the satellite up to and including time t.

- $t(n)$ is an upper bound of the time needed to insert one entire UTL packet (n LTL segments) into the Relay buffer.

- t_{buffer}: in the worst case the n-th LTL packet arriving at the Relay Node finds $(W_{LTL}^{max} - 1 + B_{th})$ segments waiting in the Relay buffer. In this case the n-th packet is transmitted after a time t_{buffer} given by

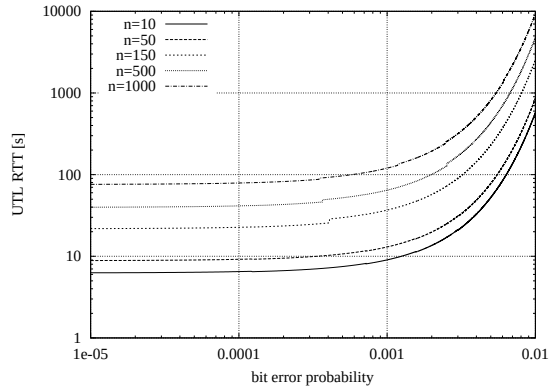
  $$t_{buffer} = \frac{LTL'_{len}}{B_{sat}}(W_{LTL}^{max} + B_{th}) \tag{3.12}$$

- t_{ACK} is the (total) transmission time of a full UTL ACK, it it obtained as the sum of the transmission time experienced by the ACK on all links

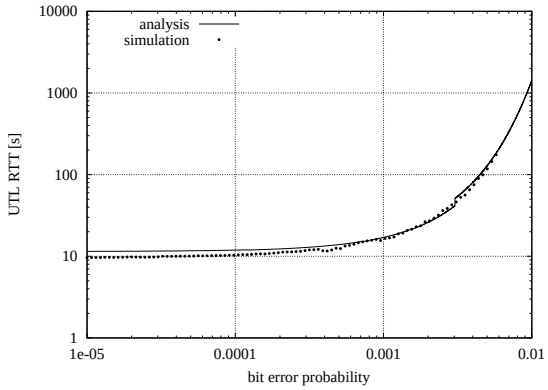
  $$t_{ACK} = \sum_{i \in \mathcal{P}} \frac{ACK_{len}}{B_{wired}(i)} + \frac{ACK'_{len}}{B_{sat}} \tag{3.13}$$

  where we consider the presence of a unique satellite link (B_{sat}) and ACK'_{len} is the number of bit needed to transmit an UTL ACK over the satellite channel.

Eq. (3.10) can be used to obtain the worst case estimate of the UTL round trip time in the error free case. Moreover, by considering a lossy link for what concerns the satellite channel, this equation can be still applied replacing B_{sat} with $B_{sat}^{(eq)}$ in the previous equations, where B_{sat}^{eq} is the equivalent satellite bandwidth in presence of satellite channel errors.



(a) Estimate of RTT_{UTL} as a function of the satellite channel bit error rate by varying the UTL batch size (n).



(b) UTL round trip time: comparison between estimate and simulation, $B_{wired}(1) = B_{wired}(2) =$ 2 Mbps, $B_{sat} =$ 128 Kbps, $RTT_{wired}(1) = RTT_{wired}(2) = 100$ ms, $n = 100$.

Figure 3.4: Characterization of the UTL round trip delay (RTT_{UTL}).

If ε is the STPP PDU error probability[9], B_{sat}^{eq} is computed as $B_{sat}^{eq} = B_{sat}(1-\varepsilon)$. In Fig. 3.4(a) we report the RTT_{UTL} estimate as a function of the satellite channel bit error probability by varying n. We considered the scenario in Fig. 3.2 with $B_{wired}(1) = B_{wired}(2) = 2$ Mbps, $RTT_{wired}(1) = RTT_{wired}(2) = 100$ ms, $RTT_{sat} = 0.5$ s, $LTL_{len} = 1$ kbyte and $B_{sat} = 128$ Kbps. In Fig. 3.4(b) we report the comparison between simulation points and analysis considering the same parameters for $n = 100$.

3.5 Derivation of the Function $TX(\cdot)$

We refer to standard TCP that, after the connection setup phase, is initiating the transmission phase. At the beginning of this phase, the TCP is in Slow-Start, i.e., its congestion window, W, is incremented by one full segment for each received ACK. This phase finishes when W reaches the TCP window threshold value, W_{thresh} that at the beginning of the connection is always set to the maximum congestion window size W_{max}. Here we are interested in the computation of the function $TX(t)$ (assuming no packet errors), i.e., on the number of full transmitted TCP segments between time 0 (where the connection started) and time $t \geq 0$. We call t_{tx} the time needed to transmit a full TCP packet, $t_{tx} = TCP_{len}/B_{wired}$, where TCP_{len} is the total length (header and payload)[10] of a TCP packet expressed in bits, whereas B_{wired} is

[9] Here we consider the independent error case.

[10] In this document we consider the TCP segments to be of fixed size given by the Maximum segment size, MSS.

the available bandwidth on the terrestrial part of the connection. We approximate[11] the number of TCP segments transmitted in one entire RTT_{wired} with the quantity: $n_{rtt} = \lfloor RTT_{wired}/t_{tx} \rfloor$. Moreover, the instant t can be written as the sum of two terms:

$$t = t_1 + t_2 = \left\lfloor \frac{t}{RTT_{wired}} \right\rfloor RTT_{wired} + \Delta_t \tag{3.14}$$

where the first term (t_1) accounts for the number of complete rounds while the second ($t_2 = \Delta_t$) gives the fraction of time elapsed in the current round. In the no delayed ACK case, the window size at the generic round i is equal to: $W(i) = \min(2^{i-1}, W_{max})$ The number corresponding to the current round (starting from round 1) is given by: $c_r = \left\lfloor \frac{t}{RTT_{wired}} \right\rfloor + 1$. Hence the maximum window W_{cr} reached in the current round (c_r) can be computed as $W_{cr} = W(c_r)$. $TX(t)$ can be computed by summing $TX(t_1)$ to $TX(t_2)$. $TX(t_1)$ is computed as follows:

$$TX(t_1) = \begin{cases} \sum_{i=1}^{c_r-1} \min(2^{i-1}, n_{rtt}) & c_r > 1 \\ 0 & \text{elsewhere} \end{cases} \tag{3.15}$$

For what concerns $TX(t_2)$:

$$TX(t_2) = \begin{cases} p & W_{cr} \geq p \\ n_{rtt} & W_{cr} < p \end{cases} \tag{3.16}$$

where p is the integer number of transmission periods in t_2: $p = \lfloor (t - t_1)/t_{tx} \rfloor$. Finally, $TX(t)$ is obtained by $TX(t) = TX(t_1) + TX(t_2)$.

3.5.1 Dimensioning of the UTL Window Size

The UTL window size must be chosen to ensure that during the time elapsed between the departure of a UTL packet and the instant at which the End-to-End acknowledgment is received, the satellite link is always filled by transmitted segments. By expressing the UTL window size (W_{UTL}) in unit of UTL packets, this condition is verified when the following inequality holds

$$W_{UTL} \geq \left\lceil \frac{B_{sat} RTT_{UTL}}{UTL'_{len}} \right\rceil \tag{3.17}$$

where UTL'_{len} is the number of bits needed to transmit a full UTL packet over the satellite channel (including STPP overhead). The relation in Eq. (3.17) is valid as long as the satellite link is a bottleneck for the connection, i.e., B_{sat} is lower than the bandwidth of all the links upstream the satellite channel.

3.5.2 Dimensioning the Relay Buffer Threshold

The correct dimensioning of the Relay buffer threshold (B_{th}) is a pivotal point in the design of the PETRA architecture. This threshold is responsible of flow control at the Relay Nodes and (as will be

[11] The error introduced in the computation of $TX(t)$ is negligible when B_{wired} is high. When this error is not negligible (at very low B_{wired}), $TX(t)$ is always less than the actual number of transmitted segments, hence the analysis performed in this document is still valid (being a worst case dimensioning).

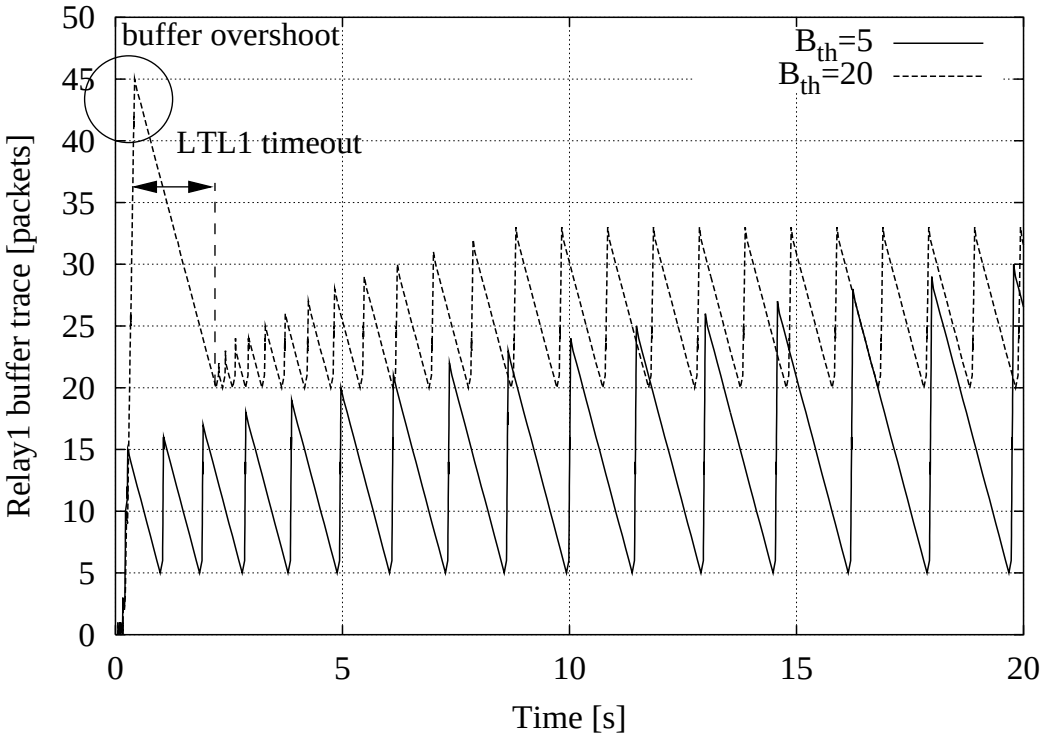


Figure 3.5: Relay Buffer trace considering $B_{wired}(1) = B_{wired}(2) = 2$ Mbps, $B_{sat} = 128$ Kbps, $RTT_{wired}(1) = RTT_{wired}(2) = 100$ ms, $RTT_{sat} = 0.5$ s, $W_{max}^{LTL} = 32$ LTL segments, $LTL_{len}(payload) = 1$ Kbyte, $LTL_{len}(header) = 40$ bytes.

shown in the following) misconfigurations could lead to LTL premature timeouts. Focusing on the scenario depicted in Fig. 3.2, we consider first the Relay node placed in the leftmost side of the figure. When $B_{sat} \geq B_{wired}(1)$, the choice of B_{th} at such Relay is irrelevant because the buffer in this situation is always empty. In the case where $B_{sat} < B_{wired}(1)$ instead a correct dimensioning of the threshold is required. In particular, if the values of B_{th} is too large, the following events would occur

1. In the initial connection phase a large number of LTL segments (W^*) can be stored into the Relay buffer before the LTL ACK flow is stopped (see Fig. 3.5).

2. The LTL during this time period experiences a very short round trip time (equal to RTT_{wired}) and estimates the timeout ($T_o(LTL)$) accordingly.

3. After the ACK stopping point, a long period is needed to the satellite link to restore the buffer occupancy below B_{th} and during this period the LTL flow remains stopped.

4. If the length of this period is greater to or equal than $T_o(LTL)$ (the timeout estimate in point 2) then a LTL timeout event has to be waited for.

In order to avoid the LTL timeout problem, B_{th} must be set to a small value. In this way W^* is limited and the LTL timeout event probability is decreased. In general, the smaller B_{th}, the lower the LTL timeout event probability. However, a necessary condition to ensure that the satellite link is always

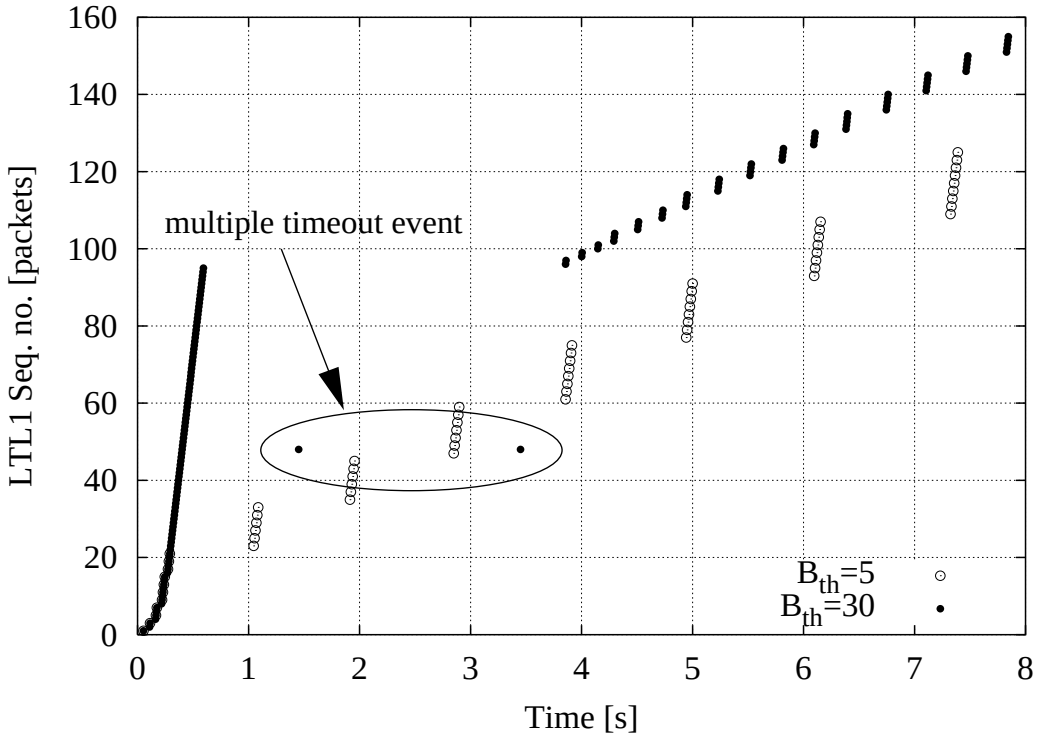


Figure 3.6: LTL packet trace with $W_{max}^{LTL} = 50$ LTL segments, $B_{wired}(1) = B_{wired}(2) = 2$ Mbps, $B_{sat} = 128$ Kbps, $RTT_{wired}(1) = RTT_{wired}(2) = 100$ ms, $RTT_{sat} = 0.5$ s, $LTL_{len}(payload) = 1$ Kbyte, $LTL_{len}(header) = 40$ bytes.

filled by transmitted segments is that the Relay buffer *must be never emptied*. This condition poses a lower bound on the minimum allowed value of B_{th}

$$B_{th} \geq B_{th}^{min} \tag{3.18}$$

Now, noting that:

- The time elapsed at the Relay buffer between the instant when the buffer size decreases below B_{th} (the LTL ACK flow is restored) and the reception of the first LTL packet of the restored flow is equal to $RTT_{wired}(1)$.
- The satellite channel during this time period consumes $\lceil B_{sat} RTT_{wired}(1)/LTL'_{len} \rceil$ LTL segments (LTL'_{len} is the number of bits needed to transmit a full LTL packet over the satellite link).

Condition in Eq. (3.18) can be written as follows

$$B_{th} \geq \left\lceil \frac{B_{sat} RTT_{wired}}{LTL'_{len}} \right\rceil \tag{3.19}$$

Hence, the B_{th} that minimizes the timeout event probability is the minimum value verifying the condition in Eq. (3.19), i.e., $B_{th}^{min} = \lceil B_{sat} RTT_{wired}/LTL'_{len} \rceil$. As an example, in Fig. 3.5 we report the Relay buffer occupancy as a function of the time for the two cases where $B_{th} = 20 (\gg B_{th}^{min} = 1)$ and $B_{th} = 5$. In the last case the timeout event is avoided and the satellite channel is always filled by

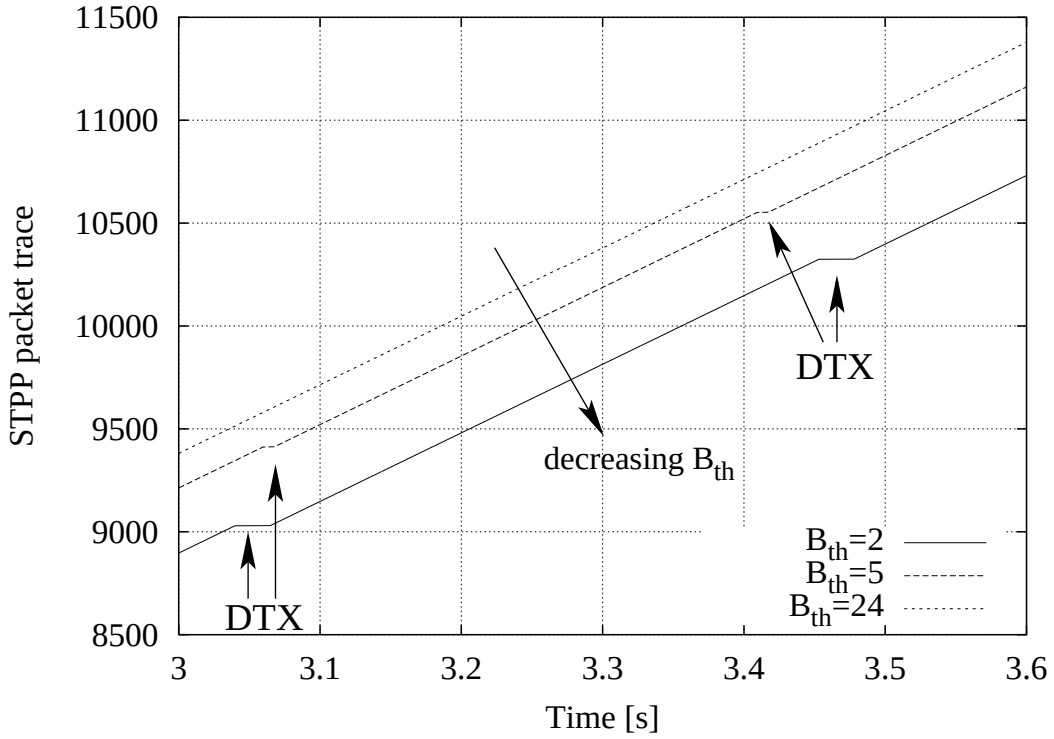


Figure 3.7: STPP PDU trace: discontinuous transmission problem, $W_{max}^{LTL} = 32$ LTL segments, $B_{wired}(1) = B_{wired}(2) = 4$ Mbps, $B_{sat} = 2$ Mbps, $RTT_{wired}(1) = RTT_{wired}(2) = 100$ ms, $RTT_{sat} = 0.5$ s, $LTL_{len}(payload) = 1$ Kbyte, $LTL_{len}(header) = 40$ bytes.

transmitted PDUs. On the contrary, when B_{th} is too large a spurious timeout event is observed for the reasons discussed above. Note that the problem addressed here could heavily degradate the performance of the LTL running over terrestrial paths, especially in the case where $B_{sat} \ll B_{wired}(1)$. Moreover, in Fig. 3.5 ($B_{th} = 20$) we have only one timeout event (at the beginning of the connection); after this timeout the LTL connection stabilizes and no more timeouts are observed. However, some particular cases exist in which timeout events occur repeatedly when the buffer threshold is misconfigured. In Fig. 3.6 we show how for particular values of the system parameters, a misconfigured B_{th} can lead to multiple LTL timeout events, whereas in Fig. 3.7 we report the discontinuous transmission effect on the STPP flow due to a too small B_{th} value (B_{th}^{min} in this case is equal to 24 LTL segments). Similar considerations apply to the Relay node in the rightmost side of Fig. 3.2.

3.5.3 Dimensioning the Relay Buffer Size

We first proceed to the dimensioning of the buffer on the leftmost side of Fig. 3.2.

Consider the buffer behavior reported in Fig. 3.3. We observe that in order to avoid buffer losses, the relay buffer size (B_{size}) must be greater than the value reached in point D. Specifically, at time C exactly B_{th} LTL segments are stored into the buffer, while during the interval $(C, D]$, at most W_{max} further segments arrive at the relay node. Moreover, during $(C, D]$ exactly $(B_{sat}/LTL'_{len})(D - C)$ segments

are consumed by the satellite level. Hence, the buffer size in point D, $BS(D)$ is given by

$$\begin{aligned} BS(D) &= B_{th} + W_{max}^{LTL} - \frac{B_{sat}}{LTL'_{len}}(t_{LTL}W_{max}^{LTL}) \\ &= B_{th} + W_{max}^{LTL} - \frac{B_{sat}}{B_{wired}(1)}\frac{LTL_{len}}{LTL'_{len}}W_{max}^{LTL} \end{aligned} \tag{3.20}$$

where t_{LTL} is the transmission time of one LTL packet over the first terrestrial link, $t_{LTL} = LTL_{len}/B_{wired}(1)$. If the satellite bandwidth is completely utilized the correct value of B_{size} is given by

$$B_{size} = \left\lceil B_{th} + W_{max}^{LTL} - \frac{B_{sat}}{B_{wired}(1)}\frac{LTL_{len}}{LTL'_{len}}W_{max}^{LTL} \right\rceil \tag{3.21}$$

Due to channel impairments, the available bandwidth B_{sat} may be less than the one used in Eq. (3.20). If this is the case, the correct value of the buffer size is computed as follows

$$B_{size} = B_{th} + W_{max}^{LTL} \tag{3.22}$$

We suggest to use Eq. (3.22) in order to set the relay buffer size at Node 1 under all possible satellite channel error conditions.

Note that, for the correct working of the presented network architecture, the dimensioning of the relay buffer is a crucial point. In fact, if a packet is lost due to buffer overflow, no End-to-End retransmissions of the UTL batch including that packet are performed by UTL and a UTL timeout event has to be waited for. In this case, the connection is closed by UTL upon timeout timer expiration. For this reason, the buffer size *must be* dimensioned in order to completely avoid the buffer overflow problem. Similar reasoning apply to the dimensioning of the buffer placed on the rightmost part of Fig. 3.2, where input and output buffer flows are due to STPP and LTL, respectively.

3.5.4 Dimensioning of the STPP Packet Size

Another issue in the design of the STPP protocol is the choice of the STPP packet size. Regarding this point, there is a trade-off between the overhead added by STPP PDU headers and CRC fields and the STPP PDU error probability. Defining P_e as the bit error probability at the STPP level[12], the maximum STPP throughput is given by[13]

$$f(P, O, P_e) = \frac{P}{P+O}(1 - (1 - P_e)^{P+O}) \tag{3.23}$$

where P and O are $PDU_{len}(payload)$ and $PDU_{len} - PDU_{len}(payload)$ as defined in Subsection 3.4.1. To find the optimal value for P given $O = O^*$ and $P_e = P_e^*$ it is sufficient to solve

$$\left.\frac{\partial f(P, O, P_e)}{\partial P}\right|_{O=O^*, P_e=P_e^*} = 0 \tag{3.24}$$

[12] It is the bit error probability after channel coding and interleaving operations. Here an independent error process is considered.

[13] Considering that the STPP level is rightly confi gured so to have a continuous transmission fbw, i.e., the available bandwidth over the satellite link (B_{sat}) is fully exploited.

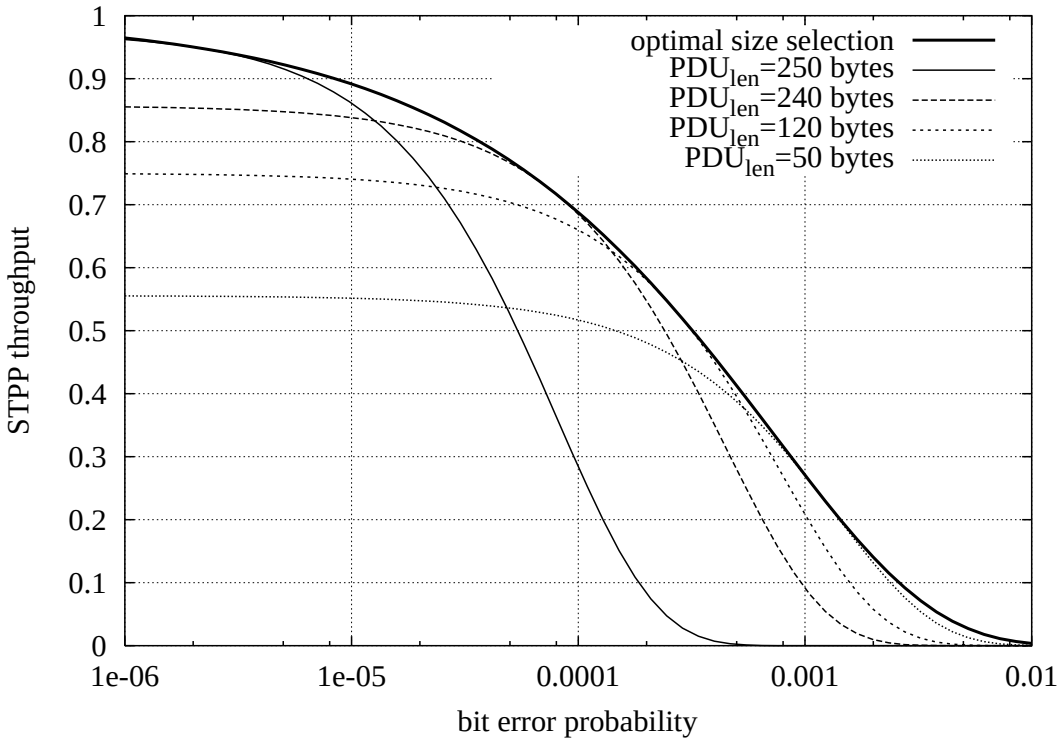


Figure 3.8: STPP throughput as a function of the satellite channel bit error probability. Comparison between fixed PDU_{len} and optimal size selection.

The optimal value of the PDU payload size is then given by

$$P_{opt} = \frac{-O^*\sqrt{p} + \sqrt{O^*(4 + pO^*)}}{2\sqrt{p}} \tag{3.25}$$

where $p = |\log(1 - P_e^*)|$. In Fig. 3.8, we report the STPP throughput as a function of the bit error probability considering both the case where O is constant and the case where (solid bold line in Fig. 3.8) O is chosen in order to maximize the STPP throughput (using Eq. (3.25)).

3.5.5 Dimensioning the Bandwidth in the Reverse Channel

In order to dimension the satellite reverse channel, the bandwidth requirements must be evaluated. Let first evaluate the average size of a STAT message, $SSTAT_{len}$. This is given by the size of the STPP PDU header and the average size of the mask utilized to identify the STPP PDUs which have been received corrupted or have not been received at all. Accordingly, the average size of the STAT message, $SSTAT_{len}$. can be evaluated as follows

$$SSTAT_{len} = O + \alpha \cdot n_e \tag{3.26}$$

where:

- O is the STPP PDU header size.
- α is the number of bits required to signal an erroneous STPP PDU in the STAT mask.

- n_e is the average number of corrupted PDUs in a time interval equal to the satellite round trip time, RTT_{sat}. If we denote ε_{PDU} the PDU error probability in the satellite forward channel, n_e can be computed as follows

$$n_e = \varepsilon_{PDU} \cdot B_{sat} \cdot RTT_{sat}/PDU_{len} \tag{3.27}$$

Now, the average bandwidth required in the satellite reverse channel, $B_{STPP}^{(Rev)}$, can be easily evaluated

$$B_{STPP}^{(Rev)} = SSTAT_{len}/T_{stat} \tag{3.28}$$

3.6 Performance Evaluation

In this Section PETRA performance is evaluated by using analysis where possible and simulation otherwise. PETRA performance will be compared to the performance obtained in the following cases:

- *TCP E2E Case*: Traditional TCP-NewReno is utilized End-to-End. Moreover, in the satellite segment a Selective Repeat ARQ scheme is implemented at the link layer to counteract wireless channel errors.
- *Split-TCP Case*: A splitted-TCP solution is utilized[14]. In the satellite channel, a Selective Repeat ARQ technique which assigns higher priority to retransmission is considered. The receiver sends back a NACK for each erroneous PDU. At least one acknowledgment message per round trip time is transmitted to advance the sender window.

The last solution allows keeping a constant information about the status since each erroneous packet is tracked. However, when the forward channel error rate is relevant this scheme is bandwidth consuming. The measure of the reverse bandwidth gain obtained by using PETRA with respect to Split-TCP is reported in the following Subsection 3.6.3.

3.6.1 Effects of Link Errors in the Satellite Forward Channel

Results shown in this Section have been obtained by simulating the transfer of a 10 Mbytes file. In Fig. 3.9, we show the throughput performance versus the bit error probability achieved using PETRA. We considered different values of T_{stat} and assumed the satellite feedback channel and the wired segments to be error free. For the sake of comparison, in the same figure throughput performance obtained in the Split-TCP and in the E2E case are also reported. Throughput has been evaluated as follows:

$$\text{throughput} = \frac{\text{data transferred [bit]}}{\text{transfer time [s]} \times B_{sat}\text{[bit/s]}} \tag{3.29}$$

In Fig. 3.9, it is evident that the choice of T_{stat} does not have significant impact on the throughput performance (it is only necessary that at least one STAT message per round trip time is correctly received to advance the window at the sender). The throughput achieved by PETRA is considerably higher than in the E2E Case and close to that achieved in the Split-TCP Case. This is due to the fact that a correct management of the relaying buffer is performed so that losses due to buffer overflow and LTL timeout

[14] We consider Split-TCP because it achieves the best throughput performance [86].

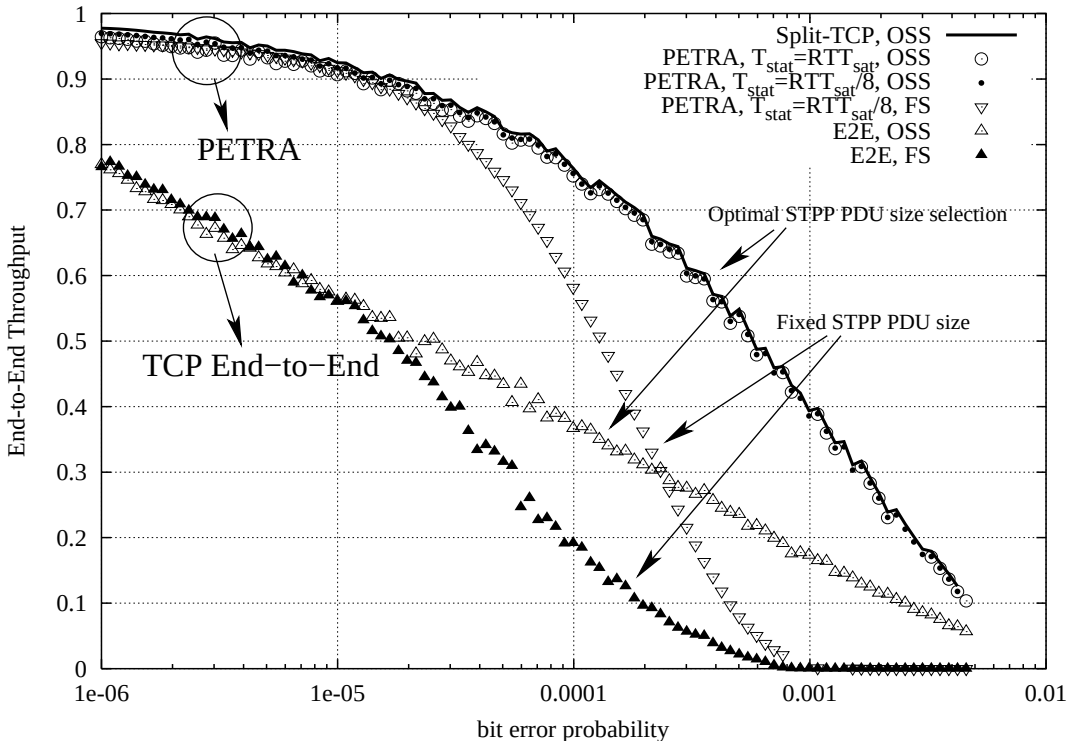


Figure 3.9: End-to-end Throughput as a function of the satellite channel error probability. $RTT_{wired}(1) = RTT_{wired}(2) = 100$ ms, $RTT_{sat} = 0.5$ s, $B_{sat} = 384$ Kbps, $B_{wired}(1) = B_{wired}(2) =$ 2 Mbps, $LTL_{len}(payload) = 1$ Kbyte, $LTL_{len}(header) = 40$ bytes, $W_{max}^{LTL} = 32$ LTL segments, error-free terrestrial segments, 10 Mbyte data transfer, $T_o(UTL) = 20$ s.

events are avoided. Obviously, the good PETRA performance is obtained at the expense of higher computational complexity and the overhead introduced by the double transport layering. Moreover, to evaluate the impact of an optimal STPP PDU size selection as a function of the forward channel error rate, in Fig. 3.9 two sets of results are shown:

- *OSS*: The optimal STPP PDU size is utilized.
- *FS*: A constant STPP PDU size is utilized for any value of the bit error probability. In Fig. 3.9, the payload size has been set to 600 bytes, whereas the STPP overhead (header and CRC fields) is fixed to 20 bytes in both cases (OSS and FS).

Fig. 3.10 shows the average delivery delay of the STPP segments versus the bit error probability in the satellite forward channel. In other words, while Fig. 3.9 contains the average end-to-end throughput, Fig. 3.10 allows measuring what happens over the satellite portion of the network and focusing on the role of the parameter T_{stat}. If the bit error probability is low, the impact of T_{stat} on the delay is negligible, but if the channel is severely errored, the performance heavily depends on its choice. In more detail, the performance of PETRA is close to the one achieved in the Split-TCP case if the STATUS message is transmitted frequently ($T_{stat} = RTT_{sat}/8$); the delay increases when the information about the status is operated after a larger lapse of time. It is interesting to observe that the delay of the STPP segments over the satellite link has no impact on the end-to-end throughput, as clear from Fig. 3.9, where the

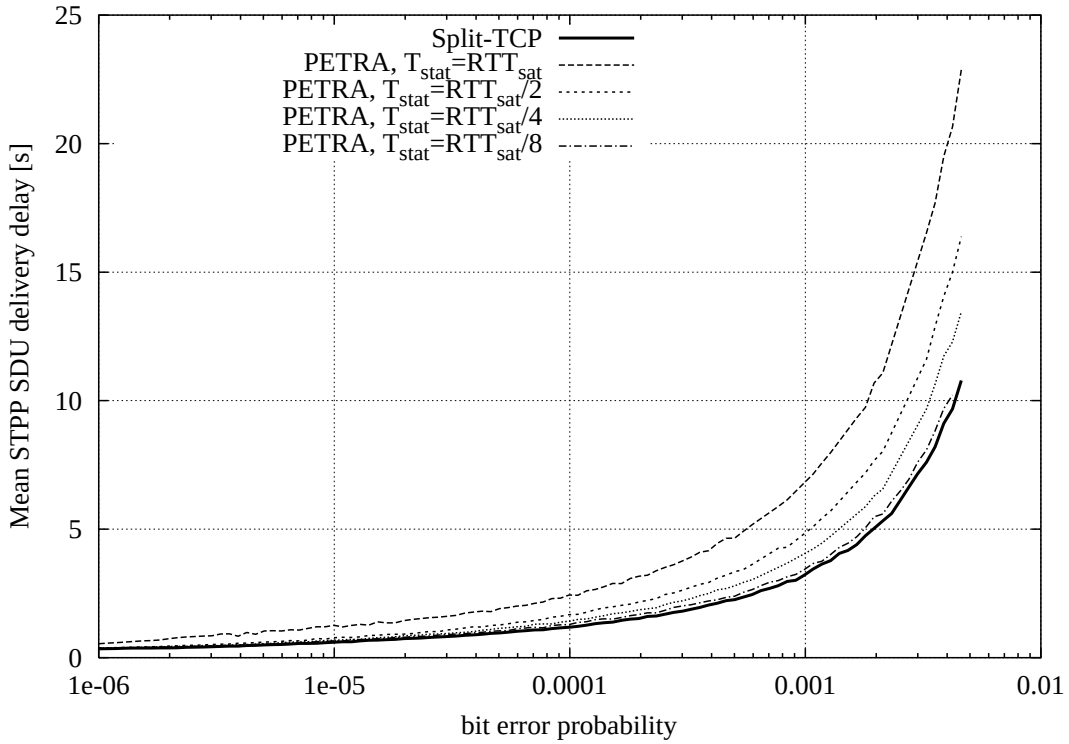


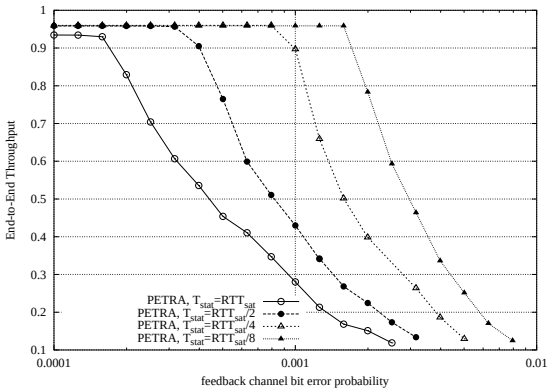
Figure 3.10: Mean STPP SDU delivery delay as a function of the satellite channel bit error rate. $RTT_{wired}(1) = RTT_{wired}(2) = 100$ ms, $RTT_{sat} = 0.5$ s, $B_{sat} = 384$ Kbps, $B_{wired}(1) = B_{wired}(2) =$ 2 Mbps, $LTL_{len}(payload) = 1$ Kbyte, $LTL_{len}(header) = 40$ bytes, $W_{max}^{LTL} = 32$ LTL segments, error-free terrestrial segments, 10 Mbyte data transfer, $T_o(UTL) = 20$ s.

performance offered by "PETRA $T_{stat} = RTT_{sat}$" and "PETRA $T_{stat} = RTT_{sat}/8$" is completely equivalent. It means that the overall architecture, by means of properly dimensioned buffering at the relay entities, can match the delay increase without affecting the end-to-end throughput performance. This is not true if the feedback channel is errored, as will be clear in the next figures.

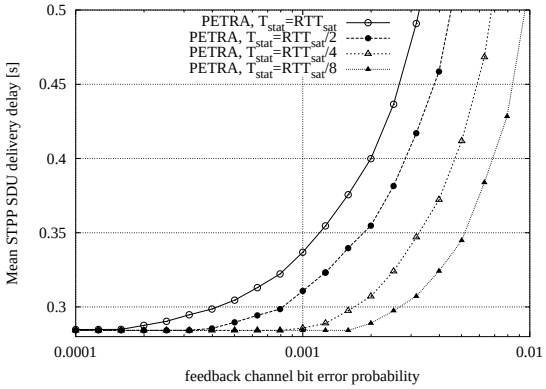
3.6.2 Effects of Link Errors in the Satellite Feedback Channel

Here we assume a very low (10^{-9}) satellite forward channel bit error probability, while we consider the satellite feedback channel unreliable. In Figs. 3.11(a) and 3.11(b), we report the throughput performance and average LTL packet delivery delay, respectively. Both throughput and delay are given versus the bit error probability in the satellite feedback channel for different values of the STAT message period, T_{stat}. The results presented in both figures were obtained by considering the following simulation parameters: STPP $PDU_{len} = 620$ bytes, $RTT_{sat} = 0.5$ s, $RTT_{wired}(1) = RTT_{wired}(2) =$ 100 ms, $B_{wired}(1) = B_{wired}(2) = 10$ Mbps, $B_{sat} = 384$ Kbps, $LTL_{len}(header) = 40$ bytes, $LTL_{len}(payload) = 1$ Kbyte, the forward channel bit error probability is 10^{-9}.

In Figs. 3.11(a) and 3.11(b), we observe that performance improves as the T_{stat} value decreases. This is because lower values of T_{stat} result in a larger number of STAT messages which allow a more efficient error recovery as well as a faster advancement of the sliding window. As envisaged in the previous Subsection, if, as in this case, the feedback channel is severely errored, also a slight delay of the



(a) PETRA end-to-end throughput as a function of the reverse channel bit error probability by varying T_{stat}.



(b) PETRA, mean STPP SDU delivery delay as a function of the reverse channel bit error probability by varying T_{stat}.

Figure 3.11: Performance as a function of the reverse channel error probability.

order of tens of milliseconds (Fig. 3.11(b)) corresponds to an heavy end-to-end performance degradation (Fig. 3.11(a)). This highlights the importance of the correct delivery of the STATUS information for the system performance, i.e., *at least one STATUS message must be correctly received in every RTT_{sat} in order to advance the STPP window*.

Anyway, the results offered by PETRA scheme are really satisfying: on one hand, bit error probabilities above 0.001 are a real worst case and very rare over real satellite systems; on the other hand, also in these cases, PETRA can be configured (e.g., $T_{stat} = RTT_{sat}/8$) so to be very robust against forward and feedback channel errors and to offer performance suited for any type of application.

3.6.3 STPP Performance

Let $B^{(Rev)}_{Split-TCP}$ the average bandwidth utilized in the satellite reverse channel for the *Split-TCP* case. Then, we can compute the *relative bandwidth gain* as:

$$\text{gain} = \frac{B^{(Rev)}_{Split-TCP} - B^{(Rev)}_{STPP}}{B^{(Rev)}_{STPP}} \tag{3.30}$$

Observe that analytical evaluation of the gain value can be easily achieved. In fact, the value of $B^{(Rev)}_{STPP}$ can be computed as given in Eq. (3.28), whereas $B^{(Rev)}_{Split-TCP}$ is given by:

$$B^{(Rev)}_{Split-TCP} = \frac{\max(1, n_e) \cdot O}{RTT_{sat}} \tag{3.31}$$

where n_e is given in Eq. (3.27).

In Fig. 3.12 we report the relative feedback bandwidth gain as a function of the satellite channel bit error rate. We considered the optimal PDU size and different values of α. Observe, that the gain heavily depends on the way in which the error mask is stored into STAT PDUs (α factor, defined in

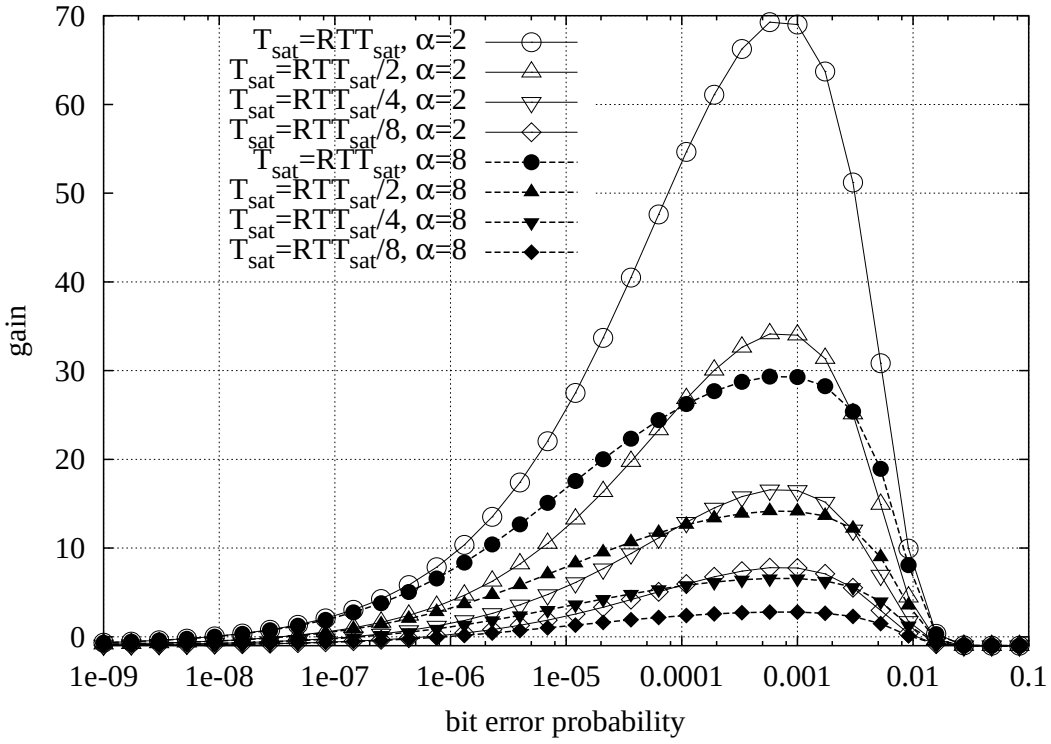


Figure 3.12: Relative bandwidth gain between STPP and NACK based algorithm (Split-TCP).

Section 3.5.5 as the average number of bits required to signal an erroneous STPP PDU in a STATUS message). TCP split is a bandwidth consuming scheme; PETRA allows to save bandwidth; the measure of the saving depends on the average number of bits employed to signal the status. A simple algorithm could be implemented by transmitting a vector (the STAT mask) containing: the sequence number of the first packet whose status is described ($SNfirst$, 8 bits), the sequence number of the last packet ($SNlast$, 8 bits) and ($SNlast - SNfirst + 1$) fields of 1 bit where the symbol "0" means *correct* and "1" means *errored*. In this case, if the status of 100 segments is transmitted, the average number of bits employed (α) is $116/100 = 1.16$. Alternatively, if all the sequence numbers were transmitted, 100 fields of 8 bits each would be required ($\alpha = 8$). Other schemes, including compression algorithms developed for image processing, can be applied, so reducing the α value (also below 1). On the other hand, some redundancy may be necessary to protect status information and, in this case, the α value would increase. A couple of example values have been chosen for our tests: $\alpha = 2$ and $\alpha = 8$.

At low bit error rates the NACK based solution can outperforms the STPP one (when $RTT_{sat}/T_{stat} > 1$). This is due to the fact that STPP always transmits RTT_{sat}/T_{stat} STAT messages in each RTT_{sat}. Instead, at low error rates, the number of NACKs sent in a round trip time in the Split-TCP Case is equal to one. Moreover, observe that the relative bandwidth gain increases as T_{stat} increases. It is important to observe that the solution "PETRA $T_{stat} = RTT_{stat}/8$", which offers good and robust performance, can guarantee also a relevant bandwidth gain if employed with a low α value. The value $\alpha = 2$, shown in the results, may be further reduced, as said above.

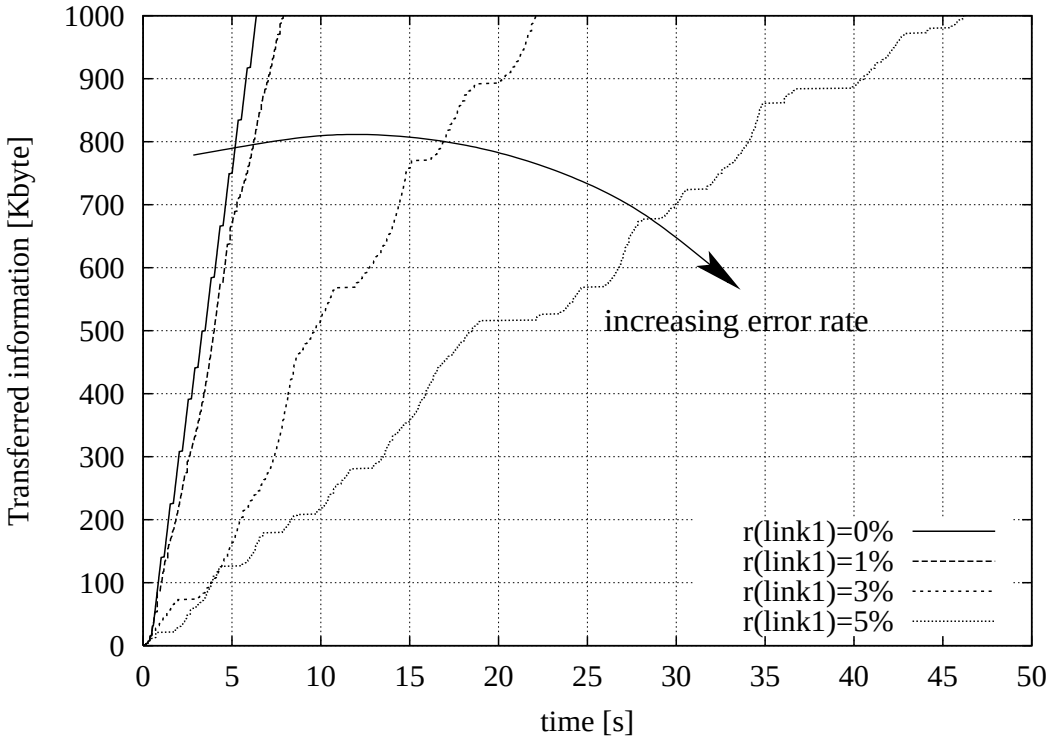


Figure 3.13: End-to-end transferred data: $RTT_{wired}(1) = RTT_{wired}(2) = 100$ ms, $RTT_{sat} = 0.5$ s, satellite channel bit error rate $P_e = 0.0001$, $B_{sat} = 2$ Mbps, $B_{wired}(1) = B_{wired}(2) = 2$ Mbps, $LTL_{len}(payload) = 1$ Kbyte, $LTL_{len}(header) = 40$ bytes, $W_{max}^{LTL} = 32$ LTL segments, 1 Mbyte data transfer, $T_o(UTL) = 20$ s.

3.6.4 Effects of Additional Errors on Terrestrial Paths

Referring to Fig. 3.2, we label as $link1$ and $link2$ the leftmost and the rightmost terrestrial links, respectively. Moreover, we denote the packet error rate affecting the two above links as $r(link1)$ and $r(link2)$.

Here, as an example we consider only the case in which errors affect the first terrestrial link, i.e., $link1$. In Fig. 3.13, we show the simulation traces of the end-to-end transmitted bytes for different values of the link error rate $r(link1)$. As expected, as $r(link1)$ increases, performance decreases. As an example, when compared to the error free case, the LTL *transfer time* triples when the packet error rate is $r(link1) = 3\%$, whereas it becomes nine times longer by considering an error rate of 5%. This is because due to the above errors, the wired segment becomes the bottleneck and therefore, the satellite pipe cannot be filled.

3.7 Conclusions

In this Chapter a satellite adapted transport architecture (PETRA) has been presented. This architecture can be deployed to improve performance in communication scenarios involving satellite systems.

PETRA is based on the split-TCP policy. The end-to-end connection is divided into different segments, where a different transport protocols can be deployed in each network segment. Accordingly,

customized solutions can be used in the satellite segments, while traditional TCP should be applied in the terrestrial segments. This provides fairness with respect to other TCP flows in the network, and allows standard TCP applications to be easily ported to our architecture. In order to guarantee reliability, the end-to-end semantic is maintained by introducing an appropriate Upper Transport Layer (UTL).

The proposed architecture has been presented in detail, and all the design parameters have been appropriately dimensioned. Both analysis and simulation results have been presented to assess our architecture, which has shown very good performance. A proper tuning of the parameters guarantees satisfying results as well as bandwidth savings, also in presence of relevant bit errors rates both over forward and feedback channels.

Chapter 4

Throughput Analysis of TCP/IP over Wireless Links with Finite Round Trip Delay

In this Chapter, an accurate analytical model for TCP over correlated channels (e.g., as induced by multi-path fading) and a finite round trip delay is introduced. In particular, models and analysis for studying four versions of TCP are developed. The TCP versions considered here are Old Tahoe, Tahoe, Reno, and NewReno [97] [44] [14]. The focus is on a single wireless TCP connection, and the correlated packet loss/error process is modeled as a Discrete Time first-order Markov Chain (DTMC) [52]. The model presented here explicitly incorporates important aspects such as *slow start, congestion avoidance, fast retransmit* and *fast recovery*. The main findings of this study are that (1) an increasing round trip time may significantly affect the throughput performance of TCP, especially when an independent channel is considered, (2) New Reno performs better than Reno and Tahoe when the channel is uncorrelated, whereas Tahoe's recovery strategy is the most efficient when the channel correlation is high and (3) the maximum window size does not play a determinant role in increasing throughput performance in both correlated and independent channels. While some of these conclusions confirm what other authors have observed in simulation studies, our analytical approach sheds some new light on TCP's behavior.

4.1 Introduction

The increasing popularity of wireless networks indicates that wireless links will play an important role in the future internetworking. Transport Control Protocol (TCP) is a reliable, end-to-end, transport protocol that is widely used to support applications like *e-mail, telnet, ftp,* and *http*. TCP was designed primarily for wireline networks where packet losses are caused mainly by congestion. Several modifications have been proposed to the TCP loss-recovery and congestion-control mechanism to improve data throughput, including Reno, New Reno, and Vegas.

In the recent literature, many papers have appeared on the topic of TCP performance in wireless systems. Some of those papers have as their main goal to describe new behaviors, usually observed from results obtained by simulation. Some others are more concerned with modeling TCP operation by means

of analytical techniques, in order to provide more general tools and to develop more accurate descriptions while controlling the complexity. The goal of this study is to provide an accurate analytical model for TCP operation.

The analytical characterization of TCP has been already investigated in previous studies. In [70], for example, the TCP throughput is obtained for large round trip delays, but this paper focuses on channel described by means of an independent and identically distributed (*iid*) error process. In [8], Hellal *et al.* studied the behavior of TCP Reno and Vegas considering an error-free wireline network, where a bottleneck link is the only cause of packet losses. On the other hand, Padhye *et al.* [85], have derived a useful formula for the TCP Reno throughput over wireline networks, considering the error process as *iid*. In [85], the correlation among losses is tracked assuming that, considering a window of transmitted packets, whenever a packet is lost, all the other packets in the same window are lost as well (this model is referred here with the term *quasi static channel*). All these papers consider a *static* (or *quasi static*) channel and a wired network. A correlated channel has been considered in [68] and in [106], where TCP is studied over a wireless link and a two-state Markov model is used to describe the channel burstiness. An instantaneous feedback is considered, i.e., the acknowledge message (ACK) is received immediately after the completion of the packet transmission. This idealized model was adopted to limit the analytical complexity of the approach. This simplification leads to overestimating the TCP performance as the bandwidth-delay product increases.

In [5] a TCP mean throughput analysis over finite round trip delay channels is reported considering both independent and correlated losses. In that paper, the delay between the instant where a packet loss occurs and the instant in which it is detected by the TCP sender is neglected. This assumption leads to accurate results where the round trip time is small. However, this is not true for large round trip times, e.g., as the ones envisioned in 3G cellular networks. In these scenarios End-to-End TCP round trip times can grow up to $RTT = 0.5$ seconds and the delay between error events and error detection can be estimated as $RTT + 3\mu$, where μ is the TCP packet transmission time. Moreover, in [5] it is assumed that the congestion window restarts from 1 after multiple timeout events and that, in the correlated error case and when the timeout timer-expiry period is greater than the bad state duration, the error event is detected by the Fast Recovery algorithm with probability one (without entering timeout). In [16] the TCP throughput performance in correlated channels with finite round trip delay is also investigated. This analysis enables qualitative and meaningful comparison between different schemes, but the accuracy of the model is not addressed. The distribution of the number of consecutive packet drops in a bad state is imposed, regardless of the time duration between the transmission of these packets. This assumption is only valid when packet transmission is continuous. However, TCP is a bursty protocol that introduces (sometimes considerables) intermittent idle durations during the lifetime of a connection. This assumption holds when the window size is always equal to or greater than the bandwidth delay product of the link. This is not true in general and the assumption above in these cases may lead to inaccurate results.

The analysis presented in the following uses a semi-Markov chain to find throughput performance. However, it is worth noting that it has the potential to be extended to find energy consumption metrics as well. To the best of our knowledge, this analysis is more accurate than any other analysis previously proposed in the literature. All the simplifying assumptions discussed above have been removed in the model presented here, by constructing a semi-Markov chain able to jointly track window evolution, successes,

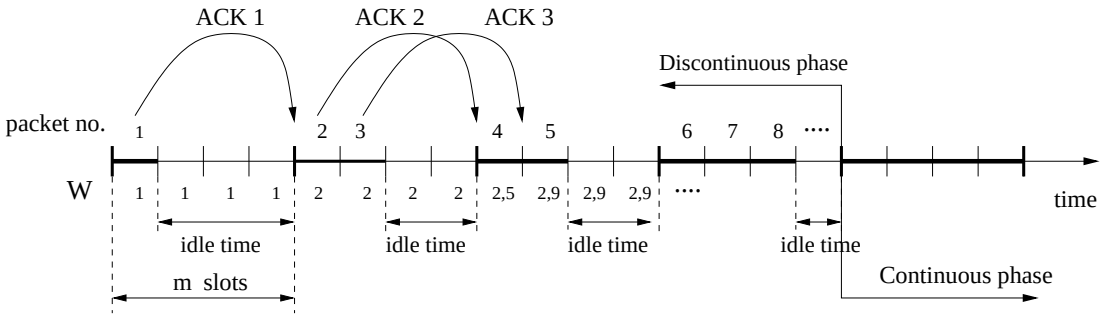


Figure 4.1: Discontinuous transmissions during TCP Slow-Start phase.

transmissions and underlying channel state. Moreover, the assumption made for analytical tractability by previous works (see [106] and [5] as examples) that a timeout timer is associated with each TCP packet has been removed. Instead, a single timeout is considered for the whole connection (as in a real TCP data transfer).

In order to take into consideration the bursty nature of the channel, a two-state Discrete Time Markov Channel (DTMC) is considered, as in [106]. The time is slotted, the single slot duration is equal to the TCP packet transmission time and the channel process (DTMC) evolves slot by slot according to its transition probabilities. Moreover, the work presented in [106] is extended here considering a non instantaneous feedback, i.e., ACKs arrive exactly $m-1$ slots after the packet transmission, where m is the round trip time value. The introduction of this feedback delay leads to what is referred here as the "discontinuous transmission phase" (see Fig. 4.1). In fact, after the transmission of the first packet of the connection, the TCP sender enters an idle time waiting for the reception of an ACK message. When this ACK arrives, the TCP window size, W, is incremented by one (*Slow-Start phase*), allowing the receiver to transmit two consecutive packets. According to the protocol rules, W is incremented by one segment for each new received ACK until W reaches the *Slow-Start threshold*, W_{th}. When W becomes larger than W_{th} the sender enters the so called *Congestion-Avoidance* phase and W is incremented by $1/W$ for each received ACK[1].

During this phase, TCP transmits packets in bursts of length equal to W and enters an idle time waiting for the ACK reception. This discontinuous phase finishes when W reaches the round trip value, i.e., when W becomes equal to m. It is important to note that the assumption of a round trip time greater than zero implies that the TCP sender takes more time to fill the bandwidth-delay product of the link, since the window grows more slowly, and this may significantly affect the TCP performance.

The main findings of this study are that (1) an increasing round trip time may significantly affect the throughput performance of TCP, especially when an independent channel is considered, (2) New Reno performs better than Reno and Tahoe when the channel is uncorrelated, whereas the Tahoe recovery strategy is the most efficient when the channel correlation is high and (3) the maximum window size does not play a determinant role in increasing throughput performance in both correlated and independent channels. While some of these conclusions confirm what other authors have observed in simulation studies, our analytical approach sheds some new light on TCP's behavior.

[1] In Fig.4.1 W_{th} is set to 1 segment.

This Chapter is organized as follows: in Section 4.2 the channel and the TCP models are presented in detail, Section 4.3 reports the analysis whereas results are shown in Section 4.4. Finally, in Section 4.5 some conclusions are given.

4.2 System Model

A TCP session involves three phases that are the connection setup phase, the data transfer phase and the connection tear-down phase. Since the focus of this Chapter is mainly in the bulk throughput performance of TCP, in this study only the data transfer phase is considered because it is the one that dominates the overall performance. We consider a pair of nodes, the sender and the receiver, exchanging data over a dedicated link, characterized by a two-state Markov packet error process, finite propagation delay and perfect feedback. It is assumed that the transmitter always has an infinite supply of packets to send (*Heavy Traffic* assumption). As usually done in studies taking an analytical approach, the focus is on a single TCP connection between a wireless terminal and a terminal placed in the terrestrial network. In this scenario, TCP packets experiences a round trip time (m) that is given by the sum of the contributes due to the wireless and the terrestrial network. The wired link s considered to be error-free, while we model the errors over the wireless channel by means of the Markov model introduced above. A detailed description of such model will be given in Section 4.2.2.

4.2.1 TCP Algorithms Description

In this Section, we first describe the transmitter and receiver processes in TCP OldTahoe, Tahoe, Reno and NewReno. In the following we proceed with our description of the various TCP algorithms. The TCP receiver is common for all TCP versions, while the TCP transmitter depends on the version considered. *The receiver* accepts packets out of sequence, but will only deliver them in sequence to the user's application. Moreover, the receiver sends back one ACK for every packet correctly received (no-delayed ACK receiver)[2]. The ACKs are cumulative, that is, an ACK carrying the sequence number i acknowledges all data packets up to, and including, the packet with sequence number $i-1$. Each ACK will identify the next expected packet sequence number, which is the first among the packets necessary to complete the in-sequence delivery. Hence, when a packet is lost, the transmitter keeps receiving the so called duplicate ACKs, that is, the ACK with the sequence number of the first packet lost.

The TCP transmitter operates using a sliding window based strategy as follows. At any given time t, the TCP transmitter maintains the status variables $A(t)$, $W(t)$ and $W_{th}(t)$. The lower window edge, $A(t)$, represents all data numbered up to and including $A(t)-1$ that has been transmitted and acknowledged. This variable is nondecreasing, and for each received acknowledge (ACK) with sequence number $n > A(t)$, $A(t)$ increases up to n. The congestion window, $W(t)$, defines the maximum number of unacknowledged packets the transmitter is allowed to send starting from $A(t)$. At any time $W(t)$ is limited by its maximum value[3] W_{max}. The slow-start threshold triggers the increment of $W(t)$ to realize the flow

[2] In this work we consider a no-delayed ACK receiver, even if in real TCP implementations the delayed ACK scheme [97] is used.

[3] This value can be negotiated with the receiving entity during the connection, however, in this study we consider it as a constant value.

control imposed by the sender, i.e., to slow-down or speed-up the transmission rate of packets into the network. In particular, if $W(t) < W_{th}$, each ACK causes $W(t)$ to be incremented by one. This is called the *Slow Start* phase. On the other hand, if $W(t) \geq W_{th}$, $W(t)$ is incremented by $1/W(t)$ every time a new ACK is received. This phase is called *Congestion Avoidance*.

Each TCP connection has a timeout timer that is updated at each round trip delay. If this timeout timer expires, i.e., no more ACKs are received, then a retransmission occurs. To maintain the analytical tractability we assume this timer to be fixed at a constant value, T_o.[4] Calling t the time in which this happens, $W(t^+)$ and $W_{th}(t^+)$ are set to 1 and $W(t)/2$ respectively. In this way the connection is restarted in Slow Start phase. Note, that there exists another flow control parameter imposed by the receiver called receiver advertised window $awnd$; the protocol rules impose that at any time t the transmitter is allowed to inject into the network a number of unacknowledged packets equal to the minimum between $W(t)$ and $awnd$. By setting $awnd$, the receiver can slow-down the sender transmission rate when it is not able to read data at the same speed at which the sender is transmitting.

When a loss occurs, *TCP-OldTahoe* [84] uses only the above explained timeout recovery mechanism.

TCP-Tahoe [58] [97] instead, implements a further recovery technique called *fast retransmit*. When a packet is lost and when the transmitter gets the K-th duplicate ACK[5] at time t, it behaves as if a timeout had occurred and begins retransmission setting $W(t^+)$ and $W_{th}(t^+)$ as above.

In the case of *TCP-Reno* [97] [98] [14], the Fast Retransmit procedure is implemented as in Tahoe, but the subsequent recovery phase is different. In more detail, with the reception of the K-th dupACK at time t, $W_{th}(t^+) = W(t)/2$ and $W(t^+) = W_{th}(t^+) + K$. At this point, the Reno transmitter transmits only the first lost packet (with sequence number equal to $A(t)$, this is the Fast Retransmit) and, as it waits for the ACK it may get dupACK due to the outstanding packets. For each dupACK W is inflated by one, and a packet is transmitted if allowed by A and the new value of W. This is the Fast Recovery algorithm. Finally, in t_f, when the new ACK is received the loss recovery phase finished and $W(t_f)$ is set to $W_{th}(t_f) = W_{th}(t^+)$. The Reno's loss recovery does not work for multiple losses in a window of packets [38], for example, when three packets are dropped from a window of data, the sender is forced to wait for a timeout whenever the number of packets between the first and the second dropped packet is less than $2 + 3W/4$, where W is the congestion window just before the Fast Retransmit. Note that, the loss recovery strategy in Reno performs better than Tahoe when single packet losses occur in the loss window (the window in which we have the first loss).

In the *New-Reno* algorithm [44], when K duplicate ACKs are received, the Fast Retransmit is applied as in Reno, but the loss recovery procedure is different. The highest sequence number transmitted before the loss is recorded in a variable named *recover*. The Fast Recovery phase differs from Reno when an ACK acknowledging new data, arrives. In fact, this ACK could be the acknowledgment elicited by the first retransmission, or elicited by a later retransmission. If this ACK acknowledges all the packet up to, and including *recover*, then the Fast Recovery procedure is completed as for the Reno case. On the other hand, this ACK is labeled as *partial new ACK*. In this case, the first unacknowledged packet is retransmitted, W is deflated by the amount of new data acknowledged (also A is updated accordingly)

[4] We do not explicitly model the timeout back-off mechanism. The accuracy of this simplifying assumption has been verified by simulation. Also, note that it is possible to let the timeout depend on the round trip time, m, with no modification to the analysis we present here.

[5] K is a system parameter usually set to 3.

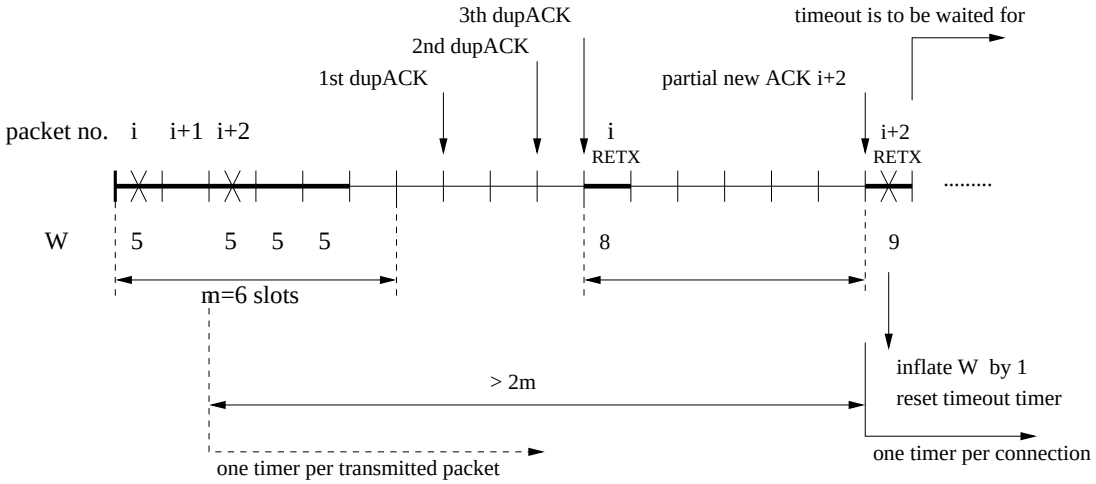


Figure 4.2: Scheduling of the timeout event in the NewReno fast recovery phase.

and then is incremented by one. So, a new packet is transmitted if allowed by the new value of A and W. The Fast Recovery is completed when the sender receives an ACK up to and including *recover*. The NewReno algorithm can recover from multiple losses in some cases.

Unlike the assumption in [106], where a timer is associated to every packet transmitted, in this work we consider only one timeout timer characterizing the connection, as in a real TCP protocol [97] [88]. This assumption allows us to correctly track the timeout whenever a partial new ACK is received, that is, when a partial new ACK is received the timeout timer is restarted (see [88]). In [106] due to the assumption of instantaneous feedback, the numerical results were not so different from the ones relative to the real TCP protocol. However, when a finite round trip is considered, performance is significantly affected, so it is important to correctly track the timeout event. Fig. 4.2 shows how the timer is rescheduled during the NewReno fast recovery phase when both a single timer and a timer per packet is used. Note that, when the recovery phase fails, and the second case is considered, the timeout timer expires at least $2m$ slots earlier.

4.2.2 Channel Model

We model the correlated packet error process using a discrete-time first-order Markov model as proposed in [117]. The pattern of errors is described by the transition matrix

$$\mathbf{M} = \begin{pmatrix} p_{BB} & p_{BG} \\ p_{GB} & p_{GG} \end{pmatrix} \tag{4.1}$$

where p_{BG} is the transition from bad to good, i.e., the conditional probability that successful transmission occurs in a slot given that a failure occurred in the previous slot, and the other entries in the matrix are defined similarly. Note that $1/(1 - p_{BB})$ represents the average length of a burst of errors, which is described by a geometric *r.v.*

Given the matrix $\mathbf{M}$, the channel properties are completely characterized. In particular, it is possible

f_dT	ε	F(dB)	p_{GG}	p_{BB}	$(1-p_{BB})^{-1}$
0.01	0.001	29.9978	0.999329	0.329452	1.49132
	0.01	19.9782	0.997518	0.754308	4.07013
	0.1	9.77322	0.991872	0.926851	13.6708
0.08	0.001	29.9978	0.999007	0.008239	1.00831
	0.01	19.9782	0.990680	0.077286	1.08376
	0.1	9.77322	0.940354	0.463188	1.86285
0.64	0.001	29.9978	0.999000	0.001185	1.00238
	0.01	19.9782	0.990019	0.011834	1.01198
	0.1	9.77322	0.90182	0.116376	1.13170

Table 4.1: Markov model parameters for different values of ε and f_dT.

to find the average slot error probability[6], $\varepsilon = p_{GB}/(p_{GB} + p_{BG})$, that turns out to be dependent on the *fading margin*, indicated with F, (that express the physical characterization of the channel), and the *normalized Doppler bandwidth product*, f_dT, where T is the packet duration [117]. By choosing different ε and f_dT values, we can establish fading channel models with different degrees of correlation in the fading process. When f_dT is small, the fading process is very correlated, on the other hand, for higher values of f_dT, successive samples of the channels are almost independent. In Table 4.1 the fading margin F, the transition probabilities p_{GG} and p_{BB} and the average burst length $(1-p_{BB})^{-1}$ are reported for several values of f_dT and ε.

A detailed analysis of the packet loss with memory is presented and the case of independent and identically *iid* errors is also considered for comparison[7].

4.3 Analytical Approach

The analysis is based on a Markov/renewal reward approach. The joint evolution of the window parameters and the channel state can be tracked by a random process $X(t) = (C(t-1), W_{th}(t), W(t))$, where $W(t)$ and $W_{th}(t)$ are the window size and the slow start threshold in slot t, respectively, and $C(t-1)$ is the channel state in slot $t-1$ (bad, B, or good, G, corresponding to an erroneous or correct transmission, respectively). Time is discrete, and the slot (packet transmission time) is the time unit. Unfortunately, this process is not Markov, since its evolution starting from a certain state, also depends on other quantities not accounted for in $X(t)$ (such as the number of outstanding packets). Following the approach proposed in [106], we sample the process at appropriate instants t_k. In particular, by choosing as sampling instants the slots immediately following those in which either a timeout timer expires or a loss recovery phase is successfully completed, we obtain a process $X(k)$=$X(t_k)$ which is Markov. In fact, immediately after a timeout event, the window size shrinks to 1 and, from the point of view of the

[6]This is the steady-state probability that the channel is in bad state in a time slot, and is not the same as the error probability experienced by transmitted packets because TCP's window adaptation tries to avoid bad channel conditions. The analytical approach developed in the following takes this into account.

[7]This case is obtained from the correlated channel by imposing $p_{BB} = p_{GB} = \varepsilon$.

window adaptation algorithm, no outstanding packets are present. Therefore, at these instants, knowledge of $(C(t-1), W_{th}(t), W(t))$ is all there is to know to characterize the window/channel evolution in the future. Likewise, in the instant immediately after the slot in which the loss recovery phase was successfully completed, by definition, all outstanding packets have been acknowledged. Again, $(C(t-1), W_{th}(t), W(t))$ is all we need to know to characterize the future evolution of the window/channel. We observe that, from the protocol rules, the value of $W(t_k)$ can only be equal to 1 (timeout case) or to $W_{th}(t)$ (successful loss recovery). Finally, note that unlike in [106], the channel state at time $t_k - 1$ can be either erroneous or correct not only in the timeout case, but also for a successful loss recovery. In fact, even if the loss recovery phase must be ended by a successful transmission, we receive the relative ACK after a round trip time. Therefore, the state space of the process $X(k)$ is given by

$$\begin{aligned} \Omega_X &= \left\{(C, W_{th}, 1), C = B, G, 1 \leq W_{th} \leq \left\lceil \tfrac{W_{max}}{2} \right\rceil\right\} \\ &\cup \left\{(C, W_{th}, W_{th}), C = B, G, 1 \leq W_{th} \leq \left\lceil \tfrac{W_{max}}{2} \right\rceil\right\} \end{aligned} \tag{4.2}$$

where the first set correspond to timeout and second set corresponds to successful recovery phase. Note that, the total number of states[8] is in this case $4\lceil \frac{W_{max}}{2} \rceil - 2$.

In order to evaluate metrics of interest, such as throughput, the above information is not sufficient, so we consider a semi-Markov process, which admits $X(k)$ as its embedded Markov chain. We label transitions of the chain $X(k)$ with transition metrics, which track the events which determine time delay, transmissions, and successes. For a given transition, let N_d be the associated number of slots, N_t the number of transmissions, and N_s the number of successful transmissions.

4.3.1 Semi–Markov Analysis

Let us define as *cycle k* the time evolution of the system between two consecutive sampling instant t_k and t_{k+1}, where t_k is the k-th sampling instant determined as explained in the previous section. The statistical behavior of a cycle only depends on the channel state at time t_{k-1} and on the slow start threshold and window size at time t_k, so the system state is given by: $X(k) = (C(t_{k-1}), W_{th}(t_k), W(t_k)) \in \Omega_X$, where Ω_X is the set of all possible values of $X(k)$ (state space of the sampled process). For simplicity of notation in the following we indicate $C(t_{k-1})$ as C. Also, we assume that the process is stationary, so that all statistics are independent of k. Let $n \geq 1$ be the first slot of the cycle to contain an erroneous transmission. Because of the assumption of a not instantaneous ACK message, the TCP packet transmission in each cycle can be discontinuous, so, to compute the distribution probability of n, we must take in account the only slots in which transmissions actually occur. Conditioned on the channel state in slot $t_k = 0$, the probability to have the first transmission error in slot $t = n$ is 0 when the sender is idle, whereas, when a transmission is allowed, it is given by

[8] States $(C, 1, 1)$, $C = G, B$ are contained in both sets.

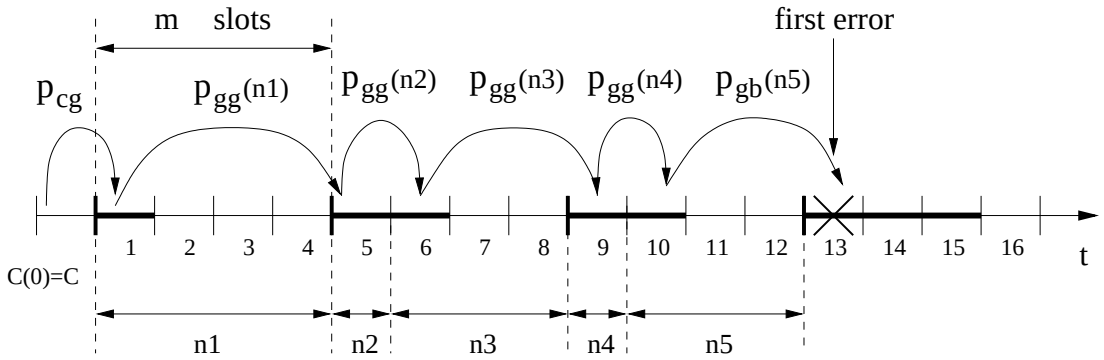


Figure 4.3: Distribution probability of the first error at slot n.

$$\begin{aligned} \alpha_C(n) &= \text{P [first error at } t = n | X(k) = (C, W_{th}, W)] \\ &= \begin{cases} p_{CB} & n = 1 \\ p_{CG} p_{GG}(n_1) p_{GG}(n_2) .... p_{GB}(n_l) & n > 1 \end{cases} \end{aligned} \tag{4.3}$$

where $\sum_{i=1}^{l} n_i = n - 1$ and $p_{GG}(n_i)$, (or $p_{BG}(n_i)$), represents the probability that the transmission in slot k is successful given that the transmission in slot $k - n_i$ was successful (unsuccessful). Note that the probability to have the first error at time n given the channel state at time 0 ($\alpha_C(n)$) is a deterministic quantity once C, W_{th} and W have been fixed. In fact, W_{th} and W are all we need to know to determine the error-free protocol evolution (transmission and window processes). Hence these processes can be easily tabulated. In order to explain the meaning of Eq. (4.3) consider the situation depicted in Fig. 4.3, in which we have assumed a round trip time of $m = 4$ slots, $W_{th} = 1$ and $W = 1$. In this case $\alpha_C(n) = p_{CG} p_{GG}(n_1) p_{GG}(n_2) p_{GG}(n_3) p_{GG}(n_4) p_{GB}(n_5)$, $n = 13$, where $p_{GG}(n_1)$ is the n_1-step probability ($n_1 = 4$), that is the probability that slot 5 is successful given that slot $5 - n_1$ was successful. The other entries in the expression are obtained similarly. For a given C, W_{th} and W, the transmission evolves following the protocol rules up to and including time n. Therefore we can easily tabulate the transmission mask that represents the sequence of slots in which a transmission actually occurs. Moreover, $\alpha_C(n)$ is directly derived from this mask as previously described.

Let $Y(k)$ be the system state at time n in cycle k and let Ω_Y be the set of all possible values of $Y(k)$. Note that, at time n there are a number of outstanding packets (packets *in flight*) that depends on $W(n)$, $W_{th}(n)$ and n. Also, by definition we know that the channel state at that time is B. To determine the state at the beginning of cycle $k + 1$ the slow start threshold plays an important role. In fact, the exact number of packets that the sender is allowed to transmit after the first error, (n_{tx}), depends on the number of packets in flight at time n. In particular, during the recovery phase, the TCP window size is further incremented by the number of incoming ACKs elicited by these packets. Therefore, defining W_{fin} as the final value reached by W and n_{acks} as the number of incoming ACKs, we can simply derive n_{tx} as $\lceil W_{fin} \rceil - 1$, since the error at time n always consumes one window unit. Referring to slot n, n_{acks} is

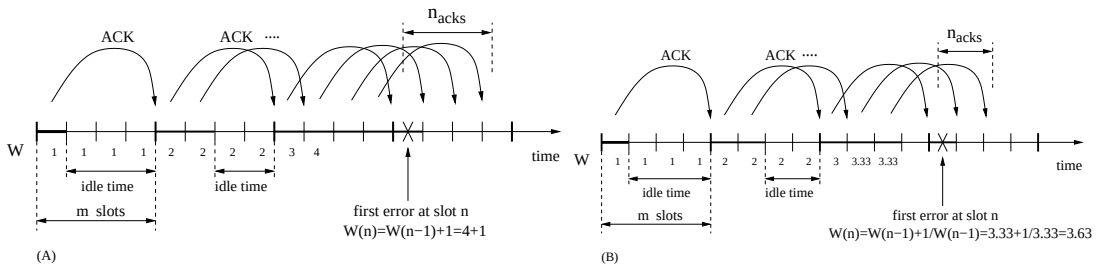


Figure 4.4: Number of ACKs received after the first error: (A) Slow-Start case, $W(1) = 1$ and $W_{th}(1) = 5$. (B) Congestion-Avoidance case, $W(1) = 1$ and $W_{th}(1) = 3$.

obtained as follows

$$n_{acks} = \begin{cases} \min\{\lceil W(n) \rceil - 2, m - 1\} & W(n) \leq W_{th}(n) \\ \min\{\lceil W(n) \rceil - 1, m - 1\} & W(n) > W_{th}(n) \end{cases} \tag{4.4}$$

where $W(n)$ represents the maximum number of outstanding (unacknowledged transmitted) packets at time n. Since the value of the round trip time is m, and the transmission before the error event is assumed to be error-free, $\min(\lceil W(n-1) \rceil, m)$ gives the number of packets transmitted in the round preceding slot n. In Eq. (4.4) we take in the minimum $m-1$ instead of m because we are evaluating the number of ACKs received after the error. In more detail, m is the maximum number of packets sent in the round preceding slot n and so it represents the maximum number of incoming ACKs that can be received from slot n onward; with $m-1$ we account for the ACK received in slot n. The two terms $\lceil W(n) \rceil - 2$ and $\lceil W(n) \rceil - 1$ in Eq. (4.4) are an estimate of $\lceil W(n-1) \rceil - 1$, and are chosen according to the value of $W(n)$ and $W_{th}(n)$. If $W(n) \leq W_{th}$, we are in the situation depicted in Fig. 4.4, case A, where the window size in the previous round is equal to $W(n)-1$ and the ACK relative to the first of the $\lceil W(n) \rceil - 1$ packets is the one that determines the increment of W at time n. In this case the number of ACKs received after the error event is equal to $W(n) - 2$. The situation in which $W(n) > W_{th}(n)$ is analogous (see Fig. 4.4 case B), but here at slot n, W is incremented by $1/W(n-1)$, so $\lceil W(n) \rceil$ usually remains equal to $\lceil W(n-1) \rceil$. Note that this last assumption is not always true, since some values of $W(n-1)$ could exist such that the above condition is not verified, i.e., $\lceil W(n-1) + 1/W(n-1) \rceil \neq \lceil W(n-1) \rceil$. In this case, to exactly derive $W(n-1)$ from the knowledge of $W(n)$, the state space Ω_X would have to be further expanded, but this would lead to such a big state space to make the problem analytically intractable. For this reason, when $W(n) > W_{th}(n)$, we approximate n_{acks} as $\lceil W(n) \rceil - 1$. However, we have verified by simulation that our assumption has a negligible effect on the performance results for a wide range of the system parameters, and is therefore entirely acceptable in general. Once computed n_{ack}, W_{fin} is derived in a straightforward way from the knowledge of $W(n)$ and $W_{th}(n)$ by using the TCP protocol window incrementing rules. Specifically, W_{fin} is computed by using the following formula

$$W_{fin} = r_{n_{acks}} \tag{4.5}$$

where

$$\begin{aligned} r_0 &= W(n) \\ r_i &= r_{i-1} + f(r_{i-1}) \quad i \geq 1 \end{aligned} \tag{4.6}$$

$$f(x) = \begin{cases} 1 & x < W_{th} \\ 1/x & x \geq W_{th} \end{cases} \tag{4.7}$$

We can observe that, in our model, $Y(k)$ is described by $(W_{th}(n), W(n))$, where $1 \leq W_{th}(n) \leq \lceil W_{max}/2 \rceil$ and $1 \leq W(n) \leq W_{max}$.

As described in [106], we can separate the system evolution in two parts. The first part, in which the system makes a transition from state $X(k) \in \Omega_X$ to a state $Y(k) \in \Omega_Y$, is characterized by the transition matrix $\mathbf{\Phi}^{(1)}(z)$; whereas the second part, in which the system makes a transition from state $Y(k) \in \Omega_Y$ to a state $X(k) \in \Omega_X$, is characterized by the transition matrix $\mathbf{\Phi}^{(2)}(z)$. The statistic of the cycle is fully described by the matrix

$$\mathbf{\Phi}(z) = \mathbf{\Phi}^{(1)}(z)\mathbf{\Phi}^{(2)}(z) \tag{4.8}$$

whose entries are the transition functions associated to transitions from Ω_X to itself. In practice the function $\mathbf{\Phi}^{(1)}(z)$ is responsible of tracking the error-free protocol evolution until the instant in which the first error event occurs, whereas $\mathbf{\Phi}^{(2)}(z)$ is used to track the system evolution from the error to the instant where it is resolved (either by means of timeout or retransmissions, i.e., Fast Recovery and Fast Retransmit algorithms) and a new good channel period is entered. The variable z is a vector of transform variables, $z = (z_d, z_t, z_s)$, where z_d tracks the delay, z_t the number of transmissions and z_s the number of successes. More precisely, we can define

$$\Phi_{ij}(z_d, z_t, z_s) = \sum_{N_d, N_t, N_s} \xi(N_d, N_t, N_s) z_d^{N_d} z_t^{N_t} z_s^{N_s} \tag{4.9}$$

where $\xi(N_d, N_t, N_s)$ is the probability that the system makes a transition to state j in exactly N_d slots, and that in slots $\{1, 2, ..., N_d\}$ N_t transmission attempts are performed and N_s successes are counted, given that the system was in slot i at time 0.

In particular we can note that the transition matrix of the embedded Markov chain is given by $\mathbf{P} = \mathbf{\Phi}(1, 1, 1)$ whereas, the matrix of the average delays can be found as

$$\begin{aligned} \boldsymbol{D} &= \left. \frac{\partial \mathbf{\Phi}(z_d, z_t, z_s)}{\partial z_d} \right|_{z_d, z_t, z_s = 1} \\ &= \boldsymbol{D}_1 \mathbf{\Phi}^{(2)}(1, 1, 1) + \mathbf{\Phi}^{(1)}(1, 1, 1) \boldsymbol{D}_2 \end{aligned} \tag{4.10}$$

where

$$\begin{aligned} \boldsymbol{D}_1 &= \left. \frac{\partial \mathbf{\Phi}^{(1)}(z_d, z_t, z_s)}{\partial z_d} \right|_{z_d, z_t, z_s = 1} \\ \boldsymbol{D}_2 &= \left. \frac{\partial \mathbf{\Phi}^{(2)}(z_d, z_t, z_s)}{\partial z_d} \right|_{z_d, z_t, z_s = 1} \end{aligned} \tag{4.11}$$

The average number of transmissions, $\boldsymbol{T}$, and successes, $\boldsymbol{S}$, can be found similarly. According to the renewal reward theory described in [52] [93] [108], we can evaluate the average TCP throughput as

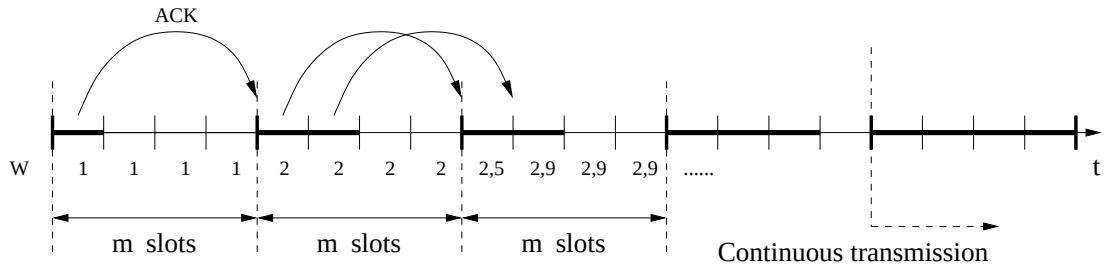


Figure 4.5: Evolution of the window size.

$$\text{throughput} = \frac{\sum_{i \in \Omega_X} \pi_i \sum_{j \in \Omega_X} P_{ij} S_{ij}}{\sum_{i \in \Omega_X} \pi_i \sum_{j \in \Omega_X} P_{ij} D_{ij}} \tag{4.12}$$

where π_i, $i \in \Omega_X$, are the steady-state probabilities of the embedded Markov chain (transition matrix $\boldsymbol{P}$). The average number of transmissions per slot can be similarly computed, by using T_{ij} instead of S_{ij}. This last metric is related to the energy consumption of the protocol [116].

4.3.2 Computation of $\mathbf{\Phi}^{(1)}(z)$

Since the first part of a cycle consists of error-free transmissions, and since all versions of TCP considered here have the same window adaptation mechanism as long as there are no errors, the computation of $\mathbf{\Phi}^{(1)}$ applies to all versions of TCP.

Let $X = X(k) = (C, W_{th}, W)$ the starting state of cycle k. The first part of the cycle has a duration of $N_d = n$ slots with probability $\alpha_C(n)$, $n \geq 1$. The window size at time n is a deterministic function $w(n, m, W_{th}, W)$ of m, W_{th} and W. So we can denote the window size at time n by

$$W(n) = w(n, m, W_{th}, W) \tag{4.13}$$

Therefore, this deterministic function can be easily tabulated. It can be observed that the evolution of the window strongly depends on the round trip time value. Fig. 4.5 shows an example of window evolution, where W_{th} in the first slot is equal to 1. Note that W remains equal to one segment for the first m slots, since the first ACK message arrives exactly a round trip time later. The successive increments depend on the value of W_{th} and are always triggered by incoming ACKs. In more detail, the window size at time n increments if and only if a new ACK arrives, that is, if m slots earlier a successful transmission occurred. Similar considerations can be made for any initial values of W and W_{th}, so that in general the error-free window evolution is computed applying recursively the following expressions and taking $\min(W_n, W_{max})$

$$W_n = \begin{cases} W & n = 1, ..., m \\ W_{n-1} + f(W_{n-1}) & W_{n-m} \geq (n - \lfloor \frac{n}{m} \rfloor m), \\ & n = km + u;\ k \in \mathbb{N}, \\ & u = 1, 2, ..., m-1 \\ W_{n-1} + f(W_{n-1}) & W_{n-m} \geq m, \\ & n = km;\ k \in \mathbb{N} \\ W_{n-1} & \text{elsewhere} \end{cases} \tag{4.14}$$

where $f(\cdot)$ is defined in Eq. (4.7). One can note that in the discontinuous phase, the window size increments more slowly than in the continuous transmission mode. So, for large m, W takes more time to reach its maximum value. This strongly limits the throughput performance since the link is not fully exploited. The transmission mask, $TX(i)$, $1 \leq i \leq n$ is derived in a straightforward manner from the knowledge of $W(n)$

$$TX(i) = \begin{cases} 1 & \Delta(W_i, m) \geq 0 \\ 0 & \Delta(W_i, m) < 0 \end{cases} \tag{4.15}$$

where

$$\Delta(W_i, m) = \begin{cases} W_i - m & i = km;\ k \in \mathbb{N} \\ W_i - (i - \lfloor \frac{i}{m} \rfloor m) & \text{elsewhere} \end{cases} \tag{4.16}$$

With the function $\Delta(W_i, m)$ we are checking whenever the window in position i suffices to ensure a transmission in that slot. In more detail, by subdividing (starting from the first slot of the cycle) the time in rounds of length m, $s_{cr} = i - \lfloor i/m \rfloor m$ gives the number of slots covered by i in the current round. So, remembering that the protocol, at any time, allows a maximum of W_i unacknowledged packets, a packet in position i can be transmitted only if $W_i - s_{cr} \geq 0$. Similarly, when i is an integer multiple of m, we have a transmission only if the window size in that slot is greater or equal to m. Moreover, at time n the delay N_d is equal to n slots, but the number of packet transmissions, N_t, is no longer equal to n like in [106], but it is equal to $\eta = \eta(n, m, W_{th}, W)$, that is the number of packets actually transmitted. This value is computed using the transmission mask as follows

$$\eta(n, m, W_{th}, W) = \sum_{i=1}^{n} TX(i) \tag{4.17}$$

The number of successes is $N_s = \eta - 1$. Therefore, once the starting state X and the final state $Y = (W_{th}(n), W(n))$, have been selected we can write

$$\Phi_{XY}^{(1)}(z_d, z_t, z_s) = \sum_{n \in \mathcal{C}(X,Y)} \alpha_C(n) z_d^n z_t^\eta z_s^{\eta-1} \tag{4.18}$$

where $\mathcal{C}(X, Y) = \{n : (W_{th}(n), w(n, m, W_{th}, W)) = Y\}$.

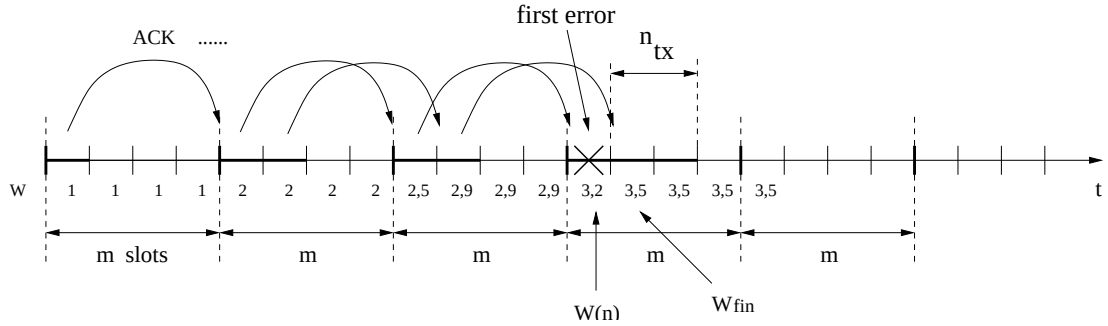


Figure 4.6: Assumption of the first error in the first slot on a round trip.

4.3.3 Computation of $\mathbf{\Phi}^{(2)}(z)$

The second part of the cycle is characterized by a transition function, $\mathbf{\Phi}^{(2)}(z)$, that depends on the way the different TCP versions handle packet loss recovery, and therefore it must be computed separately in the various cases.

Define the time in which the first error occurs as time 0, so that the first slot in the second part of the cycle corresponds to time 1. Consider, also, $\varphi_{ij}(k,n)$ as the probability that there are k successes in slots 1 through n and that the channel is in state j at time n, given that the channel was in state i at time 0. Let $Y = (W_{th}, W)$ be the starting state for the second half of the cycle. Then, referring to Eq. (4.5), the number of packets that the transmitter is allowed to send after the error is $n_{tx} = \lceil W_{fin} \rceil - 1$.

4.3.4 Computation of $\mathbf{\Phi}^{(2)}(z)$ for OldTahoe

In the case of TCP OldTahoe, there are neither Fast Retransmit nor Fast Recovery. Note that the second part of the cycle, initiated by the loss at time 0, has duration which is deterministically equal to T_o, since every loss can only be recovered by timeout. The next cycle, then, will start in state $X(k+1) = (C, \lceil W_{fin}/2 \rceil, 1)$ with transition function

$$p_{BC}(T_0 - 1) z_d^{T_0 - 1} z_t^{n_{tx}} z_s^{N_s} \tag{4.19}$$

where N_s is a random variable. Since the exact analysis is tedious to compute, we use here a simple approach based on a bounding technique. In such cases, instead of considering the average of the reward function for each transition, $E[N_s]$, we take $0 \leq N_s \leq n_{tx}$, where $N_s = 0$ correspond to the case in which none of the transmissions after the first error is successful, while the upper bound counts all these transmissions as correct. This approach gives two analytical bounds for the throughput performance.

Moreover, here, a simple approximation is used, that is, we consider that the first error in the cycle always occurs in the first slot of a round trip time, as shown in Fig. 4.6. This is equivalent to assuming a continuous transmission after the first error and only in this case the analysis above is correct. In correlated channels, since the error burst exactly starts from the loss at time 0, the assumption to have a continuous transmission after the first error leads to a pessimistic case, i.e., where the burst is more likely affecting all the packets transmitted after time 0. Another approximation is the one used to account for the timeout value. In particular, we assume that a single timeout timer is used for the TCP connection and

that it is re-initialized at the beginning of time 0, i.e., exactly when the first error occur. This justifies the duration of the cycle (T_0, see Eq. (4.19)). In real TCP implementations, the instants in which the timeout timer is restarted depend on the round trip time value, m, on the transmission mask and on where errors occur. For these reasons an exact tracking of this process, though in principle possible, turns out to be very tedious and the assumption above is used instead. We proved by simulations that this assumption has a negligible effect when T_0 is large compared to m, that is also the case of interest. The same assumptions will be implicitly considered in the sequel.

4.3.5 Computation of $\Phi^{(2)}(z)$ for Tahoe

In the case of Tahoe, there is no Fast Recovery. Once Fast Retransmit is triggered, regular transmission is resumed, starting from the first unacknowledged packet and by setting the TCP window to 1. On the other hand, timeout is expected for.

1. $\mathbf{n_{tx} < K}$:

   If $n_{tx} < K$, fast retransmit can not be triggered since the number of packets that can be further transmitted is less than K. So, K duplicate ACKs will never be received. In this case, the timeout timer will expire and the lost packet will be retransmitted in slot $t_{k+1} = T_o$. Note that the value of the window size at the timeout instant will be greater than W, since new ACKs can arrive from the previous round trip time. Therefore, the window size advances till W_{fin}. Hence, after the timeout, the algorithm will set $W_{th} = \lceil W_{fin}/2 \rceil$, $W = 1$. Then, the transition function to state $X(k+1) = (C, \lceil W_{fin}/2 \rceil, 1)$ is

   $$p_{BC}(T_0 - 1) z_d^{T_0 - 1} z_t^{n_{tx}} z_s^{N_s} \tag{4.20}$$

   N_s is bounded as in the OldTahoe case, in particular, $N_s = 0$ gives a lower bound for the throughput performance, while the upper bound is obtained by setting $N_s = n_{tx}$.

2. $\mathbf{n_{tx} \geq K}$:

   Case 2.1 - Fast Retransmit is Not triggered: If fewer than K slots in n_{tx} are successful, fast retransmit will not be triggered. At this point, we rely on the approximation above that each error occurs in the first slot of the round. Only where this condition is verified, in fact, are the n_{tx} packets following the erroneous one at time 0 transmitted continuously, and the following analysis is exact. The same consideration applies to the analysis presented in the rest of the Chapter. The destination values of W_{th} and W will be as in the previous case. Let $\mathcal{A}(C', j)$ be the event that there are j successful slots in $\{1, 2, ..., n_{tx}\}$ ad that the channel in slot n_{tx} is C'. Then, $P[\mathcal{A}(G, j)] = \varphi_{BG}(j, n_{tx})$ and $P[\mathcal{A}(B, j)] = \varphi_{BB}(j, n_{tx})$. The transition function from state Y to state $X(k +$

$1) = (C, \lceil W_{fin}/2 \rceil, 1)$ is then given by

$$p_{BC}(T_o - n_{tx} - 1) z_d^{T_o - 1} z_t^{n_{tx}} \sum_{j=0}^{K-1} P[\mathcal{A}(B,j)] z_s^{N_s(B,j)}$$
$$+ p_{GC}(T_o - n_{tx} - 1) z_d^{T_o - 1} z_t^{n_{tx}} \sum_{j=0}^{K-1} P[\mathcal{A}(G,j)] z_s^{N_s(G,j)} \tag{4.21}$$

where $0 \leq N_s(B,j), N_s(G,j) \leq j$ and the two terms account for the two possibilities for the channel state at time n_{tx}. Note that the sums are limited to $K-1$ instead of n_{tx} since the number of successes must be less than K for the considered case of fast retransmit not triggered.

Case 2.2 - Fast retransmit is triggered: If the K-th duplicate ACK is received, fast retransmit is triggered m slots after the K-th success in $\{1, 2, ..., n_{tx}\}$. Define as $K \leq i \leq n_{tx}$ the slot in which the K-th successful transmission occurs. Then the next cycle starts in slot $i+m$, the destination state is $X(k+1) = (C, \lceil W_{fin}/2 \rceil, 1)$, and the transition function is given by

$$\varphi_{BG}(K,i) \sum_{C' \in \{G,B\}} \sum_{j=0}^{n(i)} \left\{ P_{C'}[j|n(i)] p_{C'C}(m - n(i) - 1)\, z_d^{i+m-1} z_t^{n_{txeff}(i)} z_s^{N_s} \right\} \tag{4.22}$$

where $n(i) = n_{txeff}(i) - i$ and $n_{txeff}(i)$ is the number of packets transmitted between the error at time 0 and the beginning of the recovery phase (when the K-th duplicate ACK is received)

$$n_{txeff}(i) = \begin{cases} n_{tx} & n_{tx} < i + m - 1 \\ i + m - 1 & n_{tx} \geq i + m - 1 \end{cases} \tag{4.23}$$

C' is the channel state in the slot where the $n_{txeff}(i)$-th packet is transmitted, whereas C is the channel state in the slot just before the one in which the next cycle starts. The function $P_{C'}[j|n(i)]$ represents the probability to have j correct packets in $\{i+1, ..., n_{txeff}(i)\}$

$$P_{C'}[j = 0|n(i) = 0] = \begin{cases} 1 & C' = G \\ 0 & C' = B \end{cases} \tag{4.24}$$

$$P_{C'}[j|n(i) > 0] = \varphi_{GC'}(j, n(i)) \tag{4.25}$$

The quantity N_s can be bounded by $0 \leq N_s \leq K + j$, where 0 and $K + j$ represent the lower-bound and the upper-bound case, respectively. The lower bound accounts here for the very worst case where all the $K + j$ packets correctly transmitted in slot 1 through $n_{txeff}(i)$ are retransmitted during following cycles.

4.3.6 Computation of $\Phi^{(2)}(z)$ for Reno

Since TCP Reno differs from Tahoe only after fast retransmit is triggered, all the events considered for TCP Tahoe and corresponding to no fast retransmit still apply in this case. Hence, in the following we only need to consider the case in which fast retransmit is triggered (*case 2.2*), i.e., the case in which

the K-th successful transmission occurs at time i in $\{1, 2, ..., n_{tx}\}$. Let $\mathcal{B}(K)$ be the event that the packet failure at time 0 is followed by K consecutive successes, and let $\mathcal{B}(i, l_1)$, $K < i \leq n_{tx}$, $0 < l_1 \leq K$ be the event that the K-th success occurs at time i and the first loss after the loss in 0 occurs at time l_1 (note that since $i > K$, there must be a packet loss before the K-th success). The probabilities of these events are given as follows

$$P[\mathcal{B}(K)] = p_{BG} p_{GG}^{K-1} \tag{4.26}$$

$$P[\mathcal{B}(i, l_1)] = \begin{cases} p_{BB}\varphi_{BG}(K, i-1), \\ \qquad l_1 = 1;\ i = K+1, ..., n_{tx} \\ p_{BG} p_{GG}^{l_1-2} p_{GB}\varphi_{BG}(K - l_1 + 1, i - l_1), \\ \qquad l_1 = 2, ..., K;\ i = K+1, ..., n_{tx} \end{cases} \tag{4.27}$$

Case 2.2.a: Consider first the occurrence of the event $\mathcal{B}(K)$. Since at the end of slot $K + m - 1$ the K-th duplicate ACK is received, the retransmission is performed in slot $K + m$.

- If this retransmission is successful, the loss recovery phase is successfully completed and a new cycle starts at time $K+2m$. In this case, the destination state is $X(k+1) = (C, \lceil W_{fin}/2 \rceil, \lceil W_{fin}/2 \rceil)$[9] and the transition function is given by

$$P[\mathcal{B}(K)] p_{GG}(m) p_{GC}(m-1) z_d^{K+2m-1} z_t^{n_{txeff}(K)+1} z_s^{N_s+1} \tag{4.28}$$

  where we account for the probability to have the error at time 0 followed by K correct packets ($P[\mathcal{B}(K)]$) and a correct retransmission (term $p_{GG}(m)$, m slots after the transmission of the K-th packet); the term $p_{GC}(m-1)$ accounts for the channel state in the slot preceding the start of the new cycle. N_s can be bounded as $K \leq N_s \leq n_{txeff}(K)$, giving the throughput lower and upper bound, respectively.

- If, on the other hand, the retransmission is a failure, the protocol will stop and wait for an ACK which will never be transmitted. In this case only the timeout will eventually resolve the deadlock. According to the TCP Reno rules, upon receiving the K-th duplicate ACK the window size will be updated to $W' = \min\{\lceil W_{fin}/2 \rceil + K, W_{max}\}$, so that the new state after timeout is $X(k+1) = (C, \lceil W'/2 \rceil, 1)$ with transition function

$$P[\mathcal{B}(K)] p_{GB}(m) p_{BC}(T_o - K - m - 1) z_d^{T_o-1} z_t^{n_{txeff}(K)+1} z_s^{N_s} \tag{4.29}$$

  N_s is bounded as in the previous case and N_t accounts for the retransmission of the lost packet in addition to the $n_{txeff}(K)$ transmissions.

Case 2.2.b: Consider the occurrence of the event $\mathcal{B}(i, l_1)$. In the case of multiple losses in a congestion window, TCP Reno is able to successfully complete the Fast Recovery phase only when the congestion window at the first error is large enough. So, in this article we consider that multiple losses event always leads to deadlock and consequent timeout[10].

[9] According with Reno's rules, after the reception of the K-th duplicate ACK, W_{th} is set to half the window size W_{fin} and never changed until the successful completion of the lost recovery phase, where the window size is set to $W_{th} = \lceil W_{fin}/2 \rceil$.

[10] This assumption is better verifi ed as the round trip increases.

- If the retransmission at time $i+m$ is a failure, we can observe that the protocol behavior is similar to the previous case, i.e., the next cycle will start in state $X(k+1) = (C, \lceil W'/2 \rceil, 1)$, with transition function

$$P[\mathcal{B}(i,l_1)]p_{GB}(m)p_{BC}(T_o - i - m - 1)z_d^{T_o-1}z_t^{n_{txeff}(i)+1}z_s^{N_s} \tag{4.30}$$

  where the term $P[\mathcal{B}(i,l_1)]p_{GB}(m)$ represents the probability that the K-th correct packet (after the loss at time 0) is transmitted in slot i and that the retransmission of the first lost packet is unsuccessful. In this case, a time out event must be waited for; this will happen exactly T_o slots after the beginning of slot 0. Here, N_s can be bounded as $l_1 - 1 \leq N_s \leq n_{txeff}(i)$.

- On the other hand, if the retransmission at time $i+m$ is successful the system will timeout at the end of slot $T_o + i + 2m - 1$. In fact, according to the protocol rules, we consider only one timeout timer associated to the connection and this timer is reset (see [88]) at the reception of the ACK relative to the loss at time 0 (at the end of slot $i + 2m - 1$). The window size will be updated to $W'' = \min\{\lceil W_{fin}/2 \rceil + K + 1, W_{max}\}$ (the ACK for the successful retransmission causes the window to be further increased by one with respect to the previous case). The next cycle will restart in state $X(k+1) = (C, \lceil W''/2 \rceil, 1)$ with transition function

$$P[\mathcal{B}(i,l_1)]p_{GG}(m)p_{GC}(T_o + m - 1)z_d^{T_o+i+2m-1}z_t^{n_{txeff}(i)+1}z_s^{N_s} \tag{4.31}$$

where N_s can be bounded as $l_1 \leq N_s \leq n_{txeff}(i) + 1$. In fact, in this case, the number of successes is at least one more than in the previous case.

4.3.7 Computation of $\Phi^{(2)}(z)$ for NewReno

Finally, let us consider TCP NewReno. The only case in which it is different from Reno is when there are multiple losses within the same congestion window and the retransmission performed due to Fast Retransmit is successful (second bullet of case 2.2.b). All other cases are the same as in Reno. Refer to the definition of $\mathcal{B}(i,l_1)$, $K < i \leq n_{tx}$, $0 < l_1 \leq K$ as the event in which the K-th success occurs at time i and the first loss after the loss in 0 occurs at time l_1. If the K-th successful transmission is performed at time $i > K$, then besides the first lost packet (at time 0) there are $i - K$ losses in slot l_1 through $i - 1$.

Successful Loss Recovery

Let n_{error} be the total number of packets in error in slots $\{0, 1, ..., n_{txeff}(i)\}$. Conditioned on i, this variable is given by the sum of a deterministic and a random term. The former is the number of errors up to slot i (that is $i - K + 1$), while the latter, that we call n_{loss}, $0 \leq n_{loss} \leq (n_{txeff}(i) - i)$, represents the number of errors in $\mathcal{Q} = \{i + 1, ..., n_{txeff}(i)\}$. Let also $P_{C'}[n_{loss}]$ be the probability to have exactly n_{loss} losses in $\mathcal{Q}$ and to have channel state C' in slot $n_{txeff}(i)$. This probability is computed as

$$P_{C'}[n_{loss} = 0 | n(i) = 0] = \begin{cases} 1 & C' = G \\ 0 & C' = B \end{cases} \tag{4.32}$$

$$P_{C'}[n_{loss} | n(i) > 0] = \varphi_{GC'}(n(i) - n_{loss}, n(i)) \tag{4.33}$$

where $n(i) = n_{txeff}(i) - i$. If we have n_{error} consecutive successful retransmissions, loss recovery is successfully completed, since at time $i + mn_{error}$ the last lost packet is successfully retransmitted and the corresponding ACK (received at the end of slot $i + mn_{error} + m - 1$) will acknowledge all outstanding packets. A new cycle will then start at time $i + (n_{error} + 1)m$ in state $X(k+1) = (C, \lceil W_{fin}/2 \rceil, \lceil W_{fin}/2 \rceil)$, since W_{th} remains equal to $\lceil W_{fin}/2 \rceil$ and W is set to W_{th} upon completion of the Loss Recovery Phase. The transition function corresponding to this event is

$$\begin{aligned} P[\mathcal{B}(i,l_1)] \sum_{C' \in \{G,B\}} \sum_{n_{loss}=0}^{n(i)} \Big\{ & P_{C'}[n_{loss}|n(i)] \\ & \times p_{C'G}(m - n(i))\, p_{GG}(m)^{n_{error}-1}\, p_{GC}(m-1) \\ & \times z_d^{i+(n_{error}+1)m-1}\, z_t^{n_{txeff}(i)+n_{error}} \Big\} z_s^{n_{txeff}(i)+1} \end{aligned} \tag{4.34}$$

where $n(i) = n_{txeff}(i) - i$ and $n_{error} = i - K + 1 + n_{loss}$. Note that, since the recovery phase ends successfully, all packets in slots $\{0, 1, ..., n_{txeff}(i)\}$ are eventually received correctly, so N_s is exactly given by $n_{txeff}(i) + 1$. Note that C is the channel state in the slot just before the one in which the new cycle begins, whereas C' represents the channel state in slot $n_{txeff}(i)$. In Eq. (4.34), we consider the case of multiple losses in a window. The total number of losses has been separated in two contributions. The first is relative to the number of lost packets between the loss at time 0 and the transmission of the K-th correct packet ($i - K + 1$ losses). The second contribution, instead, accounts for the number of losses between the transmission of the K-th correct packet and the slot in which the fast retransmit starts (n_{loss} losses). As observed above, the total number of losses is given by $n_{error} = i - K + 1 + n_{loss}$. In Eq. (4.34) we sum over all possible values of n_{error} and over all possible values of the channel state in the slot where the last of the $n_{txeff}(i)$ packets is transmitted. From here, we use the term $p_{C'G}(m - n(i))$ to evolve the channel state until the slot in which fast retransmit starts[11]. In z_d we account for the total delay, i.e., we sum to i the time needed to recover from the n_{error} losses after the K-th successful packet (transmitted is slot i), in z_t we count all the transmitted packets, whereas in z_s we count only successes ($n_{txeff}(i)$ packets sent before the beginning of the recovery phase plus the retransmission of the loss at time 0).

Unsuccessful Loss Recovery

Consider now the case in which the loss recovery phase does not end successfully, i.e., after the K-th duplicate ACK there are fewer than n_{error} successes. Suppose that, after the successful retransmission of the first loss at time 0, we have exactly j consecutive good retransmissions ($0 \le j \le n_{error} - 2$) followed by a failure. This event has probability $p_{GG}^j(m) p_{GB}(m)$. After the j-th success the value of the window is given by $W''' = \min\{\lceil W_{fin}/2 \rceil + K + 1 + j, W_{max}\}$. The initial state of the next cycle corresponds to the timeout relative to the $(j+2)$nd loss in the window, since the first $j+1$ have been successfully

[11] Since we are in the successful loss recovery case, we assume that the first and all following retransmissions are successful.

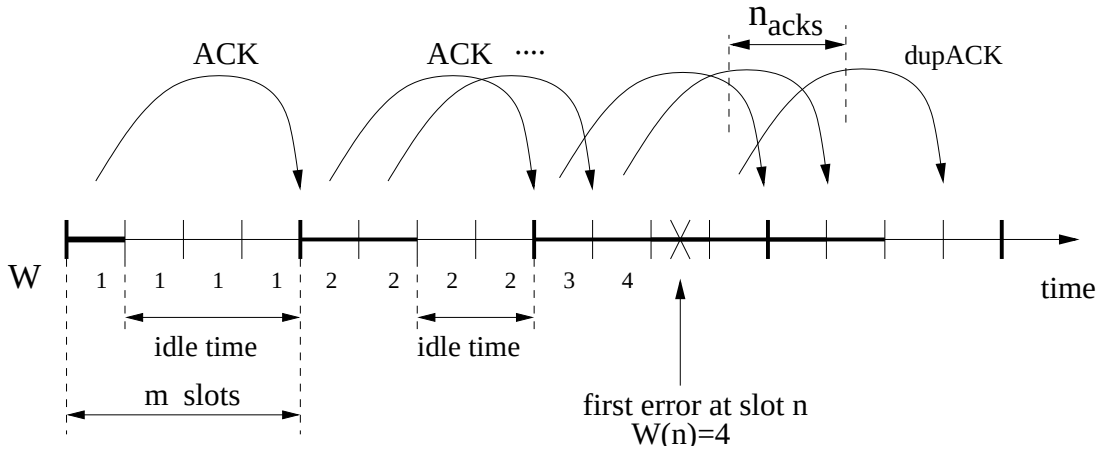


Figure 4.7: n_{acks}.

retransmitted, and is $X(k+1) = (C, \lceil W'''/2 \rceil, 1)$. The transition function in this case is given by

$$P[\mathcal{B}(i,l_1)] \sum_{C' \in \{G,B\}} \sum_{n_{loss}=0}^{n(i)} \Bigg\{ P_{C'}[n_{loss}|n(i)] p_{C'G}(m-n(i)) \times \sum_{j=0}^{n_{error}-2} \Big[p_{GG}(m)^j \, p_{GB}(m) \, p_{BC}(T_o-1) \times z_d^{i+(j+2)m+T_o-1} \, z_t^{n_{txeff}(i)+j+2} \, z_s^{j+1} \Big] \Bigg\} \tag{4.35}$$

where $n(i) = n_{txeff}(i) - i$ and $n_{error} = i - K + 1 + n_{loss}$. In the equation above, similarly to Eq. (4.34), we sum over all possible values of n_{error} and C'. In this case, however, not all packets in error are successfully recovered. We account for this fact by means of the sum over j. Here, a timeout event must be waited for, and it will happen exactly $T_o - 1$ slots after the first retransmission failure (term $p_{BC}(T_o - 1)$).

Finally, the transition function matrix $\mathbf{\Phi}^{(2)}$ is computed by assigning to each entry $\mathbf{\Phi}^{(2)}_{XY}$ the sum of all the transition functions leading from state X to state Y, as explained in [106].

4.4 Results

In this Section the throughput performance of the various versions of TCP are analyzed.

First of all we note that, in our study, we have considered some approximations in order to keep the model analytically feasible. In the following we point out these approximations in order to better clarify the analysis and to permit a better understanding of the results presented in this Section. First of all, Eq. (4.4), gives an overestimate of the number of ACKs received after the first error. This is true especially for high error probabilities and large values of m. Under these conditions, in fact, the number of packets in flight could be less than the window value at the time in which the error occurs (see Fig. 4.7). This happens when the transmission in the round preceding the one where the error occurs is discontinuous. In any case, the use of Eq. (4.4) always leads to an analytical upper bound for the window

size reached after the first error (W_{fin}). For this reason, what we model is an approximated version of the real protocol that corresponds to an analytical upper-bound for what concerns throughput performance of the real protocol. Hence, the throughput lower and upper bounds considered in the computation of the matrix $\mathbf{\Phi_2}(z)$ must be viewed as the upper and the lower bounds for the approximate model and not for the real protocol. In any event, as will be clear from the results presented, the approximation considered is very good.

In Fig. 4.8.A we report the throughput of TCP Tahoe as a function of the channel error probability P_e by fixing $f_dT = 0.01$ (correlated channel case). In the figure three kinds of curves are reported: the analytical throughput upper and lower bound and the values obtained by simulation. As the round trip time (m) increases analysis and simulation differ. In particular, the throughput obtained from the analysis is slightly higher than the one obtained by simulation. This is justified by the way in which the quantity n_{ack} has been computed.

In Fig. 4.8.B we compare simulation results and analysis for TCP Reno. Here, the approximation introduced in Eq. (4.4) is less influential than in the previous case. This is justified by the fact that, after a successful recovery phase both W and W_{th} are set to $\lceil W_{fin}/2 \rceil$. By doing so, the performance is less influenced by the value of W_{th} with respect to the Tahoe case (where $W = 1$ and $W_{th} = \lceil W_{fin}/2 \rceil$)[12]. Moreover, in this figure we note some differences between simulation and analysis when both m and P_e are low. These differences are due to our assumption that TCP Reno always times out in the case of multiple losses in a window. This assumption, in fact, is good only when either m or P_e has a large value, in any other case $W(n)$ could be sufficiently large to ensure the recovery of multiple losses. However, the approximation above appears to be very good as m increases.

Fig. 4.8.C shows the same comparison for TCP New Reno. Here, no simplified assumptions are made regarding the recovery process. Simulation results in this case match almost perfectly with the analysis.

In Fig. 4.9 we compare the throughput[13] of New Reno and Tahoe by varying the channel memory (f_dT). When errors are correlated ($f_dT = 0.01$), New Reno presents a slightly lower throughput than Tahoe. In this case, a burst (b) of erroneous packets follows the error at time n. At this point, TCP Tahoe reacts to the reception of the K-th duplicate ACK by re-starting the connection with the Slow-Start algorithm. From this point on, the $b + 1$ lost packets are retransmitted as if a timeout event had occurred. In the New Reno recovery phase, instead, it is possible to transmit only one packet per round trip time (m slots). So, to recover from the $b + 1$ losses and restore the normal transmission mode a total of $m(b + 1)$ slots are needed. For large values of b (e.g. for $f_dT = 0.01$) this strategy (New Reno) leads to a larger recovery time with respect to Tahoe. This is the reason why, in this case, New Reno is affected by a lower throughput. Note that this problem is not present when instantaneous feedback ($m = 1$) is considered, while it becomes more relevant as m increases.

The situation is reversed at low channel correlation ($f_dT \in \{0.08, 0.64\}$). Here, the best recovery strategy is the one implemented by New Reno. This is justified by the fact that New Reno has a more persistent recovery strategy, able to recover multiple losses without leading to a timeout event. This event, in uncorrelated channel is more probable than in correlated environments, due to the fact that the error event (the error burst) probability in this case is larger even if the average packet error probability is the

[12] Here W_{th} is determinant for the time needed by W to reach m, i.e., to fi ll the bandwidth-delay product.

[13] Since the curves relative to the throughput lower and upper bounds are very close to each other, in the following graphs we report only the upper curve.

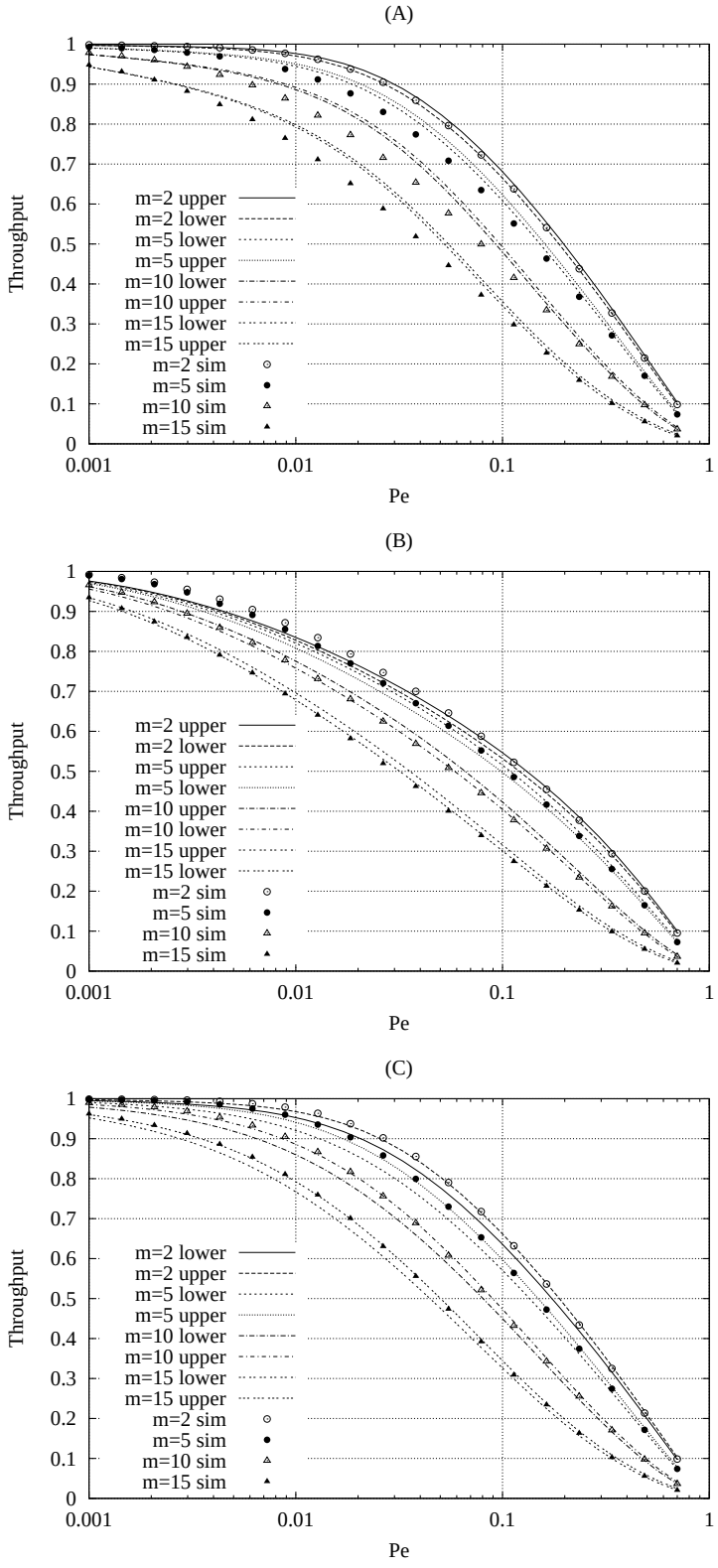


Figure 4.8: Throughput comparison between analysis and simulation with $f_dT = 0.01$, $K = 3$, $W_{max} = 16$, $T_o = 100$. (A) Tahoe, (B) Reno, (C) New Reno.

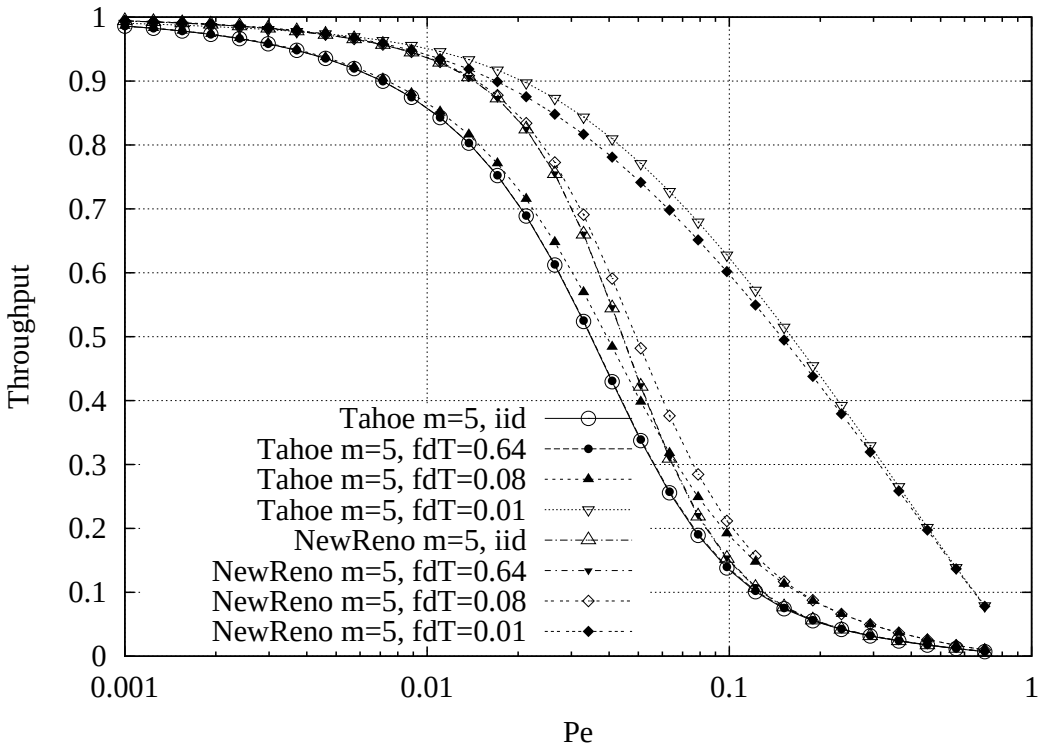


Figure 4.9: Throughput (upper bound) comparison by varying f_dT, $K = 3$, $W_{max} = 16$, $m = 5$, $T_o = 100$.

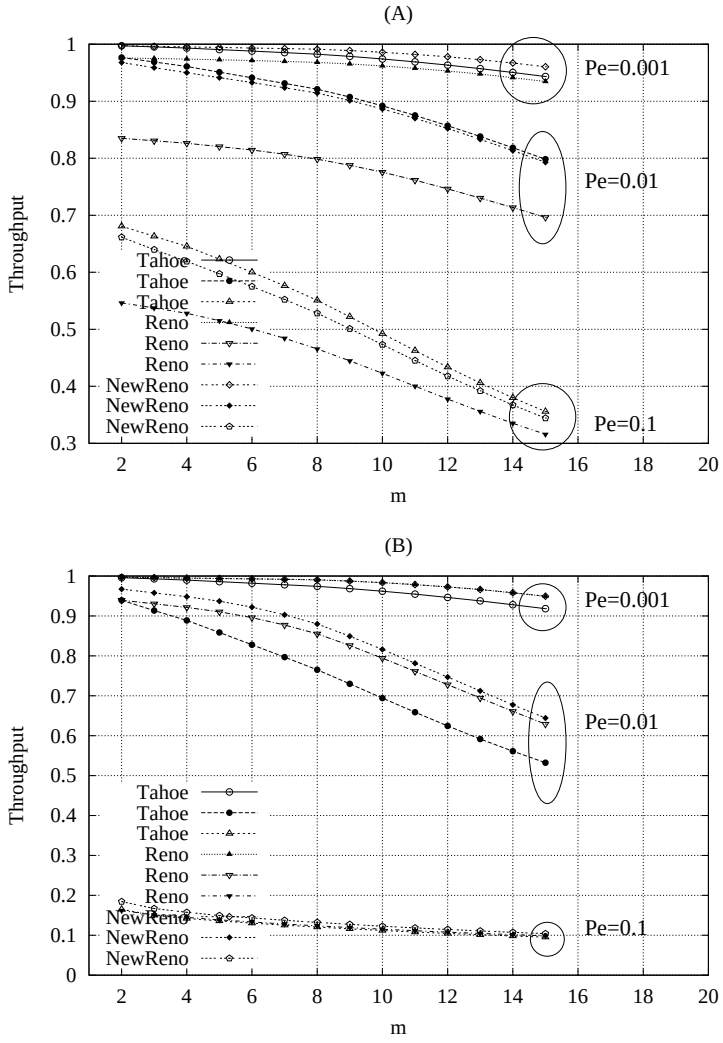


Figure 4.10: Throughput (upper bound) versus m with $K = 3$, $W_{max} = 16$, $T_o = 100$. (A) $f_dT = 0.01$, (B) $f_dT = 0.64$.

same.

Note also that, from the point of view of the protocol performance, a channel characterized by $f_dT = 0.64$ is equivalent to the independent error case (*iid*).

The throughput as a function of m is reported in Figs. 4.10, for the cases of $f_dT = 0.01$ (A) and $f_dT = 0.64$ (B). First of all, we note that uncorrelated errors have a higher impact on throughput than correlated channels. Moreover, in the $f_dT = 0.01$ case, TCP Reno is the one with the lower throughput, and New Reno and Tahoe present almost the same performance. Where $f_dT = 0.64$, instead, Tahoe becomes the worst, New Reno the best, whereas Reno performs in the middle.

Figs. 4.11 and 4.12 show the TCP throughput as a function of the channel error probability P_e by varying the channel correlation (f_dT) and W_{max}. We note that the curves depend weakly on the W_{max} value. In particular, when the channel is correlated ($f_dT = 0.01$) the greatest differences are in the New Reno case, i.e., where the Fast Recovery and Fast Retransmit algorithms are used. In such cases, in fact,

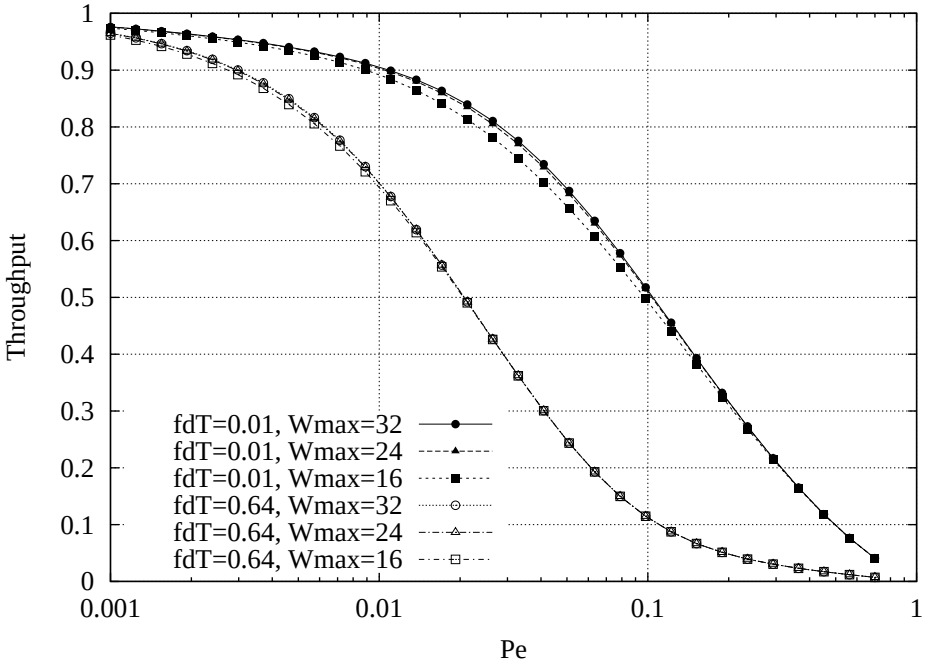


Figure 4.11: Tahoe, throughput (upper bound) versus P_e, by varying W_{max} and f_dT with $K = 3$, $m = 10$ and $T_o = 100$.

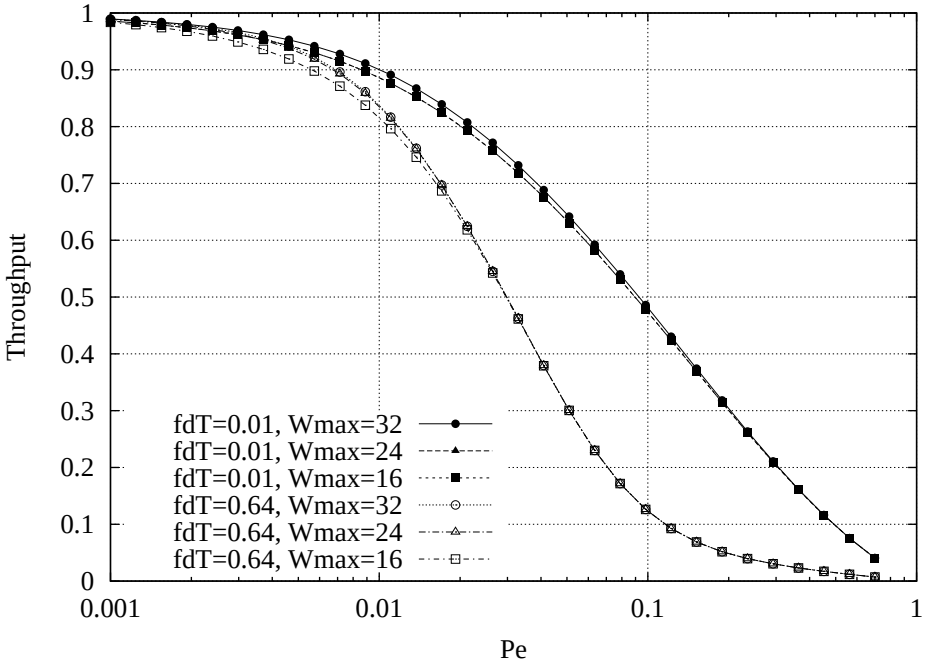


Figure 4.12: NewReno, throughput (upper bound) versus P_e, by varying W_{max} and f_dT with $K = 3$, $m = 10$ and $T_o = 100$.

the success of these recovery algorithms depends on the maximum window size reached during the error-free phase (first cycle in our analysis): the larger the value of the window, the higher the probability that, after the first error, K packets are sent with success. This results in a higher probability of triggering the Fast Recovery algorithm, thereby avoiding timeout and increasing the performance.

We can conclude that the maximum window size must be greater than the bandwidth-delay product ($\geq m$) in order to reach the continuous transmission phase by avoiding the waste of available resources. However, once it has been set following this rule, its size is not determinant to improve performance regardless of the degree of correlation characterizing the packet error process.

4.5 Conclusions

In this Chapter, an analytical framework, based on Markov renewal theory, has been developed to describe the throughput performance of a TCP protocol working over a wireless link. The main goal of our analysis was the study of TCP performance in a link characterized by a finite round trip delay. Even though some approximations are introduced for analytical tractability, comparison with simulation results shows that the considered approach is very accurate and is able to predict very precisely the throughput performance of various versions of TCP. The main findings of this study are that (1) an increasing round trip time may significantly affect the throughput performance of TCP, especially when an independent channel is considered, (2) New Reno performs better than Reno and Tahoe when the channel is uncorrelated, whereas the Tahoe recovery strategy is the most efficient when the channel correlation is high and (3) the maximum window size does not play a determinant role in increasing throughput performance in both correlated and independent channels.

Chapter 5

UMTS RLC Parameters Setting and their Impact on Higher Layers Performance

In the previous Chapters, the performance of TCP has been investigated both analytically and by simulation. From these studies, the effect of the losses due to the wireless channel and the implications of the error burstiness on protocol performance have been clarified. In particular, it has been proved that TCP performs poorly over radio channels, since it is unable of discriminating between packet losses due to congestion (occurring in the wired part of the network) and packet corruptions due to the wireless channel impairments. In more detail, TCP reacts to every error as if a congestion phenomenon had occurred, by slowing down its transmission rate or even timing out (i.e., restarting the transmission at the lowest speed and introducing long idle times). Obviously, these mechanisms are effective over wired networks, where congestion is the primary reason for packet losses. In such networks, indeed, the idling and retransmission is a good choice which is aimed at decreasing the flow injected into the network by TCP, i.e., at alleviating the traffic in an already congested network. On the contrary, in wireless networks, or in general in networks including a wireless link, a different solution has to be found for TCP to cope with the inefficiencies mentioned above. A possible solution, which is the one of shielding TCP from wireless errors by using link layer ARQ (Automatic Retransmission reQuest), is discussed in this chapter.

The chapter is subdivided into two parts. In the first part we present some results for TCP operating over a standard Selective Repeat ARQ protocol. The chief aim of this preliminary study is to highlight the benefits as well as the trade–offs involved in the usage of ARQ algorithms. In the second part of the chapter a more complex ARQ algorithm is considered. In particular, performance evaluation is carried out considering the Radio Link Control (RLC) level of the Universal Mobile Telecommunications System (UMTS). The focus, in this case, has been on the link layer parameters setting and its impact on both forward and backward link performance. Moreover, End-to-End performance is obtained considering the TCP flow. The main findings of the investigation reported in the next are that: i) a correct setting of RLC parameters is pivotal to avoid deadlock events, that would lead to a dramatic performance degradation; ii) schemes using receiver-driven retransmission requests give the best performance; iii) the EPC mechanism [4] gives very good performance in terms of delay, throughput and energy; iv) anomalous behaviors, leading to delay jitter performance degradation, are observed for some receiver-driven schemes. Again, EPC is able to overcome these problems.

PART I: TCP over a Link Layer Selective Repeat ARQ Scheme

In this first part of the chapter some results for a TCP operating over a Selective Repeat link layer ARQ are given. In the scenario considered here, a mobile terminal is connected to the cellular network and it is transmitting data to a fixed terminal, which is placed on the fixed Internet. A serving base station is in charge of ensuring user connectivity and providing power control. A Wideband-CDMA cellular system is considered, where, as assumed in previous chapters, several parameters[1] can be triggered to obtain W-CDMA channel traces. These traces are used to characterize the wireless medium, whereas fixed delay and IP error rate are taken into account for what concerns the wired part of the connection. The TCP protocol is responsible for the End-to-End error recovery, whereas over the wireless link, a link layer ARQ is used to retransmit the packets corrupted over the wireless channel. The main function of ARQ is here to shield the TCP (which is operating at higher layers) from the errors occurring over the radio path. In the next Section results are given to understand the effectiveness of the ARQ protocol.

5.1 Results for TCP operating over a SR-ARQ

In this section some preliminary results on the performance of the TCP level running over a Selective Repeat link layer ARQ are presented. In all the obtained graphs the presence of the ARQ level appears to be very important to improve both throughput and energy. Here, the energy metric is labelled as E_{cb} and consist in the average energy expenditure per correct information bit transmitted. This metric accounts for retransmissions, TCP idle times, spurious retransmissions and any other form of redundancy. According to the results reported in the next, we can affirm that ARQ is able of recovering losses in a transparent way by considerably reducing the packet error probability P_{SDU} on the TCP data packet flow. This P_{SDU} reduction leads to a decreasing number of TCP retransmissions, that in turn lead to a reduction of E_{cb}. Some metrics and system parameters will be considered in the remaining of this chapter: the system load is expressed in terms of number of users in the cellular system (N_u), the user mobility is accounted by means of the Doppler frequency (f_d), the Time Transmission Interval (TTI) is the time period over which the physical layer interleaving is performed, L is the maximum number of retransmission allowed at the ARQ layer, P_{SDU} is the IP packet error probability, with S we refer to the TCP throughput and with E_{cb} and E_t to the mean energy per correct information bit and to the total consumed energy, respectively.

As a first results, in Fig. 5.1, the TCP throughput is plotted against the average Frame Error Rate (FER) for a system where $N_u = 120$, $f_d = 6$ Hz, $TTI = 1$ by varying the TCP version. Two sets of curves are presented, in the first one the RLC operates in *transparent mode*, i.e., L is set to 0, while in the second set the RLC is configured with $L = 10$. From the graph, the beneficial effect of ARQ with $L = 10$ is evident, in particular, for a FER of 0.1, with $L = 0$ we observe a very low throughput, whereas the throughput approaches 0.8 in the case where $L = 10$. Moreover, in this last case, the effect of the TCP protocol version vanishes.

In Fig. 5.2(a) the average TCP Window Size is reported instead of the throughput value, whereas in

[1] As system load, user mobility interleaving depth and physical layer coding strength.

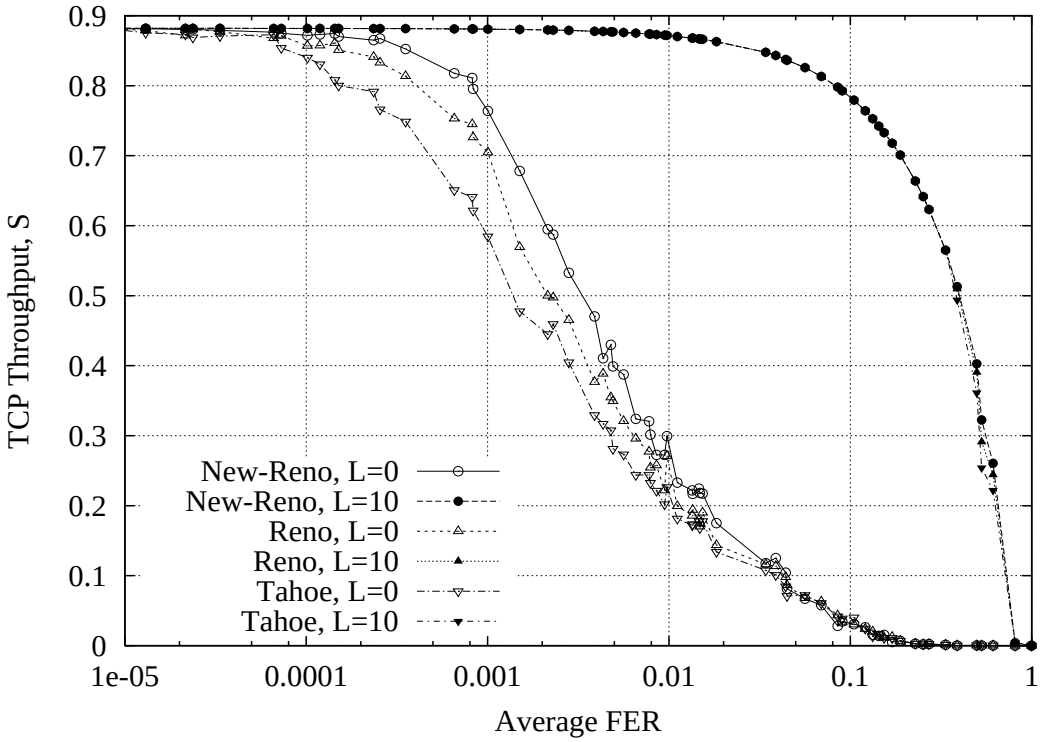


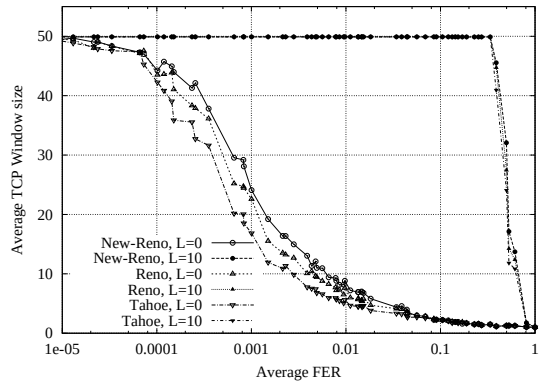
Figure 5.1: TCP throughput as a function of the FER.

Fig. 5.2(b) we report P_{SDU}. For all these metrics the advantage due to an underlying link layer appears very evident.

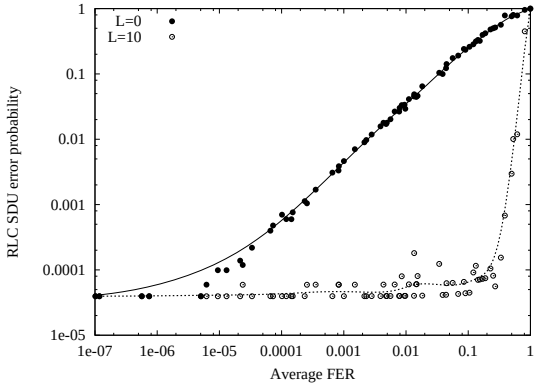
In Fig. 5.2(c) the total consumed energy, E_t, is reported for both the $L = 0$ and the $L = 10$ cases. From a first look to that figure, one may assert that the $L = 10$ case, for high FER values, is affected by a higher energy consumption. However, for what concerns E_{cb} (see Fig. 5.2(d)), the situation is reverted. In conclusion, the low E_t value in the case of $L = 0$ is simply due to a low throughput, i.e., in such cases the protocol is rarely transmitting and it is going in timeout repeatedly. Furthermore, after each timeout, the last unacknowledged packet is retransmitted; this last behavior is the one responsible for the increased E_{cb}.

In Fig. 5.2(e) the TCP throughput is reported by varying f_d and L. From the figure, it is clear how the influence of the physical parameter f_d also vanishes by increasing L. In fact, with a transparent mode link layer, the throughput is higher for correlated channels (i.e., low f_d values) and presents a minimum for the *iid* case. By setting L to 10, instead, all the curves overlaps by identifying a unique trend.

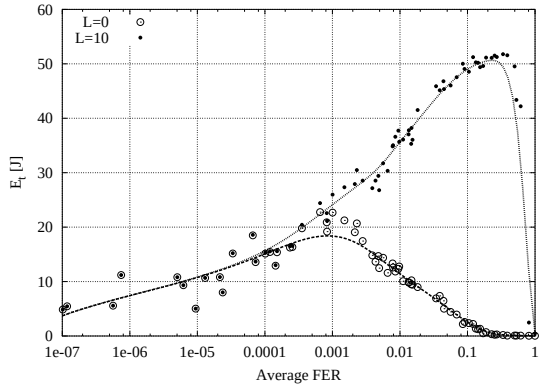
Finally, in Fig. 5.2(f) the dependence on TTI is plotted. In more detail, a greater TTI always leads to a lower FER. However, by comparing performance between users affected by the same FER a unique trend is observed. This means that the burstiness of the underlying channel is not determinant for the throughput when an efficient link layer protocol is considered. As will be shown in the remaining of this thesis, the error burstiness is instead important for what concerns delay performance.



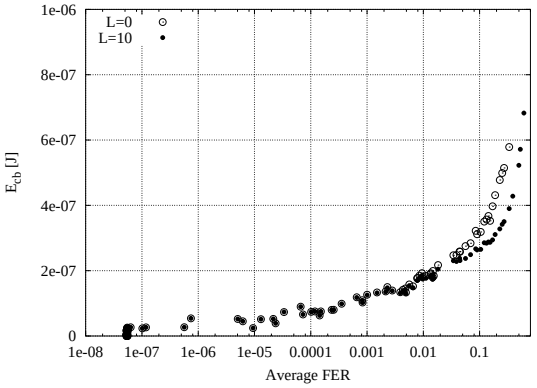
(a) Mean TCP window size Vs. FER.



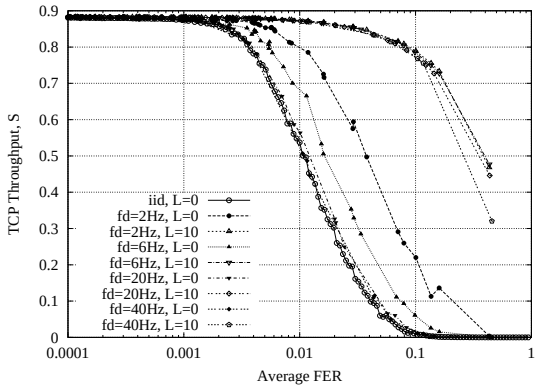
(b) P_{SDU} Vs FER.



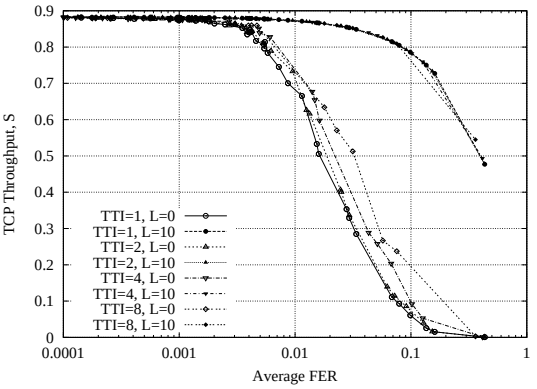
(c) E_{cb} Vs. FER.



(d) E_t Vs FER.



(e) Impact of the f_d value.



(f) Impact of the TTI value.

Figure 5.2: Impact of ARQ on TCP performance.

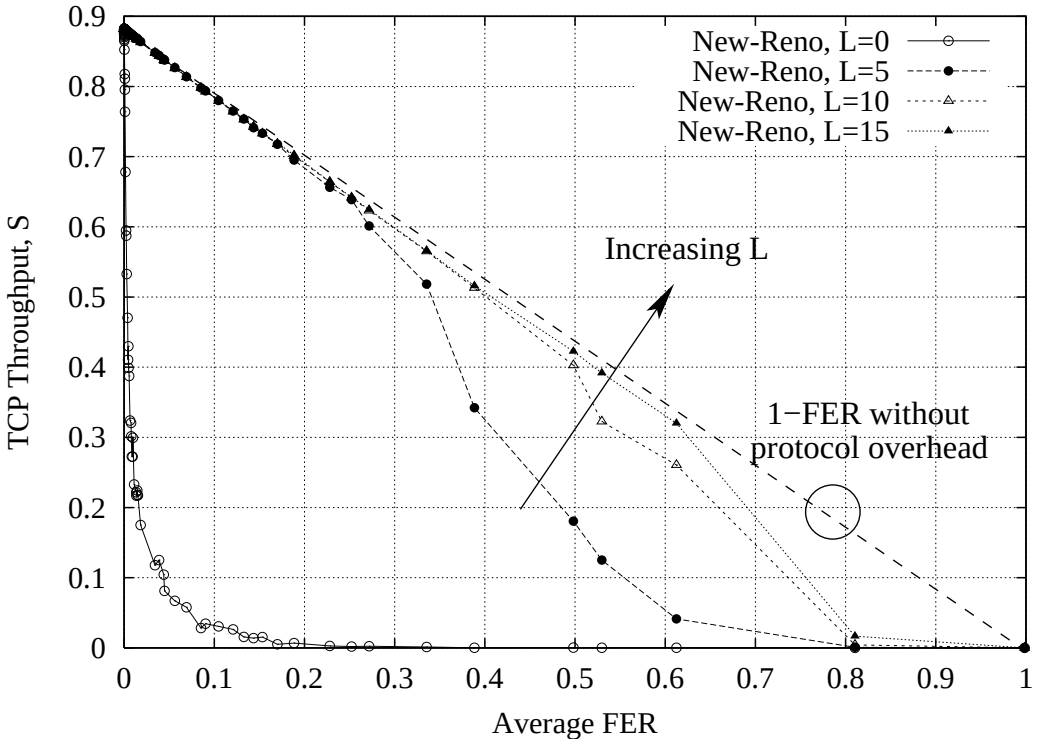


Figure 5.3: TCP throughput by varying L.

In Fig. 5.3 we report the TCP throughput, S, as a function of the FER by varying L. Here, the graph is plotted by using a linear scale in order to highlight that the TCP throughput curve as L increases[2] approaches the line $S = 1 - FER$. This formula correspond to the theoretical throughput curve of the Selective Repeat scheme. In the regions where this equality holds the TCP protocol is behaving as a packet source, i.e., the TCP packet transmission time[3] is simply modulated by the ARQ action and the TCP timeout events are completely avoided.

Finally, an interesting trade-off figure is shown in Fig. 5.4. Here, the TCP throughput is plotted against E_{cb} for various FER values by varying L (which is chosen in the set $L \in \{0, 5, 10, 15\}$). All the curves have a strictly monotonic decreasing behavior, that is the reason why the better choice in terms of both throughput and energy efficiency is to configure the ARQ level with a large L, i.e., it is not worth limiting L from both a throughput and energy expenditure point of view. Moreover, for each value of the FER we can locate a threshold value of L, say L^*, such that increasing L beyond L^* does not leads to any advantage. In the presented figure, this behavior can be observed for $FER = 0.1$ and $FER = 0.25$; in these curves the points corresponding to $L = 5$, $L = 10$ and $L = 15$ overlap.

[2]Once the TCP and ARQ protocol overheads have been removed.
[3]It is the time that the ARQ employs to transmit all the PDUs composing the TCP packet.

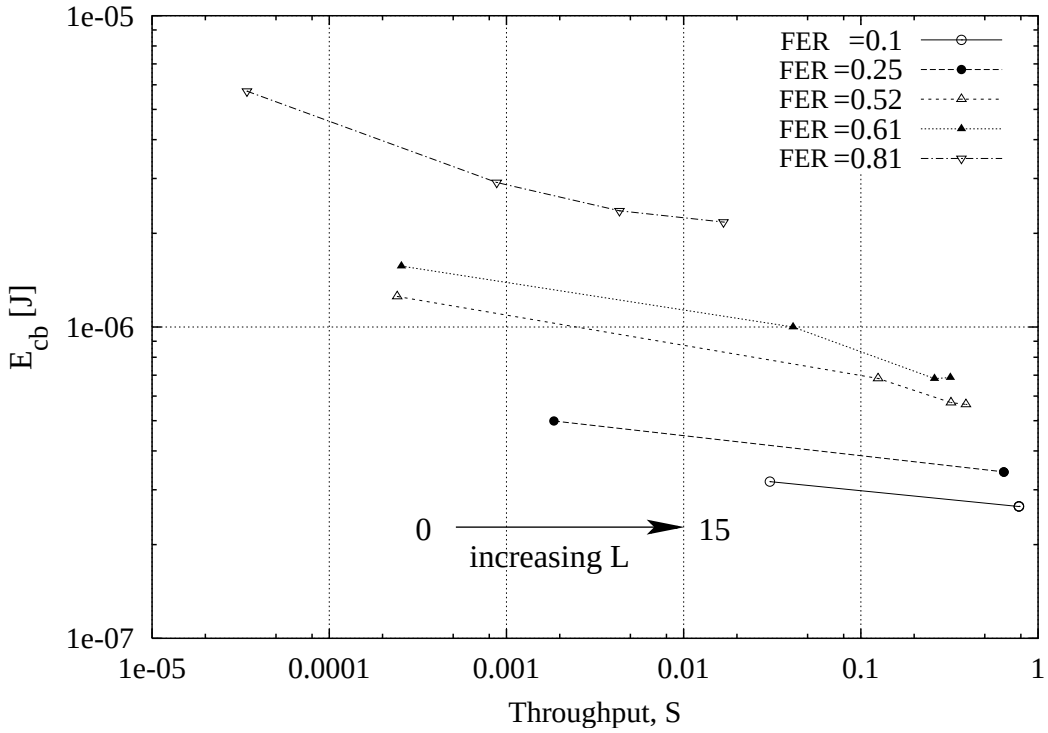


Figure 5.4: Trade-offs between throughput and energy.

PART II: TCP over the UMTS RLC

The Universal Mobile Telecommunications System (UMTS) is a 3rd generation system based on the W–CDMA technique, where voice, high-speed data traffic and multimedia applications can be supported. In UMTS, the Radio Link Control (RLC) layer is placed on top of the Medium Access Control (MAC); its function is to counteract the residual errors (after physical layer processing) by performing retransmissions of lost data, thus providing a reliable data transmission channel.

In this work we are interested in the characterization of higher layer protocols performance in UMTS and, in particular, in the effect of the RLC parameters setting. This understanding is pivotal for higher layer flows to fulfill the desired QoS requirements. Here, this study is carried out considering the TCP protocol as the transport layer mechanism. The TCP assumption that losses are due to network congestion becomes problematic in wireless links, where indeed losses are often due to signal degradation. The RLC layer can overcome this fact by transparently recovering from channel errors before the TCP timeout expiration occurs. However, depending on the RLC configuration, this goal can be achieved with different delay, throughput and energy consumption performance.

Several contributions on the RLC setting have been proposed so far in the literature. In [53], simulation results are reported to highlight the importance and the role of a correct setting of some RLC parameters such as the *Pool Empty TX Buffer*, *Pool Empty RETX Buffer*, *Pool Every X SDUs* and *SDU Discard Timer* [4]. In [71], instead, the influence of the maximum number of allowed retransmissions (MaxDAT) is studied in detail. The effect of the RLC window size is considered in [54] and the setting of the various RLC timers has been investigated in [103]. In this work, we go further with respect to all these

contributions for the following reasons: following the guidelines derived in the previous research, we propose three different and complete RLC setting modes and we evaluate their delay, energy and throughput performance. The effect of independent and correlated losses is considered. The EPC mechanism [4] and its benefits are investigated.

The remaining of this chapter is structured as follows: in Section 5.2 the reference scenario is introduced, in Section 5.3 the main RLC parameters are presented and the three different RLC setting modes are specified. The performance of these RLC schemes is then evaluated in Sections 5.4 and 5.5, where some important metrics are shown for the independent and correlated error case, respectively. In Section 5.6 higher layer packet delay performance is reported, by highlighting an anomalous behavior for some of the proposed schemes and finally, Section 5.7 concludes the chapter.

5.2 Reference Scenario

We consider a UMTS mobile Terminal Equipment (TE) that is communicating with its serving Base station (BS) through an uplink Dedicated CHannel (DCH). The RLC level is used between ME and RNC [1] to recover from residual errors at the output of the physical layer. Then, TE data is forwarded to a Fixed Host (FH) placed in the Internet. Errors in the RLC forward channel (where data packets flow) are modeled by means of a discrete-time two-state Markov Model that evolves frame by frame (the frame duration is 10 ms). Despite its simplicity, this model allows us to derive some precise and useful indications on the impact of error burstiness on the RLC performance. All PDUs transmitted in a bad state are considered to be lost, whereas the ones sent during good states are always correctly received. The Frame Error Rate (FER) is defined as the average probability to be in the bad state. Errors are also considered for what concerns the backward RLC flow, i.e., regarding RLC acknowledgments (called *Status Reports* [4]). Here, an independent error process is adopted, i.e., status messages are lost with a fixed probability, P_{status}. The RLC layer round trip time has been considered to be constant and equal to RLC RTT=100 ms. The simulation tool used to perform our performance evaluation is similar to the one described in [53].

5.3 UMTS RLC Scheme: Configuration Parameters and Settings

In this section, we first give a brief description of the RLC layer mechanism and of the main parameters involved in its configuration, then we give some guidelines for their correct setting and finally we propose three RLC configuration modes, namely, *delay*, *resource* and *throughput optimized*.

5.3.1 Basic concepts

The RLC is basically a Selective Repeat (SR) ARQ scheme where some new features have been added to improve performance and to provide a fully programmable and flexible retransmission mechanism. Packets incoming from higher layers (RLC SDUs) are segmented in RLC PDUs, that are the packet units at the RLC. A Cyclic Redundancy Check (CRC) field is attached to each RLC PDU to perform error detection. The receiver RLC entity based on the CRC field and on the gap detection technique[4] sends

[4]The received PDU flow is checked for discontinuities in the sequence numbers.

back to the transmitter *Status Report* messages containing a list with the PDUs requested for retransmission. The sender can force the transmission of status reports by sending special *Poll* packets[5]. At the receiver a *Poll PDU* causes the immediate transmission of a new status report[6]. This procedure is useful to avoid deadlock situation and, in general, to control the amount of acknowledgment messages sent by the receiver (backward flow). In addition, many features have been included in the RLC to reduce the retransmission delay, prevent deadlock and reduce the feedback channel overhead. Their detailed explanation can be found in the standard [4] as well as in [53] [71]. The detailed explanation of the RLC parameters is out of the scope of this chapter. The interested reader is referred to the references above.

5.3.2 RLC Schemes Proposal

Some general guidelines must be kept in mind when we are configuring the RLC layer:

- At least one mechanism to force the periodic transmission of status messages must be accounted for. This is necessary in order to avoid deadlock at the RLC sender. For this purpose, two mechanisms must be set: the *Poll Timer* and the *Poll Empty TX Buffer*. The *Poll Empty TX buffer* provides to the transmission of a *Poll PDU* when the last PDU in the transmission buffer is sent. Once the Poll has been sent, the *Poll Timer* forces a periodic retransmission of the Poll (when the timer expires). This retransmission process is stopped when the requested *Status PDU* is finally received. These features are set in all the schemes proposed in the next to obtain deadlock free mechanisms. Some useful insights on these settings can be found in [53].

- Poll packets should be sent frequently to reduce the retransmission delay [53]. By this way more status reports are sent back to the receiver that, as a consequence, can rely on promptly updated retransmission information. However, the feedback flow (*status reports*) is directly related to the backward channel energy expenditure. Clearly there is a trade-off of the polling frequency.

- *Sender-Driven* schemes have the advantage of controlling the sender window so as to avoid stall situations. In pure Sender-Driven approaches, it is the sender that, based on the state of the sender window, RLC queues and RLC flow at the transmitter, controls the transmission of status messages by polling the receiver.

- In *Receiver-Driven* schemes, it is the receiver that, relying on the *Check Missing PDUs* [4] mechanism, infers lost packets and sends back *status reports* accordingly, i.e., the backward flow is governed by the receiver.

We verified that *Sender-Driven* schemes are always inferior to *Receiver-Driven* mechanisms and that the best performance is achieved in the hybrid case, i.e., when both sender and receiver features are used together. In the following, referring to the concepts introduced above, three different RLC setting modes are presented:

[5] A Poll bit is set in the normal data PDU.

[6] Unless the *Prohibit Status Timer*, whose function is explained in the following, is set.

SCHEMES →	**RO**	**DO**	**EPC**
RLC transmitter params			
Poll Empty TX Buffer	yes	yes	yes
Poll Timer	120 ms	120 ms	150 ms
Poll Every X PDUs	no	no	no
Poll Every X SDUs	no	no	no
Poll Empty RETX Buffer	yes	no	no
Poll Window (%)	80 %	80 %	80 %
Periodic Poll	no	no	0
Prohibit Poll Timer	no	no	no
RLC receiver params			
EPC	no	no	yes
Prohibit Status Timer	60 ms	110 ms	110 ms
Check Missing PDUs	yes	yes	yes

Table 5.1: RLC schemes configuration parameters.

- **Delay Optimized (DO)**: in this scheme the *status reports* transmission is controlled by the *Check Missing PDUs* feature. To avoid deadlock at the sender, the *Poll Window* mechanism must be activated, that is, the transmission of a *Poll PDU* is forced whenever the sender window increases beyond a given threshold. The aim of this scheme is to aggressively and promptly retransmit lost PDUs so as to minimize the delay. This is achieved thanks to a continuously updated (and redundant) backward status PDU flow. In particular, the receiver *Prohibit Status Timer* is set to a value lower than the RLC layer round trip time (RLC RTT). When this timer is activated, *status PDUs* can be periodically sent at its expiration times. Setting the timer value is a method for controlling the maximum number of *status messages* flowing back in a RLC RTT. Finally, in the DO scheme, a *Poll PDU* is sent when the retransmission buffer is empty (*Poll Empty RETX Buffer* feature).

- **Resource Optimized (RO)**: this scheme is similar to the previous one except for the fact that the *Poll Empty RETX Buffer* is not set and the *Prohibit Status Timer* is set to a value greater than RLC RTT. In this way at most one *status message* can be sent in every round trip time. This scheme has the advantage of saving bandwidth (and energy) in the feedback channel, but this comes at the cost of a lower retransmission promptness and, as a consequence, of longer RLC SDU delays.

- **EPC throughput optimized**: both the EPC mechanism [4] and the *Check Missing PDUs* are enabled. The *Poll Window* mechanism is triggered to send a *Poll PDU* whenever the sender window reaches 80 % of its maximum size.

These settings are summarized in Table 5.1.

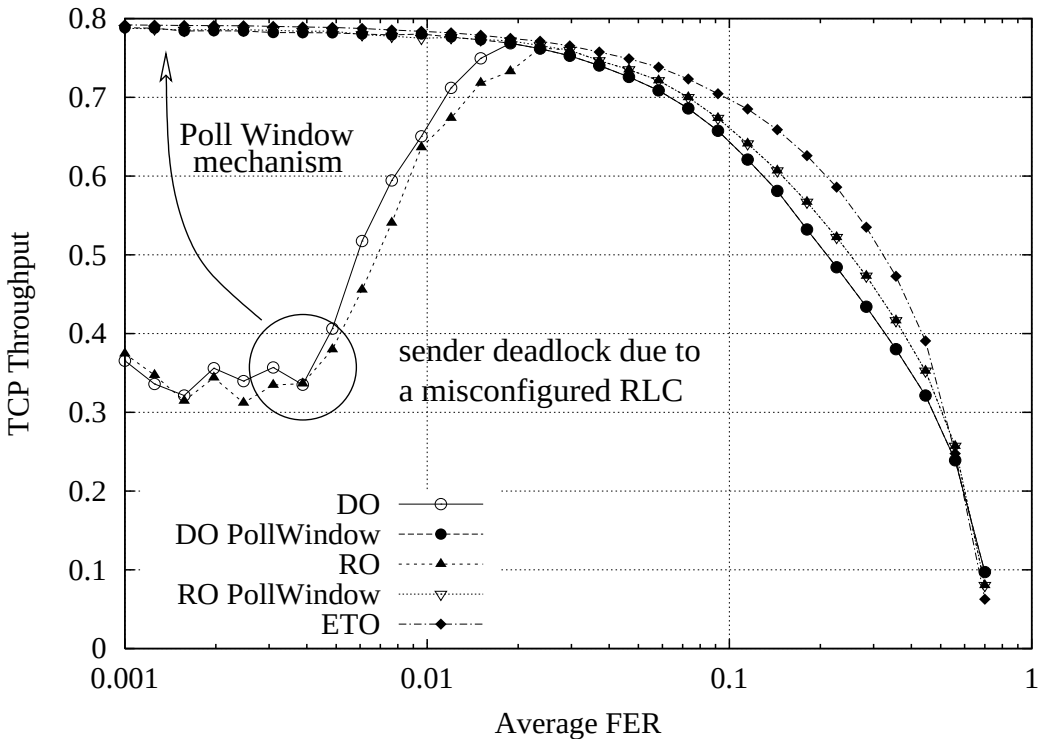


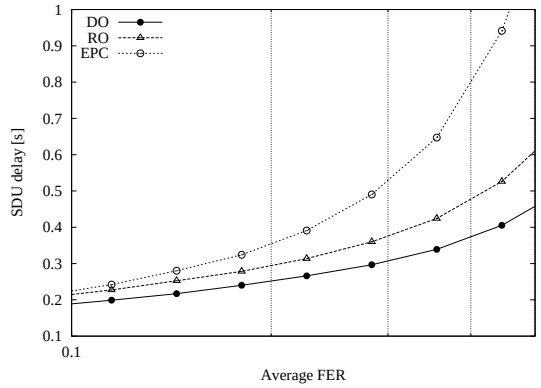
Figure 5.5: TCP throughput comparison between the selected schemes in an independent channel, 100 Kbyte file transfer.

5.4 Comparison between RLC Schemes in an Independent Channel

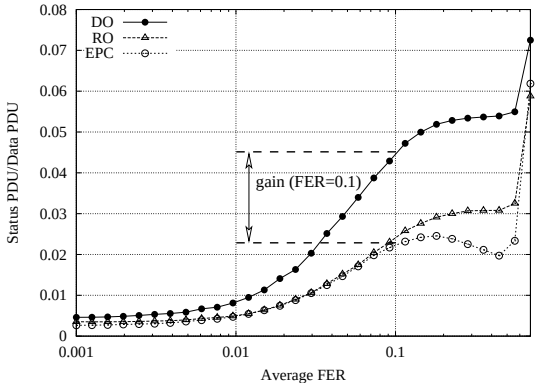
In the results reported here, we focus on FTP data transfer performance, i.e., we consider a continuous data source modulated by the TCP window mechanism. Two data transfers, of 5 and 100 Kbyte are taken into account to highlight the role of the TCP Slow-Start phase. The discontinuous transmission phase, due to TCP, is dominant in the 5 Kbyte case, where the Slow-Start is dominant, whereas in the 100 Kbyte case, the discontinuous transmission is less influential since here the TCP window size is large enough to fill the transmission channel. We have performed intensive simulations to characterize the proposed schemes. In the sequel, due to space constraints, we only report some of them to highlight the main findings of our investigation.

In Fig. 5.5 we compare the throughput for the schemes presented in Section 5.3.2. To show the importance of the window control feature, the RO and DO schemes are also reported here in their version without window control (i.e., where the *Poll Window* mechanism is not activated). It appears evident how the throughput in that case is heavily degraded at low FER. The reason for this phenomenon is that the transmission of *status PDUs* is triggered by the detection, at the receiver, of missing PDUs, but when the error probability is low this is a rare event. As a consequence, few status reports are sent back to the transmitter and the RLC window may be saturated[7]. The mean SDU delay (100 Kbyte case) is reported

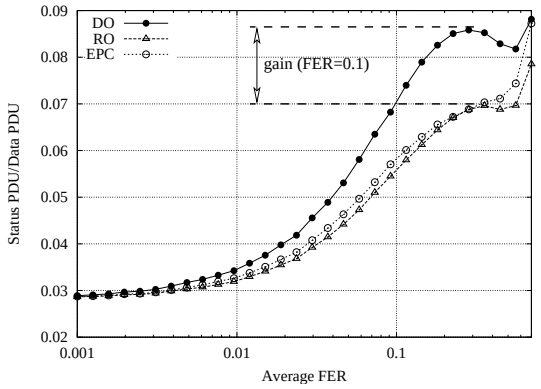
[7]The role of a status message is also to advance the transmission window by allowing the transmitter to release from the



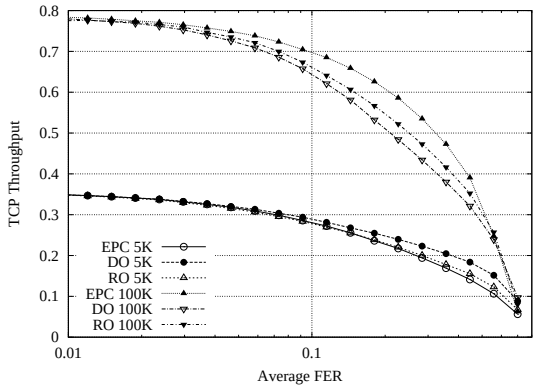
(a) SDU delay in an independent channel, 100 Kbyte file transfer.



(b) Feedback channel efficiency, 100 Kbyte data transfer.



(c) Feedback channel efficiency, 5 Kbyte file transfer.



(d) TCP throughput as a function of the FER. 5 and 100 Kbyte FTP data transfers.

Figure 5.6: Performance in an *iid* channel.

in Fig. 5.6(a). From that figure, it is clear how the DO scheme can effectively reduce the SDU delivery delay. The EPC scheme for this metric is the one with the worst performance, whereas the RO performs in the middle. The same relationship among schemes is maintained for the 5 Kbyte case (not shown here). In Figs. 5.6(b) (100 Kbyte) and 5.6(c) (5 Kbyte) we focus on the channel efficiency defined as the ratio between the number of transmitted data PDUs (forward flow) and the number of status PDUs flowing over the backward channel, i.e., the average number of status messages sent back to the transmitter for every transmitted data PDU. It is worth noting that the EPC scheme is able to save bandwidth in the feedback channel; in both the 100 and 5 Kbyte data transfer cases DO is the scheme with the highest feedback channel utilization. Referring to $FER = 0.1$, the relative saving obtained by RO and EPC schemes is roughly 17 % and 50 % for the 5 and the 100 Kbyte transfer case, respectively. In Fig. 5.6(d), we report the TCP throughput for the 100 and the 5 Kbyte transfer case. From the results presented in this section the following conclusions can be drawn:

- For what concerns the TCP throughput metric, EPC is the best scheme for the 100 Kbyte case, i.e., for large file transfers where the continuous transmission dominates, whereas DO performs better for short file transfers. The reason behind this is that when the Slow-Start phase dominates, i.e., when the transmission is discontinuous, it is important to decrease the feedback delay in order to permit a fast TCP window increase. When the TCP window size is large enough to fill the channel this behavior is no longer observed. In this last case, instead, a lower delay comes with a lower throughput as well.
- The RO scheme for all the considered metrics presents an intermediate behavior.
- The EPC scheme is the one where feedback messages are sent back with the highest efficiency, for both short and large file transfers. This scheme presents the highest throughput in the 100 Kbyte case, whereas its throughput is the lowest for short file transfers. The reason behind this is that optimizing the number of status reports flowing back to the transmitter leads to slightly longer delays in the TCP window advancement phase during the Slow-Start discontinuous phase.

5.5 Comparison between RLC Schemes in a Correlated Channel

In this section we report some results on the effect of the channel correlation on throughput and SDU error probability performance for the schemes introduced above. We utilize a Two State Markov Channel model to characterize the UMTS Frame error sequence, as explained in Section 5.2. An interesting result is reported in Fig. 5.7, where the FER is fixed at a high value ($FER = 0.5$) and the burst length (b) is varied in the set $b \in [1/(1 - FER) = 2, 30]$. From that figure, we can observe that the throughput curve increases for low b values, whereas it starts decreasing as b increases. This is due to two opposite behaviors. On the one hand, an increased b leads to a lower SDU delay. On the other hand, an increased b always leads to an increased SDU error probability[8]. The shorter SDU delay leads to a

buffer those packets that have been correctly received.

[8]In general, especially for high FER this metric is greater than zero because the maximum number of retransmissions per PDU (MaxDAT) is set to a fi nite value. In this chapter we consider MaxDAT=10. The effect of the MaxDAT parameter has been studied in detail in [71].

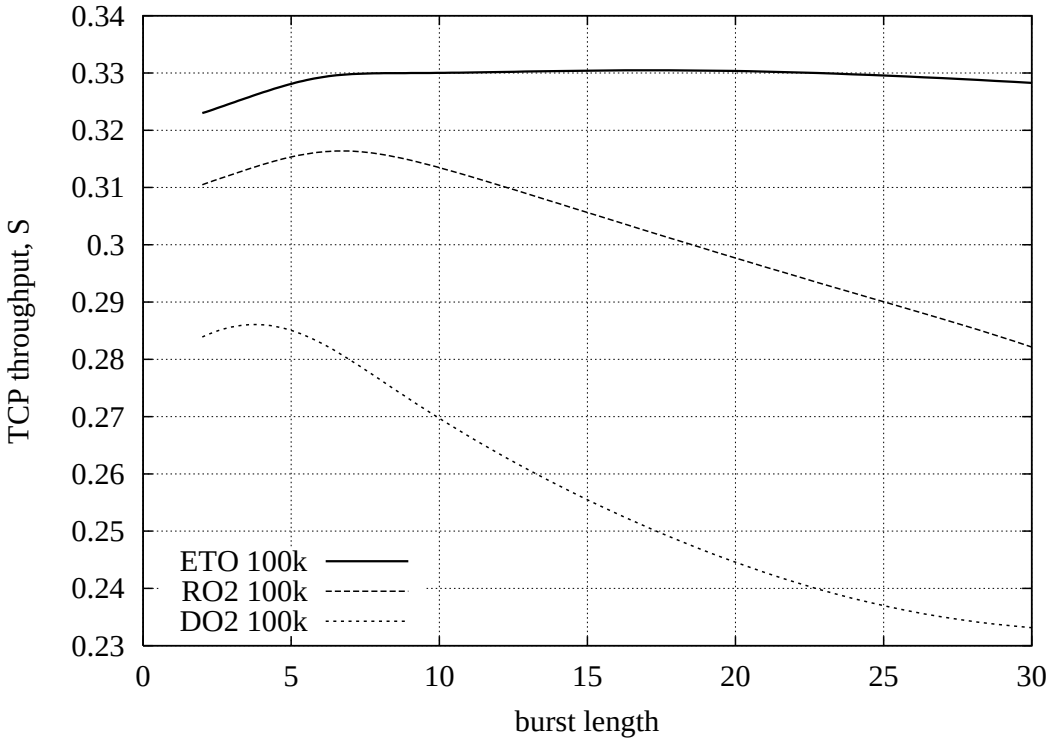


Figure 5.7: Throughput Vs burst length.

faster Slow-Start phase, and to the positive slope of the throughput curve in Fig. 5.7, for low b values. This effect is more notable for short file transfers, where the Slow-Start phase dominates. Moreover, as b increases the probability that a SDU is discarded increases as well. A SDU is discarded when one of the PDUs composing it reaches the maximum number of allowed retransmissions without having been correctly delivered. The increased SDU error probability leads to a reduction of the throughput, and this effect dominates in the second region of the curve in Fig. 5.7, i.e., for large b values. In Fig. 5.8, we report the SDU error probability as a function of the burst length. It is worth noting here that the EPC scheme is almost insensitive to an increased burst length. The explanation of this fact is found in the way this mechanism works [4]. In more detail, a timer is used to account for the time elapsed between the transmission of a *status report* and the reception of the first retransmitted PDU. This timer is usually set to an estimate of the RLC RTT. When this timer expires and no PDUs are received as a reply to the previous status report, then a new status report is transmitted and the timer is recomputed based on a new RTT estimate. In practice the number of status reports sent and their sending times is adaptively set based on the number of successfully retransmitted PDUs and on the estimated time between the sending of the status report to require their retransmission and their correct reception[9]. In the presence of a large burst of errors, the EPC at each RTT adaptively increases the inter-status sending time period, so that fewer status PDUs are sent with respect to the DO and RO schemes.

From the results presented in this section we can conclude that the EPC timer mechanism is able to manage the *status report* flow in a better way. The decision on the *status report* sending times is based on the detection of erroneous/lost PDUs (by means of the *Check Missing PDUs* feature) as well as on a

[9]For more details on the EPC mechanism the reader is referred to [4].

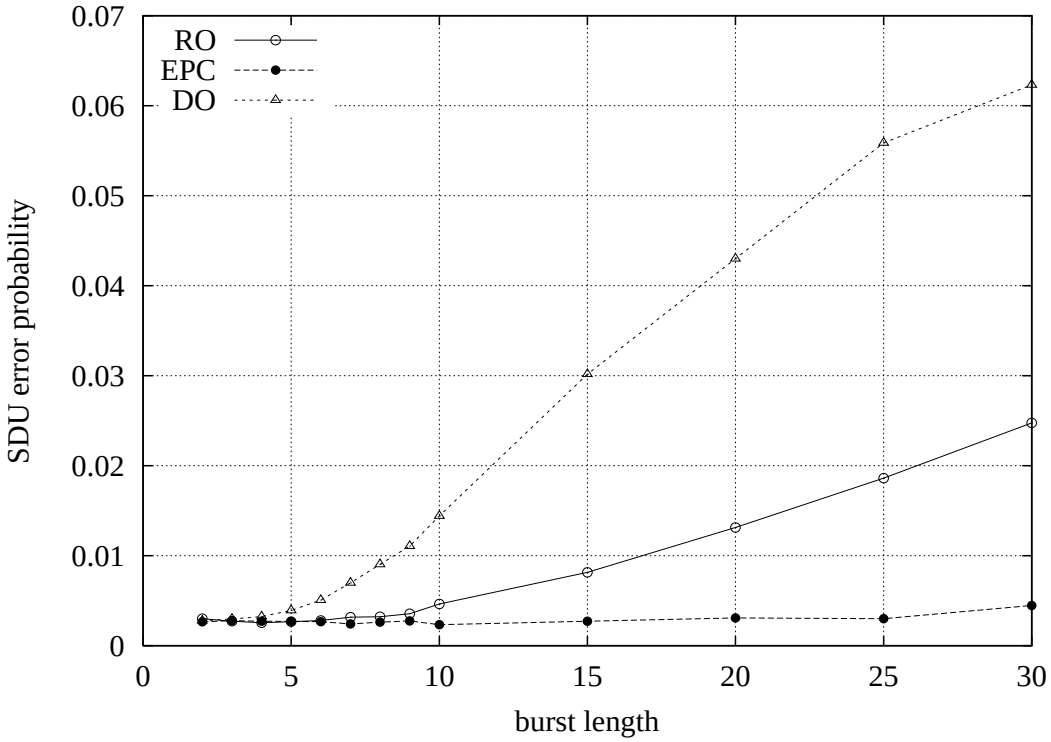


Figure 5.8: P_{SDU} Vs burst length.

consistent estimate of the RLC RTT[10]. This leads to a more efficient utilization of the feedback channel that in turn leads to:

- Increased throughput for large file transfers.
- Higher robustness against error bursts.
- Higher stability in terms of throughput performance under correlated errors.

5.6 Jitter Performance

In this section we consider the delay jitter metric and we show how the EPC mechanism is able to obtain superior performance also in this case. The jitter is defined here as the mean delay between the release (at the RLC receiver entity) of SDU packets. In Fig. 5.9 we report the delay jitter standard deviation as a function of the FER considering a *status PDU* error probability (feedback channel packet error probability) of $P_{status} = 0.05$. In the same figure, the mean jitter delay is reported for comparison. These results have been obtained considering the W-CDMA channel traces as in Section 5.1. The results in Fig. 5.9 are relative to a scenario where $N_u = 100$ users are randomly placed in a coverage area of 25 UMTS cells (wrapped around to avoid border effects) considering a slow fading condition, i.e., every user is characterized by a Doppler Frequency of $f_d = 6$ Hz.

[10]Intended here as the time elapsed between the transmission of a status report and the reception of the first retransmission elicited by that status report.

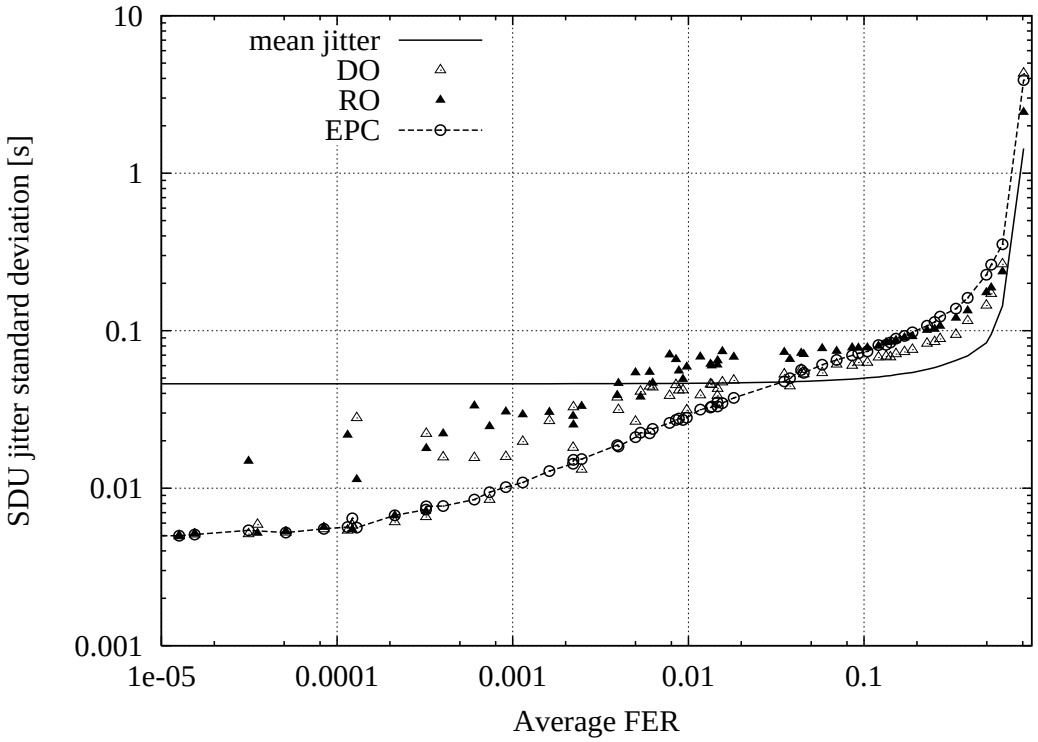


Figure 5.9: SDU Jitter standard deviation Vs FER.

Looking at Fig. 5.9 an anomalous behavior can be observed for the DO and the RO schemes. At low FER, their standard deviation is very high and as large as the mean jitter value. This behavior can be explained using the example situation in Fig. 5.10. In the first case (Fig. 5.10) we consider that two PDUs in a row are lost. The status report relative to the first PDU is lost, whereas that relative to the second packet is correctly received. Upon reception of the last status report, the two erroneous PDUs are retransmitted and a Poll is inserted in the second PDU (thanks to the *Poll on Empty RETX Buffer* feature). In the second case, only one PDU is lost and its status message is lost as well. At this point, no mechanism exists to ensure the correct reception of the status report. Suppose that no further PDUs are corrupted. Then, a *Poll PDU* is forced when the sender window reaches the *Poll Window* % of its maximum value. And only this *Poll* can trigger a new status report to be sent. In this case, a long time is needed to recover from the error since no mechanism is adopted to ensure the correct reception of the first status report. This causes the sender window to advance, by transmitting new packets and without performing the retransmissions requested in the lost *status*.

Once again, the EPC is able to overcome this problem. In fact, due to the EPC algorithm, reports are automatically resent based on round trip time estimates. The receiver, in this case, does not have to wait for the reception of a correct *Poll PDU* to re-schedule the *status message* retransmission.

5.7 Conclusions

In this chapter, the effectiveness of Selective Repeat ARQ link layer solutions in shielding the TCP from wireless channel errors is highlighted first. Then, in the second part of the chapter, TCP perfor-

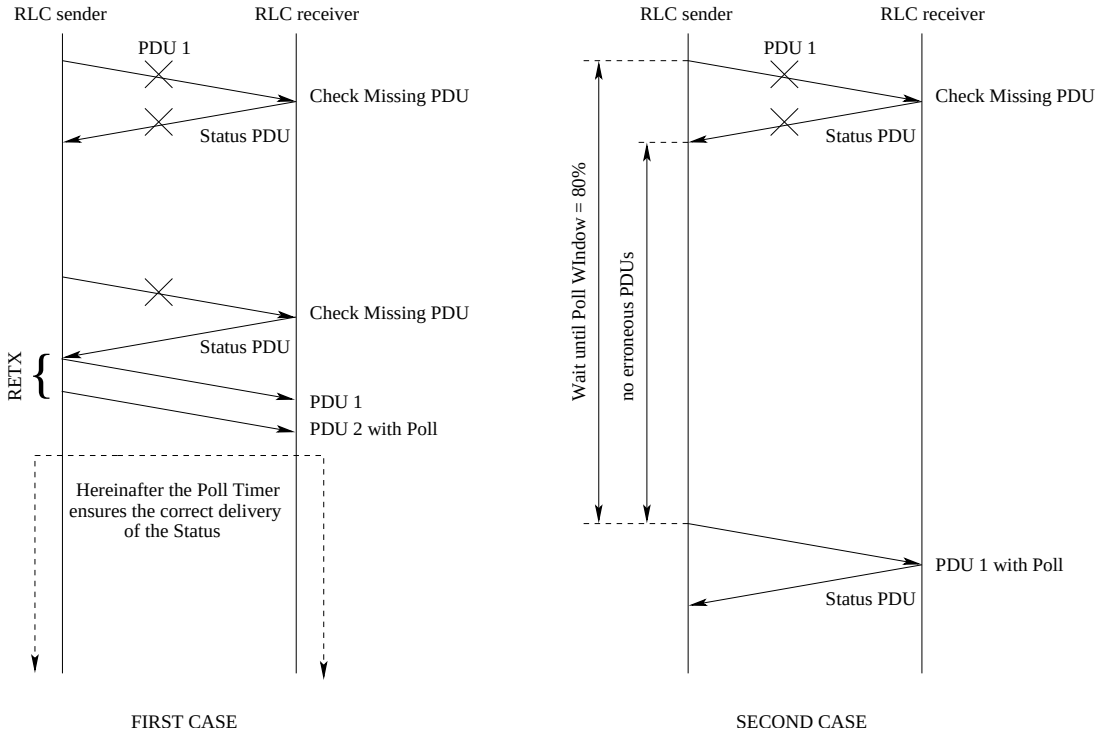


Figure 5.10: Error recovery in RO and DO schemes.

mance is evaluated considering the UMTS Radio Link Control as the protocol at the link layer. Some guidelines for the setting of the UMTS RLC parameters have been presented. Moreover, three different schemes have been proposed so as to minimize the delivery delay (DO scheme), the throughput (EPC scheme) and to achieve a trade-off between these two metrics (RO scheme). From the obtained results it is clear how the RLC setting can highly impact the higher layer performance. Based on the delay and throughput metrics, we suggest to use delay optimized schemes (DO) when the transmission is mainly in the discontinuous phase. In this case, the solution that minimizes the delivery delay also minimizes the throughput, when window based protocols (such as TCP) are considered at the transport layer. Regarding the energy consumption, the EPC scheme always represents the best choice, whereas DO is the worst. The RO scheme performs in the middle for all the considered metrics. Based on these observations a new possibility opens up, that is to use delay optimizing schemes when the transmission is discontinuous and to switch to energy optimized solutions when the channel is filled by transmitted packets. As a last conclusion, we observe that for jitter delay sensitive applications, the EPC scheme, among the schemes proposed in this chapter, is the only one with a stable behavior.

Chapter 6

Delay Analysis of Selective Repeat ARQ Algorithms over Wireless Channels

In the previous Chapter 5, link layer retransmission schemes have been considered and the advantages deriving from their usage have been discussed through simulation. Here, we go further with respect to these results, by focusing on the analytical modeling of ARQ algorithms. In particular, our main interest is the evaluation of ARQ delay statistics, which are directly affecting higher layers performance as well as link layer performance and buffer dimensioning. An analytical evaluation of such statistics is, in fact, useful to understand the behavior of such error control schemes as well as the implications of the channel error statistics on the user perceived performance. Moreover, such knowledge can be used to dimension several part of the system (such as the ARQ buffer) and to take run-time decisions about the actions to be taken at the link layer, e.g., whenever a retransmission for a packet should be required or not. These decisions often depend on the instantaneous quality experienced by the final user (running application). At this regard, several metrics can be considered; as an example, we report here the residual packet error probability, the video quality (which could be expressed in terms of video PSNR) or the packet dropping probability due to out of time reception of delay sensitive packets. Adaptive algorithms, operating directly at the link layer[1], which take into account such metrics to adapt their behavior so as to fulfill application requirements, are possible. Recently, several solutions exploiting these concepts have been or are going to be proposed as new link layer schemes for modern wireless devices and architectures[2].

Even if these algorithms are out of the scope of the research described in this Chapter, it is worth noting here that link layer delay statistics together with a detailed knowledge of the link layer behavior, turn out to be of practical relevance in their design. This gives us the motivation to go further with respect to the results provided in previous Chapters by providing here a detailed study of such statistics.

The study is carried out considering different channel models. The *iid* packet error model is considered first. This is the simplest model for packet errors, but is still of practical interest in some modern

[1] In the next Chapter, several algorithms are proposed to achieve an efficient delivery of multicast video streaming flows in 3G-WCDMA cellular networks.

[2] The reader is referred here to some either recent or running projects such as the Italian RAMON or the European AMBIENT NETWORKS, which both propose modifications at the link layer level to adapt the user perceived quality to fulfill application requirements.

wireless systems[3]. Beside the investigation on the independent error case, delay statistics are also obtained considering a Two-State and an N-State Markov model. For each channel model, the dependencies on the model parameters are highlighted and, where possible, approximations of the exact analysis are given to reduce its computational complexity.

6.1 Introduction and Motivations

In this first part of the Chapter, we compute ARQ packet delay statistics considering an independent packet error error model. As will be shown in Section 6.2 this simple model is still a reasonably good approximation for the actual errors experienced in a 3^{rd} generation cellular system such as the UMTS. For this reason, in the following (3G) cellular systems are discussed in some detail, by highlighting the services envisioned over such networks, the role of link layer techniques in the provisioning of such services and the importance of computing/estimating link layer delay statistics. The analysis in the correlated error case will be presented in Section 6.4 and 6.5 considering the Two-State and the N-State Markov channel model, respectively.

3G systems are intended to provide global mobility support, with wide range of services including telephony, paging, messaging, Internet, high quality video and broadband data. To realize a certain network Quality of Service (QoS), a Bearer Service with clearly defined characteristics and functionalities has to be set up from the source to the destination. The characteristics of this Bearer Service depend on the kind of traffic in which the user is interested. One of the challenges in such complex systems is to counteract the errors due to the wireless medium (propagation phenomena) and to the interference. With this aim at the physical layer both channel coding and interleaving are employed. In addition, ARQ techniques may be used at the link layer to reduce the residual error probability at the output of the physical layer. ARQ solutions use queueing and retransmission of lost packets to combat channel errors. The effect of link layer retransmissions is twofold. On the one hand, they are able to reduce the residual packet error probability. On the other hand, they introduce a random delay on the delivery of link layer packets that translates into a variable delay for the application layer flow. To better understand the importance and the implications derived from these additional delays let us refer to a video/audio streaming data transfer. In that case, packets arriving too late at the receiver should be discarded as they do not respect the timing constraints imposed by the streaming application running at higher layers. That is, due to the ARQ retransmissions process, the application streaming buffer at the receiver gets empty (*buffer starvation*) which results in gaps in the play-out process. This results in a sudden degradation of the user perceived performance. Therefore, when ARQ is employed, the degree of satisfaction of the user is directly affected by the error/delay statistics at the link layer output. In other words, the correct understanding of the delays involved in the ARQ retransmission process and their effect on higher layers performance is a pivotal point to provide and maintain the user desired QoS.

Our focus here is on the characterization of the delay statistics when ARQ is utilized at the link layer.

[3]One example for such systems is the UMTS cellular network, where as will be shown in the remaining of the Chapter, thanks to channel coding and interleaving, the errors experienced by link layer packets in many cases can be fairly well described by an independent model.

The fundamental ARQ schemes can be classified into Stop-and-Wait (SW), Go-back-N (GBN) and Selective Repeat (SR) ARQ [72] [82]. SR-ARQ is widely used due to its superior throughput performance. In ARQ schemes, the transmitter sends packets (PDUs) consisting of payload and error detection codes (a Cyclic Redundancy Check field is inserted into each packet). At the receiver side, based on the result of the error detection procedure, acknowledgment messages are sent back to the transmitter (ACK or NACK, according to the result of error detection). The sender schedules packet retransmissions based on such messages.

In the presence of an ARQ protocol, we can subdivide the overall PDU delay in three contributions [64]. The first contribution is the *queueing delay in the source buffer*, i.e., the time between the PDU release by higher layers and the instant of its first transmission over the channel. This term depends on the channel behavior, the ARQ technique and the PDU arrival process. The second contribution is commonly referred to as *transmission delay*, and is the time elapsed between the first transmission and the correct reception of the PDU. This term depends on the channel behavior and on the ARQ retransmission technique. The last contribution, referred to as *re-sequencing delay*, is the time spent in the receiver re-sequencing buffer. In more detail, even if the sender transmits packets in order, due to random errors and consequent retransmissions, they can be received out of sequence. In that case, PDUs with higher identifiers must wait in the receiver re-sequencing buffer until all the PDUs with lower identifiers have been correctly received. By considering a tagged PDU, this last term depends on errors experienced by all PDUs sent in the same round-trip in which the tagged one has been transmitted for the first time. The *re-sequencing delay* is non zero when *in-order-delivery* is considered. Another possibility is to configure the link layer in the *out-of-order* delivery mode. In that case PDUs are passed to higher layers without waiting for the correct reception of out of sequence packets and the re-sequencing delay at the link layer is equal to zero. We refer as *delivery delay* to the sum of *transmission* and *re-sequencing* delay.

Several studies have been performed on the delay performance of the SR-ARQ protocol over a wireless channel [64] [39] [65] [15] [91] [92] [110] [50] [73] [7]. In [65] Konheim derived the exact single link layer PDU delay distribution considering a finite round-trip delay and an independent (*iid*) error process. The independent channel has been considered by several other authors [15] [91] [92] [50]. Rosberg and Shacham in [91] derive the distribution of the buffer occupancy and of the re-sequencing delay at the receiver under a heavy traffic assumption[4] to give to the network designer some guidelines on the choice of the buffer capacity at the receiver. Rosberg and Sidi in [92] analyzed the joint distribution of transmitter and receiver buffer occupancies. In [29] the authors investigate the effect of forward/backward channel memory on ARQ error strategies using flow graph analysis; GBN- and SR-ARQ are compared in terms of their throughput efficiency. In [110] Zorzi and Rao studied the adequateness of Two and Three state Markov channel models for predicting delay of ARQ/queueing systems in correlated fading channels. In that work, they show that a three state Markov model, in many cases, can provide very accurate delay predictions, whereas a simple Two state model can lead to good throughput estimates, but it is inadequate for predicting delays. In [39] Fantacci investigates the SR-ARQ performance over a time varying channel [115] [117] [109] [112] deriving the mean packet delay and the mean queue length for both the zero and the finite round trip delay case. Some interesting results on how different error statistics affect ARQ

[4] A new PDU to be sent over the channel is always available at the transmitter ARQ buffer.

performance are reported by Lu and Chang in [73]. In particular, they considered both the k-th Order Markovian Channel Error model and the Gap Error model. In [64] Kim and Krunz account for a time varying channel, a finite round-trip delay and a Markovian traffic source. Here, a mean analysis is developed for all the ARQ delay contributions. The delivery delay distribution in the out-of-order delivery case has been studied in [7], where a general framework is presented to obtain Redundancy Check failure, throughput efficiency and single PDU delivery delay performance.

The main drawback of the analytical approaches presented so far in the literature is their computational complexity that makes their usage very difficult in a Mobile terminal Equipment (ME) due to energy, memory and time constraints. These techniques are time consuming due either to the manipulation of matrices whose size quickly increases with the number of link layer PDUs sent in a full link layer round trip time, m, or to the presence of recursive formulations that are memory and time consuming as m increases. For instance, the complexity of the algorithm proposed in [65] increases exponentially with the round trip time value. Hence, real-time, fast and accurate estimate of the delay statistics experienced both by link layer PDUs and higher layer packets (seen as an aggregate of link layer PDUs) still remains an open issue. These estimates can be useful to obtain *delay-aware* cost functions to be used, for example, when delay sensitive flows are transmitted over the channel to adapt link and/or physical layer settings to the delay requirements imposed by those flows. In real-time services, packets arriving too late are useless; as they are passed to the application layer they are discarded and the system resources consumed for their transmission (and possible retransmissions if a link layer is considered) are wasted. Hence, the control of the maximum delay perceived by application layer packets (that translates in the control of the maximum number of allowed retransmissions at the link layer) is a pivotal point in order to save system resources while achieving a better quality (saved resources can be exploited to send new data instead of useless retransmissions of old packets). The simplest measure for the quality perceived by the final user is the residual packet[5] error probability at the output of the link layer (P_{pkt}). However, when delay constrained flows are considered, this metric is no longer sufficient, because packets whose time deadline has expired (even if correctly received) are discarded and are a source of error as well as corrupted packets. The presence of a link layer is able to reduce the residual error rate P_{pkt}, but the delay experienced by the packets becomes stochastic due to the random nature of the errors affecting the wireless link. In this case, a more accurate estimate of the packet drop rate is needed. Such estimate can be obtained accounting for the delay constraints as follows: $P_{drop} = P_{pkt} + (1 - P_{pkt})\text{Prob}\{\text{delay} > d_{max}\}$, i.e., a packet is considered erroneous either if it is corrupted or if the delay that it experiences during its transmission over the wireless link exceeds some delay constraint (d_{max}), where d_{max} is intended as the maximum delivery delay that (for instance) an IP packet can tolerate when it is transmitted over the wireless link.

The chief aim of this work is to derive analysis to characterize the delivery delay statistics $\text{Prob}\{\text{delay} > d_{max}\}$ for both the *in-order* and the *out-of-order delivery* case. Moreover, from the manipulation of the analytical results, we directly derive simple and accurate heuristics able to describe such statistics with minimal computational effort. The merit of the analysis presented in the next is to go further into the characterization of ARQ delivery delay statistics, investigating also the delivery delay regarding higher layers packets (link layer SDUs), that in our analysis are considered composed of an integer number (K)

[5] Here with the term *packet* we mean the data packet unit assembled at the output of the link layer that, in general, is composed by an aggregate of several link layer PDUs.

of link layer packets. Simple, fast and accurate heuristics for the estimation of SDU statistics are reported considering both in-order and out-of-order delivery of link layer SDUs. These heuristics could be effectively used for example in channel adaptive algorithms as an estimate of the delay perceived by packets belonging to the application layer data flow (*delay utility functions*).

6.2 On the Accuracy of an Independent Channel Model

In this section as a sample scenario, we refer to a UMTS cellular system where W–CDMA is used as the radio interface ([1] [69] [83]) and we evaluate by simulation how the PDU error process at the UMTS link layer (RLC [4]) is characterized in terms of burst length (b) and average PDU error rate (p). Both channel coding (convolutional with rate $1/2$) and interleaving are considered. The simulated scenario is composed of 9 hexagonal cells, where a base station (Node B) is placed at the center of each cell and a given number of users are considered to be able of moving inside the coverage area (in the results reported here we consider a population of 100 users with speed uniformly distributed in $\{0, 7, 49\}$ Km/h). The whole cell structure is wrapped around in order to avoid border effects. As an example scenario we consider that each user is using a downlink dedicated channel, where the traffic source is continuous as can occur when either streaming flows or FTP file transfers are considered. The cell radius is 200 m, a downlink dedicated channel (DCH) is allocated for each user where the minimum and maximum powers are $P_{min}^{downlink} = -15$ dBW and $P_{max}^{downlink} = -5$ dBW, respectively. Path Loss, Shadowing and multi-path fading phenomena are considered as well.

In Figs. 6.1 and 6.2 the average PDU error burst length (b) is reported as a function of the mean PDU error rate (p). Vertical error bars are reported to indicate the confidence interval (95%) of mean burst length measurements, whereas plus and cross symbols are used to plot the 80-th percentile of the burst length b. In the first figure, we consider a link layer with a logical[6] bit rate of 30 Kbps and a large interleaving depth (80 ms), whereas in Fig. 6.2 the RLC bit rate is higher (120 Kbps) and a smaller (40 ms) Time Transmission Interval (TTI, i.e., the interleaving depth) is considered. As a first observation, one can note that at low RLC bit rates the error process tends to be independent (the *iid* case is reported for comparison as a solid bold line in both figures). Hence, the independent PDU error assumption in this case seems to be a good approximation. Instead, as the RLC logical bit rate increases (Fig. 6.2) more PDUs can be sent in each radio frame and the PDU error process is likely affected by longer bursts. It is worth noting that a larger TTI has a beneficial effect on system performance. The effect of a longer TTI is twofold: on the one side the RLC round-trip delay increases (negative effect). On the other side the interleaving is more effective in breaking the error bursts at the bit level (positive effect). Moreover, in the case where $TTI = 80$ ms, a typical value for the RLC round-trip delay is 220 ms. Hence, considering a typical RLC PDU size of 360 bits we have that $m \approx 75$ PDUs can be sent during a full RLC round-trip delay. Referring again to Fig. 6.2, we can note that $b \ll m \approx 75$ and in that case (see Fig. 6.3 where we report the delivery delay distribution obtained for an aggregate of K RLC PDUs (RLC SDU) considering a Two-State Channel Error Model [115] [117] [109] [112] for a fixed PDU error rate $p = 0.1$) the delivery delay statistics of a RLC SDU, after a few round-trip times, is quite close to the one achieved for the independent case (the burst length observed in the simulations is lower than 3 for $p = 0.1$). In

[6]The available bit rate before coding, i.e., the useful data bit rate.

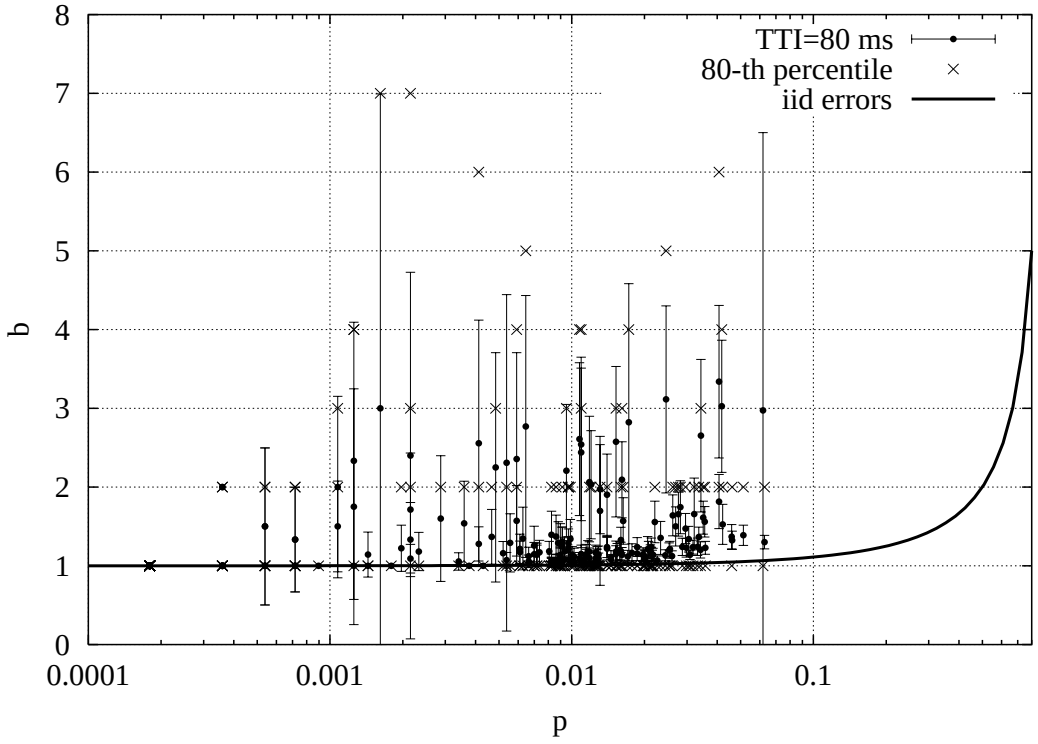


Figure 6.1: DCH Spreading Factor = 128 (RLC bit rate = 30 Kbps), Code Rate = $1/2$, Time Transmission Interval (TTI) = 80 ms, RLC PDU length = 360 bits.

conclusion, the *iid* channel model for a UMTS system can be a good approximation for the link layer error process. This is mainly due to the large interleaving value used at the physical layer and (at high bit rates) to the large value of the round-trip delay ($b \ll m$). Moreover, one can note from Fig. 6.3 that an approximation for the bursty case can be achieved by a rigid rotation of the *iid* curve. This rough approximation, in many cases, should suffice as a heuristic to drive delay adaptive algorithms. For these reasons, in this work we consider an independent link layer packet error process. This model, despite its simplicity, in many cases gives a good approximation for the actual error process as seen at the link layer of a UMTS system. The extension to the correlated case is being studied.

6.3 Analysis over an Independent Channel Model

In general, the link layer of a third generation cellular system (see for instance [4] for the UMTS case) can operate either in the Unacknowledged Mode (UM) or in the Acknowledged Mode (AM). Moreover in AM, higher layer packets (Service Data Units or SDUs) can either be released *in-order* or *out-of-order* to upper layers. In the *in-order delivery* case, a given SDU, say SDU number i is released to higher levels only after all SDUs with lower identifier have been correctly received or discarded. A SDU is said to be discarded when one or more PDUs composing it reached the maximum number of retransmission attempts without being correctly received. In this last case the SDU may be passed to the higher layer and is

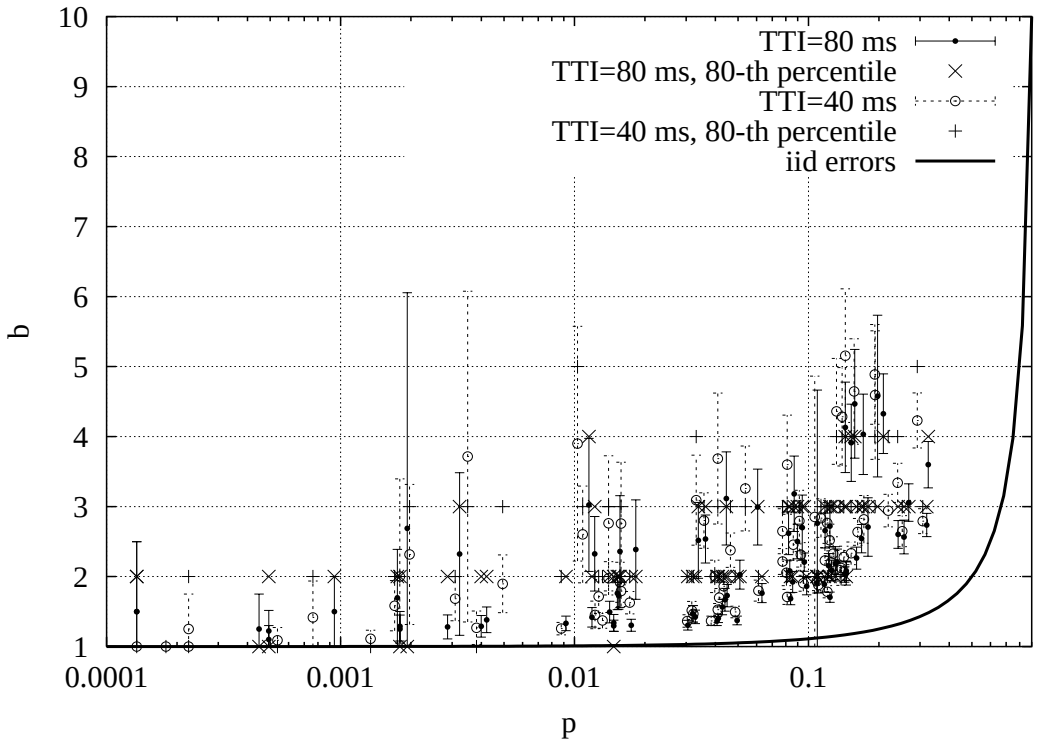


Figure 6.2: DCH Spreading Factor = 32 (RLC bit rate = 120 Kbps), Code Rate = $1/2$, Time Transmission Interval (TTI) $\in \{40, 80\}$ ms RLC PDU length = 360 bits.

marked as erroneous. In the *in-order-delivery* case we refer to the PDU delivery delay as the time elapsed between the instant where the PDU is transmitted for the first time over the channel and the instant where it can be released in-order to upper levels, i.e., no outstanding PDUs with lower identifier are present. The SDU delivery delay is defined as the time elapsed between the instant where the first PDU composing the SDU is transmitted for the first time over the channel and the instant where all the PDUs composing it have been received correctly in-order. Instead, when the RLC is configured to support the *out-of-order delivery* of SDUs, a given SDU, say SDU i, may be passed to higher layers when all the PDUs composing it have been correctly received (or discarded), regardless of the transmission status (correctly received, delivered, in flight) of other SDUs. In this case the SDU delivery delay is equal to the SDU transmission delay and corresponds to the time elapsed between the instant in which the first PDU composing a SDU is transmitted for the first time over the channel and the instant where all the PDUs composing the SDU have been correctly received. In this case the SDU can be passed to upper levels regardless of the status of previously transmitted SDUs. Among other settings, another important parameter is the maximum number of retransmission attempts permitted for each link layer PDU (referred to as L in this work). In the next section, we first investigate the delivery delay statistics considering in-order-delivery of link layer SDUs and limited retransmission attempts, whereas in Section 6.3.6 the out-of-order delivery of link layer SDUs is considered.

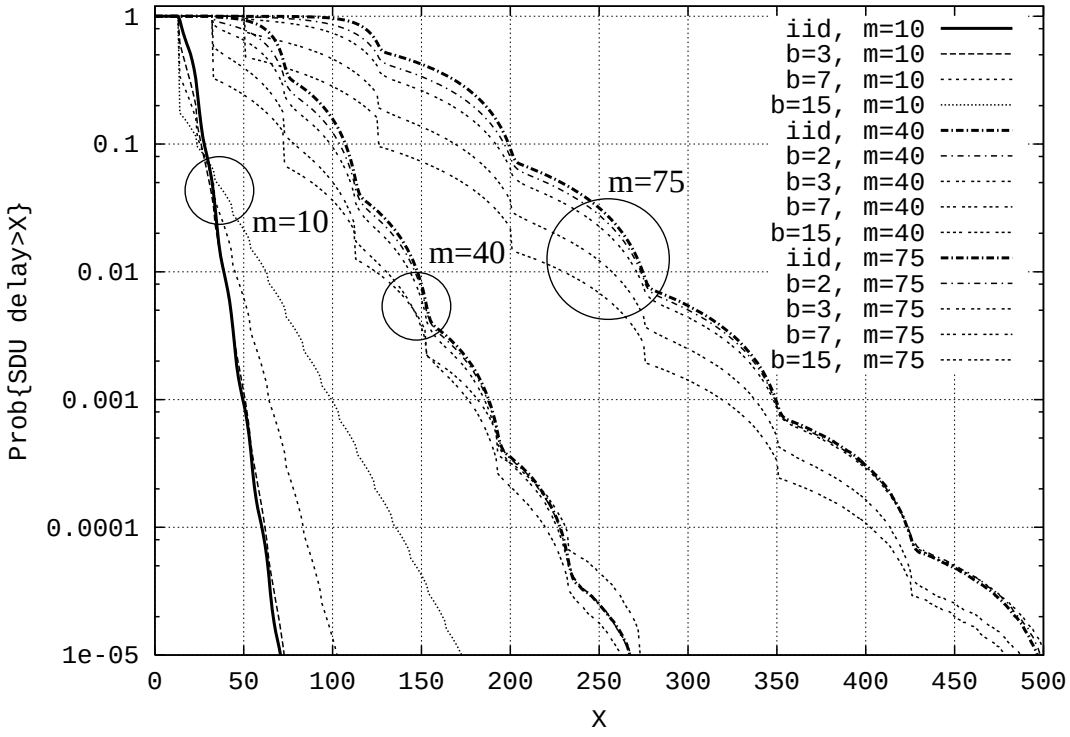


Figure 6.3: Link layer SDU Complementary cumulative delivery delay distribution as a function of the burst length b considering $m \in \{10, 40, 75\}$, $p = 0.1$ and $K = 14$.

In the next the following quantities are computed

- In Section 6.3.1 we compute the single PDU delivery delay distribution in the in-order delivery case, $P_d[t]$.

- The transmission delay statistics regarding a link layer SDU $P_{tx}[K, t]$ is derived in Section 6.3.3. K is the (integer) number of PDUs composing a link layer SDU.

- In Section 6.3.4, the link layer SDU delivery delay statistics in the in-order delivery case $P_d[K, t]$ is reported. From $P_d[K, t]$ the SDU complementary cumulative delivery delay distribution ($ccdf[K, t]$) is obtained.

- Based on the analysis, an accurate approximation for the complementary cumulative distribution in the in-order delivery case $ccdf[K, t]$ is derived in Section 6.3.5.

- Finally, in Section 6.3.6 a very accurate and simple approximation for the SDU delivery delay in the out-of-order delivery case is reported.

6.3.1 Single PDU Delivery Delay Statistics ($P_d[t]$) for In-Order Delivery of RLC SDUs

The problem to be solved is to find the delivery delay distribution of an aggregate of K link layer PDUs[7]. Following the procedure discussed (and justified) later, in Section 6.3.4, this statistics can be computed by means of a discrete-time convolution between two statistics: the single PDU delivery delay distribution ($P_d[t]$) and the K PDUs transmission time distribution ($P_{tx}[K,t]$). $P_d[t]$ gives the probability that a given single PDU is delivered in-order in exactly t slots[8], whereas $P_{tx}[K,t]$ is used to track the probability that K new PDUs are transmitted over the channel in t slots. Both $P_d[t]$ and $P_{tx}[K,t]$ depend on L, RTT and p, i.e., the maximum number of retransmissions allowed for each PDU, the round trip time and the PDU error probability, respectively.

In this section, we compute the delivery delay regarding a single PDU, i.e., the time needed by a PDU transmitted at time t to be correctly delivered at the receiver side. We consider in-order delivery, i.e., the PDU is delivered in-order only if all PDUs with lower identifier transmitted before it have been correctly received or have been transmitted $L+1$ times without any success. A given PDU can be released to higher layers only when all PDUs with lower identifier have been released. In the following analysis we say that a PDU is *resolved* in the slot where it is either correctly transmitted or discarded (due to the reaching of the maximum number of allowed transmissions, i.e., transmitted $L+1$ times without any success). We label a PDU as *unresolved* if it has been transmitted fewer than $L+1$ times without success.

We consider a pair of nodes that are exchanging packets through a noisy wireless link where the link layer is configured in the AM mode. The time is slotted, and the slot time corresponds to the single PDU transmission time. Moreover PDUs are transmitted continuously, i.e., there are no empty slots (*Heavy traffic assumption* see [91] [64]). This assumption can be justified in the case where a FTP file transfer or a video/audio streaming flow is considered. In these cases, in fact data are likely sent back-to-back without idle times in order to respect the timing imposed by the video/audio data flow and to avoid underutilization of the assigned resources. These are very important classes of service for which we expect such kind of behavior. Nevertheless, in the cases where this is not true the heavy traffic assumption, at least for the in-order-delivery-delay, is surely a worst case delay analysis. This can be explained as follows: in the in-order-delivery case a tagged packet has to wait for the correct reception of all (and only) the other packets sent in the same round in which it has been transmitted for the first time (see later in this section for a justification of this fact). In the heavy traffic case, we always have that the number of packets sent in a round is at its maximum value (i.e., the number of packets that can be transmitted in one round trip). Hence, we also have the highest probability that at least one packet is corrupted in that round, i.e., that the tagged packet will have to wait for out-of-order packets.

Moreover, let m be the (integer) number of PDUs transmitted in a full RLC RTT. We consider that an acknowledgment message indicating the delivery status of a given PDU transmitted in the generic slot t is available at the sender at the beginning of slot $t+m$. Now consider a tagged PDU transmitted in slot t. Moreover, consider all PDUs transmitted in slot $\{t-m+1, t-m+2, \ldots, t-1\}$, i.e., the last $m-1$ PDUs transmitted before the tagged one (we say that these PDUs are the packets transmitted in the same window in which the tagged PDU has been transmitted for the first time, in the analysis we refer to these PDUs as *blocking PDUs*). Then, we refer as *starting window* to the set of the tagged PDU plus

[7] Throughout this study we consider that a SDU is composed by an integer number (K) of link layer PDUs.

[8] The slot duration corresponds to the packet transmission time.

these $m-1$ packets. Note that these $m-1$ PDUs are the only ones that the tagged PDU must eventually wait for after its correct delivery, i.e., the only PDUs with lower identifier. In fact, a PDU sent for the first time in window position i is re-sent in the same window position until success or until $L+1$ transmission failures. In any case, only from this point on can a new PDU be transmitted over the channel using the same window position occupied by such PDU. Given this transmission/retransmission behavior and that PDU identifiers are incremented sequentially in increasing order, it is clear that each PDU must eventually wait for the resolution (intended either as correct reception or the $(L+1)$-st transmission failure) only of the PDUs contained in its starting window. This simple but effective model to characterize ARQ has been widely used in the literature so far (e.g., [64] is the most recent work using it). Here, we use it to characterize the delay performance of a 3G link layer scheme. Note that 3G retransmission schemes have been enriched by new timers and features with respect to traditional ARQ solutions. These additional features have been necessary to ensure a fault-tolerant, fully-programmable and deadlock free scheme that, however, basically remains a Selective Repeat algorithm. Moreover, our idealized scheme can be seen as a best case for the delay performance achieved using a 3G compliant link layer because in our model packets are always acknowledged in the shortest possible time ($m-1$ slots), so the time spent between their previous transmission and the following retransmission is also minimized. As a consequence, the delivery delay for a given link layer packet is the shortest as well. The only configuration where this is not true is when the link layer is programmed to transmit multiple copies of the same packet in the same round trip. However, this is an energy-, resource-inefficient and particular case designed to decrease delivery delay at the cost of a degraded throughput performance. Nevertheless, our results are still valid as a best case estimate that is useful to obtain delay evaluations (that is the aim of our study). Extensions of the model to better describe a 3G link layer behavior are being studied. However, it is worth noting that the insights derived here are general and still hold in a more realistic scenario.

In the following we compute the delivery delay statistics for the tagged PDU. With the term PDU delivery delay we indicate the number of slots elapsed between the instant in which the tagged packet is transmitted for the first time over the channel and the slot in which this PDU is finally released in-order to higher layers, i.e., the slot where both the tagged PDU and all the $m-1$ blocking PDUs have been resolved. Moreover, we compute such statistics at the sender side, i.e., we say that the tagged PDU can be released in-order in the slot where the last unresolved PDU (comprising the tagged packet itself) is transmitted and resolved at the sender side. Note that this statistics differs with respect to the statistics at the receiver side by a constant term (the sum of the single path propagation delay and physical layer processing, D_s). In order to proceed with the analysis, let us subdivide the time in rounds of m slots. In particular, we refer to round 0 as the one including the PDUs transmitted in slots $\{t-m+1, t-m+2, \ldots, t\}$, where t is the slot in which the tagged PDU has been transmitted for the first time over the channel. Without loss of generality, in the following we will assume that the tagged packet is transmitted in position m of round 0, i.e., $t=m$.

To perform an exact analysis of the PDU delay statistics, it is necessary to keep track of the number of retransmission attempts performed for each PDU transmitted in the starting window and of its delivery status (resolved or unresolved). Hence, to describe the probabilistic evolution of the tagged PDU delivery process, we would build a chain characterized by at least $(2\times(L+1))^m$ states. However, for large values

of m, this would lead to such a large state space to make the problem very difficult to solve. To avoid this, in the following analysis, we limit the amount of information to be tracked. In particular, at any time, we only track the full state (transmission attempts and delivery status) of the tagged PDU and the number of unresolved blocking packets. The resulting analysis is not exact but will be shown to be very accurate.

Moreover, we use the probability $P[\text{error}] = p$ to decide whether a PDU in a given slot is erroneous or not. Note that, given that a PDU is transmitted in error in a slot, the probability that it has reached its maximum number of transmission attempts is equal to the probability that the PDU has been already transmitted for L times without any success before the considered slot ($P[\text{L prev. errors}] = p^L$). So, the probability that a PDU is discarded in a slot, i.e., the PDU has been transmitted in error for $L+1$ consecutive times, is equal to $P[\text{discard}] = P[\text{error}]P[\text{L prev. errors}] = p^{L+1}$. Hence, a PDU in a given slot is transmitted successfully with probability $q = 1-p$, is discarded with probability $p_d = p^{L+1}$, whereas it remains unresolved without being discarded with probability $1-q-p_d = p(1-p^L)$.

Let $u \in \{0,1\}$ be a boolean variable indicating the tagged PDU resolving state, i.e., whenever the tagged packet has been resolved or not. In particular, we use the notation $u=0$ and $u=1$ for resolution and no-resolution, respectively. Using this notation, we define $\psi[n,i,u]$ as the joint probability that, at the end of round i, $i \geq 0$ there are n unresolved PDUs among positions $(i-1)m+1$ through $(i-1)m+m-1$ and the state of the tagged PDU[9] is u. In particular, the probability to have, at the end of round 0, n blocking PDUs and the tagged packet state equal to u is given by

$$\psi[n,0,u] = \binom{m-1}{m-n-1} \sum_{k=0}^{m-n-1} \left[\binom{m-n-1}{k} q^k p_d^{m-n-1-k} (1-q-p_d)^n p^u q^{1-u} \right] \tag{6.1}$$

where $n \in \{0,1,...,m-1\}$, $u \in \{0,1\}$, $q = 1-p$, $p_d = p^{L+1}$ and p is the PDU error probability. In the equation above, we compute the probability that n out of the $m-1$ PDUs transmitted (in round 0) are unsuccessful, but they have been transmitted less than $L+1$ times (term $(1-q-p_d)^n$, in this case they must be retransmitted in the next round and in the same position), that the remaining $m-1-n$ PDUs are resolved either due to their correct transmission (term q^k) or to their discard (term $p_d^{m-n-1-k}$), and that the tagged PDU is in state u (where $u=0$ means correctly transmitted).

The function $\psi[n,i,u]$, for $i>0$, is evaluated recursively in the following way

$$\begin{aligned} \psi[n,i,0] &= \sum_{k=n}^{m-1} \Big(\psi[k,i-1,1]q + \psi[k,i-1,0] \Big) \varphi(k,n) \\ \psi[n,i,1] &= \sum_{k=n}^{m-1} \psi[k,i-1,1] p \, \varphi(k,n) \end{aligned} \tag{6.2}$$

where $\varphi(k,n)$ is the probability to resolve (correctly receive or discard) $k-n$ PDUs over k in any order, and is computed as follows

$$\varphi(k,n) = \sum_{r=0}^{k-n} \binom{k}{n} \binom{k-n}{r} q^r p_d^{k-n-r} (1-q-p_d)^n \tag{6.3}$$

Moreover, the probability to have, at the end of round i, n unresolved packets among positions $(i-1)m+1$ through $(i-1)m+m-1$ and the tagged PDU in state 0 ($\psi[n,i,0]$) is computed as the probability to

[9] Following our definition of round this PDU is always transmitted in the last slot of each round, i.e., in this case in position im.

have $k \in \{n, \ldots, m-1\}$ unresolved blocking PDUs (term $\psi[k, i-1, 1]q + \psi[k, i-1, 0]$) in the previous round $(i-1)$ and that exactly $k-n$ of these PDUs are resolved in round i (term $\varphi(k, n)$). The term $\psi[k, i-1, 1]$ is multiplied by q to account for the case where the tagged packet is resolved at the end of round i. A similar reasoning is made in the computation of $\psi[n, i, 1]$. In this case the term $\psi[k, i-1, 0]$ is absent because a resolved PDU can never be marked as unresolved. In this case the term $\psi[k, i-1, 1]$ is multiplied by p to reflect that the tagged packet is still unresolved at the end of round i.

The delivery delay statistics, indicated as $P_d[t = \xi m + u]$, $\xi \geq 0$, $u \in \{0, 1, \ldots, m-1\}$, i.e., the probability to have a delivery delay for the single PDU equal to t slots is approximated by means of the ψ function as follows

$$P_d[t = \xi m + \eta] = \begin{cases} \psi[0,0,0] & t = 0 \\ \sum_{k=0}^{m-1} \psi[k, \xi-1, 1] q \, \mathcal{C}(k) & \eta = 0, t \in \mathcal{D} \\ \sum_{k=1}^{\eta} \psi[k, \xi, 0] \, \mathcal{C}(k) \, P_\eta & \eta \neq 0, t \in \mathcal{D} \\ 0 & \text{elsewhere} \end{cases} \tag{6.4}$$

where $\mathcal{D} = \{1, 2, \ldots, Lm\}$, $\eta \in \{0, 1, \ldots, m-1\}$ and $\xi \geq 0$. $\mathcal{C}(k)$ is the probability to resolve k PDUs in any order and is given by

$$\mathcal{C}(k) = \sum_{i=0}^{k} \binom{k}{i} q^i p_d^{k-i} \tag{6.5}$$

P_η is the probability that the last of the k PDUs in error is transmitted in position η of round ξ, where $\eta \in \{1, 2, \ldots, m-1\}$. P_η is computed as follows

$$P_\eta = \frac{\binom{\eta-1}{k-1}}{\binom{m-1}{k}} \tag{6.6}$$

In Eq. (6.4), the way $P_d[t]$ is computed depends on the value of η. In more detail, if the tagged packet is released in-order at a given time $\xi m + (\eta = 0)$, this means that it is released in the slot in which it resolved[10]. Given that, all blocking packets are necessarily resolved up to and including the end of round ξ. When $\eta \neq 0$, instead, at the time in which the tagged packet is resolved, there is at least one blocking packet to be resolved. In this case, $P_d[t]$ is evaluated summing over $1 \leq k \leq \eta$ the probability ($\psi[k, \xi, 0]$) that k blocking packets are unresolved at the end of round ξ and that these packets are all resolved (term $\mathcal{C}(k)$) in the current round (round $\xi + 1$). Moreover, given that PDU errors are described by means of an *iid* process, the position of the last resolved blocking packet is distributed in a uniform manner [91]. We then evaluate the probability of resolving the last blocking PDU in position η using Eq. (6.6). $P_d[t]$ is reported in Fig. 6.4 and compared against the one computed by simulation for $m = 40$, $p = 0.1$ and $L = 3$.

6.3.2 Statistical Properties of $P_d[t]$

In this section we investigate the main characteristics of the function $P_d[t]$. In what follows, for the sake of simplicity, we refer to the case where L is unlimited. The obtained results hold also in the limited

[10] Remember that the tagged PDU is always sent at the end of each round.

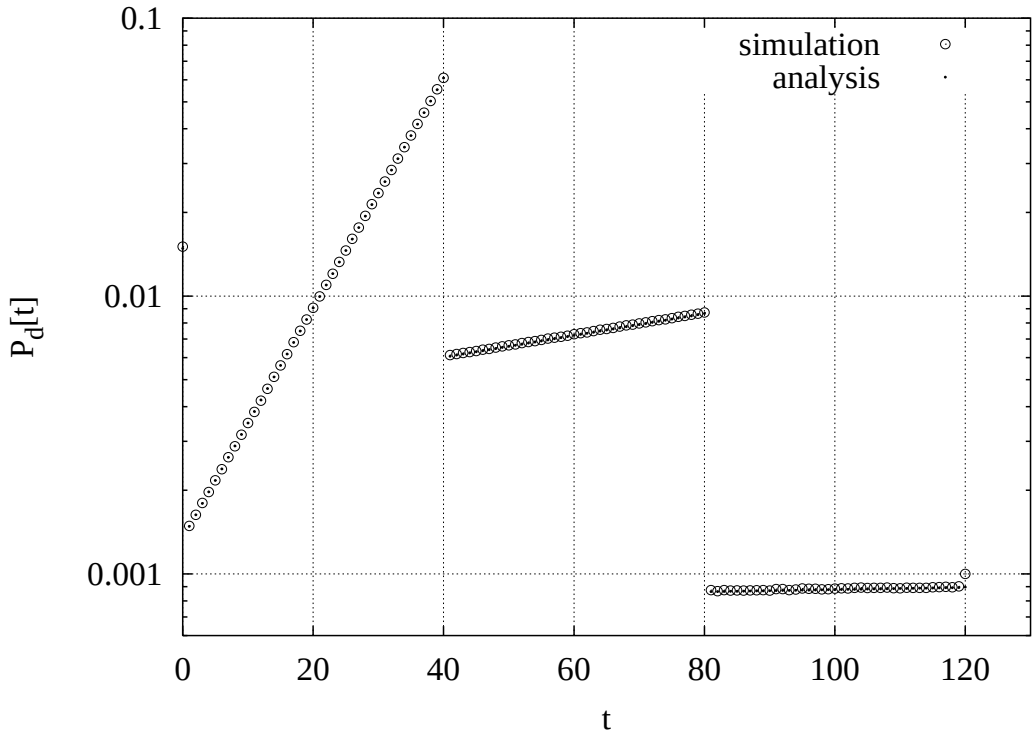


Figure 6.4: Single PDU delay statistics, $P_d[t]$, comparison between analysis and simulation for $m = 40$, $p = 0.1$ and $L = 3$.

case.

As above, let round 0 be the one where the tagged packet is transmitted for the first time. Moreover, express the delivery delay as $d = \xi m + \eta$, $\xi \geq 0$, $\eta \in \{0, 1, \ldots, m-1\}$. The probability that an erroneous PDU in round 0 is transmitted correctly in a round up to and including round r is given by $P_c[r] = 1 - p^r$. The probability that the tagged packet is resolved in position m of round s, $s > 0$, i.e., at time $d_1 = sm$ is given by

$$P_d[d_1 = sm] = \sum_{k=0}^{m-1} \psi[k, 0, 1] P_c[s]^k p^{s-1}(1-p) \tag{6.7}$$

where we compute the probability to have k blocking packets, that all of them are resolved before slot d_1 (term $P_c[s]^k$) and that the tagged packet is resolved exactly in slot d_1 (term $p^{s-1}(1-p)$). Now, let us compute the probability to resolve the tagged PDU at time $d_2 = sm + 1$, i.e., in the first slot of the following round

$$P_d[d_2 = sm + 1] = \sum_{k=0}^{m-1} \psi[k, 0, 1] P_c[s]^k p^{s}(1-p) \tag{6.8}$$

in this case $\psi[k, 0, u]$ must be viewed as the probability that the PDU occupying position 1 in round 0 was in error ($u = 1$) and that exactly k packets out of the remaining $m - 1$ are erroneous, $P_c[s]^k$ is the probability that PDU in slot $\{2, 3, \ldots, m\}$ are transmitted correctly up to and including round s and $p^s(1-p)$ represents the probability that the PDU transmitted in the first position of round 0 is resolved

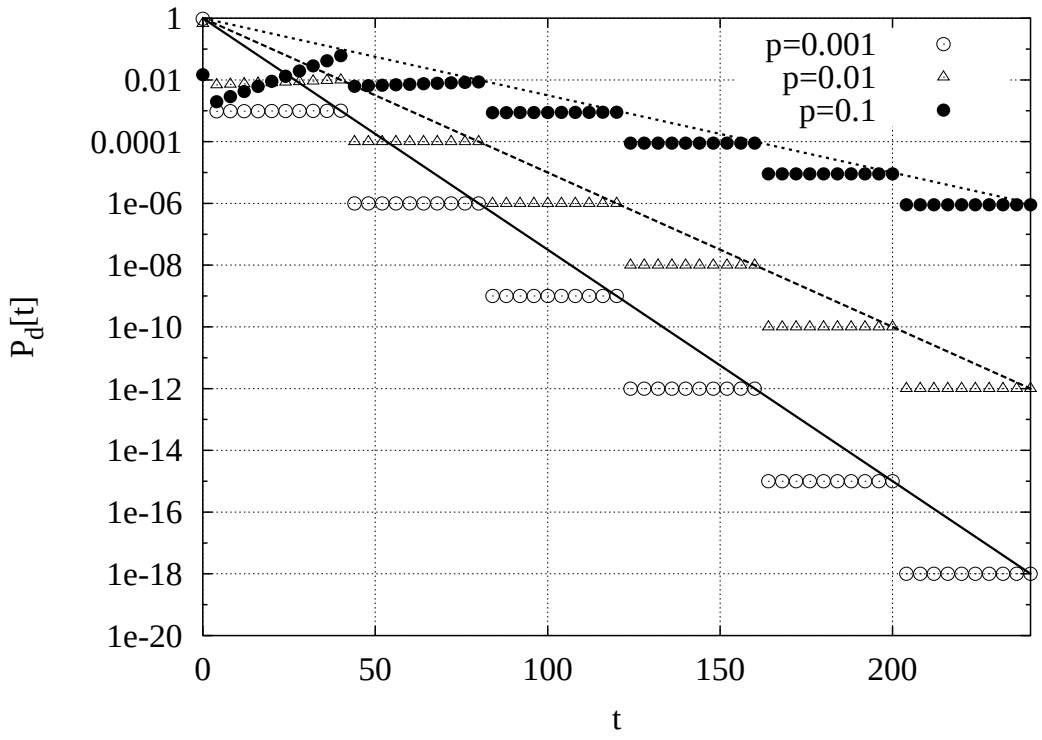


Figure 6.5: Asymptotic behavior of $P_d[t]$ obtained by analysis, $m = 40$, $p = 0.1$ and $L = 7$.

in slot d_2. Observing Eqs. (6.7) and (6.8), it is clear that the following relation holds

$$P_d[sm + 1] = P_d[sm]p \qquad s \geq 0 \tag{6.9}$$

This is a general result that will be very useful in order to find very fast and accurate approximations of such statistics. Moreover, using again these equations, we can find the difference between the probability to resolve the tagged packet in the first slot and at the end of a given round ($s > 0$)

$$P_d[(s-1)m + 1] - P_d[sm] = \sum_{k=0}^{m-1} \psi[k, 0, 1]p^{s-1}(1-p)\Delta[s, k] \tag{6.10}$$

where $\Delta[s, k] = P_c[s]^k - P_c[s-1]^k$. At this point, by rewriting $P_c[s]^k$ as $(1-p^s)^k = \sum_{i=0}^{k} \binom{k}{i}(-p^s)^i$ it is easy to verify that $\Delta[s, k]$ tends to 0 as s increases. In practice this convergence is very fast and, as a consequence[11], $P_d[t]$, after a short transient phase, becomes almost constant inside each round (see Figs. 6.4 and 6.5). These considerations allow us to assert that the asymptotic behavior of the delivery delay statistics is described by the following equation

$$y[t] = p^{\frac{t-t_0}{m}} \tag{6.11}$$

[11]This is ensured by the fact that $P_d[t]$ is non-decreasing inside each round, i.e., $P_d[sm + 1] \leq P_d[sm + 2] \leq \cdots \leq P_d[sm + m]$, $s \geq 0$. This can be verifi ed from Eq. (6.4).

In more detail, Eq. (6.9) and the fact that, after a first transient phase, all points in a round have the same value allow us to conclude that, in the logarithmic domain, and after a transient phase, the values of the delivery delay probability in the first (last) slot of each round are placed over a straight line. This observation allows us to find, after a short transient phase, the exact behavior of $P_d[t]$ simply using Eq. (6.11). Given m and p, the only parameter that needs to be computed is the value of t_0 that can be found by solving the equation $P_d[sm] = y[sm]$ for s sufficiently large[12].

In Fig. 6.5 we report $P_d[t]$ and the straight lines indicating the asymptotic behavior by varying p considering $m = 40$ and $L = 7$.

6.3.3 Link Layer SDU Transmission Delay Statistics

In this section we compute the transmission delay statistics of an aggregate of K PDUs (link layer SDU), i.e., the statistics of the time elapsed between the instant in which the first PDU composing a SDU is transmitted over the channel and the slot in which the K-th PDU (the last composing the SDU) is transmitted for the first time over the channel. First of all, note that a new PDU is sent over the channel in the generic slot t if and only if a PDU had been resolved m slots earlier (in slot $t - m$). In fact, each PDU at the time of its first transmission occupies a given position in its starting window. Such PDU is then transmitted every m slots in the same window position until success or until $L + 1$ transmission failures have occurred. In other words, a given window position is occupied for transmission by a single PDU up to and including the slot where it is finally resolved. Only from this point on, can the slot be assigned to a new PDU[13]. Hence, the number of PDUs resolved in a generic time interval, say $[t, t + N]$, is exactly equal to the number of new PDUs transmitted in the time interval $[t + m, t + m + N]$. In particular, the *new transmission process* is simply the *enabling process* deterministically right shifted by m slots.

In the following analysis, we adopt the same assumptions made in the previous section, i.e., we do not track the exact state of each PDU but we decide whenever a PDU is successfully transmitted, still unresolved or discarded using the probabilities q, $1 - q - p_d$ and p_d, respectively.

In this section, we refer to the *SDU transmission time* as the number of slots elapsed between the time in which the first PDU composing the SDU is transmitted for the first time and the slot where the last (the K-th) PDU is transmitted for the first time over the channel. Hence, the probability that a full SDU is transmitted in exactly t slots, say in the time interval $[i, i + t]$, $i \geq 0$ corresponds to the probability that $K - 1$ new PDUs are taken from the input queue and transmitted for the first time over the channel in slot $i + 1$ through $i + t$ given[14] that the first PDU composing the SDU packet is transmitted for the first time in slot i. We refer to this probability as $P_{tx}[K, t]$. Moreover, recall that a packet resolution in a given slot implies the transmission of a new packet m slots apart, this probability is the same as that of resolving $K - 1$ packets in $t - 1$ slots, say $[i - m + 1, i + t - m]$, where the last packet must be resolved in the $(t - 1)$-st slot (slot $i + t - m$) given that we have the first resolution in slot $i - m$.

Hence, a good approximation of the transmission delay statistics, $P_{tx}[K, t]$, can be obtained using the

[12] Note that for p values up to and including 0.1, $s = 1$ suffices to derive a very accurate approximation for t_0.

[13] We say, in this case, that the resolution of a given PDU *enables* the transmission of a new PDU over the channel m slots apart, i.e., in the same window position of the next round. In the sequel this process will be referred to as *enabling process*.

[14] Where the last (K-th) PDU must be transmitted for the first time in slot $i+t$, whereas the remaining $K-2$ PDUs (excluding the first and the K-th) can be transmitted in any order in slots $i + 1$ through $i + t - 1$.

following formula

$$P_{tx}[K,t] = \begin{cases} \left(q+p_d\right) \sum_{r=0}^{K-2} \binom{t-2}{r} \binom{t-2-r}{K-2-r} p_d^r \, q^{K-2-r} (1-q-p_d)^{t-K} & K \le t \le t_{max} \\ 0 & \text{elsewhere} \end{cases} \tag{6.12}$$

where $t_{max} = m(L+1)+K-1$. In the equation above, we compute the probability to have $K-2$ PDU resolutions over $t-2$ slots in any order (terms p_d^r and q^{K-2-r}) and that the PDU transmitted in the last (the t-th) slot is also resolved (term $q+p_d$). Note that the statistics above is conditioned on having the resolution of the first PDU (of K) in the first slot. Moreover, $P_{tx}[K,t]$ is zero for $t < K$ and $t > t_{max}$. The first case is trivial, whereas the second case is justified as follows. In the worst case, the first PDU is transmitted for the first time in a window where all the remaining $(m-1)$ PDUs are also transmitted for the first time. Moreover, in the very worst case all these PDUs are retransmitted repeatedly for L times in each of the following L rounds. Only from this point on, can the remaining $K-1$ PDUs be transmitted in position 1 through $K-1$. Note that this reasoning is valid as long as $K-1 \le m$, but it can easily be extended to the more general case in which $K-1 > m$. In this work we focus on the case where $K-1 \le m$ because it is a very common situation under typical logical channel settings in UMTS. For instance, considering a link layer logical bit rate of $B = 120$ Kbps, a typical PDU size of 360 bits, and a link layer round trip time of $RTT = 220$ ms, we have that about 75 PDUs are transmitted in a full round-trip ($m \approx 75$). In addition, a SDU of 1 Kbyte is transmitted in 23 PDUs. Moreover, for a fixed RTT, m increases with B. However, our analysis, with minor changes (it is sufficient to change t_{max}), still holds also in the case where $K-1 > m$.

In Fig. 6.6, we compare the results obtained from Eq. (6.12) with the ones achieved by simulation. By observing this figure we can conclude that the statistics derived using Eq. (6.12) is reasonably accurate for small L values, and it tends very quickly to the exact statistics as L increases.

In the next, we compute the asymptotic behavior of $P_{tx}[K,t]$. In particular, we are interested in the computation of the ratio $P_{tx}[K,t+1]/P_{tx}[K,t]$ as a function of t. Observing that $\binom{i+1}{j} = \frac{i+1}{i+1-j}\binom{i}{j}$, $i > j$, we can rewrite $P_{tx}[K,t+1]$ as

$$P_{tx}[K,t+1] = (1-q-p_d)\frac{t-1}{t+1-K} P_{tx}[K,t] \tag{6.13}$$

that holds for any t such that $K \le t < t_{max}$. From Eq. (6.13) it is straightforward to note that $P_{tx}[K,t+1]/P_{tx}[K,t]$ for large values of t tends to the constant value $(1-q-p_d)$. In this case $P_{tx}[K,t]$, can be very well approximated by a straight line (in the logarithmic domain). These results will be used later, in Section 6.3.5, to derive fast and accurate approximations of SDU delay statistics.

6.3.4 SDU Delivery Delay Statistics for In-Order Delivery Case

In this section we derive the delivery delay statistics of a link layer SDU, i.e., the number of slots elapsed between the instant where the first PDU composing that SDU is transmitted for the first time over the channel and the instant in which the full SDU is received and can be passed in-order to higher layers. To find such statistics, we subdivide the SDU delivery time in two contributions, where the first one is given by the time elapsed between the first transmission of the first PDU composing the SDU and the slot

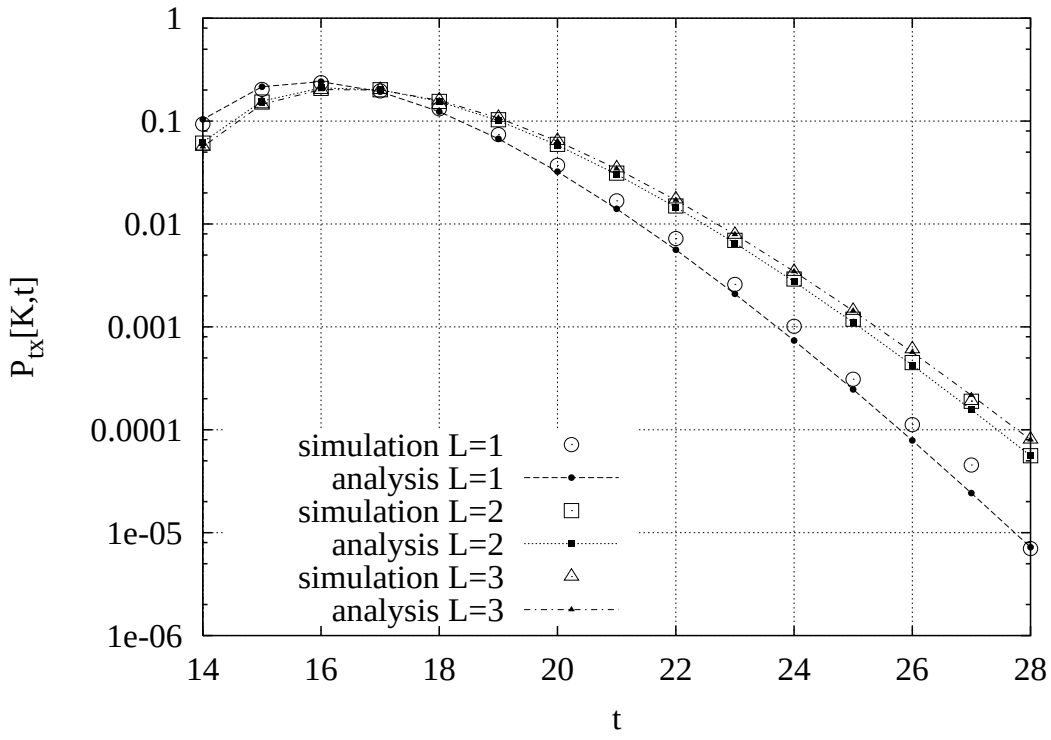


Figure 6.6: Transmission delay statistics, $P_{tx}[K,t]$, comparison between analysis and simulation for $m = 40$, $K = 14$ and $p = 0.1$.

where the last (K-th) PDU is transmitted over the channel. To track this delay we use the transmission statistics computed in Eq. (6.12). From this point on, we use the single PDU delivery delay statistics in Eq. (6.4) to track the time needed for PDU K to be delivered in-order. Note that the slot where this last PDU can be delivered in-order to higher layers is the same slot where the whole SDU can be delivered in-order. This is justified by the fact that the PDUs that eventually block the delivery of the K-th packet at the instant of its first transmission are only the ones with an identifier lower than the one assigned to that PDU. In conclusion, by using a discrete-time convolution product of these two contributions, we are able to obtain the delay statistics $P_d[K,t]$ of a full SDU

$$P_d[K,t] = \begin{cases} 0 & t < K \\ \sum_{r=K}^{t} P_{tx}[K,r] P_d[t-r] & t \geq K \end{cases} \tag{6.14}$$

In Fig. 6.7 we compare the SDU delivery delay statistics obtained by simulation against the one obtained analytically using Eq. (6.14). Here, we note that simulation points can be estimated only until error probabilities of the order of $p \approx 10^{-5}$. For lower values of p the statistics can not be obtained due to the rare occurrences of the corresponding events. In Fig. 6.8, instead, we report simulation and analytical results by varying the number of PDUs composing one SDU (K). As can be observed from these figures, analysis and simulation points are in very good agreement. Note also that $P_d[K,t]$ is zero for $t > (L+1)m + K - 1$, i.e., where $P_{tx}[K,t]$ is zero.

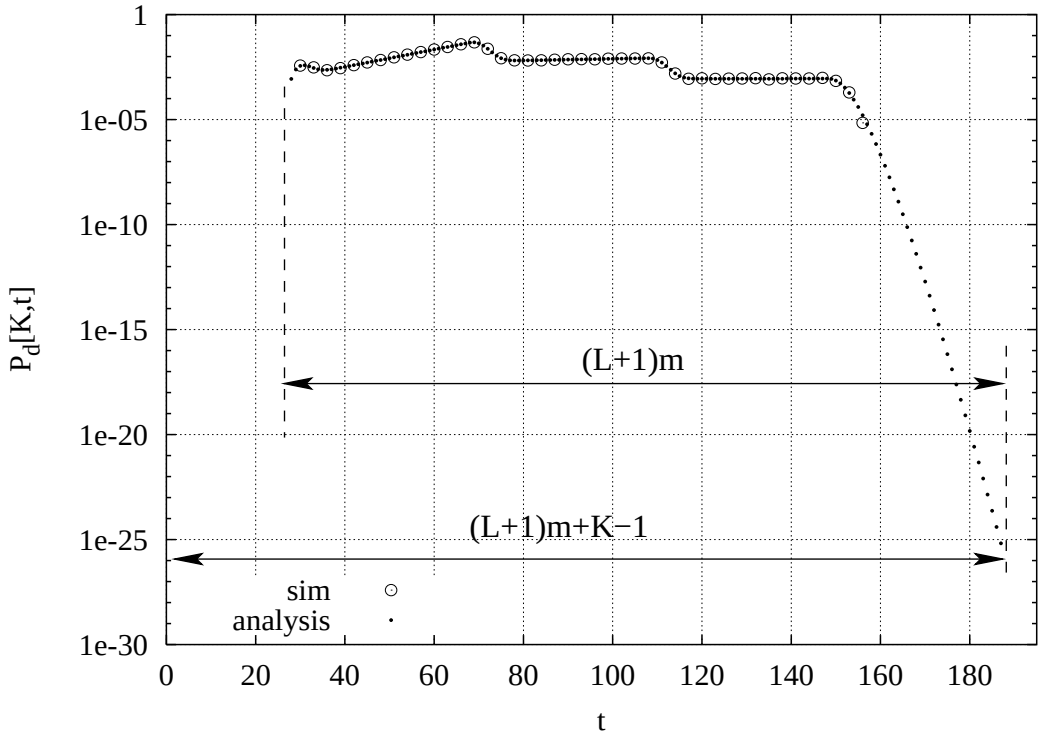


Figure 6.7: $P_d[K,t]$ for $m = 40$, $K = 28$, $L = 3$ and $p = 0.1$.

The most useful quantity is the complementary cumulative distribution function (*ccdf*) of $P_d[K,t]$, i.e., the probability that the SDU delivery time exceeds a given number of slots, formally

$$ccdf[K,t] = 1 - \sum_{x=0}^{t} P_d[K,x] \tag{6.15}$$

Moreover, recall that our analysis is not inclusive of the single path propagation delay (D_s) regarding the delivery of the K-th PDU (Section 6.3.1). The complementary delivery delay statistics comprehensive of D_s is computed as follows

$$\text{Prob}\{\text{SDU delay} > t\} = \begin{cases} 0 & t < D_s \\ ccdf[K, t - D_s] & t \geq D_s \end{cases} \tag{6.16}$$

6.3.5 Accurate Approximation of the SDU Complementary Cumulative Distribution Function, $ccdf[K,t]$

We believe that the *ccdf* statistics would be very useful if available at any user terminal. It can be used, for example to predict in advance performance metrics, or used directly in some kind of cost function.

The main drawback of the analysis presented in the previous section is that it is computationally too complex to be effectively used in a mobile terminal, since the involved computations would be time and energy consuming. Moreover, unless the time spent for the computation of the statistics is small with

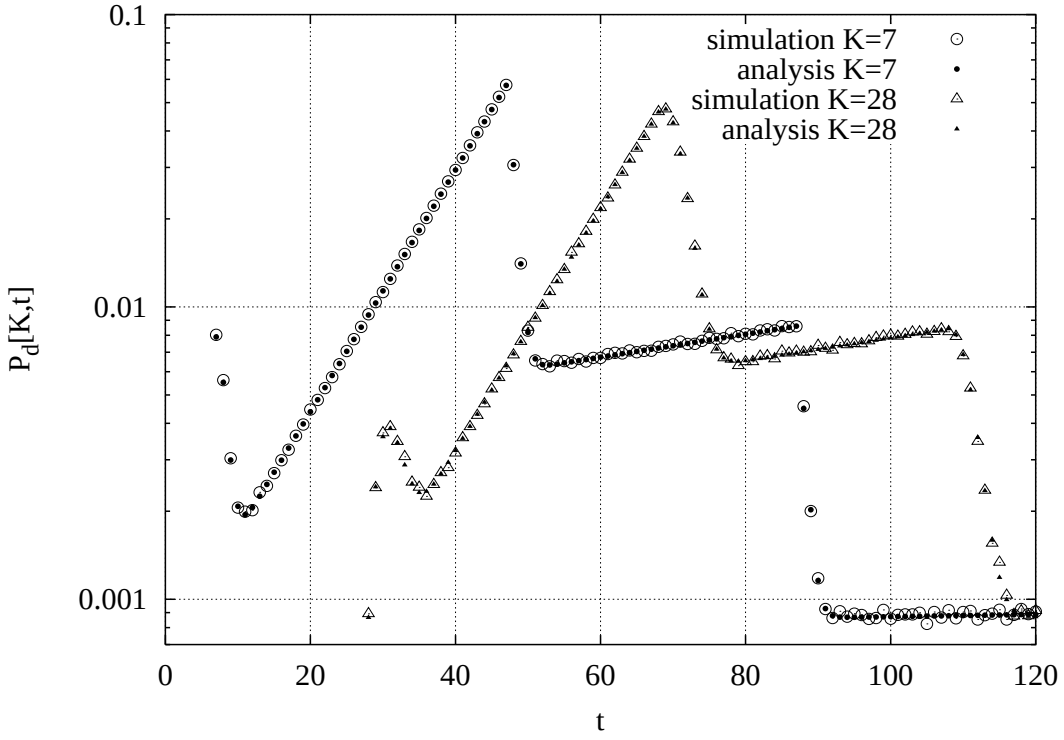


Figure 6.8: $P_d[K,t]$, comparison between $K = 7$ and $K = 28$ for $m = 40$, $L = 3$ and $p = 0.1$.

respect to the period of time in which the channel behavior (p) remains constant, then it results to be useless. For this reason, in addition to energy requirements we have computation delay requirements dictated by the stationarity period characterizing the PDU error process. For these reasons, here, after investigating the major properties of the complementary cumulative distribution, we will propose a very low-complexity and fast heuristic able to accurately approximate such statistics.

In Fig. 6.9, the cumulative complementary distribution $ccdf[K,t]$ is plotted by varying L for $m = 40$ and $p = 0.1$. First of all, we note that this statistics presents a cyclic behavior, where the cycle length is equal to m. Moreover, the property expressed by Eq. (6.11) still holds, i.e., in the logarithmic scale the points at the beginning of each round (slots $K + sm + 1$, $s \geq 0$) are aligned on a straight line (solid bold line in Fig. 6.9), described again by Eq. (6.11) (appropriately setting the parameter t_0). Moreover, for a given value of L, the statistics follows the behavior of the one where L is unlimited approximately until $t = K + Lm$; whereas from this point on (in the last round where the $ccdf$ is greater than zero, i.e., in $t \in (K + Lm, K + (L + 1)m - 2]$[15]) $ccdf[K,t]$ starts decreasing very quickly. It is very interesting to note that the slope concerning this last part appears to be the same as the one characterizing the SDU delivery delay statistics ($P_d[K,t]$) and the SDU transmission delay statistics ($P_{tx}[K,t]$), both reported for comparison in Fig. 6.9. Therefore, in this last part, the statistics follows a straight line (in the logarithmic domain) whose behavior at time t is characterized by Eq. (6.13). To obtain a good approximation of this last part, we can simply use Eq. (6.13) by taking the limit slope, i.e., the one reached when t grows

[15] $P_d[K,t]$ is zero for $t > (L + 1)m + K - 1$, so $ccdf$ is zero for $t > (L + 1)m + K - 2$.

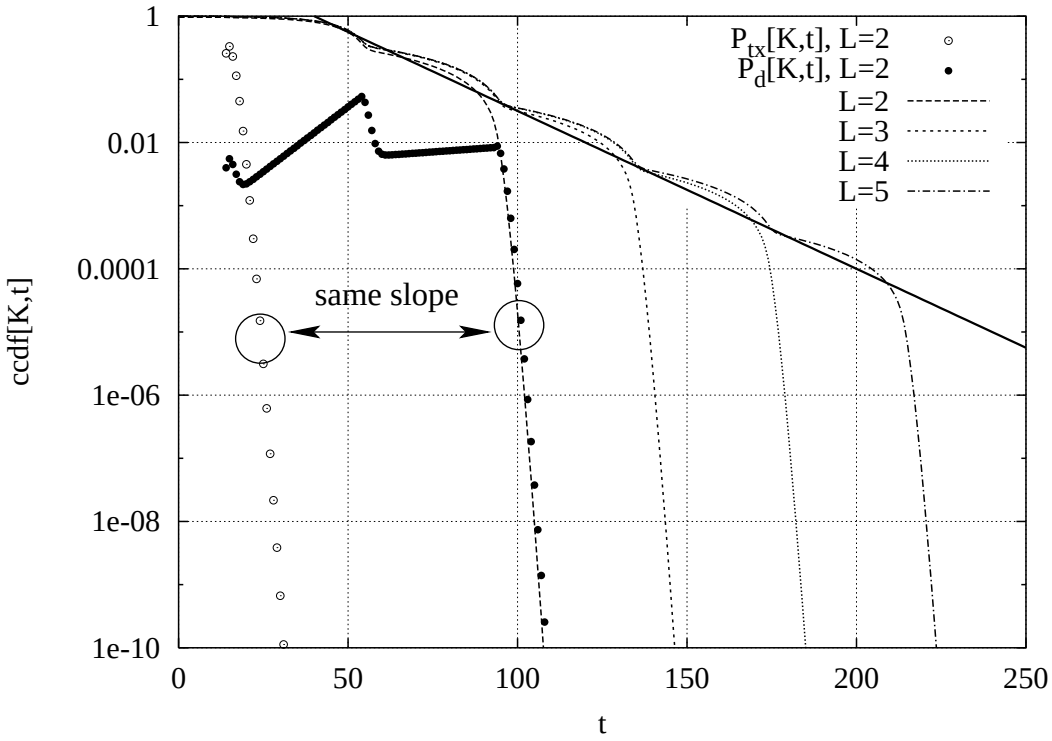


Figure 6.9: $ccdf[K,t]$ by varying L and p for $K = 14$, $m = 40$.

to infinity. In conclusion, for $t \geq K + Lm$, the complementary cumulative distribution can be well approximated by

$$y_f[t] = (1 - q - p_d)^{t - t'_0} \tag{6.17}$$

Now, in order to find a heuristic able to entirely describe such statistics, we need to generate the first part, i.e., the one observed for $0 \leq t \leq K + Lm$. This is quite simple using Eq. (6.11) to find the $ccdf$ values at the beginning of each round and by noting that, in the linear domain, the points inside each round are aligned on a straight line. In conclusion, the behavior of $ccdf[K,t]$, for $0 \leq t \leq K + Lm$ can be well approximated by a piecewise linear function, whereas we can use Eq. (6.17) to approximate the tail of the distribution ($K + Lm < t \leq K + (L+1)m - 2$).

In the next, we first report the function used to fit the $ccdf$ statistics when L is unlimited. Let t_i be the first slot of round i, $i \geq 1$, then $t_i = K + (i-1)m + 1$. As discussed above, we can use the function $y[t]$ to find the probability corresponding to the points at the beginning of each round, t_i. Moreover, we can approximate the behavior of $ccdf$ between these points by means of a line whose extremes are $y[t_i]$

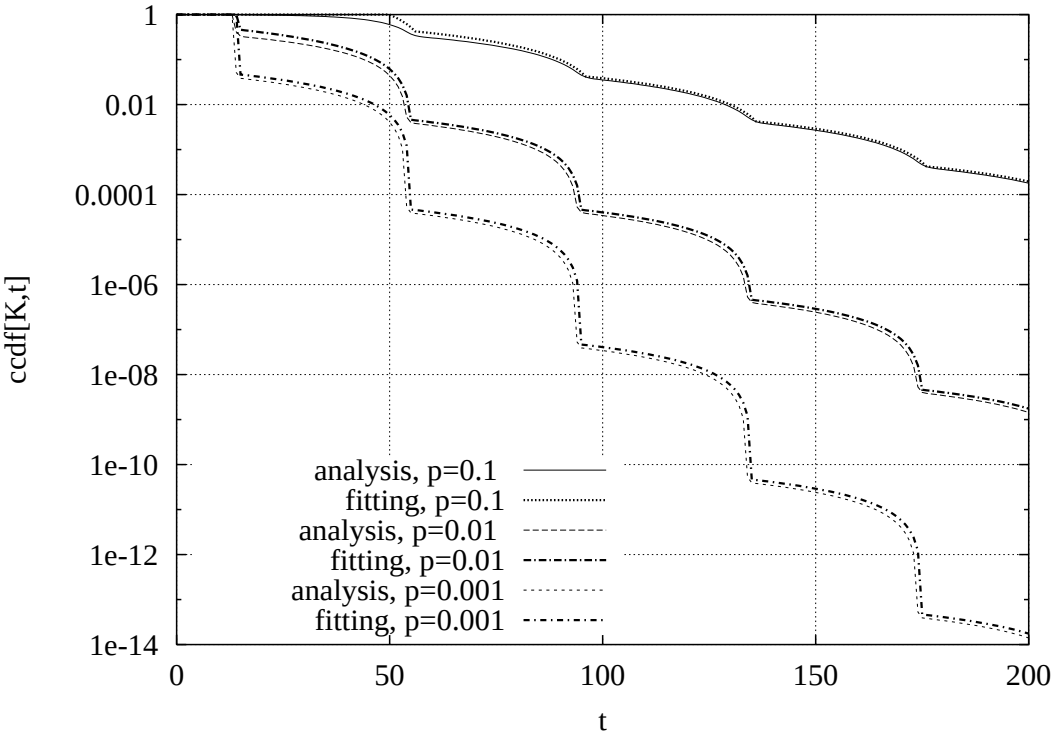


Figure 6.10: $ccdf[K,t]$, comparison between analysis and fitting for $K = 14$, $m = 40$, L unlimited.

and $y[t_{i+1}]$. In practice, *ccdf* can be well approximated by the following function

$$f'[K,t] = \begin{cases} 1 & t \in [0, K-1] \\ 1 - \psi[0,0,0]q^{K-1} & t = K \\ y[t_i] + \dfrac{(y[t_{i+1}] - y[t_i])(t - t_i)}{m} & t \in [t_i, t_{i+1}), \\ & \forall i \text{ s.t. } i \geq 1 \end{cases} \tag{6.18}$$

In Fig. 6.10 we compare the cumulative distribution obtained analytically against the one derived using Eq. (6.18). As can be observed from that figure, the approximation is very accurate for any p.

Now, we investigate how to approximate $ccdf[K,t]$ when L is limited. For this purpose, we consider the approximation given by Eq. (6.18) for $t \in [0, K + Lm]$, whereas, to fit the tail of the *ccdf* ($t \in (K + Lm, K + (L+1)m - 2]$), we use the function $y_f[t]$. Formally

$$f[K,L,t] = \begin{cases} f'[K,t] & t \in [0, K + Lm] \\ y_f[t] & t \in (K + Lm, K + (L+1)m - 2] \\ 0 & t > K + (L+1)m - 2 \end{cases} \tag{6.19}$$

the parameter t'_0 in $y_f[t]$ can be computed by requiring that $y[K + Lm + 1] = y_f[K + Lm + 1]$, i.e., that

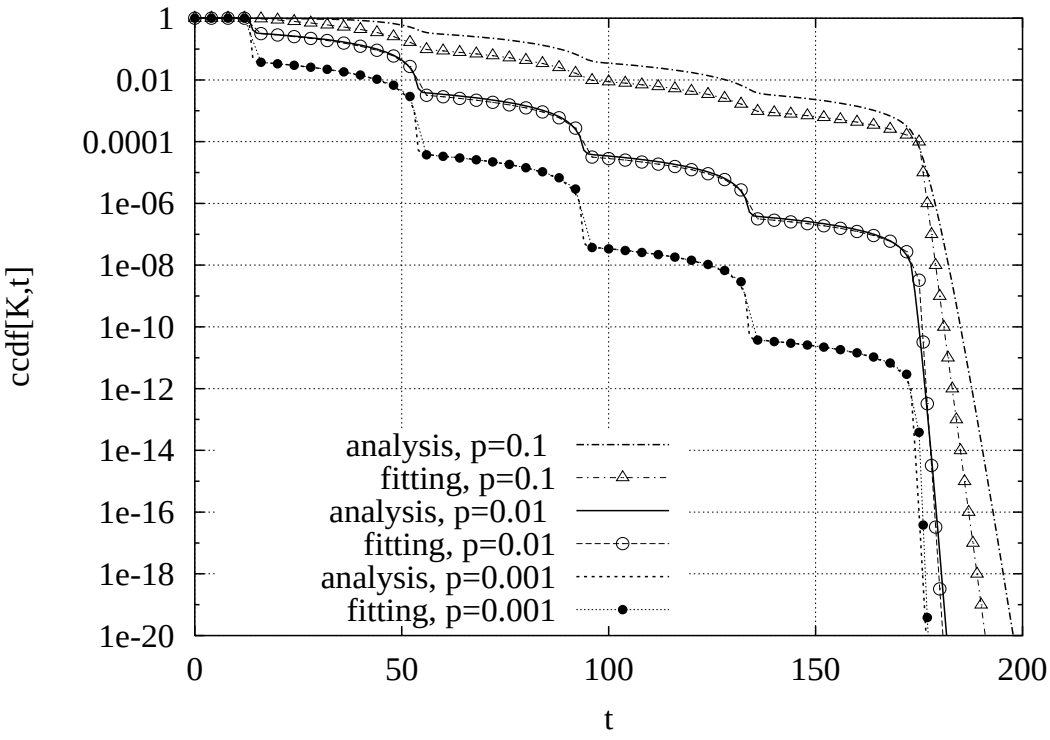


Figure 6.11: Comparison between $ccdf[K,t]$ and $f[K,L,t]$ for $K = 14$, $m = 40$, $L = 4$.

the first point of $y_f[t]$ match with $y[t]$ in the last round. Hence, t_0' is found as

$$t_0' = t_{L+1} - \frac{\log(y[t_{L+1}])}{\log(1-q-p_d)} \tag{6.20}$$

At this point, to obtain the complete statistics from Eq. (6.19) the only parameter that needs to be specified is t_0 (Eq. (6.11)). This parameter, for a given K, can be accurately fitted and stored in a lookup table as a function of p. This can be achieved obtaining the analytic curves and fitting $y[t_i]$ with $ccdf[K,t_i]$ for a sufficiently large i (in order to capture the asymptotic behavior of the statistics in the logarithmic domain). Moreover, the number of points to be stored in such a table could be limited by exploiting some properties of t_0 as a function of p, i.e., observing that the behavior of t_0 as a function of p is linear for $p \geq 0.004$ and that, for $p \leq 0.001$, it is linear in the $\log p$ domain. Fig. 6.10 has been obtained using this first method.

Another very effective way to estimate t_0 is to consider the probability $ccdf[K,K] = 1-\psi[0,0,0]q^{K-1}$, i.e., the probability to deliver the SDU in more than K slots. Now, supposing that $ccdf[K,t_1 = K+1]$ is well approximated by[16] $y[t_1]$ and taking $ccdf[K,K]$ as an estimate of this quantity, we can find t_0 as follows

$$t_0 = K + 1 - m \times \frac{\log(1-\psi[0,0,0]q^{K-1})}{\log p} \tag{6.21}$$

[16]Note that Eq. (6.11), after a transient phase, gives the exact value of $ccdf[K,t_i]$, $i \geq 1$. Moreover, the length of this phase for small p values tends to 0 and the following approximation holds: $ccdf[K,t_1] \approx y[t_1]$.

In Fig. 6.11 we compare the distribution obtained analytically against the one derived from Eq. (6.19), and using this last method to evaluate t_0. At low p values, t_0 is approximated very well and the statistics is fitted accurately. However, as p increases ($p = 0.1$ in that figure), the assumptions made in the computation of t_0 are less accurate (the fitting, in this case is less accurate too).

The heuristic expressions given in Eqs. (6.17)–(6.21) make it possible to accurately approximate the delay statistics of aggregates of K link layer packets by means of a piecewise linear function. The parameters needed for this computation are the round trip time (expressed in number of packets transmitted, i.e., normalized and rounded up to the packet transmission time), and an estimate of the link layer packet error probability, p. From these values, it is easy to compute $\psi[0,0,0]$ ($q(q+p_d)^{m-1}$) and to find $f'[K,t]$ and $f[K,L,t]$. These functions are then used to estimate the delay ccdf from Eqs. (6.11) and (6.17) based on the estimate of the parameter t_0 as given in Eq. (6.21). As the number of parameters to be estimated and stored is small and the approximate analytical expressions are simple, this approximate technique can be easily implemented on terminals with constrained resources, so that delay-driven algorithms can be implemented on handsets.

In the next section we find the statistics of such an aggregate of K link layer PDUs in the out-of-order delivery case.

6.3.6 SDU Delivery Delay Statistics in the Out-of-Order Delivery Case

In the out-of-order delivery case, each link layer SDU can be passed to higher layers when all the PDUs composing it have been correctly received regardless of the delivery status of other SDUs.

In the following we compute the delivery delay distribution regarding a single SDU. As in previous sections, we proceed starting from the slot where the first PDU composing that SDU is transmitted for the first time over the channel. Without loss of generality, we assume that such PDU is transmitted in position 1 in its transmission round (referred to as round 1). We refer as slot $t = 1$ to the slot where this transmission occurs. Note that, for the reasons discussed in previous sections, we are sure that in slot $1 - m$ a successful transmission occurred. Hence, the whole statistics is conditioned on the fact that a correct transmission occurred in slot $1 - m$. Once again, we assume K to be the (integer) number of link layer PDUs composing one SDU.

In the next, we present an approximate approach that is able to fit very accurately the SDU delivery statistics when $K \ll m$ or $K \leq m$ and $p \ll 1$. This analysis is valid as long as $K \leq m$. However, note that in UMTS, due to the large link layer round trip time (up to 220 ms), this condition is likely verified. In the computation of the out-of-order delivery delay statistics we relax the hypothesis of having a finite L, i.e., we assume an infinite number of retransmission attempts.

Our analysis is based on the consideration that, when $K \ll m$ and/or $p \ll 1$, with almost probability 1, all the K PDUs composing the tagged SDU are transmitted for the first time in round 1. Where this hypothesis holds, the delay statistics, i.e., the probability that a tagged SDU is released in slot η, $\eta \in \{1, 2, \ldots, m\}$ of round ξ, $\xi \geq 0$ given that the first PDU composing it is transmitted for the first time

in slot 1 can be found with great accuracy using the following set of equations

$$P_d[\xi m+\eta] = \begin{cases} 0 & \xi m+\eta < K \\ f_1(\eta,K)q^K & \xi = 0,\ \eta \geq K \\ \sum_{i=K}^{m}\sum_{k_1=1}^{\eta}\left[f_1(i-\eta+1,K-k_1+1)\ f_2(\eta,k_1)\mathcal{P}(K,k_1,\xi)\right] & \xi > 0,\ \eta < K \\ \sum_{k_1=1}^{K} f_1(\eta,k_1)\mathcal{P}(K,k_1,\xi) & \xi > 0,\ \eta \geq K \end{cases} \quad (6.22)$$

where $\xi \geq 0$, $\eta \in \{1,2,\ldots,m\}$. The functions f_1, f_2 and $\mathcal{P}$ are reported below

$$f_1(x,y) = \binom{x-2}{y-2} p^{x-y} q^{y-1} \quad (6.23)$$

$$f_2(x,y) = \begin{cases} 1 & x = 1 \\ f_1(x,y) & x > 1 \end{cases} \quad (6.24)$$

$$\mathcal{P}(K,k_1,\xi) = (1-p^{\xi})^{K-k_1}(1-p^{\xi+1})^{k_1-1}p^{\xi}q \quad (6.25)$$

In particular, in Eq. (6.22), for $\xi m+\eta \in \{K,\ldots,m\}$, i.e., $\xi = 0$ and $\eta \in \{1,2,\ldots,m\}$, we compute the probability that, in round 0, there are K successes[17] in slot 1 through η and that the last (the K-th) success is in position η (term $f_1(\eta,K)$), i.e., all the SDU is transmitted during the first round and the last PDU composing it (the K-th) is transmitted in position η. Then, we multiply this probability by the probability that all these K PDUs are transmitted successfully in round 1 (term q^K).

For $\xi m+\eta > m$, we subdivide the case where $\eta < K$ from the case where $\eta \geq K$. In the first case, we compute the probability (term $f_2(\eta,k_1)$) of transmitting k_1 PDUs in round 1 (or equivalently of having k_1 successes in round 0) in slot 1 through η, where the last successful PDU (the k_1-th) is transmitted in slot η. Moreover, with the term $f_1(i-\eta+1,K-k_1+1)$ we account for the remaining $K-k_1$ transmissions (of the PDUs composing the tagged SDU) to be placed in slots $\eta+1$ through m of the first round (where i is the number of slots needed to transmit the K PDUs, i.e., the position where the K-th PDU is transmitted). After that, we use the $\mathcal{P}$ function to compute the probability that the k_1-1 PDUs transmitted in slot j_1, $j_1 < \eta$, are correctly received up to round $\xi+1$ (term $(1-p^{\xi+1})^{k_1-1}$), that the PDU in position η is transmitted correctly in slot $\xi m+\eta$ (term $p^{\xi}q$) and that the $K-k_1$ PDUs in positions j_2, $j_2 > \eta$ are successfully transmitted up to and including slot ξm (term $(1-p^{\xi})^{K-k_1}$).

When $\eta \geq K$, instead, we first compute the probability that k_1 PDUs are transmitted in positions $\{1,2,\ldots,\eta\}$ (in the first round) and that the last (the k_1-th) of these PDUs is transmitted exactly in position η (term $f_1(\eta,k_1)$). Then, using the function $\mathcal{P}$ we account for the probability that the PDUs in slots $\{1,2,\ldots,\eta-1\}$ are correctly transmitted up to and including round $\xi+1$, that the k_1-th PDU is transmitted correctly in position η of round $\xi+1$ and that all the remaining PDUs ($K-k_1$, that are transmitted in position $j \in \{\eta+1,\ldots,m\}$) are successfully transmitted up to and including round ξ.

In Fig. 6.12 we report the comparison of the cumulative complementary delay distribution ($ccdf$) between the *in-order* and *out-of-order* delivery cases. As expected, the out-of-order case is characterized

[17] As noted above these successes will enable K new PDU transmissions in the following round.

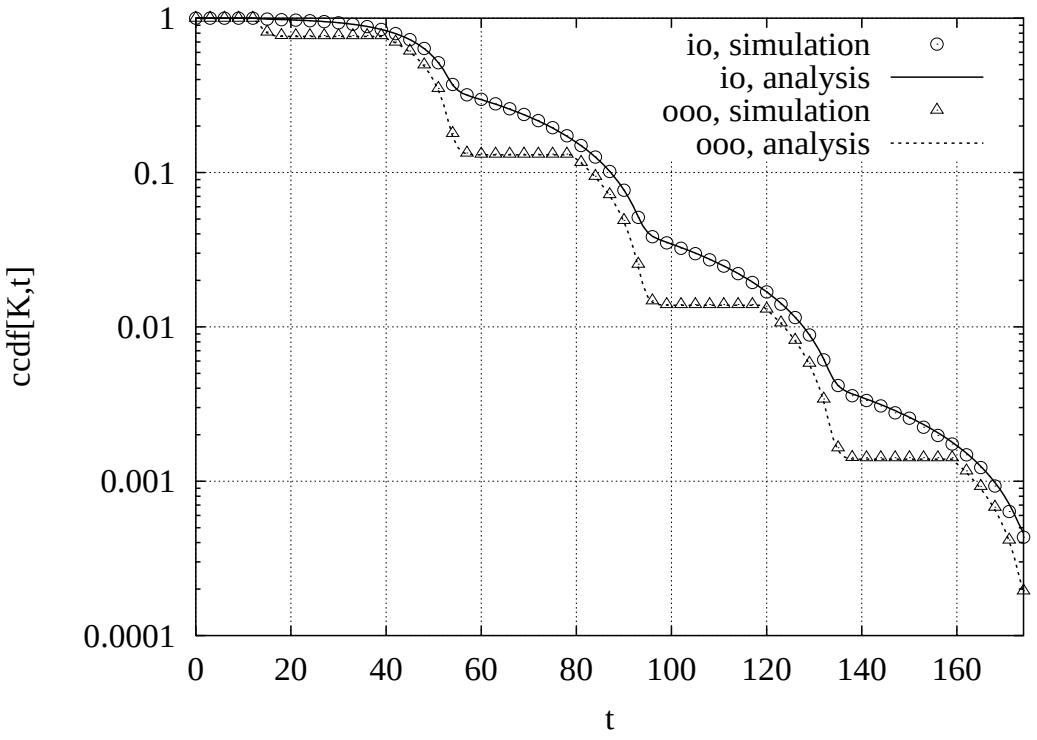


Figure 6.12: $ccdf[K,t]$: comparison between in-order (io) and out-of-order (ooo) delivery cases considering $K = 14$, $p = 0.1$ and $m = 40$.

by the lowest delivery delay; for some values of t, in this case, the cumulative delivery delay probability is reduced by a factor of 3 with respect to the *in-order* delivery case. This could be very useful in the presence of delay constrained flows. It is worth noting that also in this case the delivery delay statistics are characterized by a cyclic behavior that can be captured using Eq. (6.11) by appropriately tuning the t_0 parameter. In this case, the statistics can be accurately fitted noting that the first part of each round is characterized by a constant value, whereas after this phase the distribution starts decreasing until the beginning of the following round. Hence, by using twice Eq. (6.11), i.e., to track the starting point of each round and the ending point of the corresponding constant phase, and using again straight lines to approximate the decreasing behavior after constant periods, it is straightforward to obtain accurate and fast heuristics also for the out-of-order delivery case. These heuristics are not shown here due to space constraints but can be obtained based on what explained in previous sections.

6.4 Analysis over a Two-State Markov Channel Model

In the remaining of the Chapter, the focus will be put on more complicated channel models. In the following sections, the delay statistics at the link layer will be achieved considering a Two-State Markov Channel model (which is introduced in Section 6.4.1). In Section 6.4.2, the analytical framework to ob-

tain delay statistics in such a case is presented. In Section 6.4.3, an approximated approach is given in order to reduce the statistics computation complexity. Moreover, in Section 6.4.4 a methodology to obtain such delay statistics for aggregate of link layer packets will be provided. This is an extension of the analysis presented in Sections 6.4.2 and 6.4.3 which enables the computation of application layer packets statistics. Some results for the Two-State Markov Model are reported in Section 6.4.5.

In the last sections of the Chapter (Section 6.5 through 6.5.4) delay statistics will be derived under a generic N-State Markov Model, which, by means of an opportune tuning of its transition probabilities, is able to provide a fairly good approximation for the underlying fading process in a wide range of scenarios.

6.4.1 Two-State Markov Model

We assume here that the wireless channel is characterized by means of a two-state Discrete Time Markov Chain (DTMC) [52] [93], whose states are named 0 and 1. The related transition probability matrix $\mathbf{P}$ and the corresponding i-step transition probability matrix $\mathbf{P}(i)$ are defined as follows

$$\mathbf{P} = \begin{pmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{pmatrix} \tag{6.26}$$

$$\mathbf{P}(i) = \mathbf{P}^i = \begin{pmatrix} p_{00}(i) & p_{01}(i) \\ p_{10}(i) & p_{11}(i) \end{pmatrix} \tag{6.27}$$

The steady-state channel error probability is $\varepsilon = \frac{p_{01}}{p_{10}+p_{01}}$, while the average error burst length is given by $b = 1/p_{10}$. We model the errors in the channel with the hypothesis that transmissions occurring during state 1 are always erroneous, whereas state 0 is error free. This is a reasonable assumption in many cases [117] [115]. From a theoretical point-of-view, even such a simple model is able to give the necessary insight for the analysis. ARQ system model and assumptions are the same considered in Section 6.3.1.

6.4.2 Exact Analysis

Let us compute the delivery delay statistics of Selective Repeat ARQ for a single PDU. As in Section 6.3.1, we mark a PDU of interest, called in the following *tagged PDU*, and to track its successful delivery. Note that this implies a joint tracking of the transmissions of the tagged PDU, as well as all previous PDUs.

Some remarks about the notation used for the rest of the Chapter are presented here. The slot in which the tagged PDU is transmitted for the first time will be indicated as slot $t = m$. The m-sized window from slot 1 to slot m will be called *fundamental window*, due to the important role it plays in the analysis. We will finally consider the delivery of the tagged PDU as complete when all PDUs with smaller id number have been correctly received. This is not completely appropriate, since the release time is evaluated at the receiver. However, the delay between these two instants is a constant D_s (known a priori and approximately equal to $m/2$), that is the sum of the path delay and the processing delay. Since this term does not affect the analysis, we will ignore it for simplicity. Thus, in the following we will study the statistics of $P_d[k]$, defined as the probability that the delivery delay equals k slots plus the

constant term D_s. For example, $P_d[0]$ is the probability that the tagged PDU is released at the instant of its reception, i.e., the first transmission attempt is successful and the re-sequencing delay is zero.

To evaluate the release of the tagged PDU requires checking whether each PDU with lower id has been correctly received or not. At first glance, this seems to require a large amount of memory, as the delivery of the tagged PDU can be blocked by a PDU with an arbitrary low id. However, it would not be hard to analytically prove that the memory required to study the delivery of the tagged PDU equals the status of exactly m PDUs. A formal proof of this statement can be found in [94]. In particular, the tagged PDU is released upon correct reception of all PDUs transmitted in the fundamental window. Given the above notation, this means that the PDUs that block its release are necessarily transmitted in one of the slots between 1 and $m-1$. Intuitively speaking, this fact can be justified from the SR ARQ behavior previously explained: m slots after its transmission, each PDU is either identified as correctly received or retransmitted. Thus, each PDU with lower id than the tagged one is either correctly received before time 1 (hence, even at the instant of first transmission of the tagged PDU) or transmitted during the fundamental window.

At the same time it can be easily proven that the transmission at time 0 must be correct (otherwise the transmission at time m could not be the *first* transmission of the tagged PDU, rather it would have been a retransmission), and also that each transmission of a new PDU occurring at time $t > m$ can not affect the delivery of the tagged PDU, as the new PDU must have an id larger than the tagged one. Hence, our study ignores all PDU arrivals after time m. The problem to be solved is therefore to find the time it takes for all PDUs transmitted in slots 1 through m to be eventually received correctly, given that a successful transmission occurred in slot 0.

To evaluate the resolution of the entire fundamental window we formulate this algorithm, in which the slots in position $t \geq 1$ are marked as follows

1. Every slot $t \geq 1$ begins unmarked. Let $t = 1$.
2. If every slot from t on is marked, the procedure ends. If slot t is marked but there are unmarked slots left, increase t until an unmarked slot is encountered.
3. If in t the transmission is successful, mark with the label *resolved* every slot in position $\kappa m + t$, with κ integer, $\kappa \geq 0$. Increase t by 1 and go to step 2.
4. Else, in t an erroneous transmission occurs. In this case, mark only slot t with the label *unresolved*, increase t by 1 and go to step 2.

For example, let $m = 3$ and suppose that a good channel state at $t = 1$ is followed by a burst of four erroneous slots and then the channel is again good for three more slots. The algorithm gives: 1=resolved, 2=unresolved, 3=unresolved, 4=resolved (despite the channel error, as it was previously marked), 5=unresolved, 6=resolved, 7=resolved, 8=resolved. After slot 8, the algorithm ends, because every slot in a higher position is marked as resolved.

It is straightforward to prove that this is always true, i.e., after a slot with a sufficiently high position every other slot is marked as resolved. This is equivalent to saying that there is somewhere a sequence of m consecutive resolved slots, after which every further slot is also resolved.

This algorithm has an intuitive justification as follows: if slot $1 \leq t \leq m$ contained an erroneous transmission, the tagged PDU could not be released, as at least one PDU in the fundamental window has not been correctly transmitted. By marking slot t as *unresolved*, we imply that a retransmission will be scheduled in slot $m + t$. If this retransmission is successful, then slot $m + t$ will be marked as *resolved*, otherwise it will be marked as *unresolved* again (with the effect of a further retransmission in slot $2m + t$ and so on until success). On the other hand, if a slot t contains a successful transmission, i.e., is marked as *resolved*, we can neglect each transmission in slot $\kappa m + t$, with κ positive integer. In fact, once a packet is successfully transmitted at time t, each slot $\kappa m + t$ with $\kappa > 0$ contains a transmission of a PDU with larger id than the tagged one. By recalling that for our purposes future arrivals are ignored, we can mark not only slot t but also each slot $\kappa m + t$ as resolved, that implies that it does not affect the delivery of the tagged PDU.

In general, slot $m+t$ (with $t \geq 1$) will be marked as resolved if slot t was itself resolved or the channel state in slot $m + t$ is good, and will remain unresolved otherwise. Since the delivery of the tagged PDU corresponds to the resolution of the entire fundamental window, we can check when the first sequence of m resolved slots occurs. Incidentally, note that after this sequence every slot is resolved. *Thus, a suitable model to track the relevant events is one in which memory is kept about the resolved/unresolved status of the $m - 1$ most recent past slots*.

In other words, at any given time t, we need to know the status of slots $t-m+1, t-m+2, \ldots, t-1$. A binary variable is therefore assigned to each slot to carry this information. Considering t as the current slot, $b_k = 1$ if slot $t - m + 1 + k$ is still unresolved, and $b_k = 0$ otherwise, for $k = 0, 1, \ldots, m - 2$. We obtain in this way a string of bits denoted by $\mathbf{b}$ that keeps memory of which slots are yet to be resolved. An equivalent representation for the bitmap $\mathbf{b}$, used in the following to simplify the notation, is the integer $i = \sum_{k=0}^{m-2} b_k 2^k$.

We also need to specify the status of the current slot, i.e., slot t. In this case a binary variable is no longer sufficient, since we also need to track the channel state, which is necessary to determine the future evolution of successful transmissions. Notice that the Markovian nature of the channel evolution makes it possible to ignore the channel state in slots $t - m + 1, t - m + 2, \ldots, t - 1$ once the channel state in t is known. The only channel state required for the analysis is thus that of the current slot. Three situations are possible: the channel is good, which implies that the slot is resolved (if it was not resolved already, the good channel state makes it resolved now); the channel is bad and the slot is resolved, in a previous transmission; the channel is bad and the slot is still unresolved. These three possibilities will be denoted by 0, 1 and 2, respectively, and the associated variable will be denoted by ω.

Consider now the random process $X(t) = (i(t), \omega(t))$ which jointly tracks slot-by-slot the Markov channel evolution and the status of the m latest slots. This process is a Markov chain. In order to determine the possible transitions $X(t) \to X(t + 1) = (i', \omega')$ and the corresponding transition probabilities, suppose at time t the bitmap $\mathbf{b}$ is $(b_0, b_1, \ldots, b_{m-2})$, where the most significant bit b_{m-2} denotes the status of the most recent among the past slots. At time $t+1$ this bitmap is clocked one position into the past, i.e., $\mathbf{b}' = (b'_0, b'_1, \ldots, b'_{m-3}, b'_{m-2}) = (b_1, b_2, \ldots, b_{m-2}, f(\omega))$, where $f(\omega) = 1$ if $\omega = 2$ (current slot at time t was still unresolved), and $f(\omega) = 0$ if $\omega = 0, 1$. (More compactly, in this case $f(\omega) = \lfloor \omega/2 \rfloor$.)

Regarding the value of $\omega' = \omega(t + 1)$, note the following. If at time t $b_0 = 0$, the corresponding slot has already been resolved, and therefore $\omega' = 0$ or 1 according to the channel state at time $t + 1$. On the

other hand, if $b_0 = 1$, the slot is still unresolved at time t, hence we have $\omega' = 0$, if the channel at time $t+1$ is good (slot is resolved at this time), and $\omega' = 2$ otherwise (slot remains unresolved). There are only two possible destinations for $X(t+1)$, given $X(t)$, since the shift of the bitmap is deterministic and the only random variable is the channel state which can assume two values. More precisely, the transition probabilities are given as follows

- if i is even (i.e., $b_0 = 0$), then

$$P[X(t+1) = (i', \omega') | X(t) = (i, \omega)] = \begin{cases} p_{xy} & \text{if } i' = \lfloor \frac{i}{2} \rfloor + \lfloor \frac{\omega}{2} \rfloor 2^{m-2}, \\ & x = \lceil \frac{\omega}{2} \rceil, \omega' = y, y = 0, 1 \\ 0 & \text{otherwise} \end{cases} \tag{6.28}$$

- if i is odd (i.e., $b_0 = 1$), then

$$P[X(t+1) = (i', \omega') | X(t) = (i, \omega)] = \begin{cases} p_{xy} & \text{if } i' = \lfloor \frac{i}{2} \rfloor + \lfloor \frac{\omega}{2} \rfloor 2^{m-2}, \\ & x = \lceil \frac{\omega}{2} \rceil, \omega' = 2y, y = 0, 1 \\ 0 & \text{otherwise} \end{cases} \tag{6.29}$$

where the use of $\omega' = 2y$ in the latter case means that a good channel $y = 0$ leads to $\omega' = 0$ whereas a bad channel $y = 1$ leads to $\omega' = 2$, i.e., the situation of bad channel and unresolved slot. According to the above rule, the transition probability matrix can be built, with two non-zero entries per row.

In order to find the delay statistics, we proceed as follows. Let $\boldsymbol{\pi} = [\pi_0 \ \pi_1 \ \cdots \pi_K]$ be a $1 \times K$ vector whose $K = 3 \cdot 2^{m-1}$ entries represent the probabilities that the system starts in a given state. $\boldsymbol{\pi}$ is computed as follows

$$\text{if } \omega \text{ is even (0, 2):} \quad \pi_{(i,\omega)} = p_{0 b_0} \left[\prod_{j=1}^{m-2} p_{b_{j-1} b_j} \right] p_{b_{m-2} \frac{\omega}{2}} \tag{6.30}$$

$$\text{if } \omega \text{ is odd (1):} \quad \pi_{(i,\omega)} = 0 \tag{6.31}$$

Let $\boldsymbol{e}_0$ be a column vector of all zeros except for the entries corresponding to states $(0,0)$ and $(0,1)$, that are equal to 1. If $\boldsymbol{T}$ is the transition matrix of the Markov chain $X(t)$, we determine

$$\mathcal{P}_c[k] = \boldsymbol{\pi} \boldsymbol{T}^k \boldsymbol{e}_0, \quad k \geq 0 \,. \tag{6.32}$$

The distribution $\mathcal{P}_c[k]$ is the probability that the delivery delay is less than or equal to k. Finally, $P_d[k]$ is determined as

$$P_d[0] = \mathcal{P}_c[0], \ \ P_d[k] = \mathcal{P}_c[k] - \mathcal{P}_c[k-1] \ \ \forall k > 0 \,. \tag{6.33}$$

Another interesting distribution is the cumulative complementary distribution of the delivery delay statistics, $ccdf[x]$, defined as:

$$ccdf[x] = \text{Prob}\{\text{delay} > D_s + x\} = 1 - \mathcal{P}_c[x]. \tag{6.34}$$

This value, which can be directly derived in our model, represents the probability that the delivery delay exceeds x slots. Henceforth, it has great importance in estimating whether delay constraints are met for real-time applications.

6.4.3 Approximated Analysis

Even though the above analysis is exact, it has the main drawback of having a complexity which is exponential in m. In more detail, the number of states that is needed to track the ARQ window evolution is growing as 2^m. Hence, for large values of the round trip delay, the computation of the exact statistics becomes both memory and time expensive making it impossible to compute such statistics directly at the link layer. In this Section, we propose a simple approximate approach that allows to reduce the computational complexity enabling the computation of the statistics for large m. In more detail, instead of tracking the exact position of each unresolved slot, we propose to analyze the resolving process simply by tracking the number of remaining slots to be resolved.

To do that, we consider rounds of m slots and build a Markov chain in which the state is represented by $X(r) = (n_e(r), C(r))$, $r \geq 1$, where $n_e(r)$ and $C(r)$ are the number of unresolved slots in round r and the channel state in the last slot of round r, respectively. This Markov chain evolves round by round, i.e., m slot at a time. Round 1 is the one comprising slots 1 through m. Note that, in this Markov chain, the knowledge about the position of unresolved slots is completely ignored. Moreover, in order to keep track of errors and corrections, we arbitrarily assume that all unresolved slots are deterministically placed at the end of each round. This simple approximation rule allows us to neglect the exact position of each unresolved slot and to achieve a complexity linear in m.

Let $\phi_{ij}(k, n)$, $i, j \in \{0, 1\}$ be the probability that k slots in $\{1, 2, \ldots, n\}$ are successful and the channel state is j at time n, given that the channel state was i at time 0. This is a well-known function that can be derived either in recursive [52] or close [112] form.

Now, let $\Psi_{0j}(e, r)$ be the probability of having e ($1 \leq e \leq m$) unresolved slots in round r and that the channel in the last slot of round r is j given that the channel in slot 0 was correct. This function can be computed recursively as follows

$$\Psi_{0j}(e, r) = \begin{cases} \phi_{0j}(m - e, m) & r = 1 \\ \sum_{k=e}^{m} \sum_{i \in \{0,1\}} \Psi_{0i}(k, r-1) \mathcal{R}_{ij}(k - e, k) & r > 1 \end{cases} \tag{6.35}$$

where,

$$\mathcal{R}_{ij}(q, k) = \begin{cases} \phi_{ij}(q, k) & k = m \\ \sum_{c \in \{0,1\}} p_{ic}(m - k) \phi_{cj}(q, k) & k < m \, . \end{cases} \tag{6.36}$$

The function $\mathcal{R}_{ij}(q, k)$ is used to compute the probability that q slots are resolved over k in round r, and that the channel state in the last slot of round r is j given that the channel state in the last slot of the previous round ($r - 1$) was i, and that all unresolved slots are deterministically grouped at the end of each round. Note that the recursive expression (Eq. (6.35)) is initialized (round $r = 1$) by exploiting the knowledge of the channel at time 0 and computing the mean probability to have e erroneous transmissions in any order in that round. Moreover, the probability to have e unresolved slots, $1 \leq e \leq m$, at the end of round r, $r > 1$ ($\Psi_{0j}(e, r)$) is obtained by considering the probability to have k ($\Psi_{0i}(k, r-1), e \leq k \leq m$) unresolved slots after $r - 1$ rounds and that exactly $k - e$ of these slots are resolved in round r ($\mathcal{R}_{ij}(k - e, k)$). Finally, these probabilities are summed over $e \leq k \leq m$ and over $i \in \{0, 1\}$ to account for all the admitted values of unresolved slots and channel states at the end of round $r - 1$. Observe

that the probability to be in state $X(r) = (n_e(r), C(r))$, $1 \leq n_e \leq m$, $C \in \{0, 1\}$, $r \geq 1$ is given by $\Psi_{0C}(n_e, r)$.

Now, let us write the time index t as $t = \xi m + \eta$, where $\xi \geq 0$ and $1 \leq \eta \leq m$ are the number of full rounds and the number of slots in the current round covered by t, respectively. With this decomposition, time t belongs to round $\xi + 1$, whereas ξ is the previous round.

In the following, we discuss an approximate approach for the evaluation of the probability $P_d[k]$. As a first step, using the function $\Psi_{0j}(e, \xi)$, we evaluate the probability that the last PDU is resolved in round $\xi + 1$. Note that this is the probability of having $e > 0$ outstanding errors in round ξ and zero in round $\xi + 1$. This evaluation assumes that the e outstanding errors correspond to consecutive slots within a round. Conditioned on the last PDU being resolved in round $\xi + 1$, we can approximate the probability distribution of the specific slot in round $\xi + 1$ in which this resolution takes place by just assuming that all positions of the error burst within the round are equally likely.

More specifically, we adopt two slightly different approaches. In the first approach, called *Burst at the End (BE)*, we first compute the probability that $1 \leq e \leq m$ erroneous packets ($\Psi_{0j}(e, \xi)$) are inherited from the previous round ξ and we evaluate the probability to resolve these PDUs by grouping them in a single burst at the end of round $\xi + 1$, i.e., in position $m - e + 1$ through m of that round. Note that following this procedure the exact position of the erroneous PDUs is not tracked and what we obtain is the approximate probability for the release of the tagged PDU in round $\xi + 1$. However, what we need is not just the probability to release the tagged PDU in a given round, but the probability that the tagged packet is released in slot k of that round, $1 \leq k \leq m$. To derive an approximation for this probability, we assume that the slot in which the last of the e PDUs is resolved is uniformly distributed between position e and m of round $\xi + 1$. The resulting distribution is a simplified version of the actual one that is indeed more complicated, even though the agreement is good also for large values of m.

In the second approach, named in the sequel *Shifted Burst (SB)*, we evaluate the probability of resolving the burst of consecutive erroneous slots inherited from the previous round (ξ) in a uniformly chosen position of the final round. Unlike in the previous approach, where the burst could not end in slot $k < e$, in this case we allow any value for k (between 1 and m). If k is smaller than the size of the burst, we admit that the burst is cyclically shifted. E.g., if the last position of a burst of length 3 is the second slot, we split the shifted burst in two sub-burst sequences such as the erroneous slots will be the first two and the last one. Note that the burst can be in any position with equal probability.

In both strategies we assume that the real instant of resolution in the final round $\xi + 1$ (which is neglected, as the analysis is carried out round–by–round) is one of the slots between $\xi m + 1$ and $(\xi + 1)m$ with equal probability. The difference is in the order between the evaluation of the resolution probability and the uniform distribution assumption among the slots in round $\xi + 1$. BE first evaluates the probability of resolving the errors concentrated in a burst at the end of round $\xi + 1$; after that, this probability is uniformly distributed among the possible slots in that round. SB, instead, considers directly that the burst ends in position $\xi m + 1 \leq k \leq (\xi + 1)m$ (with probability $1/m$); in this case the contributions of all the m-cyclically shifted bursts where the last erroneous packet occupies position k are summed together to give the resolution probability in that slot.

Let us define the function ρ as the probability that the channel in the last slot of round ξ is j and that exactly e ($1 \leq e \leq m$) slots are yet to be resolved at the end of round ξ and that these e slots are all

resolved in round $\xi + 1$ (final round) given that the last unresolved slot is in position p, $e \leq p \leq m$. Formally

$$\rho(j, e, \xi|p) = \Psi_{0j}(e, \xi) p_{j0} (p - e + 1) p_{00}^{e-1} \tag{6.37}$$

This function is the probability of resolving a burst accounting for its length (e), the position occupied by its last element (p) and the channel state (j) at the end of round ξ. In the following we report the detailed description of the two approaches.

Burst at the End (BE) Approach

The delivery delay statistics is computed in the following way

$$P_d[k] = \begin{cases} \dfrac{1}{m} \displaystyle\sum_{j \in \{0,1\}} \sum_{e=1}^{m} \rho(j, e, \xi|m) u[\eta - e] & \eta \neq m \\ \dfrac{1}{m} \displaystyle\sum_{j \in \{0,1\}} \sum_{e=1}^{m} \rho(j, e, \xi|m) e & \eta = m \end{cases} \tag{6.38}$$

where $k = \xi m + \eta$, with $\xi \geq 0$ and $1 \leq \eta \leq m$. The function $u[\cdot]$ is the discrete time unit step, i.e., $u[n] = 1$ for $n \geq 0$, whereas $u[n] = 0$ for $n < 0$, n integer.

The probability of the first passage through states $(0, C)$ in round $\xi+1$, i.e., for $k \in \{\xi m+1, \ldots, \xi m+m\}$ is computed by considering all the unresolved slots inherited from round ξ to be placed at the end of round $\xi+1$. Moreover, this probability is subdivided among slots $\xi m+\eta$, $\forall\, \eta \in \{1, \ldots, m\}$ by assuming that the final unresolved slot is distributed by means of the function $\mathcal{P}_E(\eta|e)$ that is the approximate probability that the burst ends in position η given that it consists of e slots. $\mathcal{P}_E(\eta|e)$ is defined a priori as follows

$$\mathcal{P}_E(\eta|e) = \begin{cases} 0 & \eta < e \\ 1/m & e \leq \eta < m \\ e/m & \eta = m \end{cases} \tag{6.39}$$

To sum up, in this approach we first resolve all the unresolved slots inherited from the previous round ξ by deterministically grouping it at the end of the round. Thus, the last unresolved slot is always in position m of each round. What we obtain in this way is the probability of release of the tagged packet in round $\xi + 1$. After that, we subdivide such probability among all slots $\xi m + \eta$, with $1 \leq \eta \leq m$ by considering the distribution $\mathcal{P}_E(\eta|e)$.

Shifted Burst (SB) Approach

In this approach, we also consider that the unresolved slots inherited from round ξ are grouped in a single burst, but unlike in BE, we evaluate the resolution of such a burst assuming that its last element can be cyclically shifted (with probability $1/m$) between position 1 through m of round $\xi + 1$. With this assumption the delivery delay statistics can be written as

$$P_d[k] = \begin{cases} \dfrac{1}{m} \displaystyle\sum_{j \in \{0,1\}} \sum_{e=1}^{m} \rho(j, e, \xi|\eta) u[\eta - e] & \eta \neq m \\ \dfrac{1}{m} \displaystyle\sum_{j \in \{0,1\}} \sum_{e=1}^{m} \left[\rho(j, e, \xi|m) + \sum_{y<e} \rho'(j, e, \xi|y) \right] & \eta = m \end{cases} \tag{6.40}$$

where:

$$\rho'(j,e,\xi|p) = \Psi_{0j}(e,\xi)p_{j0}p_{00}^{p-1}p_{00}(m-e+1)p_{00}^{e-p-1}. \quad (6.41)$$

With Eq. (6.40), we compute, for each $\eta \in \{1,\ldots,m\}$ in round $\xi+1$, the probability to resolve the burst given that its last slot is in position η, where $\eta \geq e$. Thus, when $\eta \geq e$, we guarantee that the first slot of the burst is in position f with $f \geq 1$. For the correction of the burst of unresolved slots, see the function $\rho(j,e,\xi|p)$ with[18] $p=\eta$.

When $\eta = m$, the contributions of m-cyclically shifted versions of the burst are also considered. In more detail, we split the e-sized (where e are the residual errors from the previous round ξ) burst in two parts where the first part is composed by the y slots in position $\{1,\ldots,y\}$, whereas the second part is composed by the $e-y$ slots in position $\{m-e+y+1,\ldots,m\}$.[19] These contributions are then resolved by means of the function $\rho'(j,e,\xi|y)$. ρ' is used to evaluate the probability that the channel in the last slot of round ξ is j and that exactly e ($1 \leq e \leq m$) slots are to be resolved at the end of round ξ and that these e erroneous packets are all resolved in round $\xi+1$ given that they are subdivided in two bursts, where the first one occupies positions $\{1,\ldots,y\}$ and the second occupies positions $\{m-e+y+1,\ldots,m\}$.

6.4.4 Computation of the Delivery Delay Statistics of an Aggregate of ARQ PDUs

In this Section we use the delivery delay statistics $P_d[t]$ previously derived[20] in order to obtain the delivery delay statistics of an aggregate of $K \geq 1$ ARQ PDUs: as will be shown in the following K is not a limiting factor for the model complexity. Even if the procedure for finding such aggregate statistics is illustrated here for a Two-State Markov Channel error model, the principles behind the procedure are general and, through some standard techniques presented in [52], it can be easily translated into its generalized version, i.e., accounting for an N-State Markov Chain.

In any event, our task here is to find out the delivery delay statistics of such an aggregate of K packets. With the term *delivery delay statistics*, we mean the number of slots elapsed between the time in which the first of the K PDUs is transmitted for the first time over the channel and the slot where all the K packets are correctly received and none of them is waiting for *in-order* delivery, i.e., all the K PDUs can be passed *in order* to higher levels.

As previously noted for the tagged PDU, the transmission of the first PDU (of K) in a given window position implies that a correct transmission occurred m slots earlier (in the same window position, see Fig. 6.13). Moreover, when a PDU is lost in a given slot, that PDU will surely be retransmitted m slots later in the same window position. So, each PDU is transmitted/retransmitted in the same window position until success, and only from this point on, the window position occupied by that PDU can be used to transmit a new packet. The key point is the following: each correct transmission in a given round enables a new transmission in the same window position in the following round[21], so counting the number of slots in which the channel is error-free in a given time interval, say $[i, i+N]$, is equivalent to counting how many new packets are transmitted over the channel in the interval[22] $[i+m, i+N+m]$. In practice,

[18] The same function has been used in the BE approach, but with $p=m$.

[19] Note that, for these bursts, the m-th slot of round ξ is always erroneous. That is the reason why they are assigned to the case where the resolution occurs in position $\eta = m$.

[20] The single PDU delivery delay statistics.

[21] Hereinafter, we refer to this process as new transmissions *enabling process*.

[22] Because each error-free slot enables the transmission of a new packet m slots apart.

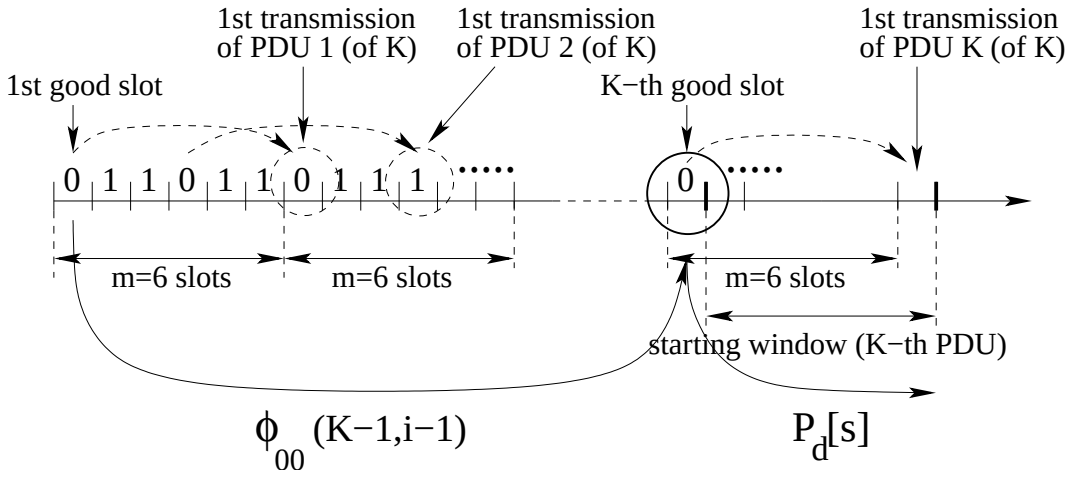


Figure 6.13: Transmission of K ARQ PDUs over the channel.

the process given by new transmissions is simply the *enabling process* deterministically shifted forward in time by m slots (see Fig. 6.13). Hence, the probability distribution regarding the transmission of K packets is the same characterizing the enabling of K new packet transmissions. This last distribution, i.e., the probability distribution of the time i elapsed between the slot in which the first transmission of the first PDU (of K) is enabled and the instant in which the transmission of the K-th packet is enabled is given by $\phi_{00}(K-1, i-1)$.[23] In the sequel we refer to this probability as $P_{en}[K, i]$. $P_{en}[K, i]$ can be computed as follows

$$P_{en}[K,i] = \begin{cases} 0 & i < K \\ \phi_{00}(K-1, i-1) & i \geq K \end{cases} \tag{6.42}$$

Note that here, PDU K plays the role of the tagged PDU in the previous Section, so the slots in which PDU K is enabled and transmitted can be viewed as slot 0 and m, respectively. Furthermore, all the out-of-order packets that PDU K has to eventually wait for after its correct transmission are only the ones transmitted between slot 1 and slot m (Fig. 6.13), i.e., the memory needed to account for the in-order delivery of all the K PDUs does not depend on K, but only on the value of the round trip delay (m).

In conclusion, we can subdivide the delivery delay of the K PDUs in two parts: in the first part we account for the time between the enabling of the first transmission of PDU 1 and the slot in which the transmission of PDU K is enabled ($\phi_{00}(K-1, i-1)$). In the second part, we track the delivery delay regarding the last (K-th) PDU, starting from the slot where the transmission of such PDU is enabled. Moreover, remember that *enabling process* and *new transmission process* are the same process deterministically shifted in time by m slots, and that the delivery delay regarding PDU K is characterized by the function $P_d[s]$ that starts to track the channel m slots before the first transmission of that packet, i.e., in the last slot tracked by $P_{en}[K, i]$. Thus, the distribution characterizing the delay between the instant of the first transmission of PDU 1 and the instant in which all K PDUs can be correctly delivered in-order

[23] $\phi_{rs}(x, y)$ is defined here as the probability to have x good slots out of slots $\{1, 2, \ldots, y\}$ and to have the last slot y in state s, given that slot 0 is in state r. $\phi_{00}(K-1, i-1)$ is then the probability of having K good slots over i, $i \geq 0$. See [52] and [112] for its derivation in recursive and close form, respectively.

can be found as

$$P_d[K,s] = \begin{cases} 0 & s < K \\ \sum_{i=0}^{s} P_{en}[K,i] P_d[s-i] & \text{otherwise} \end{cases} \tag{6.43}$$

Also in the aggregate case, we can easily derive the complementary delivery delay distribution; as above

$$ccdf[K,x] = 1 - \sum_{s=0}^{x} P_d[K,s] \tag{6.44}$$

This function is very important because it is directly related to the delivery time of higher level packets. In the TCP/IP case, for example, it could be used to compute the timeout event probability, i.e., the probability that a given higher level packet experiences a delay larger than the value of the TCP timeout. Performance of the TCP protocol are directly related to this probability; thus, the QoS perceived by the final user is directly related to this metric as well. Another interesting case is the UDP, where the cumulative distribution could be useful to decide whenever the delay perceived by higher level packets fulfills application requirements. For example, in the case of voice traffic we have some delay constraints due by the maximum tolerable interactive delay. Hence, this statistics can be easily translated into performance metrics for any kind of transport protocol/application operating at higher levels.

6.4.5 Results for the Two-State Markov Channel Error Model

The delivery delay statistics $P_d[k]$ has been computed according to the above analysis, for various values of the channel error probability ε and the channel burstiness b. To test the accuracy, we used a simulator in which we implemented the simple transmission of packets with a SR ARQ scheme applied to the same scenario; thus, we empirically measured the delivery delay statistics, in addition to deriving them from the exact analysis.

In Fig. 6.14, we evaluate $P_d[k]$ and compare the case of independent (*iid*) channel with different values of the correlation b. In any case, the shape of the delivery delay statistics presents a step-wise behavior with a logarithmic-constant gap after every positions κm, κ integer. Moreover, both in the *iid* and the correlated error case, the resolving probability presents an increasing behavior with the maximum placed at the end of each round. In the *iid* case, this effect is due to the larger number of combinatorial events leading to the resolution of an m-sized window in the last slot. This behavior vanishes after a few rounds, where $P_d[k]$ becomes almost constant within a given m-sized window. In addition to this, over bursty channels another effect is present. In fact, when the channel correlation is large, an erroneous tagged PDU transmission at time m likely comes with the erroneous transmission of the $m-1$ packets in position 1 through $m-1$. In this case, all packets are likely released in the subsequent error free period and this leads to a larger probability of resolving the window at the end of a round.

An interesting value is $P_d[0]$. When the delivery delay is 0, the tagged packet is released at the end of the slot in which it is received, i.e., both the tagged packet transmission and the $m-1$ previous transmissions are correct. In this case the transmission delay is equal to half the round trip delay and the re-sequencing delay is zero. It can be observed that $P_d[0]$ in the bursty case is higher, due to the higher probability to have a whole window of correct slots when errors occur in burst. This phenomenon has a larger impact if the error probability is high (see Fig. 6.14(b)). As a matter of fact, the probability of delivering the tagged PDU within a delay equal to m can be heavily underestimated by considering the

errors in a bursty channel as independent. The value of $P_d[0]$ has a great impact on the throughput. Thus, we can say that Fig. 6.14 confirms the results presented in [73], where the throughput performance of ARQ on a correlated channel is shown to be better than in the case of static channel. On the other hand, the slope of each curve decreases with increasing b, that is, the larger the bursts, the higher the probability of having large delays. In fact, on average, the channel recovers from an error, i.e., is restored into the good state, after a number of slots equal to b. Thus, for high k we see an increase of the probability of large delivery delays due to the channel burstiness.

In Fig. 6.15(a) $P_d[k]$ is plotted in the *iid* case for various values of ε. When ε is low, the maximum value of $P_d[k]$ is in $k = 0$. As ε increases, the maximum of $P_d[k]$ shifts to the right. The same behavior can be observed when the channel is correlated (see Fig. 6.15(b)). Note the difference in shape for each round between the two cases.

The different behavior when the channel is correlated can also be observed by looking at Fig. 6.16(a), where the mean delivery delay is reported as a function of ε by varying b. In this graph, simulation points are also plotted to test the correctness of the analysis. In Fig. 6.16(b) the mean delivery delay is reported against the error burstiness b by varying ε. The first value of b on the leftmost part of the graph corresponds to the *iid* case ($b = 1/(1-\varepsilon)$). For each ε, the mean delivery delay is monotonically decreasing as a function of b. In other words, the *iid* case is the one characterized by the highest mean delivery delay under all channel conditions.

Fig. 6.17 reports the delivery delay standard deviation, simulation points are reported for comparison. Unlike for the mean delivery time, this metric in the *iid* case can not be interpreted as a bound. In fact, its role with respect to the correlated case depends on both b and ε. Moreover, its behavior is clearly different from that of the other curves. However, this again shows that considering the channel as *iid* can be a misleading assumption when the channel is correlated.

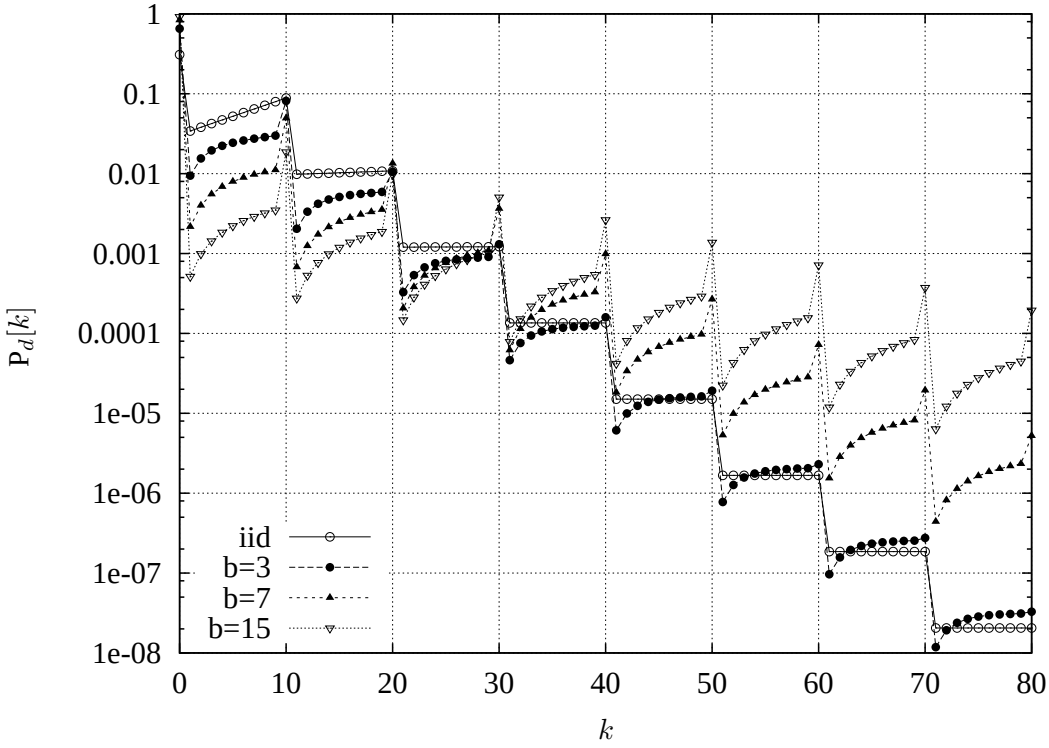
In Fig. 6.18 we report the complementary distribution $ccdf[x]$ by varying b, and plotting the same graph in linear (Fig. 6.18(a)) and in logarithmic scale (Fig. 6.18(b)). Again, from Fig. 6.18(a) it is clear that the *iid* case is not a suitable model when errors are correlated. In particular, we emphasize the higher probability of a rapid delivery (i.e., for low x) in the correlated case, also when the correlation is low, e.g., $b = 3$. Even after a full round ($x = m$) there is a gap in the curves: for instance, $ccdf[x = m = 10]$ in the *iid* case is almost twice that in the correlated case with $b = 15$. Other differences, for higher values of x can be observed from Fig. 6.18(b).

In the next, we focus on the comparison between the exact analysis and the approximate approaches presented in Section 6.4.3. This comparison is reported in Figs. 6.19(a) and 6.19(b) for an *iid* and a correlated channel ($b = 7$), respectively. In these figures, the round trip time has been considered to be large ($m = 30$). In general, the approximate approaches are in excellent agreement with the exact statistics. The only region where the approximations fail is for uncorrelated channel and at low delays. In an *iid* channel, in fact, errors do not occur in bursts, and so the approximation made in Section 6.4.3 that unresolved slots are disposed in a bursty way in this case does not hold. However, the effect of this approximation vanishes very quickly as the delay increases and the approximation becomes very close to the exact curve. In the *iid* case, both BE and SB approaches give the same results. When the channel is correlated (Fig. 6.19(b)), instead, the statistics obtained from BE and SB are in good agreement with the exact curve for any value of the delay. The BE approach overestimates the delivery delay statistics

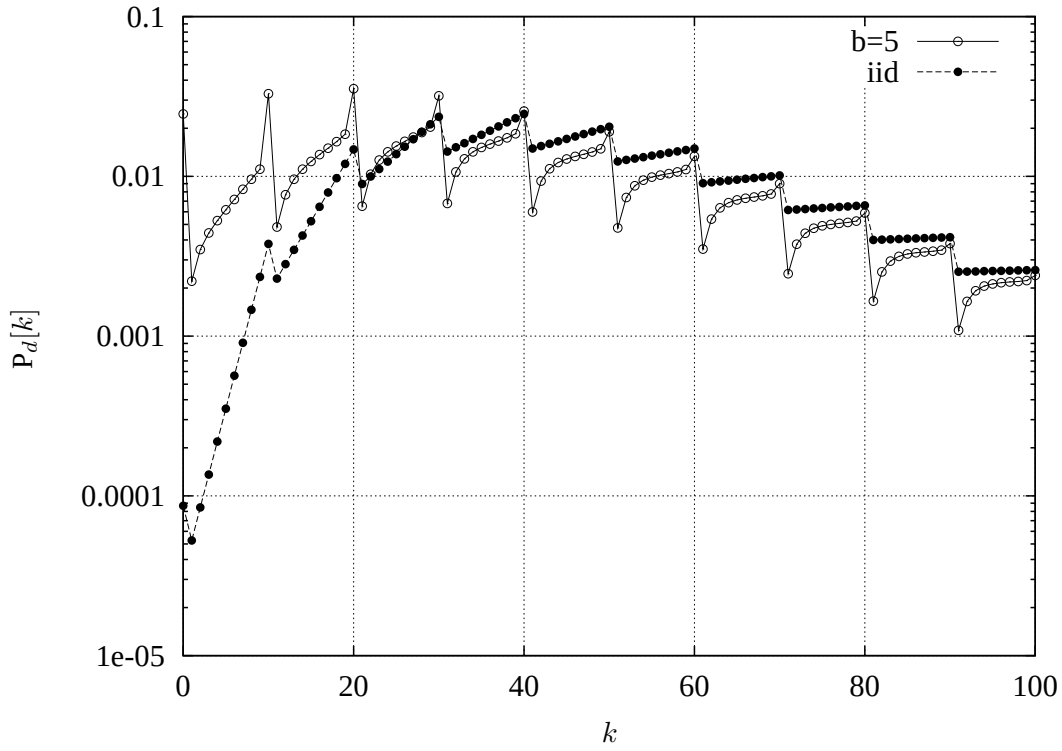
at the beginning of each round. The SB approach, instead, appears to underestimate the exact statistics for any value of the delay. Moreover, the estimate obtained from BE degrades as the delay k increases until, for a very large k, all points in the round are aligned over a straight line. SB, instead, gives a good approximation also for large values of k. The points derived using SB are the closest to the exact curve, except for $k = im$, $i \geq 1$, where the best estimate is given by BE. For what concerns the cumulative delay distribution (Fig. 6.20), BE gives the best estimate for any value of b. Moreover, in the independent case, the burst assumption only results in a small discrepancy regarding the first round. For any other value of k, exact statistics and approximation match almost perfectly. In the correlated case ($b = 7$), the BE approach gives the best complementary cumulative delay distribution estimate. From the obtained results, we can conclude that the estimate of the delivery delay statistics is reasonably accurate for any b, so approximate methods could effectively be used in real systems enabling a fast and less memory expensive computation of the cumulative delay distribution also for large values of m.

In Fig. 6.21(a) we report the delivery delay distribution of an aggregate of K PDUs by varying K. Simulation points are reported to show the correctness of the analysis[24]. In Fig. 6.21(b) we report $P_d[K, s]$ with $K = 14$ by varying the channel correlation. The complementary delivery delay distribution $ccdf[K, x]$ is reported in Fig. 6.22 for $K = 14$, $m = 10$, $\varepsilon = 0.1$ by varying the channel burstiness b. Finally, observe that both shape and behavior of $P_d[K, s]$ are similar to the one characterizing $P_d[s]$; the only differences are: $P_d[K, s]$ is right-shifted by $K - 1$ slots with respect to $P_d[s]$ and it presents a smoother behavior than the one characterizing $P_d[s]$. The shift is due to the minimum time needed to transmit all the K PDUs over the channel (K slots), whereas the smoothness is a property of the discrete convolution product (Eq. (6.43)).

[24] Simulation fails at low ($P_d[s] \leq 10^{-5}$) probabilities due to the rare occurrence of the corresponding events and to the finite simulation time.

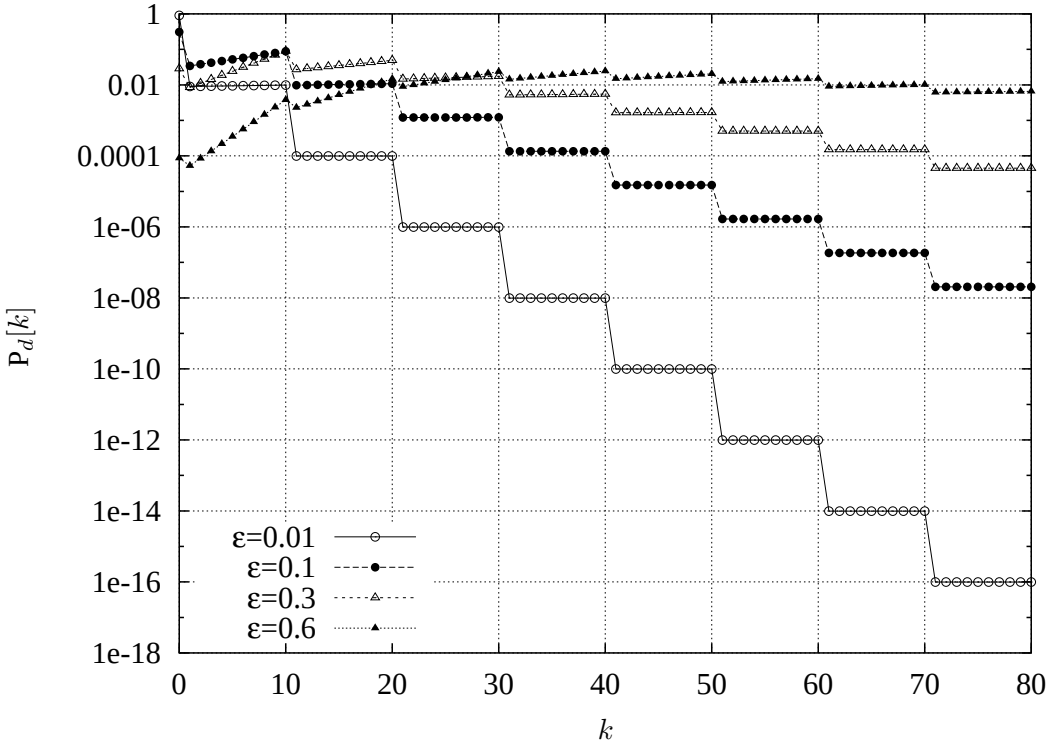


(a) $\varepsilon = 0.1$

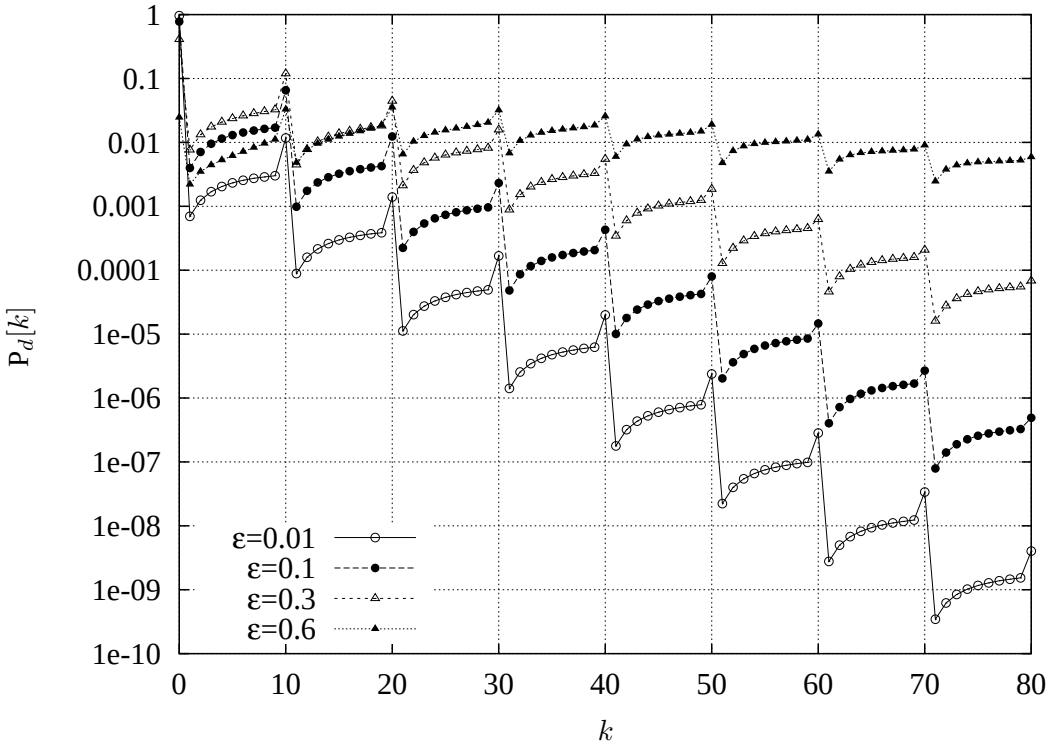


(b) $\varepsilon = 0.6$

Figure 6.14: $P_d[k]$, comparison between the *iid* and a correlated channel with $b = \{3, 7, 15\}$ by considering $m = 10$.

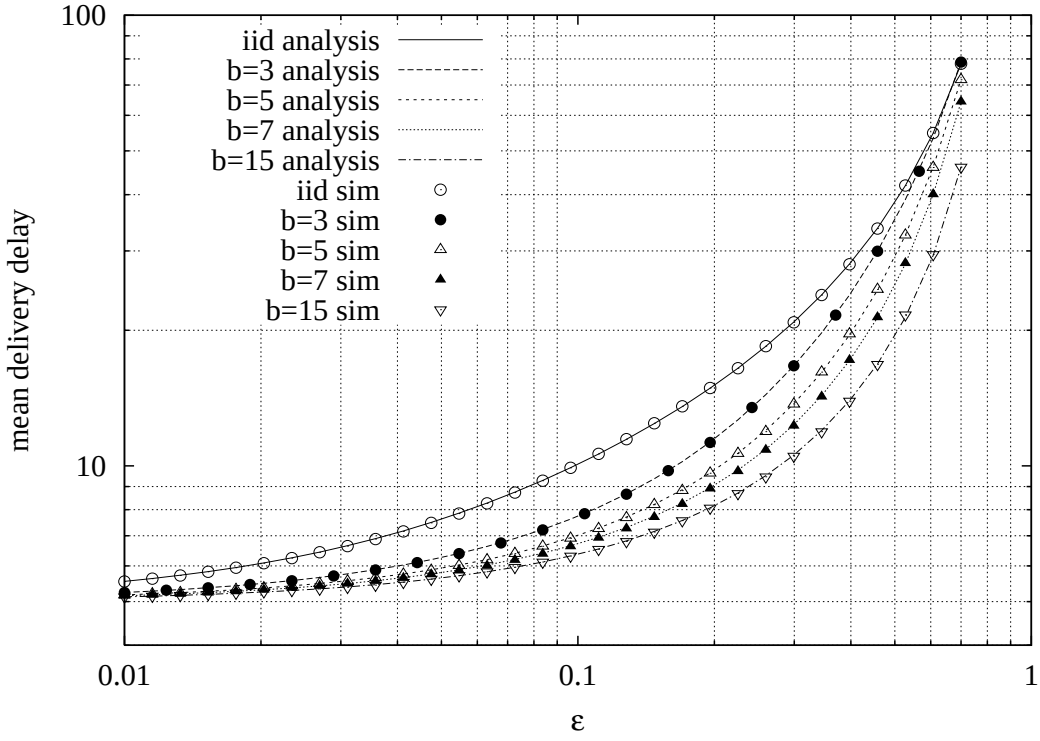


(a) *iid* channel.

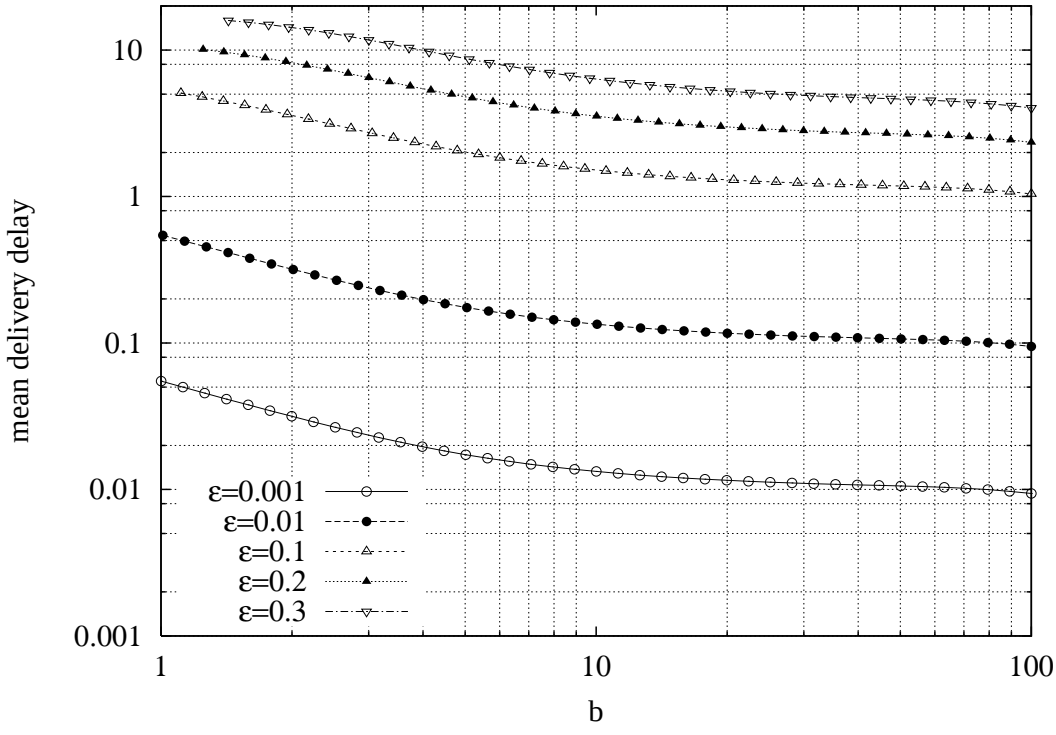


(b) Correlated channel with $b = 5$.

Figure 6.15: $P_d[k]$ as a function of ε by considering $m = 10$.



(a) Mean delivery delay as a function of ε.



(b) Mean delivery delay as a function of b.

Figure 6.16: Mean delivery delay.

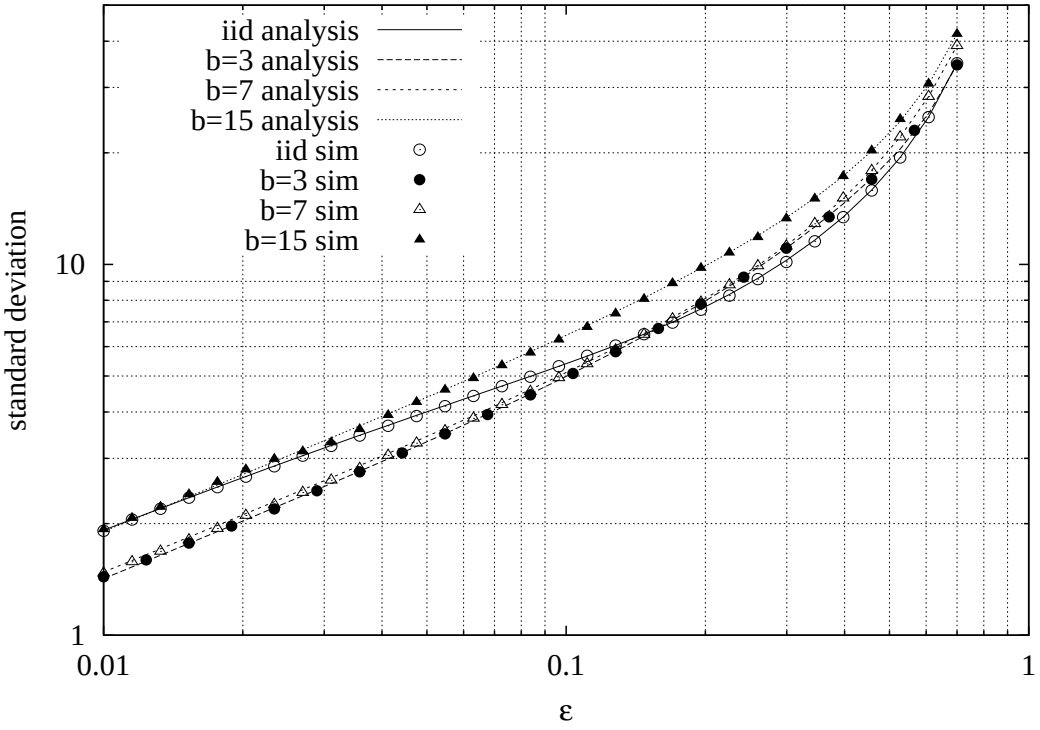
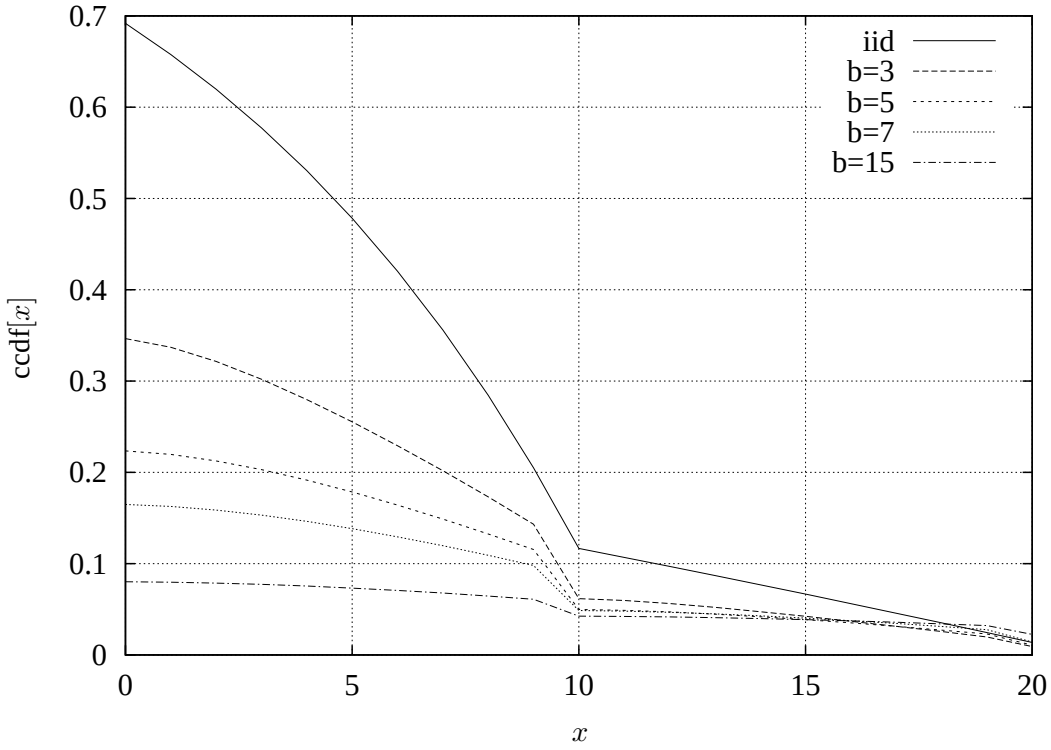
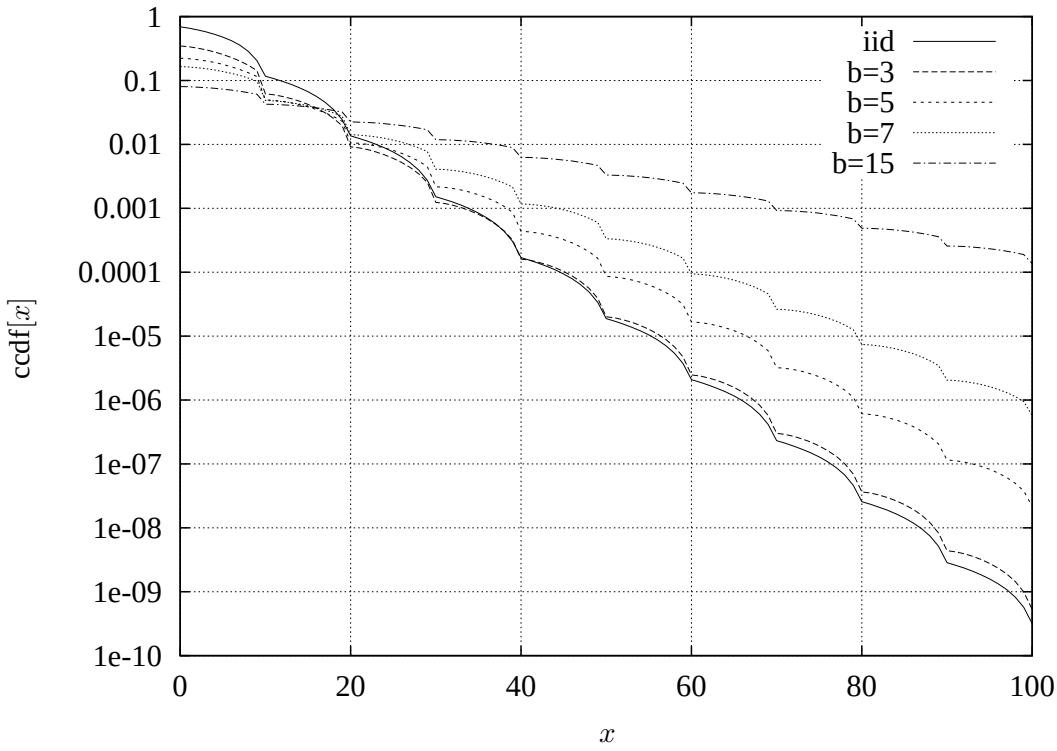


Figure 6.17: Delivery delay standard deviation as a function of ε.

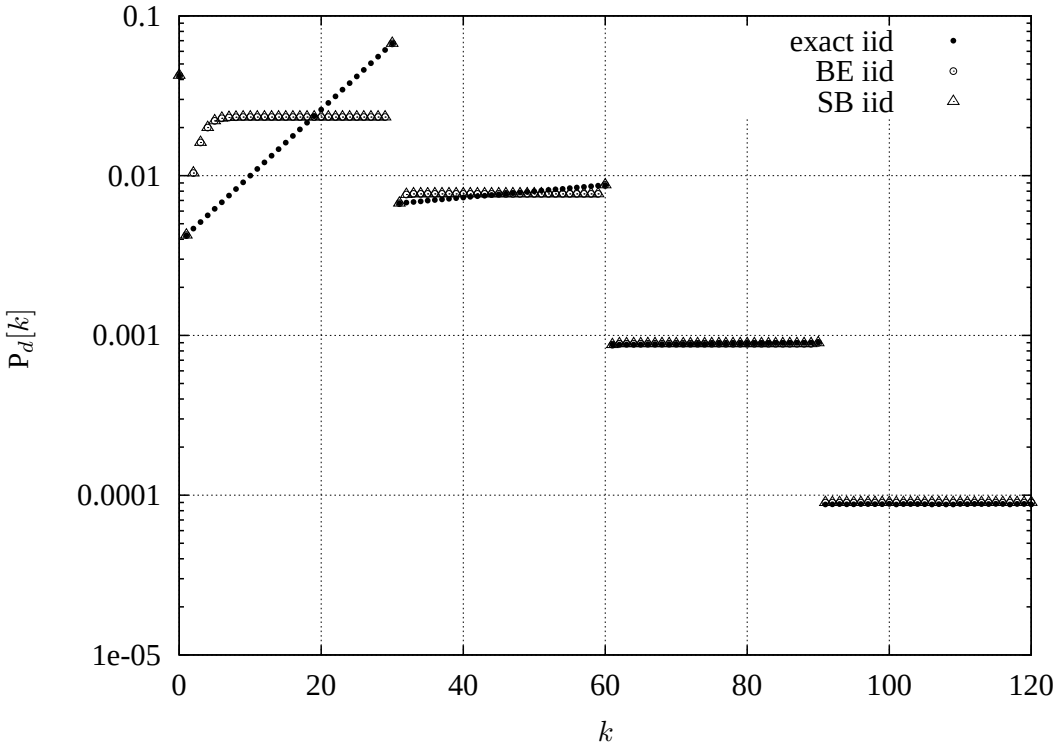


(a) Linear-scale.

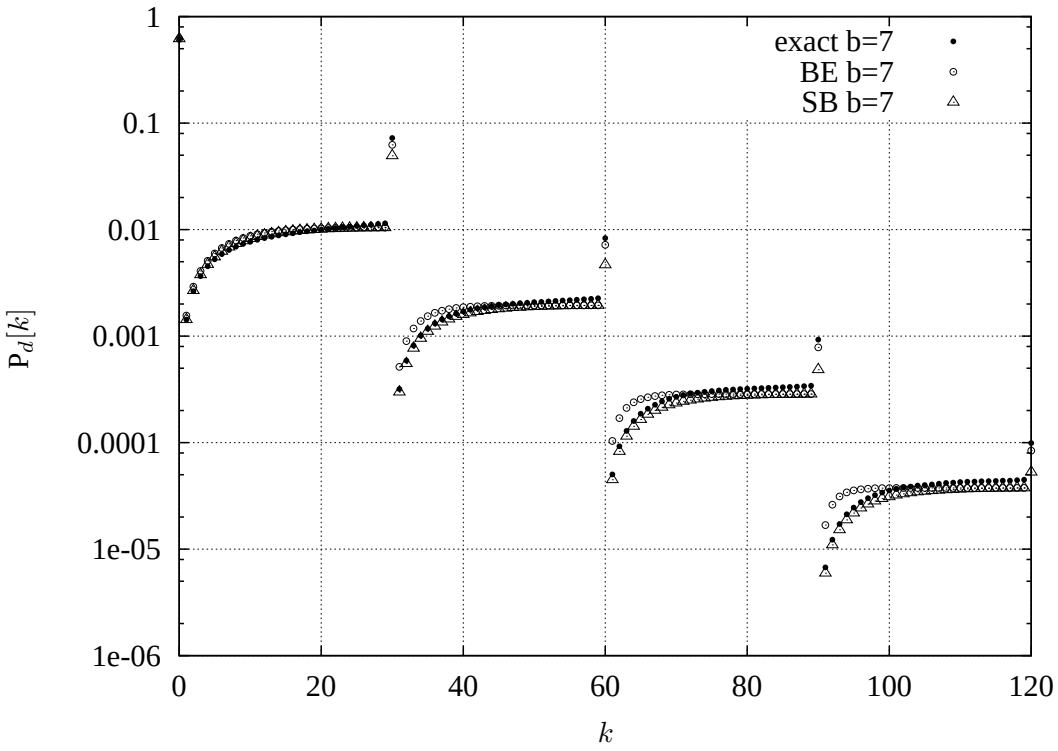


(b) Logarithmic-scale.

Figure 6.18: Cumulative complementary delivery delay distribution, $ccdf[x]$ ($\varepsilon = 0.1$, $m = 10$).



(a) *iid* channel.



(b) Correlated channel with $b = 7$.

Figure 6.19: Comparison between approximation and exact statistics, $m = 30$, $\varepsilon = 0.1$.

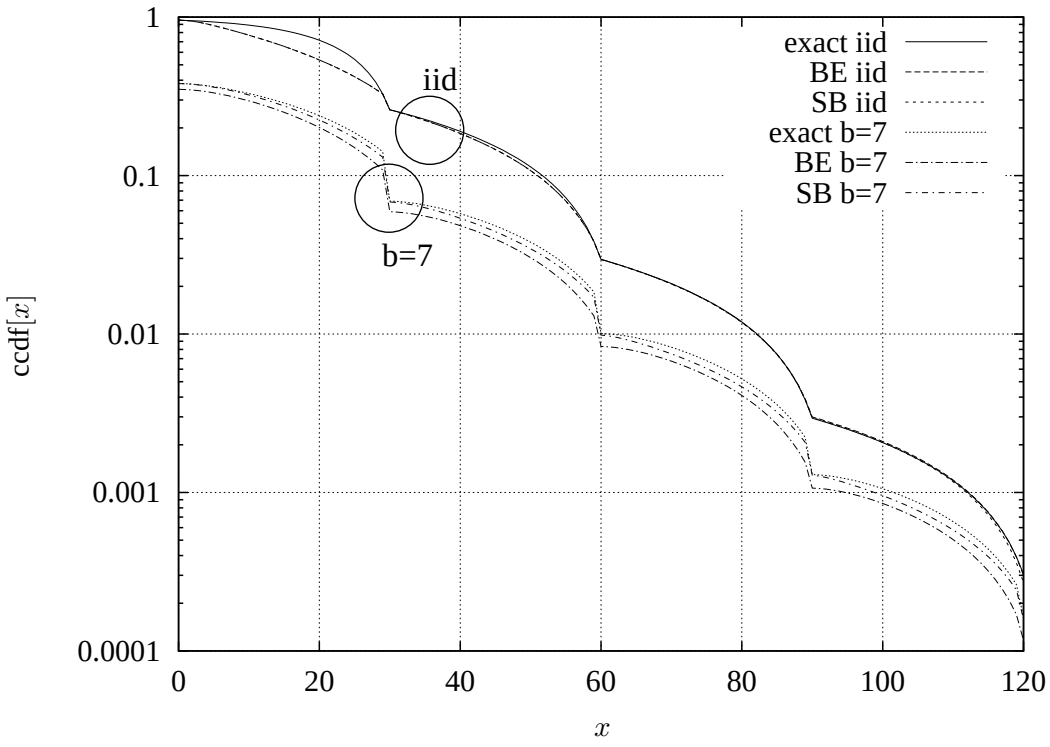
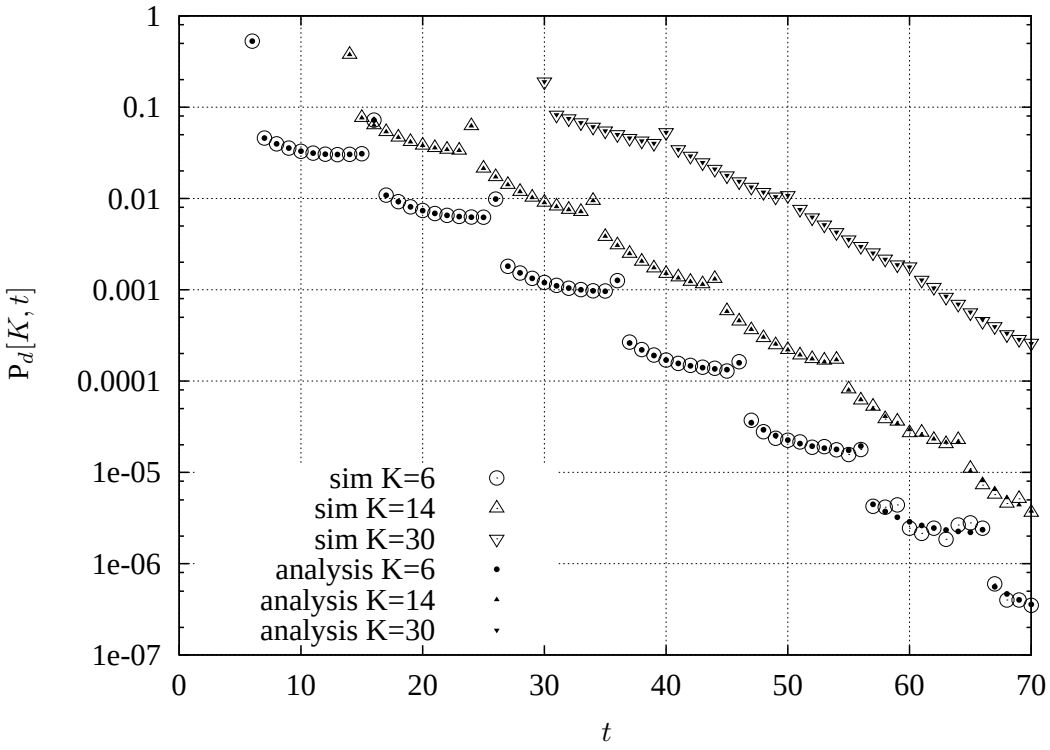
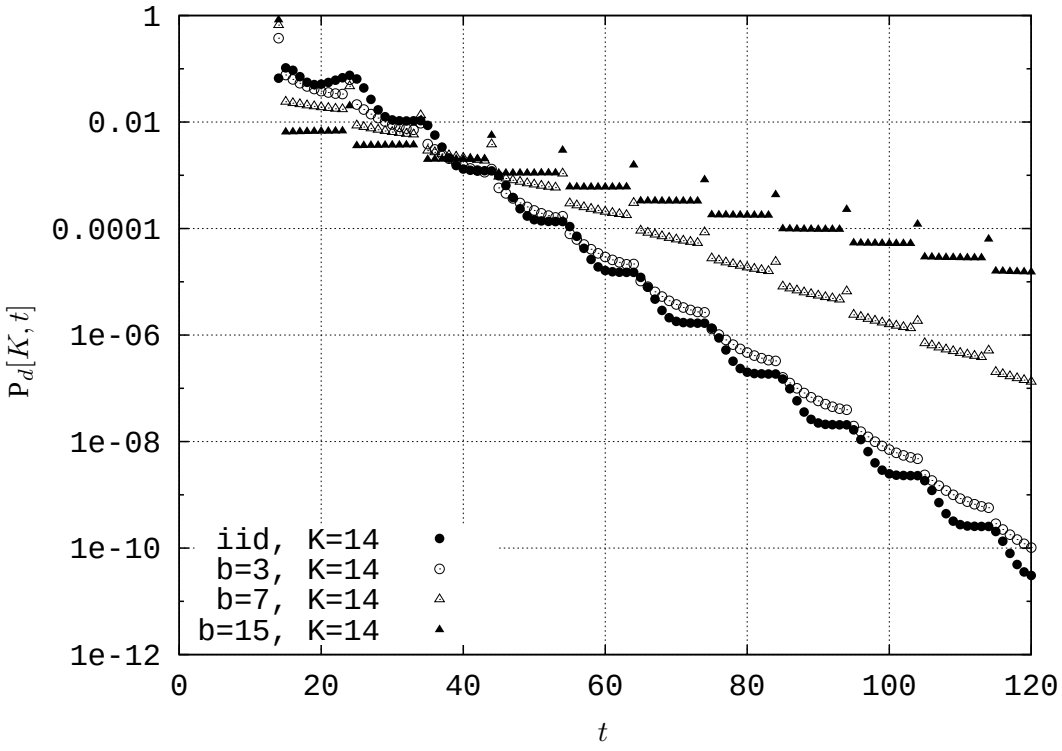


Figure 6.20: Complementary cumulative delivery delay distribution, $ccdf[x]$: comparison between approximate and exact case.



(a) Aggregate statistics by varying K and considering $\varepsilon = 0.1$ and $b = 3$.



(b) Aggregate statistics by varying b and considering $\varepsilon = 0.1$ and $K = 14$.

Figure 6.21: Aggregate delivery delay statistics: comparison between analysis and simulation.

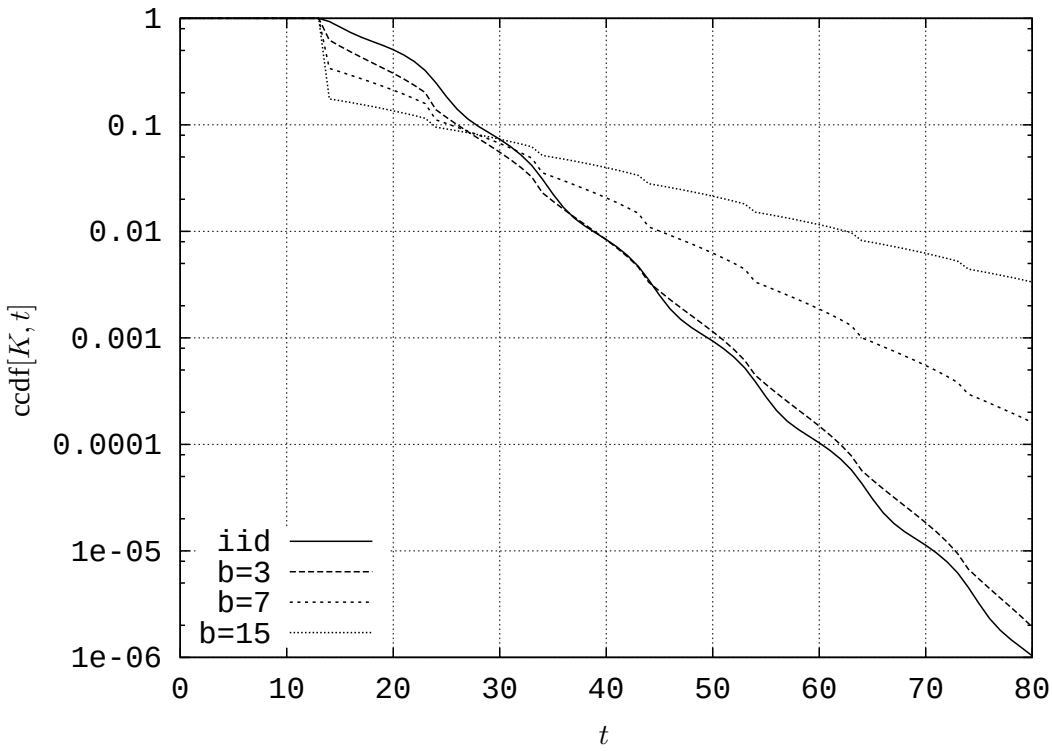


Figure 6.22: Aggregate complementary delivery delay distribution.

6.5 Analysis over an N-State Markov Channel Model

In the remaining of the Chapter, the analysis presented in Section 6.4.2 is generalized in order to account for a generic N-State Markov Error Model. In the next Section 6.5.1, the N-State Markov Chain is introduced first. Subsequently, in Section 6.5.2, some procedures to derive the Markov chain transition probabilities are presented. These techniques account for physical channel characteristics such as user mobility (i.e., Doppler frequency). In Section 6.5.3 the analysis to derive link layer delay statistics is presented in detail. Some preliminary results are reported in Section 6.5.4. Later on, in Section 6.5.5, further results are presented to evaluate the accuracy of the statistics obtained from the Markov modeling approach. Finally, Section 6.6 concludes the Chapter.

6.5.1 N-State Channel Model

Consider an N-State Discrete Time Markov Model, where the slot duration corresponds to the ARQ packet transmission time. We account here for a general Markov model where states $0, 1, \ldots, \nu - 1$ correspond to an error–free transmission of the packets, whereas the remaining states $\nu, \nu + 1, \ldots, N - 1$ mean erroneous transmission. Formally, each state $n \in \{0, 1, \ldots, N - 1\}$ is associated with a packet error probability $P_e[n] = u[n - \nu]$, where $u[\cdot]$ is the unit step, i.e., $u[n] = 1$ if n is greater than or equal to 0, and $u[n] = 0$ otherwise. The model is fully described by the $N \times N$ transition probability matrix $\mathbf{P} = \{p\}_{ij}$, where p_{ij} is the probability that the state in the next slot is j given that the state in the current one is i.

The above model comprises, as a particular case, the widely used [48, 102] K-state model where $\mathcal{S} = \{0, 1, ..., K - 1\}$ is the set of the states, t_{ij} is the transition probability from state i to state j and every state in $\mathcal{S}$ is characterized by a PDU error rate $\epsilon_k \in [0, 1]$, i.e., a PDU transmitted when the channel is in state k is erroneous with probability ϵ_k. This last Markov Chain is completely specified by the pair $(\mathbf{T}, \mathcal{E})$, where $\mathbf{T}$ is the transition probability matrix and $\mathcal{E}$ is the state error probability vector. The model is equivalent to considering $N = 2K$ states (with $\nu = K$) in $\mathcal{S}' = \{0, 1, \ldots, \nu - 1 = K - 1, \nu = K, \ldots, N - 1\}$, where the transition probabilities p_{ij}, $0 \leq i, j \leq N - 1$ are derived as follows:

$$p_{ij} = \begin{cases} t_{xy}(1 - \epsilon_y) & j \in \{0, 1, \ldots, K - 1\} \\ t_{xy}\epsilon_y & j \in \{K, K + 1, \ldots, N - 1\} \end{cases} \tag{6.45}$$

where $x = i - Ku[i - K]$, $y = j - Ku[j - K]$, $0 \leq x, y \leq K - 1$. Moreover, note that in the extended model states $\{0, 1, \ldots, K - 1\}$ are error free, i.e., a PDU is always transmitted correctly in these states, whereas in states K through $N - 1$ PDUs are always transmitted erroneously. In the sequel, we extend this Markov model of the underlying channel to a further Markov chain tracking the ARQ layer queue. Moreover, we show how it is possible to derive a Markov representation of a Rayleigh fading channel. The combination of these techniques allow us to analytically derive higher layer statistics from the physical layer error process.

6.5.2 Derivation of the N-State Markov Model

In this section we report the procedures that we have considered to derive an N-State Markov model of a Rayleigh fading channel (see Fig. 6.23). Several techniques to deal with this problem have been

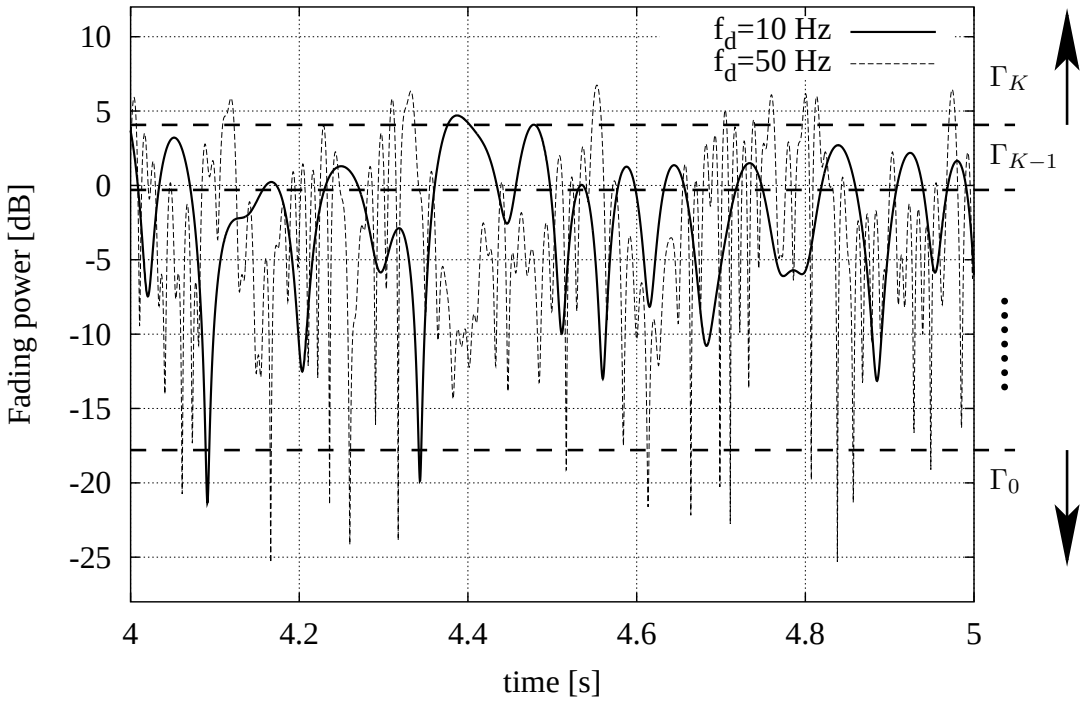


Figure 6.23: Sample behavior for the Rayleigh fading channel.

presented in previous research (see [99, 102, 104], among others). Here, the simple method in [102] is considered first. Later on, and in order to gain some understanding on the impact of the selected method, we propose a new approach which keeps into account the shape of the packet error probability function. This is done with the aim of using a given number of Markov channel states (N) in the most efficient way, i.e., in order to better describe the packet error behavior at the ARQ layer. Since the scope of the following analysis is mainly to propose and validate a framework to obtain link layer delay statistics, we limit here our investigation to these two techniques. In spite of their simplicity, these methods are effective and lead to an accurate description of the ARQ delay statistics.

Let Γ denote the received signal to noise ratio (SNR). The pdf of Γ is exponential as follows [102]:

$$p_{\Gamma}(\gamma) = \frac{1}{\gamma_0} e^{-\gamma/\gamma_0}, \;\; \gamma \geq 0 \tag{6.46}$$

where $\gamma_0 = E[\Gamma]$. Let $0 = \Gamma_0 < \Gamma_1 < \cdots < \Gamma_{K-1} < \Gamma_K = +\infty$ be $K+1$ thresholds for the SNR. The Rayleigh channel is said to be in state $k = 0, 1, \ldots, K-1$ if the received SNR is in the interval $[\Gamma_k, \Gamma_{k+1})$. Moreover, associated with each state there is an error probability ϵ_k that is the PDU error rate experienced in state k. We define $\mathcal{F}(\gamma)$ as the function mapping the instantaneous SNR level γ into the conditional PDU error probability. Once the threshold levels are chosen for every state, the PDU error rate in the generic state k is found as $\epsilon_k = (\int_{\Gamma_k}^{\Gamma_{k+1}} \mathcal{F}(\gamma) p_{\Gamma}(\gamma) d\gamma)/\theta_k$, where θ_k is the steady state probability to be in state k. In this work we assume a $\pi/4$-DQPSK modulation scheme [104], i.e., the bit error probability can be approximated as $\varepsilon(\gamma) \approx (4/3)\text{erfc}(\sqrt{\gamma})$. $\mathcal{F}(\gamma)$ is then derived as $1-(1-\varepsilon(\gamma))^L$,

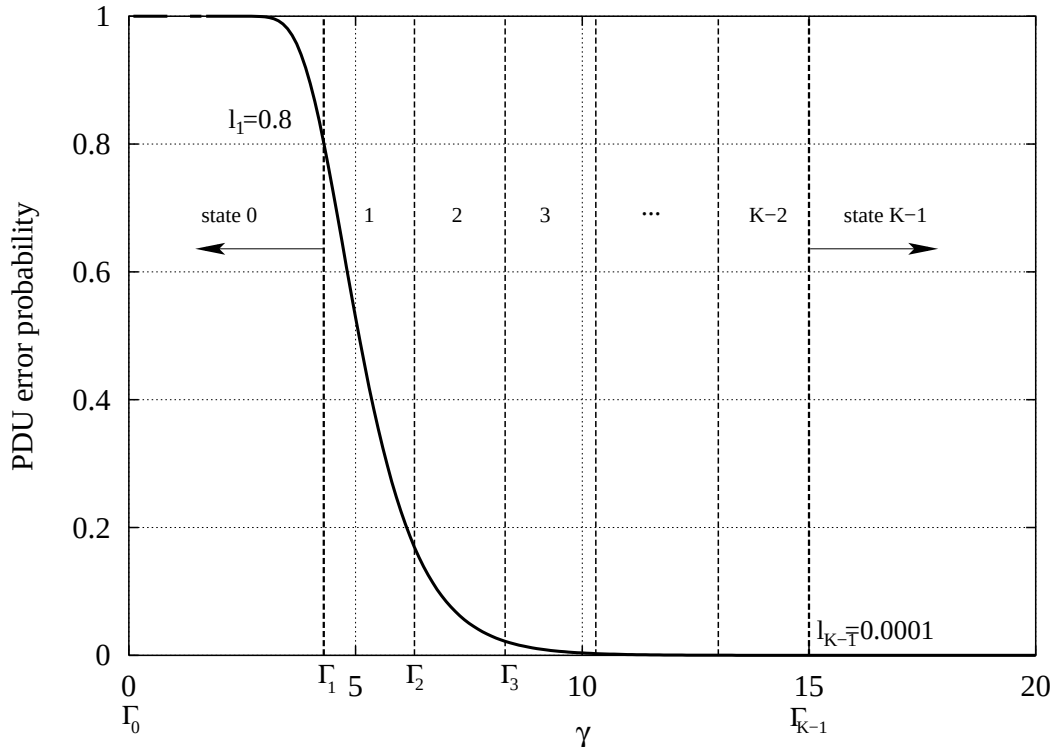


Figure 6.24: SNR values partition method.

where L is the ARQ packet length expressed in bits.[25] The steady state probability θ_k is computed as:

$$\theta_k = \int_{\Gamma_k}^{\Gamma_{k+1}} p_\Gamma(\gamma) d\gamma = e^{\Gamma_k/\gamma_0} - e^{\Gamma_{k+1}/\gamma_0} \tag{6.47}$$

The simplest approach for choosing the SNR thresholds [102] is to consider $\theta_k = 1/K, \forall\ k = 0, \ldots, K-1$. In this case, the threshold levels can be easily estimated by recursively applying Eq. (6.47), given that Γ_0 is known. However, this procedure leads to a rough estimation of the underlying fading process [104] [99]. For this reason, we consider here an improved threshold selection criterion. We first choose two numbers, l_1 and l_{K-1} so that l_1 is close to zero and l_{K-1} is close to 1. Then we choose the first (Γ_1) and the last (Γ_{K-1}) unknown thresholds such that $\Gamma_1 = \mathcal{F}^{-1}(l_1)$ and $\Gamma_{K-1} = \mathcal{F}^{-1}(l_{K-1})$. Once Γ_1 and Γ_{K-1} are known, θ_0 and θ_{K-1} can be evaluated by Eq. (6.47). In this procedure we assign first the states 0 and $K-1$ to the SNR levels corresponding to a PDU error rate that is smaller than l_1 and larger than l_{K-1}, respectively. At this point, we use the remaining $K-2$ states to characterize the SNR interval between Γ_1 and Γ_{K-1}, i.e., where the PDU error rate is in the range $[l_1, l_{K-1}]$. The remaining $K-2$ thresholds are chosen to satisfy $\theta_k = (1-\theta_0-\theta_{K-1})/(K-2) = e^{\Gamma_k/\gamma_0} - e^{\Gamma_{k+1}/\gamma_0}$. In practice, the aim of our method is to assign the remaining states $1, 2, \ldots, K-2$ to the region containing the transition of the function $\mathcal{F}(\gamma)$. A graphical representation of this procedure is reported in Fig. 6.24. Once the

[25] More complicated expressions could be used to account for the use of error correction codes.

thresholds have been computed, the transition probabilities are derived as in [99] according to:

$$t_{ij} = \frac{\int_{\zeta_i}^{\zeta_{i+1}} \int_{\zeta_j}^{\zeta_{j+1}} f_{R_1R_2}(r_1, r_2, \rho) dr_1 dr_2}{\theta_i}$$
$$f_{R_1R_2}(r_1, r_2, \rho) = \frac{4r_1r_2}{\lambda} e^{-(r_1^2+r_2^2)/\lambda} I_0(2\rho r_1 r_2/\lambda) \tag{6.48}$$

where $\zeta_i = \sqrt{\Gamma_i/\gamma_0}$, $f_{R_1R_2}(r_1, r_2, \rho)$ is the bivariate Rayleigh joint pdf [99], $\rho = J_0(2\pi f_d T_p)$ is the correlation of two samples of the underlying Gaussian process that are spaced by T_p seconds, f_d is the Doppler frequency, T_p is the ARQ PDU transmission duration, $J_0(\cdot)$ and $I_0(\cdot)$ are the Bessel function and the modified Bessel function of the first kind and order zero. The Markov Chain $(\mathbf{T}, \mathcal{E})$ derived with this method is then transformed into the $N \times N$ transition matrix $\mathbf{P}$ following the procedure presented in Section 6.5.1. The matrix $\mathbf{P}$ is finally used in the analysis presented in the following Section 6.5.3 to derive the ARQ delay statistics.

6.5.3 Computation of the Delivery Delay Statistics in an N-State Markov Channel

Our goal here is the computation of the delay statistics for a single PDU transmitted using Selective Repeat ARQ. In order to do so, we develop a model which tracks the successful delivery of the PDU of interest (called *tagged PDU*), as well as all previous PDUs. First of all, suppose the tagged PDU is transmitted for the first time in slot $t = m$. This implies that all previous PDUs (i.e., those whose identifier is smaller than the tagged PDU) excluding the $m-1$ PDUs transmitted in slots 1 through $m-1$ have been successfully received, and that in slot 0 a successful transmission occurred (otherwise in slot m we would have a retransmission). Therefore, the tagged PDU is finally released upon correct reception of all PDUs transmitted in slots 1 through m. On the other hand, all PDUs which are transmitted for the first time during slots $t > m$ must have a larger id than the tagged PDU, and therefore do not affect the delivery of the tagged PDU. We can then ignore all future PDU arrivals in our study.

The problem to be solved is therefore to find the time it takes for all PDUs transmitted in slots 1 through m to be eventually received correctly, given that a successful transmission occurred in slot 0. Consider the evolution of the system after slot m. If slot 1 contained an erroneous transmission, a retransmission will be scheduled in slot $m+1$. If this retransmission is successful, then slot $m+1$ will be marked as *resolved*, otherwise it will be marked as *unresolved* (with the effect of a further retransmission in slot $2m+1$ and so on until success). On the other hand, if slot 1 contained a successful transmission, then slot $m+1$ corresponds to a slot which was already resolved, and will be itself resolved (recall that for our purposes future arrivals are ignored, and therefore once a packet is successfully delivered the corresponding slot remains empty). In general, slot $m+k$ (with $k > 0$) will be marked as resolved if either slot k was itself resolved or the channel state in slot $m+k$ is good, and will remain unresolved otherwise. Since in case of transmission failure in an unresolved slot a packet is rescheduled for transmission m slots later, a suitable model to track the relevant events is one in which memory is kept about the resolved/unresolved status of the $m-1$ most recent past slots, i.e., at any given time t, we need to know the status (resolved/unresolved) of slots $t-m+1, t-m+2, \ldots, t-1$. A binary variable is therefore assigned to each slot to carry this information.

Considering t as the current slot, $b_k = 1$ if slot $t-m+1+k$ is still unresolved, and $b_k = 0$ otherwise, for $k = 0, 1, \ldots, m-2$. This string of bits keeps memory of which slots are yet to be resolved, and can also be represented by the integer $i = \sum_{k=0}^{m-2} b_k 2^k$. In addition, we need to specify the status of the current slot, i.e., slot t. In this case a binary variable is no longer sufficient, since we also need to track the channel state, which is necessary to determine the future evolution of successful transmissions. (Note that the Markovian nature of the channel evolution makes it possible to ignore the channel state in slots $t-m+1, t-m+2, \ldots, t-1$ once the channel state in t is known.) For the current slot, many situations are possible, in particular there are three main cases: the channel is in a *good* state, which implies that the slot is resolved (if it was not resolved already, the good channel state makes it resolved now); the channel is in a *bad* state and the slot is resolved (in a previous transmission); the channel state is *bad* and the slot is still unresolved. These three possibilities comprise ν, $N-\nu$ and $N-\nu$ states of the channel, respectively, where ν is the number of good states (with a packet error probability equal to zero) in the Markov chain.

Thus, these $2N-\nu$ states for the last PDU condition will be denoted in numerical order, i.e., by $\{0, 1, \ldots, \nu-1\}$, $\{\nu, \nu+1, \ldots, N-1\}$, $\{N, N+1, \ldots, 2N-\nu-1\}$, respectively. The associated variable will be denoted by ω. Consider now the random process $X(t) = (i(t), \omega(t))$ which jointly tracks slot-by-slot the Markov channel evolution and the status of the m latest slots. According to the above definitions and discussion, this process is a Markov chain. In order to determine the possible transitions and the corresponding transition probabilities, suppose at time t the bitmap which describes the slot status is $\boldsymbol{b} = (b_0, b_1, \ldots, b_{m-2})$, where the most significant bit b_{m-2} denotes the status of the most recent among the past slots. At time $t+1$ this bitmap is clocked one position into the past, i.e., $\boldsymbol{b'} = (b'_0, b'_1, \ldots, b'_{m-3}, b'_{m-2}) = (b_1, b_2, \ldots, b_{m-2}, f(\omega))$, where $f(\omega) = 1$ if $\omega \geq N$ (current slot at time t was still unresolved), and $f(\omega) = 0$ if $\omega < N$. More compactly, in this case $f(\omega) = u[\omega - N]$, where $u[\cdot]$ is the unit step. Regarding the value of $\omega' = \omega(t+1)$, note the following. If, at time t, $b_0 = 0$, the corresponding slot has already been resolved, and therefore $0 \leq \omega' \leq N-1$ according to the channel state y at time $t+1$, so that $\omega' = y$. On the other hand, if $b_0 = 1$, the slot is still unresolved at time t, and therefore we have $0 \leq \omega' \leq \nu - 1$ if the channel at time $t+1$ is good (slot is resolved at this time) and $N \leq \omega' \leq 2N - \nu - 1$ otherwise (slot remains unresolved). In the former case, it is again $\omega' = y$, whereas in the latter $\omega' = y + N - \nu$. Note that given $X(t)$ there are only N possible destinations for $X(t+1)$, since the shift of the bitmap is deterministic and the only random variable is the channel state which can assume N values. More precisely, the transition probabilities are given as follows:

- if i is even (i.e., $b_0 = 0$), then:

$$P[X(t+1) = (i', \omega') | X(t) = (i, \omega)] = \begin{cases} p_{xy} \text{if } i' = \lfloor \frac{i}{2} \rfloor + u[\omega - N] 2^{m-2}, \\ \quad x = \omega - (N-\nu) u[\omega - N], \\ \quad \omega' = y, y = 0, 1, \ldots, N-1 \\ 0 \quad \text{otherwise} \end{cases} \tag{6.49}$$

- if i is odd (i.e., $b_0 = 1$), then:

$$P[X(t+1) = (i', \omega') | X(t) = (i, \omega)] = \begin{cases} p_{xy} \text{if } i' = \lfloor \frac{i}{2} \rfloor + u[\omega - N] 2^{m-2}, \\ \quad x = \omega - (N - \nu) u[\omega - N], \\ \quad \omega' = y + (N - \nu) u[y - \nu], \\ \quad y = 0, 1, \ldots, N-1 \\ 0 \quad \text{otherwise} \end{cases} \tag{6.50}$$

where the use of $\omega' = y + (N - \nu)u[\omega - N]$ in the latter case means that a good channel $0 \leq y \leq \nu - 1$ leads to $\omega' = y$ whereas a bad channel $\nu \leq y \leq N - 1$ leads to $\omega' = y + N - \nu$, $N \leq \omega' \leq 2N - \nu - 1$, i.e., a situation of bad channel and unresolved slot. According to the above rules, the transition probability matrix can be built, which will have only N non-zero entries per row.

In order to find the delay statistics, we proceed as follows. First of all, let us define an appropriate function τ:

$$\begin{gathered} \tau : \mathcal{I}_N^{m-1} \rightarrow \{0,1\}^{m-1} \\ \tau(\boldsymbol{\beta}) = \tau(\beta_0, \beta_1, \ldots, \beta_{m-2}) = (d_0, d_1, \ldots, d_{m-2}) \\ \text{s.t.} \quad d_j = u[\beta_j - \nu] \quad \forall j = 0, 1 \ldots, m-2 \end{gathered} \tag{6.51}$$

where $\mathcal{I}_N = \{0, 1, 2, \ldots, N-1\}$. The meaning of $\tau(\cdot)$ is to transform vectors ($\boldsymbol{\beta}$) of base-N digits into binary digits so that the output digit is 0 if the input digit is less than ν, and 1 otherwise. That is, if $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_{m-2})$ contains the Markov channel states ($\beta_j \in \{0, 1, 2, \ldots, N-1\}$) in $m-1$ consecutive slots, each element of $\boldsymbol{d} = (d_0, d_1, \ldots, d_{m-2})$ is a binary digit equal to 0 for a slot were a successful transmission has occurred (good channel), whereas $d_i = 1$ corresponds to an erroneous transmission in slot i (bad channel state).

Let $\boldsymbol{\Pi} = [\Pi_0 \ \Pi_1 \ \cdots \ \Pi_{\mathcal{M}}]^T$ be a column vector whose $\mathcal{M} = (2N - \nu) \cdot 2^{m-1}$ scalar entries represent the probabilities that the system starts in a given state. This *starting state* is defined as the system state at time $t = m$, when the tagged packet is assumed to be transmitted.

It is easy to see that the value of $i(m)$ (i.e., the resolved/unresolved status for all the slots in slots 1 through $m-1$) only depends on the channel evolution. In this case, the correspondence between PDU status and channel state is one to one, i.e., the $\tau(\cdot)$ function can be used to compute the starting state for the fundamental window. On the contrary, as $t > m$, $i(t)$ is used to track the system memory and the PDUs status depend both on the channel evolution and on the transmission history in previous rounds. For example, the status of a resolved PDU will remain unvaried in the following slots, regardless on the channel evolution. In this case the system is evolved according to Eqs. (6.49) and (6.50).

Let us consider the binary-wise representation of $i(t)$, which we have already named $\boldsymbol{b}$. If $\boldsymbol{\beta}$ is a sequence of values in $\{0, 1, \ldots, N-1\}$ representing the evolution of the channel from slot 1 to slot $m-1$, we recognize that $\boldsymbol{b} = \tau(\boldsymbol{\beta})$. That is, the $\tau(\cdot)$ function is used here to translate the information regarding the channel state in slots 1 through $m-1$ (N possible values) into the corresponding resolved/unresolved status (two possible values).

To evaluate the value of $\omega(m)$ instead, we have to keep in mind that the channel in slot 0 is bound to be error-free. Hence, $\boldsymbol{\Pi}$ is computed as follows:

- if $\omega \in \{0, 1, \ldots, \nu-1\} \cup \{N, N+1, \ldots, 2N-\nu-1\}$:

$$\mathbf{\Pi}_{(i,\omega)} = \sum_{z=0}^{\nu-1} \frac{\pi_z}{\mathcal{S}_\pi} \sum_{\boldsymbol{\beta} \in \mathcal{G}_b} p_{z\beta_0} \left[\prod_{j=0}^{m-2} p_{\beta_{j-1}\beta_j} \right] p_{\beta_{m-2}g} \tag{6.52}$$

- if $\omega \in \{\nu, \nu+1, \ldots, N-1\}$:

$$\mathbf{\Pi}_{(i,\omega)} = 0 \tag{6.53}$$

where $\mathcal{S}_\pi = \sum_{j=0}^{\nu-1} \pi_j$, $g = \omega - (N-\nu)u[\omega - N]$ and $\mathcal{G}_{\boldsymbol{b}} = \{\boldsymbol{\beta} \in \mathcal{I}_N^{m-1} : \tau(\boldsymbol{\beta}) = \boldsymbol{b}\}$ and π_z, $z \in \{0, 1, \ldots, N-1\}$ is the Markov Channel steady state probability of state z.
Let $\boldsymbol{e}_0 = [(i_0, \omega_0)\ (i_1, \omega_1)\ \cdots\ (i_\mathcal{M}, \omega_\mathcal{M})]^T$ be a column $\mathcal{M}$-sized vector of all zeros except for the entries corresponding to states $(i, \omega) \in \{(0,0), (0,1), \ldots, (0, N-1)\}$, that are equal to 1. If $\mathbf{\Phi}$ is the transition matrix of the Markov chain $X(t)$, we determine:

$$\mathcal{P}_c[k] = \mathbf{\Pi}\mathbf{\Phi}^k \boldsymbol{e}_0, \quad k \geq 0\,. \tag{6.54}$$

$\mathcal{P}_c[k]$ is the probability that the delivery delay is less than or equal to k slots. Finally, the delivery delay statistics $P_d[k]$ is determined as:

$$P_d[0] = \mathcal{P}_c[0]\,, \quad P_d[k] = \mathcal{P}_c[k] - \mathcal{P}_c[k-1] \quad \forall k > 0\,. \tag{6.55}$$

Note that $P_d[t]$ is determined by neglecting the propagation delay (that can be approximated as $t_{prop} = m/2$). In fact, what we are obtaining here is the statistics at the transmitter, whereas the actual delay process is evaluated at the receiver. However, since t_{prop} is constant these two distributions are simply the same distribution shifted by this constant factor. For this reason, without loss of generality, t_{prop} will not be considered in the sequel.

6.5.4 Results for the Delay Statistics over an N-State Markov Channel

In this section, we report some examples for the delivery delay statistics and we discuss the goodness of a Markov channel model in the approximation of the ARQ packet delay statistics in a Rayleigh fading channel. In the following Section 6.5.4, some examples for the link layer delivery delay statistics over a fading channel are reported first. Later on, in Section 6.5.5, the accuracy of the Markov modeling approach will be quantitatively discussed considering the effect of the Doppler frequency f_d and of the number of Markov channel states N.

As a first result, in Figs. 6.25 and 6.26, we report the delivery delay statistics considering $f_d = 10$ Hz and $f_d = 80$ Hz, respectively. In both graphs, the statistics obtained by simulation is compared against the two threshold selection methods, i.e., the equal probability method ($\theta_k = 1/K$) and the refined procedure presented in the previous section. For both $f_d = 10$ Hz and $f_d = 80$ Hz the equal probability criterion leads to a rough estimation of the underlying fading process and to a poor approximation of the delay distributions as well. As a second observation, one can note that a Markov approximation of the actual channel error process is not able to perfectly match the real statistics. Even though the two distributions are in good agreement, the one derived using the Markov model can not reproduce the behavior characterizing the actual distribution. This is, indeed, a limitation of the Markov model that,

even when a large number of states is considered, does not perfectly fit the actual fading process statistics. However, it is worth noting that the fading is a complex process that we are trying to approximate using a relatively simple model. In this sense, the obtained statistics is reasonably close to the real ones. In general, the match is good for the first three retransmission rounds, then the two distributions differ and the Markov model approach fails to reproduce the behavior of the actual distribution. It is also worth noting that the substantial differences observed in the autocorrelation function between the Markov and the fading model [99] are not observed for the delay performance of the ARQ protocol, where indeed the analytical performance is quite close to simulations.

In Figs. 6.27 and 6.28 we report some curves to qualitatively discuss[26] the dependence on the number of states of the Markov model (N). In general, the fit between simulation and analysis improves as N increases; this is particularly true when the Doppler frequency is low (i.e., $f_d = 10$ Hz). However, as the number of states increases ($N = 30$ in the presented graphs) the resulting statistics remains unchanged and no further improvements are observed. Moreover, taking as an example the $f_d = 10$ Hz case, an increase in N leads to a better approximation for the first part of the statistics, whereas the match is degraded elsewhere. In order to obtain better results it would be interesting to investigate how the statistics improves considering a different approach to derive the Markov chain. For instance, in [20] the authors considered the fading derivative as an additional dimension for this purpose. In that paper, they proved that this method can somehow reproduce the oscillatory behavior of the autocorrelation function. In [19] some results are reported concerning the mean throughput value of the SR ARQ protocol. Further investigations on how these techniques can improve the ARQ delivery statistics are left for future research.

6.5.5 Evaluation of the Accuracy of the Link Layer Delay Statistics

In this Section, we present some results on the accuracy of the delay statistics derived in Section 6.5.2. We refer here to $P_d^{an}[t]$ as the statistics derived analytically, i.e., to the distribution derived by means of the Markov modeling approach, whereas we refer to $P_d^{sim}[t]$ as the delay distribution which has been directly measured by simulating the Selective Repeat ARQ algorithm over a Rayleigh fading channel (a Jakes simulator has been used at this point to reproduce the Rayleigh fading behavior [61]).

In order to weight the difference between these two statistics, we consider here the well known Kullback Leibler distance (see [31], p.18). The Kullback distance $D(p \parallel q)$ between the two generic distributions $p(x)$ and $q(x)$ is a measure of the inefficiency of assuming that the exact distribution is $q(x)$ when the true distribution is $p(x)$. For example, if we knew the real distribution of the random variable ($p(x)$), then we could construct a code with average length $H(p)$ (where H is the entropy associated to the distribution p). If, instead, we used the code to describe the approximate distribution $q(x)$, we would need $H(p) + D(p \parallel q)$ bits on average to describe the random variable. The Kullback distance arises as an expected logarithm of the likelihood ratio of the two distributions:

$$D(p \parallel q) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \tag{6.56}$$

where $\mathcal{X}$ is the (common) dominium of the distribution functions.

[26] A quantitative comparison is reported later on in Section 6.5.5.

In Figs. 6.29 and 6.30 we report the distance between the two distributions $P_d^{an}[t]$ and $P_d^{sim}[t]$ as a function of the Doppler frequency (f_d) and of the number of states used to build the Markov chain approximating the underlying Rayleigh channel. In Fig. 6.29, the distance is plotted as a function of the number of states N that are used to build the Markov chain. A set of curves is drawn here by varying the Doppler frequency f_d. It is interesting to observe that the distance metric has a minimum, i.e., an optimal number of states for the construction of the Markov chain exists. Moreover, in all cases we considered this optimal number of states is upper bounded by 10. Hence, more than 10 states are useless to increase the accuracy of the estimated delay statistics[27]. In the figure, the independent (*iid*) error case (where a single state is used to describe the error process) is also reported for comparison. It is clear that the independent error assumption provides poor results, even at high Doppler frequencies. We can also observe that the Gilbert Elliot model (four state Markov chain), for $f_d \geq 30$ Hz can provide fairly good approximations. It however fails to approximate the delay statistics over correlated channels ($f_d = 10$ Hz). Over such channels, in fact, the accuracy of the Gilbert model (four states) is one order of magnitude worse than the precision achievable with $N = 6$.

As a last result, in Fig. 6.30, the distance metric is plotted as a function of f_d reporting both the SNR partitioning methods considered in this paper. The independent error case is also reported for comparison. Again, we can observe how the independent error assumption[28] fails to accurately model the underlying channel. From that figure, it is also evident that the equal probability method ($\theta_k = 1/K$) provides a rough approximation for the delivery delay statistics. The problem, in such case, is that the state partitioning is not designed taking into consideration PDU error probability function aspects (Fig. 6.24). Finally, we observe that all the curves obtained from the proposed partitioning method converge to a unique point as f_d increases. For such f_d values ($f_d \geq 100Hz$) a large N does not seem to affect the precision of the obtained statistics.

6.6 Conclusions

In this Chapter, a complete analytical framework for the link layer delay statistics evaluation has been presented. Both independent and correlated error models have been considered. For every channel model, exact approaches have been presented first. Subsequently, and where possible, approximated analyses have been devised to decrease the computational complexity so as to allow the delay statistics computation directly at battery operated and, therefore, energy limited mobile terminals. In the last part of the Chapter, delay statistics are obtained for the generic N-State Markov Channel Model, which, in most cases can give an accurate and simple characterization for the underlying physical fading channel process. This last set of results has provided useful insights on the accuracy of the channel characterization by means of such Markov models. In particular, it has been found that the Markov approach is sufficiently accurate to track delay statistics at higher layers.

The research presented in the Chapter has been motivated by the need for a deeper understanding of the impact of link layer solutions on higher layer applications and by the need for delay estimates to be

[27] Note that these conclusions hold with the specifi c SNR partition method adopted here. It is not excluded that other partition methods can lead to a continuously increasing accuracy as a function of the number of states N.

[28] A single state is used to model the Rayleigh fading channel.

used to empower link layer algorithms. Such estimates, in fact, can be incorporated into runtime algorithms at the link layer in order to adapt the protocol behavior to the time–varying user requirements as well as to channel/network time–varying QoS conditions.

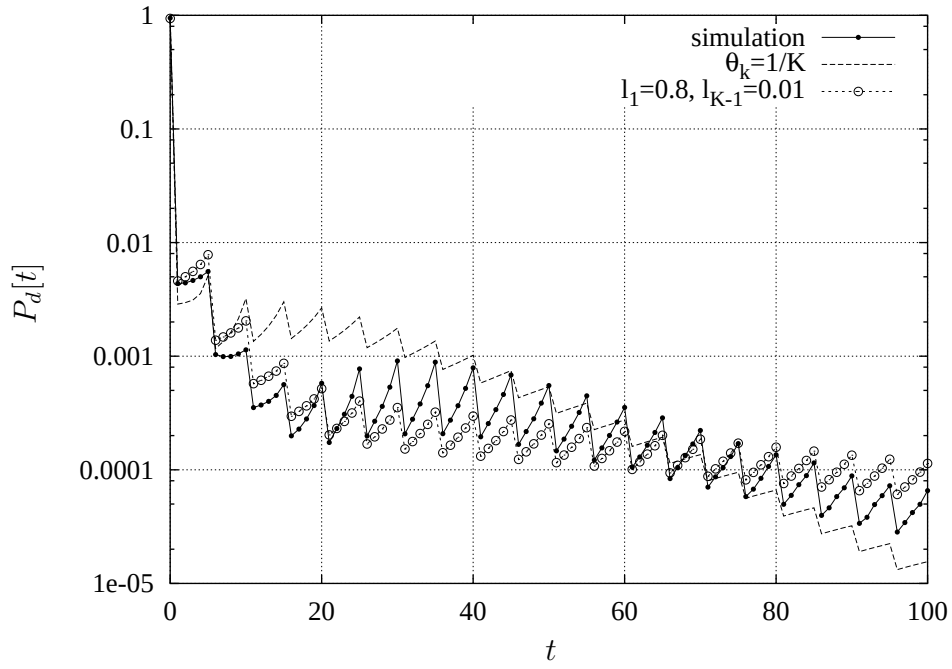


Figure 6.25: Delivery delay statistics: comparison between Markov Channel analysis and Rayleigh channel simulation with $L = 360$ bits, bit rate 1024 Kbps, $f_d = 10$ Hz, $f_d T_p = 0.00343$, $m = 5$, $N = 10$.

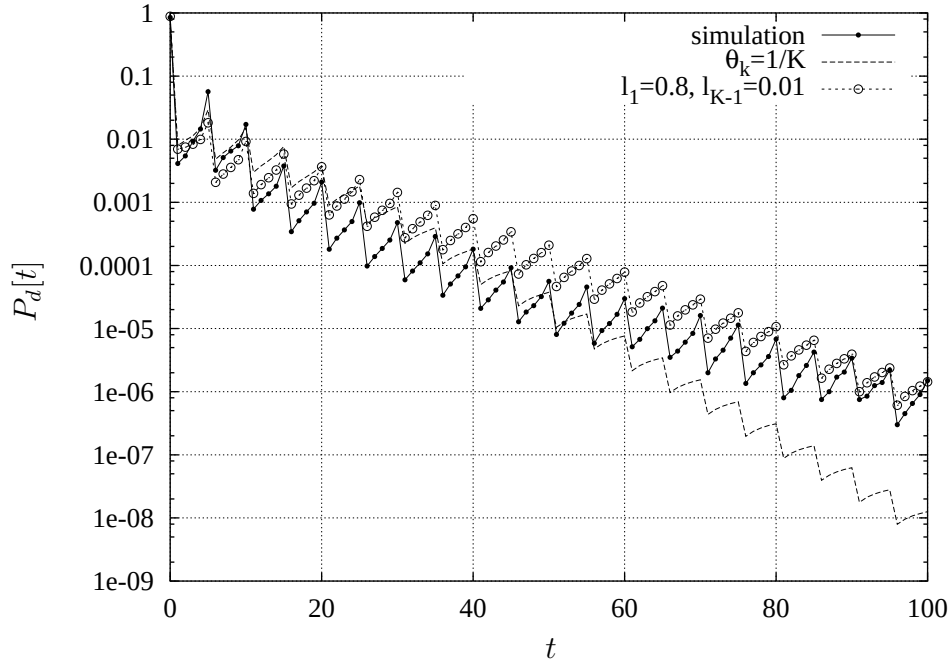


Figure 6.26: Delivery delay statistics: comparison between Markov Channel analysis and Rayleigh channel simulation with $L = 360$ bits, bit rate 1024 Kbps, $f_d = 80$ Hz, $f_d T_p = 0.0274$, $m = 5$, $N = 10$.

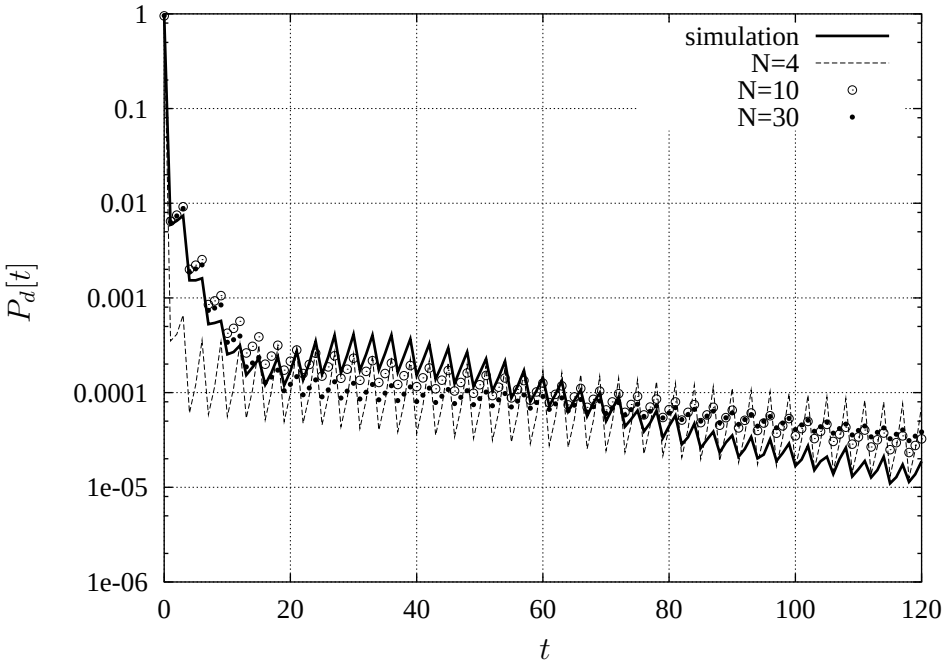


Figure 6.27: Delivery delay statistics: comparison between Markov Channel analysis and Rayleigh channel simulation as a function of N with $L = 360$ bits, bit rate 1024 Kbps, $f_d = 10$ Hz, $m = 3$.

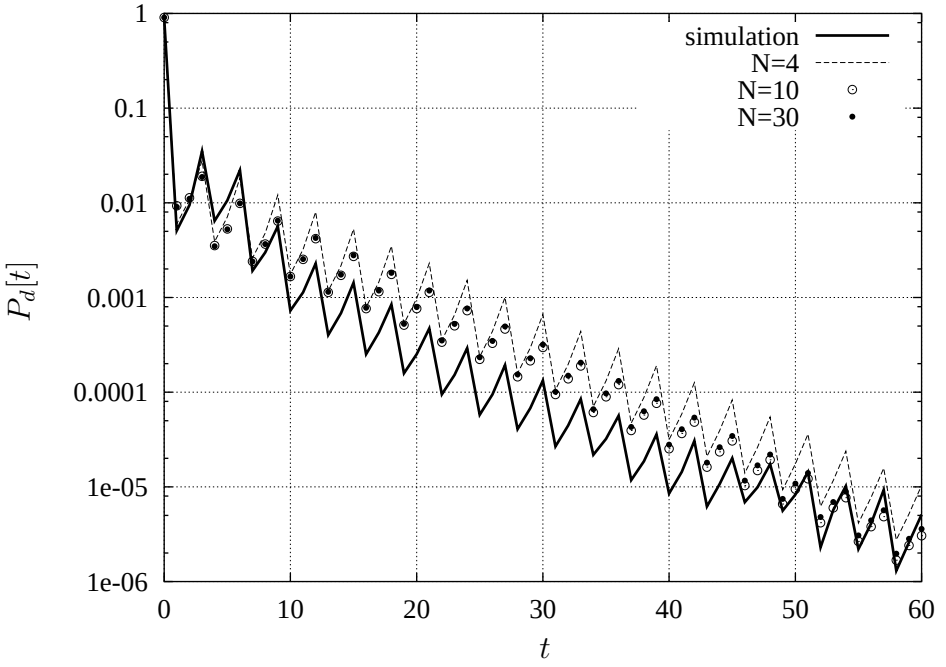


Figure 6.28: Delivery delay statistics: comparison between Markov Channel analysis and Rayleigh channel simulation as a function of N with $L = 360$ bits, bit rate 1024 Kbps, $f_d = 100$ Hz, $f_d T_p = 0.0343$, $m = 3$.

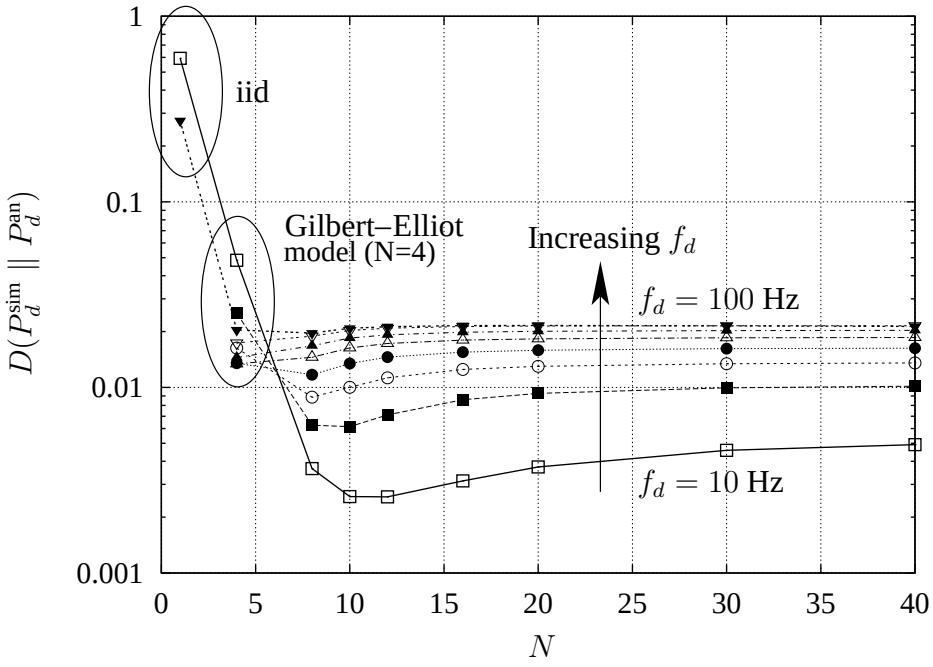


Figure 6.29: Kullback Leibler distance between the distributions obtained by simulation ($P_d^{\text{sim}}[t]$) and analysis ($P_d^{\text{an}}[t]$) as a function of the number of states N, by varying the Doppler frequency f_d.

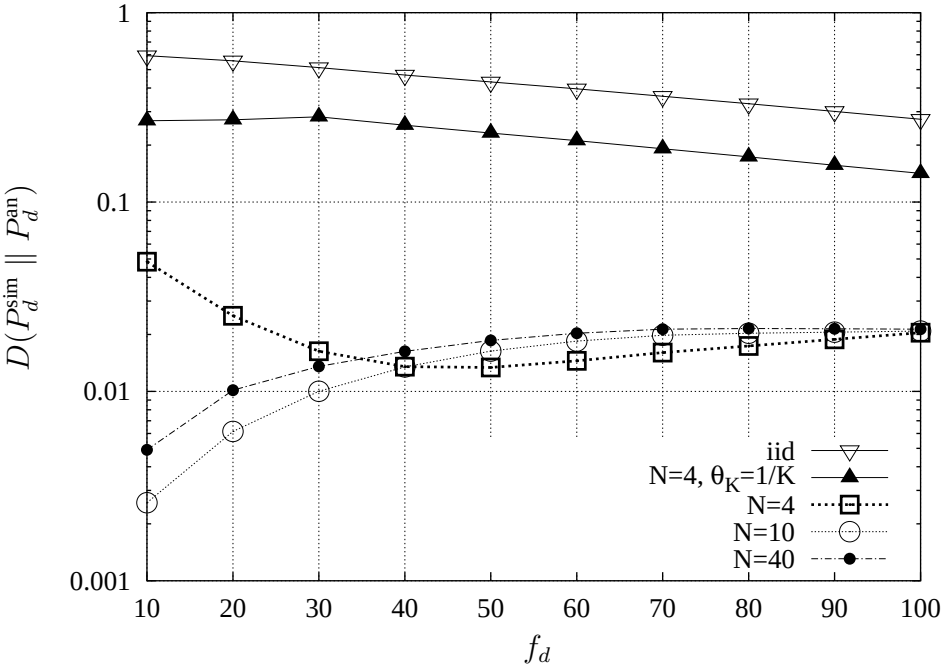


Figure 6.30: Kullback Leibler distance between the distributions obtained by simulation ($P_d^{\text{sim}}[t]$) and analysis ($P_d^{\text{an}}[t]$) as a function of the Doppler frequency f_d, by varying the number of states N.

Chapter 7

Error Control Techniques for Efficient Multicast Data Delivery in 3G Cellular Systems

In this Chapter we go further with respect to the previously illustrated research, by focusing on ARQ error control algorithms for the transmission of a multicast flow over a common channel in a 3G cellular system. In this case, for efficiency reasons the multicast flow is sent through a common downlink channel to all interested users. Subsequently, every user asks for the retransmission of lost data to the base station controller (by means of NACK messages and through, for instance, a dedicated uplink channel[1]). In such a scenario, ARQ algorithms have to deal with possibly multiple retransmission requests. This fact, in general implies a substantial degradation of both channel efficiency and delay, since the single forward channel has to be used to satisfy multiple retransmission requests. As will be highlighted in this Chapter, plain ARQ algorithms are ineffective in such a case and new solutions have to be found to improve the error correction capability in the multi–user scenario. The main focus of the research presented in this Chapter is to propose and validate such new solutions.

We focus on a single 3G cell, where some mobile terminals are interested in the reception of the same data stream. For efficiency reasons, the multicast paradigm is used to carry such flow to these users. The problem to be solved here is how to achieve an efficient (in the sense of bandwidth usage) local recovery of the errors occurring in the transmission to the interested users.

The chief aim of the research that will be presented throughout this Chapter is to investigate how ARQ and FEC techniques can be used and possibly combined together (hybrid ARQ schemes, HARQ) to combat wireless channel errors and achieve good forward channel utilization performance. Particular attention will be paid to the characterization of the achievable quality in the video streaming multicast case, by also giving useful insights regarding the dimensioning of the application play–out buffer.

[1] In order to improve the feedback channel efficiency, an uplink shared channel can also be used. However, feedback considerations are not explicitly addressed here, since in this study we are mainly interested in the evaluation and optimization of forward channel metrics. The feedback channel optimization is taken into account indirectly throughout the study. A deeper investigation is left for future research.

7.1 Introduction

In this Chapter, we deal with the Multicast delivery problem in 3G Cellular Systems. Let us assume that a multicast flow, which has been transmitted by a server placed somewhere in the Internet or in the 3G wired network, has to be delivered to a set of interested multicast users[2] in a 3G cell, where a serving base station is responsible for the delivery of that flow to all the interested multicast users in its coverage area. Since our main focus here is on the delivery that is taking place in this last part of the connection, i.e., in the wireless link between serving base station (BS) and interested users, we disregard the possible error occurring over the wired part of the connection. We also disregard additional delays and out-of-ordering. Instead, at the BS, we consider an error-free and timely-delivered multicast flow. Now, the problem to be solved is to deliver such a flow over the last wireless link (BS $\rightarrow$ users) in the most efficient way, i.e., keeping into account both channel efficiency and delay requirements.

In the remaining part of this document, several possible solutions to cope with this problem are investigated in detail. In particular, after presenting the system model in Section 7.2, in Section 7.3 we will introduce a few error control algorithms, that, thanks to some forward error correction capabilities, are able to outperform standard ARQ (Automatic Retransmission reQuest) solutions. The performance of these schemes is investigated later on in Section 7.5, whereas, in Section 7.6 some additional results will be reported for the video streaming case. Finally, Section 7.7 concludes the Chapter.

Throughout this Chapter, important aspects such as channel efficiency (throughput), delivery delay, play–out buffer dimensioning and achievable video quality will be explicitly and quantitatively addressed, by leading to important guidelines regarding the practical implementation of multicast delivery solutions. Particular attention will be devoted to the multicast video streaming case, which is expected to be one of the most important services to be supported in fore coming 3G networks.

7.2 System Model

We consider here a 3G cellular system, where W–CDMA is used as the radio interface (see, for instance, UMTS as a possible system using such technology [1] [69] [83]). The service area is composed by 9 hexagonal cells, where a base station is placed at the center of each cell and a given number of users is considered to be able of moving inside the coverage area. Propagation phenomena are modeled through standard techniques, by considering log-normal slow fading, fast fading[3] and path loss[4]. A simple power control algorithm has been implemented following the basic algorithm which can also be found in [83]. In practice, a SIR target (SIR_{th}) is assigned to each user in the system. Such a value has been chosen according to quality requirements depending on channel coding and modulation. In more detail, the downlink transmission power is dynamically varied by a constant multiplicative increase/decrease factor. Considering a specific user, its downlink transmission power is increased when the instantaneous SIR

[2] That, for this purpose, are joining the specific multicast group to which the flow is addressed.

[3] Which has been simulated here by means of the well known Jakes fading simulator [61].

[4] The simple Hata model is considered here. In this model the attenuation A as a function of the distance r is modeled as $A(r) = Kr^{\beta}$, where K and β are constants.

(Signal to Interference Ratio) is below SIR_{th}, and it is increased elsewhere. Minimum and maximum transmission powers are $P_{min}^{downlink} = -15$ dBW and $P_{max}^{downlink} = -5$ dBW, whereas the power factor used in the following results is $\Delta = 0.5$ dB. For what concerns channel coding and interleaving, we consider here a convolutional half rate Viterbi decoder[5] operating over an interleaving interval TTI of 80ms.

A first set of users $N_{DCH} = 200$ is communicating through a Dedicated CHannel (DCH) whose bit rate is 30 Kbps (corresponding to a physical Spreading Factor of $SF = 128$). These users are placed randomly at the beginning of the simulation and are moving following a pseudo-linear mobility model[6]. The speeds for such users has been uniformly chosen in the set $\{0, 7, 49\}$ Km/h. For these users, a simple handover algorithm is run, where the best base station[7] is always assigned as the serving one. The power control procedure is dynamically executed for each user as explained above.

A second set of users N_{CCH}, is receiving the multicast flow through a Common downlink CHannel (CCH) whose bit-rate is indicated here as B_r.[8] These users can also be on the move, but their serving base stations remain unchanged. Let us better clarify this point. CCH users are randomly placed at the beginning of the simulation, shadowing and path loss are chosen according to the log-normal and the exponential model, respectively, and are kept constant for the whole simulation time. Subsequently, in order to emulate some degree of mobility, their Doppler frequency f_d is selected, but without changing their spatial coordinates. By this way, we are able to control their fast fading as they were on the move *but* without reflecting it into a change of their spatial positions. Therefore, it is possible to investigate multicast delivery algorithms (which is, in fact, the main focus here) disregarding the multicast handover management that, by itself, constitutes a problem to be properly handled. The common channel power has been fixed to a constant value $P_{CCH} = P_{max}^{downlink}$ throughout the whole simulation.

For illustration purpose, in Figs. 7.1 and 7.2, we report some statistics on the link layer packet error model for the common channel (multicast) users. The following parameters have been considered for the CCH users: $B_r = 120$ Kbps, $f_d \in \{2, 40\}$ Hz, link layer packet length $PDU_l = 360$ bits[9]. In Fig. 7.1, we report the mean burst length (b) as a function of the packet error probability ε, whereas in Fig. 7.2 its cumulative distribution is plotted by varying the Doppler frequency (f_d). The independent error case ($b = 1/(1 - \varepsilon)$) is also reported for comparison. Due to the limited burstiness values in both scenarios, some performance improvements appear to be possible at the link layer through an adequate design of packet–based FEC coding schemes. Some algorithms which are exploiting such concepts will be presented in the following Sections.

[5] A software implementation of the Viterbi algorithm has been used here to derive bit error traces from physical SIR traces. Encoder and decoder have been confi gured according to the specifi cation which can be found in [1].

[6] The trajectories are obtained from a composition of linear movements, during which the users are moving along straight lines.

[7] The one heard with the strongest signal.

[8] Here, we refer to the useful bit rate at the link layer level, i.e., after physical layer and MAC processing (such as coding and rate matching).

[9] Inclusive of header, payload and CRC checksum.

7.3 Link Layer Algorithms for Multicast Streaming

When the multicast flow is transmitted by means of a common channel, different users are in general characterized by independent channel error processes. This is very important since it can make traditional link layer (LL) retransmission approaches inefficient as the multicast group size (N_u) increases[10].

For illustration purpose, a fully reliable multicast service is considered at this point. In that case, a packet needs to be retransmitted if at least one user has not correctly received it. Moreover, an erroneous packet needs to be retransmitted until all the users in the multicast group have received it correctly. The problem is that, as N_u increases, the probability that at least one user needs a retransmission at a given time increases as well and, as a consequence, the forward link throughput is heavily degraded by retransmissions, while the available bandwidth for new transmissions becomes very low.

As N_u increases simple ARQ may not be sufficient to cope with this problem, because it simply retransmits the lost packets whenever a retransmission request (Not Acknowledgment, NACK message) arrives at the base station, but it does not account for the possibly different error processes that, at every receiver, are affecting the transmitted packet. To cope with this problem, we propose to exploit packet-based FEC techniques directly at the link layer level. A rich literature can be found on the topic [56] [81] [79] [80].

By this approach, some error recovery is performed directly at the receiver side. This is realized by pro-actively adding some redundancy into the multicast flow. This redundancy can be independently exploited, at each receiver, to recover from losses. If local recovery succeeds, no retransmissions are required and the forward bandwidth can be utilized to allocate new packet transmissions. In more detail, the multicast flow at the link layer of the sending base station is divided in transmission groups (TG) of K packets each. Then, each group is passed to a packet–based encoder where H redundancy packets are generated for every TG. Finally, the whole FEC block composed by $K + H$ packets is sent over the channel. At the receiver side, every user can exploit the redundancy packets by recovering up to H erroneous or lost packets, in any order. Different coding schemes can be used for this purpose, Reed–Solomon and Tornado deserve a particular attention [24, 90]. However, in this work we are mainly interested in discussing the effectiveness of these coding strategies without going into code implementation details. Moreover, in the following, we assume that, in the first encoding phase, it is possible to generate a large (but finite) number of redundancy packets for each FEC block. These packets will be transmitted on-demand as incremental redundancy during the error recovery algorithm. The main benefits of this approach are:

- **Improved transmission efficiency**: A single parity packet can be used to repair the loss of any packet in the TG. This means that *a single parity packet* can repair the loss *of different data packets at different receivers*. This fact is extremely useful since different receivers are in general affected by independent error processes.

- **Improved scalability in terms of group size**: In ARQ schemes the sender needs to know the se-

[10] It is worth observing here that N_{CCH} is the total number of CCH users in the system, while N_u is the number of CCH users in a single cell ($\approx N_{CCH}/9$). Performance in the following Sections will be reported referring to this second quantity.

quence number of each lost packet. Instead, using parity packets for loss repair, the sender only needs to know the maximum number of lost packets by any receiver but not their sequence number. So, the feedback is reduced from per–packet feedback to per–TG feedback. In fact, depending on the number of lost packets, a given number of new redundancy packets, obtained from the original K packets, can be transmitted over the channel (incremental redundancy) so that the original data is recovered if at least K packets are correctly received among all the received packets for the TG (N FEC block packets plus incremental redundancy packets).

- **Improved feedback channel performance**: Thanks to the pro-actively added redundancy, some error recovery is possible at the user terminals, without the need for retransmissions. As a consequence, the number of acknowledgment messages that the users are sending back to the base station is considerably decreased. This FEC beneficial effect has been largely investigated in previous work (see [74] as an example), where techniques to limit the ACK collision problem have been considered. These results and algorithms still apply in our scenario as well. Results on ACK collision avoidance will not be explicitly considered in our contribution where the emphasis is mainly put on the proposal and the presentation of schemes for the multicast delivery in 3G networks. Performance investigation is then given, in some detail, considering forward channel metrics. This has been done to prove the effectiveness of such a FEC based transmission scheme even over Wideband-CDMA channel traces.

Now, we describe three hybrid ARQ (HARQ) algorithms that can be used at the LL level to improve both efficiency and delay performance with respect to a plain Selective Repeat ARQ. The hybrid schemes are described in the following [41]:

HARQ1: The sender collects groups of K packets and then feeds them into the packet–based encoder obtaining H redundancy PDUs so that a total number of $N = K + H$ PDUs is available for each TG, as discussed above. The entire TG (N PDUs) is sent to all receivers over the CCH channel. Note that at the sender side some amount of redundancy (H PDUs) is pro–actively added to the forward multicast flow. As said above, this redundancy can be independently exploited by each multicast receiver to recover the original K packets when the number of lost PDUs a TG is less than or equal to H. When a multicast receiver can not decode a TG (less than K PDUs are correctly received out of the $K + H$ transmitted), it sends back to the transmitter a NACK message for that TG including the TG identifier. At the sender side, all NACKs for the same TG are collected and the entire TG (N PDUs) is retransmitted if at least one multicast user required its retransmission. As will be shown in the next section this solution gives good performance in terms of delay, but it is not very efficient in terms of throughput. At least one drawback is present in that scheme. In fact, *a priori* retransmitting all the PDUs in a TG may not be a good choice, since a large amount of redundancy is always retransmitted without checking for the real needs of the users. In some cases, this will impair the channel efficiency without leading to any improvement.

HARQ2: As in the previous scheme, at the sender side TGs of N PDUs each (K data packets plus H

redundancy PDUs) are sent first. Then, each receiver checks for errors in each TG and replies accordingly. If a receiver detects less than K correct PDUs for a TG it sends back to the sender a NACK including the TG identifier. The sender collects incoming NACKs and, if the number of collected NACKs for a given TG is greater than zero the following procedure is activated:

- The K original PDUs included in the erroneously received TG are fed again to the packet–based encoder to obtain $\xi \geq 1$ redundancy packets, i.e., ξ redundancy packets for that TG that are however different from all redundancy packets previously transmitted.

- The ξ new redundancy PDUs (*incremental redundancy*) are sent over the CCH channel.

In practice, we have that the number of retransmitted redundancy packets for each received NACK is always equal to ξ. In this scheme, each receiver collects all the received packets for a TG, i.e., the N PDUs sent in the first TG transmission plus the, say R, redundancy PDUs sent in the following retransmissions (triggered by NACKs). The original K PDUs in a TG can be recovered if the number of correct PDUs, N_{ok}, over the $N + R$ PDUs is greater than or equal to K. This scheme, when ξ is set to one, tries to maximize the channel efficiency. In fact, at each retransmission request the minimum amount of redundancy (one packet) is retransmitted to all users (only one PDU), thereby limiting as much as possible the probability to retransmit useless redundancy packets. However, if at least one user needs more than one new PDU to obtain the K original data packets he will require a further retransmission. For this reason this scheme is also characterized by the largest delay for the correct delivery of a TG.

HARQ3: At the sender side the TG of N PDUs (K data packets plus H redundancy PDUs) is sent first. Each receiver checks for errors in each TG and replies accordingly. In this scheme incremental redundancy is also used. To better explain how the algorithm works suppose that, in addition to the N PDUs in the first transmission, R redundancy PDUs have already been sent over the channel for a given TG. In this case, each receiver checks for the number of correctly received PDUs (N_{ok}) among the $N+R$ PDUs sent. Let us refer to a given user $i \in \mathcal{N}$, where $\mathcal{N}$ is the set of multicast users in the cell. If $N_{ok}(i) \geq K$ the original K PDUs can be obtained and the TG is correctly received by user i. Otherwise, if $N_{ok}(i) < K$, user i sends back a NACK including the *TG identifier* and $r_i = K - N_{ok}(i)$, i.e., the minimum number of new redundancy PDUs needed for the correct decoding of the K data PDUs at that terminal. The sender collects incoming NACKs and computes $R_{max} = \max_{i \in \mathcal{N}}(r_i)$. Then, $R = R_{max}$ new redundancy PDUs are encoded for that TG and are transmitted over the CCH channel. This procedure is repeated until all users in $\mathcal{N}$ are able to correctly decode the K data PDUs, i.e., when $N_{ok}(i) \geq K$ $\forall i \in \mathcal{N}$. This last scheme tries to achieve a trade–off between channel efficiency and delay. In fact, for each retransmission request, the minimum number (channel efficiency $\uparrow$) of PDUs needed to ensure that all multicast receivers will be able to recover the original K packets is sent (delay $\downarrow$). Note that R_{max} new PDUs guarantee the successful decoding of a TG by all receivers only if no channel errors occur, or if channel errors are such that each receiver is able to decode at least r_i PDUs out of the R_{max} transmitted.

7.4 Performance of Error Recovery Algorithms over an Independent Channel

In this Section, we report some preliminary results to test and compare the algorithms introduced above. A simple independent channel model will be used in this Section, due to the possibility to obtain analytical expressions from which useful insights can be derived. Accurate channel traces, obtained from the simulation tool explained in Section 7.2, will be considered later on in Section 7.5 and 7.6.

The performance metrics that we are looking at in this Section are: the *channel efficiency* (or CCH channel throughput) and the *higher layer packet delivery delay*. The higher layer packet is the LL SDU unit, that is the packet passed to the LL level to be processed and sent over the channel by higher protocol levels. With the term SDU delay we refer to the time elapsed between the transmission of the first LL PDU composing one SDU to the instant in which the full SDU has been correctly received by every user in the cell[11].

In the results discussed in the following Sections, we consider a LL logical bit rate of $B_r = 120$ Kbps (Spreading Factor equal to $SF = 16$) a LL round trip time (RTT) of 220 ms (this the maximum value for the LL RTT in a 3G network and it is due to the large interleaving depth of 80 ms) and a LL PDU length of 360 bits. With these values, we have that about 77 PDUs are transmitted in a LL RTT. Moreover, we consider a fixed LL SDU packet length of 500 bytes which is a typical value for the mean frame length of video streaming flows [43].

As a first result, in Figure 7.3 we report the SDU complementary cumulative delivery delay distribution (ccdf) for the HARQ1 and a standard ARQ algorithms. The number of multicast users is set to $\{10, 100\}$ and the PDU error probability is considered to be equal to $p = 0.1$ for every user. From this figure we can observe that as the group size increases (N_u), hybrid ARQ largely limits the SDU delays. In particular, the SDU delivery statistics, after a certain point (in $d \leq 800$ ms), starts decreasing very suddenly. The statistics regarding the simple ARQ, instead, is shifted to the right without any shape change. This means that hybrid ARQ techniques are more robust with respect to an increasing number of multicast users in the system.

In Figure 7.4, we report the mean SDU delivery delay, defined as the mean time needed to correctly transmit a full SDU to all users in the multicast group as a function of the multicast group size, N_u, considering that all users are characterized by an independent channel with the same PDU error probability $p = 0.1$. From this figure it can be observed that in the simple ARQ case, the behavior of the mean SDU transmission time is logarithmic in N_u (approximately equal to $420 + 90\ln(N_u)$). In other words, the SDU delivery delay tends to infinity as N_u increases. Also in the hybrid ARQ case the SDU delivery delay tends to infinity as $N_u \to \infty$, but here the delay increases very slowly with N_u. This is an important fact since it considerably improves the system scalability in terms of multicast group size. As expected, the HARQ1 scheme is the one with the lowest delay, HARQ2 is the one with the longest and HARQ3 is

[11] In order delivery is considered here, i.e., a correctly received SDU at the receiver side can be passed to higher layers only when all the SDUs with lower identifier have been successfully transferred.

a compromise between HARQ1 and HARQ2.

As the last result in this section, we focus on the channel efficiency η, which is defined as the number of PDUs correctly transmitted over the common channel divided by the total number of transmitted PDUs:

$$\eta = \lim_{t \to +\infty} \frac{N_{ok}(t)}{N_{tot}(t)} \tag{7.1}$$

where $N_{ok}(t)$ and $N_{tot}(t)$ are the number of correctly received and the total number of transmitted PDUs in the interval $[0, t]$, respectively. In $N_{ok}(t)$, each packet is counted only once, even if multiple copies could have been sent during retransmissions. Redundancy packets are not accounted for in N_{ok}, whereas they are counted in N_{tot}. In the following, we present a simple way to analytically derive the channel efficiency under the independent channel error assumption. We label as $i \geq 0$ the transmission round for a FEC block, where $i = 0$ correspond to the first transmission, whereas $i \geq 1$ is used to track the following retransmission rounds. Moreover, we consider that, at every retransmission, a constant number ξ of redundancy PDUs is encoded for the TG and sent over the channel. Note that, the efficiency of the HARQ3 algorithm is equal to the efficiency of HARQ2 when $\xi = 1$ since, in both schemes, the minimum amount of redundancy is sent to accomplish error recovery. The only difference among these two algorithms is that retransmissions are distributed differently. However, since the channel is independent, the retransmitted PDU's position is irrelevant, while their number is the only quantity affecting η. We have verified by simulation that the HARQ1 performance can be well approximated by HARQ2 with $\xi = N$. Now, we introduce some quantities that will be used in the analysis: N_u is the number of multicast users, i.e., the number of users served by the common channel and p_i is the PDU error probability for user $i \in \{1, 2, \ldots, N_u\}$. The probability that a TG is correctly received by every user in a transmission round up to and including round $j \geq 0$ can being computed as:

$$P[\leq j] = \begin{cases} \prod_{i=1}^{N_u} (1 - p_i^{j+1}) & \text{ARQ} \\ \prod_{i=1}^{N_u} (1 - f[p_i, j]) & \text{HARQ} \end{cases} \tag{7.2}$$

where:

$$f[p, j] = \sum_{e=N+\xi j-K+1}^{N+\xi j} \binom{N + \xi j}{e} p^e (1-p)^{N+\xi j-e} \tag{7.3}$$

where the function $f[\cdot, \cdot]$ is used to compute the probability that less than K PDUs have been correctly received among the $N + \xi j$ PDUs transmitted for the TG up to and including round j. This quantity corresponds to the probability that additional redundancy PDUs are still needed to correctly decode the TG after j transmission rounds, i.e., that at least a further retransmission round is needed. The average number of retransmissions is given by:

$$E[\text{retx}] = \xi \sum_{j=0}^{+\infty} (1 - P[\leq j]) \tag{7.4}$$

where $\xi = 1$ for the ARQ scheme, whereas $\xi \geq 1$ for HARQ. Finally, the channel efficiency can be derived as:

$$\eta = \begin{cases} \dfrac{1}{1 + E[\text{retx}]} & \text{ARQ} \\ \dfrac{K}{N + E[\text{retx}]} & \text{HARQ} \end{cases} \tag{7.5}$$

Figure 7.5 gives the channel efficiency as a function of N_u considering $p_i = 0.1, \forall i \in \{1, 2, \ldots, N_u\}$. In the figure, we report the approximated throughput for the HARQ1 scheme proposed above ($\xi = N$) together with other results obtained by diminishing the number of packets sent during every retransmission ($\xi = 7$ and $\xi = K$). As observed above, the HARQ2 scheme is characterized by the highest throughput, whereas HARQ1 is characterized by the worst performance, i.e., the optimization of the delay in the HARQ1 scheme comes at the cost of sending a large amount of incremental–redundancy packets. This increases the probability of recovering a TG in a short time, but at the same time decreases the throughput, since more unnecessary packets are sent at each retransmission request. All the hybrid algorithms outperform simple ARQ as N_u becomes greater than three. On the other side, as the multicast group is smaller than $N_u = 3$, the pro–actively added redundancy in HARQ1, HARQ2 and HARQ3 (*a priori* data protection) is more channel consuming than performing retransmissions only (simple ARQ). However, it is worth noting that hybrid ARQ schemes lead to a higher throughput as N_u increases and that, for large N_u values, they still achieve an acceptably high channel efficiency, whereas simple ARQ in such a case has very poor throughput performance. Another interesting result is that the performance of scheme HARQ3 lies on the throughput upper bound (that, as observed above, is given by scheme HARQ2). This scheme may be a good candidate to be effectively used for multicast data delivery since it is characterized by a good channel efficiency and also its delay performance is not too much worse with respect to the HARQ1 algorithm.

7.5 Results concerning Buffer Requirements and Channel Efficiency

In this Section, the performance of the algorithms proposed above will be addressed in detail focusing on accurate 3G W–CDMA channel traces. In this case, channel errors tend to be correlated and, as a consequence, the proposed FEC solutions may become less effective. In order to cope with this problem or, at least, to limit its impact on the overall performance, we introduce a further interleaving directly at the link layer (Section 7.5.1). Essentially, we try to break the error bursts, by spreading them over several different FEC blocks. This will improve the coding efficiency at the cost of some additional complexity. The interleaving procedure is illustrated in the next Section, whereas the performance evaluation is reported later on in Section 7.5.2.

7.5.1 Link Layer Packet Level Interleaving

At the LL of each serving Base Station (BS), the following packet-based FEC coding algorithm is applied. The PDU flow is grouped in *Transmission Groups* (TG) of K packets, then each TG is fed to a FEC encoder to obtain H LL parity packets. Finally, both the K and the H packets are put together to form a N packets FEC Block (FB) [56]. These H pro-actively added redundancy packets can be exploited by each user to locally recover from up to H erroneous packet in every FB, in any order.

As highlighted in previous research [81], packet-based FEC techniques are very effective to offer low residual PDU error rates to multiple users receiving the same data from a common transmission channel. However, the effectiveness of such techniques decreases as the link layer error burstiness increases. When error bursts are too long, the added redundancy is likely lost and is useless in recovering from errors. In this case, the redundancy only wastes the available channel resources. To overcome to this fact, we apply a matrix interleaving on the FB flow prior to its transmission over the channel. Let us refer to the interleaving buffer size (expressed here in number of LL PDUs) at the LL as B. Then, as reported in Fig. 7.6, PDUs are first disposed in a $I \times N$ matrix[12], where $I = B/N$ is the interleaving depth. Thereafter, link layer PDUs are sent reading the matrix by columns, i.e., the transmitted sequence will be: $\{1, N+1, 2N+1, \ldots, (I-1)N+1, 2, 2N+2, (I-1)N+2, \ldots, N, 2N, \ldots, IN\}$. Note that all PDUs belonging to the same block are transmitted over the channel spaced by $I-1$ packets.

As an example, in Fig. 7.7, the LL packet residual error rate is reported for the 80-th and the 90-th percentile of the CCH users as a function of the interleaving buffer size. The graph has been obtained considering a first set of DCH users ($N_{DCH} = 200$) that are on the move as explained above, while CCH ($N_{CCH} = 1000$) users are static[13]. It is worth noting that, if the interleaving buffer is large enough, the FEC can completely avoid losses in 80% ($B = 10$ Kbyte) and 90% ($B \approx 30$ Kbyte) of the cases.

7.5.2 Throughput and Delay Performance

To obtain the results presented in this Section, the 3G system simulator has been configured according to what explained in Section 7.2. By its execution, a set of CCH channel traces have been obtained for every user in the system. Further, these traces have been used (off-line, i.e., in a subsequent simulation phase) as the input for a HARQ simulator, from which performance measures have been finally obtained (see Fig. 7.8). A common logical channel[14] bitrate of $B_r = 120$ Kbps will be considered in the results reported in the sequel.

As a first set of results, we focus on the channel efficiency (η). In Fig. 7.9, η is plotted considering $N_u = 20$ as a function of the number of redundancy PDUs which are inserted in every FEC block (H) and maintaining the ratio K/N at a constant value. The CCH users Doppler frequency considered to obtain this graph is $f_d = 40$ Hz (to reflect a moderate mobility scenario). A first set of curves (bold ones) are obtained considering $K/N = 0.9$, whereas the second set is achieved with $K/N = 0.8$. The simple ARQ case is reported for comparison. As an example, focusing on the case $H = 8$ and considering the case "HARQ $I = 2$", the usage of HARQ leads to the throughput gains reported in Table 7.1.

The motivation behind the choice of the case "HARQ $I = 2$, $H = 8$" is illustrated in Figs. 7.10 and 7.11, where the mean LL packet delivery delay and its complementary cumulative statistics (ccdf) are reported, respectively. From these figures it is clear that, in addition to the good achievable throughput performance, the selected case is also a good compromise for the delay, which is still of the same order of magnitude of the one introduced by Selective Repeat ARQ. Note that as I increases, the delivery delay

[12] Matrix indexes are expressed in units of PDUs.

[13] A Doppler frequency of $f_d = 2$ Hz has been considered in such a case. Due to the low f_d value, long LL bursts are experienced by CCH users.

[14] The useful link layer bitrate, i.e., not inclusive of physical channel coding, rate matching and MAC overheads.

SCHEMES →	**ARQ**	**HARQ K/N=0.8**	**HARQ K/N=0.9**
Throughput	0.667	0.69	0.73
Throughput Gain (ARQ Vs HARQ) [Kbps]	0	2.8 Kbps	7.6Kbps

Table 7.1: HARQ throughput gains considering $I = 2$ and $H = 8$ and $f_d = 40$ Hz.

also increases since, due to the interleaving process, a longer time has to be waited for to send new FEC blocks. This long delay also impacts the outstanding retransmissions since the packets contained in the matrix are always sent in a row.

In Figs. 7.12 and 7.13, η is plotted for $N_u = 50$ and $N_u = 100$, respectively. The others system parameters are maintained unchanged. The results, in these cases, are presented in the following Table 7.2. The advantage of HARQ solutions becomes clear as N_u increases.

SCHEMES →	**ARQ**	**HARQ (K/N=0.8)**	**HARQ (K/N=0.9)**
$N_u = 50 \downarrow$			
Throughput	0.5	0.6	0.64
Throughput Gain (ARQ Vs HARQ) [Kbps]	0	12 Kbps	16.8 Kbps
$N_u = 100 \downarrow$			
Throughput	0.374	0.5	0.54
Throughput Gain (ARQ Vs HARQ) [Kbps]	0	15.2 Kbps	19.92 Kbps

Table 7.2: HARQ throughput gains considering $I = 2$ and $H = 8$ and $f_d = 40$ Hz.

In the sequel we look at the play–out buffer requirements when a video streaming flow is being transmitted over the CCH channel. The trade-off between buffer requirements and channel efficiency is reported in Fig. 7.14. What is referred here as *maximum buffer size* (B_{size}) is the buffer dimension which is needed to avoid that buffer starvation occurs at the application play–out buffer (Fig. 7.15). To track the play–out buffer occupancy during the simulation, we consider a variable input flow λ that corresponds to the link layer outgoing flow[15], whereas we account for a constant output flow μ which is set at the value $\mu = B_r \times \eta$.[16] As can be clearly seen in Fig. 7.14, by tuning the coding redundancy H, it is possible to trade channel efficiency for buffer requirements (B_{size}). The curves in Fig. 7.14 have a first part where the slope $\mathcal{S} = \mathrm{d}B_{size}/\mathrm{d}\eta$ is still limited, and a second part, in which $\mathcal{S}$ suddenly increases. Obviously, this second part should be avoided, since only marginal throughput advantages can be achieved at the expense of a substantial increase of B_{size}. Moreover, for illustration purpose, two points have been marked in this figure ($N = 40, H = 8$ and $N = 40, H = 4$) to highlight how the same choice for the FEC block parameters translates in terms of B_{size} and η for different N_u values.

In order to gain some insights in the low mobility scenario ($f_d = 2$ Hz), in Fig. 7.16 we plot the

[15]Considering *in-order* delivery of link layer packets.

[16]The evaluation of the required buffer size B_{size} is therefore executed off-line, after the evaluation of η.

FEC error correction probability by varying H, f_d and I. The FEC error correction probability is defined here as the probability that the pro–actively added redundancy PDUs can successfully correct the errors occurring during the first transmission of a FEC block. In such a case we do not need further packets to be retransmitted. From the figure it is clear that the added redundancy is effective and that its convenience is higher over heavily correlated channels ($f_d = 2$ Hz). Over such channels, also the interleaving depth has a positive impact. The throughput/buffer relationships, in such a case, are very similar to what reported in Fig.7.14.

In the following, we give some insights on the impact of the system load, i.e., on the effect of an increased number of unicast users (background traffic) on the channel efficiency of the proposed HARQ algorithms. For this purpose, let us consider a number of CCH users in the system equal to $N_{CCH} = 300$, i.e., we add 100 DCH users, characterized by the same mobility model and system parameters presented above. In this case, the system interference is increased and, as a consequence, the packet error probability (PEP) experienced by the multicast users is increased as well. The increased PEP is then reflected on the channel efficiency which is considerably reduced. As an example, in the case where $N_u = 50$, $K = 32$, $H = 8$ and $I = 2$, η approaches 0.32, i.e., it has almost halved with respect to the previous case ($N_{CCH} = 200$). The reason behind this is that a small portion of the multicast users, in such a case are characterized by a very high error probability. These users experience a very unfavorable channel and frequently ask for retransmissions. As a consequence, they are heavily impacting the forward channel throughput, since in the proposed schemes every retransmission request is satisfied. In some cases this behavior should be avoided. Consider for example the transmission of a video streaming flow. In that case some residual errors can be tolerated, but the flow has to be transmitted in a timely manner and respecting the constraints imposed by the users play–out buffers (to avoid the buffer starvation phenomenon). In such a scenario, it is unacceptable to have such a small portion of the users congesting the forward channel transmission, since they will waste the system resources leading to a poor video quality for the remaining users in the system. Instead, it would be better to keep the transmitted flow to a reasonable throughput value, by accepting some degree of degradation especially at the users experiencing a bad channel state. In such a situation the best policy is no more to retransmit the lost packets for every user, instead, it would be better to control the number of retransmissions to maintain the forward channel throughput to an acceptable value (i.e., to avoid, for example the buffer starvation phenomenon) by limiting, for example, the number of retransmissions for the bad channel users.

As a first solution to this problem, one could use the simple but effective strategy presented in the following. A limit on the number of satisfied retransmission requests is imposed. In particular, after the first reception of a FEC block, every user communicates its reception status to the base station, by sending a NACK message when the FEC block is undecodable. At the sender side, the incoming NACKs are collected and the retransmission process for the corresponding block is initiated only if the number of retransmission requests is large enough, i.e., if the number of received NACKs for that block exceeds a given threshold τ (that can be expressed as a percentage of the number of multicast users in the cell (N_u)). However, even if this mechanism is very simple, it presents the following major drawbacks:

- It does not limit the number of NACKs flowing on the backward channel. In this case, in fact, the

sender still needs to acquire the NACK messages from every multicast user in the cell in order to evaluate the percentage (τ) of requests. For this reason, even if the algorithm is able to improve the forward channel efficiency, by possibly denying the retransmission process for a block, the NACKs are still generated and sent over the uplink channels by every user unable to decode that block. As a consequence, error prone channels will lead to a highly loaded uplink channel which in turn causes non trivial problems such as collisions and the need for uplink management procedures.

- The retransmission process can be denied for those users with good channel condition. Such users rarely ask for retransmissions, but their requests could be dropped if τ is large enough. In these cases, the system is unfair, since a *good user*, which is rarely asking for the channel resource is penalized in the same manner as the *bad users* that are continuously triggering new retransmissions.

In order to solve the drawbacks discussed above, we present here a second solution that is based on a probabilistic approach rather than on the selection of the hard threshold τ. The task of the base station controller is to keep the forward channel efficiency to a reasonable level η^*. In order to maintain such level, some retransmission requests have to be denied, as above. Let us define an overall retransmission acceptance probability P as the probability that the retransmission process is initiated for a given FEC block. This probability comes from the superposition of the acceptance probabilities at every user. Let us better explain this point. The generic user i fails to decode a FEC block with probability p_{ei} and subsequently decides whether or not a retransmission for that block should be initiated with probability p_{ri}. Then, user i sends a NACK for the generic FEC block with probability $p_i = p_{ei} \times p_{ri}$. In some sense, we are adding a probabilistic filter after the decoding process in order to decide whether the retransmission for an undecodable block has to be requested. It is important to observe that the mechanism is operating directly at the user terminal, by denying (in a probabilistic manner) the transmission of NACK messages. This is very useful in order to obtain a simple and distributed algorithm and to limit the number of NACKs flowing over the backward channel. At the sender side, the retransmission for a block is initiated if at least one NACK is received for that block, i.e., using the standard algorithm presented in the previous sections. The modifications are performed at the receiver side only adding the probabilistic decision in the NACKs sending process. Hence, the overall retransmission probability can be obtained as:

$$P = 1 - \prod_{i=1}^{N_u} (1 - p_i) \tag{7.6}$$

where p_i is the block retransmission request probability for user i. Let us pose the following constraint: $p_i = p^* \; \forall i \in \{1, 2, \ldots, N_u\}$, which means that every user has the same probability to send a NACK over the backward channel, i.e., the same resource request capability is assigned to every user. Thanks to this assumption p_{ri} can be found as:

$$p_{ri} = \min\left(1, \frac{p^*}{p_{ei}}\right) \tag{7.7}$$

where:

$$p^* = 1 - (1 - P)^{1/N_u} \tag{7.8}$$

To run the algorithm the sending base station only needs to communicate P and N_u to every multicast user in the cell. The users will then independently estimate p_{ei} and obtain p_{ri} according to Eq. (7.7). The

target channel efficiency η^* can then be selected by appropriately setting the P parameters. The algorithm has the following advantages:

- It can be simply implemented at every multicast user (in a distributed manner).
- Only two global parameters (P and N_u) are needed, at every user, to run the algorithm. These parameters can be disseminated through an initial setup phase and subsequently updated by the base station controller according to various needs such as a change in N_u and/or in η^*.
- Since NACKs are inhibited directly at their generation point, i.e., at the users' terminals, the mechanism can highly reduce the backward channel utilization.

In the following graphs, we report some performance metrics to prove the effectiveness of these new solutions. In Fig. 7.17, we report the channel efficiency η as a function of the global parameter P considering $N_u = 100$ and by varying f_d. In Fig. 7.18, instead, we plot the relationship between p_{ei} and p_{ri} by varying P. Clearly, p_{ri} is inversely proportional to the block error probability p_{ei}. The algorithm is introducing a soft-degradation of the users' performance, by limiting the retransmissions for those users in bad channel states and, at the same time, avoiding the resource wastage deriving from the fulfillment of their continuous retransmission requests.

In Fig. 7.19, we report the cumulative distribution concerning the number of users with residual packet error probability less than or equal to the value reported in the abscissa comparing the two methods (fixed threshold and probabilistic) discussed above and considering the same target efficiency $\eta^* = 0.6$ for both schemes. It is clear how the probabilistic approach can achieve better performance, by resulting in a lower error probability for those users with a medium/low channel error rate. The performance regarding the users experiencing bad channel conditions is unchanged. For the same throughput value, the probabilistic approach allows a fair redistribution of the retransmissions resources among users. The algorithm is referred to as fair since every user has the same retransmission request capability. Obviously, it can be seen as unfair since the users in a bad channel condition are disadvantaged. However, this is necessary in order to limit the system resource wastage.

Finally, in Fig. 7.20, we report the percentage of users with residual error probability lower than ε as a function of the useful downlink bandwidth (reported in abscissa). The useful downlink bandwidth is defined here as the percentage of B_r that is used, on average, to carry new data packets, without accounting for retransmissions and coding overhead. The curves in the graph are obtained by varying the aggregate probability P. Obviously, there is a trade–off of the P parameter: at low P the retransmission probability is negligible and the forward channel throughput is high, but at the expense of a larger number of unsatisfied users. On the other hand, as P increases the retransmission probability increases and more users are able to correct their data. The P parameter can be chosen to cut the right trade–off between channel efficiency and users' satisfaction. In the graph, the points labelled as 'Phy FEC only' are used to mark the case where only physical layer processing (FEC and interleaving) is considered. As can be seen comparing these points with the ones achieved using HARQ, the advantage of the retransmission algorithm is considerable, even at small P values (i.e., at large useful bandwidth values).

7.6 Video Streaming Results

In this Section, we report some results on the effectiveness of the proposed algorithms when a H.263 video stream is transmitted over the forward channel. The results will be expressed here using the well-known video PSNR metric [43] as the performance indicator for the received video quality. A PSNR of 100 dB is used to represent the perfect quality case, i.e., when the original and the corrupted (transmitted) streams do not differ. The focus will be mainly on the advantages of the proposed HARQ solutions with respect to classical ARQ algorithms. The following system parameters will be considered in our analysis: number of DCH users in the system $N_{DCH} = 200$, number of multicast (CCH) users per cell $N_u = 50$, High mobility scenario (multicast users Doppler frequency $f_d = 40$ Hz). The video sequences considered in this work to carry out the PSNR performance evaluation are the *Highway* and the *NEWS* QCIF sequences that can be found on–line (among other standard video traces) in [40]. The H.263 encoder and decoder used here have been derived from the well known H.263 Telenor encoder/decoder suite version 2.0, written in standard C language and publicly available [96].

As a first set of results, in Fig. 7.21 we report the PSNR and the useful bandwidth as a function of the maximum number of allowed retransmission rounds in the error control algorithm, whereas in Fig. 7.22 performance metrics are plotted for different K/N values. With the term useful bandwidth, we mean the fraction of the downlink channel that, on average, can be exploited for the transmission of new data. The useful bandwidth has been normalized here to the channel bandwidth B_r. Moreover, we plot the video PSNR which has been obtained by weighing the differences between the original transmitted frames and the received corrupted frames [42]. In Fig. 7.23, we report the methodology which has been used to obtain the PSNR performance [42, 43]. First of all, the original YUV video sequence has been encoded in the H.263 format. Then the residual bit error traces, obtained at the output of the HARQ simulator, have been used to corrupt the H.263 video sequence. Finally, the corrupted video sequence has been decoded again into the YUV format and compared with the original one. A video-meter tool [42] has been used to compute the differences between the two videos and to translate them into PSNR values. This procedure has been repeated for every multicast user in the system. From Figs. 7.21 and 7.22, the following observation can be made:

- The PSNR performance quickly saturates, i.e., two retransmission rounds seem to be enough to achieve satisfactory PSNR values (≥ 60 dB).
- The inter-block interleaving is also effective, especially after the second retransmission round. Even if not reported here, also the interleaving has a negative effect on the delay performance (and play–out buffer dimensioning).
- In accordance with the PSNR saturation, also the useful bandwidth becomes flat after the third retransmission round. This indicates that further retransmissions are rarely required and that, for this reason, are leading to a small performance increase.
- An increase of the K/N ratio is beneficial for the useful bandwidth (a gain of almost 10 Kbps is observed), but is highly impacts the delay performance (and the play–out buffer occupancy, not

reported here).

Further, in Figs. 7.24, 7.25, 7.26 and 7.27, we report the mean, the standard deviation and the maximum PSNR for the 20-th, 30-th, 50-th and the 70-th percentile of the multicast users in the cell, respectively. In these graphs, histograms have been used to report the mean PSNR value on top of every histogram bar, vertical lines have been used to report the standard deviation. In addition, horizontal lines are also drawn to indicate the maximum PSNR value measured for every percentile. From these graphs, it is evident how the mean PSNR in the HARQ case is consistently higher than the one in the ARQ. Often (observe Figs. 7.24, 7.25 and 7.26), the mean PSNR value for HARQ dominates the maximum PSNR values in the ARQ case. In general, with HARQ solutions two retransmissions are enough to obtain an acceptable video quality $PSNR \geq 60$ dB even for low percentiles (i.e., 20-th percentile), whereas such performance is possible only after three or four retransmission with ARQ. This implies that the number of retransmission rounds can be decreased when HARQ is being used, by conserving a satisfying performance level and, at the same time, by decreasing the link layer delays (i.e., decreasing the requirements for the receiver play–out buffer). As an example, observe that one retransmission is enough, in the HARQ case, to achieve a perfect quality (PSNR=100 dB) for 50 % of the multicast users in the system, but four retransmissions are needed by the ARQ scheme. One retransmission is not enough to achieve a perfect quality even for 70 % of the users (Fig. 7.27) for ARQ.

Similar results have been obtained for different video traces, as an example, in Fig. 7.28, we report the PSNR cumulative distribution for the *NEWS* standard video sequence [40] considering HARQ and ARQ. HARQ performance is plotted using bold lines and a PSNR of zero dB means that the H.263 decoder has failed the decoding process due to errors in H.263 headers. An anomalous deadlock event (with subsequent termination of the decoding process) can occur when the residual errors affecting the H.263 video sequence are hitting important parts of the H.263 headers. In such a cases, the entire video reproduction is compromised with an anomalous termination of the play–out process. These cases do not depend on our evaluation approach, but are rather strictly related to the decoding process that has therefore to be improved, or at least additional mechanisms should be implemented to provide extra-protection for H.263 headers. Fig. 7.28 has been obtained after a long measurement process where the video sequence has been run for every user for more than 400 times, by randomly changing the offset used to read the channel traces. Also from this figure, the superiority of HARQ solutions is clear. Moreover, it is worth noticing that, even using HARQ, and even after four retransmission rounds the deadlock probability is not zero. As an example, considering two retransmission rounds and the HARQ algorithm, one user in 100 will experience, on average, a deadlock event. This definitely calls for some improvements to be added in the decoding phase in order to cope with corrupted H.263 headers.

7.7 Conclusions

In this Chapter the multicast delivery in 3G Cellular Networks has been investigated in detail. Some error control algorithms have been proposed to provide throughput efficient and delay limited solutions. Their performance has been investigated over an independent channel first. Later on, accurate W–CDMA

channel traces have been considered. Particular attention has been paid to the understanding of the impact of the system parameters (such as mobility and system load) on system requirements (such as play–out buffer size) and to the investigation of the effectiveness of the proposed schemes for the transmission of video streaming flows. The solutions proposed here are promising since they allow for a performance improvement over standard ARQ techniques over a wide set of channel error statistics and system parameters.

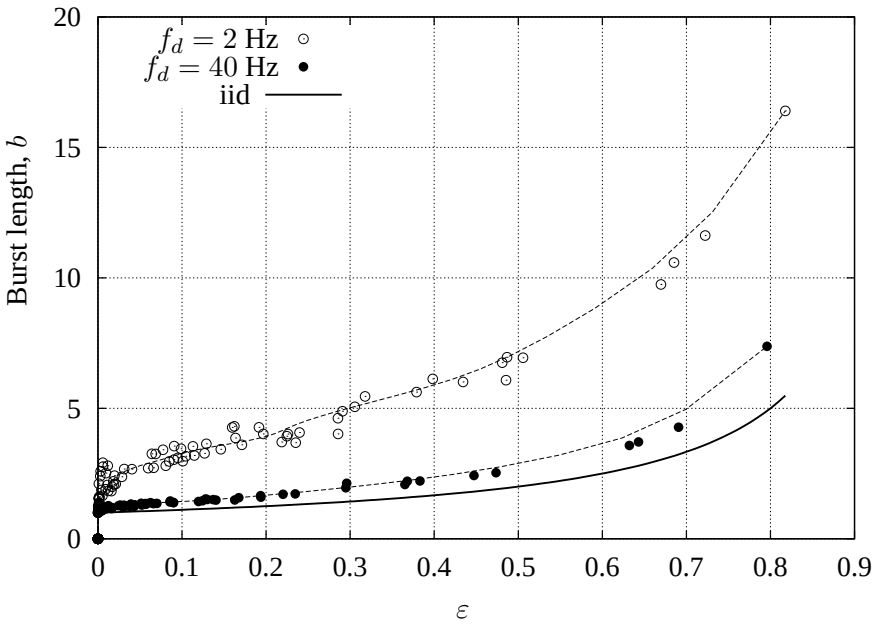


Figure 7.1: Average error burst length (b) *Vs* packet error probability ε.

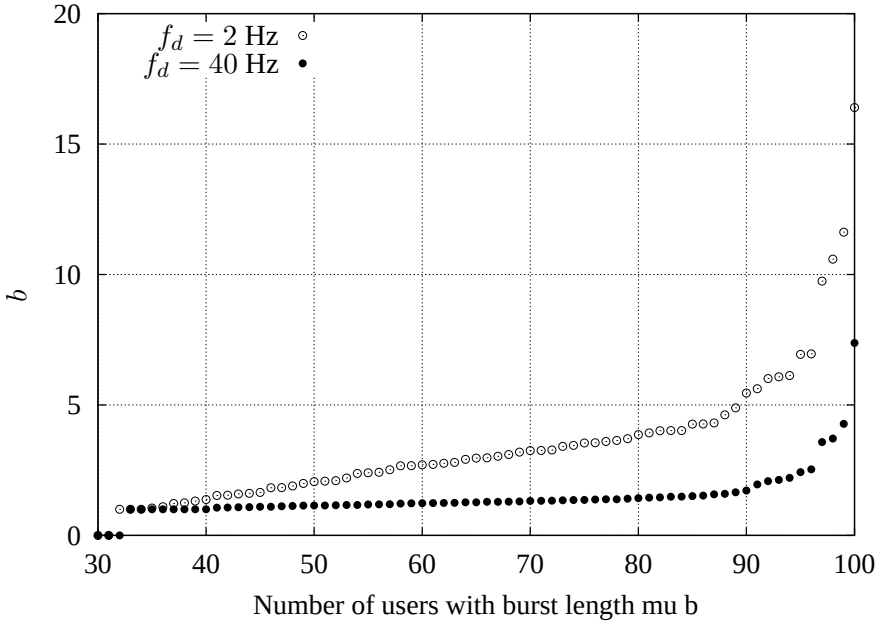


Figure 7.2: Cumulative distribution of the burst length.

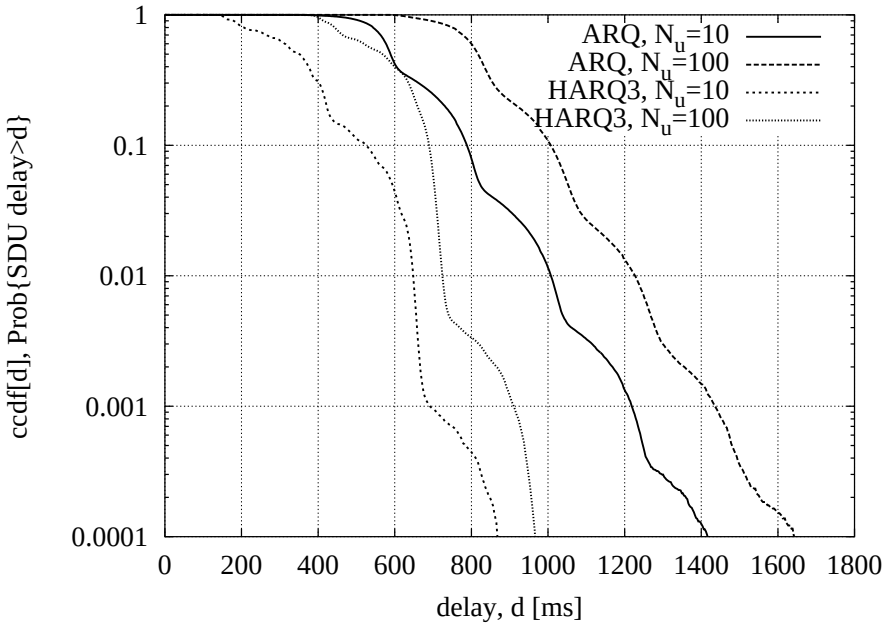


Figure 7.3: SDU complementary cumulative delivery delay statistics (ccdf) by varying N_u, $p = 0.1$, independent PDU error processes. Comparison between HARQ3 FEC(K=15, N=19) scheme and selective repeat ARQ.

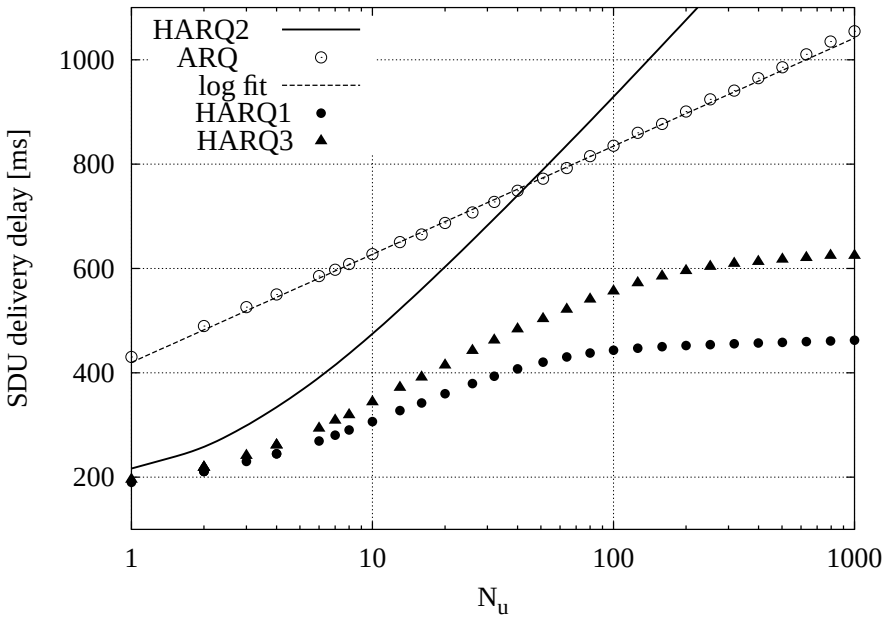


Figure 7.4: Mean SDU delivery delay as a function of N_u, $p = 0.1$, *iid* channel. Comparison between HARQ FEC(K=15, N=19) schemes and selective repeat ARQ.

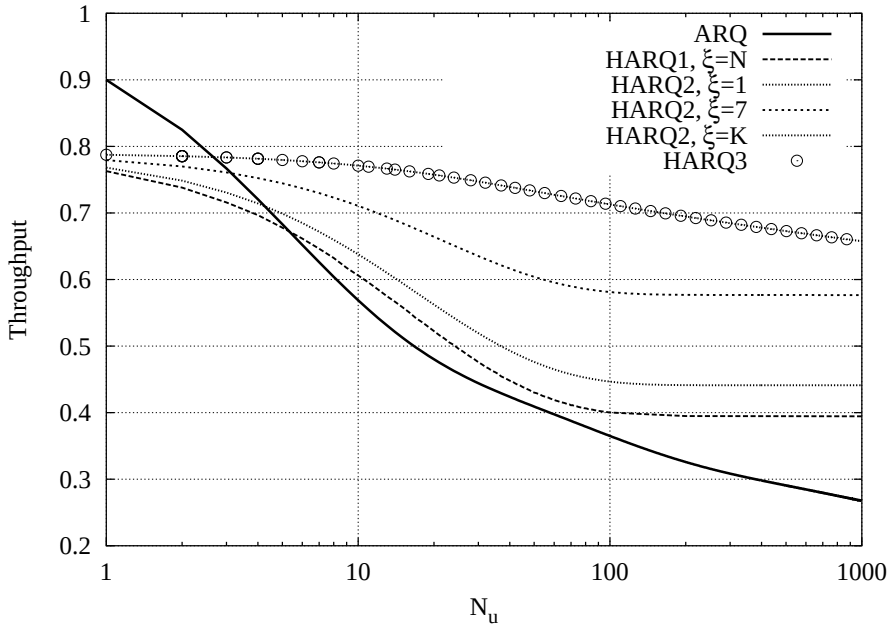


Figure 7.5: Throughput as a function of the number of multicast users in the cell N_u, independent channel, $p = 0.1$, FEC(K=15, N=19).

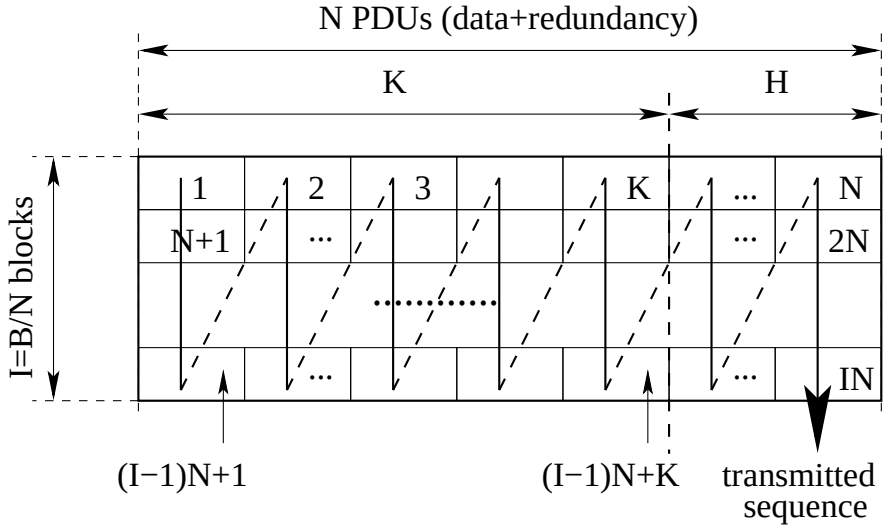


Figure 7.6: Link Layer packets interleaving.

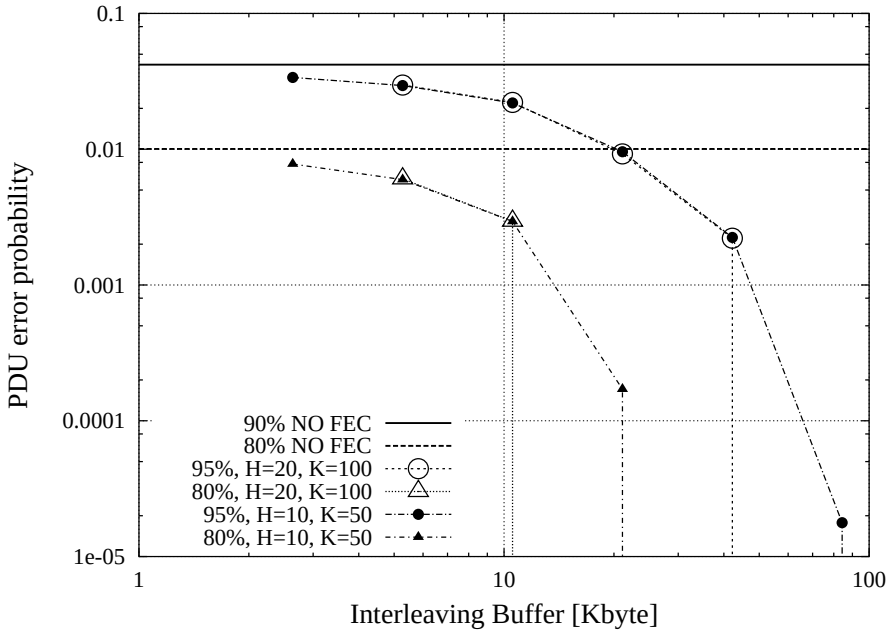


Figure 7.7: Effectiveness of packet-based FEC with interleaving. Bursty case ($f_d = 2$ Hz).

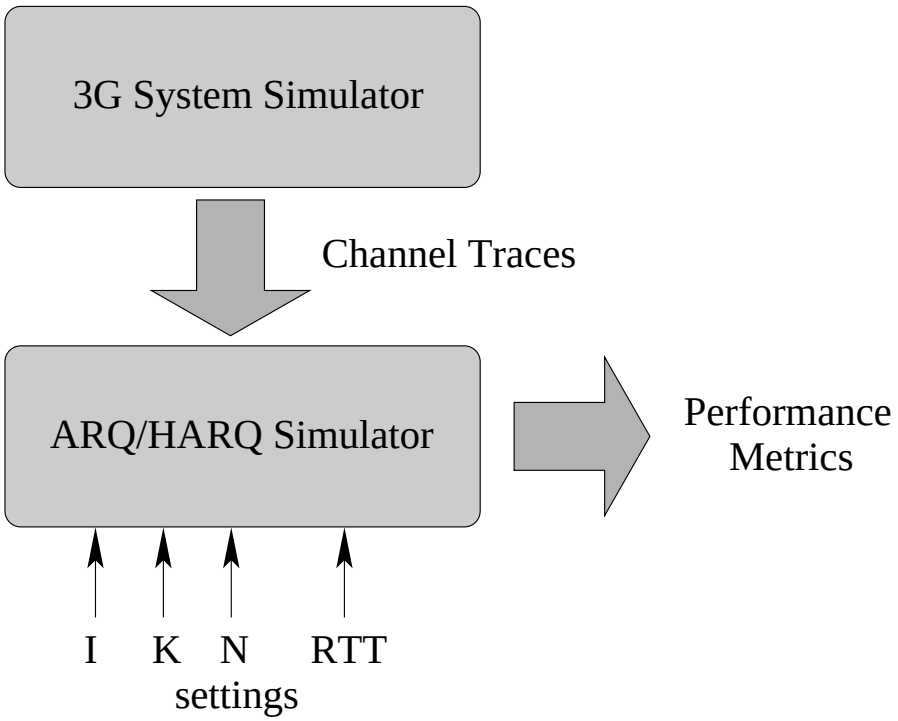


Figure 7.8: Apoproach followed to characterize ARQ and HARQ algorithms over 3G channel traces.

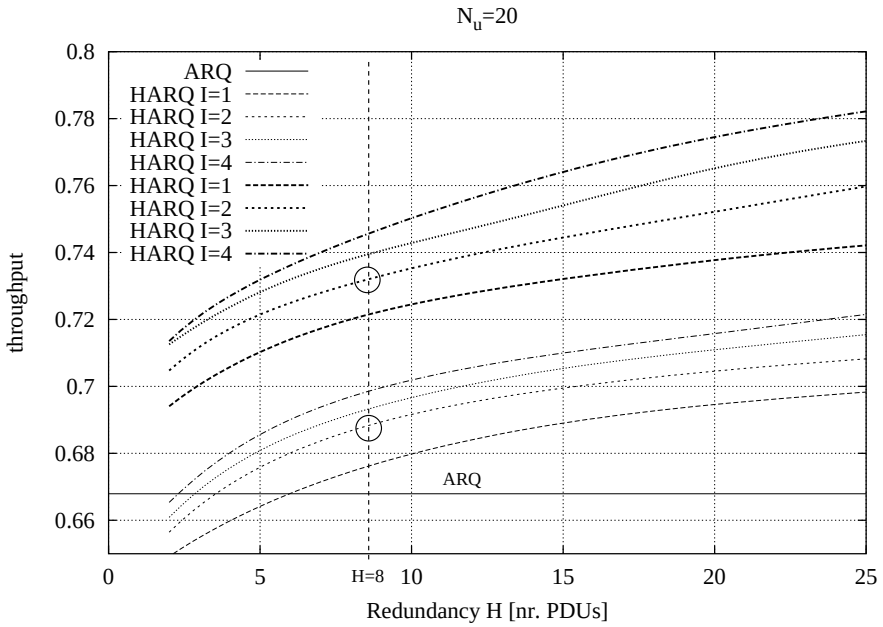


Figure 7.9: Channel Efficiency as a function of H for $N_u = 20$ and $f_d = 40$ Hz.

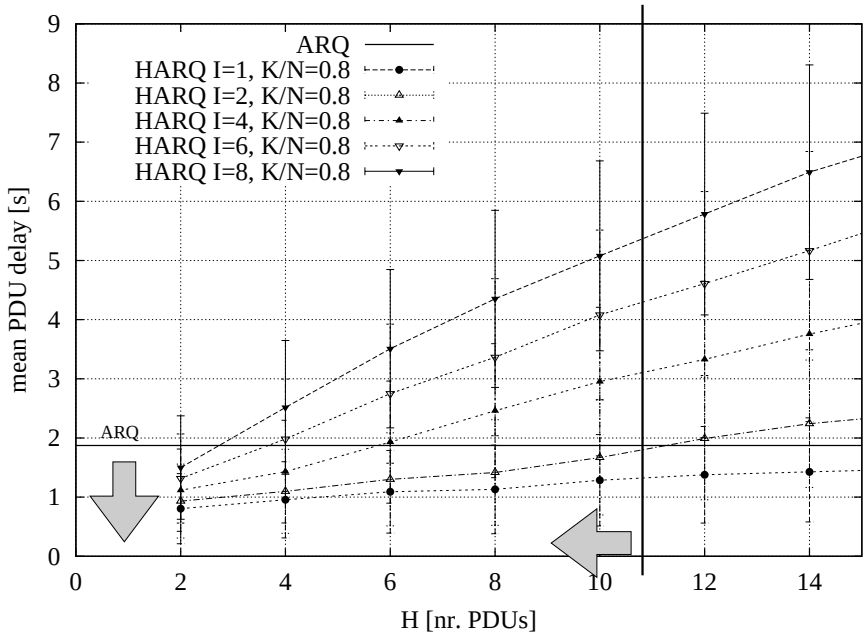


Figure 7.10: Mean delivery delay and its standard deviation. $N_u = 20$, $K/N = 0.8$ and $f_d = 40$ Hz.

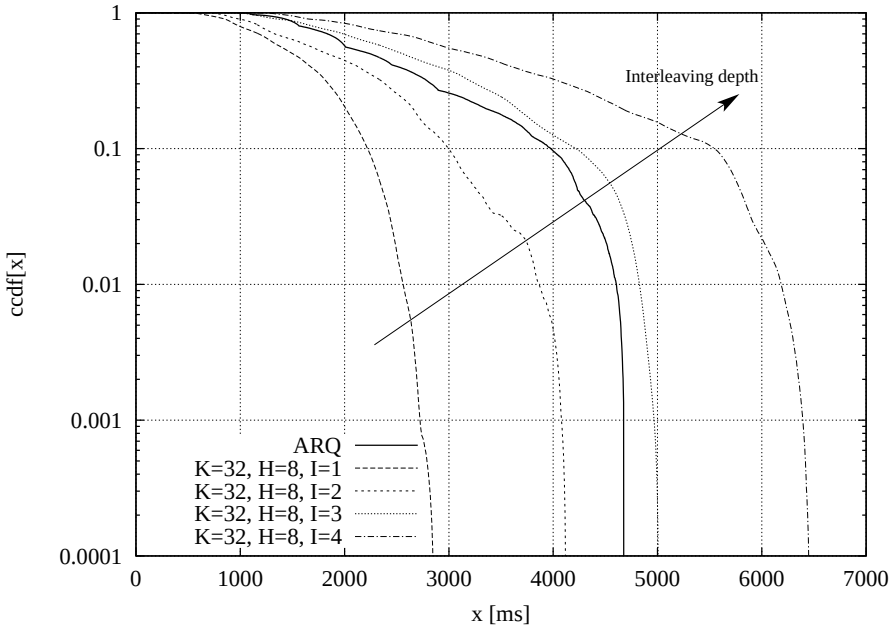


Figure 7.11: Delivery delay cumulative complementary distribution. $N_u = 20$, $K/N = 0.8$ and $f_d = 40$ Hz.

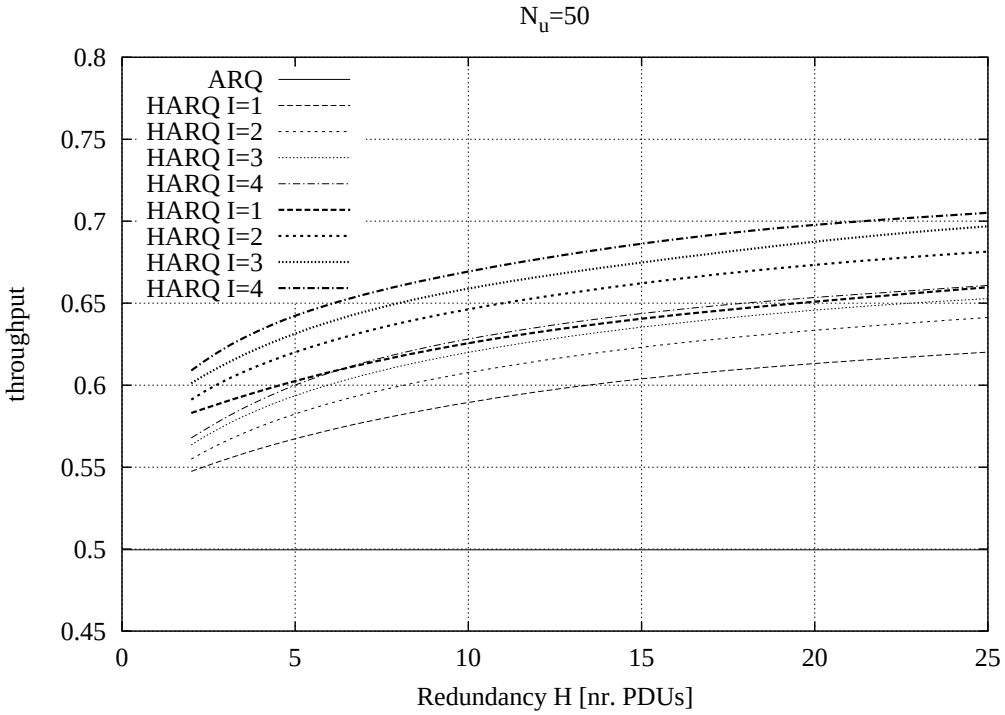


Figure 7.12: Channel Efficiency as a function of H for $N_u = 50$ and $f_d = 40$ Hz.

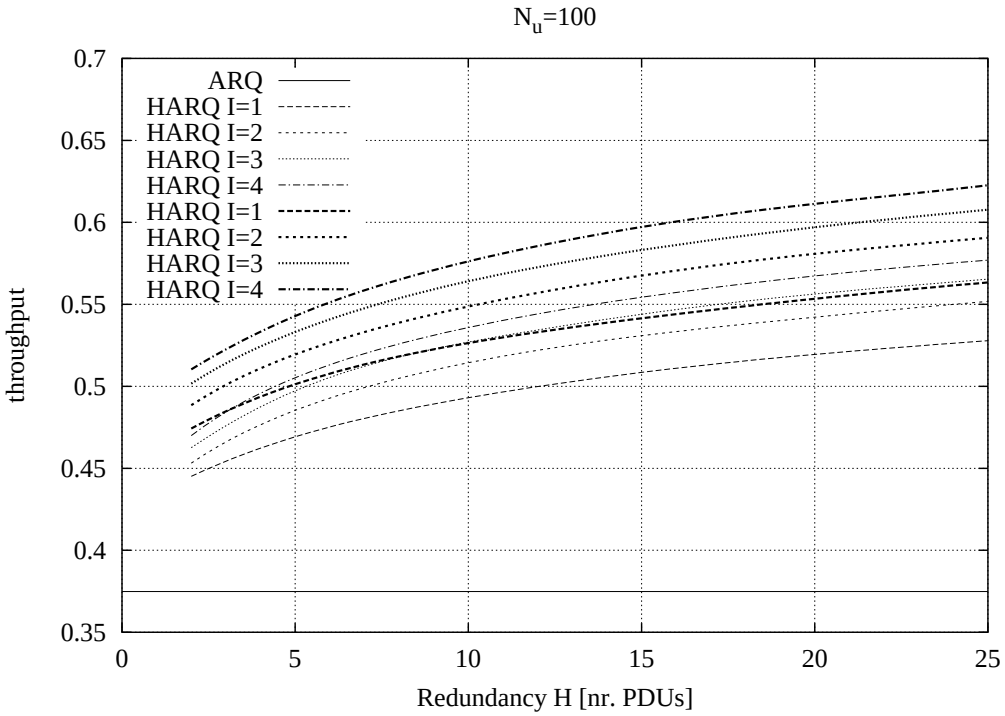


Figure 7.13: Channel Efficiency as a function of H for $N_u = 100$ and $f_d = 40$ Hz.

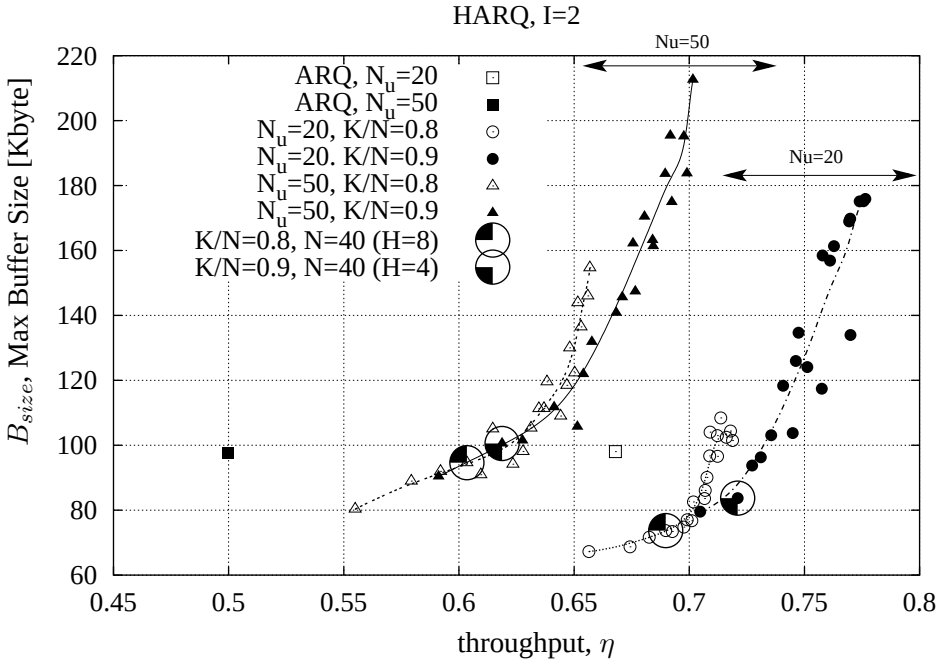


Figure 7.14: Trade-offs between buffer requirements and channel efficiency. $f_d = 40$ Hz.

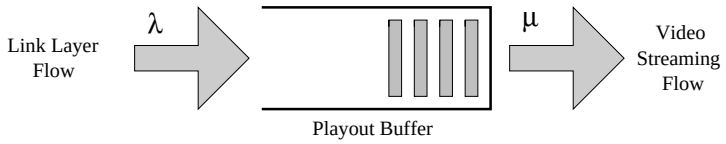


Figure 7.15: Application play-out buffer.

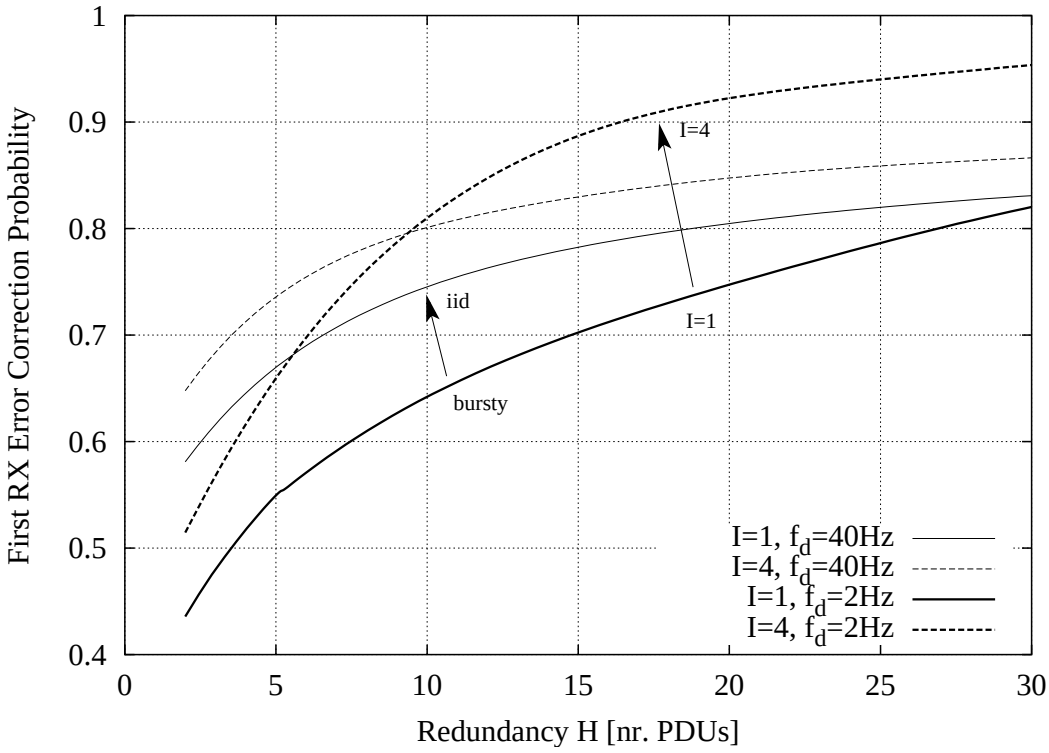


Figure 7.16: FEC correction probability.

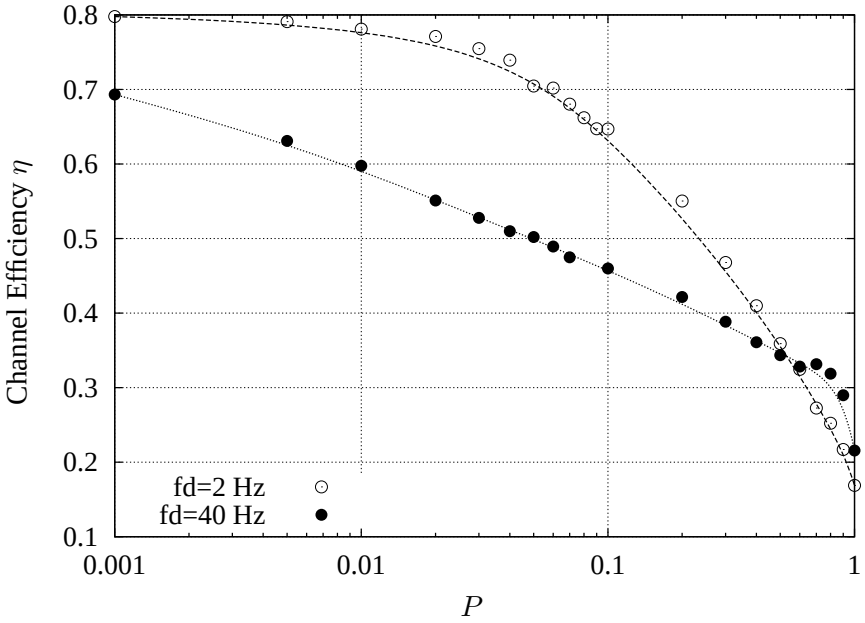


Figure 7.17: Channel efficiency control by means of the probabilistic NACK acceptance method.

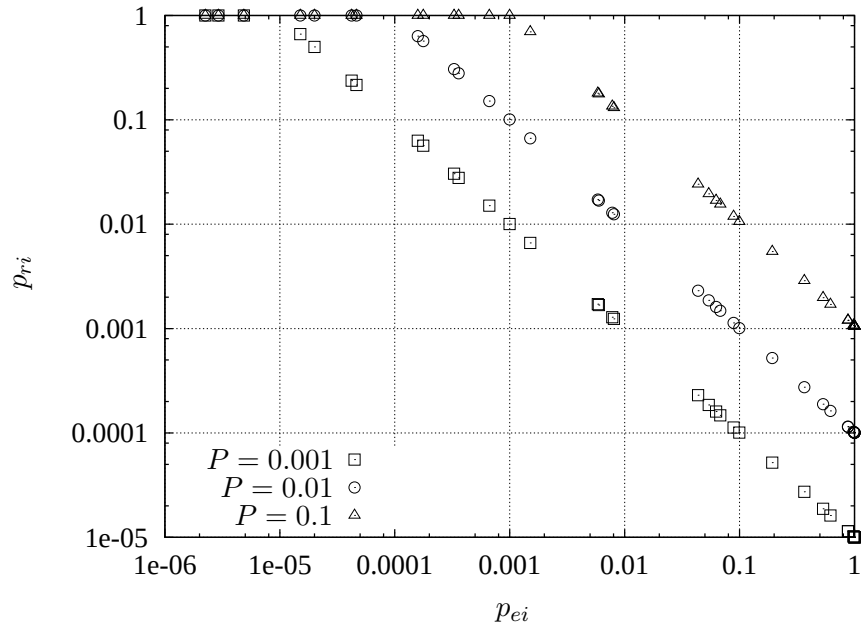


Figure 7.18: Correspondence between error (p_{ei}) and NACK acceptance (p_{ri}) probabilities by varying the global probability P.

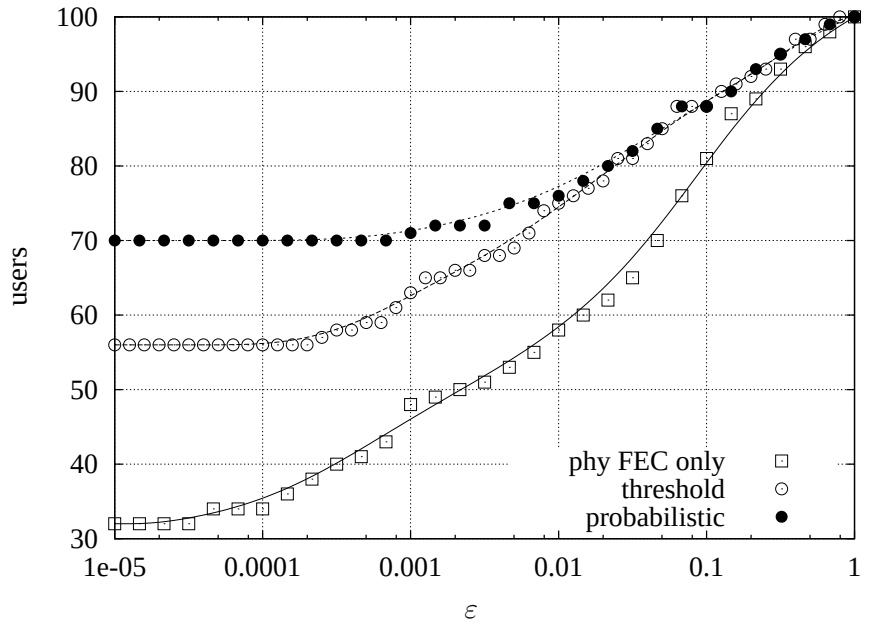


Figure 7.19: Cumulative distribution of the number of users with residual packet error probability $\leq \varepsilon$: comparison between probabilistic and fixed threshold method.

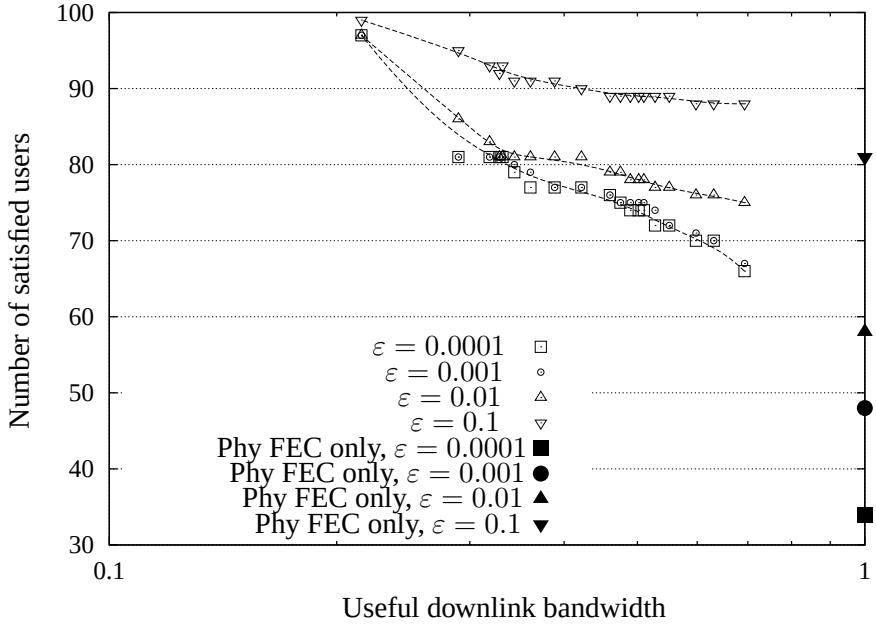


Figure 7.20: Percentage of satisfied users: percentage of users with residual packet error probability lower than ε as a function of η by varying P.

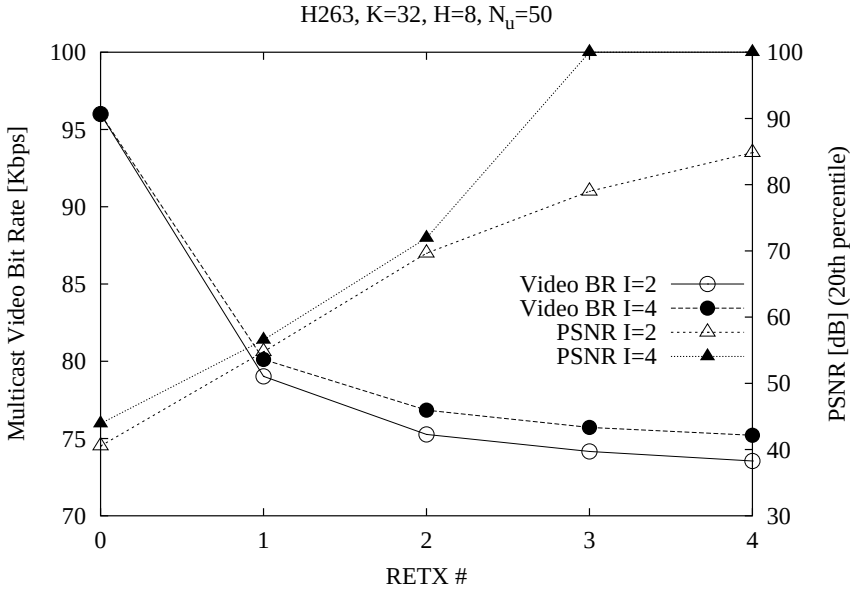


Figure 7.21: Video PSNR and useful bandwidth as a function of the maximum number of allowed retransmission rounds per FEC block.

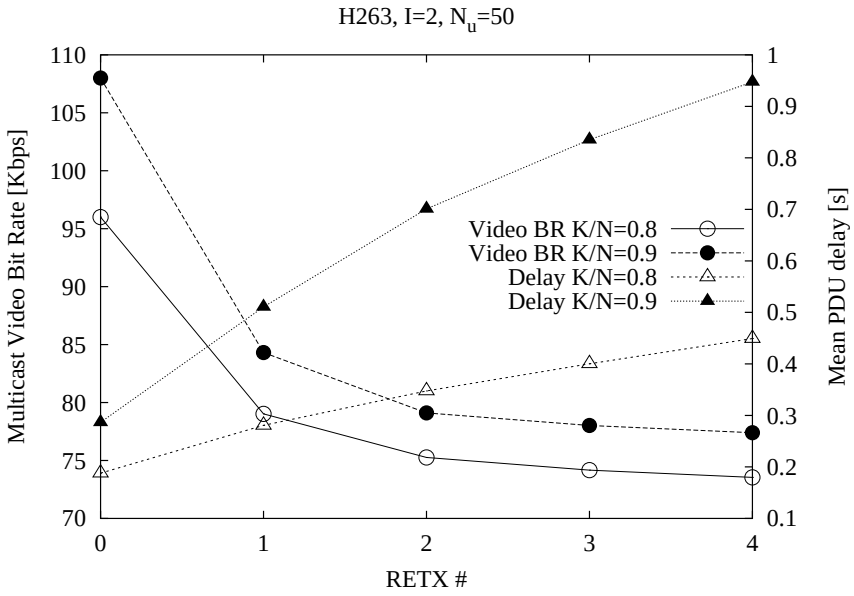


Figure 7.22: Delay performance and useful bandwidth as a function of the maximum number of allowed retransmission rounds.

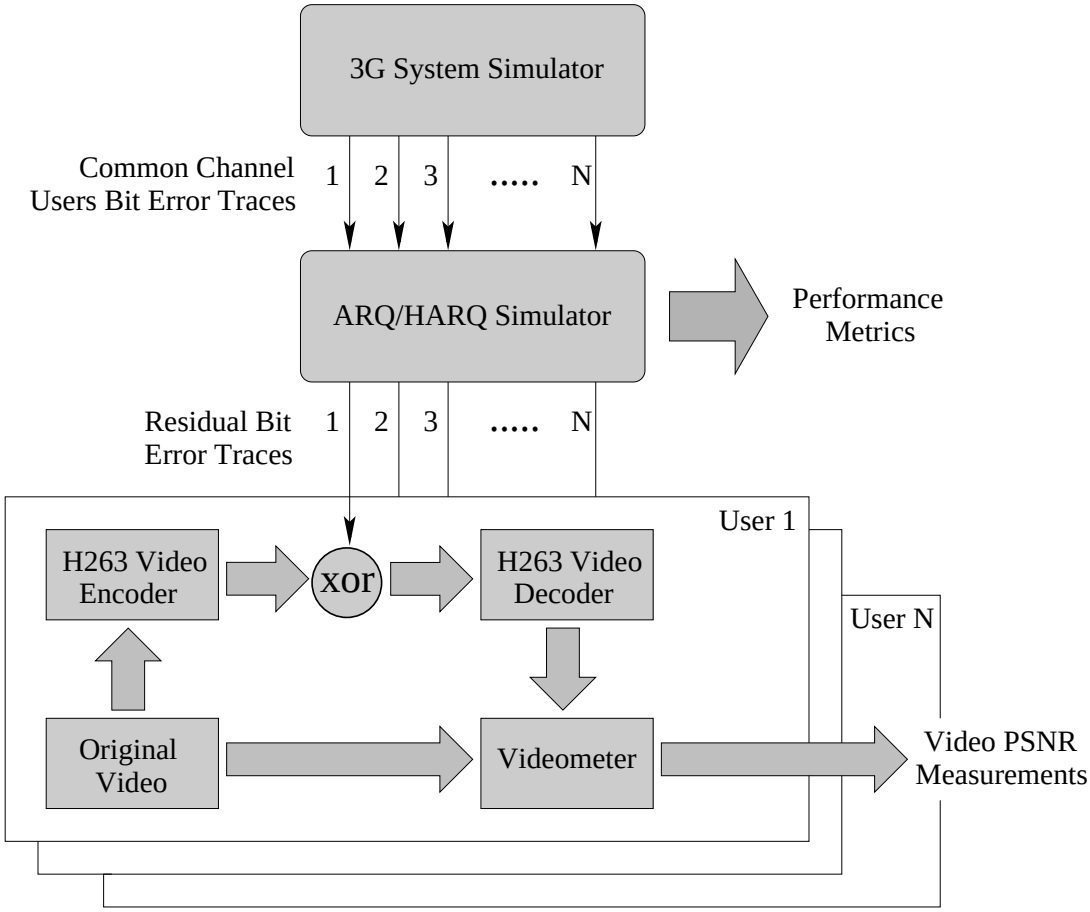


Figure 7.23: Approach followed to gather Video PSNR statistics.

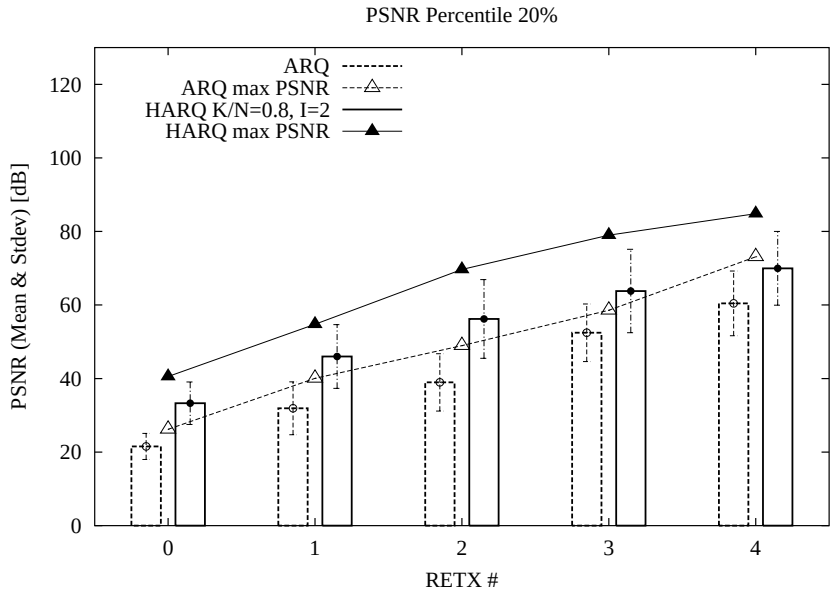


Figure 7.24: 20-th percentile of the PSNR as a function of the number of retransmission rounds.

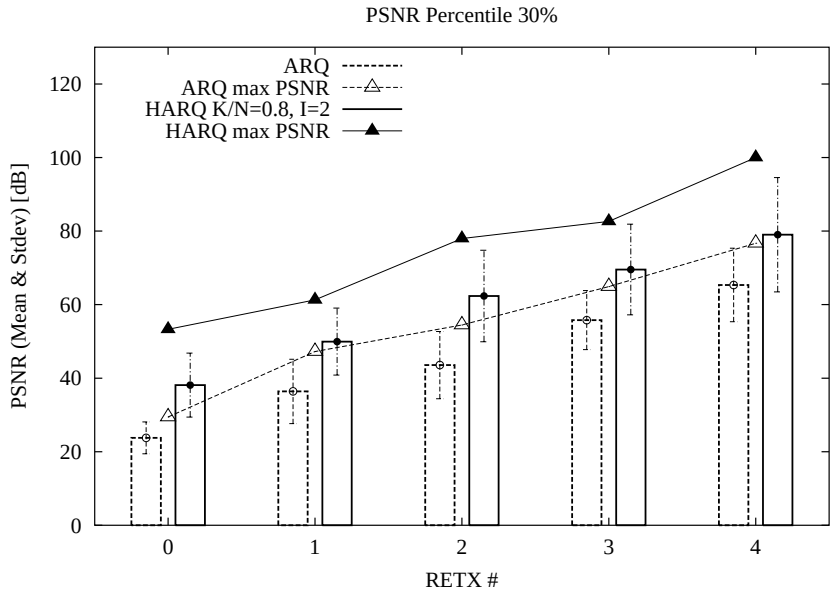


Figure 7.25: 30-th percentile of the PSNR as a function of the number of retransmission rounds.

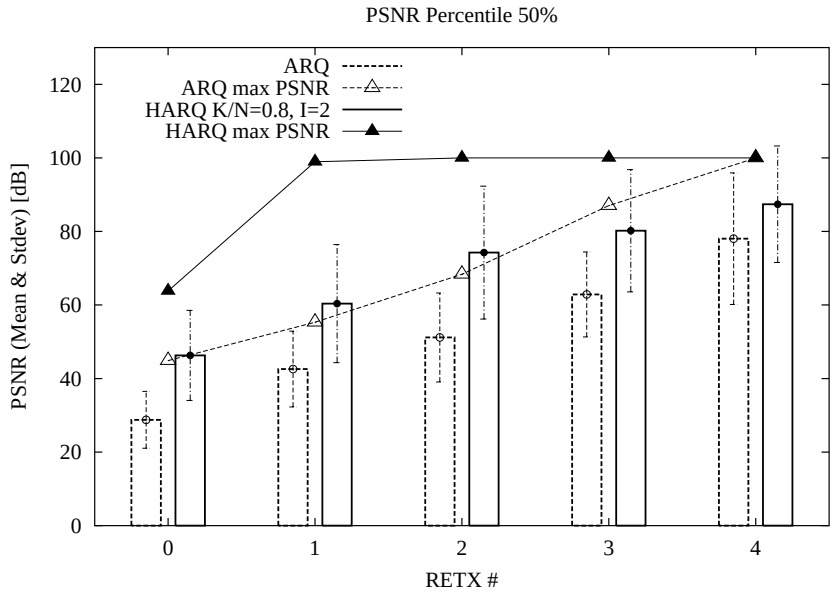


Figure 7.26: 50-th percentile of the PSNR as a function of the number of retransmission rounds.

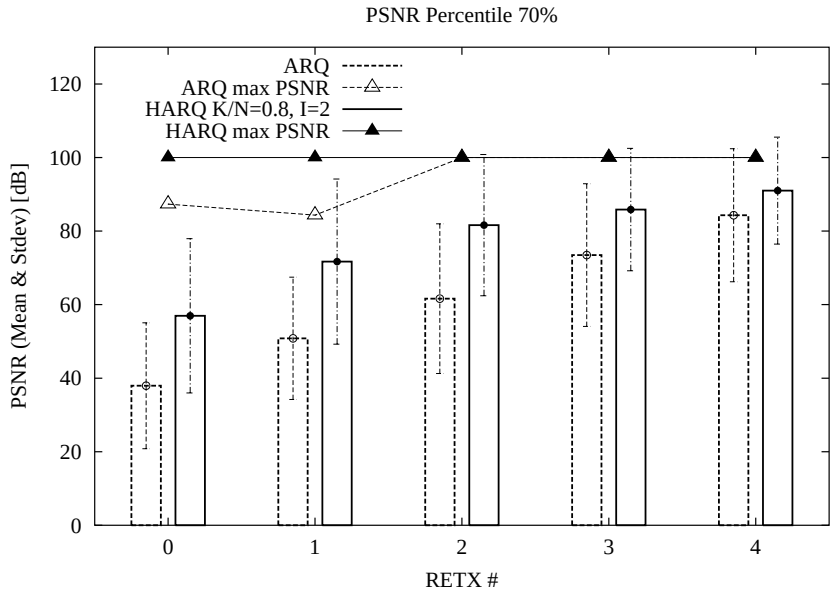


Figure 7.27: 70-th percentile of the PSNR as a function of the number of retransmission rounds.

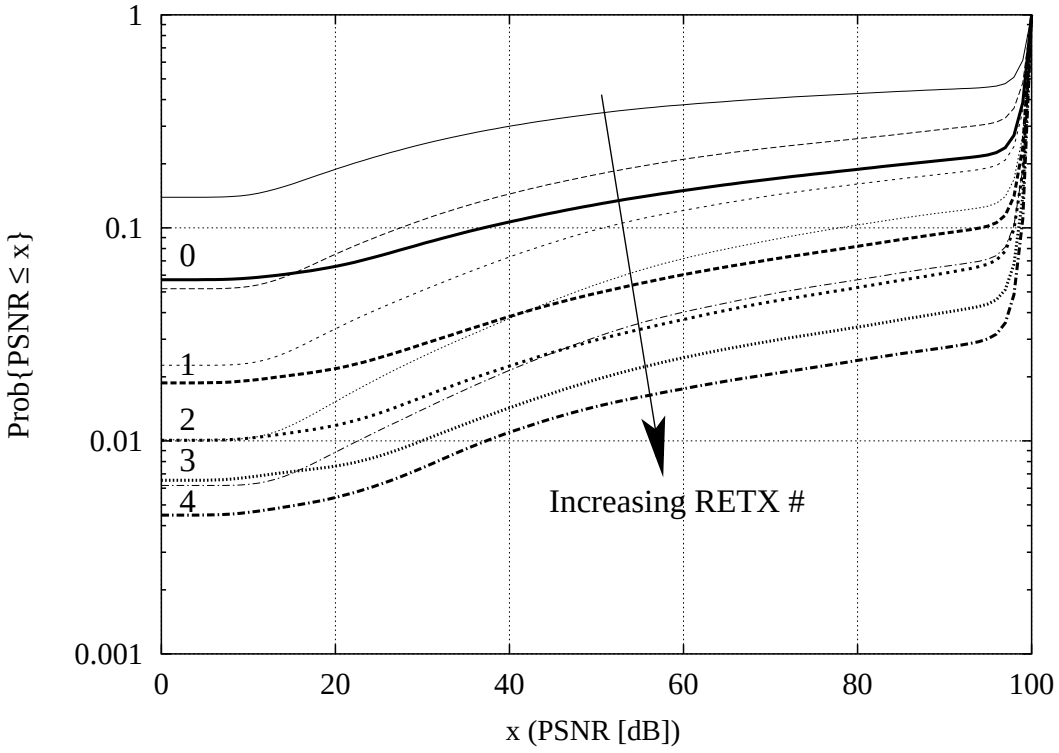


Figure 7.28: PSNR cumulative distribution function for the *NEWS* standard video sequence by varying the maximum number of allowed retransmission rounds ($0 \rightarrow 4$) for HARQ (bold lines) and ARQ.

Bibliography

[1] "3GPP Third Generation Partnership Project." [Available] `http://www.3gpp.org`.

[2] "Special issue on: Energy Management in Personal Communications and Mobile Computing," *IEEE Personal Commun. Mag.*, vol. 5, June 1998.

[3] "Consultative Committee for Space Data Systems (CCSDS), Space Communications Protocol Specification-Transport Protocol, CCSDS 714.0-B-1," May 1999. Blue Book.

[4] 3GPP TS 25.322, "Third Generation Partnership Project: Technical Specification Group Radio Access Network; Radio Link Control (RLC) Specification (Release 5)." [Available] `http://www.3gpp.org`.

[5] A. A. Abouzeid, S. Roy, and M. Azizoglu, "Stochastic modeling of TCP over lossy links," in *Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications, INFOCOM*, vol. 3, (Tel-Aviv, Israel), pp. 1724–1733, Mar. 2000.

[6] D. Adami, M. Marchese, and L. S. Ronga, "TCP/IP based Multimedia Applications and Services over Satellite Links: Experience of an ASI/CNIT Project," *IEEE Personal Commun. Mag.*, vol. 8, pp. 20–27, June 2001. Special Issue on Multimedia communications over satellites.

[7] M. Airy and J. M. Harris, "Analytical model for radio link protocol for IS–95 CDMA systems," in *Proceedings of IEEE VTC 2000-Spring*, vol. 3, (Tokyo, Japan), pp. 2434–2438, May 2000.

[8] O. Ait-Hellal and E. Altman, "Analysis of TCP vegas and TCP reno," in *Proceedings of the IEEE International Conference on Communications, ICC*, vol. 1, (Monréal, Québec, Canada), pp. 495–499, June 1997.

[9] I. Akyildiz, G. Morabito, and S. Palazzo, "Research Issues for Transport Protocols Satellite IP Networks," *IEEE Personal Commun. Mag.*, vol. 8, pp. 44–48, June 2001.

[10] I. Akyildiz, G. Morabito, and S. Palazzo, "TCP-Peach: A New Congestion Control Scheme for Satellite IP Networks," *IEEE/ACM Trans. Networking*, vol. 9, pp. 307–321, June 2001.

[11] I. Akyildiz, G. Morabito, and S. Palazzo, "TCP-Peach for Satellite Networks: Analytical Model and Performance Evaluation," *International Journal of Satellite Communications*, vol. 19, pp. 429–442, September/October 2001.

[12] M. Allman, S. Dawkinks, D. Glover, J. Griner, T. Henderson, J. Heidemann, S. Ostermann, K. Scott, J. Semke, J. Touch, and D. Tran, "IETF RFC2760: Ongoing TCP Research Related to Satellites," February 2000.

[13] M. Allman, D. Glover, and L. Sanchez, "IETF RFC2488: Enhancing TCP Over Satellite Channels using Standard Mechanisms," January 1999.

[14] M. Allman, V. Paxson, and W. Stevens, "IETF RFC 2581: TCP congestion control," April 1999.

[15] M. Anagnostou and E. Protonotarios, "Performance analysis of the Selective Repeat ARQ protocol," *IEEE Trans. Commun.*, vol. 34, pp. 127–135, Feb. 1986.

[16] F. Anjum and L. Tassiulas, "On the behavior of Different TCP Algorithms over a Wireless Channel with Correlated Packet Losses," in *Proceedings of ACM International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS*, (Atlanta, Georgia, USA), pp. 155–165, 1999.

[17] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Trans. Networking*, vol. 5, pp. 756–769, December 1997.

[18] C. Barakat, E. Altman, and W. Dabbous, "On TCP Performance in a Heterogeneous Network: A Survey," *IEEE Commun. Mag.*, vol. 38, pp. 40–46, Jan. 2000.

[19] P. Bergamo, D. Maniezzo, A. Giovanardi, G. Mazzini, and M. Zorzi, "An improved Markov Chain Description for Fading Processes," in *IEEE ICC 2002*, vol. 3, (New York), pp. 1347–1351, Apr. 2002.

[20] P. Bergamo, D. Maniezzo, A. Giovanardi, G. Mazzini, and M. Zorzi, "An improved Markov model for Rayleigh fading envelope," *IEE Electronics Letters*, vol. 38, pp. 477–478, May 2002.

[21] V. Bharadwaj, J. S. Baras, and N. P. Butts, "An Architecture for Internet Service via Broadband Satellite Networks," *International Journal of Satellite Communications*, vol. 19, pp. 29–50, January/February 2001. Special Issue on IP.

[22] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "IETF RFC3135: Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," June 2001.

[23] D. E. Brooks, C. Buffinton, D. R. Beering, A. W. William, D. Ivancic, M. Zernic, and D. J. Hoder, "ACTS 118x Final Report, High Speed TCP Interoperability Testing," July 1999. NASA/TM-1999-209272.

[24] J. W. Byers, M. Luby, and M. Mitzenmacher, "Accessing multiple mirror sites in parallel: Using Tornado Codes to speed up downloads," in *Proceedings of IEEE INFOCOM*, (New York, US), pp. 275–283, Mar. 1999.

[25] R. Caceres and L. Iftode, "Improving the performance of reliable transport protocols in mobile computing environments," *IEEE J. Select. Areas Commun.*, vol. 13, pp. 850–857, June 1995.

[26] A. Calveras and J. P. Aspas, "TCP/IP over wireless links: Performance Evaluation," in *IEEE VTC*, vol. 3, (Ottawa, Ontario, Canada), pp. 1755–1759, May 1998.

[27] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links," in *Proceedings of ACM Mobicom 2001*, (Rome, Italy), pp. 287–297, July 2001.

[28] N. Chaskar, T. Lakshman, and U. Madhow, "Tcp over wireless with link level error control: analysis and design methodology," *IEEE/ACM Trans. Networking*, vol. 7, pp. 605–615, October 1999.

[29] Y. J. Cho and C. K. Un, "Performance Analysis of ARQ Error Controls under Markovian Block Error Pattern," *IEEE Trans. Commun.*, vol. 42, feb/mar/apr 1994.

[30] A. Chockalingam and G. Bao, "Performance of TCP/RLP protocol stack on correlated fading DS–CDMA wireless links," *IEEE Trans. Veh. Technol.*, vol. 49, pp. 28–33, January 2000.

[31] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. New York: John Wiley & Sons, INC., 1991.

[32] S. Dawkins, G. Montenegro, M. Kojo, V. Magret, and N. Vaidya, "IETF RFC 3155: End-to-end Performance Implications of Links with Errors," August 2001.

[33] M. Degermark, "IETF RFC 2507: IP Header Compression," Feb. 1999.

[34] M. Degermark, M. Egan, B. Nordgen, and S. Pink, "Low Loss TCP/IP Header Compression for Wireless Networks," in *Proceedings of ACM Mobicom*, (New York, USA), Nov. 1996.

[35] A. Ephremides, "Guest Editor," *International Journal of Satellite Communications*, vol. 19, January/February 2001. Special Issue on IP.

[36] ETSI SMG2, "Evaluation report for ETSI UMTS Terrestrial Radio Access (UTRA) ITU-R RTT candidate," September 1998. [Available] `http://www.etsi.org`.

[37] European Space Agency, "ARTES 3, The ESA Multimedia Initiative." [Available] `http://www.estec.esa.nl/artes3`.

[38] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno and SACK TCP," Dec. 1995. [Available] `ftp.ee.lbl.gov/papers/sacks.ps.Z`.

[39] R. Fantacci, "Queueing analysis of the selective repeat automatic repeat request protocol for wireless packet networks," *IEEE Trans. Veh. Technol.*, vol. 45, pp. 258–264, May 1996.

[40] F. Fitzek and M. Reisslein, "MPEG-4 and H.263 Video Traces for Network Performance Evaluation," *IEEE Network*, vol. 15, pp. 40–54, Nov./Dec. 2001. [Available] `http://http://trace.eas.asu.edu`.

[41] F. Fitzek, M. Rossi, and M. Zorzi, "Error Control Techniques for Efficient Multicast Streaming in UMTS Networks," in *Proceedings of SCI 2003*, (Orlando, Florida, US), July 2003.

[42] F. Fitzek, P. Seeling, and M. Reisslein, "VideoMeter Tool for YUV bitstreams," *IEEE Network Mag., software tool for networking*, vol. 17, Jan. 2003. [Available] `http://www.eas.asu.edu/~mre/`.

[43] F. Fitzek, P. Seeling, M. Reisslein, M. Rossi, and M. Zorzi, "Investigation of the GoP Structure for H.26L Video Streams," Dec. 2002. Technical Report. [Available] `http://www.eas.asu.edu/~mre/GoP.pdf`.

[44] S. Floyd and T. Henderson, "IETF RFC 2582: The New Reno Modification to TCP's Fast Recovery Algorithm," April 1999.

[45] N. Ghani and S. Dixit, "TCP/IP Enhancement for Satellite Networks," *IEEE Commun. Mag.*, vol. 37, pp. 64–72, July 1999.

[46] A. Giovanardi, G. Mazzini, M. Rossi, and M. Zorzi, "Improved Header Compression for TCP/IP over Wireless Links," *IEE Electronic Letters*, vol. 36, pp. 1958–1960, Nov. 2000.

[47] A. Giovanardi, G. Mazzini, V. Tralli, and M. Zorzi, "Some results on power control in Wideband CDMA cellular networks," in *Proceedigs of WCNC*, (Chicago (IL)), September 2000.

[48] A. J. Goldsmith and P. P. Varaiya, "Capacity, mutual information, and coding for finite-state Markov channels," *IEEE Trans. Inform. Theory*, vol. 42, pp. 868–886, May 1996.

[49] R. Goyal, R. Jain, M. Goyal, S. Fahmi, B. Vandalore, S. Kota, N. Butts, and T. vonDeak, "Buffer Management and Rate Guarantees for TCP over satellite-ATM networks," *International Journal of Satellite Communications*, vol. 19, pp. 93–110, January/February 2001. Special Issue on IP.

[50] N. Guo and D. Morgera, "Frequency-Hopped ARQ for Wireless Network Data Services," *IEEE J. Select. Areas Commun.*, vol. 12, pp. 1324–1337, Oct. 1994.

[51] T. R. Henderson and R. H. Katz, "Transport Protocols for Internet-Compatible Satellite Networks," *IEEE J. Select. Areas Commun.*, vol. 17, pp. 326–344, Feb. 1999.

[52] R. A. Howard, *Dynamic Probabilistic Systems*. New York: Wiley, 1971.

[53] S. Hsuan-Jung and Q. Zhang, "Performance of UMTS Radio Link Control," in *Proceedings of IEEE ICC*, vol. 5, (New York, US), pp. 3346–3350, Apr. 2002.

[54] X. Hua, C. Yi-Chiun, X. Xiao, E. Gonen, and L. Peijuan, "Performance analysis on the Radio Link Control protocol of UMTS," in *Proceedings of IEEE VTC*, vol. 4, (Vancouver, Canada), pp. 1755–1759, Sept. 2002.

[55] J. Huber, D. Weiler, and H. Brand, "UMTS, the mobile multimedia vision for IMT 2000: a focus on standardization," *IEEE Commun. Mag.*, vol. 38, pp. 129–136, September 2000.

[56] C. Huitema, "The case for packet level FEC," in *International Workshop on Protocols for High Speed Networks (PsHSN 1996)*, (INRIA, Sophia Antipolis, France), pp. 109–120, October 1996.

[57] W. Ivancic, D. Brooks, B. Frantz, D. Hoder, D. Shell, and D. Beering, "NASA's Broadband Satellite Networking Research," *IEEE Commun. Mag.*, vol. 37, pp. 40–47, July 1999.

[58] V. Jacobson, "Congestion Avoidance and Control," in *Proceedings of ACM Symposium on Communications architectures and protocols, SIGCOMM*, (Stanford, CA, USA), pp. 314–329, Aug. 1988.

[59] V. Jacobson, "IETF RFC 1144: Compressing TCP/IP Headers for Low Speed Serial Links," Feb. 1990.

[60] V. Jacobson and R. Braden, "IETF RFC2018: TCP extensions for long–delay paths," October 1988.

[61] W. C. Jakes, *Microwave Mobile Communications*. Wiley–IEEE Press, May 1994.

[62] C. Keramane, "The Wireless World Web," *IEEE Multimedia*, vol. 7, pp. 10–14, April/June 2000.

[63] F. Khan, S. Kumar, K. Medepalli, and S. Nanda, "TCP performance over CDMA2000 RLP," in *Proceedings of IEEE VTC Spring*, pp. 41–45, May 2000.

[64] J. G. Kim and M. M. Krunz, "Delay Analysis of Selective Repeat ARQ for a Markovian Source Over a Wireless Channel," *IEEE Trans. Veh. Technol.*, vol. 49, pp. 1968–1981, Sept. 2000.

[65] A. G. Konheim, "A queueing analysis of two ARQ protocols," *IEEE Trans. Commun.*, vol. 28, pp. 1004–1014, July 1980.

[66] S. Kota, R. Jain, and R. Goyal, "Broadband Satellite Network Performance," *IEEE Commun. Mag.*, vol. 37, pp. 94–95, July 1999. Guest Editorial.

[67] H. Kruse, M. Allman, J. Griner, and D. Tran, "Experimentation and modelling of HTTP over satellite channels," *International Journal of Satellite Communications*, vol. 19, pp. 51–69, January/February 2001. Special Issue on IP.

[68] A. Kumar, "Comparative performance analysis of versions of TCP in a local network with a lossy link," *IEEE/ACM Trans. Networking*, vol. 6, pp. 485–498, August 1998.

[69] J. Laiho, A. Wacker, and T. Novosad, *Radio Network Planning and Optimization for UMTS*. John Wiley & Sons, Ldt, Oct. 2001.

[70] T. V. Lakshman and U. Madhow, "The performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss," *IEEE/ACM Trans. Networking*, vol. 5, pp. 336–350, June 1997.

[71] F. Lefevre and G. Vivier, "Optimizing UMTS Link Layer Parameters for a TCP Connection," in *Proceedings of IEEE VTC*, vol. 4, (Rhodes, Greece), pp. 2318–2322, May 2001.

[72] S. Lin, D. Costello, and M. Miller, "Automatic-repeat-request error control schemes," *IEEE Commun. Mag.*, vol. 22, pp. 5–17, Dec. 1984.

[73] D. Lu and J. Chang, "Performance of ARQ protocols in nonindependent channel errors," *IEEE Trans. Commun.*, vol. 41, pp. 721–730, May 1993.

[74] D. D. Lucia and K. Obraczka, "Multicast feedback suppression using representatives," in *Proceedings of IEEE INFOCOM*, (Kobe, Japan), pp. 463–470, Sept. 1997.

[75] M. Marchese, "Performance Analysis of the TCP Behavior in a GEO Satellite Environment," *Computer Communications Journal*, vol. 24, pp. 877–888, May 2001. Special Issue on the Performance Evaluation of Telecommunication Systems: Models, Issues and Applications.

[76] M. Marchese, "Proposal of a Modified Version of the Slow Start Algorithm to Improve TCP Performance over Large Delay Satellite Channels," in *Proceedings of IEEE ICC 2001*, vol. 10, (Helsinki, Finland), pp. 3145–3149, June 2001.

[77] M. Marchese, "TCP Modifications over Satellite Channels: Study and Performance Evaluation," *International Journal of Satellite Communications*, vol. 19, pp. 93–110, January/February 2001. Special Issue on IP.

[78] M. Marchese, *Quality Survivability and Reliability of Large Scale Telecommunications*. John Wiley & Sons, New York, P. Stavroulakis ed., January 2003. Chapter: TCP/IP - based Protocols over Satellite Systems: A Telecommunication Issue.

[79] A. J. McAuley, "Reliable Broadband Communications Using a Burst Erasure Correcting Code," in *Proceedings of ACM SIGCOMM 1990*, (Philadelphia, PA, US), pp. 287–306, Sept. 1990.

[80] J. Nonnemacher, *Reliable Multicast Transport to Large Groups*. Ph.D Thesis, Ecole Polytechnique Federale de Lausanne, France, 1998. [Available] `http://citeseer.nj.nec.com/nonnemacher98reliable.html`.

[81] J. Nonnenmacher, E. Biersack, and D. Towsley, "Parity–based loss recovery for reliable multicast transmission," *IEEE/ACM Trans. Networking*, vol. 6, pp. 349–361, Aug. 1998.

[82] D. Bertsekas and R. Gallager, *Data Network*. Prentice Hall, 2 ed., 1992.

[83] H. Holma and A. Toskala, *WCDMA for UMTS: Radio Access for Third Generation Mobile Communications*. John Wiley & Sons, Ldt, 2000.

[84] J. Postel, "IETF RFC 793: Transmission Control Protocol - Darpa Internet Program - Protocol Specification," September 1981.

[85] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP reno performance: a simple model and its empirical validation," *IEEE/ACM Trans. Networking*, vol. 8, pp. 133–145, Apr. 2000.

[86] C. Partridge and T. J. Shepard, "TCP/IP Performance over Satellite Links," *IEEE Network*, vol. 11, pp. 44–49, September/October 1997.

[87] G. Patel and S. Dennett, "The 3GPP and 3GPP2 movements toward an all-IP mobile network," *IEEE Personal Commun. Mag.*, vol. 7, pp. 62–64, August 2000.

[88] V. Paxson and M. Allman, "IETF RFC 2988: Computing TCP's Retransmission Timer," Nov. 2000.

[89] S. J. Perkins and M. W. Mutka, "Dependency Removal for Transport Protocol Header Compression over Noisy Channels," in *Proceedings of IEEE ICC*, (Montreal, Canada), pp. 1025–1029, June 1997.

[90] L. Rizzo, "Effective Erasure Codes for Reliable Computer Communications Protocols," *ACM Computer Communication Review*, vol. 27, pp. 24–36, Apr. 1997.

[91] Z. Rosberg and N. Shacham, "Resequencing delay and buffer occupancy under the Selective Repeat ARQ," *IEEE Trans. Inform. Theory*, vol. 35, pp. 166–173, Jan. 1989.

[92] Z. Rosberg and M. Sidi, "Selective-Repeat ARQ: The joint distribution of the transmitter and the receiver resequencing buffer occupancies," *IEEE Trans. Commun.*, vol. 38, pp. 1430–1438, Sept. 1990.

[93] S. M. Ross, *Stochastic Processes*. New York: Wiley, 2 ed., 1996.

[94] M. Rossi, L. Badia, and M. Zorzi, "Exact statistics of arq packet delivery delay over Markov channels with finite round-trip delay," in *Proceedings of IEEE Globecom*, vol. 6, (San Francisco, California, US), pp. 3356–3360, Dec. 2003.

[95] B. Sarikaya, "Packet mode in wireless networks: overview of transition to third generation," *IEEE Commun. Mag.*, vol. 38, pp. 164–172, September 2000.

[96] Signal Processing & Multimedia Group, The University of British Columbia, "H.263/H.263+ Research Library, Release 0.2." [Available] `http://spmg.ece.ubc.ca`.

[97] W. R. Stevens, *TCP/IP Illustrated, Volume 1*. Addison Wesley, 1994.

[98] W. R. Stevens, "IETF RFC 2001: TCP Slow start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithm," Jan. 1997.

[99] C. C. Tan and N. C. Beaulieu, "On First-Order Markov Modeling for the Rayleigh Fading Channel," *IEEE Trans. Commun.*, vol. 48, pp. 2032–2040, Dec. 2000.

[100] V. Tsaoussidis, H. Badr, X. Ge, and K. Pentikousis, "Energy/throughput tradeoffs of TCP error control strategies," in *Proceedings of ISCC*, pp. 106–112, July 2000.

[101] V. Tsaoussidis, A. Llahanns, and H. Badr, "Wave & wait protocol (WWP): high throughput and low energy for mobile IP-devices," in *Proceedings IEEE ICON*, pp. 469–473, September 2000.

[102] H. S. Wang and N. Moayeri, "Finite-state Markov Channel–A Useful Model for Radio Communication Channels," *IEEE Trans. Veh. Technol.*, vol. 44, pp. 163–171, Feb. 1995.

[103] X. Xiao, C. Yi-Chiun, X. Hua, E. Gonen, and L. Peijuan, "Simulation analysis of RLC timers in UMTS systems," in *Proceedings of the Winter Simulation Conference*, vol. 1, (San Diego, CA), pp. 506–512, Dec. 2002.

[104] Q. Zang and S. A. Kassam, "Finite-State Markov Model for Rayleigh Fading Channels," *IEEE Trans. Commun.*, vol. 47, pp. 1688–1692, Nov. 1999.

[105] Y. Zhang, D. D. Lucia, B. Ryu, and S. K. Dao, "Satellite Communications in the Global Internet: Issues, Pitfalls, and Potential," in *Proceedings of the Internet Global Summit Conference (INET'97)*, (Kuala Lumpur, Malaysia), Jun 1997.

[106] M. Zorzi, A. Chockalingam, and R. Rao, "Throughput Analysis of TCP on Channels with Memory," *IEEE J. Select. Areas Commun.*, vol. 18, pp. 1289–1300, July 2000.

[107] M. Zorzi, G. Mazzini, V. Tralli, and A. Giovanardi, "Some results on the error statistics over a WCDMA air interface," in *Proceedigs of MMT*, (Florida (USA)), December 2000.

[108] M. Zorzi and R. Rao, "Energy constrained error control for wireless channels," *IEEE Personal Commun. Mag.*, vol. 4, pp. 27–33, December 1997.

[109] M. Zorzi and R. Rao, "On the Statistics of Block Errors in Bursty Channels," *IEEE Trans. Commun.*, vol. 45, pp. 660–667, June 1997.

[110] M. Zorzi and R. Rao, "On channel modeling for delay analysis of packet communications over wireless links," in *36th Annual Allerton Conference On Communications, Control And Computing*, (Allerton House, Monticello, IL), Sept. 1998.

[111] M. Zorzi and R. Rao, "Is TCP energy efficient?," in *Proceedings of IEEE MoMuC*, November 1999.

[112] M. Zorzi and R. Rao, "Lateness probability of a retransmission scheme for error control on a two-state Markov channel," *IEEE Trans. Commun.*, vol. 47, pp. 1537–1548, Oct. 1999.

[113] M. Zorzi and R. Rao, "Perspectives on the Impact of Error Statistics on Protocols for Wireless Networks," *IEEE Personal Commun. Mag.*, vol. 6, October 1999.

[114] M. Zorzi and R. Rao, "Energy Efficiency of TCP in a Local Wireless Environment," *MONET, Mobile Networks and Applications*, vol. 6, pp. 265–278, June 2001.

[115] M. Zorzi, R. Rao, and L. Milstein, "Error statistics in data transmission over fading channels," *IEEE Trans. Commun.*, vol. 46, pp. 1468–1476, Nov. 1998.

[116] M. Zorzi and R. R. Rao, "Energy Efficiency of TCP in a Local Wireless Environment," *Mobile Networks and Applications, Kluwer Academic Publishers*, vol. 6, pp. 265–278, July 2001.

[117] M. Zorzi, R. R. Rao, and L. B. Milstein, "On the accuracy of a first order Markov model for data transmission on fading channels," in *Proceedings of IEEE ICUPC*, (Tokyo, Japan), pp. 211–215, Nov. 1995.

Biography

Michele Rossi was born in Ferrara, Italy on october 30, 1974. He received the Laurea degree *summa cum laude* in electrical engineering from the University of Ferrara in march 2000. During the academic year 2000/2001, he was a Research Fellow at the Department of Engineering, University of Ferrara. Between april 15 to october 15, 2003, he has been with the Center for Wireless Communications (CWC) at the University of California, San Diego, under the supervision of Prof. Ramesh R. Rao. During this period he has performed research in wireless Sensor Networks.

His research interests are in: performance evaluation of TCP/IP protocols over wireless networks, TCP/IP Header Compression algorithms, performance analysis of ARQ retransmission techniques, efficient multicast data delivery in cellular networks with particular focus on the video streaming case, multicast in cellular networks and its mobility support. Recently he started to perform research in the area of Sensor Networks with particular focus in the design of energy/delay efficient routing algorithms.

During the last four years Michele Rossi has been responsible for the technical collaboration between the University of Ferrara and some national and international institutions and companies. Among others, we cite here ERICSSON, ESA (European Space Agency) and CNIT (Consorzio Nazionale Interuniversitario per le Telecomunicazioni).

Michele Rossi is currently an IEEE Member and a CNIT member.

List of Publications of Michele Rossi

International Journals

[1] M. Rossi, R. Vicenzi, M. Zorzi, "Throughput Analysis of TCP on Channels with Memory and finite round trip delay," to appear in *IEEE Transactions on Wireless Communications*.

[2] C. F. Chiasserini, F. Cuomo, L. Piacentini, M. Rossi, I. Tinnirello, F. Vacirca, "Architectures and protocols for mobile computing applications: a reconfigurable approach," *Elsevier Computer Networks*, Vol. 44, No. 4, Mar. 2004, pp: 411–582.

[3] M. Marchese, M. Rossi, G. Morabito, "PETRA: Performance Enhancing Transport Architecture for Satellite Communications," *IEEE Journal on Selected Areas in Communications* (JSAC), Vol. 22, No. 2, Feb. 2004, pp: 320–332.

[4] M. Rossi, M. Zorzi, ”Analysis and heuristics for the characterization of selective repeat ARQ delay statistics over wireless channels,” *IEEE Transactions on Vehicular Technology*, Vol. 52, No. 5, Sept. 2003, pp: 1365–1377.

[5] M. Zorzi, M. Rossi, G. Mazzini, ”Throughput and energy performance of TCP on a wideband CDMA air interface,” *Wireless Communications and Mobile Computing* (WCMC), John Wiley & Sons, Vol. 2, No. 1, Feb. 2002, pp:71–84.

[6] A. Giovanardi, G. Mazzini, M. Rossi, M. Zorzi, ”Improved Header Compression for TCP/IP over Wireless Links,” *IEE Electronic Letters*, Vol. 36, No. 23, Nov. 2000, pp. 1958–1960.

International Conferences

[1] M. Rossi, L. Badia, M . Zorzi, ”Exact statistics of ARQ packet delivery delay over Markov channels with finite round-trip delay,” *IEEE Global Telecommunications Conference, IEEE Globecom 2003*, San Francisco, Dec. 1–5, 2003.

[2] M. Rossi, L. Scaranari, M. Zorzi, ”On the UMTS RLC Parameters Setting and their Impact on Higher Layers Performance,” *IEEE Vehicular Technology Conference, IEEE VTC Fall 2003*, Orlando, Florida. Oct. 6–9, 2003.

[3] M. Rossi, F. Fitzek, M. Zorzi, ”Error Control Techniques for Efficient Multicast Streaming in UMTS Networks,” *The 7th World Multiconference on Systemics, Cybernetics and Informatics, IIIS SCI 2003*, Orlando, Florida, USA. Jul. 27–30, 2003.

[4] M. Rossi, M. Zorzi, ”On the UMTS RLC configuration under delay constraints,” *IEEE Vehicular Technology Conference, IEEE VTC Spring 2003*, Jeju, Korea, Apr. 22–25, 2003.

[5] M. Rossi, L. Badia, M. Zorzi, ”Accurate approximation of ARQ packet delay statistics over Markov channels with finite round-trip delay,” *IEEE Wireless Communications and Networking Conference, WCNC 2003*. Louisiana, New Orleans, USA. Mar. 16–20, 2003.

[6] M. Rossi, L. Badia, M. Zorzi, ”On the delay statistics of an aggregate of SR-ARQ packets over Markov channels with finite round-trip delay,” *IEEE Wireless Communications and Networking Conference, WCNC 2003*. Louisiana, New Orleans, USA. Mar. 16–20, 2003.

[7] A. Roveri, C.F. Chiasserini, M. Femminella, T. Melodia, G. Morabito, M. Rossi, I. Tinnirello, ”The RAMON module: architecture framework and performance results,” *Proccedings of 2nd international workshop on QoS in Multiservice IP Networks, QoS–IP 2003*, Milano (Italy). Feb. 24–26, 2003.

[8] G. Morabito, S. Palazzo, M. Rossi, M. Zorzi, ”Improving End-To-End Performance in Reconfigurable Networks through Dynamic Setting of TCP Parameters,” *Proceedings of 2nd international workshop on QoS in Multiservice IP Networks, QoS–IP 2003*, Milano (Italy). Feb. 24–26, 2003.

[9] D. Adami, M. Marchese, G. Morabito, M. Rossi, L. Veltri, ”Transport Protocol and Resource Management for Satellite Networks: Framework of a Project,” *5th European Workshop on Mobile/Personal Satcoms, EMPS 2002*, Baveno-Stresa, Lake Maggiore, Italy. Sep. 25–26, 2002.

[10] A. Giovanardi, G. Mazzini, M. Rossi, ”Analysis and Optimization of a Transparent Multicast Mobility Support in Cellular Systems,” *IEEE International Conference on Communications, EEE ICC 2002*, New York, USA. Apr. 28–May 2, 2002.

[11] M. Zorzi, M. Rossi, G. Mazzini, ”Performance of TCP on a wideband CDMA air interface,” *Tyrrhenian International Workshop on Digital Communications, Evolutionary Trends of the Internet*, IWDC 2001, Sep. 17–20, 2001.

[12] M. Rossi, A. Giovanardi, M. Zorzi, G. Mazzini, ”TCP/IP Header Compression: Proposal and Performance Investigation on a WCDMA Air Interface,” *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, IEEE PIMRC 2001*, San Diego, USA. Sep. 30–Oct. 3, 2001.

[13] A. Giovanardi, G. Mazzini, M. Rossi, ”An Agent-based Approach for Multicast Applications in Mobile Wireless Networks,” *IEEE Global Telecommunications Conference, IEEE Globecom 2000*, San Francisco, USA. Nov. 27–Dec. 1, 2000.