POLITECNICO DI MILANO

Analysis of Internet Transport Service Performance with Active Queue Management in a QoS-enabled network

carried out at

University of Helsinki

Department of Computer Science

under the guidance of

Prof. Kimmo Raatikainen

Dr. Markku Kojo

Dr. Antonio Capone

by

Oriana Riva

April 2003 _____

Ai miei genitori

e

a mio fratello

Pater ipse colendi
haud facilem esse viam voluit, primusque per artem
movit agros, curis acuens mortalia corda
nec torpere gravi passus sua regna veterno.
Ante Iovem nulli subigebant arva coloni:
ne signare quidem aut partiri limite campum
fas erat; in medium quaerebant, ipsaque tellus
omnia liberius nullo poscente ferebat.
Ille malum virus serpentibus addidit atris
praedarique lupos iussit pontumque moveri,
mellaque decussit foliis ignemque removit
et passim rivis currentia vina repressit,
ut varias usus meditando extunderet ignem.
Tunc alnos primum fluvii sensere cavatas;
navita tum stellis numeros et nomina fecit
Pleiadas, Hyadas, claramque Lycaonis Arcton.
Tum laqueis captare feras et fallere visco
inventum et magnos canibus circumdare saltus;
atque alius latum funda iam verberat amnem
alta petens, pelagoque alius trahit umida lina.
Tum ferri rigor atque argutae lamina serrae
(nam primi cuneis scindebant fissile lignum),
tum varie venere artes. Labor omnia vicit
improbus et duris urgens in rebus egestas.

[Virgilio, Georgiche I, 121-146]

This work was carried out at the Department of Computer Science, University of Helsinki. It is my great pleasure to thank all the people who supported me in creating this Master's Thesis.

Special thanks to my supervisor Prof. Kimmo Raatikainen who gave me the great opportunity of working at the University of Helsinki and assisted me with his experience and interest. His sense of humour and enthusiastic attitude towards research and life in general have been more than I could have ever asked for.

I wish to express my appreciation to my supervisor Dr. Antonio Capone for his interest towards my research and for his critical appraisal of this thesis.

Special thanks to Dr. Markku Kojo for providing me with invaluable advice and guidance throughout this research. He created a congenial atmosphere through regular and informal meetings. It has been a pleasure working under him.

I thank all my friends and colleagues for their valuable comments, advice and encouragement. They have created a warm and sunny atmosphere also during the cold and dark winter times, making me feel part of a great family.

My warm thanks are due minun ensimmäiselle hyvälle suomalaiselle ystävälleni, Jukka Manner, Ph. Lic., for his useful advice and his constant technical support. Thanks to my office mates, Davide Astuti, M. Sc., and Simone Leggio, M. Sc., for their friendly help and "Sicilian" vitality. Finally I wish also to thank Tiina Niklander, Ph. Lic., for her kind support and for giving me untiring help during my difficult moments.

Thanks to all my Italian friends with whom I spent nice moments by e-mail. Moreover I wish to extend my warmest thanks to all my friends and my professors at the Politecnico di Milano, who have helped me during five long years of studies whose this thesis represents the final step.

Finally I would most like to thank my parents Adele Formenti and Francesco Riva, and my brother Claudio to whom this thesis is dedicated, who trusted and motivated me during my student career. Completion of this work would have been impossible without them.

Questo lavoro e' stato sviluppato al Dipartimento di Informatica dell' University of Helsinki. E' con grande piacere che ringrazio tutte le persone che mi hanno suportato nella creazione di questa tesi.

Vorrei in primo luogo ringraziare il Prof. Kimmo Raatikainen che mi ha concesso la grande opportunita' di lavorare all'University of Helsinki e che mi ha assistito con la sua esperienza e il suo interesse. Il suo senso dell'umorismo e il suo entusiasmo verso la ricerca e la vita in genere sono stati piu' di quanto potessi desiderare.

Vorrei ringraziare e esprimere il mio apprezzamento per il mio supervisore Dott. Antonio Capone per il suo interesse verso la mia ricerca e per la sua valutazione critica di questa tesi.

Ringrazio in modo speciale il Dott. Markku Koio per i suoi consigli e la sua guida durante lo sviluppo dell'intero lavoro. E' riuscito a creare una piacevole atmosfera attraverso incontri regolari e informali. E' stato un piacere lavorare con lui.

Ringrazio tutti i miei amici e colleghi per i loro preziosi commenti, consigli e incoraggiamenti. Sono riusciti a regalarmi una calda e solare atmosfera anche durante il freddo e buio inverno finalndese facendomi sentire parte di una grande famiglia.

Grazie di cuore al mio primo grande amico finlandese, Jukka Manner, per i suoi utili consigli e il suo supporto tecnico. Vorrei anche ringraziare i miei colleghi di ufficio, Davide Astuti e Simone Leggio, per il loro aiuto e la loro vitalita'. Inoltre un sentito grazie a Tiina Niklander, per la sua gentilezza e per essermi stata vicina nei momenti piu' difficili.

Grazie a tutti i miei amici italiani con cui ho potuto trascorrere piacevoli momenti via e-mail. E in particolare grazie a tutti i professori e agli amici del Politecnico di Milano con cui ho condiviso cinque lunghi anni di studio di cui questa tesi rappresenta l'ultimo passo.

Ma soprattutto vorrei ringraziare i miei genitori, Adele Formenti e Francesco Riva, e mio fratello Claudio a cui questa tesi e' dedicata, che mi hanno motivato e incoraggiato durante i miei studi. Il completamento di questo lavoro non sarebbe stato possibile senza di loro.

Table of Contents

Table of Contents ..... **v**
List of Figures ..... **vii**
List of Tables ..... **ix**
Abstract ..... **x**
1 Introduction **1**
2 Background **6**
2.1 Transmission Control Protocol (TCP) 6
2.1.1 Protocol Operation 7
2.1.2 Congestion Control 11
2.1.3 TCP Retransmission calculation 16
2.2 Quality of Service in the Internet 17
2.2.1 QoS Technologies 18
2.2.2 Differentiated Services Architecture 19
2.3 Wireless Networking Environment 25
2.3.1 Properties of Wireless Links 27
2.3.2 Transport over Wireless Networks 28
3 Active Queue Management **35**
3.1 Need for Active Queue Management 35
3.2 RED congestion control mechanism 37
3.2.1 RED algorithm 37
3.2.2 Parameter sensitivity 42
3.2.3 Alternative RED mechanisms 43
3.3 Explicit Congestion Notification (ECN) 48
4 Test Arrangements **51**
4.1 Test Methodology 51
4.1.1 Parameter Tuning for RED 53
4.2 Experimentation Environment 54
4.3 Workload Models 57
4.4 Metrics 59
4.5 Test Cases 60
4.5.1 Tests with three competing TCP flows 61
4.5.2 Tests with TCP and UDP traffic with services differentiation 62

5 Test Results and Analysis 66
5.1 Tests with three competing TCP flows 66
5.1.1 Tests with Tail Drop 66
5.1.2 Comparison of Tail Drop and RED performance 81
5.1.3 RED Parameters 87
5.1.4 Summary 87
5.2 Tests with TCP and UDP traffic with services differentiation 89
5.2.1 Test Case 1: long TCP transfer competing with UDP traffic 90
5.2.2 Test Case 2: short TCP transfer competing with UDP traffic 94
5.2.3 Test Case 3: long TCP transfer competing with UDP traffic for a short time 100
5.2.4 Summary 102
6 Conclusions 105
References 110
7 Appendix A Riassunto in Italiano 117
8 Appendix B Full Test Results 124
8.1 Tests with three competing TCP flows 124
8.2 Tests with TCP and UDP traffic with services differentiation 130

List of Figures

Figure 2-1. The TCP segment format. 7
Figure 2-2. The connection setup. 9
Figure 2-3. The connection tear-down. 11
Figure 2-4. TCP Congestion Window 13
Figure 2-5. The DiffServ Architecture 20
Figure 2-6. The DS field in the IP header. 21
Figure 2-7. Block diagram of a traffic classifier and a traffic conditioner. 22
Figure 3-1. Detailed algorithm for RED gateways [FJ93]. 38
Figure 3-2. Instantaneous queue length [[CJOS02] 39
Figure 3-3. Dropping/Marking behaviour of RED. 40
Figure 3-4. Comparison between two marking probability method [FJ93]. 41
Figure 3-5. The RED Adaptive algorithm. 44
Figure 3-6. RIO algorithm. 46
Figure 3-7. The BLUE algorithm. 47
Figure 3-8. The DS and the ECN Fields in the IP-header [RFB01]. 48
Figure 3-9. The ECN Field [RFB01]. 48
Figure 3-10. The TCP header and the definition of bits 8 and 9 to support ECN [RFB01]. 49
Figure 4-1. Topology of testbed network. 54
Figure 4-2. A simple queueing discipline with multiple classes. [ASK99]. 55
Figure 4-3. Example of HTB queueing discipline 56
Figure 4-4. Workload model in the Tests run with 3 TCP competing flows. 61
Figure 4-5. Class hierarchy of two HTB configurations. 63
Figure 5-1. The Elapsed Time of the three competing flows for 12 third flow's start times (median values based on 20 replications). 69
Figure 5-2. The Throughput of the three competing flows for 12 third flow's start times (median values based on 20 replications). 69
Figure 5-3. Example of fairness between the biggest flows and high performance of the third one in the scenario *after 0 seconds.* 72
Figure 5-4. The case of Maximum Throughput of the third flow up to all the experiments run under Tail Drop (the third flow's *optimum* with Tail Drop). 73
Figure 5-5. The minimum throughput of the third flow in the scenario *after 2 seconds.* 74
Figure 5-6. The maximum throughput for the 3rd flow in the scenario *after 2 seconds* (Throughput=4497 Bytes/s, Elapsed Time=10.4sec and Data Transmit Time=8.7sec). 75

Figure 5-7. Minimum Data Transmit Time for the scenario *after 2 seconds* (Throughput=3907 Bytes/s, Elapsed Time=12 sec and Data Transmit Time=6.8 sec).....76

Figure 5-8. The third flow starts in the last part of the Slow Start phase of the initial flow in the scenario *after 2 seconds* (3rd flow: Throughput=2937Bytes/s, Elapsed Time=16sec).....77

Figure 5-9. The third flow starts in the middle part of the Slow Start phase of the initial flow (3rd flow: Throughput=2206 Bytes/s, Elapsed Time=21.2sec).....77

Figure 5-10. The third flow starts at the beginning of the Slow Start phase of the initial flow in the scenario *after 2 seconds* (3rd flow: Throughput=1872Bytes/s, Elapsed Time=25sec).....78

Figure 5-11. Example of great performance of the fastest flow (Throughput=9914 Bytes/s and Elapsed Time=29.5sec) to the detriment of the third one (Throughput=3370 Bytes/s and Elapsed Time=13.9sec).....80

Figure 5-12. The third flow achieves the Max Throughput (4709 Bytes/s) and the performance of the fastest flow degrades (Throughput=8814 Bytes/s).....80

Figure 5-13. Comparison between Tail Drop and RED perfomance for the third flow (data sent=46.7Kbytes).....83

Figure 5-14. The Elapsed Time and the Number of Retransmitted packets of the fastest and slowest flow (data sent=292Kbytes).....84

Figure 5-15. Example of RED fairness in the scenario *after 5.5 seconds*. The throughput of the fastest and the slowest flows are 8062 Bytes/s and 7988 Bytes/s respectively. The throughput of the third flow is 7190 Bytes/s.....84

Figure 5-16. Statistics of the Test Case 1: long TCP transfer competing with UDP traffic (median values of 20 replications).....90

Figure 5-17. Analysis of the scenario *after 3 seconds* of the Test Case 1.....93

Figure 5-18. Statistics of the Test Case 2: short TCP transfer competing with UDP traffic (median values of 20 replications).....94

Figure 5-19. Test Case 2: examples of TCP connections taken from the scenario *after 1.5 seconds* (on the left) and from the scenario *after 0 second* (on the right).....95

Figure 5-20. Test Case 2-Scenario *after 0.5 seconds:* Tail Drop + DS (on the left) and RED + DS (on the right).....97

Figure 5-21. Statistics of the additional Test Case: medium TCP traffic competing with UDP traffic (median values of 20 replications).....99

Figure 5-22. Statistics of the Test Case 3: long TCP transfer competing with UDP traffic of short duration (median values of 20 replications).....100

Figure 5-23. Test Case3: Sequence Number graphs when Tail Drop is deployed in the scenario *after 0 seconds* (left), *after 7 seconds* (middle) and *after 18 seconds* (right).....101

List of Tables

Table 4-1. The RED Parameters 53

Table 4-2. Network traffic characteristics. 58

Table 4-3. Traffic characteristics in the Tests run with 3 TCP competing flows 62

Table 4-4. The RED parameters for the HTB configurations. 63

Table 4-5. The workload models for the Test with TCP and UDP traffics. 64

Table 5-1. Statistics for the 20 replications of the TCP connections when Tail Drop is employed. 67

Table 5-2. Statistics for the scenario *after 0 seconds*, based on 20 replications. 71

Table 5-3. The performance achieved in the case of third flow's maximum throughput (scenario *after 0 seconds*) 73

Table 5-4. Statistics of the Elapsed Time when the RED algorithm is employed (20 replications). 85

Table 5-5. The statistics of the optimum in the RED tests (Max Throughput and min Elapsed Time). 85

Table 5-6. Additional tests oriented to study the tuning of RED parameters (Median value of the 3rd flow's elapsed time, based on 20 replications). 87

Table 5-7. Bandwidth Utilizations in the scenario *after 0 sec* of the Test Case 1 (long TCP transfer competing with UDP traffic). 92

Table 8-1. 20 Replications of the scenario *after 0 seconds*. Three TCP competing flows when Tail Drop discipline is employed. 125

Table 8-2. Three TCP flows when Tail Drop is employed. The third flow starts after 2 seconds 126

Table 8-3. The statistics for the scenario *after 2 seconds* when Tail Drop is in use 127

Table 8-4. The statistics for the scenario *after 2 seconds* when RED is in use 127

Table 8-5. Three TCP competing flows when Tail Drop is in use. The third flow starts *after 5.5 seconds*. 128

Table 8-6. The statistics for the scenario *after 5.5 seconds* when Tail Drop is deployed. 129

Table 8-7. The statistics for the scenario *after 5.5 seconds* when RED discipline is adopted. 129

Table 8-8. Statistics for Test Case 1: long TCP transfer competing with UDP traffic. 131

Table 8-9. Statistics for the Test Case 2: Short TCP transfer competing with UDP traffic. 132

Table 8-10. Statistics for the Test Case 3: long TCP transfer competing with UDP traffic for a short time. 132

Abstract

Active Queue Management is receiving wide attention as promising technique to prevent and avoid congestion collapse in packet-switched networks. It is a form of router queue management based on a proactive approach. By providing advanced warning of incipient congestion, end nodes can respond to congestion before router buffer overflows and finally ensure improved performance. In this thesis we represent a performance analysis of Random Early Detection (RED) algorithm. This is a recommended active queue management scheme, that is expected to provide several Internet performance advantages such as minimizing packet loss and router queueing delay, avoiding global synchronization of sources, guaranteeing high link utilization and fairness. In this study we consider a wired testbed network including two routers and a last-hop link of limited bandwidth. In the last–hop router congestion occurs and packets are dropped. The router alternatively employs the traditional Tail Drop discipline or RED algorithm as queue management mechanism. We examine the impact of both disciplines on TCP traffic and compare the results. We show that RED provides fair sharing of the bottleneck capacity and avoids phenomena such as lock-out inherent in Tail Drop. It tends to drop packets from each connection in proportion to the transmission rate the flow has on the output link. Moreover it is more available to serve incoming traffic, apart from its data load and arrival time guaranteeing steady results. Given this, we then experiment with RED by introducing services differentiation. We investigate how the arrival of high priority UDP traffic can hurt the performance of lower priority TCP traffic when they share the same bottleneck link with one or two classes of service. RED does not minimize the number of dropped packets as expected, but it manages to achieve improved performance in respect of Tail Drop. Moreover we find out that even though the arrival of UDP competing traffic generally hurts the performance of the TCP connection running, surprisingly it helps Tail Drop to improve its TCP results when it starts simultaneously with the TCP connection. Indeed it prevents the Slow Start phase from being excessively aggressive thus to avoid phenomena such as Slow Start overshoot and severe congestion states. Most of the theoretical properties of RED find confirmation in our results and we believe that RED will help to provide fair sharing of resources and improved performance in a wide range of environments, with a variable number of connections with different data loads and throughput.

1 Introduction

End-to-end congestion control mechanisms are widely employed in the current Internet in order to regulate the amount of traffic in the network and match it to the available capacity. The objective is to assure that router buffer queue lengths and packet loss rates remain reasonable, thus to prevent congestion collapse. The most popular congestion avoidance algorithms in today's Internet are based on the window mechanism of the *Transport Control Protocol (TCP)* [Pos81b]. Thanks to their ability to deliver service in times of extremely high demand and provide a solid foundation for several applications, Internet has seen an enormous success. Nevertheless, this rapid growth has led it to be accessed by more and more devices and to see increasing requests for new services and improved performance. In order to satisfy these new challenges, a number of TCP enhancements have made their way into actual implementations, but it has become clear that TCP congestion avoidance mechanisms, even if necessary and powerful, are not enough to provide good service in all the circumstances. The main problem is that there is a limit to how much control can be achieved from the edge of the network, since the view of an individual TCP connection is too limited and needs to be complemented by some feedback from the router itself.

TCP is a connection-oriented protocol and provides reliability by recovering data that is damaged, lost, duplicated, or delivered out-of-order by the Internet Communication System. It employs a sliding window protocol to achieve flow control and implements congestion control mechanisms to prevent congestion collapse. The main drawback in congestion control and avoidance mechanisms used in TCP is that the network fails to provide early congestion notification to the sources. TCP's congestion control algorithms are based on the principle that the network state of congestion is detected by packet loss. When a connection starts up, it attempts to ramp its transmission rate quickly, by exponentially increasing its congestion window. If the window reaches the maximum threshold a phase of congestion avoidance starts, in which the window is increased at a much slower rate. When the network capacity saturates, because of the load of active connections, a packet may be discarded and packet loss is detected by the source through the receipt of duplicate acknowledgements or timeout expiration. This indicates that the network is congested and the senders react reducing the transmission rate in order to avoid further packet loss and congestion collapse. Thereby the TCP senders decrease their rates only after detecting packet loss due to queue overflow, but this mechanism may need

a long time and other packets may be discarded before the source attenuates its transmission rate. The consequence is that TCP connections experience high loss rate, especially during times of congestion.

Moreover traditional Internet routers employ the *Tail Drop* discipline for managing the buffer queue occupancy. It simply sets a maximum length for each buffer queue and it enqueues packets until the maximum length is reached, then drops subsequent incoming packets until the queue is decreased below its maximum value. Such a mechanism allows the router to maintain high queue occupancy, which is clearly undesirable since it tends to discriminate against bursty traffic and to drop many packets at the same time producing global synchronization of sources. Being the data traffic in Internet inherently bursty, one way to alleviate the problem is to provide the routers with fairly large buffers in order to absorb burst arrivals of packets to reduce losses and hence maintain high link utilization. On the other hand, large buffers tend to increase queueing delays at congested routers. The traditional Tail-Drop buffer management forces network applications to choose between high utilization or low delay.

To alleviate these problems a possible solution is an advanced form of router queue management, which refers to the decision when to start discarding packets and which packets to drop at a congested router. The Internet Engineering Task Force (IETF) is working in order to provide new techniques such as *Active Queue Management (AQM)* [BCCD98] and *Explicit Congestion notification (ECN)* [RFB01], as means to prevent and avoid congestion collapse of the network. The goal of Active Queue Management schemes is to detect congestion before the queue overflows and thus signal the incipient congestion to the end nodes. In this way senders may be able to reduce their rates before congestion collapse. Active queue management algorithms may use different methods to convey congestion notification to the end-hosts. It can drop packets or, more efficiently, mark them by setting the Congestion Experienced (CE) codepoint in the packet header. The possible improvements introduced by AQM in combination with ECN could be especially beneficial for interactive and low-bandwidth traffic where the user is delay-sensitive.

A recent Internet Draft (referred to as the RED Manifesto [BCCD98]) singles out the *Random Early Detection (RED)* algorithm, proposed by Floyd and Jacobson in [FJ93], as the recommended scheme for use in the Internet. RED is nowadays widely implemented in commercially available routers. The essence of this algorithm is that the decision of dropping a packet is based on the estimation of the average queue length in the router buffer. Basically if the queue has been mostly empty in the recent past, RED will not tend to discard packets;

otherwise if it has been almost full lately, new packets will be more easily dropped. A RED gateway drops incoming packets with a dynamically computed probability just when the average number of packets queued exceeds a minimum threshold. The drop/marking probability increases as the estimated average queue size grows. In such a way this approach controls the queue length more effectively than other existing algorithms. In particular RED's goal is to discard packets from each flow in proportion to the amount of bandwidth the flow uses on the output link. Therefore, the connection with the largest input rate will have the biggest drop percentage among total dropped packets, thus to guarantee a fair sharing of the available resources and avoiding synchronization and lock-out problems. Moreover, by avoiding full queues, RED is expected to increase effective network utilization and decrease end-to-end latency generally due to long queueing delays.

Besides the need to better control packet loss and prevent congestion collapse in the Internet there is another interesting issue. The rapid growth of the Internet has led it to be accessed by more and more devices and to see an increasing request for new services and high performance. Moreover, the heterogeneity of the traffic supported has increased considerably, thanks to the development of new applications such as those based on audio and video. As a result of this evolution, a need for services differentiation in the network has become evident in order to guarantee to each kind of application an appropriate level of service.

The two main approaches to satisfy heterogeneous application requests and provide QoS services are the *Integrated Services Architecture (IntServ)* [BCS94] and *Differentiated Services Architecture (DiffServ)* [BBCD98]. IntServ is the Internet incarnation of the traditional circuit-based architecture and its main advantage relies on the high accuracy of resource management, but it has not been deployed since it is complex to implement and poor in scalability. DiffServ does not employ per-flow state and has better scaling properties. Recently AQM mechanisms have been proposed within the framework of the Internet DiffServ architecture to preferentially drop non-conforming over conforming packets. Moreover, a key-technology for realizing differentiated services is an efficient packet dropping policy, hence RED or some enhancement of it, have been studied as the discipline to adopt. In order to provide QoS on the Internet there is substantial value in following an evolutionary path in which the attention regards more router queue managements and forwarding behaviours than virtual circuits and end-to-end services.

Finally another interesting problem related to the many Internet applications is the Wireless Networking Environment. At the beginning, TCP was specified for wired networks and stationary hosts, but nowadays, nomadic users want to run their favorite applications that are built on TCP over a wireless connection, as well. TCP performs poorly over wireless links

since the nature of wireless links is quite different compared to wireline networks for several aspects, such as limited bandwidth, high latency, variable delays and elevated error rate. Therefore, TCP interprets all packet losses as notifications of congestion in the network and the corrective actions taken, such as lowering the transmission rate, tend to cause suboptimal performance. Among the numerous solutions proposed to ameliorate TCP's performance over wireless links or high loss links there are some TCP improvements such as *TCP Selective Acknowledgments Options* [FMMP00], Active Queue Management and Explicit Congestion Notification. If in general it is important to avoid congestion collapse, this is even more relevant when the network presents long delay.

This thesis presents an experimental evaluation of RED in a testbed wired environment. The key-element of the testbed network is the last-hop router of limited buffer-size that alternatively employs traditional Tail Drop discipline or RED algorithm for controlling the buffer queue occupancy. On the corresponding output link we cause a bottleneck limiting the bandwidth through means of *Linux Traffic Control* [HMVV02]. This allows us to examine how RED and Tail Drop can prevent and eventually recover from congestion and packet loss. The data communication uses the TCP and in some cases UDP protocol. We have used the recommended TCP implementation of the Linux OS. Moreover, the traffic control framework available in the Linux kernel enabled us to experiment with RED by introducing services differentiation. The main component is a queueing discipline that maps the traffic to different service classes.

The first goal of our analysis is to explore how well the state-of-the-art TCP performs when Tail Drop or RED is deployed. It is mainly an analytic study of RED algorithm and proposes an evaluation of the benefits introduced by RED over the tradition Tail Drop discipline. The result is essentially a comparative study of performance, but we also present a detailed analysis of RED mechanisms and the key reasons behind its behaviour. The second issue is to investigate how the TCP and marginally UDP behave when applied to a QoS-enabled environment, in which incoming traffic is classified into several classes of service. Consistently the empirical analysis of the tests is divided into two areas. First, we concentrate on TCP performance analysis considering some TCP connections sharing a link of limited capacity. Second, we focus on the degradation introduced on TCP performance when TCP traffic and higher priority UDP traffic compete on the same bottleneck link.

Even if our intent was not to test RED performance over a wireless link, using a slow wired link our results can be extended to a more general environment. Improvements to RED algorithm, such as *Adaptive RED* [FKSS99] or *CHOKe* [PPP00], which try to solve some of

RED's imperfections or some other proposals, such as the use of RED in combination with ECN were not considered. We did not try to cover these proposals as we concentrated our study on RED behavior, which represents the basis for further studies and enhancements.

The rest of the thesis is organized as follows. Section 2 surveys related work on controlling congestion and on providing QoS in the Internet. In particular it presents the main properties of TCP protocol. Section 3 demonstrates a significant weakness in the current congestion control mechanism in Internet. In order to address this inefficiency an active queue management algorithm is proposed: Random Early Detection (RED). We describe the theoretical approach of RED and in addition we shortly introduce some of RED's enhancements and especially Explicit Congestion Notification (ECN) that can be used in combination with RED. Section 4 explains, in terms of workload models, network configuration and metrics utilized, which tests have been run and how the results have been evaluated. We also provide simple guidelines in setting RED parameters. The experimental tests belong to two distinct test cases. The first is intended to provide a RED study applied to TCP traffic. The second is concerned with an evaluation of RED in combination with services differentiation. It involves TCP and UDP traffic. Then, in Section 5, we illustrate and analyse the results of our experiments. The evaluation is mainly based on a comparison between Tail Drop and RED's performance. The theoretical properties of RED presented before are verified and criticized. Finally, in Section 6, we give our conclusions. Three topics are covered: conclusions about the present work, a summary of contributions of new knowledge that this thesis makes and ideas for future research that have been produced during the development of this thesis. In addition to the main text it follows a References Section that lists the publications that have been referenced in this Master's Thesis. The references are ordered alphabetically by the author's surname. Section 7 is a short resume in Italian of the whole thesis. Finally Section 8 reports some test results that have been mentioned during the test analysis.

2 Background

This Section reviews the important features of the transmission control protocol as well as the current state of providing quality of service in the Internet. In the last part we describe shortly the wireless networking environment focusing on its properties and the network transport protocol used in such an environment.

2.1 Transmission Control Protocol (TCP)

The Internet Protocol (IP) [Pos81a] provides an unreliable, connectionless datagram delivery service. IP does not guarantee to deliver correctly an IP datagram at its destination, but it just assures a best effort service. The service is based on a connectionless design, which means that the delivery of each datagram is treated independently of the other datagrams belonging to the same flow. This simple structure makes IP a very flexible and robust protocol, but the upper layers, such as TCP, have to provide a more reliable service and be able to recover from situations such as packet loss, packets out-of-order, damaged packets or duplicated packets [Pos81b].

The Transmission Control Protocol (TCP) guarantees reliable transportation of data and delivery of packets in order and without errors. Moreover it provides congestion control and a fair allocation of network resources. The most important aspects related to the TCP mechanism can be summarized in this way:

- Connection-oriented
- Data Transfer
- Error Control
- Flow control
- Multiplexing
- Congestion control

2.1.1 Protocol Operation

Before proceeding we shortly give an overview of the TCP segment format, the setting up of a connection, the actual data transfer and the closing of connection.

TCP Segment Structure

All TCP data units that are used to set up a connection, transfer data and tear-down a connection have a standard format, shown in Figure 2-1. All segments are transferred between two TCP entities in the user data field of IP datagrams. The TCP segment format is composed by the header and by the data field of variable length. The normal size of the header is 20 bytes, unless other options are present.

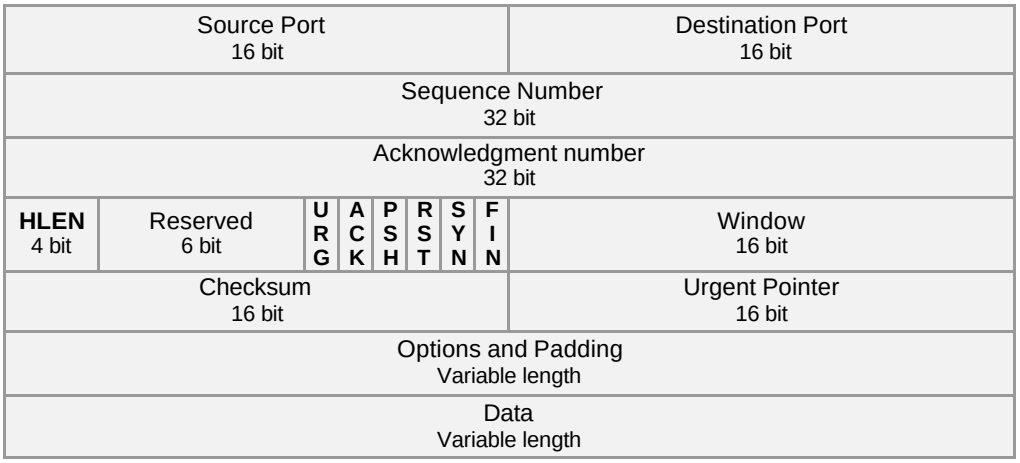


Figure 2-1. The TCP segment format.

The following fields characterize the TCP header:

- *Source Port* and *Destination Port*: the addresses of the end-points of the logical connection between two application protocols.
- *Sequence Number*: is used to differentiate segments and indicates the first byte in the data field of the segments relative to the start of the complete message. The number is linear so that the first segment sent has a number N and all subsequent segments have sequence numbers that relate to the number of bytes in the user data. For example, using a user data size of 1500 bytes and an initial sequence number 0, the first segment will have sequence number 0 and the next one 1500, the third 3000, etc.
- *Acknowledge Number*: the sequence number of the next byte this end of the connection is waiting for. All earlier bytes have been received successfully.

- HLEN: the total length of the TCP header. It indicates the number of 32-bit words it contains.
- *Reserved:* field reserved for future use.
- All TCP segments have the same header format and the validity of selected fields in the segment header is indicated by the setting of bits in the 6-bit *code-field*; if a bit is set (=1) the corresponding field is valid. Multiple bits can be set in a single segment. The 6 flags have the following meaning: URG is used if there are urgent data and in this case the *Urgent pointer* points to the urgent data; ACK contains a valid number if it is an ACK packet; PSH is set to 1 when the sender wants to use the PUSH command; RST is used to reset the connection without a precise teardown; SYN is the synchronize flag essential for the connection set-up and FIN is used in the connection tear-down. Their use is explained later in this section.
- *Window*: specifies the Receiver Advertised Window (*rwnd*), which means the number of data bytes that the receiver is prepared to accept beyond the sequence number indicated in the acknowledgment number field (The maximum value is 65.535 bytes).
- *Checksum*: CRC calculated adding to the TCP header the IP address of the source and of the destination.
- *Options and Padding*: used to fill the header and to specify some options such as the MSS value in SYN segment (the default is 536 bytes till a maximum of 65535 bytes), the window's scale that defines the measure unit for the window length specified in *Window* (the default is 1byte) and the timestamp value, explained later.

Connection Establishment

TCP is oriented to the transmission of a continuous stream of octets and converts the data flow in segments suitable for transmission through IP. The application passes the data to a TCP buffer and TCP builds segments out of them and transmits them. The segment size is limited by the Maximum Segment Size (MSS). TCP is a *connection-oriented* protocol: before end-nodes can exchange data, it is necessary to establish a connection between them. In order to avoid any ambiguity with the initial sequence number setting at both sides of a connection, each side informs the other of the initial sequence number it proposes to use.

The connection establishment is realized through the *three-way handshake* mechanism, as shown in Figure 2-2. In this phase the two hosts negotiate the MSS of the connection. The requesting end sends an initial sequence number to the receiver using a SYNchronized packet (flag SYN=1). The remote host first stores the sequence number setting for the incoming

direction, then it answers with an SYN/ACK message to acknowledge the sender's initial sequence number and to communicate its initial sequence number. Finally the initiating side responds with an ACK to the remote host's sequence number. The connection is now established.

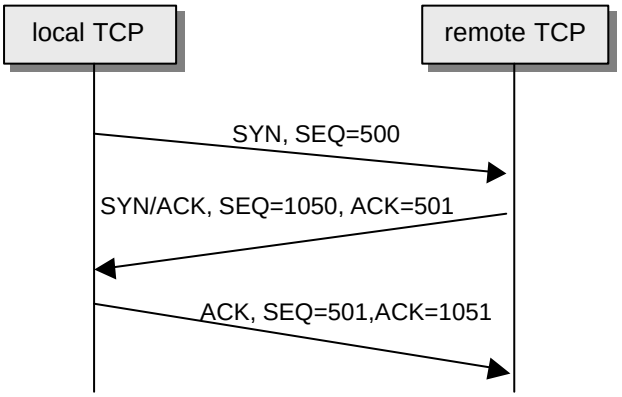


Figure 2-2. The connection setup.

Data Transfer and Error Recovery

The error and flow control functions are the main procedures associated with the data transfer. They are employed with the intent to ensure that all the segments are successfully sent and acknowledged by the receiver and finally guarantee the reliable delivery of data

TCP implements *error control* functions based on the go-back-n mechanism of retransmission. If the data have not been received within the timeout the segment has to be retransmitted. TCP assignes an exclusive sequence number for each octet transmitted and in the TCP header of each segment there is the number of the first octet of that segment. The receiver sends an acknowledgment (ACK) upon reception of a segment. The acknowledgments are cumulative: an ACK confirms all the bytes up to the given sequence number. In the ACK the sequence number of the next expected octet is carried. The sequence number is used to reconstruct the order of the segments received. Errors can be discovered thanks to the checksum.

TCP has a significant property of self-clocking, in the equilibrium state, each arriving ACK triggers a transmission of a new segment. Generally, TCP does not acknowledge a received segment immediately, but waits for a certain time in order to reduce the traffic on the link. In fact, if a data segment is sent during this time, the acknowledgment is piggy-backed into it [APS99]. If there are no packets out-of-order, no errors, no duplicate packets, all the data

are acknowledged, buffered at the receiver and then delivered to the upper layer. Otherwise, if the TCP host receives duplicate packets or out-of-order segments it does not acknowledge new data, but immediately sends an ACK segment that acknowledges the highest sequence number correctly received so far. This means that in these cases the sender may receive *duplicate acknowledgments* (DUPACK), which acknowledges the same segment as the previous ACK. When the first DUPACK is received TCP waits to retransmit the packets not yet acknowledged and when the three DUPACKs are received the Fast Retransmit algorithm is triggered. We will explain it later. In combination with the retransmission timer (RTO), on the sender side, ACKs provide reliable data delivery. A lost packet is generally indicated by the expiration of the RTO or the receipt of a duplicate acknowledgment.

To prevent a fast sender from overflowing a slow receiver, TCP uses the *Flow Control*, based on the principle of the *sliding window* mechanism. The receiver specifies the dimension of the *Receiver Advertised Window* (*rwnd*) in each data it sent to the opposite end, as an indication of the amount of data it is able to receive. An arriving ACK allows more data to be transmitted by advancing the sliding window to the right. When the total size of outstanding segments, *segments in flight* (FlightSize), fills up the receiver advertised window, the transmission of data is stopped until the sliding window advances or a larger receiver window is advertised. Specifying a *rwnd* of zero bytes is possible and can be used to force the sender into the persist mode. In this state the connection is still alive, but the transmission of new data is blocked.

Finally, to achieve *Multiplexing* communication, TCP allows the use of multiple ports within a single host. A socket is defined as the couple port or address host and network address. A pair of sockets identifies unambiguously every connection, thus a single socket can be active simultaneously in several connections.

Connection Termination

When the communication is complete the connection is closed and the resources are released. The connection tear-down is shown in Figure 2-3. When one side of the system wants to terminate the connection, it sends a FIN segment (flag FIN=1) to the other side. As a reply the node on the other end returns an ACK segment to acknowledge the receipt of the FIN segment. The connection is still open for the other direction and the host on the other side can go on sending data. When it has finished it sends a FIN message with the last data and waits for the acknowledgment. When it receives the last acknowledgment, it finally closes the

connection. If segments disappear in the middle of the connection termination, they are resent with the usual retransmissions.

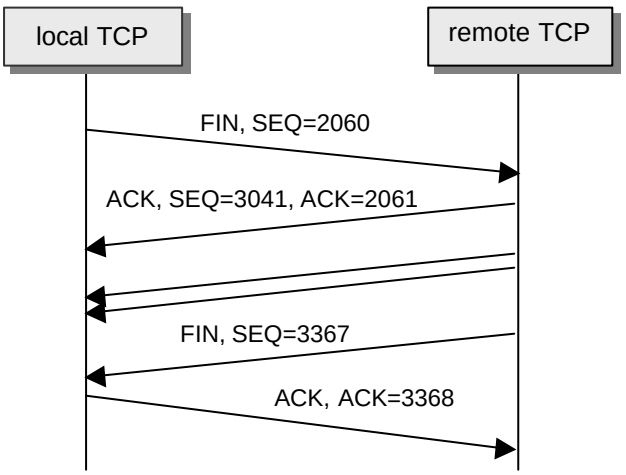


Figure 2-3. The connection tear-down.

2.1.2 Congestion Control

When one or several TCP connections are sending at inappropriately high rates the network can suffer from congestion. The router buffers saturate and some packets or their corresponding ACKs may be dropped before reaching their destination. This occurs when routers are receiving more packets than they can handle. Congestion collapse is a state in which packets are being injected into the network, but very little useful work is being accomplished. The specification of the *Receiver Advertised Window* (*rwnd)* is a way to control the rate of data incoming at the receiver, but is not enough to prevent network congestion.

Early in its evolution, TCP was enhanced by the congestion control mechanism to protect the network against the incoming traffic that exceeds its capacity. The first job of a congestion avoidance mechanism at the gateway is to detect incipient congestion and as stated in [JR88] a congestion avoidance scheme maintains the network in a region of low delay and high throughput. The congestion control is implemented in the hosts and consequently is an end-to-end congestion control. When a network is congested, more connections compete for limited resources. The congestion control mechanism tries to regulate the transmission rate of each connection in order to provide a fair sharing of network resources.

TCP congestion control is window-based. The sender rate is regulated by a *Congestion Window* (*cwnd*) that limits the amount of data a sender can have outstanding in the network. It must not send data with a sequence number higher than the sum of the last acknowledged

packet and the minimum of *cwnd* and *rwnd*. The cwnd changes its value in relation to the events the sender observes. The basic principle on which the TCP congestion control is built is to consider the packet loss as a signal of congestion. Thus the reaction to this event is to reduce the cwnd.

Two algorithms, known as Slow Start and Congestion Avoidance, regulate the reduction and the increasing of cwnd [APS99], basing their evaluation on a further variable: the slow start threshold (*ssthresh)*. Depending on its value the sender is subject to a Slow Start (*cwnd*<*ssthresh*) or Congestion Avoidance (*cwnd*>*ssthresh*) mechanism.

Slow Start

The Slow Start algorithm is based on the observation that the rate at which packets are injected into the network should be regulated by the rate at which acknowledgements arrive from the receiver should be the same. It is used at the beginning of the connection to avoid congesting the network or after repairing a packet loss detected by time-out expiration. The initial value of *cwnd* must be no more than 2 segments and the *ssthresh* may be arbitrarily high. Being *cwnd*<*ssthresh* the Slow Start algorithm is used. During this phase *cwnd* is increased by at most *sender maximum segment size (SMSS)* bytes for each ACK received that acknowledges a new packet. The sender can transmit up to the minimum of the *cwnd* and *rwnd* as said before. The growth of *cwnd* is exponential: *cwnd* roughly doubles per each RTT. Slow Start ends when *cwnd* ³ *ssthresh* or when congestion occurs. Congestion is declared when a timeout expires or three consecutive DUPACKs are received. They are both indications of a packet loss.

If a timeout expires TCP reacts decreasing *ssthresh* to be half of the number of segments outstanding into the network (*flightsize*):

$$ssthresh = \max\left(2, \min\left(\frac{flightsize}{2}, rwnd\right)\right)$$

The retransmission timer is re-calculated through the exponential backoff (the new RTO is equal to the old one multiplied by a constant value, greater than 2) and *cwnd*=1. Now *cwnd*<*ssthresh* so a new phase of Slow Start is entered. But now *ssthresh* is lower than before and this implies that the capacity of the network will not saturate quickly as previously. The sender starts to retransmit packets beginning from which has caused the timeout expire.

Congestion Avoidance

The second situation to analyse is when Slow Start ends because *cwnd=ssthresh* (or *cwnd>ssthresh)*. In this case Congestion Avoidance starts. During this phase *cwnd* is incremented by 1 SMSS only after a full window of data is acknowledged:

cwnd = *cwnd* + SMSS * SMSS/*cwnd*

for every incoming new ACK. This means that if *cwnd* permits to transmit N packets only after the reception of N ACK relative to all the packets sent, *cwnd* will increase by 1 packet (SMSS bytes). The growth of *cwnd* is linear. Figure 2-4 illustrates an example of *cwnd* variation during the phases of Slow Start and Congestion Avoidance.

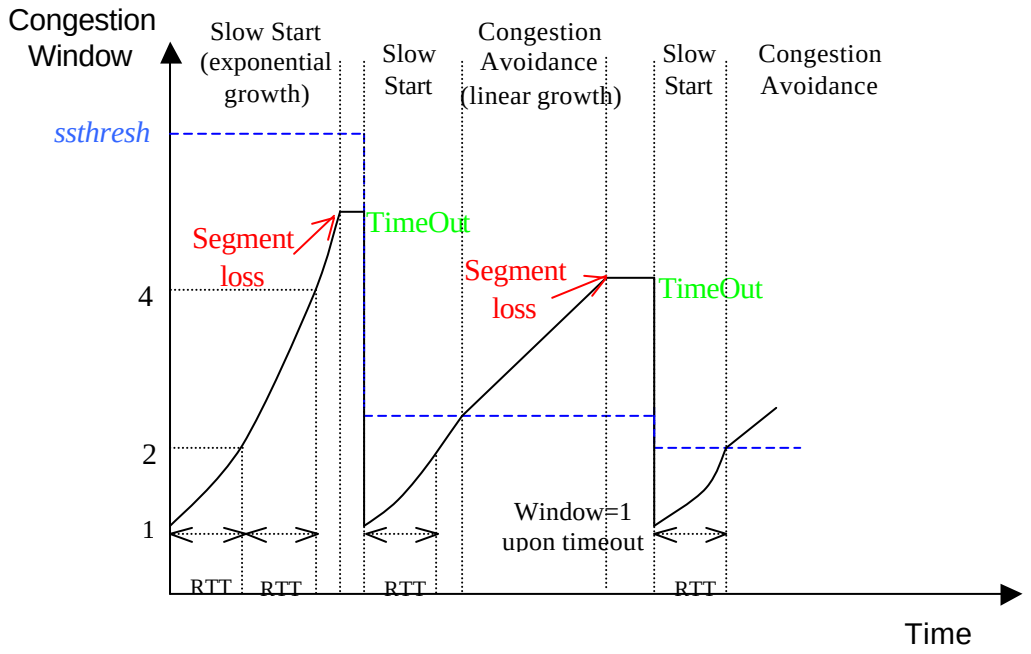


Figure 2-4. TCP Congestion Window.

Fast Retransmit/Fast Recovery

The last case to analyse is what happens when three DUPACKs are received and congestion is declared. In this description we refer to the so-called Reno algorithm described in RFC 2581 [APS99]. It also introduces the use of *TCP Selective Acknowledgment* [MMFR96].

A TCP receiver should send DUPACK as soon as possible when an out-of-order packet arrives. The meanings of a DUPACK for a sender can be various: a packet drop, a reordering of data segments and duplication of ACKs or data segments. So when the first DUPACK is

received TCP waits to retransmit the supposed packet drop because the cause may be different from that of a packet discarded. The fast retransmit algorithm uses three DUPACKs as indication of packet loss. The Fast Retransmit and Fast Recovery behave according to this model.

1. When the third DUPACK arrives *ssthresh* changes:

$$ssthresh = \max\left(\frac{flightsize}{2}, 2 * SMSS\right)$$

2. The lost packet is retransmitted.

3. *cwnd=ssthresh + 3*SMSS,* this inflates the congestion window by the number of packets (three) that have left the network and are in the receiver buffer.

4. Increase *cwnd* by one SMSS for each additional DUPACK received.

5. Transmit a packet if allowed by the new value of *cwnd* and *rwnd.*

6. When an ACK that acknowledges new packets arrives, set *cwnd* to the value *ssthresh* set in step *1.* The Fast Retransmit phase ends.

The assumption at the basis of these two algorithms is that if DUPACKs arrive it means that a packet has been lost. But if DUPACKs arrive it also means that after the packet loss data segments have arrived at the receiver buffer and this allows increasing the *cwnd* towards its previous value on arrival of each DUPACK. The problem is, what happens when there are multiple packets dropped.

If the SACK (*Selective Acknowledgment*) option [MMFR96] is not available the sender has little information about which packets to retransmit during Fast Retransmit. When it receives the acknowledgment for the retransmitted packet in the case of multiple drops it receives a partial acknowledgment and this will acknowledge some but not all packets sent before the Fast Retransmit. Therefore, the TCP sender has no other choice than wait for retransmission timeout to expire.

The Fast Retransmit and Fast Recover in NewReno algorithm

The problem of the Fast Retransmit algorithm is that it retransmits only the first segment without waiting for the timeout to expire. In RFC 2582 [FH99] is presented a solution to

respond to partial acknowledgments (in the absence of the SACK option) and the algorithm that implements this is referred to as the NewReno algorithm.

The main modifications lie in the introduction of a new variable *recover* in step 1 of the Reno algorithm and in the variation of step 6.

1. When the third DUPACK arrives and the sender *ssthresh* changes:

$$ssthresh = \max\left(\frac{flightsize}{2}, 2 * SMSS\right)$$

Record in the variable *recover* the highest sequence number sent.

2. The lost packet is retransmitted.

3. *cwnd=ssthresh + 3*SMSS,* this inflates the congestion window by the number of packets (three) that have left the network and are in the receiver buffer

4. Increase *cwnd* by one SMSS for each additional DUPACK received.

5. Transmit a packet if allowed by the new value of *cwnd* and *rwnd.*

6. When an ACK that acknowledges new packets arrives:

   a. if this ACK acknowledges all the data up to and including *recover:* set *cwnd* either to *ssthresh* in *1.* or to min(*ssthresh, flightsize + SMSS).* Fast Recovery ends.

   b. Otherwise this is a partial ACK. Then retransmit the first unacknowledged packet; deflate the congestion window with as much as the new data acknowledged; *cwnd = ssthresh + SMSS* (partial window deflating)*;* send a new segment if allowed; Fast Recovery goes on. Moreover if this is the first partial ACK received reset the RTO.

If the SACK (*Selective Acknowledgment*) option is available the receiver is able to inform the sender about which packets are missing and the TCP sender can identify precisely which segments have been lost and have to be retransmitted.

2.1.3 TCP Retransmission calculation

As regards the *error control* one of the major problems is the calculation of the Retransmission Timeout (RTO). If it is too small the sender will fill the link of segments and will retransmit segments unnecessarily degrading TCP performance. Otherwise if it is too long, in case of a packet loss, the sender will take a long time to detect the loss and to start the retransmission of segments. The optimal value depends on the Round Trip Time (RTT). TCP updates constantly the value of RTO estimating the RTT and using Karn and Jacobson's algorithms.

In the original TCP specification a smoothed RTT estimator was calculated using the following formula:

$$SRTT^{(i)} = (1-\alpha)\, SRTT^{(i-1)} + \alpha\, RTT^{(i)}$$

where α is a smoothing factor with a recommended value of 0,9. This smoothed RTT (SRTT) is updated every time a new measurement is made. Given the smoothed estimator, which changes as the RTT changes, the RFC 793 [Pos81b] recommended the retransmission timeout value to be set to

$$RTO = \beta\, SRTT$$

where β is a delay variance factor with a recommended value of 2.

[Jac88] details the problems with this approach since it cannot keep up with wide fluctuations in the RTT, causing unnecessary retransmissions. These add to the network load, when the network is already loaded. In fact although the parameter β accounts for the RTT variation, the suggested value can adapt to network loads up to 30%. Above this point, a connection will respond to load increases by retransmitting the packets that have only been delayed in transit.

In addition to the smoothed RTT estimator, we need to keep track of the variance in the RTT measurements, since calculating the RTO based on both the mean and variance provides a much better response to wide fluctuations in the round-trip times. The mean deviation is a good approximation of the standard deviation, but easier to compute [Jac88]. This leads to the following equations that are applied to each RTT measurement. The Smoothed Round Trip Time SRTT is calculated with a low-pass filter (in which α is a constant between 0 and 1, generally 1/8) and $RTT^{(i)}$ samples are defined considering the time elapsed between the transmission and the reception of the corresponding ACK:

$$SRTT^{(i)} = (1-\alpha)\, SRTT^{(i-1)} + \alpha\, RTT^{(i)}$$

The estimated mean deviation DEV and the corresponding smoothed value SDEV are:

$$DEV = |RTT^{(i)} - SRTT^{(i-1)}|$$

$$SDEV^{(i)} = \tfrac{3}{4}\, SDEV^{(i-1)} + \tfrac{1}{4}\, DEV$$

Finally the new value for the Time Out is:

$$RTO = SRTT + 4\, SDEV$$

Note that after a retransmission the value of RTT remains unchanged. Say a packet is retransmitted, a time-out occurs, the RTO is backed off and the packet is retransmitted with longer RTO, and an acknowledgment is received. Is the ACK for the first transmission or the second? This is called the *retransmission ambiguity problem.* Karn and Patridge [KP87] specify that when a timeout and retransmission occur, we cannot update the RTT estimators when the acknowledgment for the retransmitted packet arrives. This is because we do not know at which transmission the ACK corresponds. A new RTO does not have to be calculated until an acknowledgement for a segment that was not retransmitted arrives.

2.2 Quality of Service in the Internet

One definition for QoS is that it means *providing consistent, predictable data delivery service* [Hus00]. Its aim is to satisfy customer application requirements with a certain level of assurance. It includes a diverse set of service requirements such as performance, availability, reliability, security, etc. and all of these requirements are important aspects of a comprehensive network QoS.

The service realized by TCP over today's Internet is generally known as "best-effort". Using a traditional FIFO queueing discipline in the network in combination with TCP congestion control at the endpoints, sources maintain approximate fairness between themselves when they are sharing the same bottleneck link. As the need of new service has grown, the lack of service differentiation has become problematic.

2.2.1 QoS Technologies

The evolution of Internet during the last decade has been accompanied by the development of new applications, with specific requirements. The aim of service differentiation is to satisfy heterogeneous application demands providing them with the request level of service in terms of throughput, delay, jitter, bandwidth and priority. The need for services differentiation has led IETF to develop a number of architectural solutions to satisfy heterogeneous application requests and provide QoS. There are two main approaches to delivering QoS services. The first is to create for each service request a reservation state and keep it for all the connection. This is the basic idea on which the *Integrated Service Architecture (IntServ)* [BCS94] has been built. The second is to perform the classification of the traffic at the entering of the network and then to develop a policy administration function to create service outcomes. This is the principle at the basis of *Differentiated Service (DiffServ)* [BBCD98].

The IntServ architecture is the Internet incarnation of the traditional circuit-based architecture and its main advantage lies in the high accuracy of resource management even though it is poor in scalability. The major contribution of IntServ has been the implementation of the *Resource Reservation Setup Protocol* (RSVP) that allows the applications to specify at the beginning of the connection their resource requirements. On the basis of this end-to-end resource requirements definition, the intermediate network elements (routers, switches etc) can allocate the necessary amount of resources for such an application. For resources we intend link bandwidth on transmission links, router buffer capacity to hold packets in transit and CPU capacity to forward packets in real time. When subsequent packets arrive at each network element, they are scheduled in a manner that satisfies the application requirements. On one hand this service architecture is able to ensure a solid foundation for providing different classes of service in the Internet, but on the other hand it requires considerable changes to the network structure. In addition, support for such service can add a significant amount of overhead in packet processing within the network.

Given these drawbacks inherent in the IntServ architecture, the IETF has considered a more evolutionary approach to ensure service differentiation in the Internet, the DiffServ model. This is expected to have better scaling properties and to be easier to implement than IntServ. In fact the DiffServ architecture does not introduce so much overhead and does not employ per-flow state. To guarantee quality of service to the applications it uses the DS field [NBBB98] in the IP header.

The three main goals of DiffServ are:

- Keep the forwarding path simple: the packet processing at the interior of the network must be as simple as possible.
- Push complexity to the edges: any per flow activity/state should be kept strictly at the edge of the network such as in the first-hop router.
- Avoid assumptions about traffic type in order to satisfy all the new Internet applications and their requirements.

In spite of its desirable simplicity DiffServ, like IntServ, also needs improvements to guarantee predictable services and an accurate resource allocation. An interesting proposal has been to combine the two architectures, applying IntServ functionality at the edge of the network and DiffServ within the network, but it still needs further studies.

2.2.2 Differentiated Services Architecture

The approach adopted by the Differentiated Service Architecture is to address the complexity of the problem decomposing it in single tasks each independent of the other. In DiffServ the *per-hop service behaviour* is achieved through the development of the *admission control* and the *policy administration* functions. Although there are several architectures for the implementation of DiffServ, all of them are made up of two basic elements: a mechanism that monitors the entry of packets into the network and one for the processing of these packets at the interior nodes of the network.

Routers are divided into two categories: the *Edge routers*, which are responsible for the classification and the conditioning of packets and the *Core routers*, which maintain no state information and act based on the initial classification (Figure 2-5). This structure makes the network scalable since the boundary nodes that maintain full state information have only a limited number of flows going through them. During the classification the traffic stream is assigned to a particular *per-hop behaviour* (PHB), identified by a specific codepoint in the DS field of the IP header and present in all the packets belonging to such traffic. The nodes inside the network simply select the forwarding behaviour for packets through mapping the DS-codepoint (DSCP) to one of the PHBs implemented by the *Differentiated Service Domain* (DS-domain), which is a set of DS-nodes with a corresponding set of PHBs supported. No further profiling or classification is operated inside the network: the load related to the classification is let to the edge of the network [BBCD98, Hus00].

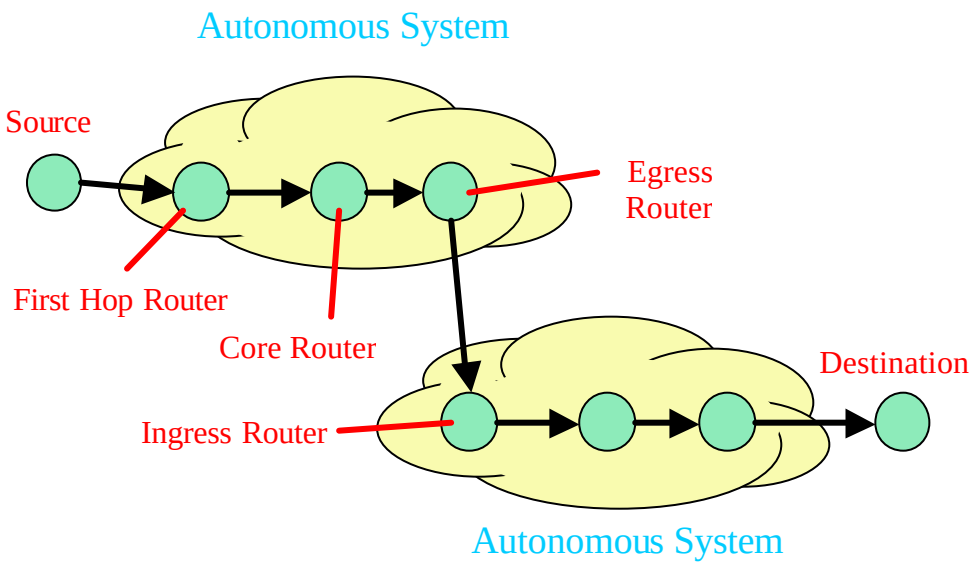


Figure 2-5. The DiffServ Architecture.

In order to understand the DiffServ architecture we can identify its basic components and analyse them individually:

- The DS field
- Classification and Conditioning of traffic entering the network
- Policing
- Per Hop Behaviour (PHB)
- Queueing and Scheduling Algorithms
- Buffer Management and Congestion Control
- Routing

The DS Field

The DiffServ architecture is based on the use of the DS field in the IP header (Figure 2-6). The DS field corresponds to the 8-bit Type of Service (TOS) of the IPv4 header or to the Traffic Class field of the IPv6 header. These 6 bits are called *DiffServ Code Point* (DSCP). Certain combinations are standardised and called *Per Hop behaviours* (PHBs). There are two main tasks related to the DS field. The former is the packet-forwarding function in which the DS field is used to select a particular per-hop behaviour inside the network and that it is simply realized using a routing table. The latter regards the routing function or more specifically the policy, the configuration network and the resource allocation, on which is based the correct working of the first function [NBBB98].

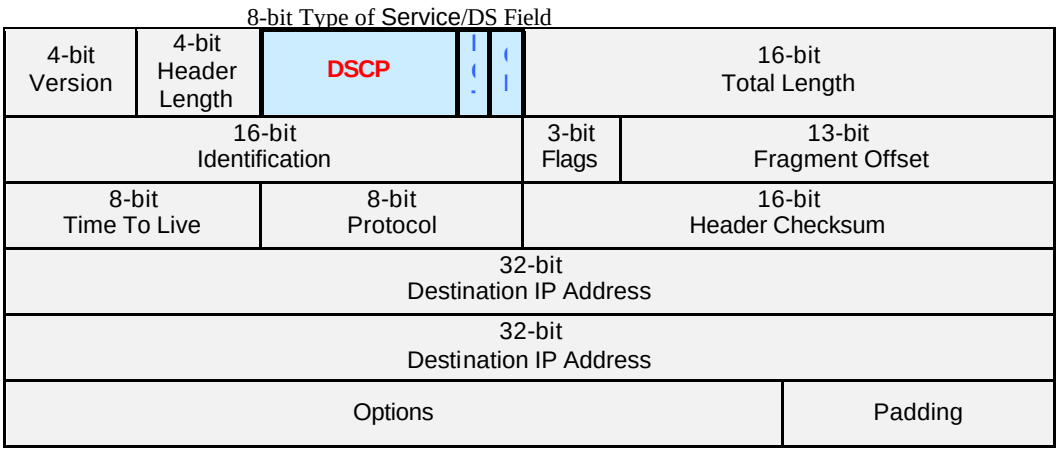


Figure 2-6. The DS field in the IP header.

Classification and Conditioning of traffic entering the network

A *Differentiated Services Domain* (DS-domain) is a contiguous area of the Internet over which a consistent set of differentiated service policies are administrated. A DS domain can correspond to different administrative domains or autonomous systems or different network topologies. The *Differentiated Service Boundary* is the edge of a DS domain, where classifiers and traffic Conditioners are implemented. The DS-boundary nodes can be further sub-divided into *Ingress* and *Egress* nodes, as shown in Figure 2-5. They represent the downstream/upstream nodes of a boundary link in a given traffic direction. A Differentiated Service Boundary is usually deployed at the ingress to the first-hop differentiated-services-compliant router that a packet will traverse during the connection, or at the egress of the last-hop differentiated services-compliant router that a packet will traverse before reaching the host. A router is defined *Differentiated Service Compliant* if it is in compliance with what was specified in RFC 2474 [NBBB98].

The DS-Boundary nodes must check the entering traffic conform to the *Traffic Conditioning Agree*ment (TCA) between their DS domain and the domain that they connect to. The TCA between two domains is derived from the *Service Level Agreement* (SLA), a service contract that defines the service a customer should receive. The SLA specifies the amount of traffic allowed for certain classes and it may include traffic conditioning and packet classification rules. Packets that do not conform to the SLA can be dropped or modified in the DS field before being transmitted.

Traffic classifiers select packets based on the header fields of the IP packets taking into account several elements:

- DiffServ field
- Protocol
- Port Number
- Source address
- Destination address
- Mac address
- Time-To-Live (TTL) field

If they consider only the DSCP value they are called *behaviour aggregate classifier* and if they also check source address, destination address and other elements they are called *multifield classifier*. *Traffic Profiles* determine the temporal properties of a classified packet stream, specifying if packets are in profile or out of profile. A *traffic conditioner* may be constituted by several elements, such as markers, shapers, meters and droppers, but it does not necessarily contain all four elements as shown in the example in Figure 2-7 . Packet *markers* set the DS field of a packet to a particular DSCP, indicating the service quality and corresponding to a particular PHB behaviour. The *Shaper* and *dropper*'s task is to mould the packet stream in order to make it compliant with the traffic profile. Finally traffic *meters* measure the temporal properties of the stream and pass the information to the marker.

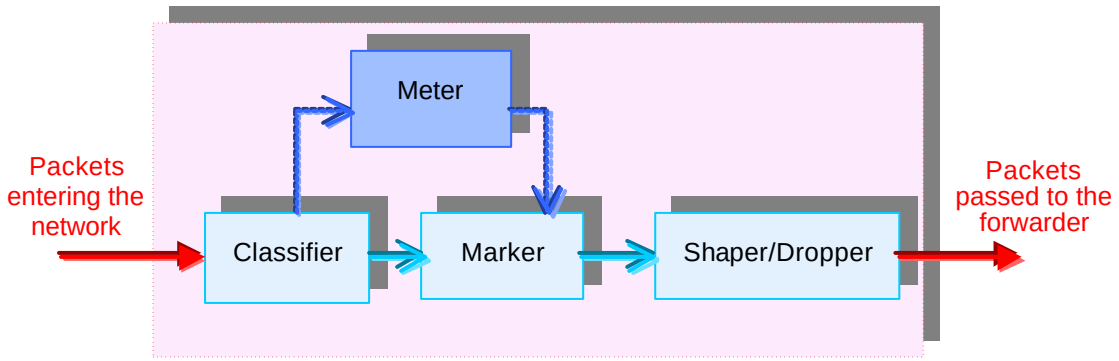


Figure 2-7. Block diagram of a traffic classifier and a traffic conditioner.

Policing

This refers to a combination of rules intended to prevent a traffic stream to grab more resources than allowed. It is based on an agreement between the Service provider and his customer. Packets that do not conform to the SLA can be dropped or transmitted with a modified value of the DS field

Per-Hop behaviour (PBH)

The DiffServ working group has specified a set of Per-Hop Behaviours that are used by the router to provide differentiated services. PHBs are encoded in the DSCP and are used to determine the treatment the packet forwarded will receive by the intermediate nodes. On the specification of the PHBs is based the resource allocation at each node in the network. PHBs may be defined in terms of their resource or of the traffic characteristics. They can be considered to be building blocks by which it is possible to construct a great set of services and they constitute the basis for future extensibility of the structure.

Currently three PHBs have been standardized: the Default (DE), the Explicit Forwarding (EF) and the Assured Forwarding (AF) PHBs. The DE PHB [NBBB98] is the common, best effort forwarding available in today's Internet. IP packets marked for this service are sent into a network without adhering to any particular rules and the network will deliver as many of these packets as possible and as soon as possible, but without any guarantees. The EF PHB [JNP99] specifies a forwarding behaviour in which packets see a very small amount of loss and a very low queueing delay. In order to guarantee that every packet marked with EF receives this service, EF requires every router to allocate enough forwarding resources so that the rate of incoming EF packets is always less than or equal to the rate at which the router can serve them. To achieve this on an end-to-end scale, EF requires that traffic be shaped and reshaped in the network. The AF PHB group [HBWW99] specifies a forwarding behaviour in which packets see a very small amount of loss. It consists of four different classes. Within each of them, IP packets are marked with one of the three possible drop preference levels to differentiate between flows in the class itself. In case of congestion the drop precedence of a packet determines the relative importance of the packet within the AF class and a congested node tries to protect packets with a lower drop precedence value from being lost. The AF's intent is to preferentially discard best-effort packets and packets which are outside of their contract when congestion occurs. By limiting the amount of AF traffic in the network and by managing the best-effort traffic appropriately, routers can then ensure low loss behaviour to packets marked with AF PHB.

Queueing and Scheduling Algorithms

A router can allocate its resources by employing several queueing algorithms and several scheduling techniques. The first function specifies how to behave in case of congestion, which packet to drop or mark .The second function is related to the decision on which link to send a packet.

Briefly we can just underline that in order to implement a DiffServ network it is necessary to map every different DiffServ traffic class on a specific couple of scheduling and queueing mechanisms. It is not still clear which is the best scheduling mechanism and the alternatives are different, such as:

- MDRR (Modified Deficit Round Robin): it extends DRR [SV94] and offers the support for delay sensitive traffic, such as VoIP, which is associated with the LLHP (low-latency, high-priority) queue. This special class is treated differently from the others in order to guarantee the associated service.
- WRR (Weighted Round Robin): it is priority based and provides service for voice traffic.
- WFQ (Weighted Fair Queueing): it is a packet scheduling technique allowing guaranteed bandwidth services. The purpose of WFQ is to let several sessions share the same link. WFQ is an approximation of Generalized Processor Sharing (GPS) [SL97]. In GPS each service has a separate FIFO queue. At any given time the N active sessions (the ones with non-empty queues) are serviced simultaneously, each at a rate of 1/Nth of the link bandwidth. GPS also allows having different service shares for each session. This means that each session has its own queue and an ill-behaved session (for example one that is sending a lot of data) will only punish itself and not other sessions. Informally GPS implements a *min-max weighted fair-share algorithm* using a small service ration. The requests are sorted in order of increasing weighted resource requirements and each request is satisfied in order, with a weighted proportion of the residual resource; the unused allocated resource is placed back in the resource pool to be shared over the remaining requests. WFQ follows this ideal model of GPS, but at the packet level, considering the distortion of the resource-sharing caused by data being quantized into packets
- IP RTP Priority (Internet Protocol Real-Time Transport Protocol Priority) [SCFJ96]: it permits to define a range of UDP/RTP ports whose voice traffic is guaranteed high priority service over all the other classes sharing the same interface.
- Priority Queueing within CBWFQ [GVP01]: it uses priority queueing for voice traffic belonging to a class.

Buffer Management and Congestion Control

In order to provide different QoS to different flows the routers need both scheduling algorithms, such as discussed above, and buffer managements schemes. Some of them are:

- Tail-Drop (TD): it drops arriving packets only when the queue saturates.
- Random Early Detection (RED) [FJ93]: it is an active queue management scheme, based on the average queue size; it drops or marks the packets following a probability algorithm.
- Longest Queue Drop (LQD) [LNO96]: in presence of congestion it drops a packet from the flow that has the longest buffer queue
- Weighted Random Early Detection (WRED)[CISCO98]
- Random Early Detection In/Out (RIO) [CF98]

Some of these algorithms are described thoroughly in section 3.2.3. We can just notice briefly that the using of active queue management mechanisms makes it possible to differentiate the performance of several TCP connections assigning them different priorities and developing a discarding strategy based on them. It is possible to realize priorization between distinct traffic aggregates for instance within the Assured Forwarding PHB group. For details see [WW02].

2.3 Wireless Networking Environment

The Internet has been in constant growth in the last two decades, reaching a greater number of users and supporting new applications. New kinds of hosts, such as mobile devices, have become more and more important and the Internet more heterogeneous. At the beginning TCP was specified for wired networks and stationary hosts, but nowadays, wireless networks represent more and more the Internet's reality and they will play an important role in its future. The wireless environment exhibits different characteristics compared to the wired one and TCP improvements are needed to assure reliable transport services to all the users regardless of the kind of links connectivity used. The nature of wireless links is quite different compared to wireline networks for several reasons.

We can subdivide Wireless networks into two main categories: Local area networks (LAN) and Wide area networks (WAN) on the basis of the dimension of the area they cover [MDKM00, Pen00]. The Wireless WANs are more problematic since they are characterized by a delay-bandwidth four or five times greater than the Wireless LANs.

Wireless LANs (W-LAN) in general are organized in a cellular topology in which the territory is subdivided into several cells, each characterized by a defined coverage area. A base station with a W-LAN transceiver is responsible for a single cell and for all the hosts moving in

that area. The base station, also called access point, is directly connected to the wired network. A handoff occurs when a mobile host moves from one cell to another and switches from the old access point to the current one.

W-LANs are characterized by high bit rates, but by a small service area of the base station. They are developed on the basis of two main standards: the High Performance Radio Local Area Network (HIPERLAN) standard and the IEEE 802.11 standard. The former offers channel speeds of 1 Mbps in a range of 800 meters or up to 20 Mbps in a range of 50 meters. The latter offers bit rates of 1 or 2 Mb/s in a range of 100 meters. They can operate just over limited distances, tens of meters or less, much smaller than those allowed in wireline. The maximum mobile host speed is about 36 Km/h for HIPERLAN/1 (HIPERLAN Type 1) and 90 Km/h for the IEEE 802.11 standard. For a more detailed analysis see [Tab00].

As regards WAN systems we can consider WAWDN (wide area wireless data networks), which are WANs specified for data transmission and include the technologies GPRS (General Packet Radio Service) and CDPD (Cellular Digital Packet Data) [Tab00]. They are characterized by lower bit rates, but a wider service area for users, about some tens of square kilometres. And then cellular networks that have been designed to provide mobile telephone voice service in the first place, but can carry data as well, even if in this case they are not very economical. Examples of these are GSM, D-AMPS and IS-95. They are characterized by lower bit rates like 9.6 kb/s for GSM [Tisal98] or 19.2 Kb/s for IS-95 [Tab00]. GPRS packet transmission offers higher transmission rates and simpler access to packet data networks than that provided by circuit switched services. In conventional GSM, a channel is permanently allocated for each user during the entire call period, while in GPRS, channels are allocated when data packets are sent or received in order to provide a more efficient use of resources for bursty data application and more flexibility. GPRS technologies represent an important step in the evolution towards third-generation cellular networks that promise to offer higher bit rates [BVE99].

At last there are the so-called long fat networks (LFNs or elephants) including satellite links and characterized by a long round trip time. W-WAN and LFNs are both L-networks since they have long round trip times, but only satellite networks are defined fat for the high bandwidth they consume. We refer to W-WAN as LTNs, long thin networks. In this brief description we do not analyse LFNs [MDKM00].

2.3.1 Properties of Wireless Links

In order to understand future improvements of the TCP protocol it is useful to summarize the most important properties of Wireless link [BKGM01, Pen00].

- *Limited Bandwidth*: wireless wide area networks present limited data rates, around some tens of kilobytes per second. Moreover, sometimes bandwidth can be asymmetric, with more capacity on the uplink than on the down one (for example the uplink rate is limited by battery use) and also it can oscillate over time, because the scheduler can decide to reallocate the resources sharing by multiple users, in order to maximize resource utilization.

- *High Latency:* in general wireless links are characterized by higher propagation delays than wired links. A typical Round Trip Time varies between a few milliseconds to one second. This affects TCP throughput and decreases the user's possibility to work interactively. We can also notice that these properties may comport in some situations unfair sharing of resources; for example in case two hosts are connected to the same Web Server, but through different links, wired in one case and wireless in the other one. The second host will experience a smaller bandwidth share cause of the longer RTT than the first one.

- *Variable delays:* the latency experienced on the link may increase unexpectedly for several reasons, such as a temporal loss of radio coverage or handover of the mobile user. This is usually referred to as delay spikes.

- *Error losses*: some wireless links are characterized by higher error rates. The BER on wired links are on the order of 10^{-6}-10^{-8}, while on the wireless one they are on the order of 10^{-3} and sometimes 10^{-1}. The error rate is determined by the current radio conditions and they can vary considerably during the connection. This is the most critical aspect when we consider wireless links and the problem is not so much related to the high number of retransmissions (since the packet was lost and the only way to recover it is to retransmit), as to the fact that lost segments cause TCP to enter a congestion avoidance phase and to keep the congestion window small. This implies a lower throughput. Forward error correction (FEC) can be used to correct the bit errors on wireless links by using an encoding algorithm, but this involves a large use of bandwidth and is efficient only in some cases.

- *User Mobility*: this aspect refers to two different cases depending on whether the service is with or without interruptions. The first situation is related to portability and is the case of a laptop that can connect to different access points and the other one is related to mobility and is the case of a user who can move and requires a continuous service. Hand-offs occur when a mobile host starts communicating with a new base station by moving from one cell to another one. The consequence can be a temporary loss of route, resulting in loss of packets.

2.3.2 Transport over Wireless Networks

Several solutions have been proposed to improve TCP's performance over wireless links or high loss links. It is important before analysing these techniques to observe how the congestion control schemes in today's Internet work, referring to [Lud99]. In general, congestion control can be realized in two different ways: through explicit or implicit congestion signals. At present, TCP's congestion control is based just on implicit congestion signals in the sense that it can infer from some particular network's signal the current state of the network. From this point of view two different techniques can be implemented: the former is based on delay measurements and the latter on packet loss rate. TCP adopts the second way since it may prove difficult to calculate a precise estimation of network delay. As a result, TCP's congestion control is entirely based on packet loss. At this point it is clear which problem these solutions have to face, eventually to turn to techniques never adopted such as those based on explicit congestion signals. There are three different kinds of approach: solutions at the link layer (LL), modifications of the TCP protocol and a proposal for new transport protocols [BSK95, DMKM01, Pen00, Vai99].

Link-layer protocols

The main idea of this kind of approach is to hide the lack of reliability of wireless links to the higher layers and attempt to offer them a channel with the same characteristics (or almost the same) of wired links, so that the existing sender protocols do not need any modifications. The transport layers remain transparent to these hardware techniques and most packet losses seen by TCP are effectively due to congestion. Using IP at the network layer requires a certain level of robustness of the link that is not guaranteed with wireless links. IP uses checksums to protect the packets it transports. They are not enough with wireless networks [MDKM00].

The two main techniques adopted by the LL protocol to improve wireless links are error correction, and most specifically FEC (forward error correction), and retransmission of packets loss, with schemes such as ARQ (automatic repeat request). But improving the reliability of wireless links is not without cost and the application of this kind of protocols can waste resources and cause transport protocols to perform badly. The two main causes for this negative result are related to timer interactions and fast retransmission interactions. The former regards the fact that setting the timer independently at transport and link layers may induce to a duplicate retransmission of packet loss, if LL timers are not set to expire faster than the TCP ones. The latter is related to the situation in which the link layer does not pay attention to delivering the packets in the correct order causing DUPACKs and fast retransmission [BPSK96].

TCP-Aware link layer: the best-known implementation of this proposal is Berkeley's Snoop protocol [BKGM01]. It is based on the introduction of a snoop agent at the base station that monitors every TCP segment that is sent or received by the mobile hosts, who are moving in the interested cell. The snoop agent maintains a soft state for each TCP connection. It buffers data packets not yet acknowledged at the base station BS and when DUPACKs are received, if the packets missing are present in the buffer it retransmits them on the wireless link and it does not forward the DUPACKs, avoiding an unnecessary fast retransmission phase at the TCP sender. The advantages introduced by the snoop protocol are based on its capacity to remedy packet loss faster than TCP and hiding the deficiencies of the connection. But this is true only if the RTT of the connection is small enough and the connection is continuous. The disadvantages are that Snoop requires the base station to be able to examine the traffic between the sender and the receiver and consequently it cannot work if the IP traffic is encrypted. Moreover, for Snoop to work efficiently it needs the data and the corresponding acknowledgements to traverse the same access point.

TCP-Unaware LL Protocols: they attempt to imitate the Snoop protocol, without making the BS TCP-aware.

TULIP (Transport Unaware Link Improvements Protocol) is not aware of which transport protocol is used, but it needs to know the type of service requested by the packet. It works like Snoop, buffers packets and retransmits packets lost in order to avoid TCP to enter into congestion avoidance or fast retransmit and it has been showed that it can reach better performance than the Snoop protocol.

DDA (Delayed duplicate acknowledgements) is based on the same principle, but with the specification that each TCP segment has to be encapsulated on LL frame and each TCP ACK has to be encapsulated in a LL ACK. The module knows a packet has been lost when it receives LL-DUPACKs: it retransmits the packet lost and waits to send the DUPACKs to the sender for a certain amount of time. If any ACK is received before the expiration of this fixed time it forwards the DUPACKs to the sender. Anyway the protocol still needs further study [MDKM00].

Split Connection

The main idea at the basis of this kind of approach is the observation that wireless and wired links present so different characteristics that it may be better to develop specialized protocols for each of them. The TCP connection is split in two different connections at the base station: one connection with the fixed host on the wired link and the other one with the mobile host on the wireless link. On the wireless link a protocol such as TCP or other may be used. Split-TCP proposals are Indirect-TCP (ITCP) and MTCP [MDKM00, Pen00]. The Mowgli system proposes a different approach that involves all protocol layers and offers the option to replace TCP/IP on the wireless link with a wireless specific protocol [KRA96].

TCP Modifications

Several improvements of the TCP protocol have been proposed. We consider just some of them. For a more detailed analysis see [Pen00].

TCP Selective Acknowledgements Options (SACK) [FMMP00, MMFR96]: it is a mechanism developed to recover from a situation of multiple dropped packets. TCP uses a cumulative acknowledgement system. In the case of multiple drops in a single window, TCP fast recovery generally performs badly because there is no way to acknowledge blocks of packets received out of order, but correct. With selective acknowledgments the sender can communicate to the receiver more precisely which packets have been received and which must be retransmitted. This additional information can be piggy-backed in a duplicate ACK segment that is sent when an out-of-order segment is received. Anyway the use of SACK is optional and, if it is supported, it must be negotiated at the beginning of the connection.

Increasing TCP's Initial Window: [AFP98] suggests increasing the initial congestion window for TCP from one segment to 4Kbytes. This improvement is due to the fact that TCP with an initial window of one segment is time-consuming in the slow start phase. A packet loss

at the beginning of the connection would lead to a RTO, as there are not enough packets in the network to produce three duplicate acknowledgments that would trigger the fast retransmit/fast recovery algorithm. Increasing the initial congestion window results in faster window growth during a slow start and can be useful for short flows and connections characterized by long RTT. However this optimization is still under discussion.

Active Queue Management [BCCD98]*:* as explained in detail later (see Section 3), this kind of algorithm allows the sender to prevent congestion collapse by controlling the average queue size of the router buffer and signalling to the sender of the incipient congestion before the queue overflows. With networks that present long delays it is much more important to avoid congestion. Random Early Detection (RED) [FJ93] is the most relevant proposal of this approach even if several enhancements have been proposed and deployed.

Explicit Notification Schemes: TCP Congestion Control is based on the assumption that most of the packet losses are due to congestion. The congestion is signalled by the absence of acknowledgements and the sender should reduce its sending rate when it is detected. The ideal TCP should retransmit a packet lost for transmission errors without taking any congestion measures. We introduce two different approaches: *Explicit Congestion Notification* (ECN) [RFB01] and *Explicit Loss Notification* (ELN) [BK98].

On one hand TCP with *Explicit Congestion Notification* (ECN) can signal the router of incipient congestion without discarding packets, but simply setting a bit in the packets header. In this way the TCP sender is able to understand if it must reduce its transmission rate because of incipient congestion (ECN bit set) or if a packet loss is due to a transmission error and it must not take congestion measures (ECN bit not set). ECN can be implemented as a complement of RED. See section 3.3 for more details.

On the other hand *Explicit Loss Notification* (ELN) explicitly signals the sender when a loss is due to link errors (ELN bit set) and implicitly the case of incipient congestion (ELN bit not set). The base station stores the information about the missing packets in the packet sequence received from the mobile host towards the destination. When a DUPACK is received the BS compares the sequence number with the indications cached and if there is correspondence sets the ELN bit in the DUPACK forwarding it to the mobile host. So the sender seeing the ELN bit set understands the packet has been lost for error on the transmission link: retransmit the packet without decreasing its congestion window. Anyway this scheme is in general avoided because it is difficult to implement.

The limited transmit [ABF01]: it is a mechanism to recover from packet loss when the congestion window at the TCP sender is small and does not allow the sender to receive the three DUPACKs required to trigger fast retransmit. This can occur in several cases for example when the flow of data to send is relatively short. It allows the TCP sender to send new data segments upon the first two DUPACKs if the receiver's advertised window is large enough and if the value of the outstanding segments is at most two more than the congestion window. Then the receiver answers to the new data sending acknowledging the same packet as before and TCP's fast retransmit phase starts.

The limited receiver window [DMKM01]: it is one proposed enhancement regarding only the TCP sender side. The delay bandwidth product is usually quite small in a slow wireless or wired link and consequently the capacity of the router buffer is limited. Being that the default receiver's advertised window is quite big the TCP sender may increase the congestion window above the delay bandwidth product and cause a packet loss in the intermediate routers due to buffer exhaustion. Several benefits can be derived by decreasing the Receiver Advertised Window as show in a study conducted by [ZPBS02].

The Eifel algorithm [LK00, LS00]: it attempts to solve the problem of spurious timeouts and spurious fast retransmit. These problems may be relevant if we consider TCP running across wireless networks where wireless access links may be subject to handover and resource pre-emption or hosts connected to the Internet may experience a radio coverage hole resulting in frequent disconnections. A spurious timeout is a timeout that expires prematurely and it is caused by a sudden increase of the Round Trip Time to a different value from what has been previously estimated. When a timeout expires TCP forces the sender to reduce his congestion window and to enter fast retransmit, decreasing the network throughput. A spurious fast retransmit occurs when the packet re-ordering length is greater or equal to the DUPACK-Threshold. The packet re-ordering length means the number of packets received correctly, but out of order. Also in this case spurious fast retransmit causes a degradation of the network throughput since the sender reduces its window and retransmits packets already received. The consequence in both cases is the so-called spurious retransmission. At the basis of both problems there is the incapacity of the receiver to disambiguate between ACK of the original transmission and ACK of the retransmission (retransmission ambiguity). The Eifel algorithm manages to solve the retransmission ambiguity using the TCP timestamp option as recommended in [IMLG02], even if it is possible to adopt other solutions like those discussed in [LK00].

*Disabling RFC1144 TCP/IP Header Compression [*IMLG02*]:* in the case of a wireless link, with a high loss rate, adopting RFC1144 header compression can have negative implications in recovering from a packet loss through fast retransmit. In fact RFC1144 specifies not to transmit the entire header of consecutive packets, but only the variations between them. If one packet is lost the synchronization is lost too.

Wireless Application Protocol (WAP)

The WAP Forum [WAP01] is an industry association that has specified standards for wireless networks. The WAP has defined a new protocol stack, based on five protocol layers: Application Layer (WAE), Session Layer (WSP), Transaction Layer (WTP), Security Layer (WTLS) and Transport Layer (WDP) [Pen00].

The transport layer WDP provides unreliable datagram service to the higher layers. If the bearer service supports IP, UDP is used instead of WDP and in the other case, with IP not supported, WDP implements an intermediate protocol that works as an adaptation level between the bearer service and the WAP protocols. A key concept at the basis of WAP protocols is that the presentation and application layers are responsible for the reliability of the service in general provided by the TCP layer. A typical scenario in which WAP is used sees a WAP client, a WAP proxy and an origin server. The WAP protocol provides the functionality of a split connections system. The client uses the WAP protocol to send requests to the WAP proxy. The proxy translates the request into a WWW request and passes it to a common web server, using a standard TCP/IP wired network. Then the web server answers to the proxy encoding the answer in a binary format in order to minimize the load on the link. At the end the proxy forwards the answer to the WAP client.

Alternative Transport Protocols

Some protocols have been developed specifically for wireless links, but their application is still problematic. Some of them are Wireless Transmission Control Protocol (WTCP) [Sinha99], Wave-and-Wait Protocol (WWP) [TBV00] and TCP-Probing [TB00].

The most important feature that differentiates them from the TCP protocol is the role the receiver plays during the connection and the congestion control mechanism. In fact the receiver controls and decides the rate the sender has to keep over the link. Basically the idea is that the receiver uses some algorithms to calculate the optimal rate for the connection and asks the sender to modify its rate according to that estimation. The most important disadvantage of this

structure is that the receiver, in general the mobile host, has to employ more complicated technologies and this means greater power consumption.

3 Active Queue Management

This Section demonstrates a significant weakness in current congestion control in the Internet. In order to address this inefficiency, an Active Queue Management algorithm has been proposed: Random Early Detection (RED). We present the theoretical approach on which is based and the most important imperfections it presents. In the last part we introduce Explicit Congestion Notification (ECN) mechanism to be used in combination with RED in order to avoid unnecessary packet drops.

3.1 Need for Active Queue Management

In the current Internet, the TCP protocol detects congestion only after a packet has been dropped at the router. The TCP source uses the receipt of the three duplicate acknowledgements or the expiration of a retransmit timer as indication of congestion and consequently reduces the congestion window. The common method for managing router queue lengths is to fix a maximum length (in terms of packets) for each queue, accept packets for the queue until this limit is reached, then refuse subsequent incoming packets until the queue decreases because a packet from the queue has been transmitted. This technique is known as *Tail Drop*, since the packet that arrived most recently, the one on the tail of the queue, is rejected and discarded when the queue saturates. This is the traditional mechanism that has been used in the Internet for years, but it has two important disadvantages, as stated in RFC 2309 [BCCD98].

1. Lock-Out

   The tail drop mechanism may allow a monopolization of queue resources by a single or few flows, denying other connections the possibility to find place in the router buffer.

2. Full Queues

   Since congestion is detected only after a packet has been discarded and this occurs just when the queue router is full, Tail Drop allows queues to reach a full status and to persist in this steady state for long. Instead, the average queue size should be kept low and fluctuations in the actual queue size should be allowed, thus to accommodate bursty traffic

and transient congestion. In fact having a small queue in the buffer reduces the end-to-end delay, but also permits TCP sessions that traverse the saturated queue not to loose all of an entire packet train, resulting in an increased throughput for the network. Buffering in the network has to work in order to absorb data bursts and transmit them in a period of low link utilization. This means that the queue limits should reflect the size of bursts we want the network to be able to absorb.

Approaches like *Random drop on full* or *Drop front on full* are similar to Tail Drop and do not solve the problem of full queues. The only difference they have from Tail Drop is the criteria used to choose which packet to discard when the queue saturates. The former chooses randomly a packet in the queue and the latter drops the packet at the front of the queue. In order to solve the problem of full queues in routers this basic queue-admission policy can be modified by establishing admission criteria that direct which packets may be discarded even though space is available on the queue, but near to congestion. This general approach is called Active Queue Management (AQM). Such an algorithm does not eliminate packet drops, but on the contrary, it discards or marks packets at an earlier stage in order to solve the congestion problem for flows that are responsive to packet drops as congestion signal. The benefits of its introduction are several.

1. First of all it allows to reduce the number of packets dropped in the router, because keeping the average queue size small the router buffer can absorb bursts without discarding packets, or better, dropping less packets in the case of queue overflows than in the network implementing Tail Drop or other algorithms. This is important not only because dropped packets represent a discard of resources, but also because TCP recovers easier from a single packet than a burst of packet drops.
2. Then it solves another drawback inherent in tail drop: the problem of synchronization. When several TCP connections are sharing a bottleneck link, implementing Tail Drop as discarding policy in the buffer of such a bottleneck, losses tend to appear periodically, producing a synchronization of the congestion window of the TCP connection. Active queue management schemes may avoid this synchronization giving the possibility to the several TCP connections to reduce their sending rate at different moments and not all the same moments. This permits to achieve higher throughput.
3. It provides lower delay on the link thanks to the reduced size of the buffer queue.
4. It solves the problem of lock-out, as defined before.

3.2 RED congestion control mechanism

One form of AQM being proposed by the IETF for deployment in the network is Random Early Detection (RED). The RED scheme was first presented in [FJ93]. Its objective is to provide control on the average buffer in order to guarantee the following benefits.

1. avoid congestion
2. minimize packet loss and queueing delay
3. avoid global synchronization of sources
4. maintain high link utilization and maximize the *global power* (the ratio of throughput to delay)
5. remove biases against bursty sources and guarantee fairness
6. be useful for a wide range of environments, with a variable number of connections with different round trip times, data loads and throughput.

All available empirical evidence shows that the deployment of active queue management mechanisms in the Internet would have substantial performance benefits and one of the major is the fact that this kind of algorithm can absorb bursty traffic. Moreover, in addition to keeping the average buffer occupancy low, it solves the problem of synchronization. The probabilistic approach allows routers to drop packets in proportion to the connection's share of the total bandwidth. Consequently the discards will generally regard the greediest connections. Thanks to all this advantages RFC 2309 [BCCD98], often referred to as the RED Manifesto, states that Internet routers should implement some active queue management mechanism to manage queue lengths, reduce end-to-end latency, diminish packet dropping and avoid lock-out phenomena. RED has been recommended as the default queue management algorithm since it requires a moderate overhead to be implemented in current gateways.

3.2.1 RED algorithm

RED congestion control mechanisms monitor the average queue size for the output queue and choose connections to notify of the incoming congestion. It discards packets that arrive at the router selectively, hence TCP connections, after they have detected lost packets, reduce their transmission rate and congestion can be prevented. Moreover, RED drops packets in a probabilistic manner and such probability grows with the estimated average size of the queue.

```
Initialization:
    avg ← 0
    count ← −1
for each packet arrival
    calculate the new average queue size avg:
        if the queue is nonempty
            avg ← (1 − w_q)avg + w_q q
        else
            m ← f(time − q_time)
            avg ← (1 − w_q)^m avg
    if min_th ≤ avg < max_th
        increment count
        calculate probability p_a:
            p_b ← max_p(avg − min_th)/(max_th − min_th)
            p_a ← p_b/(1 − count · p_b)
        with probability p_a:
            mark the arriving packet
            count ← 0
    else if max_th ≤ avg
        mark the arriving packet
        count ← 0
    else count ← −1
when queue becomes empty
    q_time ← time
```

Saved Variables:

- avg: average queue size
- q_time: start of the queue idle time
- $count$: packets since last marked packet

Fixed parameters:

- w_q: queue weight
- min_{th}: minimum threshold for queue
- max_{th}: maximum threshold for queue
- max_p: maximum value for p_b

Other:

- p_a: current packet-marking probability
- q: current queue size
- $time$: current time
- $f(t)$: a linear function of the time t

Figure 3-1. Detailed algorithm for RED gateways [FJ93].

The RED algorithm, as explained in [FJ93], consists mainly of two phases. In the first stage the RED gateway calculates the *average queue size* (*avg*) that determines the degree of burstiness allowed in the router queue. The second phase regards the calculation of the packet dropping probability, given the moving average of the queue size. The moving *avg* is computed by a low-pass filter with an exponential moving average with weight w_q:

$$avg_{n+1} = (1 - w_q) avg_n + w_q\, q_{ist}$$

where q_{ist} is the *instantaneous queue size*. *n* refers to a time granularity which is mandatory for this sort of computing. The previous expression can be interpreted as a low-pass filter through which the signal *instantaneous queue size* passes, giving as output the *average queue size*. w_q is the time constant of the filter, which prevents the *avg* from being sensitive to noise. As demonstrated in [FJ93], once fixed the minimum threshold min_{th} and the burst of packets L we want to be able to absorb, w_q is bounded in this way:

$$L + 1 + \frac{(1 - w_q)^{L+1} - 1}{w_q} < min_{th}$$

The RED algorithm estimates *avg* at each packet arrival with the inconvenience that it misses the dequeue movements when there are no packet arrivals. For example, if one packet arrives at time 0 when the instant and the average queue sizes are 500 packets and the next one arrives after 250 packets have left the buffer, RED would calculate an average queue length near to 500, because the instant queue size is 250, but the average is still 500.

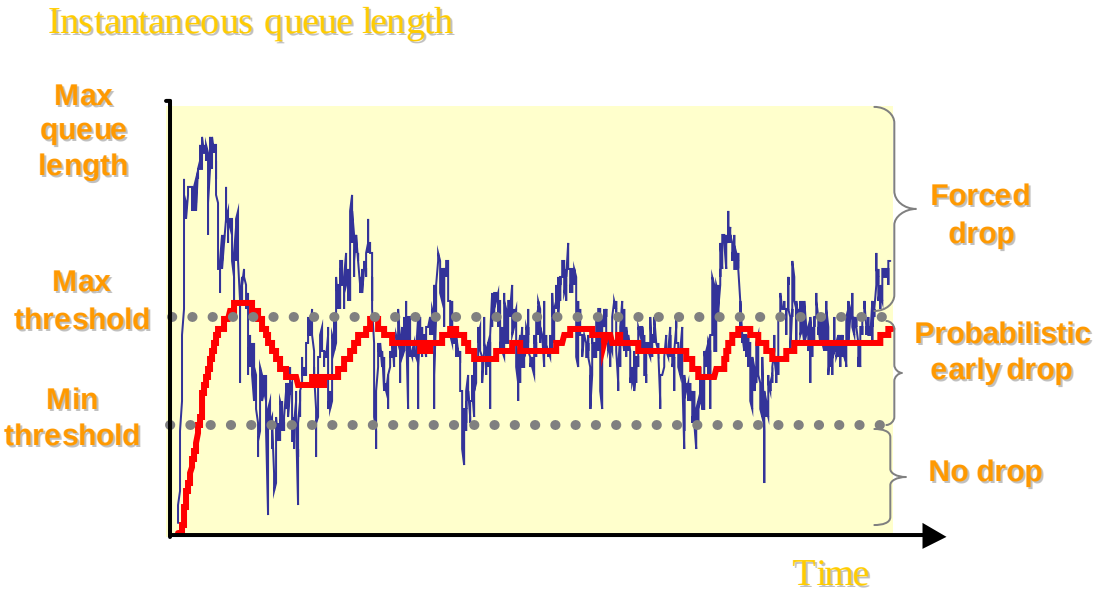


Figure 3-2. Instantaneous queue length [[CJOS02].

The RED pseudo-algorithm is formally reported in Figure 3-1. Note that the algorithm is not based on the instantaneous queue size, but on an approximation of the average queue size to improve buffer utilization through low average buffer occupancy as illustrated in Figure 3-2. When the moving average is below the minimum threshold min_{th}, no packets are dropped and the new packets are queued. If it is above the maximum threshold $max_{th,}$ every incoming packet is dropped. Between these boundary conditions, the incoming packets are marked with a *packet-marking probability p'* (intermediate) whose value is calculated on the basis of the average queue size. As *avg* varies from min_{th} to max_{th}, the packet-marking probability (intermediate) *p'* varies linearly from 0 to the max_p (Figure 3-3), which is an upper bound on the temporary packet drop probability:

$$p' = max_p\ (avg - min_{th})/(max_{th} - min_{th})$$

The final packet-marking probability *p* increases slowly with the *count* variable:

$$p = p'/(1 - count * p')$$

count is the number of unmarked packets since the last time a packet was dropped or since the *avg* exceeded $min_{th.}$

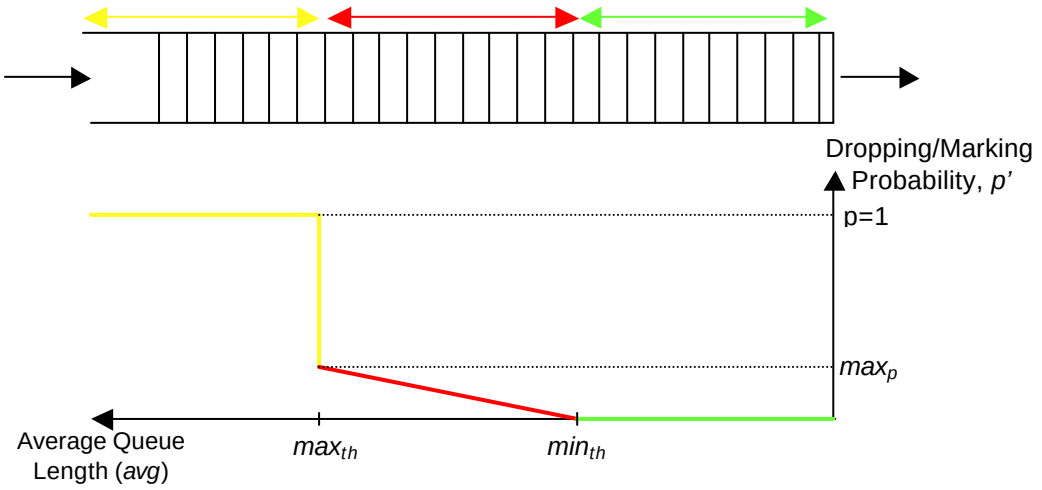


Figure 3-3. Dropping/Marking behaviour of RED.

It is interesting to note that the correction introduced by *count* in the calculation of *p* assures that the drop probability is uniformly distributed [CEP99]. We consider two different methods to determine the final dropping probability *p* as linear function of *avg*. Let X be the number of packet arrived between two successive marked packets.

1. *X is geometric distributed*

Each packet is marked with probability p', as defined before, and

$$Prob[X = n] = (1 - p')^{n-1} p'$$

which is the geometric distribution. Consequently by setting p=p' we obtain X geometric distributed.

2. *X is uniformly distributed*

On the contrary setting *p=p'/(1-count* p'),* as suggested before, we have:

$$Prob[X = n] = \frac{p'}{1-(n-1)p'} \prod_{i=0}^{n-2} \left(1 - \frac{p'}{1-ip'}\right) = p' \quad 1 \le n \le 1/p',$$
$$0 \qquad\qquad n > 1/p'$$

that is the definition of uniform distribution on the interval [1,1/*p'*]. This second result is much more desirable than the first one since marking packets at regular intervals and equally distributed guarantees not to incur in connection synchronization. In fact, Figure 3-4 shows the distribution of marked packets obtained using the first definition for *p* and setting *p'=0,02* (in the upper part) and the second one for *p'*= 0.01 (in the inferior part). The number of drop packets is approximately the same (100 out of 5000 incoming packets), but while in the first case the points indicating the marked packet are concentrated in some areas, with the second method they are equally distributed over the full interval. It is better to avoid to have marked packets close to each other because this can induce the active connections to have packets discarded at the same time and consequently to reduce their transmission rate at the same moment resulting in a global synchronization.

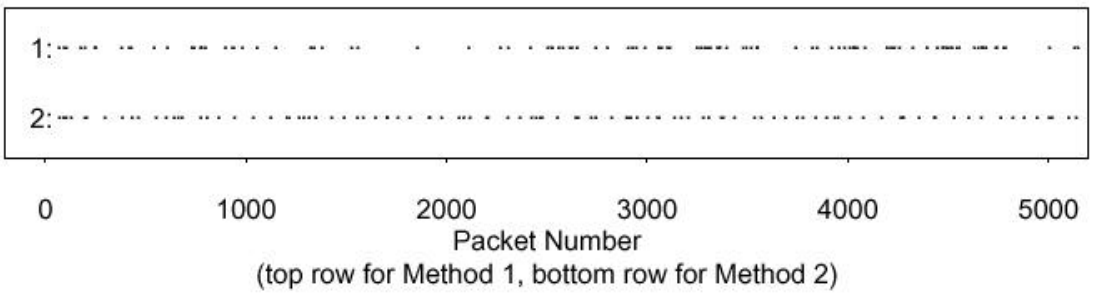


Figure 3-4. Comparison between two marking probability method [FJ93].

By default the RED algorithm follows the original version and measures the queue size in packets rather than in bytes. Modifications to this basic version have been implemented to allow the application of RED to different networks such as ATM [RBL99] and with this option the algorithm changes in this way:

$p' = max_p\ (avg - min_{th})/(max_{th} - min_{th})$

$p' = p' * L/M$

$p = p'/(1 - count * p')$

L is the *Packet Size* and *M* is *the Maximum Packet Size* specified for that connection. This modification ensures that the probability that a packet is marked is proportional to the packet size in bytes.

3.2.2 Parameter sensitivity

Unlike Tail Drop where the only free parameter is the buffer size in RED we have several parameters to set, like w_q, min_{th} ,max_{th} and max_p. In order to achieve a good congestion avoidance the parameter sensitivity must be kept low. Some rules to follow are suggested in [FJ93,Flo97] and shortly we summarize them:

1. set min_{th} high enough to guarantee a high utilization of the link. The optimal value for it depends on the link speed, propagation delay and buffer size. In the ns-2 simulator, min_{th} is set as default to 5 packets (or, for a queue measured in bytes rather than packets, 5 packets times the mean-packet-size in bytes) and according to the link characteristics it seems to be a good rule not to keep it too small, such as 1 or 2 packets because we do not allow enough burstiness in the arrival process.

2. $max_{th} - min_{th}$ must be kept large enough to avoid global synchronization. If it is small, *avg* can oscillate regularly from the minimum till the maximum threshold and the gateway will discard a lot of packets in the same moment. The [FJ93] paper recommends setting max_{th} to at least twice min_{th}, but the current rule of thumb is to set max_{th} to three or more times min_{th}. In the ns-2 simulator min_{th} and max_{th} are set to 5 and 20 packets respectively.

3. set max_p to 0.1. There is no need to set it higher than 0,1, because we do not want a router to work with high drop rates. Moreover the RED gateways perform best when the packet dropping probability changes fairly slowly as the average queue size varies.

4. set $w_q \geq 0.001$ and vary that according to the burst length L we want to be able to accommodate, without dropping any packets. For example if min_{th} is 5 packets and L is 50 packets, then w_q has to be around 0.0042. In the ns-1 and ns-2 simulator it is set to 0.002. That is why if w_q is too low, then *avg* is responding too slowly to transient congestion. If it is too high, then the estimated average queue size is too dependent on the instantaneous queue size.

The conclusion is that unfortunately there are no precise optimal values because they depend on a wide range of factors, including not only the link speed and propagation delay, but also the traffic characteristics.

3.2.3 Alternative RED mechanisms

There are a number of imperfections in RED's performance that can be resolved by adding additional mechanisms to RED gateways. We can consider some of these variations to the original RED algorithms. The proposed improvements are relative to several aspects.

Adaptive RED ([FKSS99]) focuses on the problem of parameterizing the RED algorithm in order to reach good performance in each possible scenario. For example, in the case of a bottleneck link shared by N connections, congestion notification requires each connection to reduce the traffic of $\left(1-\frac{1}{2N}\right)$. If N is large the effect of traffic reduction by each connection will be small, and on the contrary, if N is small it will be considerable. In the first case we need a more aggressive RED algorithm in order to avoid packet loss and to perform as a simple Tail Drop queue; in the second case we need a less aggressive algorithm to keep the link utilization to an acceptable level. Hence there are two main drawbacks in using RED [FGS01]:

- The average queueing delay with RED is sensitive to the traffic load and to parameters, and consequently is not predictable in advance. When the congestion is light and/or max_p is high, avg is close to min_{th} and when the congestion is heavy and/or max_p is low, the avg is near to max_{th}. In order to have some guaranteed delay with RED it is necessary to perform a frequent tuning of its parameter according to the traffic variations.
- The second point is that the throughput is also sensitive to the traffic load and to the parameters. In particular when the avg is larger than max_{th} the throughput performance decreases greatly.

The solution proposed is to provide an adaptive variation of RED parameters based on the avg. The key idea is to adapt max_p (the initial discarding probability parameter) in order to keep the average queue size between min_{th} and max_{th}. When $min_{th} \leq avg \leq max_{th}$ there are no variations, but if $avg < min_{th}$, RED must be less aggressive and $max_p = max_p + \alpha$. Otherwise ($avg > max_{th}$) RED must be more aggressive and max_p is increased: $max_p = max_p + \beta$. α and β are constant factors. A pseudo-code description of the algorithm is given in Figure 3-5.

```
Every interval seconds:
   if (avg > target and max_p ≤ 0.5)
      increase max_p:
      max_p ← max_p + α;
   elseif (avg < target and max_p ≥ 0.01)
      decrease max_p:
      max_p ← max_p * β;

Variables:
avg:  average queue size

Fixed parameters:
interval:  time; 0.5 seconds
target:  target for avg;
   [min_th + 0.4 * (max_th − min_th),
      min_th + 0.6 * (max_th − min_th)].
α:  increment; min(0.01, max_p/4)
β:  decrease factor; 0.9
```

Figure 3-5. The RED Adaptive algorithm.

RED with Penalty Box [Zhang00] is one of the first proposals intended to distinguish responsive users from unresponsive users. It is based on the observation that high bandwidth flows see proportionally larger amounts of packet loss. It maintains a list of the recent packet loss events verified in the network in order to identify all the misbehaving flows in the penalty box. These unresponsive flows are limited in the rate using a mechanism such as class-based queueing.

FRED (Flow RED [LM97]) tries to solve some particular cases of unfairness allowed in RED. The traffic traversing an Internet gateway tends to fluctuate and is generally greedy. An ideal gateway should buffer temporary excess loads and provide negative feedback if the excess load persists. This can guarantee fairness. The solutions to this problem are several. For example, keeping a separate queue for each flow in the gateway and implementing hop-by-hop flow control between the gateways can ensure fair shares of bandwidth, but even if effective it is very costly to implement. Otherwise an approach as the FIFO discipline that provides a feedback to the senders by dropping packets is simple, but unfair, and tends to penalize bursty connections. According to RED algorithms the amount of marked or dropped packets for each connection is proportional to their bandwidths. But in this way RED does not guarantee to give each connection the same fraction of the total resources and ignore the control on misbehaving flows. For example, if two TCP connections unequally share one link, discarding periodically one packet from the slowest connection will prevent it from claiming its fair share, even if the fast-speed flow experiences more packet drops. TCP connections with small window sizes are

more damaged by packet loss than those with large windows, since they need a short time, such as one RTT, to recover from multiple packet drops, while a flow with a small window may take a long timeout to recover. Moreover RED is developed for adaptive flows. TCP congestion control reacts to small increases in the loss rate with great decreases in the transmission rate. A source that sends too fast would receive an unfair share of the total bandwidth. FRED is an improvement of RED able to provide major protection for bursty and low-speed flows and it guarantees more fairness than RED in presence of different traffic types. The main difference between them is that FRED keeps the information state for those flows that have packets buffered in the router. It introduces new parameters that allow a more accurate estimation of discarding probability.

- min_q and max_q, the minimum and maximum number of packets each flow is allowed to have in the buffer;
- *qlen*, the number of packets in the buffer for each flow;
- *avgcq*, the estimation of the average per-flow buffer count;
- *strike*, the number of times each connection has failed to react to congestion notification.

FRED allows each connection to buffer min_q packets and apply discard probability to the subsequent packets. It never permits a flow to buffer more than max_q packets and it stores the times in which it has tried to do that in *strike*. Flows with high *strike* are more subject to loss packets and they cannot buffer more than *avgcq* packets. Moreover, in FRED the frequency of *avg* calculation is higher than in RED. In fact, the averaging is done at both arrival and departure in order to provide a more accurate estimation of *avg*, and when a packet is dropped *avg* does not vary. Simulation results show that FRED is often fairer than RED when handling connections with different round trip times and window sizes and it allows detecting unresponsive users. Anyway, it adds overhead to store information about the flows which have packets buffered in the router. The cost of this per-active-flow accounting is proportional to the buffer dimension and independent of the total number of connections served by the gateway.

SRED (Stabilized RED [OLW99]) is a RED-derived mechanism that attempts to improve RED performance by considering a further element in calculating discard or marking probability, that is, the estimated number of active connections or flows. The basic idea for estimating the number of active flows is to compare, every time a packet arrives at the buffer, the new packet with one entry randomly taken from the so-called *Zombie list*. When the two packets belong to the same flow (for example, they have the same destination and source addresses, the same

destination and source port numbers and the same protocol), a hit is declared and the count is incremented in the corresponding entry in the Zombie list. Otherwise the entry in the Zombie list is replaced with a probability p. Actually in the version of SRED proposed by [OLW99] the discard probability is entirely based on the instantaneous length of the buffer queue and on the number of active flows, but the adding of average queue size estimation does not present any difficulties. This improvement has the advantage to stabilize the buffer occupancy, independently of the number of active connections and to provide a way through the hit-mechanism to detect situations of unfairness, in which some flows are attempting to take more than their fair share of bandwidth (misbehaving flows).

CHOKe (CHOose and Keep for responsive flows, CHOose and Kill for unresponsive flows [PPP00]) is another proposal, based on the RED algorithm, whose goal is to approximate fair queueing and be, at the same time, simple to implement. It is based on the assumption that the FIFO buffer is a reliable indication of which flows are consuming a great amount of resources. If a packet arrives at a congested router ($avg > min_{th}$), if the $avg > max_{th}$, the packet is discarded, like in normal RED, otherwise the new packet is compared to a randomly chosen packet from the FIFO buffer. If they belong to the same flow they are both discarded, otherwise the randomly drawn packet is left intact and the new one is admitted into the FIFO buffer with a probability calculated like in the original RED, based on the *avg*. The advantage of this algorithm compared to those mentioned above is that it does not need any state information and consequently it introduces the minimal implementation overhead.

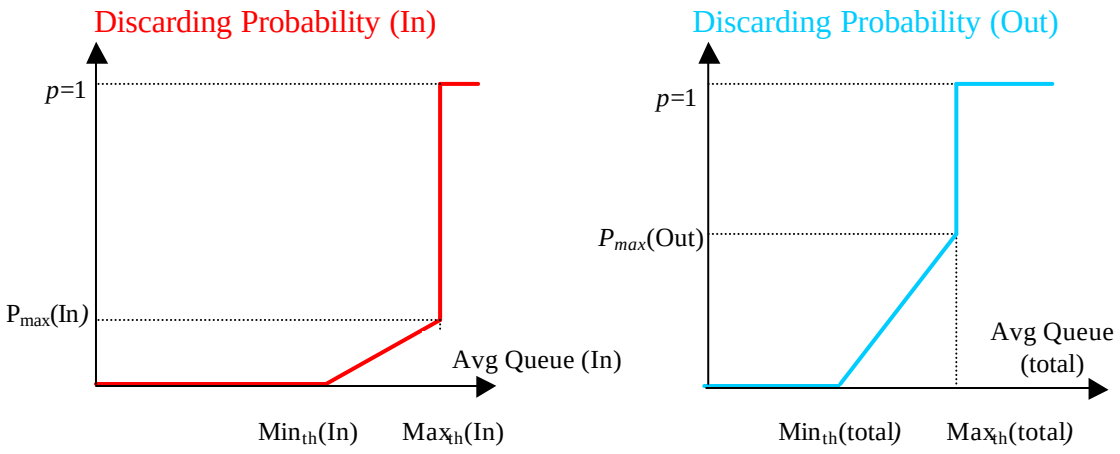


Figure 3-6. RIO algorithm.

RIO (RED with In/Out bit [CF98]) is based on the two-drop precedence policy. A packet is marked at the edge of the network as IN or OUT of its service contract and it is treated

differently inside the network, on the basis of this priority classification. The router inside the network keeps just one queue for IN and OUT packets and apply to them two different RED algorithms as we can see in Figure 3-6. Instead of using the same average queue size for both priorities, it uses the average queue size for OUT (out of profile) packets, and the average queue size without taking into account the queued OUT packets for IN (in profile) packets. In time of congestion the router starts to drop OUT packets and eventually, if congestion persists, will start to discard IN packets, as well.

WRED (Weighted RED [CISCO98]) WRED was initially proposed as RIO in [CF98]. It has been developed as an extension of the RED approach by taking into account the priority of packets. It assigns to every priority a different RED algorithm, making it possible to differentiate the performance of different TCP connections whose packets are queued in the same queue. Different QoS can be provided for different classes. For example packets from higher priority traffic is dropped with a lower probability than the standard traffic, during periods of congestion. This is implemented with two RED algorithms running in parallel. In order to reduce the packet loss rates experienced with RED a different queue management algorithm has been implemented.

BLUE [FKSS99] differs from all the classical algorithms such as RED and derivations from RED based on the control of the average queue variations, since it is based directly on packet loss and link utilization history. BLUE uses a unique marking/dropping probability p_m. If the router buffer saturates continually and packets are discarded, p_m is incremented. On the other hand, if the queue is almost empty or the link idle the p_m is reduced. This allows BLUE to learn the correct rate it needs to send back a congestion notification. The algorithm is formally shown in Figure 3-7. Besides the marking probability it introduces two new variables:

- *freeze_time*, the minimum time interval between two successive updates of p_m
- *d1* and *d2* that indicate the amount by which p_m has to be incremented or decremented respectively.

Upon packet loss (or Ql_{en} >L) event:
    if ((now-*last_update*) > *freeze_time*) then
        $p_m = p_m + d1$
        last_update=now

Upon link idle event:
    if ((now-*last_update*) > *freeze_time*) then
$p_m = p_m - d2$
last_update=now

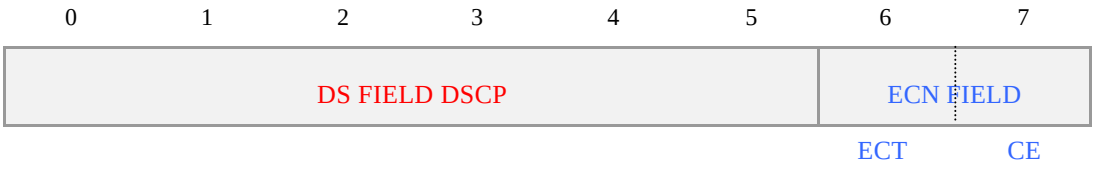
Figure 3-7. The BLUE algorithm.

Simulations prove BLUE to perform better than RED, even when operating with a smaller buffer.

SFB (Stochastic Fair BLUE [FKSS01]) is an extension of BLUE developed to solve the problem of non-responsive flows. SFB tries to identify the non-responsive flows and limit them to a fixed amount of bandwidth.

3.3 Explicit Congestion Notification (ECN)

The Active queue management algorithm detects congestion before the buffer queue overflows and signals to the end nodes the incipient congestion. As mechanism for congestion indication the router may drop packets, but a more efficient way to achieve that is marking packets. The router may use the Congestion Experience (CE) codepoint in the IP packet header as an indication of congestion for the end-nodes [RFB01].



DSCP=differentiated service codepoint
ECN=Explicit Congestion Notification

Figure 3-8. The DS and the ECN Fields in the IP-header [RFB01].

ECT	CE	Codepoint
0	0	Not-ECT
0	1	ECT(1)
1	0	ECT(0)
1	1	CE

Figure 3-9. The ECN Field [RFB01].

Considering the IP header in the IPv4 TOS octet, represented in Figure 3-8, the ECN field is defined through the bits 6 and 7, defined for experimental use of ECN [RFB01]. Marking packets is done through the ECN field and there are four possible ECN codepoints as shown in Figure 3-9. In the ECN field we distinguish the ECT (ECN-Capable Transport) bit and the CE bit. The ECN-Capable Transport codepoints, ECT (0) and ECT (1), mean that the

end-points are ECN-capable. The Not-ECT codepoints means that the packet is not using the ECN field. Then there is the CE codepoint used to signal the presence of congestion.

When the router is ready to discard packets to signal end-points of incipient congestion, at the reception of a new packet it has to check the ECT codepoint. If is set (ECN-capable flows) it should set the CE codepoint and forwards the packet, or on the contrary (not ECN-capable flows) discard packet. When the end-nodes receive a packet ECN-capable with the CE bit set, the reaction will be the same to the detection of a drop packet. It is important that the router sets the CE codepoint only in the case it would have discarded the packet. If the router is not in the presence of incipient congestion it just has to forward the packet without modifying the CE codepoint. We expect that using ECN in combination with RED the router will set the CE codepoint when the average queue size exceeds the fixed threshold, instead of dropping the packet. It is evident that the measures taken by the router against congestion must not be based on the instantaneous value of the queue, but on the average queue size, because this is subject to frequent variations.

The use of two ECN-Capable Transport codepoints is due to several reasons and one of them is that it makes it more difficult for a router to erase the CE codepoint, without being discovered by the end nodes, since they have to be able to reconstruct the original one.

In order to support the use ECN field some modifications are requested to TCP level. First of all, during connection set-up, the two end nodes must negotiate if they are both ECN capable (1). Then there is a need to find a way to communicate the reception of a CE packet from the receiver to the sender (2) and to signal the corresponding reduction of the congestion window from the sender to the receiver (3).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Header Length				Reserved				CWR	ECE	URG	ACK	PSH	RST	SYN	FIN

Figure 3-10. The TCP header and the definition of bits 8 and 9 to support ECN [RFB01].

In order to support ECN functionality, two new flags are defined in the TCP header: the Explicit Congestion Notification (ECE) and the Congestion Window Reduced (CWR) flags as shown in Figure 3-10.

(1) During the connection setup the two end-hosts decide whether to support the ECN functionality. They do that by exchanging SYN packets with ECE and CWR

opportunely set. If both the nodes are ECN-capable they will use ECN notification during the connection setting in the IP-header of each packet a ECT codepoints. For more details see RFC 3168 [RFB01].

(2) The receiver at the reception of a CE packet can inform the sender of the incipient congestion setting the ECN-Echo (ECE) flag in the TCP header. The sender seeing the ECE flag set infers that congestion was encountered in the network on the path from the sender to the receiver and reacts in the common way, as TCP congestion control indicates: it halves the congestion window and reduces the slow start threshold. It is important to notice that the TCP sender should react to an ECN at most once per roundtrip time. The TCP should ignore subsequent ECNs if the source has reacted to a previous ECN or to a dropped packet in the last round trip time.

(3) When an ECN-capable TCP sender reduces the congestion window it will set the CWR flag in the TCP header of the first new packet sent to inform the data receiver that the congestion window has been reduced. The TCP receiver uses the CWR flag received to determine when to stop setting the ECN-Echo flag in the TCP header.

Several simulations show that there are several advantages deriving from the introduction of the ECN mechanism in TCP networks [Flo94]. A main benefit of the ECN scheme is in avoiding unnecessary packet drops. This is of great importance for interactive traffic and for low-bandwidth traffic where the user is delay-sensitive. Moreover, ECN provides a fast and clear mechanism to inform the sender of incipient congestion, sparing it to wait for a retransmit timeout or the reception of three DUPACKs.

On the other hand there are two potential disadvantages of ECN. One is related to the case of non-compliant TCP connections that, even being ECN capable, ignore ECN notifications, but this is a problem also for networks that based congestion control on packet drops. The other drawback is specific for the ECN mechanism and occurs when an ECN message is discarded by the network and the notification of congestion cannot reach the end-node. But if the routers implement RED they will go on calculating the average queue size and mark packets randomly, as long as the congestion persists.

4 Test Arrangements

In this chapter we describe our test methodology and the aspects our performance experiments focus on. We explain the test methodology and the test environment in terms of workload models, metrics and TCP features used in the performance measurements. During this description we follow some advice proposed in [AF99]. There are several methods for evaluating TCP's performance and there are various environments in which to develop the study. The methodology in which an experiment studies TCP has a direct impact on the relevance and utility of the results obtained and the conclusions derived.

4.1 Test Methodology

The main aim of this thesis is to analyse how the RED algorithm can solve problems related to TCP congestion and provide improved performance especially if deployed in a QoS architecture. The main result are based on a comparative study of performance when RED or traditional Tail Drop mechanism are employed. In the first phase we study the enhancements introduced by the application of the RED algorithm, as a mechanism of congestion control, and in the second one of RED applied in a QoS-enabled environment.

To evaluate TCP performance we have used a small testbed of hosts handling packets. The advantage of using testbeds involving real hosts, connected to a small network, instead of simulators, is that we do not use an abstract TCP implementation, but code found in real operating systems. Simulations sometimes make assumptions that preserve the TCP tests from inconveniences that may be encountered in the real world. On the other hand, in experiments using real implementations we have to take into account several aspects not related to TCP working, but to the operating system or to the application protocol, which may condition the results. For example, the reason why a TCP data transfer may be slow can be related to the TCP mechanism, but also to the application level or the operating system. In a testbed, real TCP implementations are applied to real networks since effects such as scheduler delays and router packet processing time can be studied.

According to the requirements defined in Section 2 and what is suggested by IETF, we have used a TCP implementation with the most common and widely deployed features. We refer here and after to this as the *Baseline* TCP. The parameters involved in this TCP implementation are:

- Basic Congestion Control: Slow Start and Congestion Avoidance algorithms as defined in RFC 2581 [APS99].
- Congestion Control: Fast Retransmit and Fast Recovery such as proposed by New Reno congestion control defined in RFC 2582 [FH99].
- Initial Window Size: 2 segments. (TCP enhancement =4MSS)
- Selective Acknowledgements: ON.
- MSS=1460 bytes.
- Delayed Acknowledgements: ON.
- Timestamps: OFF.
- ECN: OFF.

The TCP implementation we use to run the tests is based on Linux Kernel version 2.4.17. The New Reno TCP [FH99] improves TCP's ability to recover from multiple lost packets in a window of data without relying on retransmission timeout. The *Larger Initial Window* is an enhancement that needs more evaluation. The congestion control specifications require the TCP implementations to use an initial window (IW) of one or two segments [APS99]. Since using an IW of one segment, the slow start phase is time-consuming, we have chosen an Initial Window of 2 segments. The SACK option is a widely deployed algorithm today and it has been shown that the SACK algorithm performs better than several non-SACK based recovery algorithms when more than one packet is lost from a window of data [MMFR96]. Due to the fact that the TCP acknowledgements are cumulative, there is no way to acknowledge separate blocks of segments that have been correctly received, but out-of-order. The SACK option gives the possibility for the TCP receiver to inform the data sender which segments have been received.

The Baseline TCP described is applied in two different experimental scenarios. We study the performance of a TCP connection when a simple Tail Drop algorithm is deployed and afterwards we analyse the expected improvements carried out by the application of the RED algorithm. It is expected to solve problems related to congestion losses, such as over-buffering and fair sharing of resources.

4.1.1 Parameter Tuning for RED

The performance of the RED algorithm is strongly dependent of its parameters and it is difficult to find the appropriate values that will enable RED to perform equally well, under different congestion scenarios. When the parameters of RED are not correctly set for the given scenario, the performance can approach that of the traditional Tail Drop.

Parameter	Description	Suggested Value	Value
buffer-size (bytes)	Physical buffer queue size	free	depending on traffic condition
min_{th} (bytes)	Lower threshold below which no packet is dropped	3 up to 5 packets or 3 up to 5 times *avpkt*	5 packets
max_{th} (bytes)	Upper threshold above which all incoming packets are dropped	3 or 4 times min_{th}	3,5 times min_{th}
avpkt (bytes)	The average number of bytes in a packet	free	1000
burst (packets)	The number of average-sized packets allowed to burst	$\frac{min_{th} + min_{th} + max_{th}}{3*avpkt}$	$\frac{min_{th} + min_{th} + max_{th}}{3*avpkt}$
max_p (float)	Maximum drop probability	0,1	0,1

Table 4-1. The RED Parameters

Considering in part what was suggested by Floyd in [Flo97] and the observations carried out by several studies conducted on RED's performance and dependability on its parameter, such as [Larry02], we have decided to fix RED's parameters used to set router configuration following the guidelines given in Table 4-1. They are subject to variations according to the number of TCP flows considered and eventually the load carried by UDP traffic. The buffer-size parameter is set on the basis of the studied scenario. As regard the min_{th} and max_{th} parameters the proposed setting is subject to modifications since they depend on the buffer-size dimension even if the relation is not explicitly expressed. In addition the parameters must be varied according to which performance metrics are more important in the considered application. For example, to achieve high performance in terms of TCP throughput, min_{th} must be increased and max_{th} must be set quite near to the buffer size, but on the other hand, in order to guarantee a low buffer occupation, min_{th} should be kept low.

The parameters we used in our experiments are quite near to the default ones. As regards max_{th} we have chosen a higher value in order to keep the percentage of discarded packets low and to have a wide range max_{th}-min_{th}. This choice guarantees high link utilization and excellent throughput. Finally we set max_p as suggested by default. With such a value the RED gateway marks around 1/5th of the incoming packets when the average queue size is close to max_{th}.

4.2 Experimentation Environment

The testbed network used for experiments consists of two hosts communicating via a point-to-point wired link and two routers (Figure 4-1). All four machines are Intel Pentium III 1 GHz running the Linux Operating System (kernel version 2.4.17). The first link, between Kemi-1 and Kemi-2, is 100Base-Tethernet, while the other two are 10Base-TEthernet. Through this topology and setting appropriately the parameters in the router configuration, the designed network may present, for example between Kemi-3 and Kemi-4, a bottleneck link, where packet drops and congestion occur. This allows us to analyse how different congestion controls perform when implemented in the router. In our study, in the first router the simple Tail Drop discipline is in use, while in the second one the Tail Drop or RED algorithm can be deployed. All the links in this model belong to an isolated LAN and the intent is to represent the wired network immediately after a radio access network of high speed. The QoS architecture deployed in the second router can be implemented by means of TC tools. [HMVV02].

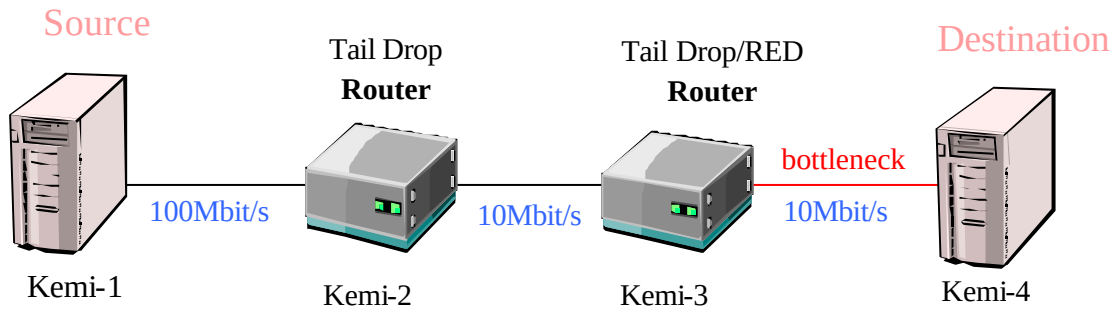


Figure 4-1. Topology of testbed network.

HTB Linux queueing discipline

Linux kernels offer a wide range of traffic control functions and they are mainly based on four conceptual components: queueing disciplined, classes, filters and policing. Each network device has a queueing discipline associated with it, which regulates how packets enqueued on that device are processed. They can use filters to distinguish among different classes of packets

and treat each class in a specific way, for example specifying one class priority over other classes. Queueing disciplines and classes are closely tied together. By queueing disciplines we can determine the way in which data is sent and note that we can only shape data that we transmit. Generally we do not have direct control (and especially with the Internet) on the amount and type of data sent. Typically each class owns one queue, but it is in principle also possible that several classes share the same queue or even that a single queue is used by all classes of the respective queueing discipline [ASK99]. Figure 4-2 shows an example of queueing discipline.

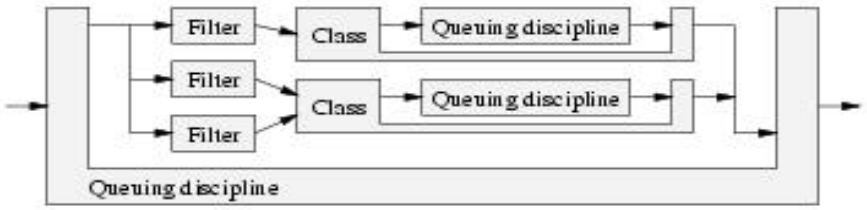


Figure 4-2. A simple queueing discipline with multiple classes. [ASK99]

In order to control the outbound bandwidth on the last link the workload generated can be classified into different classes by HTB (Hierarchical Token Bucket) queueing discipline. HTB is a packet scheduler and it is currently included in stock Linux kernels. Class Based Queueing (CBQ) is the most known and also complex qdisc deployed. When traffic enters a CBQ qdisc is classified and the so-called "filters" are consulted. They return an answer that indicates to the qdisc where to enqueue the packet, that is into which class. HTB qdisc works just like CBQ, but is more understandable and intuitive. In a HTB queue a priority can be assigned to each class. By placing different kinds of traffic in several classes and then assigning to these classes different priorities, we can control the order in which packets are dequeued and sent. In addition to this, HTB also has another nice property. It ensures that the amount of service provided to each class is at least the minimum of the amount it requests and the amount assigned to it. Thus it makes it possible to avoid starvation of any one class, since we are able to specify a minimum guaranteed rate for each class When a class requests less than the amount assigned, the remaining (excess) bandwidth is distributed to other classes which request service. For each class it is possible to specify the *rate* (the assigned bandwidth) and the *ceil* (the maximum bandwidth that can be assigned) parameters. The Priorizing traffic option is also available. It affects how the excess bandwidth is re-distributed. When the priorities are not specified or they are all of the same value, the excess bandwidth is distributed according to the rate ratios. Otherwise, when they are specified, the classes with higher priority are offered

excess bandwidth first, but always guaranteeing the respect of the rate and ceil rule. Once the classes have been decided, we set up the filters to place traffic in classes. There are several ways to do this and the filters can be based for instance on the destination and source address or on the TOS value, depending on the application.

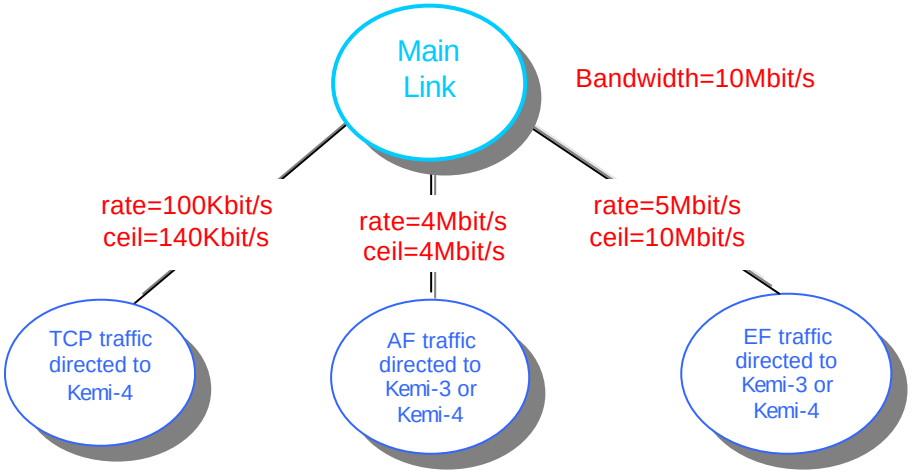


Figure 4-3. Example of HTB queueing discipline.

Considering our network topology, a HTB qdisc, in the Kemi-3 router, can separate the EF priority class from the AF one based on TOS value or also permit to distinguish the flows directed to Kemi-4 according to the port address specified. In this way the total available bandwidth of 10 Mbit/s of the second link is divided between the different classes of service. For example in the second router (kemi-3) an allocation of this kind may occur: 100Kbit/s for TCP traffic with destination Kemi-4, 4 Mbit/s for AF traffic and 5 Mbit/s for EF flows, always directed to Kemi-4. In this way HTB qdisc makes it possible to vary the bandwidth assigned to each classes and consequently cause congestion in the outgoing link. An example is represented in Figure 4-3. We have specified the rate value as proposed before and the ceil parameters too. When one or more classes are requesting less than the amount assigned, the remaining bandwidth is reallocated to the other classes in proportions to their initial allocations. The ceil argument specifies the maximum bandwidth that a class can use and limits the bandwidth a class can borrow form the others. For AF traffic the rate and ceil are the same and this means that it will always receive a maximum amount of bandwidth of 4 Mbits. For TCP traffic the maximum amount of bandwidth will be 140Kbit/s and for the EF traffic 10 Mbits/s. Thus, with this configuration, it is easy to cause congestion and study the TCP congestion control's performance through sending one or more TCP flows that can saturate the available bandwidth of 100Kbits or of 140Kbits, in case of small CBR traffic.

4.3 Workload Models

The definition of the Workload Models is extremely important in order to obtain results of acceptable confidence. Our study being mainly focused on RED performance applied in a QoS enabled architecture, we have chosen to test them in a dynamic environment with competing traffic flows. Generally the type of workload used is classified on the basis of the individual connections involved, the number of contemporary flows and their position in time.

An individual connection may carry different kinds of traffic, TCP and non-TCP traffic, such as UDP. For more details regarding the UDP protocol see Section 8.2. Having several parallel TCP connections is a common situation in the Internet. It could be the case of a user who is browsing web pages while active ftp transfer is running in the background. In a common TCP connection between two hosts, two single data flows are possible, from one host to the other one or vice versa. Anyway in most real applications the flows are unidirectional since data are sent in one direction and acknowledgments are received in the opposite one. In our tests we use single unidirectional bulk transfers consisting of a continuous flow of data packets of the maximum size allowed by the network. It is essential for our study based on a QoS enabled architecture to define several traffic classes that require different services from the network according to their priorities. For some of them delay may be essential and for other throughput can be much more relevant.

The workload source and sink computers use the proposed Baseline TCP. The traffic was generated using two different types of traffic generator: JTG (Jugi's Traffic Generator, version 1.12) for the UDP flows, and TTCP (TCP test utility) for TCP flows. TTCP is a benchmarking tool for determining TCP (and UDP) performance between two systems when sending a high volume of data over the network [Sti90]. We have employed a modified TTCP tool. It is the original version, available from [TTCP98], with some improvements made by the Department of Computer Science at the University of Helsinki. We further made some extensions to this to make it more suitable to our needs. In particular we have used it in a QoS context and we needed the specification of the TOS/DSCP field. JTG is a modification of the classic TTCP generator developed by Jukka Manner, at the University of Helsinki. It was used to generate real-time traffic towards an IP destination through a script file. The advantage of this traffic generator is double: on one hand it allows to generate traffic and assign each flow to the proper service class through the specification of the TOS/DSCP field; on the other hand the JTG package encloses tools that permit gathering information and log packets in order to calculate the desired statistics.

Traffic	Packet size (bytes)*	Protocol	Transmission rate (packets/sec)	Transmission rate (Kbit/s)*	Number of packets
VoIP (adpcm)	104 (104+28=132)	UDP	61	50,8 (64,4)	
Best Effort	5840	TCP			8
Best Effort	5840	TCP			50
Best Effort	5840	TCP			65

***** = for UDP packets the first value indicated represents the payload size in bytes or the data transmission rate. The value in parenthesis is the final value considering the UDP and IP headers .

Table 4-2. Network traffic characteristics.

Finally the Nagle algorithm [Nag84] in TCP says that a TCP connection can have only one outstanding small segment that has not yet been acknowledged ("small" means less than the segment size). In fact to send 1 byte of data we need to build a 41-byte packets (20 bytes for the IP header and 20 bytes for the TCP header). These small packets, called tynigrams, do not represent a problem on LANs, but they can add congestion on wide area networks. Referring to our tests on a small testbed network this algorithm could disturb our results. Thus in order to prevent this, we have set the size of data passed by TTCP to MSS of the connection (1460 bytes). We have also checked the minimum segment size in all the simulations. Moreover we could use a socket option TCP_NODELAY to disable the Nagle algorithm.

The traffic in each experiment was generally composed by TCP connections with different characteristics and sometimes, in addition UDP flows with UDP data packet sizes of 104 bytes as shown in Table 4-2. The study focuses on TCP connections and marginally on UDP flows. In each set of measurements the load changes by varying the number of TCP flows and the amount of data sent. The TCP packet size is the MSS size (1460bytes).

Our study will focus on the effects of RED carried by TCP connections. Generally, TCP connections can be modeled as bursty traffic, while UDP-based application as smooth traffic. TCP connections have congestion control implemented at the end host and respond to a packet drop through retransmission. However, UDP hosts do not take care of packet loss and let the upper layer application implementing congestion control. But this does not mean that RED algorithms do not have an impact on UPD applications since they are implemented in the router and not in the end-hosts.

4.4 Metrics

The performance metrics we have used in our evaluation, in order to compare the performance of different experiments include:

- **Elapsed time**: the time needed to end the entire transfer of a given size including the time to establish and close the connection. It is measured as the time from the first TCP SYN segment sent till the receiving ACK for the FIN segment.

- **Data transmit time**: the time necessary for the transmission of data, from the first to the last data packet. It is calculated from the first packet transmitted to the reception of the acknowledgement for the specified amount of data.

- **Three-way handshake phase duration**: the time necessary to establish the connection between the two hosts.

- **Throughput**: measuring it is the most straightforward way to evaluate the TCP performance. It represents the amount of user data transferred during the connection divided by the connection elapsed time. In the case of entire connection the elapsed time is calculated as the time between the moments in which the first TCP SYN message is sent (connection setup) and the acknowledgement for the TCP FIN message is received (connection tear down). In the case of partial connection it is the time between the first packet sent and the reception of the acknowledgement for the last packet in the data burst considered.

- **Fairness**: it shows the impact a connection has on a competing traffic sharing the network path. To measure fairness we use the Jari fairness index [Jai91], defined as: $f = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n\sum_{i=1}^{n} x_i^2}$ where $x_1, x_2, \ldots x_n$ are n instances of the metric of interest (e.g., the latency experienced by concurrent hosts). It lies between 0 and 1. If all the users receive equal treatment the index is 1, otherwise it may be k/n indicating that only k of the n users are receiving equal treatment and the other $n-k$ are not served. We use this index only for flows carrying the same traffic, of the same duration and starting time. In the other cases (connections of different load or different starting points) we use a macroscopic definition of fairness comparing only the flows with the same properties.

- **Number of retransmitted packets:** it can be due to a packet drop or to an excessive delay and in the second case they are unnecessary retransmissions. The parameter has to be compared to the loss rate one. This is an indication of how aggressive the TCP is and specifically the RED parameters set in the given test case. This is an essential parameter for RED evaluation since one of RED's objectives is to minimize the loss rate.

- **Retransmitted data bytes:** the total amount of bytes retransmitted during the transfer.

- **Number of duplicate ACKs received**

- **Number of triple ACKs received**

We calculate the first quartile (25% percentile), median (50% percentile), third quartile (75% percentile), the average regarding all the test replications, statistics about Min, Max values and Standard Deviation of the elapsed time, the data transmission time, the throughput, the number of retransmitted packets, the number of DUPACKs and triple DUPACKS. Finally, in the test cases involving more than one TCP flow we classify them through distinguishing between the fastest and slowest flow on the basis of the elapsed time value.

All the presented metrics were used to evaluate TCP performance. As regard the UDP traffic we are not interested in a deep analysis, thus we only introduce a simple, but significant metric: the Number of Packet Loss or equivalently the Number of Packet Received. We calculate the first quartile, median, third quartile, the average regarding all the test replications, statistics about Min, Max values and Standard Deviation like in the UDP analysis.

4.5 Test Cases

The tests are divided into two major categories in order to study separately RED's performance in different condition of traffic. First we run tests only with TCP traffic. Afterwards we add UDP traffic and we work in QoS-enabled architecture. We describe here the exact features and parameters of the tests that are run and we comment on the results in the next section. Each scenario is tested with 20 replications in order to obtain more accurate information for statistics.

4.5.1 Tests with three competing TCP flows

In these test cases we analyse the performance of 3 TCP flows sent in the network described in section 4.2with different loads and several starting times. In the testbed network we have only one source and one destination for TCP traffic and two routers. The first router has only to forward the packets without any other specific function. In order to avoid packet drops in the first two links we have set the bandwidth of the first hop to 100Mbit/s and the buffer size of the first router to 4096 packets. It implements a simple Tail Drop discipline.

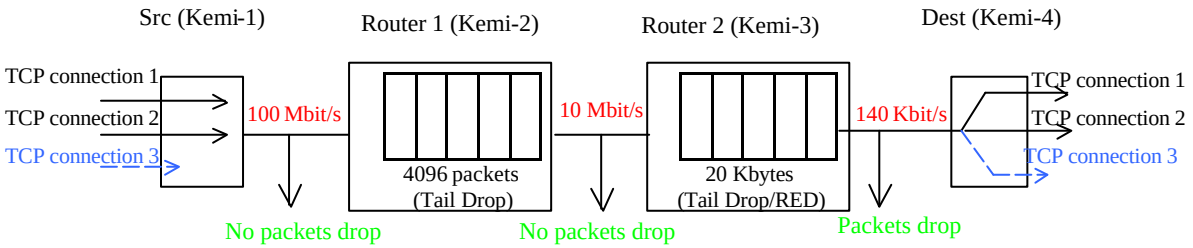


Figure 4-4. Workload model in the Tests run with three TCP competing flows.

A HTB qdisc, in the second router, allows setting the bandwidth of the third link to a much lower value such as 100/140Kbit/s, in order to simulate a bottleneck on the last hop. This is realized following the method presented in the workload Section 4.3. We have created two classes and set the rate to 100Kbit/s and ceil to 140Kbit/s in both. Then we have added a filter whose function is to enqueue all the TCP traffic received in one class and the non-TCP traffic in the other. This means that being this first set of experiments without UDP traffic the effective bandwidth offered to the TCP class is of 140Kbit/s, as specified by the ceil parameter. To the TCP class created the router applies FIFO discipline on the first part of the experiments run and RED discipline in the second one. In both cases the router buffer size is 20000 bytes. The traffic sent in the described network is shown in Figure 4-4. The workload traffic is composed only by TCP flows. Two flows of 292Kbytes are sent simultaneously and the third flow of 46.7Kbytes is sent later. The traffic is generated using the TTCP tool [TTCP98], with the length and number of buffers specified in Table 4-3.

The delay of the third flow is varied in order to gain knowledge about the various problems it encounters to start and the level of performance it manages to reach. The tests run consider these delays for the third flow: 0secs (three simultaneous flows) 0.5secs, 1sec, 2secs, 3secs, 4secs, 4.5secs, 5secs, 5.5secs, 6secs, 6.5secs and 7secs. This set of delays permit to

analyse which treatment the third flow receives when it is started during the Slow Start phase or during the Fast Retransmit phase or the Congestion Avoidance of one or both initial connections. The main aim of this way of proceeding is to compare the performance, and especially the fairness issue, achieved when the Tail Drop or RED discipline is deployed.

Flows	Source	Destination	Start time	Bandwidth (Kbit/s)	Data sent (bytes)	Length of bufs (bytes)	Number of bufs
1 TCP	Kemi-1	Kemi-4	Simultaneously	140	292000	5840	50
2TCP	Kemi-1	Kemi-4	Simultaneously		292000	5840	50
3 TCP	Kemi-1	Kemi-4	Delayed		46720	5840	8

Table 4-3. Traffic characteristics in the Tests run with 3 TCP competing flows.

Finally we report the RED parameters we have chosen for this set of experiments in Table 4-1. They are quite near to the default values. The min_{th} parameter is set to the maximum suggested, 5 times avpkt, and max_{th} is set to 3.6 times min_{th}. The max_{th} parameter is set quite high in order to reach good performance in terms of throughput. Finally we have decided to use a burst parameter of higher value than the one suggested (12 instead of 9 calculated by the presented formula).

Parameter	Suggested Value	Value
buffer-size (bytes)	free	20000
min_{th} (bytes)	3 up to 5 packets or 2 up to 5 times *avpkt*	5000
max_{th} (bytes)	3 or 4 times min_{th}	18000
avpkt (bytes)	free	1000
burst (packets)	$\frac{min_{th} + min_{th} + max_{th}}{3*avpkt}$	12
max_p (float)	0.1	0.1

Table 4-1. The RED Parameters for the Test with 3 TCP competing flows.

4.5.2 Tests with TCP and UDP traffic with services differentiation

In this set of experiments we study the performance of one TCP flow competing with UDP traffic. We have adopted two different kinds of network. Actually the testbed network is the

same as in the previous tests, but the configuration of the second router differs. In the former the router maintains only one queue to serve the incoming traffic and treats all the flows with equal priority. In the latter we introduce a simple QoS-enabled architecture. Through the Queueing discipline mechanisms we are able to give preferential treatment to UDP packets for example by enqueueing them in a separate class of service with a different bandwidth available.

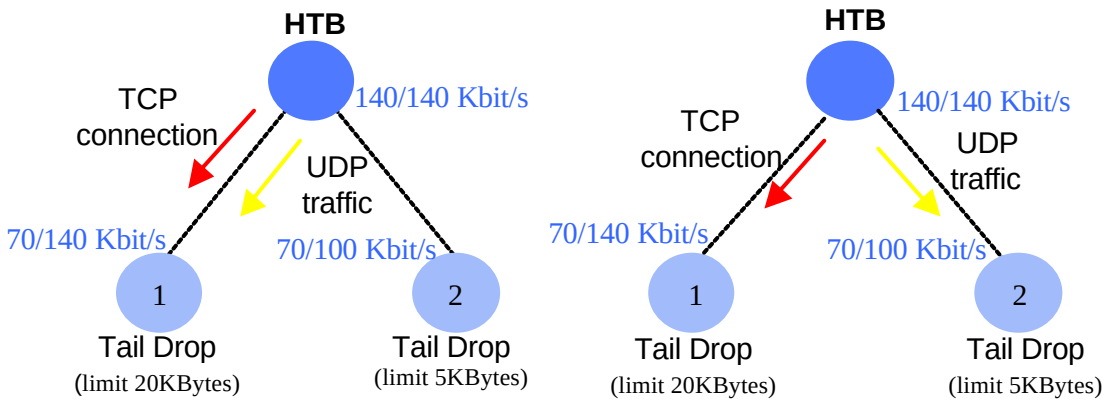


Figure 4-5. Class hierarchy of two HTB configurations.

The key element to understand these tests is the HTB topology implemented [Dev02, HMVV02]. We have four different types of configuration, two for RED experiments and two for Tail Drop ones. Those referring to Tail Drop implementations are schematised in Figure 4-5. The rate and ceil parameters are indicated and the dimension of the Tail Drop buffer in the router too. Shortly we have a hierarchy of two classes of equal priorities. In the first HTB model the TCP and UDP traffics are enqueued by the filters in the class with highest ceil and largest buffer size (class 1). In the second one the UDP traffic and TCP traffic are separated and sent to two distinct classes. In the RED configurations all the classes implement RED instead of Tail Drop.

Parameter	Class 1	Class 2
buffer-size (bytes)	20000	5000
min_{th} (bytes)	5000	800
max_{th} (bytes)	18000	3500
avpkt (bytes)	1000	150
burst (packets)	12	12
max_p (float)	0.1	0.1

Table 4-4. The RED parameters for the HTB configurations.

In addition, Table 4-4 reports the specific RED parameters used. As regards the RED parameters used in class 1, they are the same as in the previous tests run, while for the class with smaller buffer (class 2) they are calculated on the basis of the guidelines defined in Section 4.1.1

The experiments are organized in three different test cases depending on the workload model used. The combination of TCP and UDP traffics adopted is shown in Table 4-5. Note that the packet size indicates the UDP packet payload size for UDP flows and the buffer's length of TCP bulk transfer. This parameter is fixed to the same value in all the scenarios for both protocol: the MSS size for the TCP traffic, as in the previous tests, and 132 Bytes for the UDP one (104 Bytes represent the payload). The rate of UDP traffic is roughly 53 Kbit/s for all the test cases, while the duration changes. Note that in this way we guarantee that when we adopt the second HTB configuration, with separate classes for UDP and TCP traffic, the CBR flow finds always enough bandwidth (70 Kbit/s) to complete the transfer without any packet loss. Moreover the CBR's duration varies in order to last more than the TCP connection in the first two test cases and to have a shorter duration in the last type of experiment.

Workload Model	Flows	Source	Dest	Start time	Data sent (Kbytes)	Packet size (bytes)	Duration (sec)
1	TCP	Kemi-1	Kemi-4	0	391,3	5840	-
	UDP	Kemi-1	Kemi-4	after 0, 1, 2, 3 seconds	264,1	132 (104)	40
2	TCP	Kemi-1	Kemi-4	0	46,7	5840	-
	UDP	Kemi-1	Kemi-4	after 0, 0.5, 1, 1.5, 2 seconds	39,7	132 (104)	6
3	TCP	Kemi-1	Kemi-4	0	391,3	5840	-
	UDP	Kemi-1	Kemi-4	after 0, 2, 7, 12, 16, 18 seconds	33,1	132 (104)	5

Table 4-5. The workload models for the Test with TCP and UDP traffics.

According to the HTB discipline when the first type of configuration is used, both with Tail Drop and RED, the whole traffic is directed to class 1 and the available bandwidth is 140Kbit/s, since the ceil parameter enables to borrow bandwidth from the empty class 2. When the second HTB topology is employed the available bandwidth for both traffics is 70 Kbit/s. Nevertheless when the UDP starts later the TCP traffic will get all the available bandwidth of 140Kbit/s till the CBR flow starts and decreases this amount at least of the necessary

bandwidth to transfer the specified amount of data. When the CBR transfer is complete, the TCP flow, if it still needs, will borrow bandwidth again.

Before proceeding to the test analysis we can observe that, generally speaking, a single set of tests using the same testing methods, testbed in our case, is not enough to conclude that our enhancements and it must be widely implemented and the TCP changed. More simulation and emulation are needed to confirm our thesis.

5 Test Results and Analysis

This section presents the results of the tests that are outlined in Section 4.5. The tests are analysed and the reasoning for the benefits or drawbacks introduced by the use of RED algorithm instead of simple Tail Drop discipline is explained for the each Test Case. The main result is a comparative study and analysis of RED and Tail Drop.

5.1 Tests with three competing TCP flows

We compare the simulation results for the Tail Drop discipline and RED algorithm obtained using the workload model described in section 4.5.1. First we have studied the results of Tail Drop and we have evaluated the behaviour of TCP traffic for different starting points of the last flow. We have identified the most interesting scenarios. We have repeated the selected tests with the same specifications and conditions under RED. Then we have compared the performance of Tail Drop and RED for these critical points. It is not necessary to run the RED experiments for all the starting points considered with Tail Drop in order to make a reliable comparison between the two disciplines. The test results are introduced by first reporting the results with the Tail Drop discipline and then the performance achieved when RED is employed.

5.1.1 Tests with Tail Drop

Table 5-1 shows for all the tests run, with 12 different starting points of the third flow, the results achieved by each connection in terms of Elapsed Time and Throughput. The statistics are calculated on the basis of 20 replications. The column entitled Number of Exceptions reports how many times the third flow's duration has been longer than that of the initial flows. We consider these cases as exceptions since they represent situations in which the third flow does not manage to start till the initial flows are close to tear down their connections. In most of these scenarios, as shown later, the interference of the third flow is minimal and the behaviour of the initial flows is comparable to the study of two single simultaneous TCP connections.

Statistics of the fastest flow (data sent=292 Kbytes)

Scenario	Elapsed Time (seconds)							Throughput (Bytes/s)	Num Excep	Elapsed Time' (seconds)			Throughput' (Bytes/s)
after:	Min	Max	25%Perc.	Median	75%Perc	Average	Stdev	Average		Min	Max	Median	Average
0 sec	29,45	36,74	34,03	35,85	36,38	33,15	2,53	8436,70	0	29,45	36,74	35,85	8436,70
0,5 sec	29,71	36,58	33,98	35,06	35,72	33,83	1,71	8160,10	1	29,71	36,58	35,11	8428,21
1 sec	25,08	36,62	33,12	34,66	34,99	33,50	2,62	8321,80	0	25,08	36,62	34,66	8321,80
2 sec	28,21	36,53	31,02	32,93	35,05	32,02	2,67	8637,30	4	28,21	36,53	33,60	8731,75
3 secs	28,77	36,52	31,05	34,11	34,68	32,16	2,53	8593,95	0	28,77	36,52	34,11	8593,95
4 secs	29,21	36,71	32,68	34,70	35,84	32,97	2,28	8375,65	2	29,21	36,71	35,00	8556,72
4,5 sec	31,31	36,19	33,11	34,02	35,28	33,20	1,31	8288,55	1	31,31	36,19	34,20	8556,42
5 sec	24,11	36,56	33,55	34,27	35,72	32,77	2,86	8456,00	1	30,01	36,56	34,34	8513,63
5,5 sec	28,63	36,61	32,05	33,17	35,99	32,61	2,64	8513,65	0	28,63	36,61	33,17	8513,65
6 sec	25,26	35,60	29,59	32,82	34,61	31,26	2,81	8861,65	1	29,29	35,60	32,92	9011,53
6,5 sec	28,51	36,67	30,95	33,97	35,55	32,25	2,67	8571,45	0	28,51	36,67	33,97	8571,45
7 sec	28,26	36,68	32,31	34,05	35,00	33,00	2,20	8392,75	3	28,26	36,68	34,66	8611,35

Statistics of the slowest flow (data sent=292Kbytes)

Scenario	Elapsed Time (seconds)							Throughput (Bytes/s)	Num Excep	Elapsed Time' (seconds)			Throughput' (Bytes/s)
after:	Min	Max	25%Perc.	Median	75%Perc	Average	Stdev	Average		Min	Max	Median	Average
0 sec	36,55	38,18	36,59	36,67	36,70	36,45	0,44	8030,60	0	36,55	38,18	36,67	8030,60
0,5 sec	36,55	37,72	36,58	36,65	36,67	36,66	0,33	7965,85	1	36,55	37,72	36,64	7946,47
1 sec	36,55	38,56	36,58	36,66	36,69	36,77	0,48	7942,20	0	36,55	38,56	36,66	7942,20
2 sec	33,95	36,84	36,59	36,62	36,64	36,24	0,60	8066,80	4	36,58	36,84	36,62	7970,25
3 secs	35,04	36,69	36,57	36,65	36,66	36,43	0,36	8018,85	0	35,04	36,69	36,65	8018,85
4 secs	33,93	37,39	36,55	36,61	36,67	36,42	0,88	8022,25	2	33,94	37,39	36,63	7991,67
4,5 sec	33,82	36,82	36,64	36,66	36,68	36,30	0,66	8050,35	1	35,70	36,82	36,66	7975,32
5 sec	34,37	37,42	36,56	36,58	36,63	36,37	0,54	8034,35	1	36,53	37,42	36,58	7971,95
5,5 sec	36,54	37,30	36,59	36,64	36,66	36,37	0,25	8048,35	0	36,54	37,30	36,64	8048,35
6 sec	33,94	36,83	36,58	36,63	36,66	36,02	0,99	8117,20	1	33,94	36,69	36,63	8070,84
6,5 sec	36,56	37,96	36,61	36,65	36,68	36,74	0,44	7948,40	0	36,56	37,96	36,65	7948,40
7 sec	33,93	37,90	36,59	36,66	36,69	36,40	1,11	8029,65	3	36,57	37,90	36,67	7920,29

Statistics of the third flow (data sent=46,7Kbytes)

Scenario	Elapsed Time (seconds)							Throughput (Bytes/s)	Num Excep	Elapsed Time' (seconds)			Throughput' (Bytes/s)
after:	Min	Max	25%Perc.	Median	75%Perc	Average	Stdev	Average		Min'	Max'	Median'	
0 sec	6,43	19,43	9,91	12,53	14,33	12,65	6,42	3997,90	0	6,43	19,43	12,53	3997,90
0,5 sec	10,99	36,14	11,93	14,25	19,84	16,99	8,17	3076,90	1	10,99	36,14	14,25	3170,79
1 sec	10,76	32,69	12,98	14,56	17,28	16,41	7,21	3105,35	0	10,76	32,69	14,56	3105,35
2 sec	10,39	39,66	17,88	23,35	26,42	22,99	7,93	2267,25	4	10,39	31,34	21,78	2475,88
3 secs	11,89	33,66	13,77	15,60	20,26	18,11	7,79	2834,35	0	11,89	33,66	15,60	2834,35
4 secs	11,42	37,46	14,47	15,65	20,08	19,13	9,01	2777,50	2	11,42	37,46	15,37	2851,72
4,5 sec	14,18	45,71	15,00	16,89	21,86	19,62	8,49	2603,85	1	14,18	32,18	16,26	2687,11
5 sec	7,50	31,83	11,06	14,44	17,32	14,94	7,58	3576,30	1	7,50	24,89	14,43	3687,26
5,5 sec	9,92	25,99	11,04	12,03	14,14	13,89	6,93	3645,45	0	9,92	25,99	12,03	3645,45
6 sec	9,33	30,58	11,06	13,19	18,26	16,52	8,58	3307,40	1	9,33	30,58	12,60	3394,74
6,5 sec	10,27	26,39	11,45	12,09	16,22	14,38	6,71	3493,05	0	10,27	26,39	12,09	3493,05
7 sec	9,50	34,46	12,27	14,12	19,72	17,39	8,93	3132,35	3	9,50	25,74	13,17	3439,94

Table 5-1. Statistics for the 20 replications of the TCP connections when Tail Drop is employed.

We also report the statistics regarding the maximum, minimum and median values calculated with the exclusion of these particular unfair situations. However, these anomalous replications affect only few scenarios and the variations introduced in the results mostly regard the third flow.

In the first two tables, presenting the results of the fastest and the slowest flows, we observe that the performance achieved is quite similar in all the scenarios and the variance is limited. Considering the 12 scenarios the values oscillate roughly around an average throughput of 8469,7 Bytes/s, corresponding to 48% of the entire bandwidth. The same is true for the statistics of the slowest flow and the average throughput realized is roughly 8023 Bytes/s, corresponding to 46% of the total bandwidth. On the other hand analysing the last table, referring to the third flow's statistics, we observe that its results are affected by a large variance as its starting point varies. The median elapsed time varies from 6.43 seconds up to 14.18 seconds and the average throughput from 2267.25 Bytes/s up to 3997.90 Bytes/s, corresponding respectively to 13% and 23% of the available bandwidth. Note that these estimated bandwidth utilizations are not directly comparable since the elapsed times of the connections differ. The last connection makes use of the bandwidth during a time that is one third of the elapsed time of the initial flows.

In spite of the variability of the third flow's results we can find an explanation to this behaviour. When the third flow's starting time is quite close to that of the simultaneous flows (corresponding to a delay between 0 and 4 seconds) the best chance for the third flow to reach high performance is represented by the case in which it is started simultaneously with the initial connections. In fact all the statistics regarding the elapsed time of the third flow present their minimum in this scenario, except the median and the 75% percentile values that are anyway quite close to the absolute minimum. This conclusion is quite obvious if we think of how the TCP protocol is implemented. The first phase of the Slow Start is in general the most aggressive part of the entire connection, in which the flows attempt to occupy most of the available bandwidth and in which the Congestion Window is easily increased to high values. If a connection is started later it will have to compete much more to earn a part of the bandwidth and as later it is much more damaged it is. But the table also shows an interesting result. After 4.5 seconds of delay the third flow manages still to reach good performance similar and sometimes better than the first experiment with three simultaneous connections. The minimum median elapsed time occurs for a delay of 5.5 seconds, approximately the time at which the Slow Start phases of the initial flows have ended. Moreover we point out another attractive scenario in which the performance reaches the worst level and presents large unfairness. In the scenario *after 2 seconds* the elapsed time reaches high values and consequently the throughput

greatly decreases. For this delay the average throughput realizes its minimum reaching the value of 2267,25 Bytes/s, corresponding roughly to 13% of the total bandwidth (140Kbit/s).

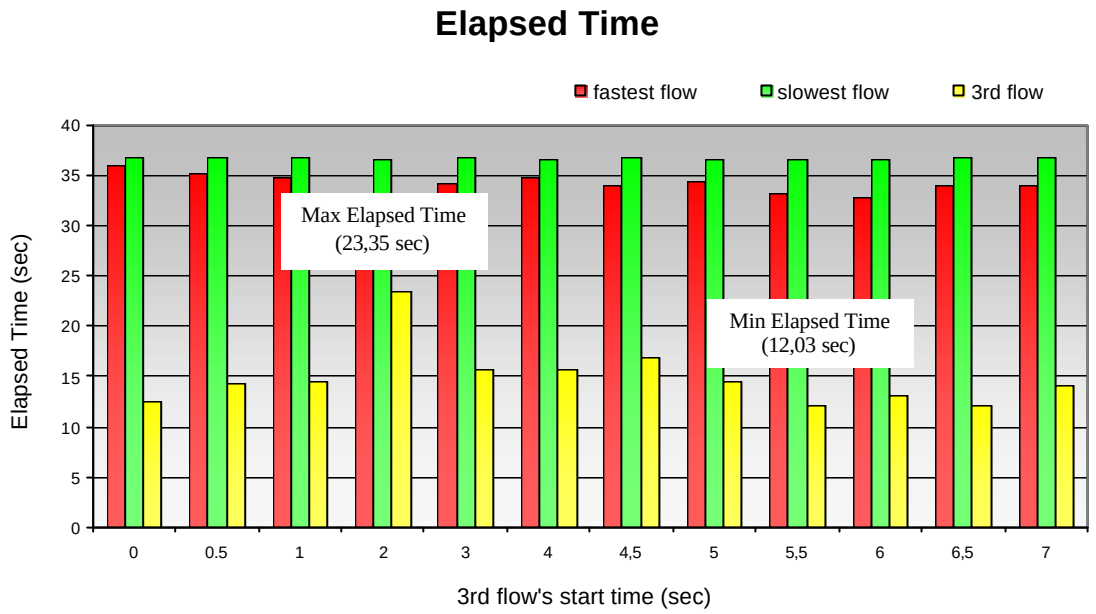


Figure 5-1. The Elapsed Time of the three competing flows for 12 third flow's start times (median values based on 20 replications).

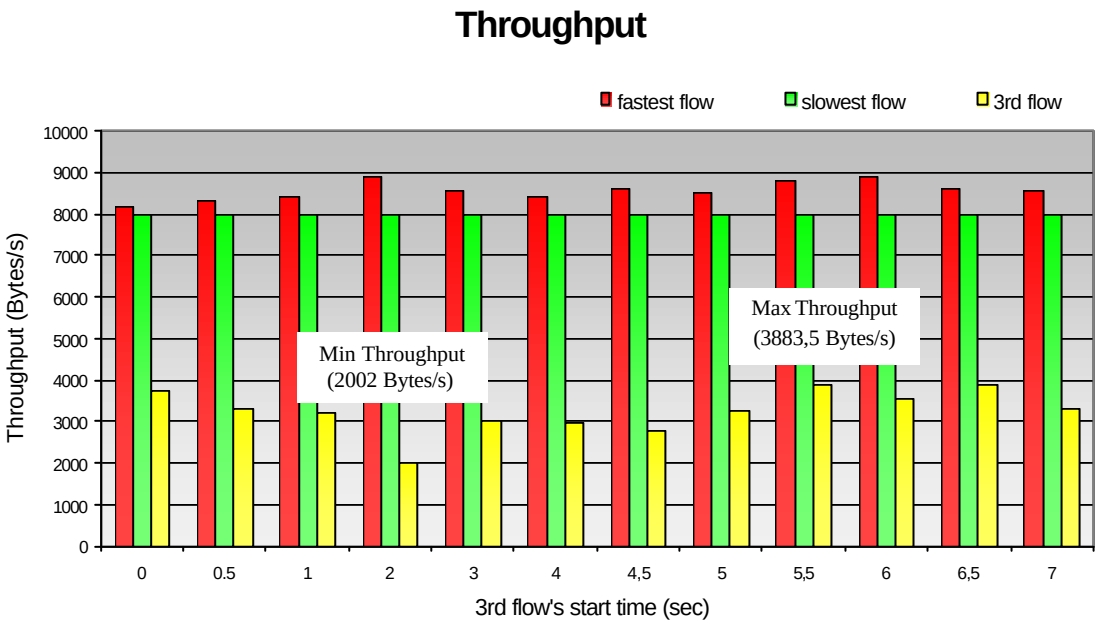


Figure 5-2. The Throughput of the three competing flows for 12 third flow's start times (median values based on 20 replications).

Figure 5-1 and Figure 5-2 outline the results described above graphically and show statistics related to the median values of the elapsed time and the throughput respectively. Note that the values reported for the three flows are not directly comparable. The initial flows always start at the same time and transfer the same data load, but the third one differs in the starting point and in the amount of data sent. Its traffic is 4/25 of that of the others. On one hand this implies that as regards the elapsed time realized and consequently the calculation of the throughput, the connection set-up and tear down phases have a largest impact on the short flow than on the longer ones. On the other hand considering that in our testbed the round trip time is quite small and consequently the 3-way handshake phase is short its influence is restricted anyway. The elapsed time experienced by the third flow increases slowly as the start time of the last flow grows. It reaches the maximum for the scenario *after 2 seconds* and afterwards it decreases and stabilizes, fluctuating around the initial values. The throughput graph follows a similar behaviour, reaching the minimum for 2 seconds and the maximum around 5.5 seconds. This tendency can be explained after a detailed analysis of the 20 replications that compose each scenario. In fact the third flow's statistics, as we have noticed before always show large variance and we need to better identify which the most recurring behaviour for different starting points are.

Analysis of the most interesting scenarios

It is quite obvious that when the third flow is started after a short time (from 0 up to 2 seconds) most of the replications will show the case of a third flow started when the first two connections are still in the Slow Start phase. Increasing the delay (from 2 up to 5 seconds) the third flow starts more often when both or one of the initial flows are close to end the Slow Start phase or to enter the Congestion Avoidance phase (after *cwnd* has reached the *ssthresh)* or Fast Retransmit (after the reception of three DUPACKs) or a new phase of Slow Start (after the first retransmitted packets for timeout expiration). After 5 seconds the last flow begins to transmit when at least one of the other flows have yet experienced some packet loss and congestion. In order to better understand this behaviour we go on to analyse into details of what happens in the three scenarios outlined above:

- the scenario *after 0 seconds;*
- the scenario *after 2 seconds*;
- the scenario *after 5.5 seconds*.

The graphs used in this section to study the flows performance show the activity on the connection as the TCP sender sees it, since they are taken from the sender side tcpdump. The

Y-axis shows the sequence number space of the connection, the X-axis gives the time that those segments were active (sent/acked). The black arrows represent the segments sent and the upper and lower limit of arrows gives the sequence number of the first and last byte in the segment. The green line tracks the ACKs returned by the receiver, the little green ticks below it indicate duplicate ACKs and the number 3 locates the triple duplicate ACKs. The yellow line is the receive window advertised by the receiver and the little yellow ticks above it mean that the receive window advertisement is the same as the previous advertisement. Finally the red letters R are places where a retransmission took place and the SYNs and FINs are respectively the syn and fin segment sent to open and close the connection. In general, for the longer flows, only the first 20 seconds of transmission are represented and for the third one the whole transfer. The details regarding the Test Cases examined are reported in Section 7.

Scenario after 0 seconds

In the scenario *after 0 seconds* (the statistics are reported in Table 5-2 and the details regarding the 20 replications in Table 8-1 in the Appendix) the three connections begin the transfer at the same time. This means that all the flows have in theory the same chance to achieve the best performance. Indeed the third flow never experiences long three-way handshake phases and never lasts more than the bigger flows.

		Data xmit time (sec)	Elapsed time (sec)	Throughput (Bytes/s)	Retx data packets	Retx Data bytes	Duplicate acks	Triple dupacks
fastest flow	Min	28,62	29,45	7947	4	5792	25	3
	Max	36,06	36,74	9914	8	11584	43	5
	25%Perc.	33,32	34,03	8026,25	4,75	6878	29	4
	Median	34,78	35,85	8144,50	5	7240	35	4
	75%Perc	35,26	36,38	8585,75	6	8194,75	39	5
	Average	32,16	33,15	8436,70	5	7136,95	32,20	4
	Stdev	2,48	2,53	682,83	1,07	1456,28	5,39	0,64
slowest flow	Min	35,34	36,55	7648	5	6897	28	4
	Max	38,17	38,18	7990	18	24073	44	6
	25%Perc.	35,63	36,59	7956,75	6	8200,25	32,25	5
	Median	35,71	36,67	7964	6	8688	35,50	5
	75%Perc	35,95	36,70	7980,25	7	10136	37	5
	Average	35,70	36,45	8030,60	6,8	9506,60	34,85	4,85
	Stdev	0,79	0,44	92,62	2,74	3637,52	4,18	0,55
third flow	Min	5,26	6,43	2405	0	0	1	0
	Max	18,25	19,43	7266	5	7240	14	2
	25%Perc.	8,83	9,91	3271,25	2	2896	5	1
	Median	10,95	12,53	3728,50	4	4729	7	1,5
	75%Perc	13,16	14,33	4716,50	4,25	5888,25	8	2
	Average	11,42	12,65	3997,90	3,35	4478,75	6,75	1,45
	Stdev	3,63	6,42	1498,06	1,54	2084,55	7,11	0,99

Table 5-2. Statistics for the scenario *after 0 seconds*, based on 20 replications.

This is the scenario in which we have observed the highest percentage of fairness cases between the fastest and slowest flow (around 50%). As regards the fairness between the three flows the third flow is not directly comparable to the performance of the initial ones, since the data transmitted are roughly less than a fifth of the biggest load. But we can evidence that only in the 20% of all replications the initial flows show fairness and the third flow's throughput reaches performance between the median and third quartile values. Moreover, consider Figure 5-3, in which we propose an example of this last behaviour. This represents one case of highest "fairness" when Tail Drop is employed. The throughput of the smallest flow is 4124 Bytes/s and that of the others are 8106 Bytes/s and 7962 Bytes/s. The Jari fairness index [Jai91] referred to the initial connections as 0.9999 for the Throughput and for the Elapsed Time too. The third flow's performance also in this "best" case is relatively poor since the throughput of the last flow is about the half of the fastest flow one and we are far from the ideal fairness between the three flows.

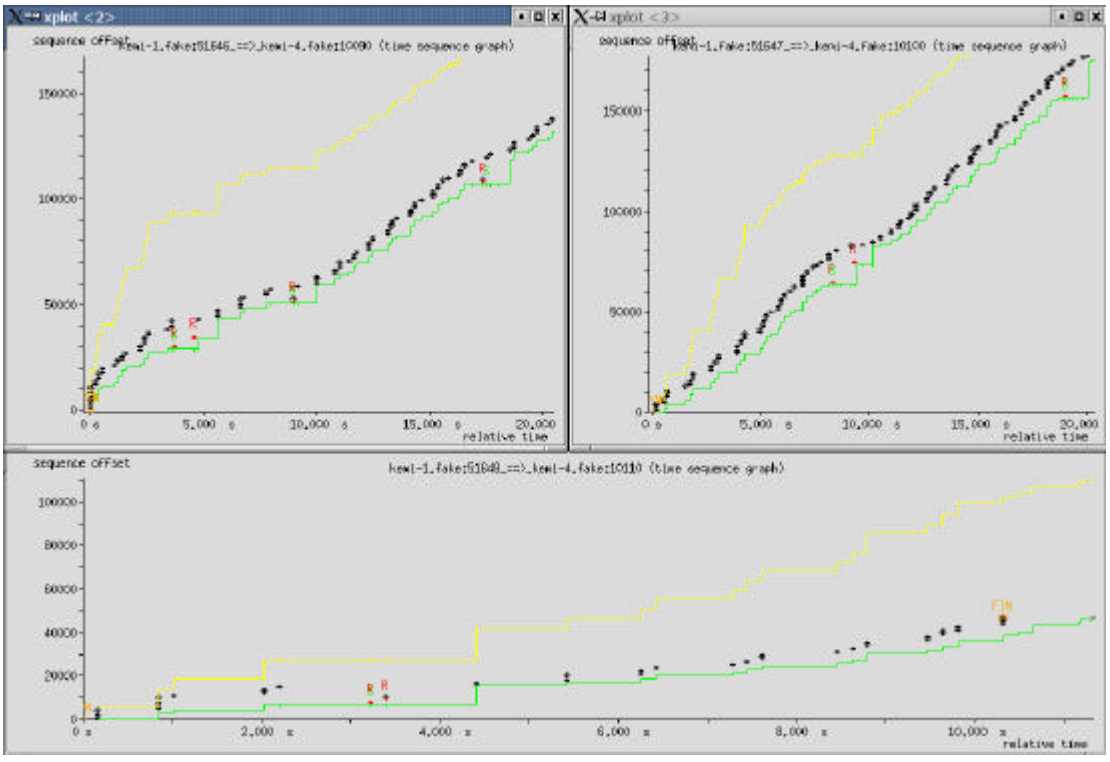


Figure 5-3. Example of fairness between the biggest flows and high performance of the third one in the scenario *after 0 seconds.*

The fact to have fairness between the biggest flows has no influence on the performance of the shortest connection. In fact, in this scenario, 6 replications out of 20 the smallest flow

manage to achieve the performance of the 75% percentile, but generally it is not favoured by the fairness or no-fairness of the other flows. For example we can consider the experiment in which the small flow carries out the maximum throughput. This is not only the maximum throughput achieved in the scenario *after 0 seconds*, but also the absolute maximum throughput out of 240 Tail Drop replications run. We will refer to this as the *optimum* achieved by Tail Drop. The relative Time Sequence (Figure 5-4) graph shows how the small flow starts to transmit soon and competes with the other to gain bandwidth. It manages to achieve a better performance than the other as confirmed by the number of packets retransmitted (0) and the number of duplicate ACKs (1) (Table 5-3).

	Data xmit time (sec)	*Elapsed time (sec)*	*Throughput (Bytes/s)*	*Retx data packets*	*Duplicate acks*	*Triple dupacks*
fastest	31.613	32.63397	8948	5	39	5
slowest	35.628	36.6409	7969	7	29	4
3rd flow	5.259	6.430064	7266	0	1	0

Table 5-3. The performance achieved in the case of third flow's maximum throughput (scenario *after 0 seconds*).

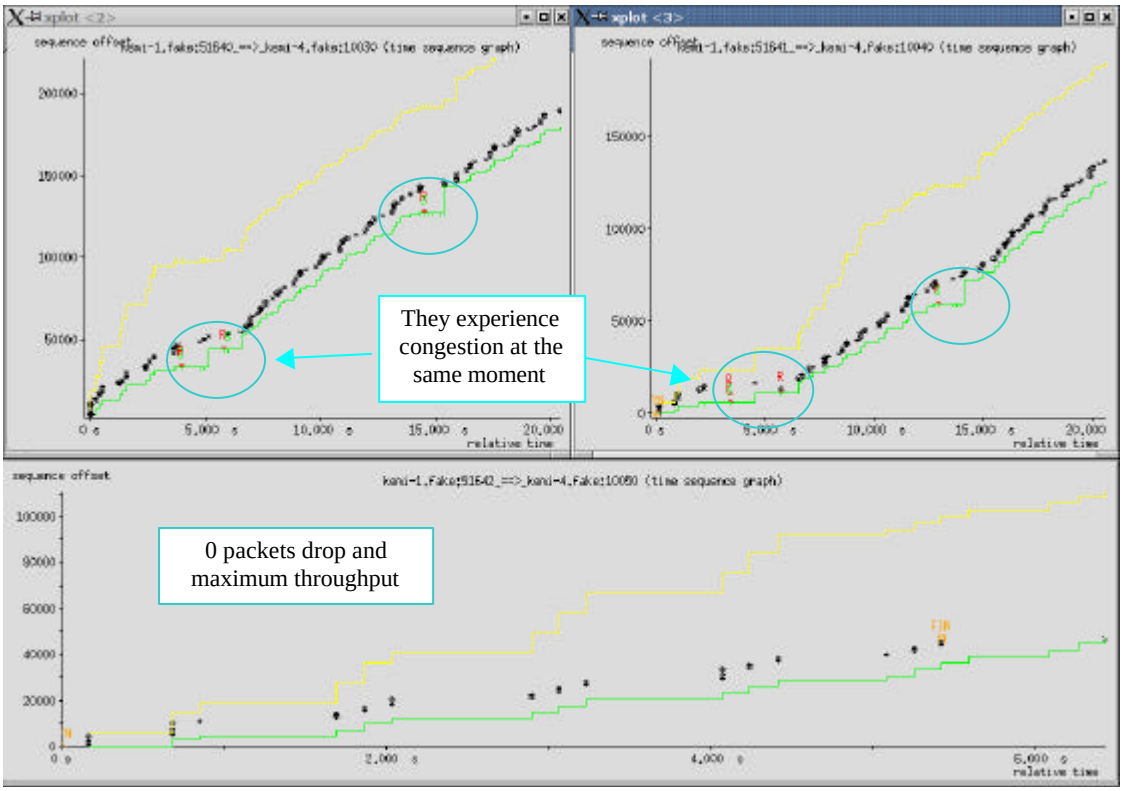


Figure 5-4. The case of Maximum Throughput of the third flow up to all the experiments run under Tail Drop (the third flow's *optimum* with Tail Drop).

Scenario after 2 seconds

The scenario *after 2 seconds (*in Section 8.1 we report the table of results) is the most critical set of experiments as regards the performance of the third flow. When the last connection starts after 2 seconds the other flows are generally in the first stage of the Slow Start phase and they are competing to gain the biggest share of bandwidth that is possible. The last flow usually manages to establish the connection quite soon, but then it can gain only a small part of the bandwidth realizing s long data transmit time. Even though the three-way handshake phase is short the data transmit time heavily influences the elapsed time and consequently the final throughput. This explains why in this scenario we have found the maximum number of cases in which the "shortest" flow lasts longer than the "longest" ones. This happens because the last flow is delayed in starting the data transfer and has to retransmit the first packet several times in order to have it acknowledged. Actually the data transmission begins when the initial flows are close to end their connection.

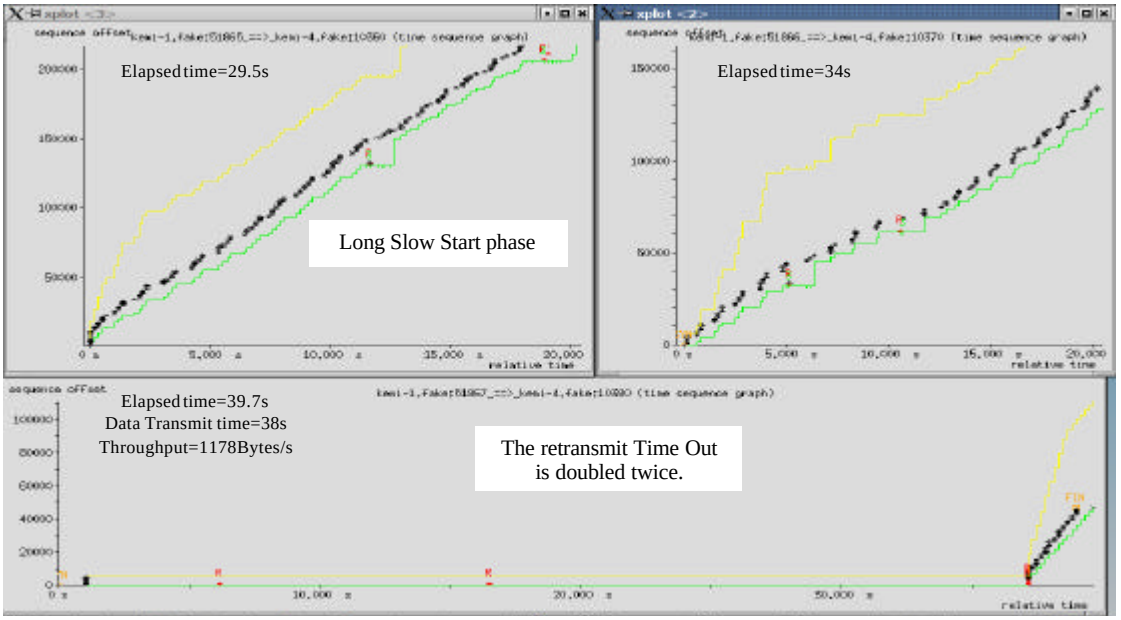


Figure 5-5. The minimum throughput of the third flow in the scenario *after 2 seconds.*

A clarifying example of this behaviour is shown in Figure 5-5, where the first packet sent by the third flow is retransmitted three times before being acknowledged. The exponential retransmit back-off rule typically doubles the retransmit timer value when a retransmitted packet is lost. This means that the loss of successive retransmissions results in very long delays. This is a clear instance of unfairness: the first flow manages to monopolize most of the bandwidth completing the transfer in a very short time, only 29.5 seconds and denies the last

connection the possibility to find place in the router buffer. The long Slow Start phase confirms its superiority. Furthermore it achieves a throughput of 9905 Bytes/sec, corresponding to more than half of the total bandwidth available. What we have observed is the so-called phenomenon of Lock-Out.

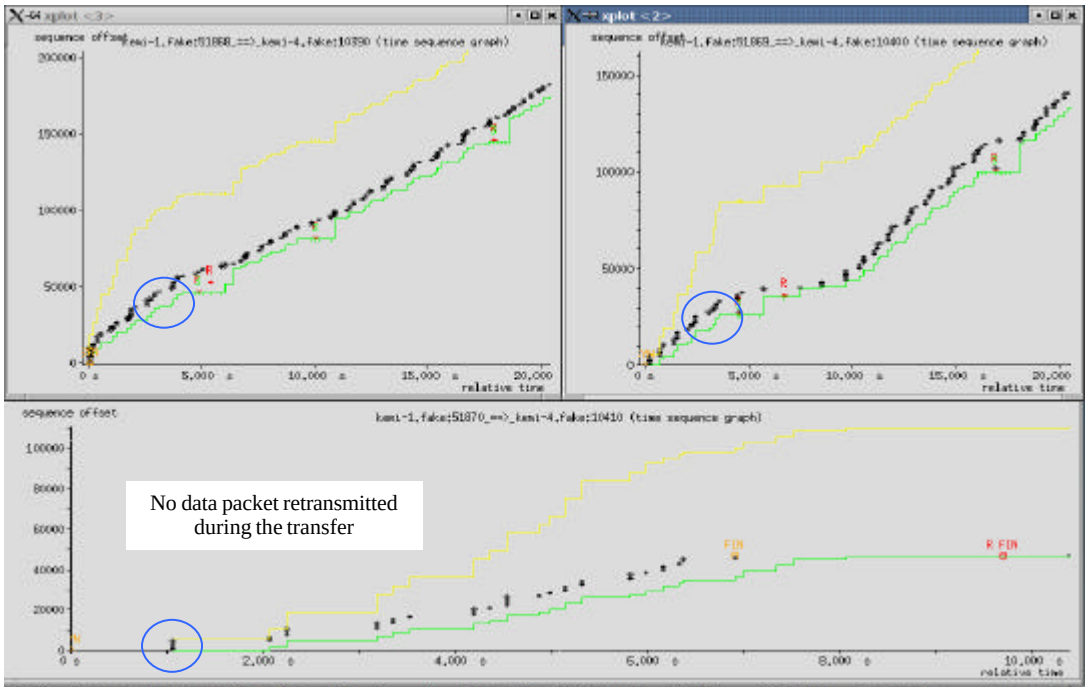


Figure 5-6. The maximum throughput for the 3rd flow in the scenario *after 2 seconds* (Throughput=4497 Bytes/s, Elapsed Time=10.4sec and Data Transmit Time=8.7sec).

The higher performance in this kind of scenario is reached in two different situations, as shown in Figure 5-6 and Figure 5-7. In the first case the third flow manages to gain bandwidth without experiencing any data packet loss during the connection. The three-way handshake phase is short and it sends the first packet when the other connections are in the second part of the Slow Start phase. Generally the performance observed when the third flow starts in the Slow Start phase of the initial flows is quite unsatisfactory, but in this case the flow starts at the end of the Slow Start when the connections are close to enter Fast Retransmit after the reception of three DUPACKs. This means that the network is congested and some packets have been dropped. The second instance shows a recurring situation found in Tail Drop experiments, but not so common in this scenario. We observe that the third flow, after a long three-way handshake phase, manages to start and realizes the minimum data transmit time. It takes advantage of the idle period of the network, due to the congestion experienced by the initial connections. In fact it starts to transmit when the other flows have received duplicate ACKs and

they have just retransmitted the packet loss. They are starting again to increase the congestion window and the conditions are similar to the ones at the beginning of the first Slow Start phase. In this scenario an analogous situation recurs only another time, but it is more frequent with bigger delay in the starting point of the third flow. This is the key aspect that justifies good results in a scenario such as that *after 5.5 seconds,* as explain later.

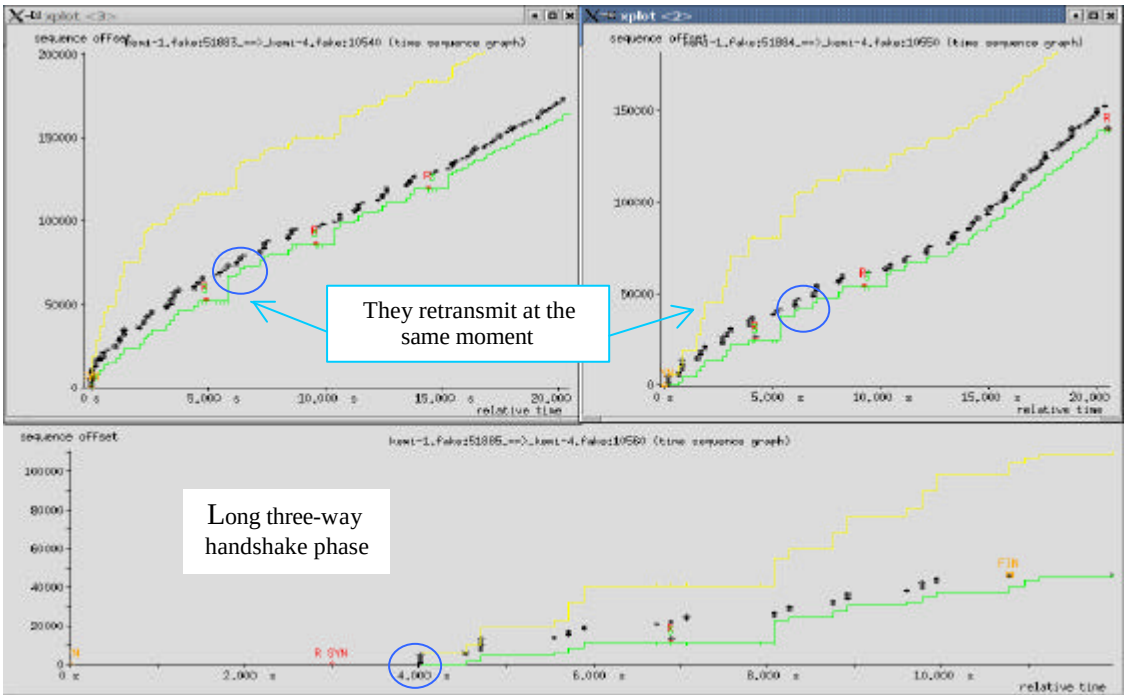


Figure 5-7. Minimum Data Transmit Time for the scenario *after 2 seconds* (Throughput=3907 Bytes/s, Elapsed Time=12 sec and Data Transmit Time=6.8 sec).

Moreover, taking a closer look to Figure 5-7 we observe a typical behaviour we have encountered often in this first set of experiments that is one of the major drawbacks of Tail Drop: the problem of synchronization. The TCP connections are frequently reducing their rates at the same moment, as is shown by the fact that the packets are often retransmitted in the same instant. This means that in most of the simulations two or all the connections are experiencing congestion at the same moment and this behaviour can heavily damage the final performance. At the same time the third flow may take advantages of this behaviour as we have seen in Figure 5-4, where the third flow realizes the *optimum* throughput under Tail Drop. The initial connections experience congestion at the same time and retransmit packets at the same moment. This means that they are reducing their transmission rates simultaneously and for this reason the third flow manages to start and finish its transmission in such a short time.

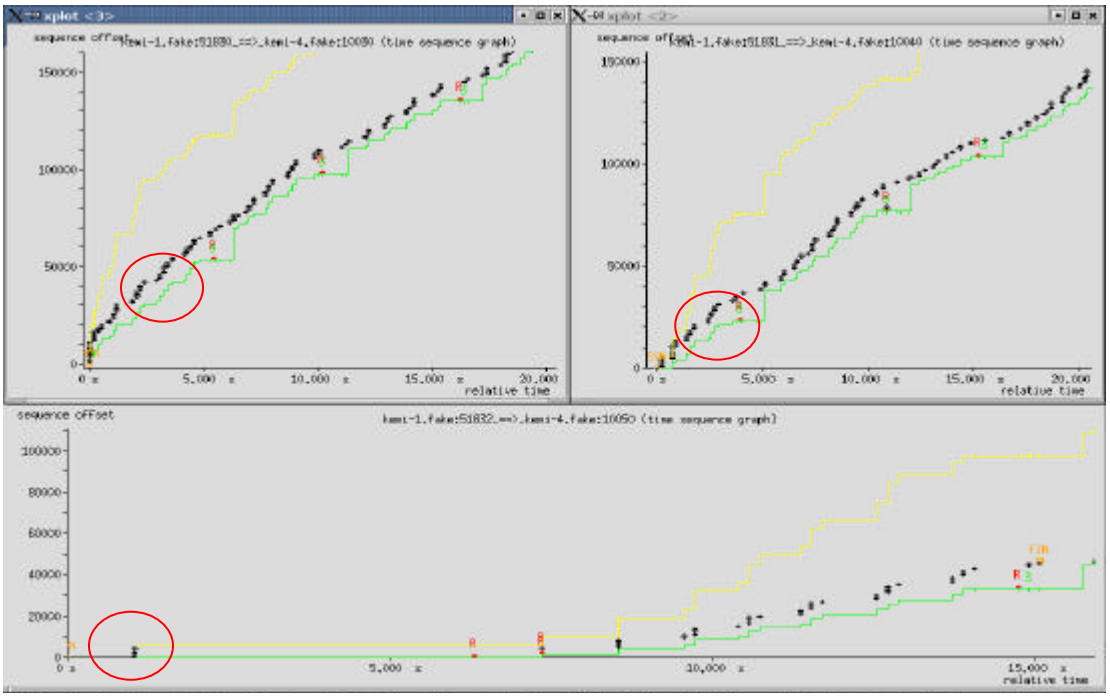


Figure 5-8. The third flow starts in the last part of the Slow Start phase of the initial flow in the scenario *after 2 seconds* (3rd flow: Throughput=2937Bytes/s, Elapsed Time=16sec).

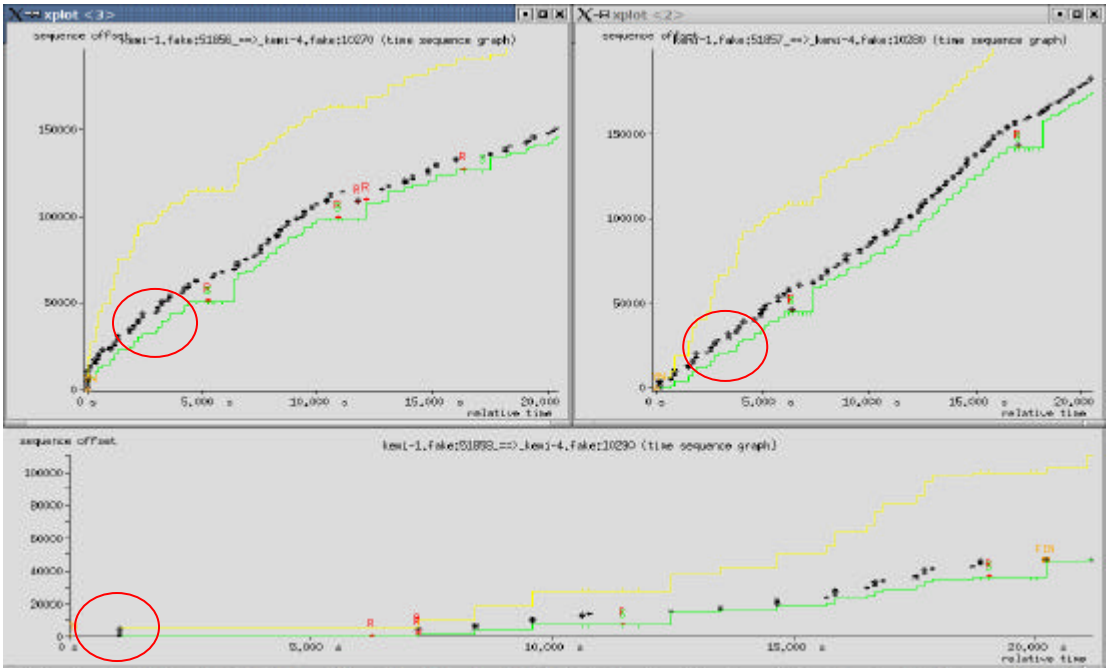


Figure 5-9. The third flow starts in the middle part of the Slow Start phase of the initial flow (3rd flow: Throughput=2206 Bytes/s, Elapsed Time=21.2sec).

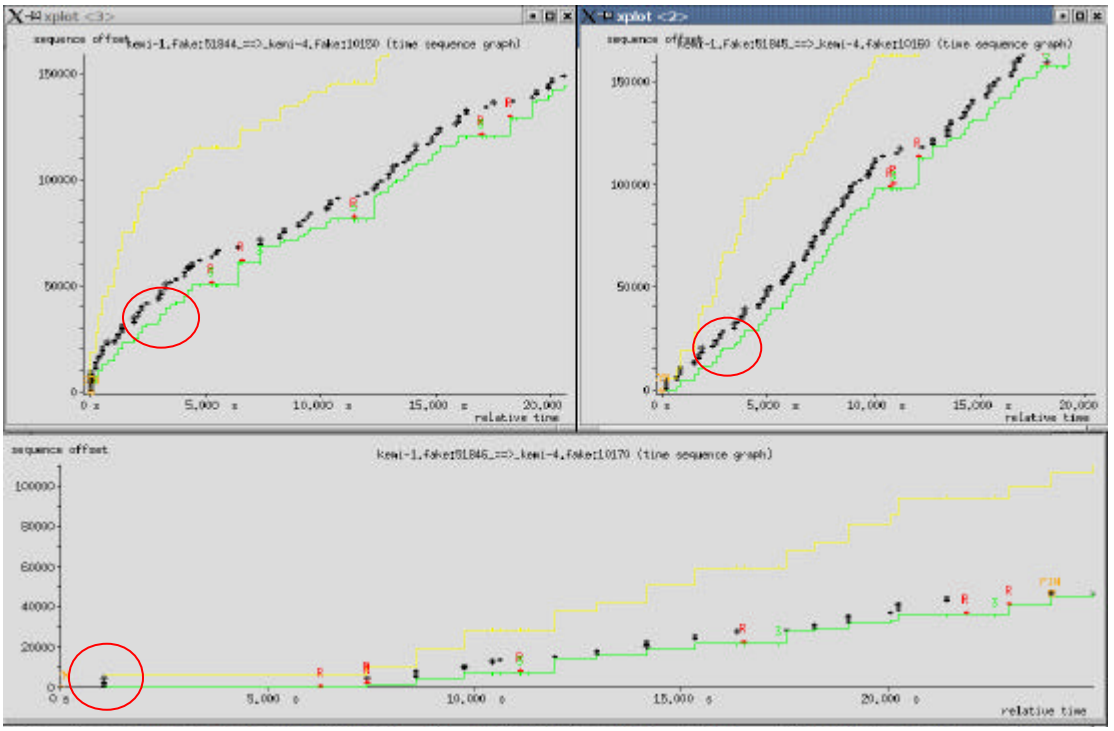


Figure 5-10. The third flow starts at the beginning of the Slow Start phase of the initial flow in the scenario *after 2 seconds* (3rd flow: Throughput=1872Bytes/s, Elapsed Time=25sec).

Finally we examine shortly three other cases to evidence how three connections started at the same moment and encountering the same conditions in the transfer of the two biggest flows, can reach quite different performance depending on the length of the Slow Start phase of one of the initial flow. We compare three examples (Figure 5-8-Figure 5-10) in which the third flow starts the transfer at the same moment and, after quite similar three-way handshake phases it encounters the same conditions regarding one of the biggest flows in transmissions. Indeed in all three cases the last flow establishes the connection 2 seconds later than the initial flows and it sends the first data packet after 1 second of connection. This instant corresponds to the third second of transmission of the initial flows as indicated by the red circles. The red circles in the left panel of each figure indicate that the starting point of the last flow falls in all the three situations in the second part of the Slow Start phase of one of the initial flows depicted. On the other hand, considering the right panel reporting the behaviour of the other initial flow we identify three different situations. In the first case the starting point coincides with the Fast Retransmit phase of the initial flow, after it has received three DUPACKs. In the second one it falls in the middle part of the Slow Start phase. Finally in the last one it corresponds to the beginning of the Slow Start phase. Studying the elapsed time and throughput values achieved in

each case we conclude that the third flow's performance decreases as the length of the initial flow's Slow Start increases. Moreover the third connection seems to be particularly favoured when it starts the transfer near to a period of congestion experienced by one of the initial flows. This is quite obvious if we think that when the network is congested the initial flow is receiving duplicated ACKs and it is not transmitting. The third flow can take advantage of this idle period to gain bandwidth. On the other hand, when one of the initial flows manages to realize a long Slow Start phase it means that few resources are currently available for the third flow.

Scenario after 5.5 seconds.

The previous discussion can help to understand what happens in the scenario *after 5.5 seconds* (for the tables of results see Section 88.1). We have noticed before that the performance of the third flow here shows a visible improvement, reaching values for Throughput and Elapsed Time sometimes bigger than in the scenario *after 0 seconds*. For a delay longer than 5.5 seconds till 7 seconds the performance became quite stable around a value lightly inferior to the early results. The study of all the 20 replications for this scenario shows a similar behaviour to what was observed in the scenario *after 0 seconds,* when the third flow was started during the Slow Start phase of the initial flows. In fact after a packet loss is detected by receiving three DUPACKs, the sender is subject to enter Fast Retransmit and after that a new Slow Start phase if *cwnd<ssthresh* or Congestion Avoidance mechanism if *cwnd>ssthresh*. If the packet loss is detected by timeout expiration the connection enters the Slow Start phase starting to retransmit from the packet lost. We have observed that in most of the replications the packet loss is generally detected by the reception of three DUPACKs and *cwnd* is usually inferior to *ssthresh.* Thus after Fast Retransmit a new phase of Slow Start is triggered. This implies that if the third flow is started in these critical points it encounters the same, or occasionally better, conditions of the early scenario, in which it was at beginning of the first Slow Start.

The three-way handshake phases are generally shorter and we have not found any "anomalous" situations: in all the replications the third flow has a shorter duration than both initial connections. In 7 cases out of 20 there is fairness between the initial flows and in 5 cases out of 20 there is fairness between the first two connections and the third one achieves optimal performance. Only in 4 cases the third connection experiences a long three-way handshake phase and the performance degrades seriously.

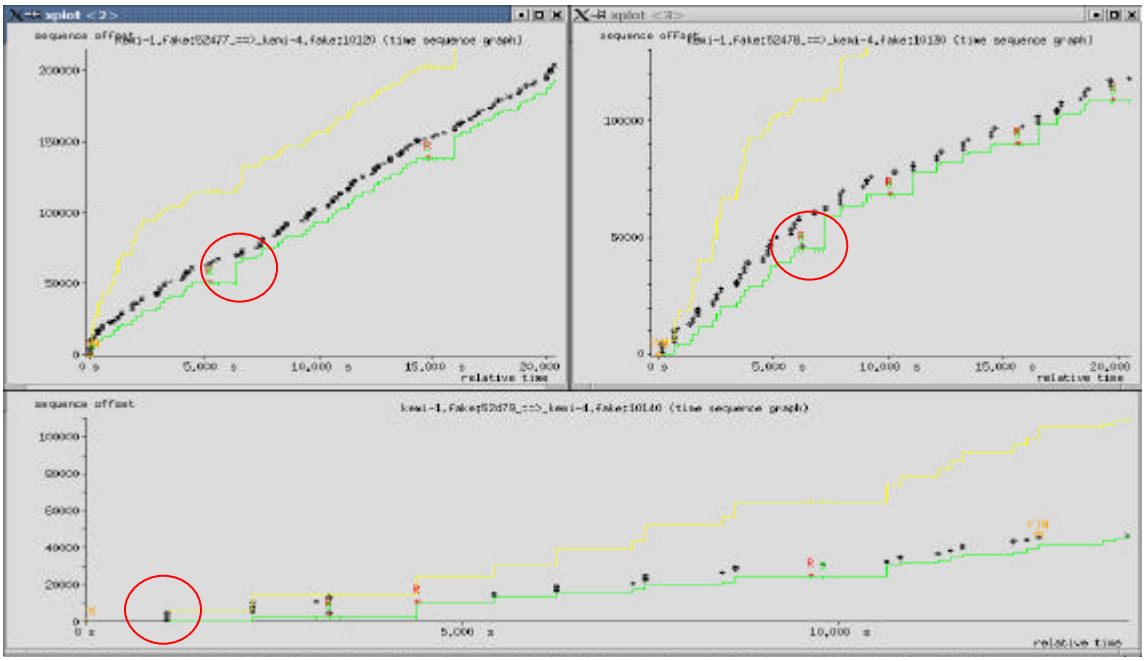


Figure 5-11. Example of great performance of the fastest flow (Throughput=9914 Bytes/s and Elapsed Time=29.5sec) to the detriment of the third one (Throughput=3370 Bytes/s and Elapsed Time=13.9sec).

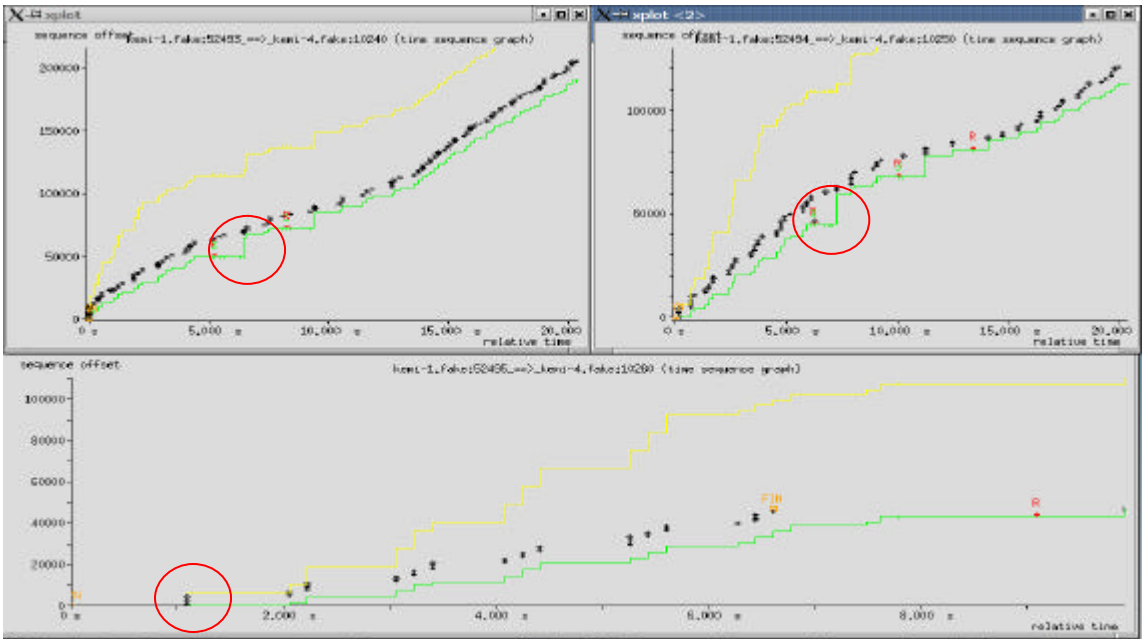


Figure 5-12. The third flow achieves the Max Throughput (4709 Bytes/s) and the performance of the fastest flow degrades (Throughput=8814 Bytes/s).

We report only two examples to show a case in which the fastest flow reaches the best performance (Figure 5-11) and, on the contrary, a case in which the third connection accomplishes the Max Throughput (Figure 5-12). In this second replication the third flow is prevailing on the initial ones as shown by the fact that the flow in the upper-left enters the second Slow Start phase and losses another packet quite soon while the third flow transmits till the end experiencing only one loss packet. Both the initial connections lose five packets during the entire transfer. In the replication of Figure 5-11 the fastest flow and the last one lose the same number of packets, even if the duration of the fastest flow is more than double of the third one.

5.1.2 Comparison of Tail Drop and RED performance

The RED tests were run only for the five scenarios of most interest: after 0 sec, 2 sec, 4 sec, 5 sec and 5.5 sec with the parameters described in Section 4.1.1. Figure 5-13 shows some statistics about the performance of the third flow, while Figure 5-14 refers to the Elapsed Time and the Number of Retransmitted Packets of the initial flows. The median value for all the statistics presented is plotted and the Min, Max, 25%Perc and 75%Perc as regards the elapsed time of the third flow. They are based on 20 replications.

First of all we observe that in most of the graphs reported the statistics of RED show small variability as the third flow's starting point changes, since moving on the x-axis they are almost constant. On the contrary, when Tail drop is employed the variation range is generally wide. Moreover, the results reached by RED are not only more stable, but also indicate higher performance. The improvements introduced regard especially the third connection, while the initial flows' results are not far from the Tail Drop's one. This fact means that RED provides higher fairness between the three flows sent and much more guarantee to the third flow for achieving good performance. Furthermore RED assures to the last flow sent performance less dependent on its starting point than Tail Drop does.

RED outperforms Tail Drop's performance of the third flow as regards the Elapsed Time, the Data Transmit Time and consequently the Throughput. Especially in the scenario *after 2 seconds* we note the improvements carried by the introduction of RED. Now the third flow is only lightly degraded by the initial flows' aggressive Slow Start phase, while the Tail Drop performance examined was very poor and unfair.

We first analyse the Elapsed Time graphs of the third flow. We have plotted the behaviour of five different statistics in order to define how large is the variation range.

Considering the range delimitated by the 25% and 75% Percentiles we notice that it is quite large in the case of Tail Drop, roughly 5.6 seconds, and more restricted with RED, around 3.6 seconds. Tail Drop is subject to the maximum variability (8.5 seconds) in the scenario *after 2 seconds* while as regards RED the major variance (4.4 seconds) is reached in the scenario *after 4 seconds*. The behaviour of the Minimum does not move so far away from the 25% Percentile while the Maximum is more irregular. Especially in the Tail Drop experiments it reaches high values as confirmed by the number of abnormal cases studied in the previous section, in which the third flow lasts longer than the initial connections that manage to achieve optimal results. In the RED experiments we have only one situation, in the scenario *after 4 seconds*, in which the elapsed time of the last flow reaches a high value (33 seconds), but the initial flows are longer than it (36.1 and 36.6 seconds) anyway.

The three-way handshake graph shows an interesting result: in the first two scenarios the values realized by the two queueing disciplines are quite the same, while around 4 seconds RED outperforms Tail Drop and the results greatly differ (the median values are 0.57 seconds for RED and 3.81 seconds for Tail Drop). Examining now the Data Transmit graph we see how the data transfer durations for both disciplines are generally similar, but they differ significantly around 2 seconds, where the transmission time for Tail Drop is more than the double of that realized with RED (9.88 seconds for RED and 20.82 seconds for Tail Drop). Thus comparing the Three-way Handshake and the Data Transmission Time behaviours we observe an interesting aspect when the Tail Drop discipline is deployed. The most critical time interval for the third connection is the range around 4 seconds (referred to the time scale of the initial flows), where the performance of the third flow is seriously damaged by the longest connections. In fact, when the third flow is started after 2 seconds the connection set-up phase is quite short (about 1.05 seconds), but the data transmission time is quite long (about 20.8 seconds). This indicates that after the connection has been established it is blocked by the biggest connections when they are around the 4^{th} second of transmission. However, when the last flow starts after 4 seconds it is more damaged in the three-way handshake (about 3.8 seconds) than in the data transmission phase (roughly 10.6 seconds). In both scenarios the critical point is around 4 seconds, which corresponds to the Slow Start's second phase of one or both initial flows. This was the conclusion we arrived at in the first session and it finds confirmation here: the most crucial interval for the third flow's performance falls in the second part of the Slow Start phase of the initial flows. Obviously the resulting elapsed time is more influenced by the data transmission time rather than the connection set-up phase and this explains why the minimal performance is for the scenario *after 2 seconds*. The Throughput behaviour and the number of Retransmitted Packets confirm this tendency: the worst

performance is for the scenario *after 2 seconds*, where the two graphs reach a minimum and a maximum respectively.

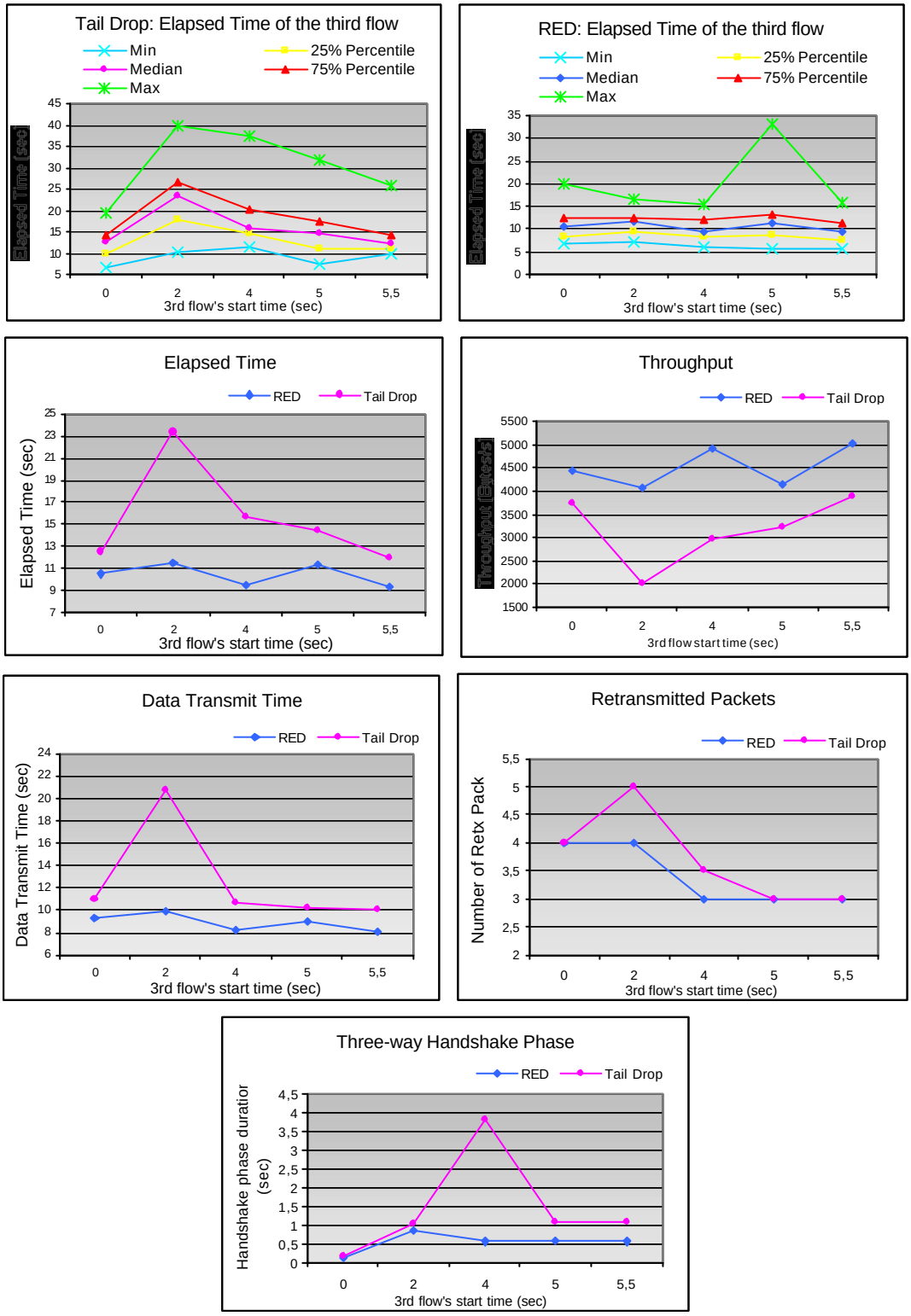


Figure 5-13. Comparison between Tail Drop and RED perfomance for the third flow (data sent=46.7Kbytes).

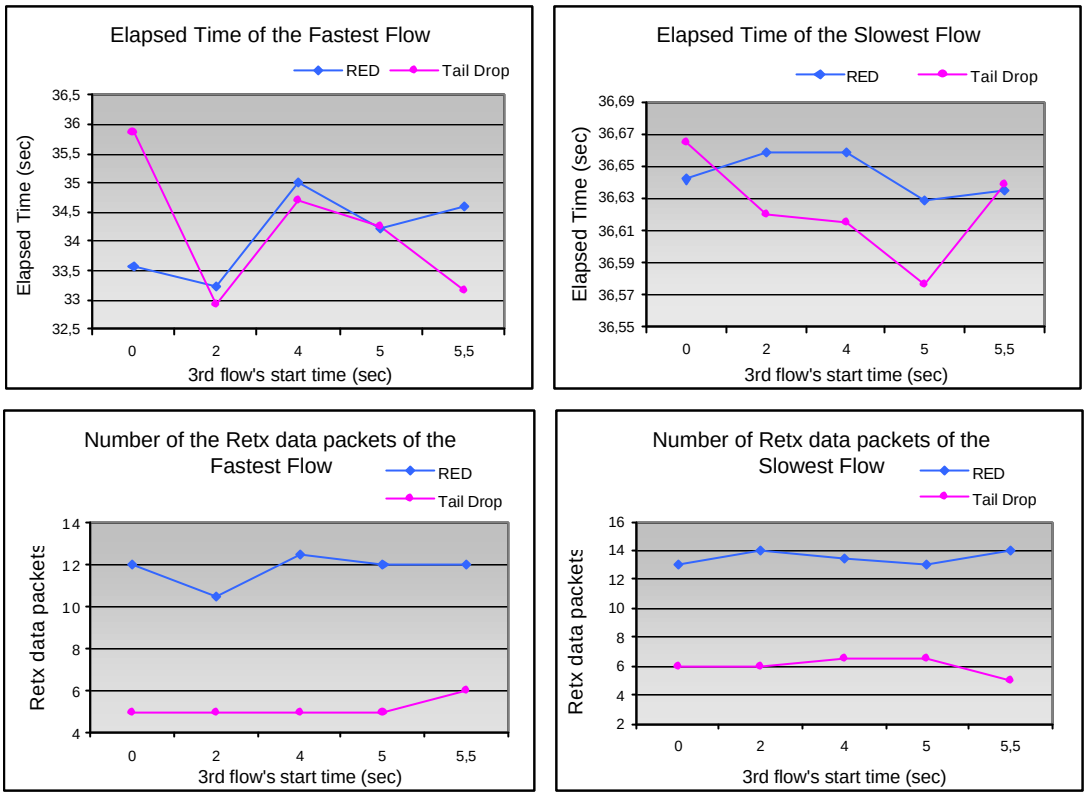


Figure 5-14. The Elapsed Time and the Numbe r of Retransmitted packets of the fastest and slowest flow (data sent=292Kbytes).

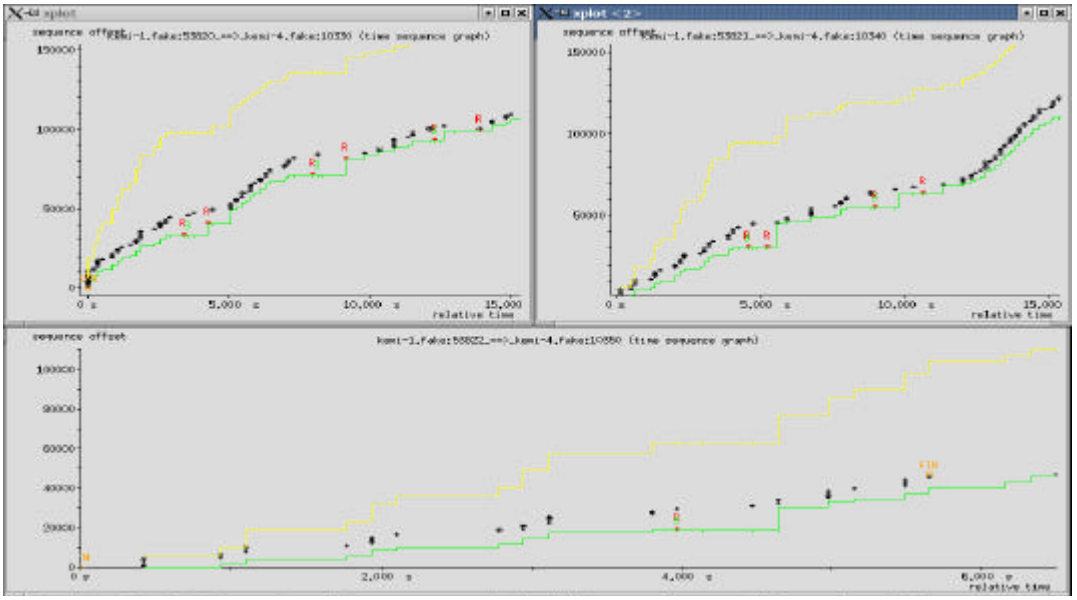


Figure 5-15. Example of RED fairness in the scenario *after 5.5 seconds.* The throughput of the fastest and the slowest flows are 8062 Bytes/s and 7988 Bytes/s respectively. The throughput of the third flow is 7190 Bytes/s.

Statistics of the fastest flow (data sent=292Kbytes)

Scenario	Elapsed Time (seconds)							Throughput (Bytes/s)	Num Excep
after:	Min	Max	25%Perc.	Median	75%Perc	Average	Stdev	Average	
0 sec	28,13	36,55	32,19	33,56	34,75	32,26	2,24	8596,70	0
2 sec	29,18	36,48	31,28	33,22	34,61	31,96	2,30	8664,80	0
4 secs	30,99	36,64	32,86	35,01	36,07	33,08	1,85	8350,20	0
5 sec	31,58	36,57	33,19	34,23	34,99	33,21	1,48	8286,20	0
5,5 sec	21,57	36,88	31,46	34,60	35,97	31,92	3,66	8870,75	0

Statistics of the slowest flow (data sent=292Kbytes)

Scenario	Elapsed Time (seconds)							Throughput (Bytes/s)	Num Excep
after:	Min	Max	25%Perc.	Median	75%Perc	Average	Stdev	Average	
0 sec	36,52	38,29	36,58	36,64	36,67	36,57	0,43	7992,55	0
2 sec	36,53	37,49	36,60	36,66	36,72	36,47	0,27	8015,75	0
4 secs	36,55	37,34	36,60	36,66	36,71	36,51	0,21	8002,40	0
5 sec	36,55	37,43	36,58	36,63	36,67	36,57	0,23	7986,45	0
5,5 sec	36,54	37,53	36,57	36,63	36,73	36,77	0,34	7942,25	0

Statistics of the third flow (data sent=46,7Kbytes)

Scenario	Elapsed Time (seconds)							Throughput (Bytes/s)	Num Excep
after:	Min	Max	25%Perc.	Median	75%Perc	Average	Stdev	Average	
0 sec	6,94	20,00	8,36	10,51	12,55	11,12	6,40	4545,00	0
2 sec	7,30	16,54	9,49	11,51	12,42	11,38	6,20	4307,85	0
4 secs	6,07	15,53	8,47	9,52	12,02	9,85	6,46	5081,90	0
5 sec	5,52	33,02	8,74	11,34	13,13	12,19	8,22	4505,00	0
5,5 sec	5,70	15,86	7,67	9,37	11,48	9,78	6,75	5147,95	0

Table 5-4. Statistics of the Elapsed Time when the RED algorithm is employed (20 replications).

	Data xmit time (sec)	*Elapsed time (sec)*	*Throughput (Bytes/s)*	*Retx data packets*	*Duplicate acks*	*Triple dupacks*
fastest flow	33,988	34,658388	8425	14	46	8
slowest flow	35,8	36,635197	7970	14	47	7
3rd flow	4.56	5.518792	8466	1	8	1

Table 5-5. The statistics of the optimum in the RED tests (Max Throughput and min Elapsed Time).

When RED is employed as regards the third flow's performance, the disparity between the scenario *after 2 seconds* and that *after 4 seconds* is not so relevant as with Tail Drop. RED outperforms Tail Drop in all the scenarios and in particular in these critical ones. In Table 5-4 we have highlighted some values that have to be compared with the previous ones, presented in Table 5-1. Around 2 seconds the median elapsed time is 11.51 seconds for RED and 23.35 seconds for Tail Drop: the latter is more than the double of the former. These are also the

maximum median elapsed times. Both minimum median elapsed times fall in the scenario *after 5.5 seconds*: 9.37 seconds for RED ad 12.03 for Tail Drop. Even though they differ, now the gap is more narrow..

As regards the elapsed time trend of the initial flows the RED and Tail Drop's results do not disagree significantly. The slowest flow's behaviours for the two disciplines are quite close to each other and the maximum gap is roughly 0.5 seconds. However, the fastest flow's graphs vary over a wider range. The two disciplines differ at most by 2.3 seconds (in the case *after 0 seconds*), while in the central scenarios they are quite similar. Comparing the RED elapsed time graphs of the fastest flow and of the third flow, we notice that the behaviours are nearly complementary. When the fastest connection lasts more, correspondingly the third flow duration decreases. The better performance of the third flow achieved when RED is in use explains why in some cases the Tail Drop discipline allows to reach higher performance as regards the fastest flow. This is done to the detriment of the last connection.

Finally we note how the number of retransmitted packets of the third flow is generally slightly lower with RED than with Tail Drop: it decreases in the scenario after 2 and 4 seconds, but later the behaviours of the two disciplines are the same. This is due to the fact that RED attempts to keep the average queue size low in order to accommodate bursty traffic and transient congestion. On the other hand Tail Drop detects congestion only after a packet has been discarded and this occurs just when the queue router is full. Tail Drop allows queues to reach a full status and to persist in this steady state for long. RED does not wait to have a full queue to start discarding packets and consequently starts to drop packets before Tail Drop does, as demonstrated by the short Slow Start phases depicted in the RED graphs.

What is important to point out is not only the number of retransmitted packets, but also the ratio between the packets dropped by the initial flows and the third transfer. When RED is deployed the initial connections retransmit roughly the double number of packets compared to the Tail Drop case. That is why RED tends to discard packets from each connection in proportion to the transmission rate the flow has on the output link. Therefore the connection with the largest data rate will have the highest drop percentage among total discarded packets. In fact, the ratio between the initial flows' discarded packets and the third flow's varies from 2.5 up to 4.3 for RED and from 1.1 up to 1.9 for Tail Drop (they are calculated considering the median value of the number of retransmitted packets). Being the smallest flow $4/25^{th}$ of the other ones, we conclude that RED also from this point of view manages to serve the TCP traffic with more fairness that Tail Drop does.

5.1.3 RED Parameters

Parameter choice in RED seems to be hard since the inventors themselves periodically change their recommended RED parameters. The ones used in our experiments were chosen following some generic guidelines, but they do not pretend to be optimal and the best in all circumstances. We have run some additional tests varying some RED parameters and we have investigated how the performance could be improved over the Tail Drop's one.

Elapsed Time varying some RED parameters (min_{th}, max_{th}, *burst*) max_p=0.1; buffer size=20000; avpkt=1000					
RED Parameters / Scenario	**5000 18000 12**	**5000 18000 9**	**4000 16000 11**	**5000 15000 8**	**5000 15000 12**
after 2 sec	11.51	8,96	12,36	9,86	10,34
after 5.5 sec	9,37	9,91	9,91	11,20	9,43

Table 5-6. Additional tests oriented to study the tuning of RED parameters (Median value of the 3rd flow's elapsed time, based on 20 replications).

In Table 5-6 we report some additional tests run tuning the min_{th}, max_{th} and *burst* parameters and only for most interesting scenarios: after 2 and 5.5 seconds. The first column presents the results for our parameters. We see that a significant improvement, especially in terms of fairness, can be achieved by decreasing the burst parameter (second column). Moreover, also diminishing max_{th} the elapsed time decreases appreciably for the scenario after 2 seconds. This is reasonable since keeping the max_{th} low the buffer queue is also kept low and consequently it can easily absorb a new incoming flow (fourth and fifth columns).

We conclude that the RED parameters selected in our test are not the optimal ones at least for the third flow's performance and as regards the two scenarios investigated, but since our study is intended to carry out a comparison between the traditional Tail Drop queueing discipline and the RED, basing our conclusions on common RED parameters makes them more general and trustable.

5.1.4 Summary

A major drawback of Tail Drop is the phenomenon of Lock-Out. The Tail Drop mechanism allows a monopolization of queue resources by one or both initial flows, denying to the third connection the possibility to find place in the router buffer. RED avoids this unfair distribution

of the network resources as partly demonstrated by the absence of abnormal tests, in which the third flow lasts more than the biggest connections. In all the RED tests run (100 replications) we have never encountered situations in which the "shortest" flow's duration is longer than the "longest" flow's one. On the contrary we have shown before how these exceptions were affecting the results of Tail Drop.

The theoretical behaviour of RED finds confirmation in these results since the third flow is obviously favoured in the second set of experiments and its results oscillate slightly as its starting point varies. All the simulation results presented in this section show the nice RED property of fairness. It is a *double fairness*. There is fairness between the different scenarios for each starting point and between the three flows too. The limited range in which the graphs are contained proves the first type of fairness. Actually as regards the second kind we did not calculate any fairness index to demonstrate that, but comparing the results of RED and Tail Drop we notice how the third flow generally reaches higher performance when RED is employed. This means that even though we cannot properly speak of fairness between the three flows and demonstrate it, we can affirm that RED generally offers more changes to the third flow to start and reach improved performance. There are some replications in which all the flows achieve the same throughput. Even if this is not a very common case, with Tail Drop we never encountered situations of such level of equality (Figure 5-15).

Moreover, another objective of RED was to minimize the number of packets dropped in the router. Unfortunately as regards this last point RED seems to be quite unsatisfactory. The kind of traffic workload utilized in these tests is partly responsible for the high portion of packets retransmitted. Repeating the tests increasing the number of TCP connections and their data load may actually show a different behaviour from what we observed. However, even if RED discards more than Tail Drop, the high number of packets retransmitted does not degrade the performance, resulting on the contrary in an increased throughput for the network, as the results show. Thus RED provides a lower delay on the link thanks to the reduced size of the buffer queue. The final result is better performance in terms of throughput, elapsed time and fairness.

However, as observed before RED's objective is not only to drop few packets, but also to discard packets from each flow in proportion to the amount of bandwidth the flow uses on the output link. Therefore the connection with the largest input rate will have the biggest drop percentage among total dropped packets. Considering the results shown RED fully carries out this issue.

Finally another RED issue is to avoid the connections synchronization giving the possibility to the three TCP flows to reduce their sending rate at different moments and not all at the same instant. We have found cases of Tail Drop synchronization as the replication in Figure 5-4 and Figure 5-7 shown, where the packets loss and the retransmissions were occurring always in the same moments, as well of RED synchronization, such as the example shown in Figure 5-15. It is quite difficult to make an evaluation regarding this aspect considering the kind of workload utilized. Two connections are started simultaneously and the third one quite soon. The transfer durations are at most of 40 seconds. Thus it is quite reasonable that the initial congestion start to retransmit packets at the same moment and consequently reduce their rate almost together.

5.2 Tests with TCP and UDP traffic with services differentiation

The purpose of these tests is first to examine the impact of UDP traffic on a TCP connection's performance when they are sharing the same bottleneck link. Second our goal is to investigate the benefits introduced by differentiating the traffic through the use of specific classes of service, instead of one general class. Indeed the incoming UDP traffic can be enqueued into the same class of service used by the TCP flow, (first kind of HTB topology) or be treated as traffic of higher priority. In this case it is directed to another class of service where it has enough bandwidth to complete the transfer without experiencing any packet loss (second type of HTB configuration). The details about the HTB configurations deployed in these tests have been given in Section 4.5.2. Additionally, we provide a simple and limited study of the UDP performance only when the first kind of configuration is adopted, since in the second one, the CBR traffic always completes the transfer without any packet loss. In order to accomplish a complete analysis the same experiment was repeated several times, depending on the TCP traffic load, varying each time the CBR flow's starting point. Finally the two possible HTB configurations are tested in combination with the traditional Tail Drop discipline or RED algorithm, thus to enable a comparison between them.

We have run tests for three different test cases. They are classified on the basis of the TCP and UDP traffic's dimension and duration. Each of them consists of several scenarios depending on the number of CBR flow starting points considered. Each scenario is tested with 20 replications. The tables of results for the three scenarios are reported in Section 8. The three test cases are:

1. Long TCP transfer competing with UDP traffic for the entire connection.
2. Short TCP transfer competing with UDP traffic for the entire connection.
3. Long TCP transfer competing with TCP traffic for a short time

5.2.1 Test Case 1: long TCP transfer competing with UDP traffic

The traffic scenario is as follows:

- A single bulk TCP transfer of long duration (roughly 391.3 Kbytes)
- One CBR flow long enough to cover the entire TCP connection's transfer (about 264.1 Kbytes sent in 40 seconds).

The test case includes four scenarios corresponding to a delay in the CBR's start time of 0, 1, 2 or 3 seconds. The starting points are fixed in such a way to analyse the TCP performance when the CBR flow starts during the Slow Start phase of the initial connection. Slow Start is overly aggressive in probing for bandwidth because of the way it handles congestion window increase. Thus we examine in which way and how much the competing traffic can influence and especially damage the TCP connection.

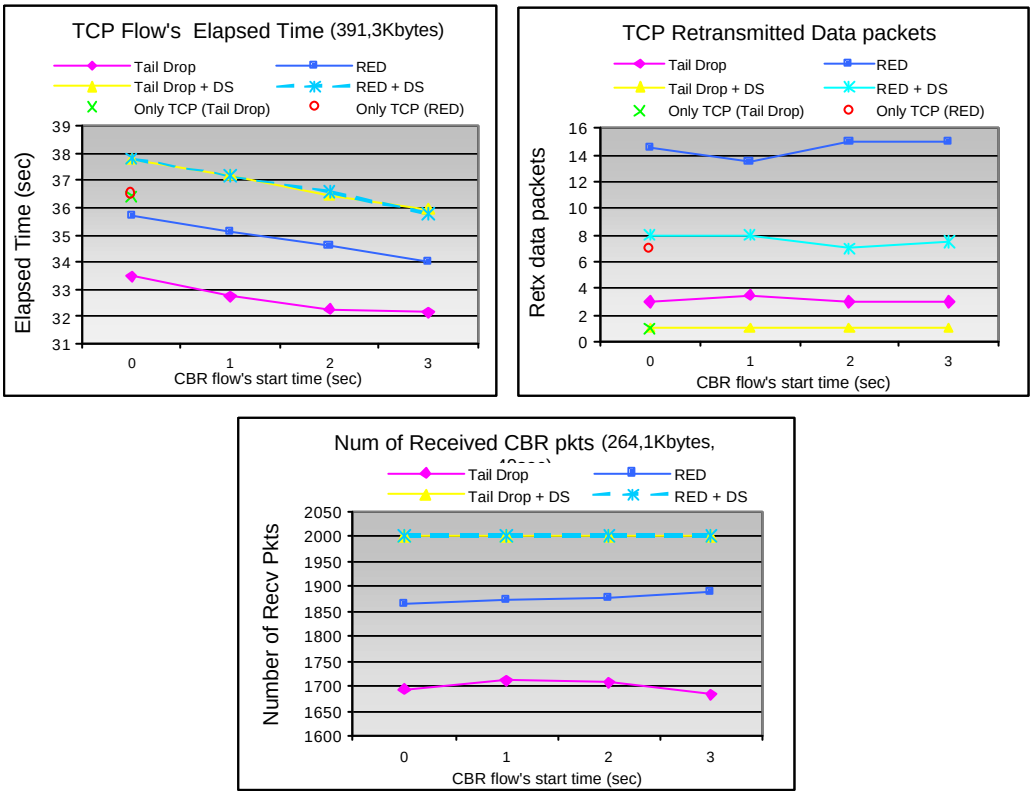


Figure 5-16. Statistics of the Test Case 1: long TCP transfer competing with UDP traffic (median values of 20 replications).

Figure 5-16 illustrates the results regarding the TCP flow's elapsed time and number of retransmitted data packets and the number of UDP packets received correctly as a function of the CBR flow's starting point. Each point on the plots is calculated as the median of 20 replications. In each scenario the four colours indicate the performance achieved when a particular HTB configuration is applied. "Tail Drop" and "RED" refer to the first kind of HTB typology with the respective disciplines deployed. All the traffic is enqueued in the same service class implementing one of the two queueing disciplines. Under this condition Tail Drop manages to ensure faster TCP transfer than RED does, but unfortunately it discards roughly 15% of CBR traffic. On the contrary, RED favours CBR traffic dropping only about 6% of its packets and provides larger TCP elapsed times. The TCP performance slightly improves for both disciplines as the CBR delay increases since TCP traffic is affected for less time by the competing UDP flow. Similarly the CBR flow improves its performance as it starts later even though the number of packets received increases slowly and only when RED is deployed. It is mostly due to this phenomenon the fact that as the CBR delay raises the UDP traffic may compete with the TCP connection when it is in the advanced part or at the end of the Slow Start stage. When RED is in use the Slow Start lasts from 2 up to 3 seconds while under Tail Drop it can reach even 8 seconds. Hence, considering that the maximum delay studied is two seconds, one soon realizes that with RED the CBR flow has more chances to begin the transfer when the TCP connection is near to end the Slow Start phase or at least to experience some packet drops. Slow Start may be very aggressive in probing for bandwidth and this leads to discarding big portions of CBR packets when Tail Drop is implemented. RED accepts the incoming flow providing more fairness.

Considering again Figure 5-16 the label "Tail Drop + DS" and "RED +DS" denote the results achieved when the two service classes configuration discussed before is implemented in combination with the respective discipline. The CBR traffic has sufficient bandwidth to serve the data traffic for the whole transfer duration (in fact no packet drop occurs). The introduction of two classes of service makes the TCP performance degrade for both disciplines since the available bandwidth is reduced, but its effect is mostly evident for the elapsed time achieved when the router deploys Tail Drop. In this case the results differ roughly 4 seconds from the previous configuration with only one class of service, while RED reports a transfer time raised with 2 seconds. This means that RED is treating the competing flows with more fairness than Tail Drop does.

In confirmation of RED fairness we have analysed the bandwidth utilization percentages, reported in Table 5-7. The case under study is the scenario *after 0 seconds*, when the traffic is served by a unique class of service. Note that the results for TCP and CBR traffic are not

directly comparable since the elapsed time of the corresponding flows are different. The CBR has fixed duration and can also last 6,5 seconds more than the TCP one. Hence it has available in this time interval the whole bottleneck bandwidth with the opportunity to improve its performance in the last phase. For this reason adding together the two median utilizations we obtain utilizations around 97-99%, which is quite unrealistic. The results for TCP and CBR must be analysed separately. The TCP and UDP data load sent correspond respectively to the 60% and 40% of the entire traffic. First we notice that Tail Drop usually presents slightly wider variation range than RED. Second the RED bandwidth utilizations are usually closer to the effective data load proportion than the Tail Drop ones. Tail Drop tends to favour the TCP traffic.

Bandwidth Utilization	Discipline	Min	Max	25%Perc	Median	75%Perc	Average	Elap.Time (sec) median
TCP flow	Tail Drop	0,65	0,71	0,66	0,67	0,68	0,67	33,47
	RED	0,61	0,64	0,62	0,63	0,63	0,62	35,69
CBR flow	Tail Drop	0,30	0,33	0,31	0,32	0,33	0,30	40,00
	RED	0,34	0,38	0,35	0,35	0,35	0,35	40,00

Table 5-7. Bandwidth Utilizations in the scenario *after 0 sec* of the Test Case 1 (long TCP transfer competing with UDP traffic).

Furthermore in Figure 5-16 we have reported the results of two tests run sending only one TCP flow of the same dimension as before (391,3 Kbytes), but decreasing the bottleneck link capacity of the amount of bandwidth necessary to serve the CBR traffic (the reduction is roughly 53Kbytes/s, considering that the total number of CBR packets sent in 40 seconds is 2001 we have estimated the consumed bandwidth). Independently of the queueing discipline adopted the TCP flow completes the transfer in less time compared to the tests with two service classes (the difference is quite small, about 1,3 seconds). Studying the Sequence Number traces we have noticed that when two service classes are deployed the Slow Start phase is usually shorter than in the case with one TCP flow, even if the number of retransmitted packets is the same. The difference is generally around 1 second. This means that the TCP performance increases because it is less affected by the CBR interference even though in theory the available bandwidth is the same.

The results of these experiments point out also the huge number of TCP data packets retransmitted when RED is deployed (roughly 5 times higher than in Tail Drop). This is due to the attempt of RED to keep the average queue size low. Consequently starting to drop packets

early, the TCP connection always experiences a short Slow Start phase (approximately half of Tail Drop's).

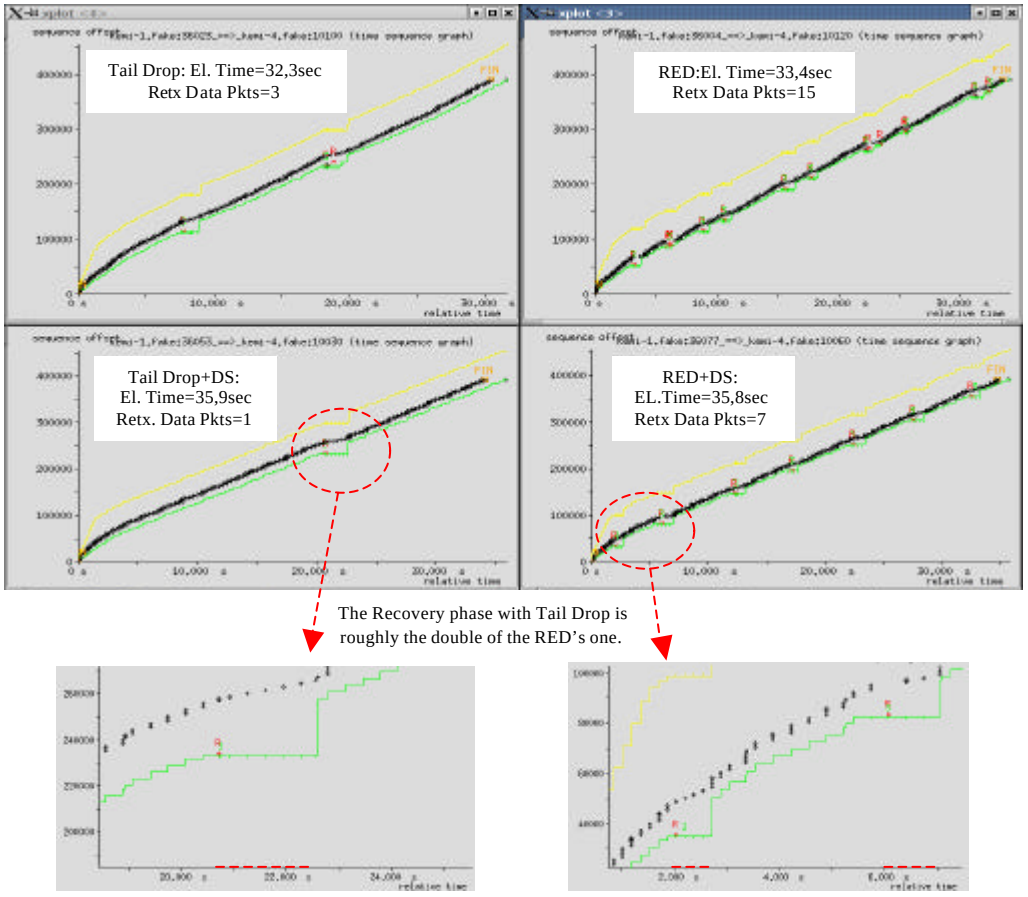


Figure 5-17. Analysis of the scenario *after 3 seconds* of the Test Case 1.

In order to understand why RED can achieve the same performance as Tail Drop in spite of the high number of packets discarded we take a closer look at four test replications, one for each type of possible HTB configuration. They are presented in Figure 5-17 and refer to the scenario *after 3 seconds.* The performance realized by each of them is quite close to the median values of the whole test as regards both the elapsed time and the number of retransmitted packets. On the left there are the Tail Drop graphs and on the right the RED ones. The number of retransmitted packets is quite elevated for RED in both configurations, but even though it is high the retransmissions are equally distributed along the whole transfer, especially when the two service classes configuration is deployed. Moreover if we analyse carefully the graphs at the bottom we see that the implementation of RED helps to recover quickly after a packet loss. The recovery phase in RED is shorter because the average queue in the router buffer is kept low and the congestion is more easily solved than in Tail Drop. Moreover the congestion window in the RED case is generally quite small. In fact in RED the black arrows' trend,

representing the data segments sent, follows the green line closely, which tracks the ACKs returned by the receiver. In Tail Drop the number of outstanding packets, not yet acknowledged, is much more larger and that is why when the congestion occurs it can involve several packets dropped. However, even though the two disciplines behave in different ways, from the results of these experiments it is fair to conclude that when the router selects the two service classes architecture, the TCP connections experience the same transfer.

5.2.2 Test Case 2: short TCP transfer competing with UDP traffic

The second set of experiments presents the same structure as the previous one, but the traffic load is reduced thus to accomplish a deeper analysis and capture some transient phenomena caused by the CBR flow arising. It is characterized as follows:

- One small TCP flow (roughly 46.7 Kbytes)
- One CBR flow that lasts for the whole duration of the TCP flow's transfer (about 39.7 Kbytes sent in 6 seconds).

The test case consists of five scenarios corresponding to the following CBR flow's starting time: after 0 sec, 0.5 sec, 1 sec, 1.5 sec or 2 sec.

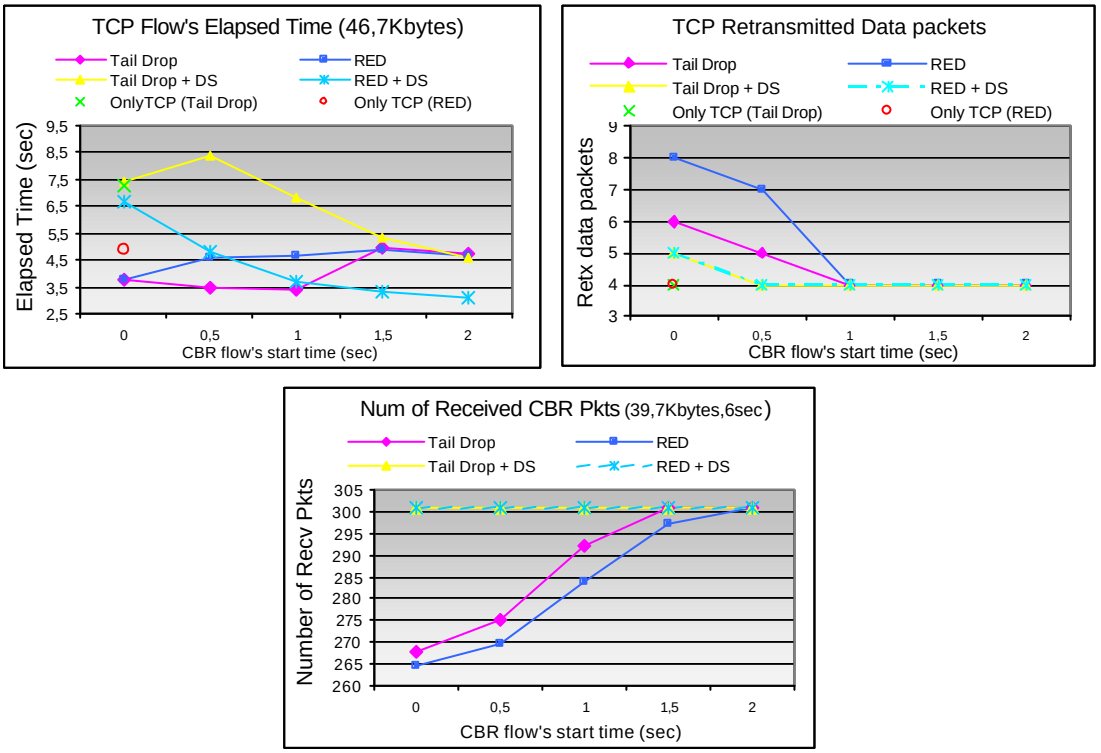


Figure 5-18. Statistics of the Test Case 2: short TCP transfer competing with UDP traffic (median values of 20 replications).

Figure 5-18 illustrates the relative statistics. Differently from the previous experiments the number of retransmitted packets observed is quite the same for both the disciplines, with the exception of the early scenarios. Being the Slow Start phases of similar duration, when the CBR flow begins the transfer it encounters analogous traffic conditions in both Tail Drop and RED tests. The length of the Slow Start phase generally varies from 2 up to 3 seconds.

When the same service class accommodates both TCP and UDP traffics (in the graph it corresponds to "Tail Drop" and "RED" curves) as regards the TCP results Tail Drop outperforms RED in the early scenarios. Tail Drop tends to favour TCP traffic when UDP traffic starts quite soon; otherwise when the CBR flow starts after 1.5 second or later the TCP elapsed time abruptly passes from 3.6 seconds to 5 seconds, thus reaching the RED performance. Concerning RED's elapsed time trend, it is quite stable if we exclude the scenario *after 0 seconds,* where the connection has proved to be extraordinarily fast. The CBR performance improves as the starting point is delayed, since it is less affected by the competition with the TCP flow.

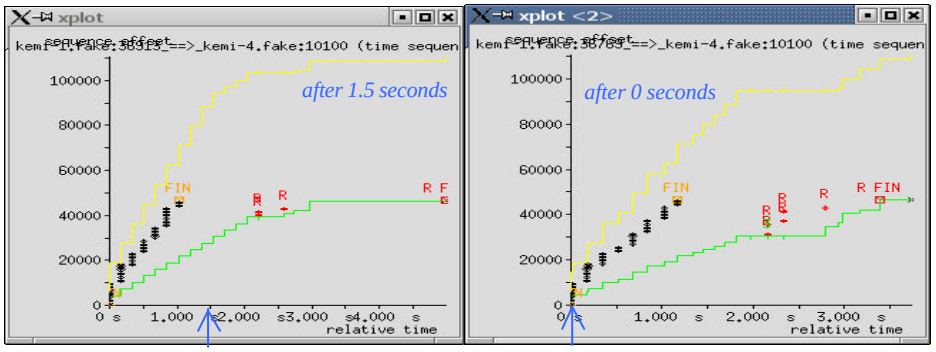


Figure 5-19. Test Case 2: examples of TCP connections taken from the scenario *after 1.5 seconds* (on the left) and from the scenario *after 0 second* (on the right).

We have studied thoroughly the Tail Drop behaviour through two significant scenarios and we have noticed an interesting phenomenon. Figure 5-19 illustrates the TCP sequence number graphs. On the right we report the situations mostly encountered in the scenario *after 0 seconds* and on the left *after 1.5 seconds*. We observe that in both the first packet lost is usually retransmitted after the 2nd second of connection. Both flows transmit the entire data load in a short time (roughly 1 second), but the acknowledgments of the last packets sent encounter some difficulties to arrive. In the trend on the right the sequence number of the first retransmitted segment indicates that more than half of the total packets sent have been acknowledged while, in the other graph, at the same instant, almost all the data packets have been correctly received. This is quite obvious since in the scenario *after 1.5 seconds* the TCP

connection can fully exploit all the link bandwidth in the early seconds, but after the CBR starts, the end of the connection becomes critical and the final packets are slowly acknowledged. The network becomes congested and the last packets are discarded. Four packets must be retransmitted and are acknowledged in a long time. Unlike in the scenario *after 0 seconds* the CBR traffic is competing from the beginning to gain bandwidth, hence the TCP connection starts to drop packets before, as demonstrated by the sequence number of the first packet retransmitted, quite lower than the previous one. In total, 6 packets are retransmitted. The connection manages to recover faster from the congestion and that is why it completes the whole transfer in a shorter time (3.7 seconds against 5 seconds of the scenario *after 1.5 seconds*).

What we have observed is the so-called phenomenon of *Slow Start overshoot*. Slow Start is overly aggressive in probing for bandwidth because of the way it handles congestion window increasing. The problem is that due to the exponential increase in the window size, it is possible to overshoot the congestion window and short flows, which transfer most of their data during Slow Start, can really suffer from this inefficiency. TCP fills up network buffers until packet loss occurs. Packet loss is followed by a period of idle time, and possibly severe reduction of the sending window. Such loss and the time needed for recovery typically do not significantly affect the long-term average throughput of a large transfer, but can be determinant on the final performance of a short transfer.

In our test we have considered quite a small TCP transfer, but the presence of CBR traffic helps TCP Slow Start to avoid overshoot or at least to alleviate its effect. That is why the results are better when UDP traffic is sent from the beginning. This fact also finds confirmation in the huge number of retransmitted packets carried over in the first two scenarios. However it is important to point out that the congestion occurs during the flow end phase. It is common knowledge that the recovery phase is poorly efficient in the last part of the connection. Like in this case the sender finishes transmitting data quite soon. The detection of a packet loss is slow, since no DUPACKs are received (this happens because the sender has completed the transfer and no new packets are sent) and the timeout expiration may take a long time. Retransmissions caused by timeout expiration are costly and degrade the final performance.

The same observation can partly explain why also RED achieves the best performance in the scenario *after 0 seconds* and retransmits such a high number of packets. At any rate, RED is smoothly influenced by the phenomenon and generally it suffers packets loss with fewer consequences in the last part of the connection and it is not subjected to serious Slow Start overshoot. The reason resides in the algorithm mechanism: RED starts discarding packets soon

thus to avoid timeout expiration and a long recovery phase. Sometimes this behaviour implies long TCP transfer, but performance more stable to changes in the external environment. Tail Drop discards fewer packets and maintains the buffer queue full. It can be faster, but it is more exposed to incoming traffic behaviour and its performance is strongly dependent on its characteristics, such as data load, duration and starting point.

As regards the behaviour of the UDP traffic when one class of service is utilized we observe that the CBR performance increases as the flow starts later. Tail Drop tends to discard a smaller portion of packets than RED, but the difference is quite contained, from 4 to 9 packets out of 301 packets.

When TCP and UDP traffics are enqueued in different service classes we notice a heavy degeneration in the TCP performance of both disciplines, due the reduction in the available bandwidth. As observed in the previous test case the worsening is more serious in Tail Drop's results. The elapsed time trend decreases as the CBR starts later since the TCP connection is affected for less time by the competing flow. If we take a closer look at the scenario *after 0.5 second*, where the Tail Drop performance is greatly unsatisfactory, we see that the TCP flow's elapsed time for Tail Drop almost double the RED one. While Tail Drop in combination with one class of service deployed reports elapsed times around 3.5 seconds under this second configuration it degenerates to a maximum of 8.4 seconds. Afterwards in the last two scenarios the Tail Drop results are approximately the same for both types of HTB configuration.

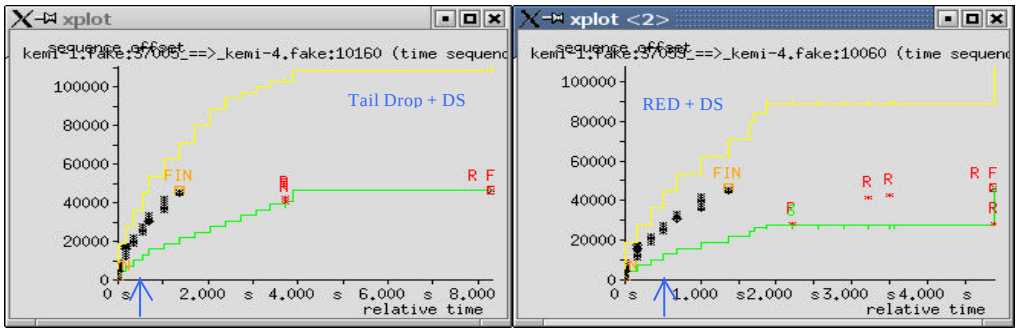


Figure 5-20. Test Case 2-Scenario *after 0.5 seconds:* Tail Drop + DS (on the left) and RED + DS (on the right).

To find an explanation to this trend we consider the comparison proposed in Figure 5-20. It refers to two TCP connections taken from the scenario *after 0.5 seconds*. The corresponding elapsed times are 8.3 seconds for Tail Drop and 4.9 seconds for RED. In the Tail Drop graph (on the left) the timeout expiration occurs: the timeout is re-calculated and a new phase of Slow Start is triggered. This happens around the 4th second of transmission and the sequence number

of the retransmitted packets is quite high, hence a few packets have not yet been acknowledged. In the RED graph (on the right) the retransmission is caused by the reception of three DUPACKs and the sequence number of the first retransmitted packet is much lower than the one in Tail Drop's example. RED starts to discard packets before and manages to recover faster from congestion occurrence because the average number of packets in the router buffer is low. However we notice that, like in the previous example, independently of the discipline adopted, the recovery phase is generally quite critical in a situation like this, in which we are considering short transfers and the packets are dropped after all the data have been sent. We also observe that Tail Drop tends to discard consecutive packets, while with RED they are distributed over a wider sequence number range. This is because Tail Drop starts to discard only when the queue is full while RED drops randomly before the queue overflows. This test also offers an example of *TCP Selective Acknowledgements Options*. In RED's trace the Fast Recovery phase is triggered and the connection retransmits only the packets whose sequence number is missing. This option is greatly helpful when algorithms such as RED are deployed, since it discards packets randomly and may cause on the receiver side the reception of blocks of packets out of order, but correct.

The results presented have shown how the incoming UDP traffic may affect the TCP connection when it is sharing the same class of service or otherwise when a specific class serves it. This also was the topic of the first section of experiments, but unlike it, here we have considered traffics of short duration in order to better investigate the presence of transient behaviour.

In both configurations the additional traffic causes congestion and slows down the TCP transfer. The impact is more serious as the TCP connection has more outstanding packets not yet acknowledged. In our experiments, the TCP flow being quite short, this critical interval falls after 1.5 seconds of transmission, as seen in Figure 5-19. Note that when the two service classes architecture is implemented the most critical interval is shifted before to 0.5 seconds and the reason can be easily found. When the second HTB configuration is deployed, even if the CBR flow starts around 0.5 second, the effective reduction of the available bandwidth occurs a bit later, since it is regulated by the HTB discipline. We did not make an estimation of how much time it can require, but considering some experiments proposed in [Dev02] we can assume that an additional delay from 0.5 up to 1 second is an acceptable hypothesis. Moreover, as expected, the adverse impact of UDP traffic generally is of minor importance if this starts when the TCP connection is near to ending. Finally the results have shown that the TCP connection can be partly favoured by the competing CBR traffic if they start simultaneously.

This can be helpful to avoid a phenomenon like Slow Start overshoot, especially when Tail Drop is deployed.

A last observation regarding the statistics of this test case refers to the elapsed time realized when only one TCP flow is sent on a link whose bandwidth has been reduced by the amount necessary to serve the CBR traffic. As indicated by Figure 5-18 under Tail Drop the elapsed time remains the same reported when two service classes are deployed. Instead, in RED tests the elapsed time is quite reduced. It reaches a median value of 4.56 seconds, but the range 25%-75%Perc is 4.1-6.8 sec. The variation of the results is related at which point the first packet is retransmitted. Later it occurs and the better is the final performance. But as pointed out several times in this section this is a very particular case and the results are heavily influenced by the time needed to have acknowledged the last packets. Analysing the corresponding traces we have seen that in Tail Drop experiments the connection is greatly damaged by packets loss in the ending part of the transfer and experiences a long recovery phase. RED, as explained before, suffers less from this inefficiency and realizes better elapsed time than Tail Drop but only in some cases. However it is difficult to find a general explanation since the RED results are quite unsettled.

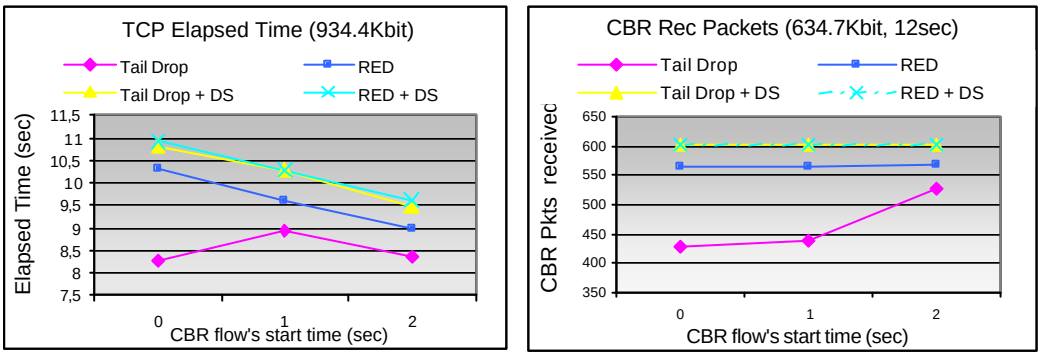


Figure 5-21. Statistics of the additional Test Case: medium TCP traffic competing with UDP traffic (median values of 20 replications).

Finally comparing the results of both test cases we observe that the second set of experiments constitutes a critical case since the transfer is very short and the retransmissions occurs when the transfer is complete. We have run an additional set of tests involving traffic of intermediate duration between the two cases presented (TCP traffic: 116.8Kbytes; UDP traffic: 79.3Kbytes in 12 seconds). Briefly we notice that the statistics resemble what we observed in the first test case with the exception of the scenario *after 0 seconds.* We have found again the phenomenon encountered with the small transfers. When the CBR transfer starts simultaneously with the TCP connection the elapsed time realized under Tail Drop decreases

and the number of packets dropped is quite huge (13 packets retransmitted with Tail Drop and 5 with RED). RED presents stable trends, similar to the first test case.

5.2.3 Test Case 3: long TCP transfer competing with UDP traffic for a short time.

Finally the third set of experiments was intended to provide a study of TCP performance when UDP competing traffic starts the transfer during the Fast Retransmit or the Congestion Avoidance phases of the TCP flow. The router buffer size fixed to 20000 Bytes in order to trigger packets lost in the TCP connection and start the CBR flow right now we have selected this kind of traffic:

- A single bulk TCP transfer of long duration (roughly 391.2 Kbytes sent)
- One short CBR flow ending before the TCP connection finishes (about 33.1 Kbytes sent in 5 seconds).

We consider several scenarios although only in the last ones we realize the desire starting condition. The scenarios of which we report the results are: after 0 sec, 2 sec, 7 sec, 12 sec, 16 sec and 18 sec. Figure 5-22 illustrates the performance achieved by TCP and CBR traffic.

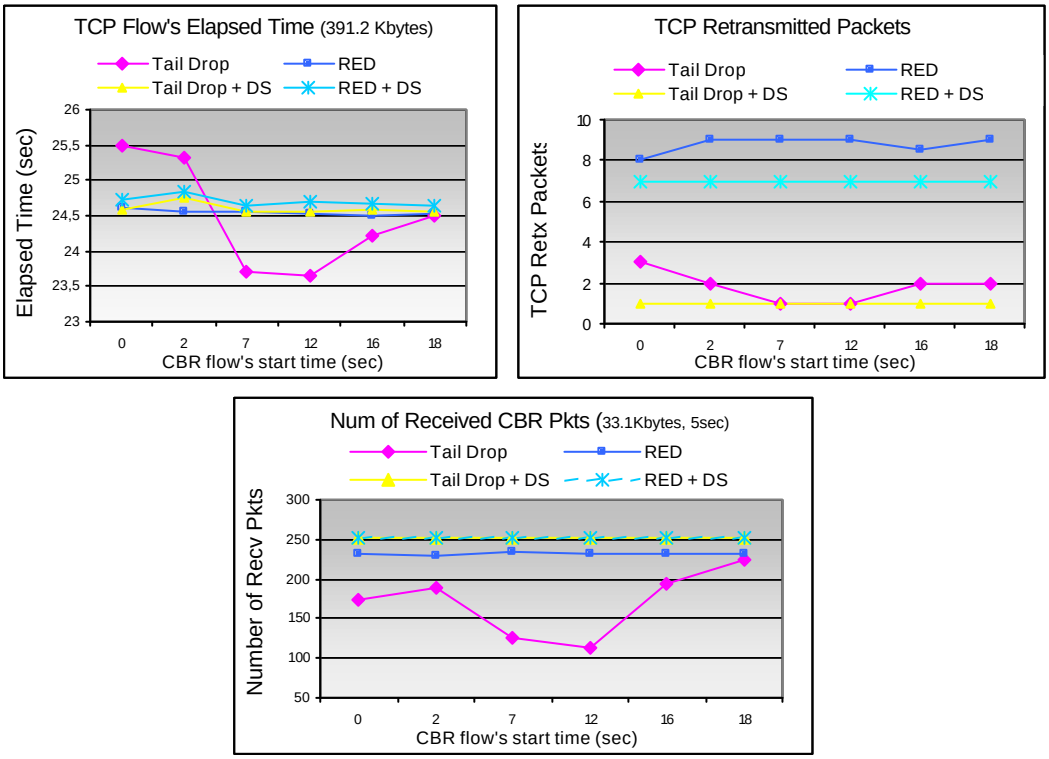


Figure 5-22. Statistics of the Test Case 3: long TCP transfer competing with UDP traffic of short duration (median values of 20 replications).

When only one class of service implementing the Tail Drop discipline is in use the trend of TCP elapsed time is rather irregular and spaces out quite a large range. In the first two scenarios the performance is poor, it improves for the median starting points and finally it worsens again for the last scenarios. The graphs of TCP and CBR follow the same trend. For example where the TCP elapsed time decreases 0.5 seconds the number of CBR packets correctly received increases with 100 packets. Differently, when RED is deployed the TCP elapsed time is quite steady around 24.5 seconds as well as the number of CBR packets received, roughly 230 packets out of 251 transmitted. The CBR flow's starting point seriously influences the TCP performance when the router adopts the Tail Drop discipline, while when RED is deployed the trend resembles a constant line. The difference between the maximum and minimum median elapsed times is 1.8 seconds for Tail Drop and 0.5 seconds for RED. The same observation is valid for the CBR traffic. This implies that RED can assure a more constant service to both traffics and even of higher performance in some cases. RED outperforms Tail Drop in serving the TCP connection in the early scenarios, but after 7 seconds of delay Tail Drop manages to complete the TCP transfer in minor time to the detriment of the CBR traffic that suffers large portions of packet losses. These results demonstrate again that there is an effective guarantee by deploying RED to achieve fair performance when traffics are sharing a bottleneck link. In particular we observe how RED is able to guarantee an optimal service to the CBR traffic since almost all the packets sent are correctly received.

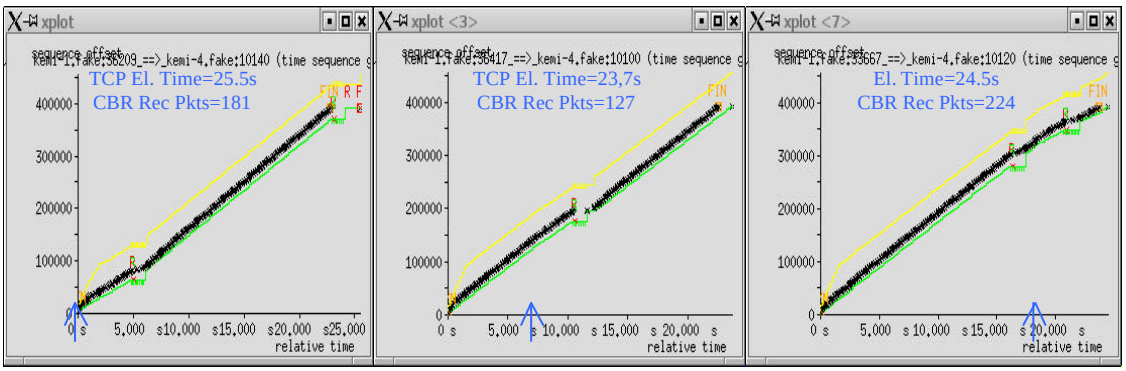


Figure 5-23. Test Case3: Sequence Number graphs when Tail Drop is deployed in the scenario *after 0 seconds* (left), *after 7 seconds* (middle) and *after 18 seconds* (right).

The trend followed by the Tail Drop statistics presented above can be summarized through three recurring situations. Figure 5-23 shows the trends of three TCP connections belonging to different scenarios. In the graph on the left the UDP traffic starts at 0 seconds and then after 5 seconds congestion is detected. The competing traffic achieves mediocre results (181 packets received out of 251). In the second panel the CBR flow starts after 7 seconds. The

TCP connection is less damaged than before by the competing traffic, as confirmed by the low number of UDP packets received. Finally in the last graph the CBR flow starts when the TCP congestion has already experienced the first packet loss. The performance is quite good and fair for both traffics. The TCP can complete in a short time (intermediate between the previous cases) and the CBR flow takes advantage from starting the transfer when the TCP connection is going out from the congestion state and is not transmitting packets so intensely as in the first Slow Start phase.

When both the classes of service are in use, Tail Drop outperforms RED slightly, but the difference in the elapsed times realized, is around 0.2 seconds. We also observe that the TCP performance achieved with this HTB configuration resembles the results of RED in the previous topology. This means that RED is less influenced than Tail Drop by the arrival of a competing traffic of higher priority, as the CBR can be considered. Anyway note that in this test case the CBR flow lasts only for 5 seconds and this partly influences the similarity between the RED results. In the first test case where the CBR lasts for 40 seconds and also in the second ones, where both traffics are quite small, the degradation introduced by the class differentiation on the TCP flow's performance is more pronounced.

5.2.4 Summary

The results presented in this second part of experiments cannot be evaluated leaving out of consideration the kind of traffic workload through which they are generated. The first and the last test cases offer a macroscopic view of the interference between TCP and UDP traffic when they are competing to gain bandwidth on the same bottleneck link. However the second set of experiments is characterized by shorter flows and provides a microscopic view of the mechanisms underlying the discipline behaviour, thus allowing a detailed study of transient phenomena.

When the network is loaded with a huge bulk TCP transfer, RED guarantees more enhanced performance to the competing CBR flow than Tail Drop does, especially when this is of brief duration. Moreover RED provides stable results to both traffics, independently of the UDP traffic's starting time. This fact demonstrates that RED is more available to serve a new incoming traffic, apart from its data load and moment of arrival and it can absorb traffic with variable characteristics with more agility. RED never falls in situations of extreme unfairness in which one flow is able to monopolize the available bandwidth stifling the emerging flow. Its experimental results are always closer than the Tail Drop's ones to the performance achieved

when two service classes are employed and the CBR traffic of higher priority is wholly satisfied. Even though in some scenarios Tail Drop outperforms RED, particularly as regards the TCP transfer, this is achieved involving a heavy detriment of the other traffic. Especially in the last test case Tail Drop behaves in an unstable way spacing over a wide performance range. Finally in these experiments with huge TCP traffic the introduction of such differentiation in the service classes causes an obvious degradation in the TCP connection's performance, since the available bandwidth is reduced in order to serve the higher priority CBR traffic, but leads the two disciplines to report the same results and also helps Tail Drop to stabilize its trend.

Therefore we can conclude that RED's fairness issue finds confirmation also in this second section of experiments. There is fairness for two reasons. Firstly the RED performance generally does not present great variance as the CBR flow's starting point changes. Secondly the bandwidth utilizations realized with RED are quite close to the data load ratios. RED guarantees fair sharing of the available bandwidth while Tail Drop tends to favour the TCP traffic, especially if it is of huge dimension, to the detriment of UDP performance.

The microscopic view offered by the second test case points out the key principle on which RED is based and in a sense determines its success. RED drops packets before the queue buffer overfills keeping the buffer occupancy low. This allows it to absorb transient congestion, as the one caused by a competing flow (in our case the CBR traffic when only one class of service is deployed) or by a drastic reduction in the available bandwidth due to the arrival of a flow of higher priority (in this test case the CBR traffic when two service classes are deployed).

In the first set of experiments, in which only one class of service is employed, the adverse impact of UDP traffic on the TCP performance is mitigated by delaying the CBR arrival. On the other hand, in this second test case we have observed the opposite phenomenon. The CBR flow improves the TCP performance if it starts early since it causes some packet drops earlier than usual. It prevents the Slow Start phase from being excessively aggressive thus to avoid phenomena such as Slow Start overshoot and severe states of congestion. Otherwise arriving later it can heavily damage the TCP performance and eventually cause a burst of data to be discarded. It is known that TCP recovers easier from a single packet than a burst of packets dropped. Hence RED, by keeping the buffer queue low, can generally avoid consecutive packet drops thus to faster recover from congestion occurrence. When the traffic is accommodated through two service classes the occurrence of the bandwidth reduction seriously hurts the TCP performance in Tail Drop tests and the performance is greatly unsatisfactory.

Finally the results of this section point out another issue related to RED implementation. The number of packets retransmitted is still much higher than with Tail Drop. Also in the previous section of experiments we have criticized this behaviour, but what we have found out here, especially in the second test case, is that it is thanks to this behaviour that RED can guarantee good performance and prevent congestion collapse. As discussed previously a limitation of our approach resides in the simplicity of the traffic workload implemented since the number of flows and the data load are limited. Considering more complex scenarios with transfers of longer duration we could provide more complete evaluation as regards this RED issue.

6 Conclusions

In the present thesis we have analysed the RED active queue management algorithm and verified how much effective mechanism is for congestion avoidance at the gateway. The expected benefits derived from its employment have been evaluated by observing the effects produced on TCP and marginally UDP service performance. Essentially our analysis is based on a comparative study between Tail Drop and RED mechanisms. We have explored how well the state-of-the-art TCP performs when Tail Drop is deployed, determined the possible improvements carried out by RED introduction and identified the key reasons behind the ameliorated or worsened behaviour. Further, we have studied the application of RED to a QoS-enabled architecture, in which the traffic (TCP and UDP) is classified according to the belonging class of service and thereby receives the specific treatment. The main problem the TCP and UDP traffic have to face up to in such an environment is the heavy congestion simulated on the last-hop router where the bandwidth is strictly limited.

We have surveyed the existing work on TCP, thus to select in the Linux TCP implementation adopted in our tests, the most common specifications and recommended improvements, such as Selective Acknowledgement option and New Reno algorithm. We have presented the RED algorithm and derived some guidelines in tuning its parameters. We have made an overview of Active Queue Management scheme mentioning several related studies and suggested improvements to RED mechanisms in order to outline the main drawbacks deriving from its deployment. In particular we have introduced ECN as one of the most promising techniques to be employed in combination with RED.

We have made experimental tests using a simple testbed wired network of four components. Two fixed hosts are connected via two routers and the last-hop router alternatively employs the Tail Drop or RED discipline. The router buffer size is fixed for both to 20Kbytes. Through Linux Traffic Control we have generated a bottleneck link on the last-hop link in order to produce congestion and analyze how the two disciplines are able to prevent or at least recover from its occurrence. The data communication uses the TCP or in some cases UDP protocol. Further we have experimented with RED by introducing services differentiation. The traffic control framework available in the Linux kernel has enabled us to adopt a queueing discipline that maps the traffic to different service classes through the use of appropriate filters.

We have not implemented a complex Differentiated Services architecture since our intent is just to produce a differentiation in the traffic service to study how high priority traffic can impact on the performance of low priority traffic. Moreover, we note that even though our experiments are not conducted in a wireless environment, being characterized by slow wired links our results can be extended to a more general environment consisting of slow links as the wireless case is.

The empirical tests have been run in two major categories that mostly differ for the kind of workload used. The first category focuses on a study of TCP services performance. It consists of three TCP competing flows carrying different data loads and starting at variable times. Two flows are sent simultaneously while the third one of smaller data load is delayed. The main issue investigated is how much fairness RED or Tail Drop can assure when the TCP connections share the same bottleneck link capacity and which level of performance can be guaranteed. The second category involves both TCP and UDP traffic and it is articulated in three phases. First, we observe the impact of UDP traffic on TCP performance when they compete to gain bandwidth over the same link of limited capacity. Second we repeat the same experiments giving higher priority to UDP traffic through the introduction of a queueing discipline in the last-hop router, which allows the specification of several classes of service. Third we compare the results of the first and second case and examine the effects introduced by services differentiation on TCP performance.

The first relevant result of our tests concerns the expected nice RED property of fairness. Our experiments demonstrate that RED guarantees fair sharing of the bottleneck link capacity while Tail Drop tends to favour one or more connections over the whole traffic. In the first category of experiments, involving three TCP competing connections, RED shows a “double” fairness. First, RED results present stability as the third flow‘s starting time varies while when Tail Drop is deployed only two seconds of difference in the arrival time of the third flow makes its elapsed time double. Indeed the Tail Drop results are heavily influenced by the third flow’s start time. When it starts in coincidence of the initial flows’ Slow Start it is adversely affected; otherwise if it finds them in congestion avoidance or fast retransmit phase it manages to gain much more bandwidth. Second, RED does not penalize the last flow like Tail Drop does. It provides to it high performance, without hurting the Tail Drop results concerning the initial connections. In some cases the three flows’ results reach quite close values. As regards the second test category, RED similarly demonstrates to be more available to serve new incoming traffic, such as the UDP one, apart from its data load and arrival time. It guarantees steady results to both TCP and UDP traffic and most of all when TCP and UDP traffic share the same

bandwidth capacity, the corresponding bandwidth utilizations are quite close to the relative data load ratios.

The second relevant result in support of RED fairness is that RED has never fallen in situations of extreme unfairness and do not allow phenomena such as Lock-Out, in which one source may be able to monopolize the queue resources denying to the other connections the possibility to find place in the router buffer. RED generally offers more changes to the incoming flow (TCP or UDP) to start and reach high results than Tail Drop does. When we considered three TCP competing flows occasionally the last flow was not able to start the connection and the corresponding source was throttled until the initial flows completed the entire data transfer.

The third result concerns the RED issue of minimizing the number of packets dropped in the router. As regards this aspect, RED fails since all our RED tests experience high rates of retransmitted packets. However the results show that even if RED discards more than Tail Drop, the large number of packets retransmitted does not degrade the performance, resulting on the contrary in improved throughputs for the network. In the second category of experiments we have proposed a detailed analysis or RED mechanism in order to investigate why such a high number of dropped packets does not adversely affect the performance. We have observed that by starting to discard packets before the buffer queue overfills, RED prevents severe congestion states. It manages to absorb transient congestion, such the one caused by competing flows or by drastic reduction in the available bandwidth due to the arrival of higher priority traffic (in our test it was represented by UDP traffic). The RED recovery phase has proved to be fast even though multiple segments are dropped. Especially if packets are lost in some critical moments, such as the connection ending, after all data load has been transmitted, Tail Drop has shown to be quite unsatisfactory. Packets are discarded in bursts and the recovery phase takes a long time. RED starts to discard early and randomly. The retransmitted packets are quickly acknowledged and avail of the SACK option.

Examining the high rates of retransmitted packets in the RED tests, we note a quite important result. Indeed RED's objective not only concerns to drop few packets, but also to discard packets from each flow in proportion to the amount of bandwidth the flow uses on the output link. Therefore the connection with the largest input rate will have the biggest drop percentage among total dropped packets. When the three TCP connections share the same bottleneck link RED fully carries out this issue by guaranteeing to each flow a retransmitted packet rate roughly proportional to the data load sent into the network.

From the results of these experiments it is fair to conclude that RED at least with this kind of workloads is not as concerned to minimize the number of packets dropped, as to avoid unnecessary packet drops at the gateway. We have demonstrated that by providing advance warning of incipient congestion and consequently starting to drop packets early, it manages to achieve high performance and guarantee fair treatment to all the traffic sources.

Tail Drop tends to drop fewer packets, but achieves lower performance and in some particular cases it suffers from phenomenon such as Slow Start overshoot, especially if with small TCP traffic. We have noticed an interesting and unexpected behaviour considering TCP and UDP transfer that compete to gain bandwidth on the same bottleneck link and employing the Tail Drop discipline in the router. The UDP traffic lasts for the entire TCP connection and is started at different times. When the traffics are quite large the TCP performance improves slightly as the UDP flow is delayed since the interference between them is reduced. Considering small transfers instead we have observed the opposite phenomenon. The UDP flow helps to improve the TCP results if it starts early since it causes some packet drops earlier than usual. It prevents the Slow Start phase from being excessively aggressive thus to avoid phenomena such as Slow Start overshoot and severe congestion states. Otherwise by arriving later the UDP load can heavily hurt the TCP performance and eventually cause a burst of data to be discarded.

Finally, as regards the results achieved with the introduction of services differentiation Tail Drop and RED performs in a similar way, except in some cases, such as the one explained above, in which Tail Drop is heavily hurt by the higher priority traffic. The UDP flow represents the higher priority flow and its class of service has enough bandwidth to serve it without any packet loss. Independently of the discipline employed the TCP traffic is adversely affected by the arrival of the UDP traffic since the available bandwidth is reduced. But we have noticed that the degradation in TCP performance caused by the introduction of two classes of service to the respect of the configuration with one is usually more serious under Tail Drop. On the other hand under this router configuration Tail Drop's results become more stable and the number of retransmitted packets decreases.

Most of the theoretical properties of RED have found confirmation in our results, even though we note that the methodology according to which the experiments have been organized has a direct impact on the relevance and utility of the results obtained and the conclusions derived. Hence the simulation approach may be appropriate for conducting further examination on several aspects. For instance in most of our tests a huge number of retransmitted packets characterizes the TCP connections. We believe that the traffic workload utilized in these tests is

partly responsible for this. Repeating the tests, increasing the number of TCP connections and their data load may help to carry out more promising results and important observations regarding the problem of connection synchronization.

Moreover, although much research effort has been focused on understanding and utilizing the RED algorithm, some interesting topics are yet to be investigated in more detail in future. First of all RED's performance is highly dependent on the setting of its parameters. No firm guidelines exist on configuring RED parameters and current suggestions fail to provide the desired performance scalability. In developing this study we have encountered several problems in setting the RED parameters correctly since the available literature is not clear and precise about this aspect. We have derived some guidelines on setting RED parameters but they need to be tested under huge and several types of traffic workload. The problem is that it is hard to find parameters that are optimal in all the possible circumstances. They depend on the traffic characteristics and for this reason adaptive approaches will be addressed in future studies more extensively.

Future research is also needed to determine the optimum average queue size for maximizing throughput and minimizing delay for different network and traffic conditions. In our study we did not consider as performance metric the router queue length. It could be useful to develop a study to evaluate how aggressive the RED algorithm is and how much it is able to absorb and accommodate packets burst.

Finally we can conclude that a deployment of the Explicit Congestion Notification (ECN) could make RED more attractive, because ECN avoids congestion-related losses. By following this path of evolution RED will be able to provide advance warning of incipient congestion more efficiently and with less waste of resources.

References

[ABF01] Allman. M., Balakrishnan H. and Floyd S., *Enhancing TCP's Loss Recovery Using Limited Transmit*, RFC 3042, January 2001.

[AF99] Allman M. and Falk A., *On the Effective Evaluation of TCP*, ACM Computer Communication Review, 5 (29), October 1999. Also available from: http://www.acm.org/sigcomm/ccr/archive/1999/oct99/allman2.pdf.

[AFP98] Allman M., Floyd S. and Partridge C., *Increasing TCP's Initial Window*, RFC 2414, September 1998.

[APS99] Allman M., Paxson V. and Stevens W., *TCP Congestion Control*, RFC 2581, April 1999.

[ASK99] Alemesberger W., Salim J. H. and Kuznetsov A., *Differentiated Service on Linux*, June 1999.

[BBCD98] Blake S., Black D., Carlson M., Davies E., Wang Z. and Weiss W., *An Architecture for Differentiated Services*, RFC 2475, December 1998.

[BCCD98] Braden R., Clark D., Crowcroft J., Davie B., Deering S., Estrin D., Floyd S., Jacobson V., Minshall G., Partridge C., Peterson L., Ramakrishnan K., Shenker S., Wroclawski J. and L. Zhang, *Recommendations on Queue Management and Congestion Avoidance in the Internet*, RFC 2309, April 1998.

[BCS94] Braden R., Clark D. and Shenker S., *Integrated Services in the Internet Architecture: an Overview*, RFC 1633, June 1994

[BK98] Balakrishnan H. and Katz R. H., *Explicit Loss Notification and Wireless Web Performance*, Proceeding IEEE Globecom Internet Mini-Conference, Sydney, Australia, November, 1998. Also available from: http://nms.lcs.mit.edu/~hari/papers/globecom98.

[BKGM01] Border J., Kojo M., Griner J., Montenegro G. and Shelby Z., *Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations*, RFC 3135, June 2001.

[BPSK96] Balakrishnan H., Padmanabham V. N., Seshan A. and Katz R. H, *A Comparison of Mechanism for Improving TCP Performance over Wireless Links*, ACM SIGCOMM '96, Stanford CA, August 1996. Also available from: http://daedalus.cs.berkeley.edu/publications/sigcomm96.pdf.

[BSK95] Balakrishnan H., Seshan S. and Katz R.H., *Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks*, ACM Wireless Networks, December 1995. Also available from: http://citeseer.nj.nec.com/balakrishnan95improving.html.

[BVE99] Bettstetter C., Vögel H. J. and Eberspächer J., *GSM Phase 2 + General Packet Radio Service GPRS: Architecture, Protocol and Air Interface*, IEEE Communications Surveys, 1999. Also available from: http://citeseer.nj.nec.com/bettstetter99gsm.html.

[CISCO98] CISCO Systems, *Weighted Random Early Detection on the Cisco 12000 Series Router*, 1998, Release 11.2 GS version of WRED. Also available from: http://www.cisco.com/univercd/cc/td/doc/product/software/ios112/ios112p/gsr/wred_gs.htm#xtocid0.

[CF98] Clark D. and Fang W., *Explicit Allocation of Best Effort packet delivery service*, ACM Transactions on Networking, August, 1998. Also available from: http://www.cs.princeton.edu/~wfang/papers.html.

[CEP99] De Cnodder S., Elloumi O. and Pauwels K., *Effect of different packet sizes on RED performance*, 1999. Also available from: http://www.icir.org/floyd/red/Elloumi99.pdf.

[CJOS02] Christiansen M., Jeffay K., Ott D. and Smith D., *Tuning RED for Web Traffic*, SIGCOMM, August 2000. Also available from: http://www.cs.unc.edu/~jeffay/talks/Maelardalen-QoS-Survey.pdf

[Dev02] Devera M., *HTB Linux queueing discipline manual-user guide*, May 2002. Also available from: http://luxik.cdi.cz/~devik/qos/htb/manual/userg.html.

[DMKM01] Dawkins S., Montenegro G., Kojo M., Magret V. and Vaidya N., *End-to-end Performance Implications of Links with Errors*, RFC 3155, August 2001.

[FKSS99] Feng W., Kandlur D. D., Saha D. and Shin K.G., *BLUE: A New Class of Active Queue Management Algorithms*, 1999. Also available from: http://citeseer.nj.nec.com/feng99blue.html.

[FKSS01] Feng W., Kandlur D. D., Saha D. and Shin K.G., *Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness*, 2001. Also available from: http://citeseer.nj.nec.com/feng01stochastic.html.

[Flo94] Floyd S., *TCP and Explicit Congestion Notification*, 1994. Also available from: http://citeseer.nj.nec.com/floyd94tcp.html.

[Flo97] Floyd S., *Discussion on setting RED parameters*, November 1997. Also available from: http://www.icir.org/floyd/REDparameters.txt.

[Flo00] Floyd S., *Congestion Control Principles*, RFC 2914, September 2000.

[FH99] Floyd S. and Henderson T., *The New Reno Modification to TCP's Fast Recovery Algorithm*, RFC 2582, April 1999.

[FJ93] Floyd S. and Jacobson V., *Random Early Detection gateways for Congestion Avoidance*, IEEE/ACM Transactions on Networking, V.1 N.4, August 1993, pp. 397-413. Also available from: http://ftp.ee.lbl.gov/floyd/red.html.

[FKSS99] Feng W., Kandlur D., Saha D. and Shin K., *A Self–Configuring RED Gateway*, Infocom, March 1999. Also available from: http://citeseer.nj.nec.com/feng99selfconfiguring.html.

[FGS01] Floyd S., Gummandi R and Shenker S., *Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management*, August 2001. Also available from: http://citeseer.nj.nec.com/floyd01adaptive.html.

[FMMP00] Floyd S., Mahdavi J., Mathis M. and Podolsky M., *An Extension to the Selective Acknowledgement (SACK) Options for TCP*, RFC 2883, July 2000.

[GVP01] Geczi C, Varga B and Pilisi I., *Differentiated Services – Network Configuration and Management (DISCMAN)*, June 2001. Available from: *http://www.eurescom.de/~pub-deliverables*.

[Hus00] Huston G., *Internet Performance Survival Guide (QoS Strategies for Multiservice Networks)*, 2000.

[HBWW99] Heinanen J., Baker F., Weiss W. and Wroclawski J., *Assured Forwarding PHB Group*, RFC 2597, June 1999.

[HMVV02] Hubert B., Maxwell G., Van Mook R., Van Oosterhout M., Schroeder P. and Spaans J., *Linux Advanced Routing & Traffic Control HOWTO*, 2002. Also available from: http://www.tldp.org/HOWTO/Adv-Routing-HOWTO/

[IMLG02] Inamura H, Montenegro G., Ludwig R., Gurtov A. and Khafizov F., *TCP over Second (2.5G) and Third (3G) Generation Wireless Networks*, Internet-Draft draft-ietf-pilc-2.5g3g-09, June 2002. Also available from: http://www.ietf.org/internet-drafts/draft-ietf-pilc-2.5g3g-12.txt.

[Jac88] Jacobson V., *Congestion Avoidance and control*, in Proceedings of ACM SIGCOMM 1988, Symposium in Stanford, CA, August 1988. Also available from: http://citeseer.nj.nec.com/jacobson88congestion.html.

[Jai91] Jain R., *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurements, Simulation and Modeling*, Wiley 1991 (p. 36).

[JNP99] Jacobson V., Nichols K. and Poduri K., *An Expedited Forwarding PHB*, RFC 2598, June 1999.

[JR88] Jain R. and Ramakrishnan K.K., *Congestion Avoidance in Computer Networks with a Connectionless Network Layer: Concepts, Goals and Methodology*, Proc. IEEE Comp. Networking Symp., Washington D. C., April 1988, pp.134-142. Also available from: http://www.cis.ohio-state.edu/~jain/papers/cr1.htm.

[KP87] Karn P. and Partridge C., *Improving Round-Trip Time Estimates in Reliable Transport Protocols*, Computer Communication Review, vol.17. no.5, pp. 2-7, August 1987.

[KRA96] Kojo M., Raatikainen K. and Alanko T., *Connecting mobile workstations to the Internet over a digital cellular telephone network*, Workshop on Mobile and Wireless Information systems MOBIDATA, November 1996, Rutgens Univ. NJ. Also available from: http://citeseer.nj.nec.com/kojo94connecting.html.

[Larry02] Z. Larry, *Adaptation of RED Parameters,* Studies on RED Parameters available from: http://ee.tamu.edu/~zzlarry/vprm_red.html.

[Lud99] Ludwig R., *A case for Flow - Adaptive Wireless Links,* Technical Report UCB //CSD-99-1053, May 1999. Also available from: http://iceberg.cs.berkeley.edu/papers/Ludwig-FlowAdaptive/flowadaptive.pdf.

[LK00] Ludwig R. and Katz R. H., *The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions,* ACM Computer Communication Review, 30(1), January 2000. Also available from: http://www.acm.org/sigcomm/ccr/archive/2000/jan00/ccr-200001-ludwig.html.

[LM97] Lin D. and Morris R., *Dynamics of Random Early Detection, SIGCOMM'97.* Also available from: http://citeseer.nj.nec.com/lin97dynamics.html.

[LNO96] Lakshman T. V., Neidhardt A. and Ott T., *The Drop From Front Strategy in TCP Over ATM and Its Interworking with Other Control Features,* Infocom96.

[LS00] Ludwig R. and Sklower K., *The Eifel Retransmission Timer, July2000.* (http://www.acm.org/sigcomm/ccr/archive/2000/july00/LudwigFinal.pdf).

[MDKM00] Montenegro G., Dawkins S., Kojo M., Magret V., and N. Vaidya, *Long Thin Networks*, RFC 2757, January 2000.

[MMFR96] Mathis M., Mahdavi J., Floyd S. and Romanow A., *TCP Selective Acknowledgments Options,* RFC 2018, October 1996.

[Nag84] J. Nagle, *Congestion control in IP/TCP internetworks,* RFC 896, January 1984.

[NBBB98] Nichols K., Blake S., Baker F. and D. Black, *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*, RFC 2474, December 1998.

[OLW99] Ott T. J. and Lakshman T.V., *SRED: Stabilized RED*, 1999. Also available from: http://citeseer.nj.nec.com/ott99sred.html.

[Pen00] Pentikousis K., *TCP in Wired-Cum-Wireless Environments,* IEEECommunications Surveys, Fourth Quarter 2000. Also available from: http://www.comsoc.org/pubs/surveys.

[Pos80] Postel J., *User Datagram Protocol,* RFC 768, August 1980.

[Pos81a] Postel J. ed., *Internet Protocol,* RFC 791, September 1981.

[Pos81b] Postel J. ed., *Transmission Control Protocol,* RFC 793, September 1981.

[PPP00] Pan R., Prabhakar B. and Psounis K., *CHOKe, A stateless active queue management scheme for approximating fair bandwidth allocation,* IEEE INFOCOM, 2000. Available from http://citeseer.nj.nec.com/context/1608007/0

[RBL99] Rosolen V., Bonaventure O. and Leduc G, *A RED discard strategy for ATM networks and its performance evaluation with TCP/IP traffic,* 1999. Available from: http://citeseer.nj.nec.com/rosolen99red.html.

[RFB01] Ramakrishnan K., Floyd S. and Black D., *The Addition of Explicit Congestion Notification (ECN) to IP*, RFC 3168, September 2001.

[Sinha99] Sinha P. et al., *WTCP: A Reliable Transport Protocol for Wireless Wide-area Networks,* Proc. ACM MOBICOM 99 Seattle, Washington, Aug. 1999.

[Sti90] R. H. Stine, *FYI on a network management tool catalog: Tools for monitoring and debugging TCP/IP internets and interconnected devices,* RFC 1147, April 1990.

[SCFJ96] Schulzrinne H., Casner S., Frederick R. and Jacobson V., *RTP: A Transport Protocol for Real-Time Applications,* RFC 1889, January 1996.

[SL97] Stamoulis A. and Liebeherr J., *GPS: Slow-Start Generalized Processor Sharing*, 1997. Available from: http://citeseer.nj.nec.com/stamoulis97gps.html.

[SV94] Shreedehar M. and Varghese G., *Efficient Fair Queuing using Deficit Round Robin,* November 1994. Also available from: http://citeseer.nj.nec.com/shreedhar95efficient.html.

[Tab00] Tabbane S., *Handbook of Mobile Radio Networks,* Artech House Mobile Communications Library, Norwood, MA 2000.

[Tisal98] Tisal J., *GSM Cellular Radio Telephony,* John Wiley & Sons, West Sussex, England, 1998.

[TB00] Tsaoussidis V. and Badr H., *TCP-Probing: Toward an Error Control Schema with energy and Throughput Performance Gains*, Proc. 8th IEEE Conf. Network Protocols, Japan, Nov. 2000.

[TBV00] Tsaoussidis V., Badr H. and Verma R., *Wave and Wait Protocol (WWP): Low Energy, High Throughput for Mobile IP Devices*, Proc. 8th IEEE Conf. Networks, Sept. 2000.

[TTCP98] *TTCP.c (Test TCP connection)*, May 1998 Also available from http://www.netcraftsmen.net/id44.htm.

[Vai99] Vaidya N., *Tutorial on TCP for Wireless and mobile host*, 1999. Also available from: http://csweb1.cs.tamu.edu/faculty/vaidya/seminars/tcp-tutorial-aug99.ppt.

[WAP01] *Wireless Application Protocol Forum* (www.wapforum.org), April 2001.

[WW02] Walter U. and Wehrle K., *Traffic Priorization and Differentiation with Active Queue Management*, 2002 Also available from: http://www.tm.uka.de/~walter/papers/walter_wehrle_ICTSM2002.pdf.

[Zhang00] Zhang Y., *The implication of End-to-End IPSEC,* Network Working Group INTERNET DRAFT, March 2000.

[ZPBS02] Zhang Y., Breslau L., Paxson V. and S. Shenker, *On the Characteristics and Origin of Internet Flow Rates*, Proceedings of Sigcomm 2002. Available from: http://www.research.att.com/projects/T-RAT/index.html.

7 Appendix A

Riassunto in Italiano

In questa Appendice riportiamo un breve estratto in lingua italiana dell'intera tesi. Il presente lavoro è stato portato a termine presso l'Università di Helsinki sotto la guida del Prof. Kimmo Raatikainen.

1. Introduzione

Meccanismi end-to-end di controllo di congestione sono ampiamente impiegati nell'attuale Internet col fine di regolare la quantita' di traffico immessa nella rete e evitare il collasso della rete stessa. Gran parte di queste tecniche si basano sul meccanismo di controllo di congestione del protocollo TCP [Pos81b] . Il successo di Internet e la sua rapida diffusione sono soprattutto dovuti all'efficienza di questi meccanismi, ma una serie di fattori quali la necessita' di nuovi servizi e di migliori performance hanno portato all'introduzione di QoS in Internet e l'applicazione di Internet ad ambiente wireless e non piu' solo wired ha evidenziato l'inadeguadezza delle attuali tecniche di controllo di congestione. Il problema principale che caratterizza il TCP e' che lo stato di congestione della rete e' basato sulla perdita di pachetti: se la sorgente rivela perdita di pacchetti riduce il proprio rate di trasmissione al fine di evitare la perdita di ulteriori pacchetti. Tuttavia questo meccanismo puo' richiedere lunghi tempi e molti pacchetti possono essere persi. Inoltre i comuni routers utilizzati in Internet impiegano discipline quali Tail Drop per controllare l'occupazione della code nel buffer. Quando i paccchetti in coda raggiungono una soglia prestabilita i successivi pacchetti vengono scartati. Questo modo di procedere consente di mantenere un'elevata occupazione del buffer, ma allo stesso tempo introduce lunghi ritardi, tende a scartare burst di pachetti e a risultare in fenomeni quali sincronizzazione delle sorgenti.

Tutti questi problemi hanno condotto l'Internet Engineering Task Force ad ideare nuove tecniche per il controllo di congestione come *Active Queue Managment (AQM)* [BCCD98] e

Explicit Congestion Notification (ECN) [RFB01]. L' obiettivo di AQM e' rilevare lo stato di congestione prima che la coda nel buffer saturi e segnalarlo agli hosts. La notificazione puo' essere fatta scartando pacchetti o marcandoli attraverso il Congestion Experienced (CE) codepoint. Un algoritmo di AQM che ha riscosso notevole interesse e' *Random Early Detection (RED)* [FJ93]. Un RED gateway inizia a scartare pachetti prima che la coda del buffer saturi e la decisone se scartare o meni un paccchetto e' basata essenzialmente sulla lunghezza media della coda nel buffer. Piu' la lunghezza media aumenta piu' facilmente verranno scartati i pachetti in arrivo.

In questa tesi presentiamo un'analisi delle performance di RED in ambiente wired. E' uno sudio volto principalmente a verificare e criticare gli attesi benefici apportati dall'introduzione di RED e i miglioramenti ripetto al tradizionale Tail Drop. Il traffico utilizzato e' prevalentemente TCP e in alcuni test UDP. Inoltre una parte dei test affronta l'applicazione di RED ad un ambiente con differenziazione di servizio.

2. Conoscenza di base

Il protocollo di controllo della trasmissione (*Transmission Control Protocol*: TCP) garantisce un trasporto affidabile dei dati e la consegna dei pacchetti in ordine e senza errori. L'unita' di traferimento fondamentale impiegata dal TCP e' il segmento. I segmenti sono utilizzati per passare informazioni di controllo, quali ad esempio instaurazione o chiusura della connessione, o dati. Il ricevitore invia risontri (ACK) per confermare l'avvenuta ricezione di un pacchetto. Il TCP fornisce controllo d'errore basato sul meccanismo di ritrasmissione go-back-n ed implementa il controllo di flusso facendo annunciare al ricevitore la quantita' di dati che e' disposto ad accettare. Il controllo di congestione e' window-based. Il transmettitore e' regolato da una *congestion window* (cwnd) che limita la quantita' di dati che puo' avere nella rete e non ancora riscontrati. cwnd varia in relazione agli eventi che osserva. Due algoritmi regolano la variazione di cwnd e sono noti come Slow Start e Congestion Avoidance. L'algoritmo Slow Start si basa sull'osservazione che il ritmo con cui i pacchetti sono immessi nella rete dovrebbe eguagliare quello con cui sono riscontrati dal ricevitore. Nella fase di Slow Start cwnd viene incrementata di SMSS (sender maximum segment size) bytes ogni volta che un pacchetto viene riscontrato. Lo Slow Start termina in caso venga rilevata congestione e quindi una perdita di pacchetto. Se cio' e' stato dovuto allo scadere del timeout cwnd e' settata a 1; se e' stato causato dalla ricezione di tre DUPACKs consecutivi si entra nell fase di Fast Retransmit/ Fast Recovery che regola la ritrasmissione dei pacchetti persi. Oppure lo Slow Start puo' terminare quando cwnd raggiunge una soglia massima prefissata. In questo caso si entra nella fase di

Congestion Avoidance secondo cui cwnd viene incrementata di SMSS bytes dopo che tutti i pacchetti contenuti nella cwnd sono stati riscontrati.

L'evoluzione di Internet e' stata acocmpagnata dallo sviluppo di nuove applicazioni come quelle basate su audio e video con specifi requisiti. Lo scopo della differenziazione dei servizi e' soddisfare applicazioni eterogenee e provvedere uno specifico livello dei servizi per ciascuna. Cio' ha condotto all'ideazione di due principali architetture: l' *Integrated Services Architecture* [BCS94] e la *Differentiated Services Archtecture* [BBCD98]. La prima alloca risorse per ogni richiesta di servizio e le mantiene per l'intera durata della connessione. Sebbene sia una struttrua accurata non e' stata poi effettivamente realizzata per la sua eccessiva complessita' e mancanza di scalabilita'. Il secondo tipo di architettura invece affronta la complessita' del problema decomponendolo in singole funzioni piu' semplici. I due elementi base di questa struttura sono un meccanismo che monitora l'ingresso dei pacchetti nella rete e uno che cura l'elaborazione dei pacchetti all'interno della rete stessa. Il traffico viene classificato e assegnato ad un particolare per-hop-behaviour identificato da un codice contenuto nel campo DS dell'header IP. I nodi interni alla rete selezionano il servizio appropriato in base al valore contenuto in questo campo. In tal modo l'elaborazione dei pacchetti all'interno della rete e' semplificata e consente una maggiore scalabilita'.

Il procollo TCP fu inizialmente pensato per reti wired, ma le reti wireless stanno assumendo sempre piu' importanza nel futuro di Internet. Il problema e' che le reti wireless presentano caratteristiche assai differenti da quelle wired come per esempio banda limitata, lunghi tempi di propagazione, ritardi variabili, alta probabilita' d'errore. Il controllo di congestione del TCP is basa interamente sulla perdita di pachetti come segnale di congestione. Ogni volta che un pacchetto e' perso il TCP riduce la finestra di trasmissione, ma in presenza di una rete wireless la perdita di un pacchetto puo' essere dovuta a diversi aspetti e soprattutto a probabilita' di errore sul link molto elevate. La riduzione continua della finestra di congestione implica throughput ridotti e performance degradate. Per ovviare a cio' sono state sviluppate soluzioni a livello di link o modifiche al procollo TCP (AQM, ECN, ELN [BK98], SACK option [FMMP00]...) o proposte di nuovi protocolli di transporto specifici per wireless links.

3. Active Queue Management

Tipicamente i router per gestire la lunghezza della coda nei buffer fissano una lunghezza massima per ogni coda e accettano pacchetti fino a che la coda non raggiunge il limite massimo; poi rifiutano i successivi pacchetti fino a che la lunghezza della coda non decresce dopo la trasmissione di un paccheto nell coda. Tale tecnica e' conosciuta col nome di *Tail*

Drop. Sebbene sia stata impiegata per anni in Internet presenta due importanti svantaggi: permette fenomeni come Lock-Out in cui una sorgente puo' arrivare a monopolizzare tutte le risorse disponibili e tende a mantenere le code piene introducendo cosi' alti ritardi e riuscendo ad accomodare con diffcolta' traffici a burst. A fronte di cio' *Active Queue Management (AQM)* e' stato proposto. L'idea base e' che tali problemi possano essere risolti iniziando a scartare (o marcare) i pacchetti prima che la coda nel buffer saturi, nonostante vi sia ancora spazio disponibile. In particolare l'IETF ha raccomandato *Random Early Detection (RED)* come forma di AQM da implementare nelle rete [BCCD98]. Il suo obiettivo e' controllare l'occupazione media del buffer cosi' da garantire i seguenti benefici:

1. Prevenire la congestione
2. Minimizzare il numero di pacchetti persi e il ritardo di coda
3. Evitare fenomeni di sincronizzazione delle sorgenti
4. Mantenere un'alta utilizzazione del link e massimizzare il throughput
5. Garantire *fairness* ed evitare discriminazioni contro traffici a burst
6. Essere applicabile nelle sitauzioni piu' svariate, a un numero di connessioni variabile con diversi round trip times, carichi e throughput.

L'algoritmo RED, come spiegato in [FJ93], monitora la lunghzza della coda nel buffer e la decisione di scartare un pacchetto in arrivo e' basata sulla stima della lunghezza media della coda, *avg*: piu' e' grande *avg* e piu' facilmente verrano scartati pacchetti. Vengono fissate due soglie sulla lunghezza della coda, una minima (min_{th}) e una massima (max_{th}). Fino a che *avg* si mantiene al di sotto della soglia minima i pacchetti vengono accettati. Quando min_{th} viene superata RED scarta i pacchetti in maniera probabilistica e infine se *avg* supera la soglia massima tutti i pacchetti in arrivo vengono rifiutati. La probabilita' di rifiuto e' tale che se recentemente la coda e' stata prevalentemente vuota diffcilmente verranno scartatiti i pacchetti in arrivo. Al contrario se e' stata per lo piu' piena, piu' facilemente RED scartera' i nuovi pacchetti. In questo modo RED puo' controllare l'occupazione media del buffer e prevenire la congestione in maniera piu' sicura di quanto Tail Drop riesca a fare. Il funzionamento dell'algoritmo dipende da alcuni parametri quali ad esempio min_{th} e max_{th} di cui non esistono dei precisi valori di default che possano garantire performance ottimali in ciasun tipo di scenario. Per questo motivo e per altre imperferzioni risontrate in RED sono state proposte innumerevoli variazioni dell'algoritmo base, anche se spesso si tratta di soluzioni assai complicate. Sono ad esempio Adaptive RED [FKSS99], FRED [LM97], CHOKe [PPP00], etc. Infine ulteriori miglioramenti potrebbero derivare per RED dall'utilizzo di *Explicit Congestion Notification (ECN)*[RFB01]. Anziche' scartare pacchetti i routers possono segnalare il pericolo

di congestione in modo piu' efficiente marcando i pacchetti attraverso l'utilizzo del campo ECN nell'header IP e evitare cosi' lo spreco di risorse causato dalla perdita di pacchetti.

4. Organizzazione dei tests

Lo scopo dei tests condotti e' sviluppare un'analisi dell' algoritmo RED e studiare come possa risolvere problemi legati al controllo di congestione del TCP. Utilizziamo un testbed costituito da quattro macchine comprendente due hosts comunicanti tramite due routers. Sull'ultimo link e' simulato un collo di bottiglia e il router che lo precede implementa alternativamente Tail Drop o RED al fine di permettere un confronto tra le due discipline. Nel secondo router utilizziamo una discplina di coda nota come HTB (Hierachical Token Bucket) che permette di limitare la banda sul link d'uscita e specificare classi di servizio. La versione del TCP utilizzata comprende le raccomandazioni base e in particolare e' abilitata l'opzione *SACK* [MMFR96] ed implementato l'algoritmo *New Reno* [FH99]. Le principali metriche utilizzate nella valutazione delle performance ottenute comprendono: tempo trascorso, tempo di trasmissione, durata della three-way hadshake, throughput, fairness (Jari fariness index [Jai91]) e numero di pacchetti ritrasmessi. Ogni test si absa su 20 repliceh e vengono calolate statistiche quali Min, Max, Mediana, Media, 25%Perc., 75'%Perc.e Deviazione standard.

I tests sono suddivisi in due categorie. Nella prima parte utilizziamo tre connessioni TCP. Due di queste iniziano a trasmettere simultaneamente e inviano un carico pari a 292Kbytes mentre la terza porta un carico di 47.7Kbytes e ha un ritardo variabile, compreso tra 0 e 7 secondi. Al fine di investigare meglio come l'inizio della terza connessione possa influenzare le perfomance dei flussi iniziali abbiamo analizzato differenti scenari di studio che si differenziano per il ritardo imposto al terzo flusso. Abbiamo esaminato 12 scenari per Tail Drop e di questi i 5 piu' interessanti sono stati ripetuti nelle medisme condizioni implementando RED. I risultati cosi' ottenuti sono poi stati confrontati. Il last-hop link ha una banda di 140Kbits. La seconda categoria di test utilizza una connessione TCP e un flusso UDP. Il traffico UDP inizia a trasmettere con un ritardo variabile. Scopo di questi tests e' indagare come le prestazioni della connessione TCP vengano degradate dall'arrivo di un flusso UDP. Nella prima fase dei tests il flusso UDP in arrivo viene servito dalla stessa classe di servizio del traffico TCP. Nella seconda fase al traffico UDP e' assegnata una priorita' maggiore e accomodato in una differente classe di servizio in grado di garantire la consegna del traffico UDP senza alcuna perdita di pacchetti. Inoltre utilizziamo tre diversi workloads. I primi due convogliano traffci TCP e UDP di pari dimensioni e tali che il traffico UDP duri per l'intera connesione TCP. I traffci TCP e UDP inviati sono rispettivamente 391.3Kbytes e 264.1Kbytes

nel primo test case e 46.7Kbytes e 39.7Kbytes nel secondo. Infine il terzo test case e' caratterizzato da un flusso TCP di 391.3Kytes e un traffico UDP di 33.1Kbytes di durata assai inferiore rispetto alla connesione TCP. Ogni esperimento e' condotto prima in presenza di Tail Drop e poi con RED e i risultati sono poi confrontati.

5. Analisi dei risultati dei tests

L'analisi e' suddivisa in due sezioni. Il primo gurppo di test condotti e' volto allo studio delle performance di tre flussi TCP che competono per acquisire banda sullo stesso link di limitata capacita'. I risultati sono presentati in forma di confronto tra le prestazioni realizzate quando Tail Drop o RED sono impiegati.

- RED riesce a garantire maggior fairness ossia le risorse disponibili vengono quasi equamente suddivise tra le tre connesioni in competizione, indipendentemente dal loro carico e dal loro momento d'arrivo. Tail Drop invece favorisce le connessioni iniziali e a volte degenera in fenomeni come Lock-Out in cui si assiste ad una monopolizzazione completa delle risorse da parte dei flussi inizali. La terza connesione inizia la trasmissione quando i due flussi iniziali stanno chiudendo la connnesione.
- Per quanto risguarda le prestazioni del terzo flusso RED garantisce risultati migliori e soprattutto piu' stabili e indipendenti dall'istante in cui subentra l'ultimo flusso. Tail Drop invece fa registrare prestazioni molto instabili. Il terzo flusso raggiunge i risultati migliori quando e' lanciato simultaneamente alle connesioni iniziali o quando queste sono in Fast Retransmit o Fast Recovery. Ossia quando la rete e' congestionata e le connesioni sono in fase di blocco. Al cotrario quando il terzo flusso inizia a trasmettere e trova i due flussi inziali in Slow Start viene fortemente danneggiato e molti dei suoi pacchetti vengono scartati. Da notare il fatto che nonostante le prestazioni del terzo flusso sono piu' soddisafcenti quando RED e' implementato le performance dei flussi inziali sono le stesse per entrambe le discipline.
- Infine un obiettivo di RED spesso citato e' la minimizzazione dei pacchetti persi nella rete. Sorprendentemente nella quasi totalita' dei test da noi condotti RED ritrasmette piu' paccchettti di quanto Tail Drop faccia. Tuttavia cio' che e' importante notare e' che i tre flussi ritrasmettono un numero di pacchetti all'incirca proporzionale al carico di dati che immettono nella rete. Con Tail Drop il terzo flusso pur trasportando un carico assai inferiore (4/25 di quello delle connessioni iniziali) subisce lo stesso numero di perdite dei flussi iniziali a conferma della non-fairess di Tail Drop.

Il secondo gruppo di test introduce anche traffico UDP e analizza l'applicazione di RED ad un ambiente con differenziazione dei servizi. Quando la rete e' caricata con grandi traffici

TCP, RED riesce ad assicurare performance migliori al traffico UDP, specialmente se di breve durata. Come gia' riscontrato prima, al contrario di Tail Drop i risultati di RED per entrambi i traffici sono molto piu' stabili e non influenzati dal momento di arrivo del flusso UDP. Tail Drop tende a favorirer il traffico TCP e l'introduzione di differenziazione dei servizi conduce ad un evidente degrado delle performance. RED serve i traffici con maggior equita' e l'utilizzazione di banda di ciascuno e' all'incirca proprozionale al carico di dati immesso nella rete. Il numero di pacchetti ritrasmessi con RED e' ancora molto elevato. Tuttavia in alcuni casi particolarmente sfavorevoli, RED iniziando a scartare paccchetti molto prima di quanto Tail Drop faccia, riesce a reagire piu' efficientemente e in tempi assai ridotti alla pedita di pacchetti e a situazioni di grave congestione. Inoltre per la stessa ragione il traffico UDP che generalmente tende a disturbare il traffico TCP, quando e' lanciato contemporaneamente alla connesione TCP aiuta a migliorare le performance di Tail Drop. Infatti il traffico UDP in competizione fin dall'inzio col flusso TCP previene la fase di Slow Start dall'essere eccessivamente aggressiva e ad incorrere in fenomeni quali lo Slow Start overshoot. In tal modo le perfomance finali sono migliori.

6. Conclusioni

Scopo di questa tesi e' analizzare le performance dell'algortimo di active queue management RED e verificare quanto possa essere efficace come meccanismo di controllo di congestione. In partcolare e' nostro intento verificare ed eventualmente criticare i benefici promessi dall'impiego si RED rispetto a discipline di coda tradizionali come Tail Drop. Inoltre il nostro sudio comprende l'applicazione di RED ad un ambiente con differenziazione di servizio.

I principali risultati ottenuti sono stati descritti nella sezione precedente. La maggior parte delle proprieta' teoriche di RED hanno trovato conferma nei nostri esperimenti anche se occorre notare che la metodologia secondo cui i test sono stati organizzati e condotti ha un impatto rilevante sull'utilita' dei risultati ottenuti e sulle conclusioni derivate. Per esempio utilizzando un worlkoad comprendente un numero maggiore di connessioni TCP e eventualmente non TCP, con diverse carattesitiche in termini di RTT e throughput potrebbe permetterci di investigare meglio alcuni aspetti. In particolare nonostante molta ricerca sia stata condotta su RED ancora vi sono punti oscuri e primo fa tutti il settaggio dei parametri di RED. Il problema e' che e' difficile trovare dei parametri ottimali in ogni circostanza e per ogni tipo di traffico. Approci piu'adattativi verranno sempre piu' presi in considerazione. Inoltre cio' che pare essere particolarmente promettente per il futuro e' un uso combinato di RED e ECN.

8 Appendix B

Full Test Results

This appendix describes the summarized test results for some test runs. Each scenario is replicated 20 times. The tables reported refer to the two test cases presented: the tests with *three TCP competing flows* and those with *TCP and UDP traffic in a DiffServ architecture.* There are two different kinds of tables, one for each type of test run.

8.1 Tests with three competing TCP flows

As regards this set of experiments and reports statistics about the TCP performance: Elapsed Time (time elapsed between the SYN message till the ACK FIN message), Data Transmit Time (time elapsed between the first data packet sent till the last one), Throughput, Number of Retransmit Data expressed in packets and bytes, number of Duplicate ACKs and Triple ACKs. We give three different percentiles, first quartile (25% percentile), median (50% percentile) and third quartile (75% percentile) and statistics about Min and Max values, Average and Standard Deviation. These tables present the results of the scenarios expressed in terms of each single replication or classifying the flows as *fastest*, *slowest* and *third* flow and reporting the related statistics. The scenarios considered are *after 0 seconds, after 2 seconds* and *after 5.5 seconds.* They can refer to Tail Drop or Red scenarios. In Table 8-2 we have outlined in red the so-called *exception,* the replication in which the third flow lasts more than the initial ones.

	N. Replication	Data xmit time (sec)	Elapsed time (sec)	Throughput (bytes/s)	Retx data packets	Retx Data bytes	Duplicate acks	Triple dupacks
1	fastest	35,888	36,744319	7947	8	11584	34	4
	slowest	36,745	36,906824	7912	18	24073	40	6
	3rd flow	9,988	12,461677	3749	5	7240	6	1
2	fastest	31,613	32,633968	8948	5	7240	39	5
	slowest	35,628	36,6409	7969	7	10136	29	4
	3rd flow	5,259	6,430064	7266	0	0	1	0
3	fastest	34,903	36,084212	8092	5	7240	43	5
	slowest	35,419	36,551463	7989	5	7240	34	4
	3rd flow	18,078	19,428297	2405	5	7240	9	2
4	fastest	35,243	36,020963	8106	6	8193	37	4
	slowest	35,852	36,672244	7962	6	8688	36	5
	3rd flow	10,153	11,327991	4124	2	2896	4	1
5	fastest	28,619	29,461292	9911	4	5792	29	4
	slowest	35,712	36,70137	7956	7	10136	30	5
	3rd flow	12,478	13,817236	3381	4	4729	8	2
6	fastest	34,901	36,079692	8093	5	7240	39	5
	slowest	35,581	36,562574	7986	8	11584	37	5
	3rd flow	7,281	8,627645	5415	2	2896	5	1
7	fastest	34,205	35,009046	8341	6	8193	28	3
	slowest	35,831	36,670533	7963	6	8688	33	5
	3rd flow	15,314	16,216206	2881	4	4729	4	1
8	fastest	33,89	35,247809	8284	4	5792	25	3
	slowest	35,9	36,571345	7984	6	8688	44	5
	3rd flow	18,249	19,426125	2405	5	7240	9	2
9	fastest	34,289	35,297612	8273	6	8688	37	4
	slowest	36,099	36,60045	7978	6	8688	39	5
	3rd flow	10,418	11,42587	4089	2	2896	5	1
10	fastest	35,409	36,588987	7981	5	7240	29	4
	slowest	37,486	37,655179	7755	7	9641	33	4
	3rd flow	10,151	11,324617	4126	2	2896	4	1
11	fastest	36,058	36,556018	7988	6	8688	39	5
	slowest	37,514	37,518128	7783	6	6897	34	5
	3rd flow	8,977	9,984105	4679	4	4729	8	2
12	fastest	28,619	29,457231	9913	4	5792	29	4
	slowest	35,708	36,697306	7957	7	10136	30	5
	3rd flow	12,468	13,813483	3382	4	4729	8	2
13	fastest	31,277	32,458567	8996	4	5792	35	4
	slowest	35,628	36,634624	7971	6	8688	28	5
	3rd flow	8,388	9,674002	4829	4	5792	14	2
14	fastest	35,307	36,487492	8003	5	7240	40	5
	slowest	35,339	36,594557	7979	5	7240	36	4
	3rd flow	15,208	15,883888	2941	5	6177	8	2
15	fastest	34,668	35,683463	8183	5	7240	34	4
	slowest	35,876	36,547712	7990	6	8688	36	5
	3rd flow	12,318	13,662439	3420	3	4344	6	1
16	fastest	34,998	36,344344	8034	6	8200	40	5
	slowest	35,676	36,685059	7960	7	8248	40	6
	3rd flow	7,288	8,631765	5413	2	2896	5	1
17	fastest	35,39	36,532968	7993	5	7240	35	4
	slowest	38,174	38,177888	7648	6	8057	37	5
	3rd flow	15,209	15,882466	2942	5	6177	8	2
18	fastest	28,619	29,454218	9914	4	5792	29	4
	slowest	35,655	36,674246	7962	7	10136	30	5
	3rd flow	12,468	13,8103	3383	4	4729	8	2
19	fastest	34,901	36,072245	8095	5	7240	39	5
	slowest	35,574	36,555105	7988	8	11584	37	5
	3rd flow	7,281	8,620205	5420	2	2896	5	1
20	fastest	34,023	34,494103	8465	7	9553	26	4
	slowest	35,649	36,65952	7965	5	7240	35	5
	3rd flow	11,488	12,599145	3708	3	4344	10	2

Table 8-1. 20 Replications of the scenario *after 0 seconds*. Three TCP competing flows when Tail Drop discipline is employed.

	N. Replication	Data xmit time (sec)	Elapsed time (sec)	Throughput (bytes/s)	Retx data packets	Retx Data bytes	Duplicate acks	Triple dupacks
1	fastest	31,702	33,803506	8638	17	24616	62	7
	slowest	35,997	36,838409	7927	8	11584	31	4
	3rd flow	26,774	31,338155	1491	6	6178	11	3
2	fastest	31,527	32,027654	9117	6	7569	32	4
	slowest	35,69	36,694088	7958	5	7240	34	5
	3rd flow	14,009	15,909196	2937	4	5792	6	1
3	fastest	27,937	28,918784	10097	5	7240	32	4
	slowest	35,45	36,624408	7973	4	5792	29	4
	3rd flow	28,729	30,788402	1517	7	10136	7	2
4	fastest	27,026	28,207221	10352	3	4344	29	3
	slowest	35,42	36,604227	7977	6	8688	25	3
	3rd flow	10,75	15,277952	3058	5	4730	5	1
5	fastest	30,745	31,53635	9259	5	7240	33	4
	slowest	35,76	36,613057	7975	5	7240	36	5
	3rd flow	14,089	15,967141	2926	4	5792	6	1
6	fastest	35,687	36,528098	7994	6	8688	35	5
	slowest	35,758	36,634575	7971	7	9641	36	4
	3rd flow	22,886	24,955195	1872	7	10136	11	3
7	fastest	27,934	28,915723	10098	5	7240	32	4
	slowest	35,58	36,592006	7980	4	5792	30	4
	3rd flow	28,729	30,785429	1518	7	10136	7	2
8	fastest	32,579	33,444718	8731	4	4680	32	4
	slowest	35,789	36,63052	7971	6	8688	32	4
	3rd flow	20,149	22,374858	2088	5	7240	7	2
9	fastest	33,569	34,911043	8364	6	8688	27	3
	slowest	35,902	36,583513	7982	7	10136	33	4
	3rd flow	21,629	23,853282	1959	5	7240	8	2
10	fastest	34,679	35,509502	8223	6	8193	29	4
	slowest	35,822	36,671519	7963	7	10136	32	4
	3rd flow	19,179	21,182387	2206	5	7240	10	2
11	fastest	30,761	31,911468	9150	7	10136	35	4
	slowest	35,738	36,579386	7983	7	10136	30	4
	3rd flow	16,619	18,521944	2522	5	6177	1	0
12	fastest	34,25	35,608213	8200	5	7240	35	4
	slowest	35,92	36,59134	7980	8	11584	32	4
	3rd flow	18,259	22,962404	2035	5	5793	4	1
13	fastest	28,3	29,480152	9905	4	5792	29	3
	slowest	32,929	33,946282	8602	4	5792	30	4
	3rd flow	37,959	39,660983	1178	5	7240	1	0
14	fastest	33,059	33,750153	8652	6	8688	41	5
	slowest	35,609	36,616593	7975	6	8688	33	4
	3rd flow	8,646	10,390237	4497	1	385	0	0
15	fastest	31,478	32,449442	8999	5	7240	36	4
	slowest	35,66	36,665	7964	6	8688	37	5
	3rd flow	22,955	24,969772	1871	6	7625	6	1
16	fastest	35,057	36,238294	8058	7	10136	35	4
	slowest	35,364	36,584929	7981	5	7240	31	4
	3rd flow	17,369	19,428886	2405	4	5792	5	1
17	fastest	34,627	35,467089	8233	6	8688	39	5
	slowest	35,752	36,594716	7979	5	7240	42	5
	3rd flow	21,499	23,72819	1969	6	8688	9	2
18	fastest	27,936	28,917879	10098	5	7240	32	4
	slowest	35,45	36,623482	7973	4	5792	29	4
	3rd flow	28,729	30,787417	1518	7	10136	7	2
19	fastest	32,625	33,41663	8738	5	7240	36	5
	slowest	35,62	36,622707	7973	6	8392	30	4
	3rd flow	6,76	11,956918	3907	2	1449	7	1
20	fastest	31,475	32,445841	9000	5	7240	36	4
	slowest	35,66	36,661178	7965	6	8688	37	5
	3rd flow	22,959	24,965493	1871	6	7625	6	1

Table 8-2. Three TCP flows when Tail Drop is employed. The third flow starts after 2 seconds.

		Data xmit time (sec)	Elapsed time (sec)	Throughput (bytes/s)	Retx data packets	Retx Data bytes	Duplicate acks	Triple dupacks
fastest flow	Min	27,03	28,21	7994	3	4344	27	3
	Max	35,69	36,53	10352	17	24616	62	7
	25%Perc.	30,13	31,02	8331,25	5,00	7240	32	4
	Median	31,61	32,93	8868,50	5	7240	34	4
	75%Perc	33,74	35,05	9420,50	6	8688,00	36	4,25
	Average	30,95	32,02	8637,30	5,85	8334,50	33,55	4,05
	Stdev	2,67	2,67	757,19	2,79	4090,47	7,27	0,89
slowest flow	Min	32,93	33,95	7927	4	5792	25	3
	Max	36,00	36,84	8602	8	11584	42	5
	25%Perc.	35,55	36,59	7969,50	5	7240,00	30,00	4
	Median	35,68	36,62	7974	6	8688	32,00	4
	75%Perc	35,77	36,64	7980,00	7	9764,75	34,5	4,25
	Average	35,29	36,24	8066,80	5,8	8358,85	32,30	4,15
	Stdev	0,64	0,60	141,63	1,28	1832,84	3,76	0,52
third flow	Min	6,76	10,39	1178	1	385	0	0
	Max	37,96	39,66	4497	7	10136	11	3
	25%Perc.	15,99	17,88	1782,75	4,75	5792	5	1
	Median	20,82	23,35	2002,00	5	7240	6,5	1
	75%Perc	23,91	26,42	2623,00	6,00	7890,75	7,25	2
	Average	20,43	22,99	2267,25	5,10	6776,50	6,20	1,40
	Stdev	7,63	7,93	1520,23	1,59	2635,68	7,52	1,19

Table 8-3. The statistics for the scenario *after 2 seconds* when Tail Drop is in use.

		Data xmit time (sec)	Elapsed time (sec)	Throughput (bytes/s)	Retx data packets	Retx Data bytes	Duplicate acks	Triple dupacks
fastest flow	Min	28,50	29,18	8004	8	11097	41	6
	Max	35,80	36,48	10007	25	36200	71	11
	25%Perc.	30,59	31,28	8437,00	10,00	14480	46,5	7
	Median	32,20	33,22	8798,50	10,5	15204	51	8
	75%Perc	33,77	34,61	9333,75	12	17376,00	52,75	9
	Average	31,13	31,96	8664,80	11,1	16048,45	49,05	7,95
	Stdev	2,25	2,30	629,06	3,56	5182,42	7,69	1,40
slowest flow	Min	35,79	36,53	7789	12	17376	41	6
	Max	37,32	37,49	7994	19	27512	61	11
	25%Perc.	35,96	36,60	7952,25	13	18824,00	46,50	8
	Median	36,12	36,66	7965,5	14	20052,5	51,00	9
	75%Perc	36,27	36,72	7978,00	15	21720	55	10
	Average	35,94	36,47	8015,75	14	20186,85	50,00	8,7
	Stdev	0,42	0,27	57,41	1,73	2560,99	5,32	1,33
third flow	Min	5,59	7,30	2825	1	1448	3	0
	Max	13,74	16,54	6396	7	10136	16	3
	25%Perc.	8,34	9,49	3763,50	3	4344	10	2
	Median	9,88	11,51	4060,00	4	4393	11	2
	75%Perc	10,36	12,42	4925,50	4,00	5792	13,25	3
	Average	9,49	11,38	4307,85	3,70	5087,35	11,05	2,10
	Stdev	2,06	6,20	1266,07	2,60	3759,65	7,47	1,22

Table 8-4. The statistics for the scenario *after 2 seconds* when RED is in use.

	N. Replication	Data xmit time (sec)	Elapsed time (sec)	Throughput (bytes/s)	Retx data packets	Retx Data bytes	Duplicate acks	Triple dupacks
1	fastest	27,152	28,63236	10198	14	20272	51	5
	slowest	35,997	36,837938	7927	9	13032	29	4
	3rd flow	21,019	25,987692	1798	4	4345	3	0
2	fastest	29,754	30,786652	9485	6	8688	30	5
	slowest	35,49	36,662302	7965	5	7240	26	4
	3rd flow	9,969	21,05688	2219	5	4346	9	2
3	fastest	35,024	36,204573	8065	6	7576	41	5
	slowest	35,53	36,651537	7967	6	8688	36	5
	3rd flow	7,769	9,996151	4674	2	2896	5	1
4	fastest	35,095	35,91481	8130	6	8001	35	4
	slowest	35,579	36,541871	7991	4	5792	27	3
	3rd flow	8,999	11,056031	4226	2	1833	0	0
5	fastest	28,443	29,453956	9914	3	4344	28	3
	slowest	35,49	36,669746	7963	5	7240	30	5
	3rd flow	11,599	13,864343	3370	3	4344	8	2
6	fastest	35,45	36,45008	8011	6	8688	40	5
	slowest	36,58	36,583544	7982	7	8097	34	5
	3rd flow	10,032	11,713612	3989	3	4344	8	1
7	fastest	35,93	36,610615	7976	7	10136	37	5
	slowest	37,123	37,296092	7829	5	6745	29	4
	3rd flow	10,128	12,032767	3883	3	4344	12	2
8	fastest	32,16	33,171186	8803	6	8688	35	4
	slowest	35,709	36,558546	7987	5	7240	37	5
	3rd flow	10,07	12,110946	3858	3	4344	9	2
9	fastest	31,949	33,130365	8814	5	7240	32	4
	slowest	35,75	36,597186	7979	5	7240	36	4
	3rd flow	8,006	9,921204	4709	1	1448	3	0
10	fastest	32,158	33,169642	8803	6	8688	35	4
	slowest	35,709	36,556989	7988	5	7240	37	5
	3rd flow	10,07	12,109387	3858	3	4344	9	2
11	fastest	33,195	33,987925	8591	8	10705	27	3
	slowest	35,467	36,634767	7971	5	7240	24	4
	3rd flow	8,44	19,759034	2364	3	1450	6	1
12	fastest	31,286	32,467048	8994	5	6128	37	4
	slowest	35,629	36,633945	7971	5	7240	29	4
	3rd flow	9,087	10,988471	4252	2	1833	7	1
13	fastest	35,925	36,605507	7977	7	10136	37	5
	slowest	37,128	37,296158	7829	5	6745	29	4
	3rd flow	10,129	12,02778	3884	3	4344	12	2
14	fastest	29,005	30,025566	9725	5	7240	29	3
	slowest	35,488	36,661778	7965	4	5792	25	4
	3rd flow	12,009	23,325586	2003	5	4346	9	2
15	fastest	32,804	33,475526	8723	6	8193	36	5
	slowest	35,479	36,650944	7967	4	5792	25	4
	3rd flow	10,81	12,725202	3671	4	4729	5	1
16	fastest	31,283	32,473723	8992	5	6128	37	4
	slowest	35,627	36,640442	7969	5	7240	29	4
	3rd flow	9,101	10,984994	4253	2	1833	7	1
17	fastest	31,943	33,123586	8815	5	7240	32	4
	slowest	35,75	36,590428	7980	5	7240	36	4
	3rd flow	8,003	9,924465	4708	1	1448	3	0
18	fastest	35,931	36,611222	7976	7	10136	37	5
	slowest	37,132	37,296015	7829	5	6745	29	4
	3rd flow	10,128	12,023611	3886	3	4344	12	2
19	fastest	28,54	29,55177	9881	4	5792	34	4
	slowest	35,459	36,637594	7970	4	5792	24	4
	3rd flow	12,609	14,982194	3118	4	5792	8	2
20	fastest	34,795	35,410581	8246	7	9497	30	4
	slowest	35,709	36,5582	7987	7	10136	38	6
	3rd flow	9,285	11,161631	4186	2	1593	7	1

Table 8-5. Three TCP competing flows when Tail Drop is in use. The third flow starts *after 5.5 seconds*.

		Data xmit time (sec)	Elapsed time (sec)	Throughput (bytes/s)	Retx data packets	Retx Data bytes	Duplicate acks	Triple dupacks
fastest flow	Min	27,15	28,63	7976	3	4344	27	3
	Max	35,93	36,61	10198	14	20272	51	5
	25%Perc.	30,90	32,05	8113,75	5,00	7240	31,5	4
	Median	32,16	33,17	8803,00	6	8440,5	35	4
	75%Perc	35,04	35,99	9116,75	7	9656,75	37	5
	Average	31,14	32,61	8513,65	6,15	8514,30	33,40	4,1
	Stdev	2,78	2,64	716,96	2,17	3193,80	5,41	0,72
slowest flow	Min	35,46	12,11	7829	3	4344	3	0
	Max	37,13	37,30	7991	9	13032	41	6
	25%Perc.	35,49	36,59	7964,50	5	6745,00	26,75	4
	Median	35,67	36,64	7969,5	5	7240	29,00	4
	75%Perc	35,81	36,66	7980,50	5	7240	36	5
	Average	35,54	36,37	8048,35	5,15	7281,00	30,35	4,2
	Stdev	0,59	0,25	54,24	1,21	1673,27	4,78	0,66
third flow	Min	7,77	9,92	1798	1	1448	0	0
	Max	21,02	25,99	4709	5	5792	12	2
	25%Perc.	9,07	11,04	3307,00	2	1833	5	1
	Median	10,05	12,03	3883,50	3	4344	7,5	1
	75%Perc	10,30	14,14	4232,50	3,25	4344,25	9	2
	Average	10,36	13,89	3645,45	2,90	3415,00	7,10	1,25
	Stdev	2,82	6,93	1317,07	1,12	1497,02	5,11	0,99

Table 8-6. The statistics for the scenario *after 5.5 seconds* when Tail Drop is deployed.

		Data xmit time (sec)	Elapsed time (sec)	Throughput (bytes/s)	Retx data packets	Retx Data bytes	Duplicate acks	Triple dupacks
fastest flow	Min	20,89	21,57	7918	7	10136	44	5
	Max	36,88	36,88	13539	18	26064	59	10
	25%Perc.	30,74	31,46	8118,25	10,75	15566	49,75	8
	Median	33,90	34,60	8439,50	12	17376	52,5	8
	75%Perc	35,41	35,97	9282,25	13	18824,00	54,25	9
	Average	31,26	31,92	8870,75	11,45	16440,90	49,90	7,95
	Stdev	3,71	3,66	1270,48	2,38	3463,66	3,54	1,26
slowest flow	Min	35,82	36,54	7781	12	17376	40	6
	Max	37,53	37,53	7991	15	21720	60	11
	25%Perc.	36,03	36,57	7950,75	13	18824,00	49,75	8
	Median	36,15	36,63	7970,5	14	19596,5	52,50	9
	75%Perc	36,45	36,73	7985,25	14,25	20296	55	10
	Average	36,33	36,77	7942,25	13,6	19547,70	51,70	8,8
	Stdev	0,51	0,34	72,92	1,04	1452,37	4,69	1,31
third flow	Min	4,56	5,70	2946	1	1448	5	1
	Max	12,74	15,86	8200	7	9073	15	3
	25%Perc.	6,38	7,67	4073,00	2	2570,25	7	1
	Median	8,06	9,37	5005,00	3	4344	9,5	2
	75%Perc	10,09	11,48	6091,00	4,00	5792	13	2
	Average	8,19	9,78	5147,95	3,20	4462,15	9,95	1,80
	Stdev	2,33	6,75	1520,77	3,12	4447,02	9,67	1,97

Table 8-7. The statistics for the scenario *after 5.5 seconds* when RED discipline is adopted.

8.2 Tests with TCP and UDP traffic with services differentiation

In this second set of experiments we have utilized and analyzed not only TCP, but also UDP traffic. Even though TCP and UDP use the same network layer (IP), TCP provides a totally different service to the application layer than UDP does. UDP is a simple, datagram-oriented, transport layer protocol. RFC 768 [Pos80] is the official specification of UDP. What is important to point out for our study is that UDP provides no reliability. It sends the datagrams that the application writes to the IP layer, but there is no guarantee that they ever reach their destination. The UDP header is generally of 8 bytes and consists of the source and destination port number (16bit for each one), the UDP length field and the UDP checksum.

UDP supplies minimized transmission delay by omitting the connection setup process, flow control and retransmission. Thanks to its simplicity, UDP meets the requirements of delay-sensitive real-time applications that can implement their own flow control and retransmission schemes. Traditionally, on wide area networks (WANs) UDP not TCP has been used as a transport layer protocol for real-time applications, such as video and audio. Moreover, UDP is able to perform multicast communications, which allows the development of applications such as network conferencing. Meanwhile, more than 80 percent of the WAN resources are occupied by TCP traffic. Hence, the quality of service (QoS) of real-time applications using UDP is affected by TCP traffic and its flow control mechanism whenever TCP and UDP share a bottleneck node. This is because TCP flow control continues to increase its window size until packet loss occurs if the advertised window is large enough.

As regards our experiments we have used *Constant Bit rate (CBR)* service category. This is used by connections that request a static amount of bandwidth that is continuously available during the connection lifetime. CBR service is intended to support real-time applications requiring tightly constrained delay variation (for instance voice, video…) but is not restricted to these applications.

In our study the only metrics we have introduced to evaluate the UDP performance is the number of packets correctly received. When the HTB configuration implementing two classes of service is deployed there are no UDP packets dropped since the class of service dedicated to UDP flow provides enough bandwidth (70 Kbit/s) to serve the data rate produced (roughly 53Kbit/s).

The tables presented report statistics related to the TCP flow sent and the CBR traffic. As regard the TCP performance we report statistics regarding the Elapsed Time (Minimum, Maximum, 25%Percentile, Median, 75%Percentile, Average, Standard Deviation) and the average Throughput of the TCP transfer. As regards the UDP traffic, it is presented only in the number of packets correctly received. The results are classified firstly according to the scenario they belong to, which expresses the delay in the CBR's starting point, and secondly inside each scenario according to the HTB topology implemented in the router and the queueing discipline under study. The maximum and minimum values among the results of each HTB configuration are highlighted.

Scenario after:	HTB Config.	TCP Flow's Elapsed Time (sec)							TCP Flow's Throughput (Bytes)	UDP Packets Received
		Min	Max	25%Perc.	Median	75%Perc	Average	Stdev	Average	Median
0 sec	Tail Drop	31,63	34,25	33,10	33,47	33,71	33,33	0,62	11484,50	1694
	RED	35,03	36,89	35,56	35,69	36,26	35,82	0,48	12556,20	1864,5
	Tail Drop+DS	37,68	37,86	37,69	37,77	37,84	37,77	0,07	11052,05	2001
	RED+DS	37,49	39,22	37,64	37,80	37,90	37,90	0,44	11067,15	2001
1 sec	Tail Drop	30,16	35,44	32,20	32,72	33,18	32,67	1,06	13209,60	1711
	RED	34,18	35,67	34,92	35,10	35,17	35,05	0,32	13019,60	1871,5
	Tail Drop+DS	37,02	37,28	37,04	37,19	37,27	37,17	0,11	6864,65	2001
	RED+DS	36,96	39,19	37,12	37,15	37,27	37,37	0,57	11858,95	2001
2 sec	Tail Drop	30,91	33,21	31,80	32,28	33,02	32,28	0,70	9844,15	1707
	RED	34,08	35,54	34,49	34,57	34,87	34,72	0,42	10622,70	1877,5
	Tail Drop+DS	36,34	36,60	36,45	36,45	36,52	36,47	0,07	8719,40	2001
	RED+DS	36,16	38,05	36,45	36,57	36,62	36,75	0,53	13010,15	2001
3 sec	Tail Drop	29,80	47,39	31,47	32,15	33,37	34,20	5,05	10024,15	1683,5
	RED	33,50	35,24	33,81	34,01	34,18	34,13	0,50	11971,85	1889
	Tail Drop+DS	35,64	36,28	35,78	35,94	36,12	35,95	0,20	10226,50	2001
	RED+DS	34,83	38,25	35,70	35,77	35,87	35,89	0,63	14153,70	2001

Table 8-8. Statistics for Test Case 1: long TCP transfer competing with UDP traffic.

Scenario after:	HTB config.	TCP Flow's Elapsed Time (sec)							TCP Flow's Throughput (Bytes)	UDP Packets Received
		Min	Max	25%Perc.	Median	75%Perc	Average	Stdev	Average	Median
0 sec	Tail Drop	3,66	8,08	3,71	3,74	3,76	4,31	1,31	11484,50	268
	RED	3,46	3,95	3,64	3,77	3,84	3,73	0,15	12556,20	287,5
	Tail Drop+DS	4,21	4,25	4,22	4,22	4,24	4,23	0,01	11052,05	301
	RED+DS	4,04	6,11	4,08	4,21	4,22	4,25	0,44	11067,15	301
0,5 sec	Tail Drop	3,44	5,30	3,46	3,46	4,01	3,91	0,79	12329,85	275
	RED	3,14	4,95	3,43	4,55	4,83	4,21	0,69	11421,55	269,5
	Tail Drop+DS	7,01	9,07	7,47	8,38	8,82	8,15	0,73	5775,45	301
	RED+DS	3,60	6,82	3,76	4,82	6,21	5,06	1,24	9796,20	301
1 sec	Tail Drop	3,34	4,91	3,34	3,36	3,36	3,59	0,51	13209,60	292
	RED	2,93	5,02	3,22	3,28	4,33	3,70	0,70	13019,60	283
	Tail Drop+DS	6,66	6,84	6,82	6,84	6,84	6,81	0,06	6864,65	301
	RED+DS	3,38	6,09	3,54	3,69	3,99	4,09	0,92	11858,95	301
1,5 sec	Tail Drop	3,23	9,85	4,54	4,96	5,00	4,97	1,31	9844,15	301
	RED	2,86	9,37	3,46	4,87	5,12	4,88	1,75	10622,70	297
	Tail Drop+DS	5,32	5,65	5,33	5,33	5,34	5,36	0,09	8719,40	301
	RED+DS	3,04	6,91	3,21	3,33	3,72	3,77	1,01	13010,15	301
2 sec	Tail Drop	4,35	4,80	4,69	4,72	4,72	4,66	0,13	10024,15	301
	RED	2,70	4,76	3,82	4,68	4,72	4,12	0,86	11971,85	301
	Tail Drop+DS	4,34	4,78	4,55	4,57	4,57	4,57	0,09	10226,50	301
	RED+DS	2,70	4,21	2,87	3,12	4,09	3,39	0,59	14153,70	301

Table 8-9. Statistics for the Test Case 2: Short TCP transfer competing with UDP traffic.

Scenario after:	HTB Config.	TCP Flow's Elapsed Time (sec)							TCP Flow's Throughput	UDP Packets Received
		Min	Max	25%Perc.	Median	75%Perc	Average	Stdev	Average	Median
0	Tail Drop	23,77	34,79	25,08	25,50	26,33	26,33	2,82	26,33	174
	RED	24,18	32,20	24,47	24,62	26,95	26,05	2,54	26,05	232,5
	Tail Drop+DS	24,52	24,88	24,56	24,57	24,58	24,58	0,08	15919,95	251
	RED+DS	24,52	30,38	24,71	24,74	25,94	25,63	1,63	15320,05	251
2	Tail Drop	24,12	38,71	25,29	25,31	26,11	27,19	4,23	27,19	188
	RED	24,40	44,41	24,45	24,55	25,21	26,00	4,49	26,00	230
	Tail Drop+DS	24,55	35,78	24,64	24,74	25,25	26,19	3,16	15110,85	251
	RED+DS	24,52	34,43	24,64	24,85	29,49	26,94	3,42	14725,85	251
7	Tail Drop	22,92	24,60	23,64	23,70	23,90	23,71	0,40	23,71	126,5
	RED	24,37	26,34	24,48	24,55	24,61	24,74	0,51	24,74	234,5
	Tail Drop+DS	24,52	32,83	24,55	24,55	24,96	25,09	1,83	15657,30	251
	RED+DS	24,58	25,46	24,63	24,64	24,77	24,74	0,21	15819,40	251
12	Tail Drop	23,15	23,87	23,59	23,66	23,67	23,64	0,14	23,64	114
	RED	24,39	25,59	24,47	24,54	24,56	24,61	0,28	24,61	232
	Tail Drop+DS	24,54	25,24	24,55	24,55	24,67	24,63	0,17	15885,00	251
	RED+DS	24,56	25,03	24,64	24,70	24,76	24,73	0,13	15822,40	251
16	Tail Drop	23,90	24,57	24,14	24,22	24,29	24,23	0,16	24,23	194
	RED	24,32	25,30	24,44	24,49	24,53	24,52	0,20	24,52	231
	Tail Drop+DS	24,55	24,74	24,56	24,60	24,66	24,62	0,07	15894,65	251
	RED+DS	24,52	25,59	24,64	24,68	24,74	24,75	0,26	15807,90	251
18	Tail Drop	23,64	24,51	24,40	24,49	24,50	24,38	0,22	24,38	225
	RED	24,34	25,63	24,46	24,52	24,59	24,64	0,38	24,64	225
	Tail Drop+DS	24,55	45,25	24,56	24,57	24,74	26,17	5,01	15290,85	251
	RED+DS	24,47	25,72	24,64	24,65	24,73	24,81	0,37	15772,25	251

Table 8-10. Statistics for the Test Case 3: long TCP transfer competing with UDP traffic for a short time.