Master's Thesis

SIMULATIVE EVALUATION OF IDCC IN A RESOURCE MANAGEMENT DIFFERENTIATED SERVICES ENVIRONMENT

George Hadjipollas

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

June 2003

UNIVERSITY OF CYPRUS
DEPARTMENT OF COMPUTER SCIENCE

SIMULATIVE EVALUATION OF IDCC IN A RESOURCE MANAGEMENT DIFFERENTIATED SERVICES ENVIRONMENT

George Hadjipollas

Supervising Professor
Andreas Pitsillidis

This thesis is submitted to the Graduate Faculty of University of Cyprus in partial fulfillment of the requirements for the Degree of Master of Science at Computer Science Department

June 2003

Acknowledges

I would like to thank my family for the support gave me during the whole six years of my studies. I am also grateful to my professors Anna Philippou and Andreas Pitsillides for their guidance during the last three years of my studies, which gave me the opportunity to write this thesis. Furthermore, I would like to express my thanks to my colleague and friend Chrysostomo Chrysostomou for his helpful feedback and comments. At last I am deeply grateful to my wife Eleni for her moral support and help during the writing of this M.Sc Thesis.

Summary

The continuously increasing use of computers in the Internet, as well as the need from new applications on quality of service, require efficient ways so the congestion in cases where the demand in bandwidth is increased decreases in minimal. For this reason, the quality of service has been turned into one of the most challenging sectors in the research of computer networks in the last years. The result of this research was the two IP architectures for quality of service: the Intserv and the Diffserv. Independently from which architecture will be selected to be used, in order to achieve quality of service we have the need to control the flow in the network.

The Resource Management in DiffServ (RMD) is a recently proposed QoS framework that extends the DiffServ architecture with new principles, necessary to provide resource provisioning and admission control. It was developed to satisfy requirements regarding support of large amount of real time traffic on costly leased transmission lines. Integrated Dynamic Congestion Control (IDCC) is a congestion control scheme used for controlling the flow in the network. It offers high utilization of the bandwidth by regulating the rate by which the ordinary traffic is send.

This M.Sc. Thesis aim to study and evaluate the performance of IDCC in a Resource Management Differentiated Services (RMDS) Environment. The main objective is to examine the combined framework reliability and robustness and the possibility that it could meet the strict QoS requirements of real time traffic. Some other issues are also examined, such as the implementation of the best effort controller and the use of a function that estimates the number of flows traversing a link. The simulation scenarios were implemented and tested with the use of the ns-2 simulator.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 The need of Quality of Service

The rapid growth of the Internet and the increased demand to the use of the Internet for time-sensitive voice and video applications, necessitate the design and utilization of new Internet architectures in order to support Quality of Service. For this reason, the quality of service has been turned into one of the most challenging sectors in the research of computer networks in the last years. The efforts for providing quality of service in the IP networks have led mainly to the creation of two architectures, the Integrated Services (Intserv) [1] and more recently the Differentiated Services (Diffserv) [2].

Because of the large diversity that the existing Internet applications present (that vary from very simply as the e-mail and FTP up to very demanding real-time, as video on demand, multimedia conferencing and IP telephony), as well as the creation of wireless networks have turned the above architectures insufficient in order to ensure the demanded QoS. Thus a new effort has begun in order to enrich those architectures with techniques that aim to provide additional and very strict quality of service. One of the categories that the researches spend a lot of effort is congestion control techniques, which try to detect the congestion in the network and take measures in order to reduce the rate by which the sources send their traffic.

1.2 Congestion Control Mechanisms

Congestion control mechanisms aim to the management of the routers queues. Depending on the mechanism one of several methods for detecting congestion may be used and send to the source congestion feedback in order to reduce their rate of sending packets through the web.

The exciting solutions [3, 4], which are based on an end-to-end approach, are becoming insufficient. In this approach sources infer congestion only from lost packets. When such an event occurs sources are decreasing their rate of sending packets. These solutions were sufficient at the time when the internet was used only for best effort traffic and the applications were not sensitive in delay or even losses of packets. Although some fixes [5, 6, 7] were proposed for those techniques (like slow start and congestion avoidance), they do not scale up easily and cannot provide fairness among the different sources.

Because of the reasons mentioned above, Active Queue Management (AQM) [8] was proposed. AQM routers can detect congestion before the queue overflows. The method used initially was Random Early Discard [9]. Based on the average queue size at the arriving time of a packet, a probability of dropping that packet is produced on which the choice of its drop depends. Although it was demonstrating better performance, many problems were raised because of its ad-hoc design and the difficulty in tuning its parameters. After that, AQM adopted a new way that allows the congestion feedback to be different than the drop of a packet. Therefore, routers are able to use the Congestion Experienced (CE) codepoint in a packet header as an indication of congestion. Explicit Congestion Notification (ECN) [10, 11] was proposed in order to provide TCP an alternative to packet drops mechanism for detecting incipient congestion in the network. The ECN scheme requires both end-to-end and network support. With this method a router can mark a packet either by dropping it or by setting a bit if the transport protocol is capable of reacting to ECN. The use of ECN for notification of congestion to the end-nodes generally prevents unnecessary packet drops that leads to a reduce impact of loss on latency sensitive flows. It also reduces the excessive delays caused by unnecessary drops. Typical representatives of AQM are RED [12] and its variants (n-RED, adaptive RED [12], RIO [13], BLUE [14, 15] and three color marking) and Fuzzy Explicit Marking (FEM) [16].

Recent studies [17] have investigated the impact of the above solutions and they have demonstrated that they are robust in a variety of simulations. However, these schemes are developed using intuition and simple ad-hoc non-linear control designs (e.g. slow start, congestion avoidance, binary feedback, additive increase and multiplicative decrease). For these reasons the analysis of their closed loop behavior is difficult. Even worse, the interaction of additional non-linear feedback loops can produce unexpected and erratic behavior [18]. Very little is known why these methods work and very little explanation can be given why they fail. The last few years the increase on demand of QoS has clearly shown the poor performance of the controlled TCP/IP internet. Thus, more effective congestion control mechanisms are needed in order to prevent serious economic losses, even a possible ‘meltdown’ of the internet.

1.3 IDCC in a Resource Management Differentiated Services Environment

The Integrated Dynamic Congestion Control (IDCC) mechanism [19, 20, 21] can be classified as an AQM mechanism that uses queue length information for congestion feedback. A differentiated services network is assumed and the formulation of the control strategy is based in the same spirit as the IP-DiffServ. There are three types of services: Premium Service, Ordinary Service and Best Effort Service, each class operating at one output port of the router. Each output port has a number of physical or logical queues: one for each traffic class and an IDCC scheme is designed for each output port.

We assume some sources that transmit packets to specific destinations. The packets traverse a number of routers before they reach the destination. Each one of the sources can be classified in one of the three services: Premium, Ordinary or Best Effort.

Premium traffic requires strict guarantees on delivery and cannot tolerate losses or even delays. It does not allow regulation of its transmitted rate and any regulation of this type has to be achieved at the connection phase. Once admitted into the network, the network has to guarantee the QoS. This is the task of the Premium Traffic Controller. On the other hand, Ordinary traffic allows the network to regulate its rate. It can tolerate delays but it cannot tolerate losses of packets. This work is performed by the Ordinary Traffic Controller. At last the Best Effort service offers no guarantees on either losses or delay and it uses any instantaneous left over capacity.

Resource Management in DiffServ (RMD) framework [22, 23, 24, 25, 26, 27] extends the Diffserv architecture with dynamic admission control and resource reservation concepts. It is a very simple framework with very good scaling properties and thus it has very low cost of implementation. In a Resource Management Differential Services Environment (RMDSE) we reserve resources (bandwidth) only in the case of Premium traffic, based on the fact that Premium Service sends packets in a constant bit rate so the amount of resources needed can be calculated. In this way we ensure the guarantees of QoS that the Premium traffic demands. On the other hand we let the IDCC to formulate the rate of the Ordinary traffic in order to achieve high utilization of the links bandwidth without drops of packets. In the case of Best Effort traffic we use CBR sources that use any instantaneous bandwidth left over from the two other services.

This M.Sc. Thesis aims to study and evaluate the performance of IDCC in a Resource Management Differentiated Services (RMDS) Environment. The main objective is to examine the combined framework reliability and robustness and the possibility that it could meet the strict QoS requirements of real time traffic. Some other issues are also examined, such as the implementation of the best effort controller and the use of a function that estimates the number of flows traversing a link. The simulation scenarios were implemented and tested with the use of the ns-2 simulator.

The rest of it is organized as follows. In Chapter 2 we overview the Differentiated Services, as well as the RMD framework. In Chapter 3 we give the IDCC congestion control algorithm and how this algorithm can be used in an RMDS Environment. In Chapter 4 we study and analyze the performance of that scheme in various scenarios using the ns-2 simulator. Finally in Chapter 5 we present the conclusions that evolve from the previous chapter and some suggestions are introduced for future work.

Chapter 2

Diffserv, and RMD frameworks

2.1 Introduction

The continuously increasing use of computers in the Internet, as well as the need from new applications on quality of service, require efficient ways so the congestion in cases where the demand in bandwidth is increased decreases in minimal. For this reason, the quality of service has been turned into one of the most challenging sectors in the research of computer networks in the last years. The first result of this research was the Integrated Services architecture (Intserv) [1]. Since IntServ failed to be adopted for widespread use because of its scalability problems, IETF (Internet Engineering Task Force) proposed a more evolutionary approach that did not require significant changes to the Internet infrastructure and provided differentiation of services (DiffServ) [2]. The rest of the chapter is organized as follows. In chapter 2.2 you can find a short description of Differentiated Services architecture. In chapter 2.3 we give the description of Resource Management in DiffServ (RMD), an admission control scheme that extends DiffServ with new principles necessary to provide resource provisioning and admission control.

2.2 Differentiated Services framework

The DiffServ architecture is based on reservation-less traffic engineering. It classifies packets into a number of service types and uses priority mechanisms in order to provide QoS to the traffic. No explicit resource reservation or admission control is employed, although network nodes do have to use intelligent queuing mechanisms to differantiate traffic.

The DiffServ working group [28] has defined two broad aggregate behaviour groups: the Expedited Forwarding (EF) Per-Hop Behaviour (PHB) [29] and the Assured Forwarding (AF) PHB [30].

The EF-PHB can be used to build a low loss, low latency, low jitter, assured bandwidth end-to-end service. This type of service provides some minimum quality of service. In order to ensure that every packet marked with EF receives this service, EF requires from 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 forward them. This can be done through a Service Level Agreement (SLA) during the connection setup. In order to preserve this property on an end-to-end basis, EF requires traffic shaping and reshaping in the network. Although there is no specific method set for this, it will most probably be a leaky- bucket buffering algorithm.

The AF-PHB group provides delivery of IP packets in four independently forwarded AF classes. Within each AF class two or three drop preference levels are used to differentiate flows. The idea behind AF is to referentially drop best-effort packets and non-contract conforming packets when there is congestion. By limiting the amount of AF traffic in the network and by managing the best-effort traffic appropriately, routers can ensure low loss behavior to packets marked with the EF PHB.

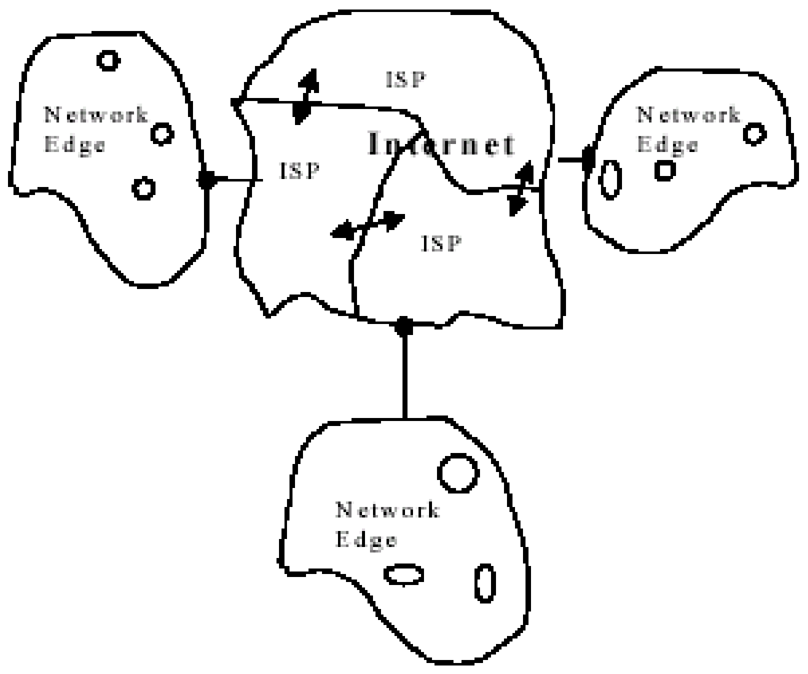


Figure 2.1 An example of the DiffServ Architecture

To differentiate aggregated traffic classes, DiffServ uses the ToS (type of service) byte in the IP header, which is now renamed as "DS (differentiated services) field" [31]. *Figure 2.1* shows an example of the basic DiffServ architecture approach. Note that the priorities are set at the edges of the network, which reduces the complexity and makes the proposed architecture scalable. However, no measures are taken to assure that the priorities would actually mean something when the packet enters the Internet (leaves the edge router).

2.3 RMD framework

RMD [22, 23, 24, 25, 26, 27] is a QoS framework that is based on standardized Diffserv principles for traffic differentiation. For the above reason it is a single domain, edge-to-edge resource management scheme. RMD extends the Diffserv principles with new ones necessary to provide dynamic resource management and admission control in Diffserv domains. It is a very simple protocol with very good scaling properties. This framework was proposed in order to meet the strict requirements imposed by IP-based wireless networks. Even though it is optimized for networks with fast and highly dynamic resource reservation schemes, it can be applied in any type of DiffServ networks.

The development of RMD was initially based on two main design goals.

- RMD should be as stateless as possible.
- Even though stateless, RMD should be able to associate each reservation to each flow and should be able to provide certain QoS guarantees for each individual flow.

In order to meet these two goals and achieve the desired scalability, the services are offered on an aggregated basis rather than per flow. Per flow state is forced as much as possible to the edges of the network. It is assumed that some nodes are stateful and will support "per-flow states". These nodes are called "edge nodes". Also we assumed that the rest of the nodes (called interior nodes) would have a simpler execution. This is achieved by using only one aggregated reservation state per traffic class. The edges will generate reservation requests for each flow, similar to RSVP [32]. In order to achieve the desired simplicity in interior nodes, a measurement-based approach on the number of the requested resources per traffic class is applied. In practice, RMD framework introduces a measurement – based algorithm that uses the requested and the available resources as input, in order to update the aggregated reservation state per traffic class in the interior nodes.

In a Diffserv domain with RMD in place, different PDR and PHR protocols can be used simultaneously. The PHR is a newly defined protocol, while the PDR can also be one of the existing protocols. Figure 2.2 illustrates the RMD and the PDR and PHR protocols.

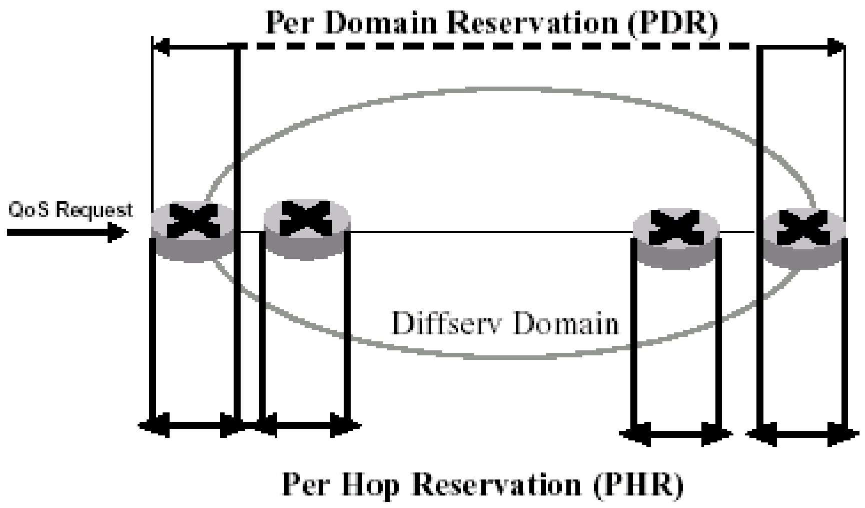


Figure 2.2 RMD Framework

2.3.1 Per Domain Reservation (PDR) protocol

PDR is implemented only at the edge nodes of the Differentiated Services domain and it is responsible for the resource management in the entire domain. It can be a newly defined protocol, or an existing one such as RSVP [32], RSVP aggregation [33] and Boomerang [34].

This protocol is responsible for the interoperation with external resource reservation protocols and PHR protocol. Thus it can be seen as a link between the external resource reservation scheme and the PHR. The linkage is done at the edge nodes by associating the external reservation request flow identifier (ID) with the internal PHR resource reservation request. This flow ID, depending on the external reservation request, can be of different formats. For example, a flow specification ID can be a combination of source IP address, destination IP address and the DSCP field.

2.3.2 Per Hop Reservation (PHR) protocol

PHR is implemented at the interior nodes of the Differentiated Services domain and it is responsible for the regulation of the resources on a per hop basis. It extends the DiffServ

Per Hop Behavior (PHB) with dynamic resource management and its main role is to provide service differentiation. It is used to perform a per hop reservation per PHB, so it is used on a per hop basis.

Two groups of PHR protocols are defined for the RMD framework:

- *Reservation based PHR group*. This type of protocols enables dynamic resource reservation per PHB in each node in the communication path edge-to-edge. The PHR aware nodes maintain one reservation state per PHB by using a combination of the soft state and the explicit release principles. The reservation request is signaled in terms of resource units (most commonly bandwidth) and a threshold of maximum available resources will be set for each PHB. Currently there is only one reservation-based PHR protocol the Resource Management in DiffServ On Demand (RODA) [37].
- *Measurement-based Admission Control (MBAC) PHR* group. This type of protocols signals the availability of resources in the communication path edge-to-edge without maintaining any reservation state in the nodes. The availability of the resources is checked by means of measurements before any "QoS requests" are admitted. The measurements are done on the average real-time traffic (user) data load. The main advantage of this PHR group in comparison with the Reservation-based PHR group is a better efficiency of resource utilization and no need for state maintenance. The only state information maintained for the measurement based PHR is related to the measured user traffic load associated to the PHB and the maximum allowable traffic load per PHB. For further elaboration on the use of MBAC in IP-based RANs see [35]. In general a PHR protocol has a set of functions associated (see [36]) and same as with the PDR it depends on the type of the network where RMD is applied whether only a subset is implemented or all of it. Currently there is only one MBAC PHR protocol called RIMA.

Chapter 3

Integrated Dynamic Congestion Control framework

3.1 Introduction

The currently congestion control schemes are developed using intuition and simple ad-hoc non-linear control designs (e.g. slow start, congestion avoidance, binary feedback, additive increase and multiplicative decrease). The analysis of their closed loop behavior is difficult. Even worse, the interaction of additional non-linear feedback loops can produce unexpected and erratic behavior [38]. Very little is known why these methods work and very little explanation can be given why they fail. The last few years the increase on demand of QoS has clearly shown the poor performance of the controlled TCP/IP internet. After the creation of the Differentiated Services new admission control mechanisms are being developed in order to provide the service differentiation in a router. One of them is Integrated Dynamic Congestion Control.

Following the spirit adopted by the IETF DiffServ working group for the Internet [28] we have the definition of classes of aggregated behavior (see Chapter 2.2). These defined classes used in this framework are:

- *Premium Traffic Service:* This type of service is the one used for low loss, low latency, low jitter thus indirectly provides some minimum quality of service. It may belong to the EF-PHB in a DiffServ architecture and it is designed for applications with strict delay and loss requirements on per packet basis.
- *Ordinary Traffic Service:* It may belong to the first class of the AF-PHB in a DiffServ architecture. It is intended for applications that can afford delays in the network and can allow their rate to be controlled by the network. It uses any left over capacity from the Premium Traffic.
- *Best Effort Traffic Service:* It may belong to the last class of the AF-PHB in a DiffServ architecture. This type of service has no QoS expectations and uses any instantaneous left over capacity from both Premium and Ordinary Traffic Services.

3.2 IDCC framework

Integrated Dynamic Congestion Control [19, 20, 21] is a scheme used for controlling traffic using information of the status of each queue in the network. It has been developed using non-linear control theory. Its methodology is general and independent from the technology used. Thus, it can be used both in TCP/IP and ATM environments. In this thesis we assume an IP DiffServ Environment that uses Reservation Resource Management. It operates locally for the Premium Traffic and it sends feedback to the sources in order to regulate their rate of sending Ordinary Traffic.

3.2.1 Congestion Control Strategies

As a reference model a K-input K-output port, output buffer is assumed (Figure 3.1). Each one of the services classes (Premium Service, Ordinary Service and Best Effort Service) is operating at one output port of the router. Each output port has a number of physical or logical queues: one for each traffic class and an IDCC scheme is designed for each output port (Figure 3.2).

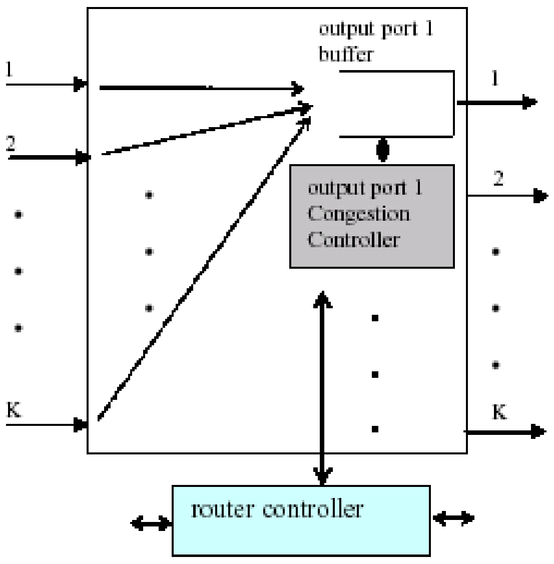


Figure 3.1 Generic output buffered K input-output router

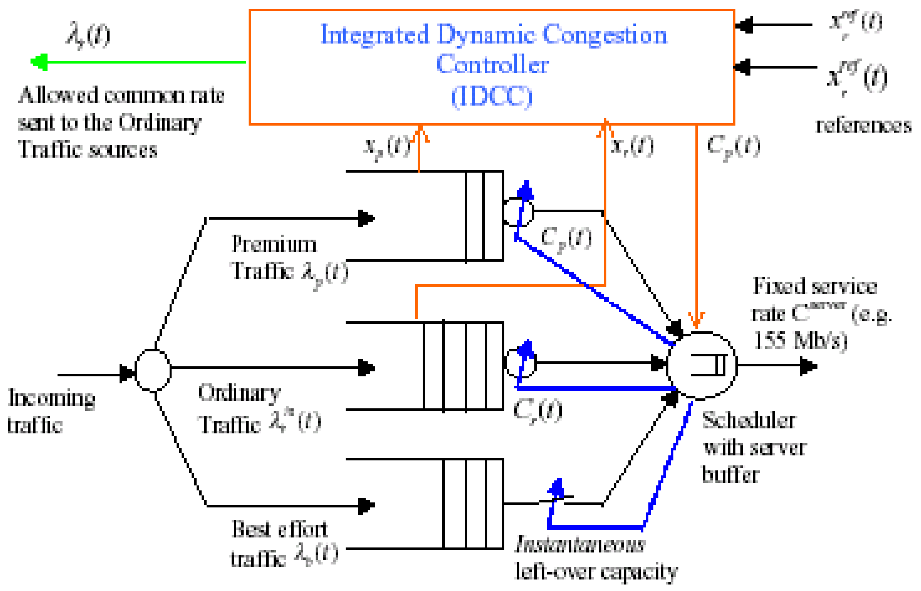

$\lambda_r(t)$
Allowed common rate sent to the Ordinary Traffic sources
Integrated Dynamic Congestion Controller (IDCC)
$x_r^{ref}(t)$
$x_r^{ref}(t)$
references
$x_p(t)$
$x_r(t)$
$C_p(t)$
Premium Traffic $\lambda_p(t)$
$C_p(t)$
Fixed service rate C^{server} (e.g. 155 Mb/s)
Incoming traffic
Ordinary Traffic $\lambda_r^{in}(t)$
$C_r(t)$
Scheduler with server buffer
Best effort traffic $\lambda_b(t)$
Instantaneous left-over capacity

Figure 3.2 Implementation of control strategy at each router output port

We assume that some sources transmit packets to specific destinations. The packets traverse a number of routers before they reach the destination. Each one of the sources can be classified in one of the three services: Premium, Ordinary or Best Effort. Premium traffic requires strict guarantees on delivery and cannot tolerate losses or even delays. It does not allow regulation of its transmitted rate and any regulation of this type has to be achieved at the connection phase. Once admitted into the network, the network has to guarantee the QoS. This is the task of the Premium Traffic Controller. On the other hand, Ordinary traffic allows the network to regulate its rate. It can tolerate delays but it cannot tolerate losses of packets. This responsibility is assigned to the Ordinary Traffic Controller. At last the Best Effort service offers no guarantees on either losses or delay and it uses any instantaneous left over capacity from the previous two classes.

The algorithms developed for the controllers are based on a non-linear dynamic model of the behaviour of a queue. The model has been verified by a number of researchers and it is of the form shown below:

$$\dot{x}(t) = -\frac{x(t)}{1 + x(t)} C(t) + \lambda(t)\, ,\, x(0) = x_o$$

Equation 3.1 Fluid Flow Equation for an M/M/1 queue

where *x*(*t*) is the state of the queue, given by the ensemble average of the number of cells *N*(*t*) in the system (i.e. queue + server) at time *t;* λ(*t*) is the rate packets arrive at the queue, and *C*(*t*) is the capacity of queue server. Note that this equation is valid for $0 \le x(t) \le x_{buffer\ size}$ and $0 \le C(t) \le C_{server}$ where $x_{buffer\ size}$ is the maximum possible queue size and C_{server} the maximum possible server rate.

3.1.1.1 Premium Traffic Control Strategy

In order to guarantee the expected QoS the approach used is to control the length of the Premium Traffic queue to be always close to a reference value (x_p^{ref}) chosen by the network administrator. The capacity of the Premium traffic is dynamically allocated up to a given maximum (C_{max}), that can be the physical server limit or a predefined given maximum. In this way the Premium Traffic is always given resources up to the allocated maximum in order to ensure the provisioning of Premium Traffic Service with known bounds. If the use of the maximum capacity is not necessary in order to maintain the desired QoS, it offers the excess capacity to the Ordinary Traffic Service.

The control objective is to choose the capacity $C_p(t)$ to be allocated to the Premium Traffic under the constraint that the incoming traffic rate $\lambda_p(t)$ is unknown but bounded by $\hat{k}_p$ so that the averaged buffer size $x_p(t)$ is as close to the desired value x_p^{ref} as possible. In mathematical terms we need to choose Cp(t) so that $\overline{x}(t) \to 0$ ($\overline{x}_p(t) = x_p(t) - x_p^{ref}$) under the constraints that $C_p(t) \leq C_{server}$ and $\lambda_p(t) \leq \hat{k}_p < C_{server}$. Based on the fluid flow equation 3.1, feedback linearization, and robust adaptive control ideas:

$$C_p(t) = \max\left[0, \quad \min\left\{C_{server}, \quad \rho_p(t)\frac{1+x_p(t)}{x_p(t)}\left[\alpha_p \overline{x}_p(t) + k_p(t)\right]\right\}\right]$$

where:

$$\rho_p(t) = \begin{cases} 0 & if \quad x_p(t) \leq 0.01 \\ 1.01x_p(t) - 0.01 & if \quad 0.01 < x_p(t) \leq 1 \\ 1 & if \quad x_p(t) > 1 \end{cases}$$

and

$$\dot{k}_p(t) = \Pr\left[\delta_p \overline{x}_p(t)\right], \qquad 0 \leq k_p(0) \leq \hat{k}_p$$

where $\Pr[\cdot]$ is a projection operator defined as:

$$\Pr\left[\delta_p \overline{x}_p(t)\right] = \begin{cases} & if (0 < k_p(t) < \hat{k}_p \ ) \\ \delta_p \overline{x}_p(t) & or\, (k_p(t) = \hat{k}_p \; and \; \overline{x}_p(t) \leq 0) \\ & or\, (k_p(t) = 0 \, and \; \overline{x}_p(t) \geq 0) \\ 0 & else \end{cases}$$

where $\hat{k}_p$ is a constant indicating the maximum rate that could be allocated to incoming Premium Traffic (e.g. through a connection admission policy), and $\alpha_p > 0$, and $\delta_p > 0$, are design constants that affect the convergence rate and performance. The analysis of the stability of the above control strategy can be found in [19, 39].

3.1.1.2 Ordinary Traffic Control Strategy

The Ordinary Traffic Service Controller regulates the flow of Ordinary Traffic into the network, by monitoring the length of the Ordinary Traffic queue and the available capacity.

The length of the Ordinary Traffic queue is compared with the reference value and using a non-linear control strategy it informs the sources of the maximum allowed rate they can transmit over the next control interval. This algorithm takes into account the leftover capacity $C_r(t) = \max\left[0, \;\; C_{server} - C_p(t)\right]$, uses the error between the queue length $x_r(t)$ of the Ordinary Traffic queue and the reference queue length $x_r^{ref}(t)$ as the feedback information ($\overline{x}_r(t) = x_r(t) - x_r^{ref}$), and calculates the common rate $\lambda_{rd}(t)$ to be allocated to the Ordinary Traffic users once every control interval Ts. Based on the fluid flow equation 3.1 and feedback linearization the controlled traffic input rate is

$$\lambda_{rd}(t) = \max\left[0, \;\; \min\left\{C_r(t), \;\; C_r(t)\frac{x_r(t)}{1+x_r(t)} - \alpha_r \overline{x}_r(t)\right\}\right]$$

where $\alpha_r > 0$ is a design constant. Once the common rate is calculated it is sent to all upstream sources. The way the latter is done is critical, as feedback delays and different feedback implementation schemes can degrade the system performance considerably and may lead to poor stability margin. In an IP network it is assumed that a new field has been introduced in the TCP header, which enables explicit transfer of feedback information. The introduction of such a field (basically the extension of ECN from 1 bit of information to several bits) has already been brought to the attention of the research community. The source does not allow its transmission rate over the next control interval Ts ms to exceed the allowed calculated common rate received. Note that any excess source demand (above calculated common rate) is queued at the source queues, rather than be allowed to enter the network, and thus lead to congestion.

3.1.1.3 Best Effort Traffic Control Strategy

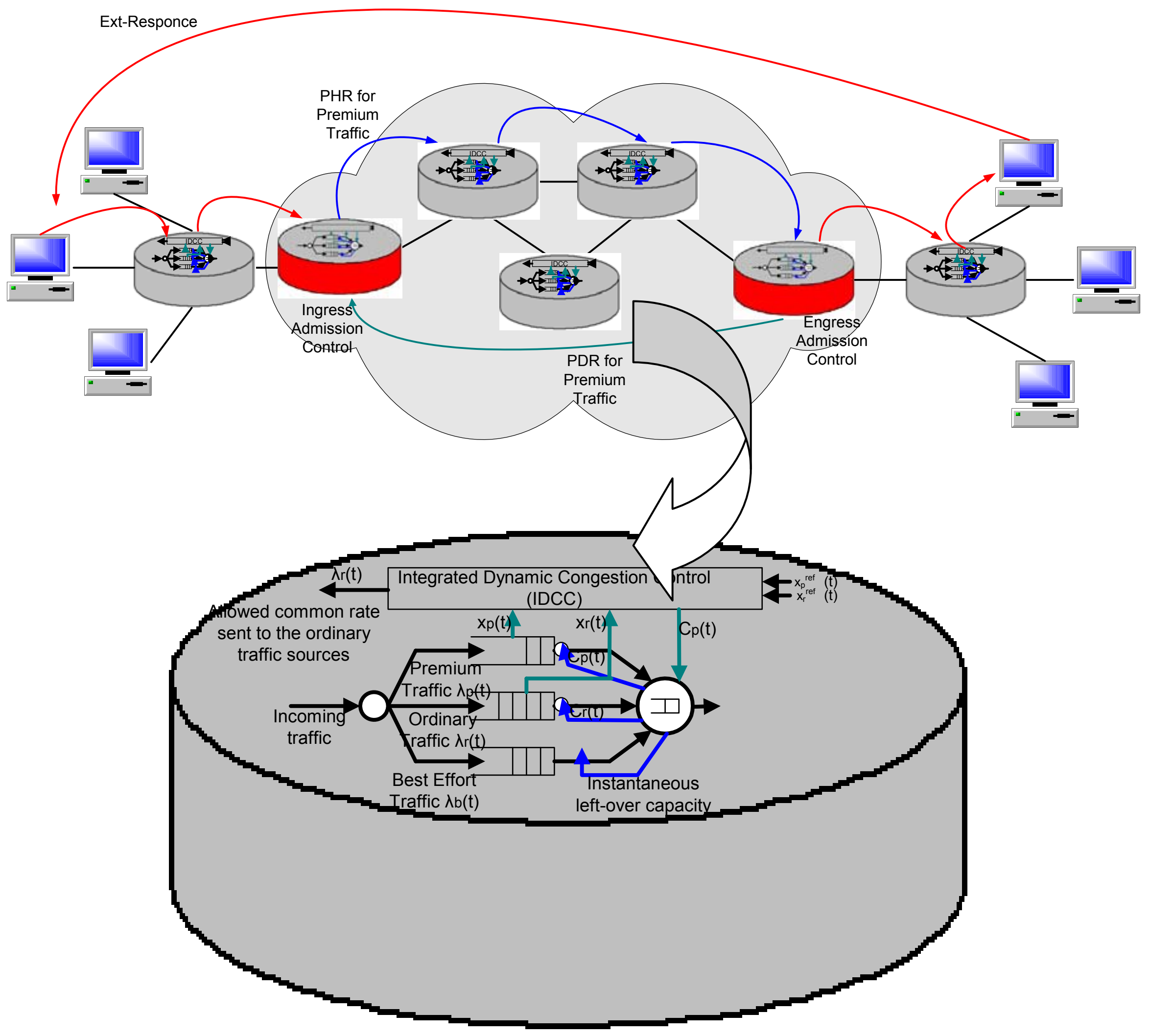
The Best Effort traffic controller operates on an instantaneous (packet or cell) time scale. It utilizes any *instantaneous* left over capacity to transmit a packet from the Best Effort buffer. This increases the network utilization during periods of insufficient supply of packets from both the Premium and Ordinary Traffic Services. Details appear in [39].

3.3 IDCC in a Resource Management Differentiated Services Environment

As seen from the previous chapter, IDCC satisfies Quality of Service (QoS) requirements through high utilization of the available resources and congestion control, by regulating the rate by which the Ordinary packets are being sent. The major problem of IDCC when it is used in a Differentiated Services Environment without Resource Reservation is its incapability to cope when the aggregated Premium traffic rate exceeds the link capacity [40]. In this thesis we will analyze the performance of this congestion control scheme when it is used in a Resource Management Differentiated Services (RMDS) Environment. In this section the way that the cooperation of the RMD and IDCC can be established is presented.

In RMD before a source (independently in which service class it belongs) starts sending traffic into the network, admission control and resource reservation is needed. In an RMDS Environment resources are reserved (most commonly bandwidth) only in the case of Premium traffic, based on the fact that Premium Service sends packets in constant bit rate and the amount of resources needed can be calculated. In this way we ensure the guarantees of QoS that the Premium traffic demands. In order packets to traverse each RMD domain PHR protocol header is used.

On an arrival of a packet to an ingress node that is classified to the one of the three classes, proposed by IDCC: Premium, Ordinary or Best Effort. If the packet is classified as Premium, it must be checked whether it is a "QoS Request" or a data packet. In the first case the PHR protocol is activated in order to request admission. In the latter, the packet is forwarded to IDCC Premium queue, which is responsible for forwarding it to the next node with bounded delay and without being dropped. In the case of an Ordinary packet, the packet is forwarded to IDCC Ordinary queue. Then IDCC calculates the rate by which the source must send ordinary data. When that happens, the PHR protocol header is updated with the new rate and at the proper time (according to the ordinary class rate) the packet is forwarded to the next node. If the packet belongs to the Best Effort class, it will be forwarded to best effort queue and if there is any instantaneous available bandwidth IDCC forwards it to the next node. The packet finally reaches the egress node. In the "QoS Request" case, the egress node is responsible for informing ingress node about the result of the reservation and the PDR protocol carry on according to that report. Otherwise, the packet is forwarded out of the DiffServ domain and follows the path towards the destination node. Figure 3.3 shows how the IDCC can be used in an RMDS environment.



Ext-Responce
PHR for Premium Traffic
Ingress Admission Control
PDR for Premium Traffic
Engress Admission Control
IDCC
Integrated Dynamic Congestion Control (IDCC)
λr(t)
Allowed common rate sent to the ordinary traffic sources
xp(t)
xr(t)
Cp(t)
Incoming traffic
Premium Traffic λp(t)
Ordinary Traffic λr(t)
Best Effort Traffic λb(t)
Cr(t)
Instantaneous left-over capacity
xp ref (t)
xr ref (t)

Figure 3.3 IDCC in an RMDS environment

Chapter 4

Performance Evaluation

4.1 Introduction

In this section we study the performance of IDCC in an RMDS environment using simulation scenarios. These scenarios were implemented and run in a recent version of NS-2 [41] simulator (Version 2.1b9a). Several scripts were written (see Appendix A) in order to transform the raw data produced by the simulator for the creation of the graphs presented in this chapter.

4.2 Simulation Scenarios

Three simulation scenarios are presented, showing the behaviour of IDCC scheme under different network conditions. Each one is trying to capture the QoS guarantees offered mainly to the two most important Class Services (Premium and Ordinary), such as packet loss, end-to-end delay and jitter, while studying the overall performance in the network obtaining throughput, rate and utilization measurements. Therefore, each scenario is divided in sub scenarios, each one modifying different scheme parameters in order to observe how do they affect the performance of it.

IDCC sources requesting QoS guarantees (bandwidth) are used for the Premium traffic, whilst Ordinary traffic is generated by saturated IDCC sources. That is, the ordinary sources have always the requested amount of data to send into the network, according to the feedback sent by the scheme. CBR traffic is used for the needs of the Best Effort traffic. In all the scenarios we assume an infinite buffer for the Best Effort traffic. For the Premium and Ordinary traffic the buffer size varies between 200 and 1000 packets.

Scenario 1 is used in order to examine the influence of the buffer size (and the target buffer sizes) as well as the effect of stops and restarts of some sources in the overall performance of the scheme. Also, under investigation is the respond of the scheme using an estimation of the maximum number of flows traversing the link, for different number of flows. Furthermore, it evaluates the QoS guarantees offered to the Premium Class Service, such as packet losses, end-to-end delay and jitter. Scenario 2 is used in order to see the response of the queue (using different buffer sizes) when the sources are gradually starting and stopping and the maximum number of flows is estimated. Scenario 3 is more complicated than the previous ones and it is mainly used in order to study the performance of the scheme under different end-to-end link delays as well as the performance of each individual source when the distance between the sources and the first common link varies are under investigation.

4.2.1 Scenario 1

Scenario 1 is used in order to examine the influence of the buffer size (and the target buffer sizes) as well as the effect of stops and restarts of some sources in the overall performance of the scheme. Also, under investigation is the respond of the scheme under the estimation of the maximum number of flows traversing the link, for different number of flows. Three main estimation cases were selected: the correct estimation, 50% overestimation and 50% underestimation. Furthermore, it evaluates the QoS guarantees offered to the Premium Class Service, such as packet losses, end-to-end delay and jitter

This network topology is separated in three sub topologies. The first one has 10 sources, from which the first one is a premium source requesting QoS (1 Mbps), whilst the rest of them are saturated IDCC ordinary sources. All the link delays in the topology are 1ms. Different buffer sizes are used in order to observe the response of the scheme under relatively small and large buffers. The selected buffer sizes were 200 and 1000 packets for both premium and ordinary class services. In the case of the 200 packets buffer size the premium target buffer was set to 30 packets and for ordinary was set to 150, whilst in the case of 1000 buffer size measurements have been made for two different settings. The first one uses 50 and 500 packets target buffer sizes for premium and ordinary services respectively and the second uses the same parameters as the 200 buffer size. The second and third topologies have the same environment as the one described above with the only difference to be the number of sources that have been set to 20 and 50 nodes respectively.

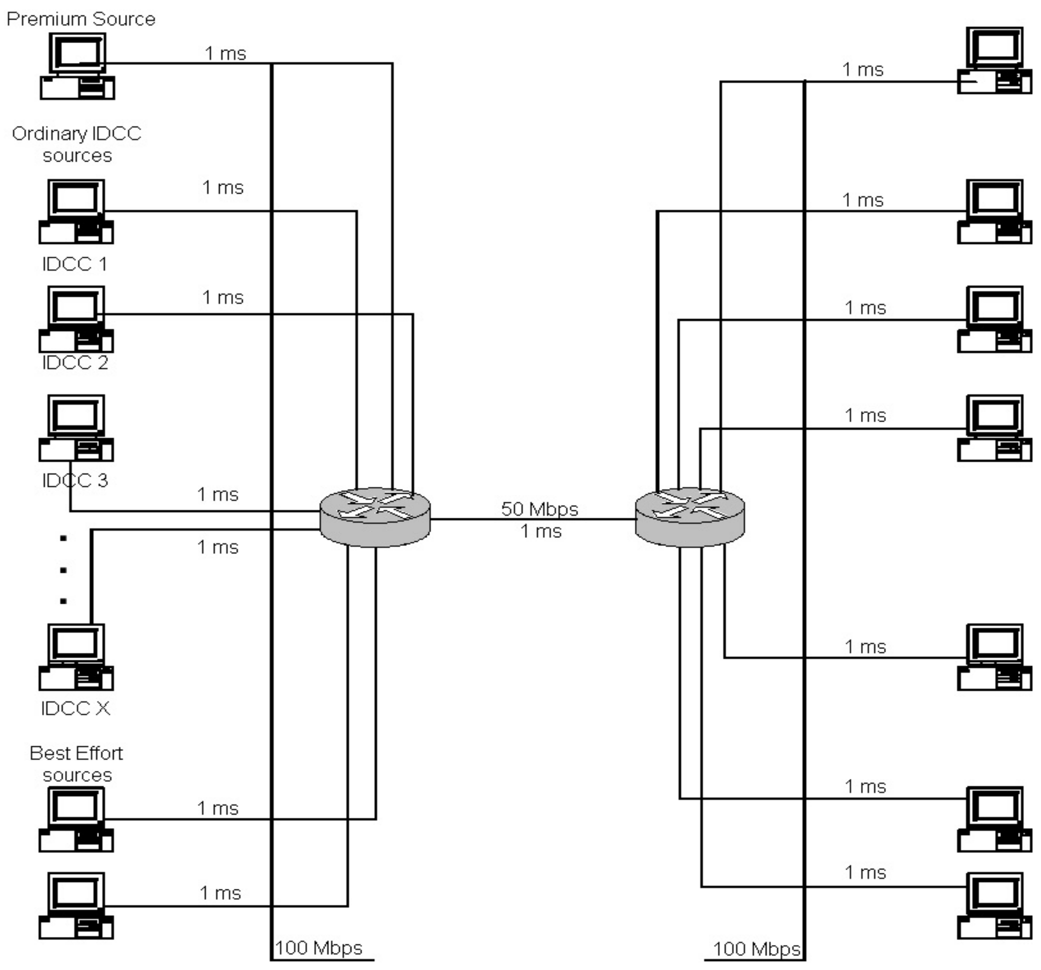


Figure 4.1 Network Topology Scheme for Scenario

The first case to study is the case where the buffer sizes were set to 1000 packets and the preferred sizes of premium and ordinary services are 50 and 500 packets respectively. Independently of the number of the sources the throughput (figure 4.2) of the underestimation in the number of flows is higher than the correct estimation which is higher than the under estimation, but the utilization of the link remains in very high levels in all the cases. In the case of 50 sources we examined the 100% overestimation in the number of ordinary sources, which has the lower utilization. The last graph of figure 4.2 compares the utilization of the link achieved by the different number of sources. 50 sources have the higher utilization, then the 10 sources and smaller the 20 sources but again the difference is very small. The above observation is very reasonable if we take in account the way that IDCC works. That is, it calculates the bandwidth remaining from the premium service and it divides it to the number of sources. So the less the number of sources in the topology the higher the rate of packets they send in order to utilize the link (figure 4.5).

The losses (figure 4.3) of the underestimate case are always more than the other cases that are very close. These results are also expected. All the estimations have one thing in common: the bandwidth left for the ordinary traffic since the bandwidth allocated for the premium traffic is fixed (1Mbps). In the underestimating case the bandwidth is supposed to be shared among fewer nodes than really exists, so the feedback is always overestimated and the sources send more packets, whilst in the overestimation case it is supposed to be shared among more nodes and the sources send fewer packets through the network. Also we must notice the small amount of time that the drops are happening that is a few seconds until the rate of the ordinary sources is stabilized.

In figure 4.4 we can see that even though we have wrong estimation on the maximum number of flows, the target ordinary buffer size is achieved in all the cases. That is probably due to the fact that one of the parameters that the ordinary controller uses in order to calculate the rate of the queue is the preferred target size of the buffer. So in the underestimation case the controller tries to slow down the rate because the maximum buffer size has been reached, whilst in the overestimation case it tries to send packets more rapidly in order to achieve the preferred buffer size. The only difference on the estimation is the time that the queue reaches the target length.

The ordinary traffic rate measurements are shown in figure 4.5. The first observation is that the different estimations do not converge to the same final state (the underestimation case has always higher rates). That is because at the first few seconds of the simulation and in the case of the underestimation scenarios we can detect many drops. Among those drops were also some requests for connection that is some connections could not be established and the final number of flows was smaller. As explained before the smaller the number of flows the higher the rate they send. Also a lot of oscillations happen in the first seconds of the simulations. This is due to the fact that only the maximum number of flows is known to the scheme and so it does need some time to converge. If the number of flows could be estimated during the simulation then these oscillations would be smaller and the losses should be less (the buffer would overflow for even less time).

The next measurements were done in order to analyse the QoS offered to the premium class service. From the theory in chapter one should expect that since we have resource reservation and admission control in the case of the premium traffic no premium packets should be dropped by the scheme. That is true and in none of the following scenarios there was a premium packet loss. The results of the end-to-end delay are shown in figure 4.6. In the first graph of that figure the end-to-end delay for 1 premium source is measured. Under the different estimations the end to end delay is exactly the same. That is because the propagation delay of the network is the same for all the estimations while the queuing delay is not affected by the rate that the ordinary packets are send into the network. The mean delay is 608.09 ms and the jitter is 3.85ms. The delay is very high (0.6 seconds) and premium traffic can not tolerate such delays. Of course that was again a predictable situation. The ordinary controller calculated the capacity needed for the premium traffic and sends packets towards the next node by that rate. In the scenario we had only one premium source sending 1Mbps and that is the reason for that delay. So we increased the number of premium sources and the end-to-end delay results for the cases of 10 and 30 premium sources are presented in the second graph. The more the premium traffic is send in the network (as long as they required capacity does not overcome the router capacity) the less the packet delay. If the traffic is less, fewer packets arrive at the buffers and the smaller the rate they depart from the it.

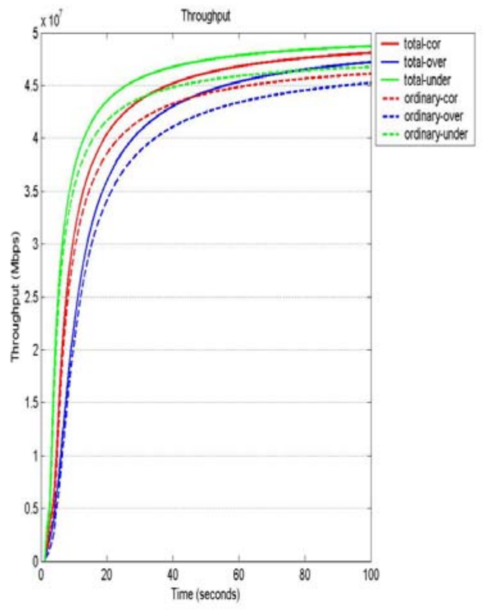
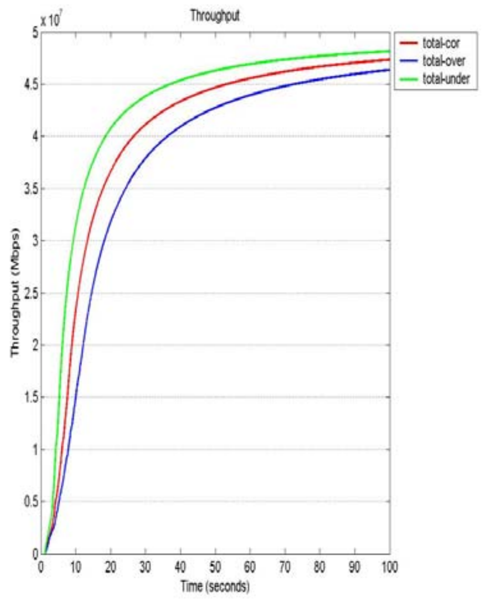
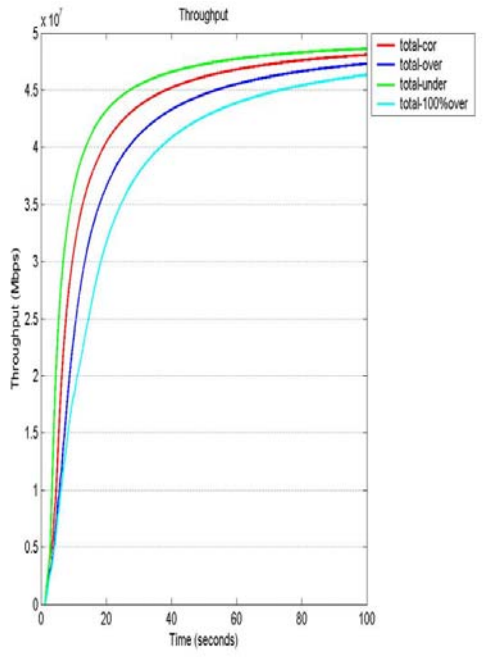
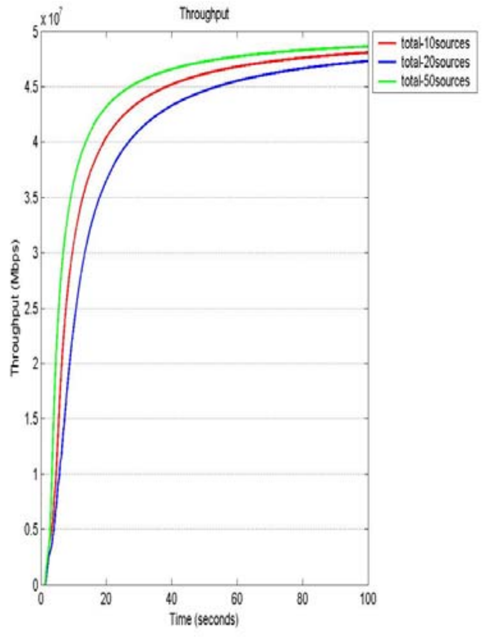
 Throughput 10 sources	 Throughput 20 sources
 Throughput 50 sources	 Combination of correct estimations

Figure 4.2 Throughput. Buffer size = 1000

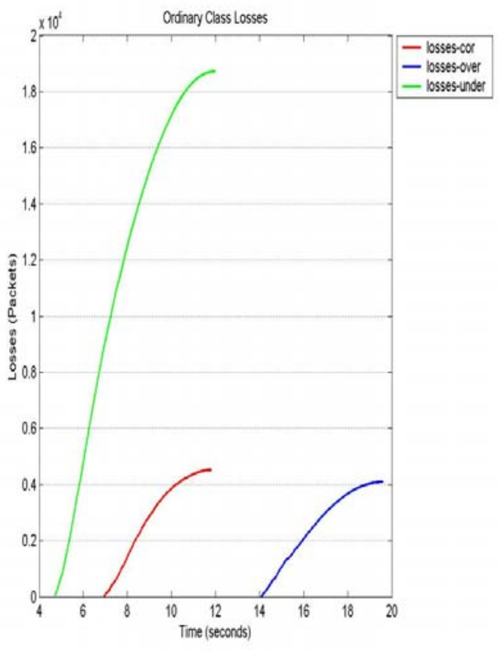
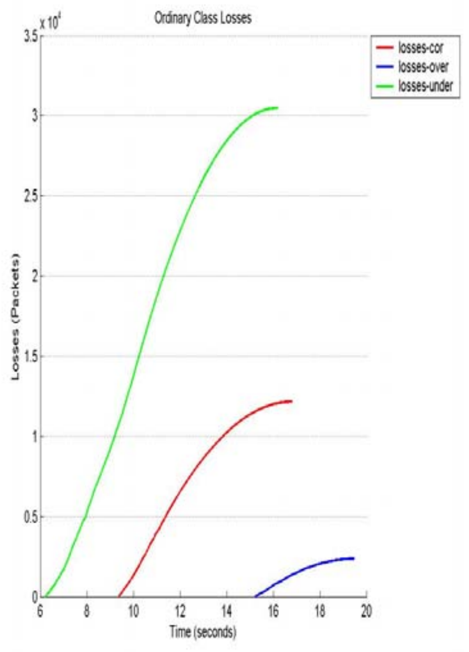
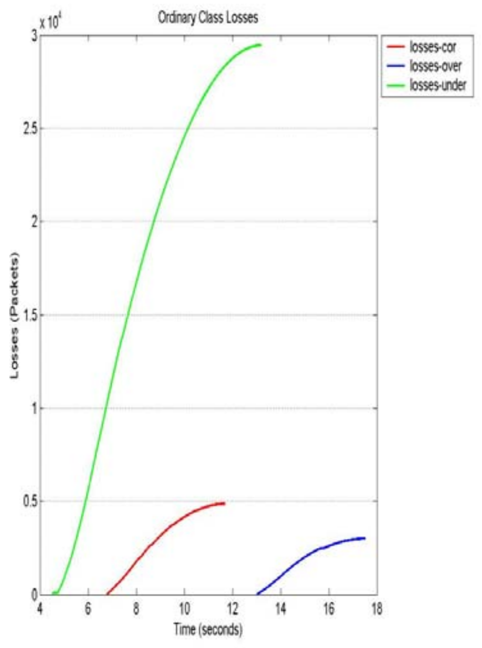
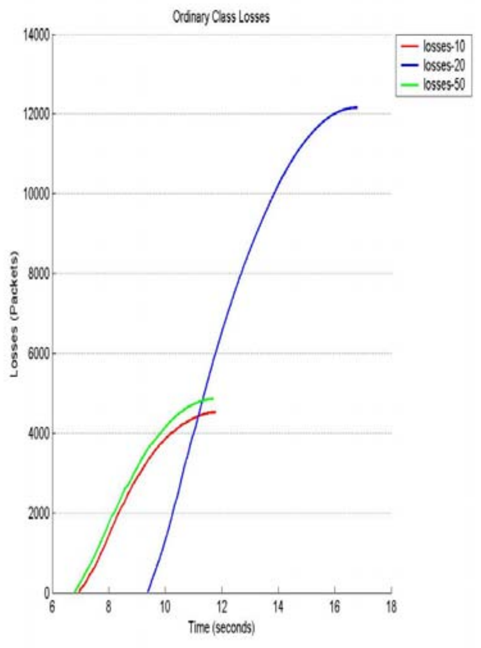
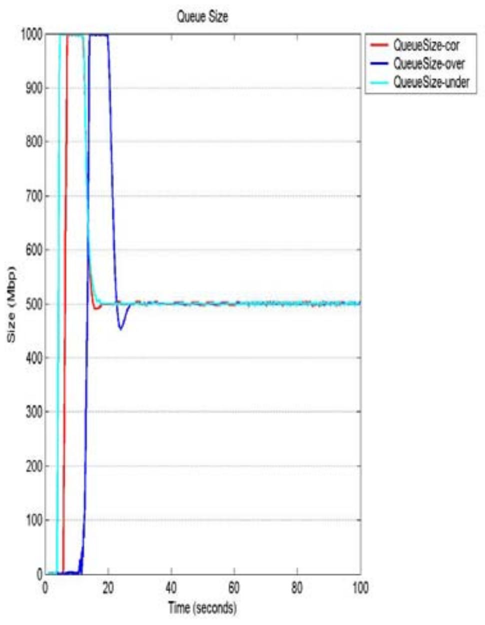
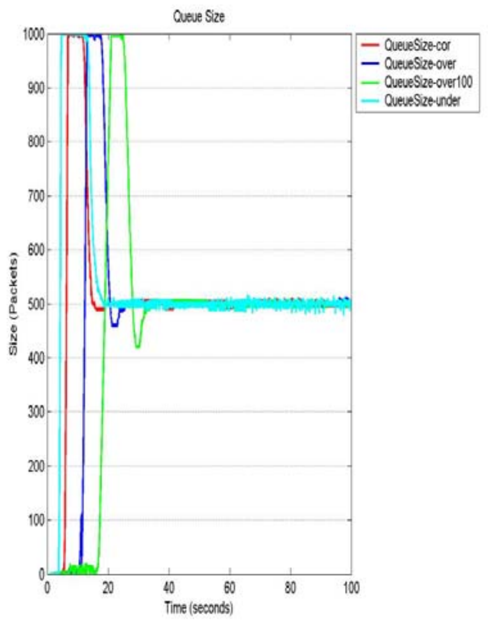
 Losses 10 sources	 Losses 20 sources
 Losses 50 sources	 Combination of correct estimations

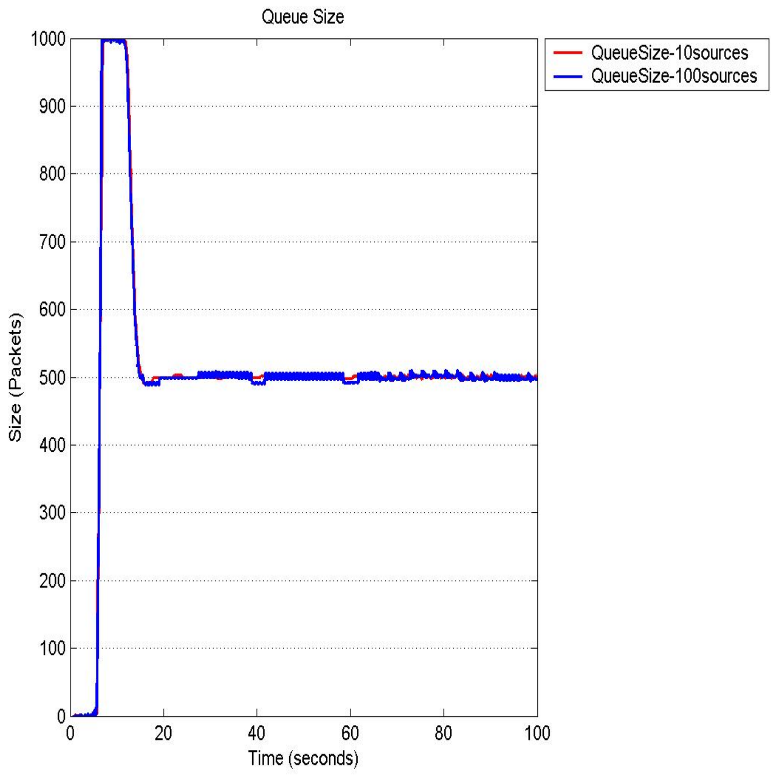
Figure 4.3 Losses. Buffer size = 1000



Ordinary Queue Size 10 sources

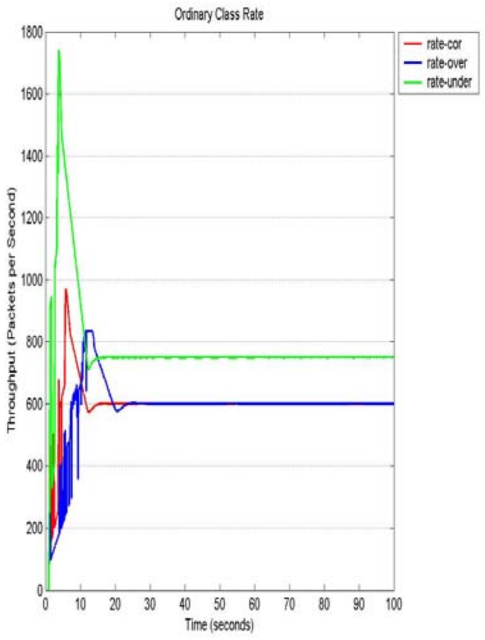


Ordinary Queue Size 50 sources

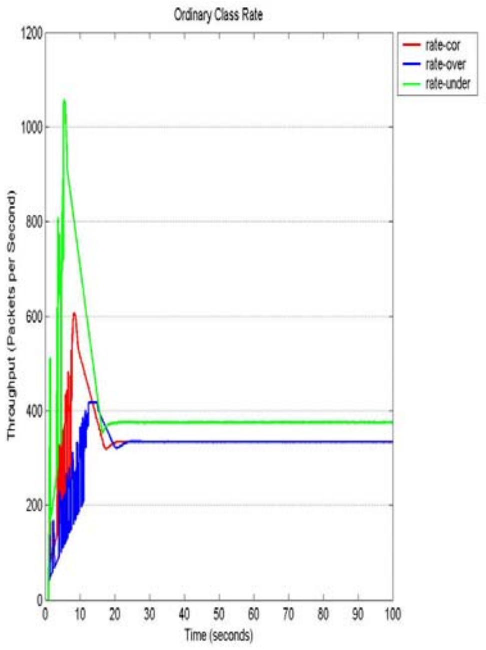


Combination of correct estimations

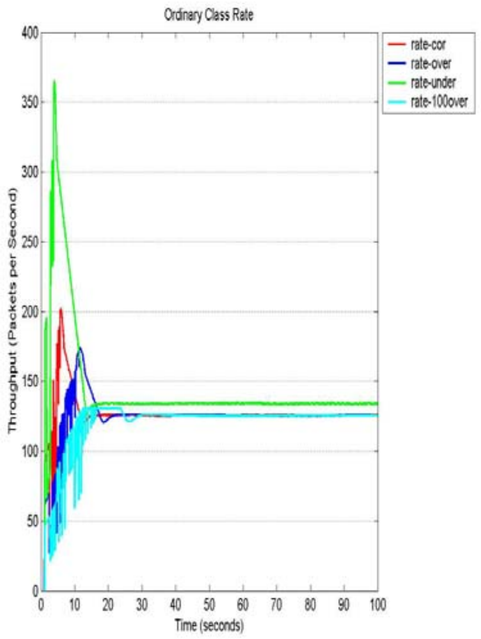
Figure 4.4 Ordinary Queue Size. Buffer size = 1000



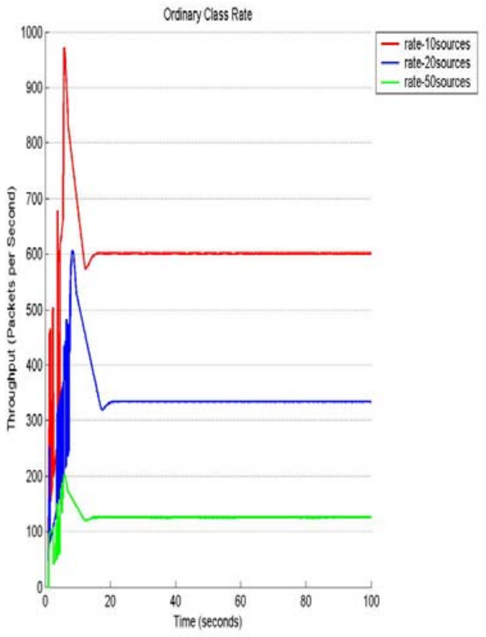
Rate 10 sources



Rate 20 sources

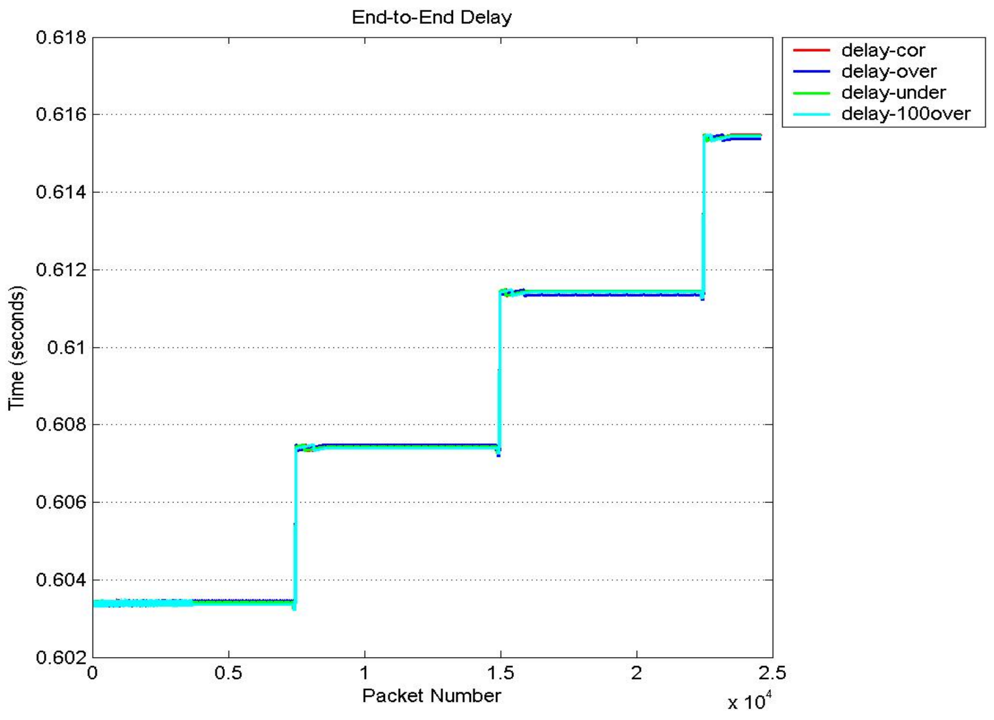


Rate 50 sources

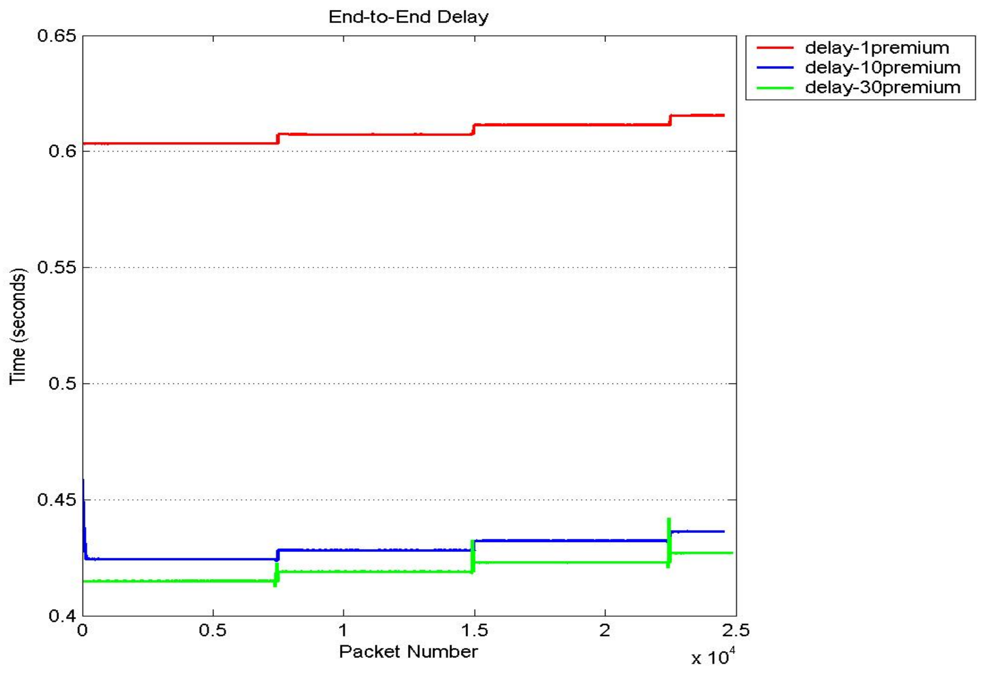


Combination of correct estimations

Figure 4.5 Rate of Ordinary Traffic. Buffer size = 1000



End to End delay in the case of 49 ordinary sources and 1 premium source



End to End delay for 1, 10 and 30 premium sources

Figure 4.6 End to End Delay for the Premium Traffic. Buffer size = 1000

Next we will examine the case of which the buffer size is 200 packets and compare it with the previous one. The simulation time in this case is 150 seconds instead of 100 seconds, because time is given in order to attain the queue stabilization. Under a quick observation of the curves of the graphs produced, someone could state that the overall behaviour is the same as the previous. But with a closer look at the numbers that observation is not so accurate. The utilization differs more than the previous case and except the underestimation case we do not notice an immediate increase of the utilization, especially in the cases of overestimations (figure 4.7). The losses (figure 4.8) are multiples compared with the previous case. For example in the case of the 50 sources, when we make the correct estimation of the sources we loose 69903 instead of 4858 packets and in the case of the underestimation things are worst: 245507 packet losses instead of 29439 (table 4.1). Furthermore, the losses do not happen in a few seconds, but they are spread all over the simulation time. The bad performance is witnessed also in the ordinary queue size (figure 4.9). The expected queue size is reached after the half simulation time passes and in the case of the 50 sources at the end of it. The bad performance of the scheme in small buffers is also realized by the effort done in order to regulate the rate of the ordinary traffic (figure 4.10). The rate achieved after the stabilization is the same with the one of the 1000 buffer, but when the simulation time is 100 seconds (before this was done in the first 20 seconds). The last measurement was the delay of the ordinary traffic service (figure 4.11). We can observe that delay is not influenced by the estimation of the sources. Also the behaviour of the delay is independent of the number of sources. The difference in the graph is because in the case of 10 sources each source sends more packets than in the case of 50 sources. With a plot of the same results according to the time the graphs could look very similar. Figure 4.12 gives the comparison of the end-to-end delay for the premium traffic for the cases of 200 and 1000 buffer sizes. In the case of 200 packets size, the delay is smaller and this is due to the decrease of the expected size of the queue (is set to 30 instead of 50 packets of the previous case). Thus the packets stay less time in the queue and that leads to smaller end-to-end delays.

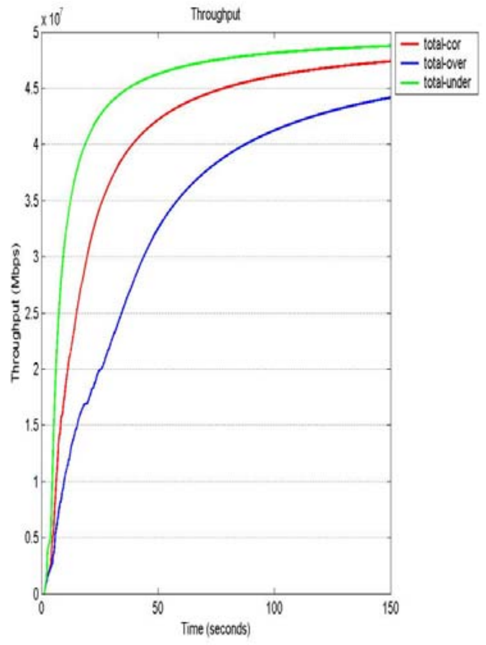
We must note that in the case of the premium traffic no losses were found.

		10 sources	20 sources	50 sources
buffer size = 1000	over estimation	4093	2371	2994
	under estimation	18719	30476	29439
	correct estimation	4521	12166	4858
buffer size = 200	over estimation	48814	47231	44578
	under estimation	260558	286338	244604
	correct estimation	73148	71467	69903

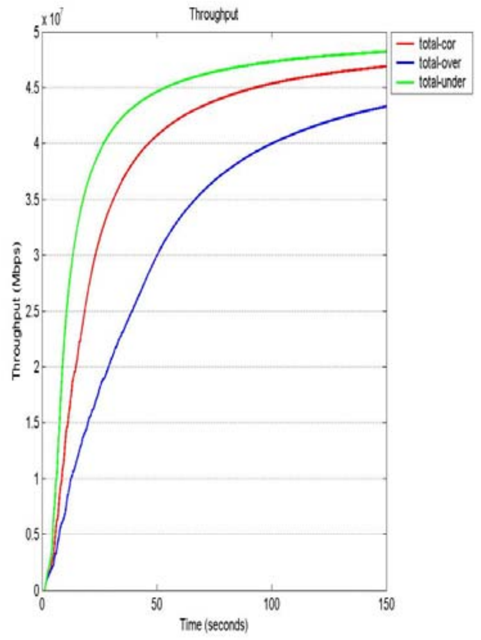
Table 4.1 Losses for buffer sizes 200 and 1000 under different estimations

The bad performance in the latter case was due the small buffer size of the routers. Because of the small buffer size, the queue very quickly was full. As we can see from figure 4.8 and table1, in the case of the correct estimation we have more than 70000 packet losses happening during the 20th and the 80th second of the simulation. For that reason no proper rate feedback arrives at the ordinary traffic sources (the packets showing congestion are lost), so the sources continue sending with very high rates. The correct rate reaches the sources after 90 seconds of simulation but already the network had very bad performance. The role of the preferred ordinary buffer size had no influence in the bad performance of the scheme and this is shown in the graphs comparing the previous cases with a new one, which uses 1000 packet buffer sizes and the preferred ordinary buffer size was set to 150.

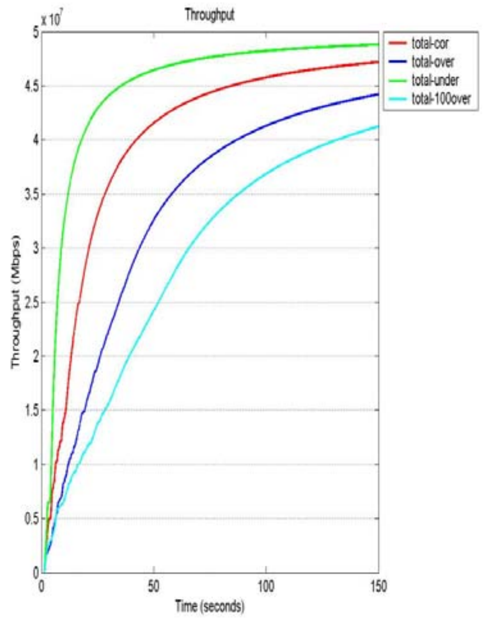
By comparing all the cases we led to very important conclusions. Firstly, by the throughput graph in figure 4.13 we can state that the throughput is only influenced by the preferred buffer sizes. The higher this target is, the higher the throughput achieved at the congested link. Another very important observation is that the number of losses is very small for bigger buffer sizes. Also having smaller preferred buffer size, leads to less drops, because the scheme tries to keep the queue to smaller levels. Thus the rate is increasing gradually instead of a very sudden change of the rate that leads to a high number of drops in very little time (figure 4.14).



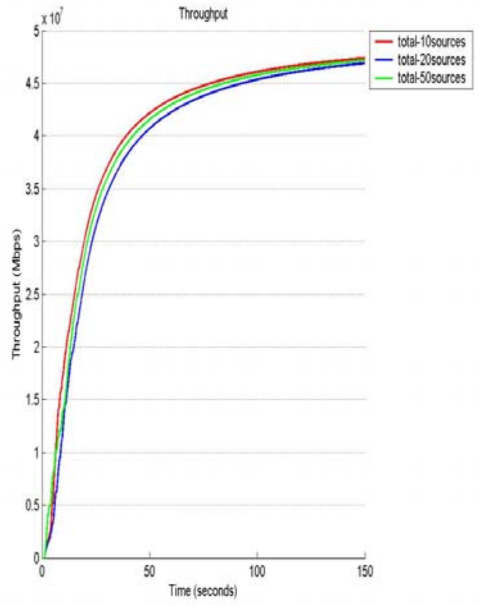
Throughput 10 sources



Throughput 20 sources

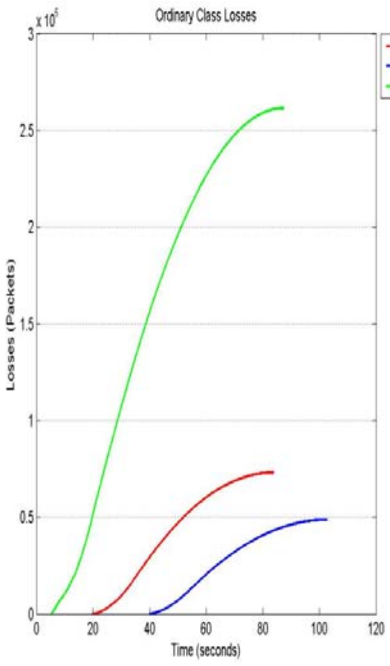


Throughput 50 sources

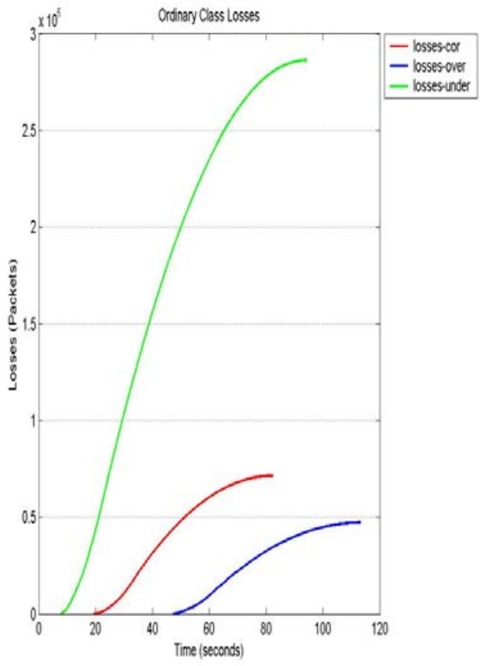


combination of correct estimation

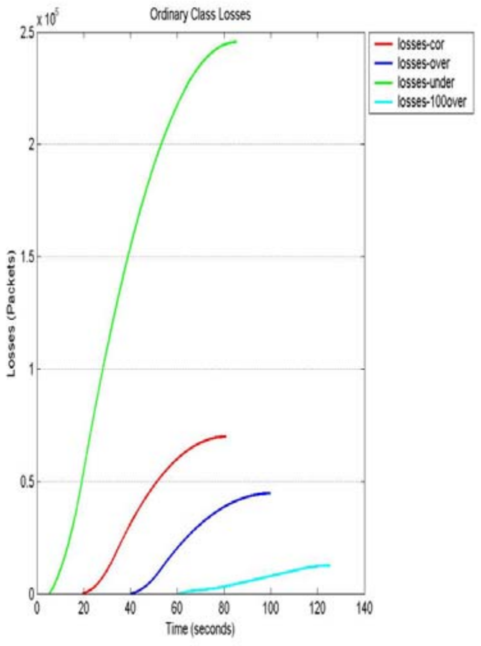
Figure 4.7 Throughput. Buffer size = 200



Losses 10 sources

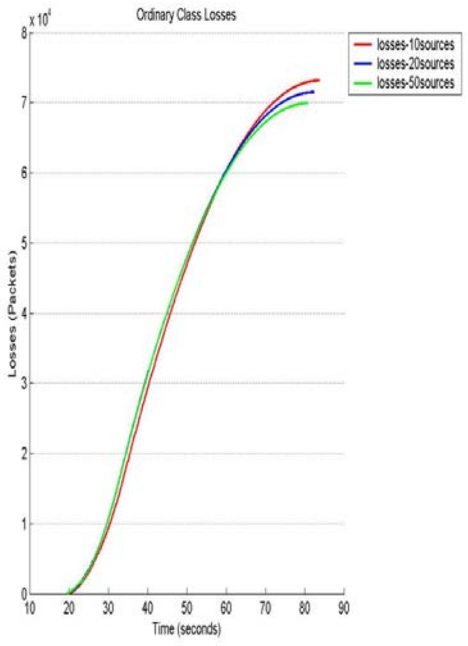


Losses 20 sources



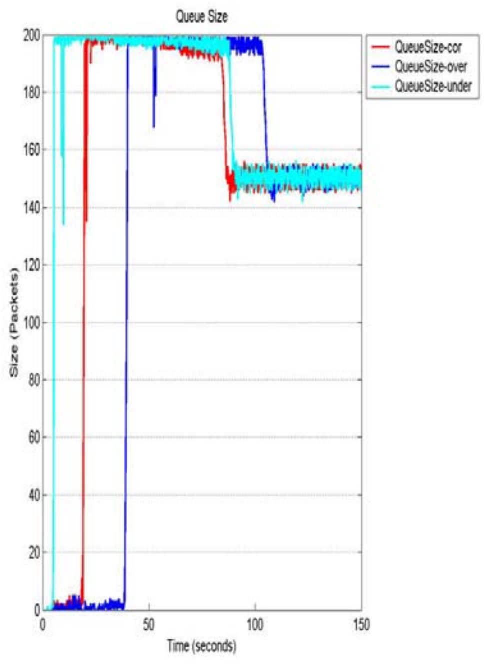
f

Losses 50 sources

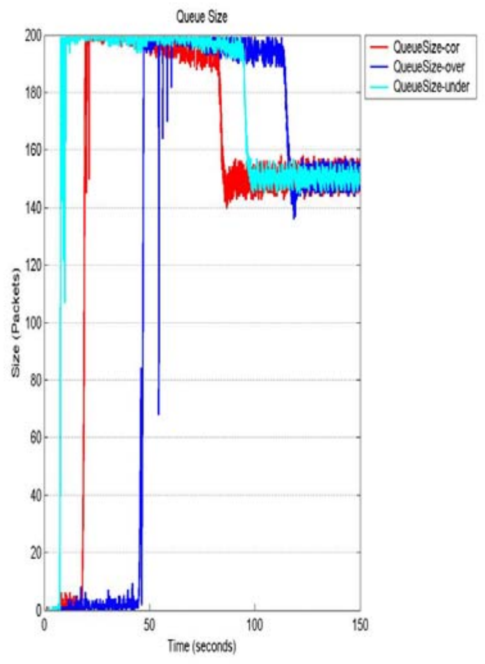


Combination of Correct Estimations

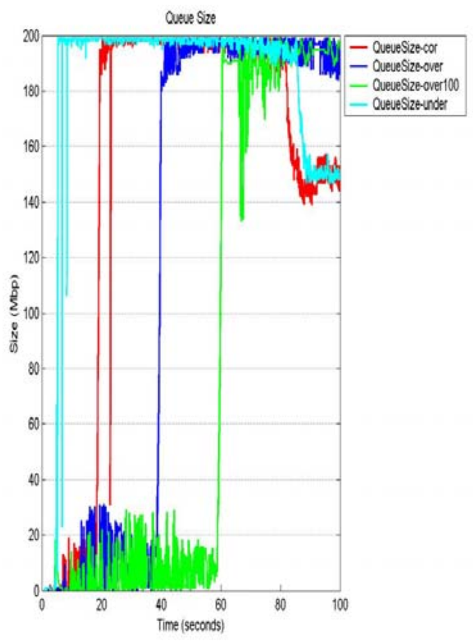
Figure 4.8 Losses. Buffer size = 200



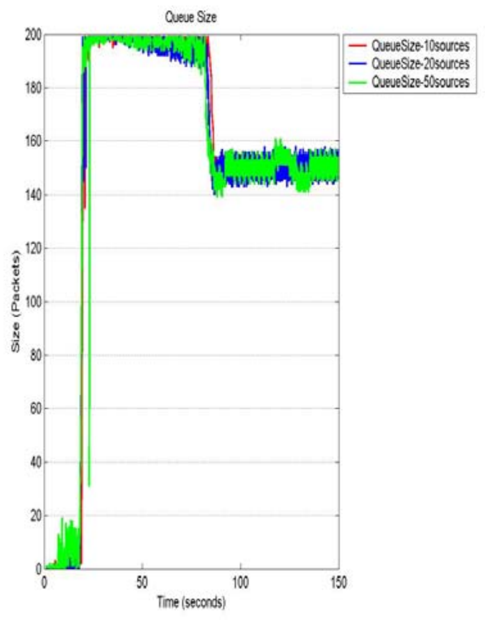
Ordinary Queue Size 10 sources



Ordinary Queue Size 20 sources

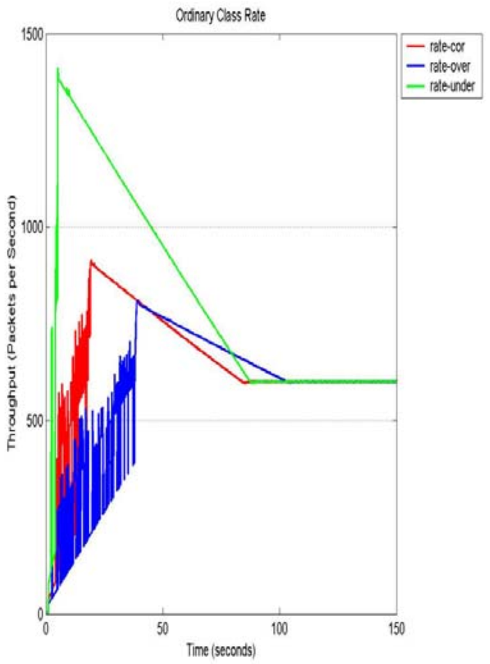


Ordinary Queue Size 50 sources

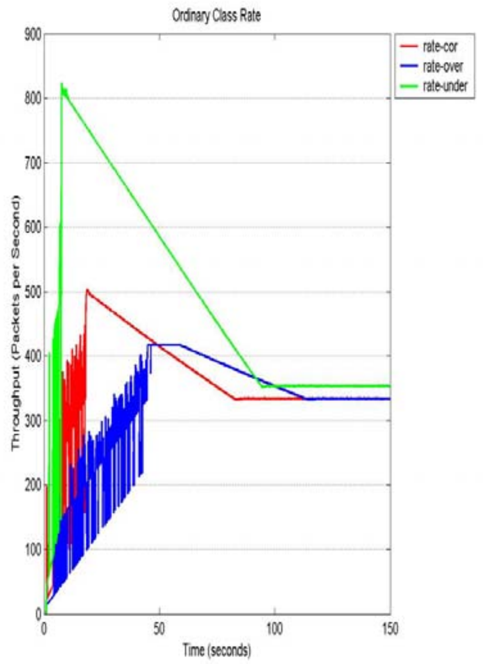


Combination of correct Estimations

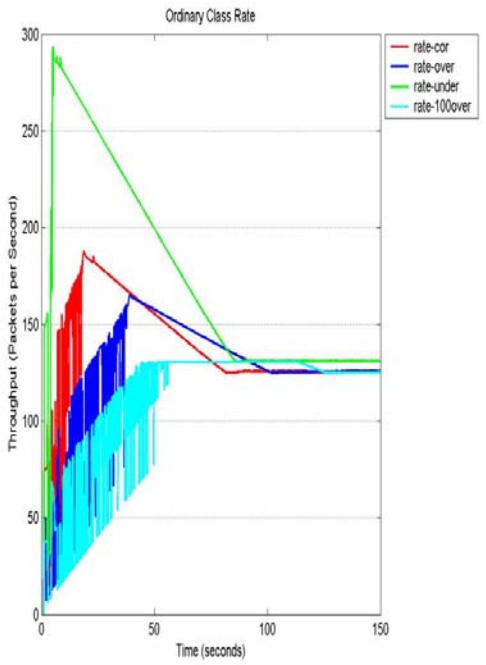
Figure 4.9 Ordinary Queue Size. Buffer size = 200



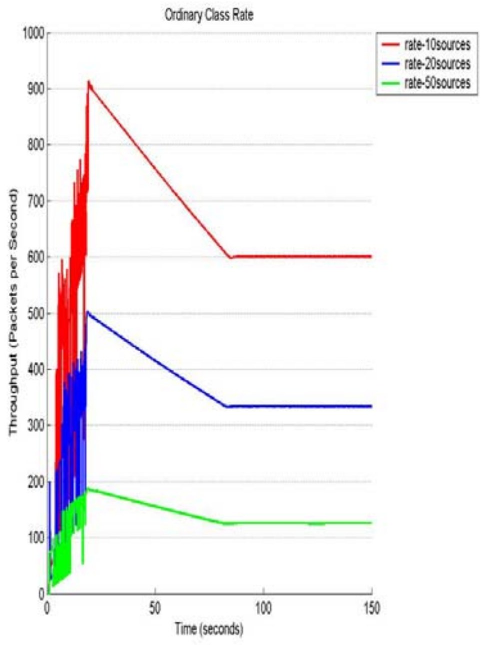
Rate 10 sources



Rate 20 sources

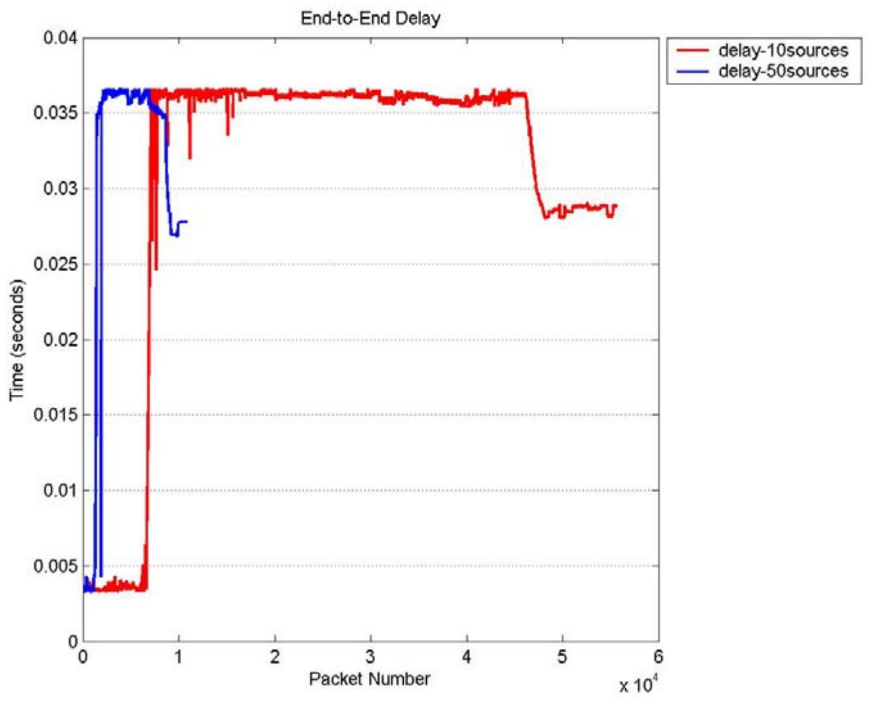


Rate 50 sources

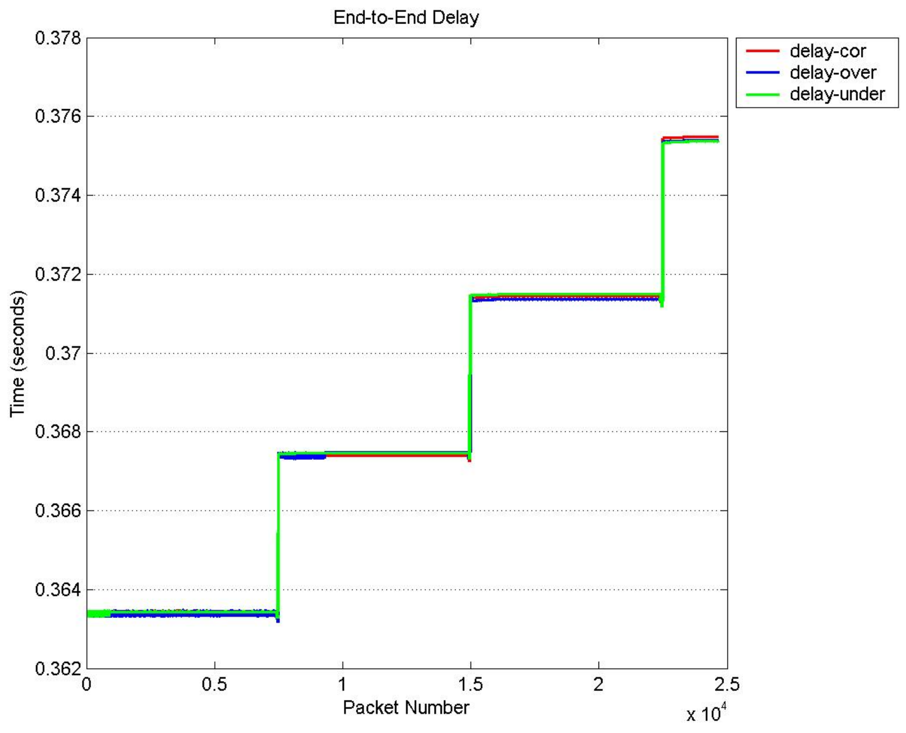


Combination of correct estimations

Figure 4.10 Ordinary Traffic Rate. Buffer size = 200



Ordinary traffic delay in the cases of 10 and 50 sources



Ordinary traffic delay 50 sources

Figure 4.11 Traffic Delays. Buffer size = 200

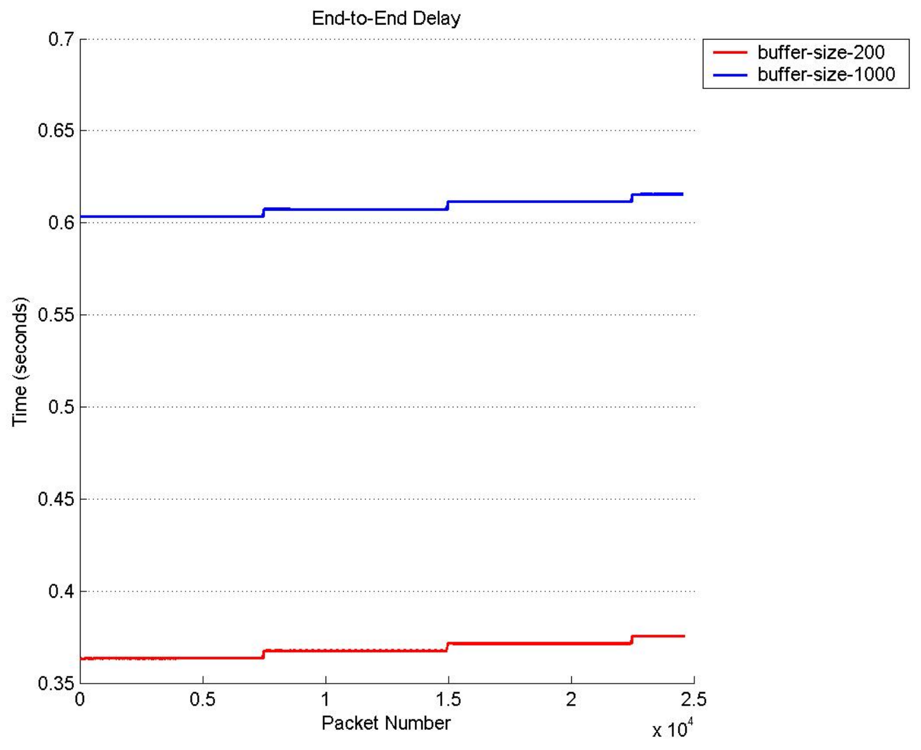


Figure 4.12 Traffic Delays for Ordinary traffic. 200 and 1000 buffer sizes

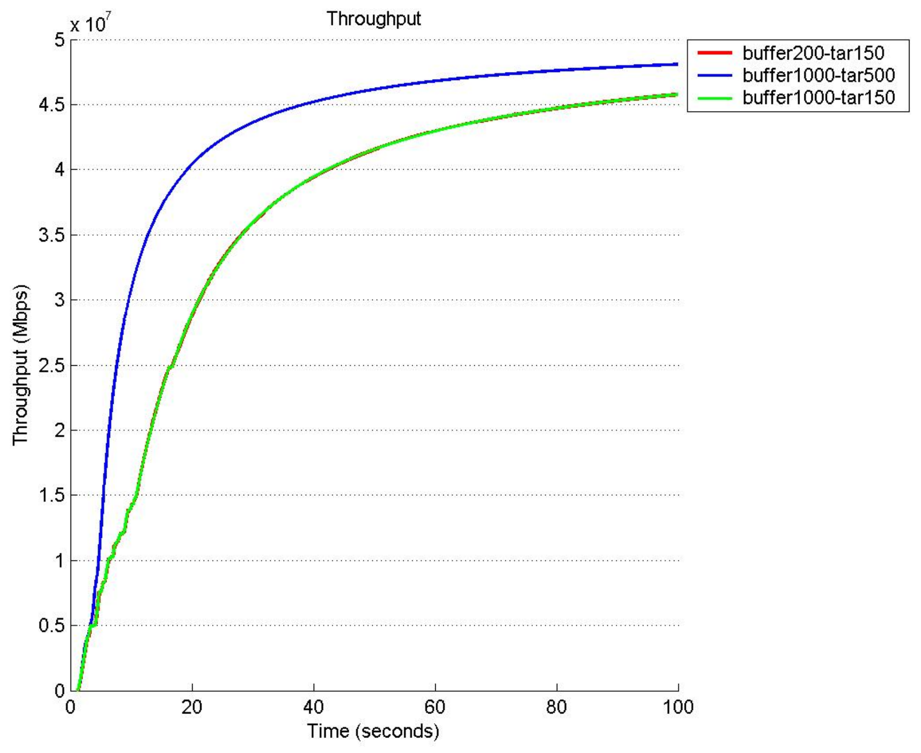


Figure 4.13 Throughput for different buffer parameters

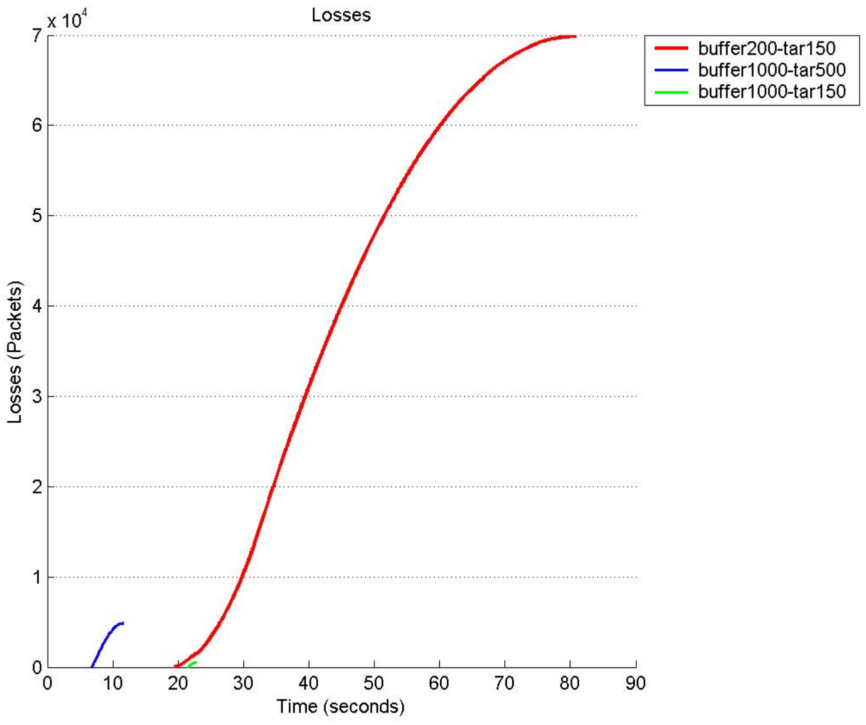


Figure 4.14 Losses for different buffer parameters

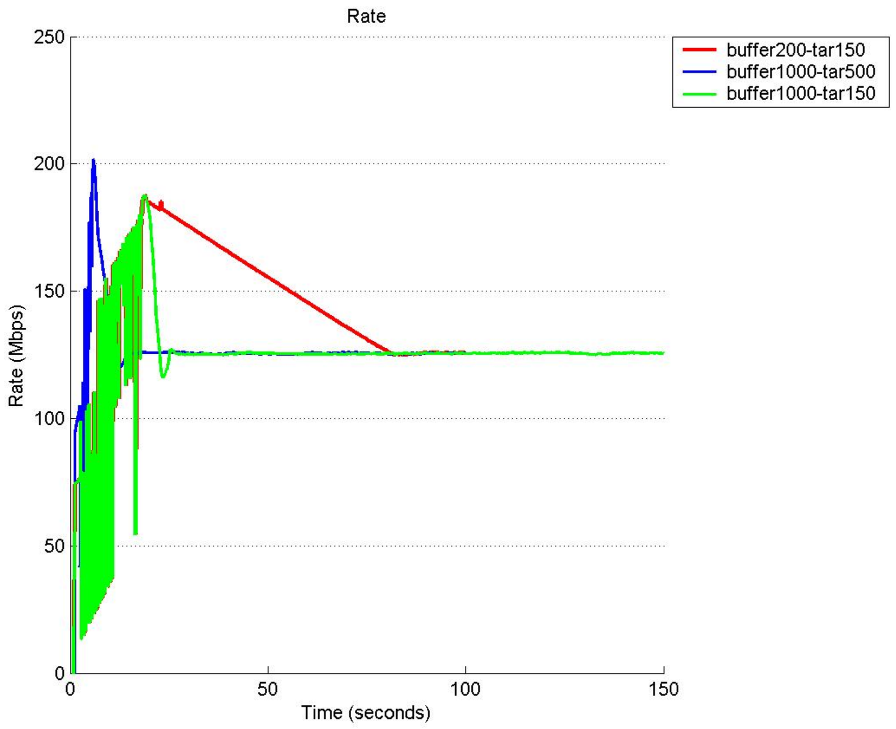


Figure 4.15 Rate for different buffer parameters

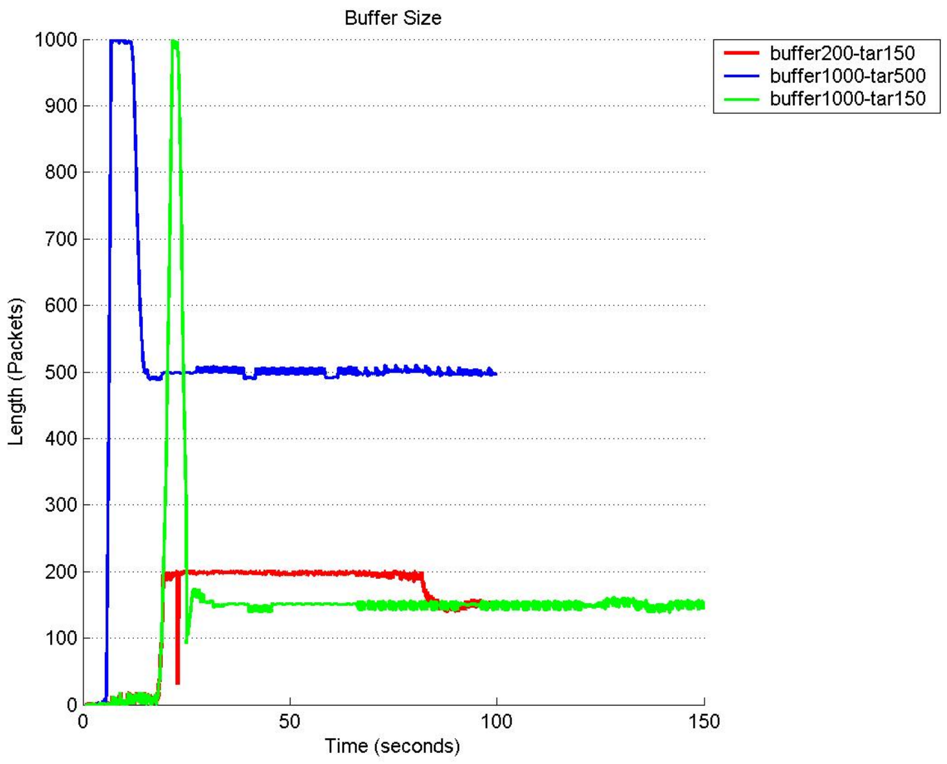
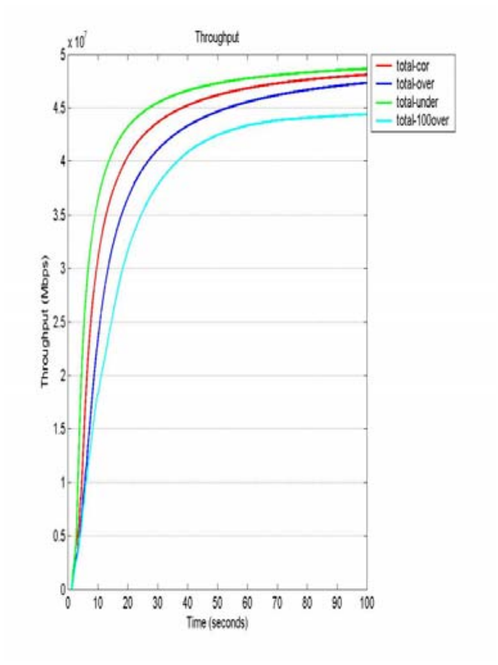


Figure 4.16 Buffer size for different buffer parameters

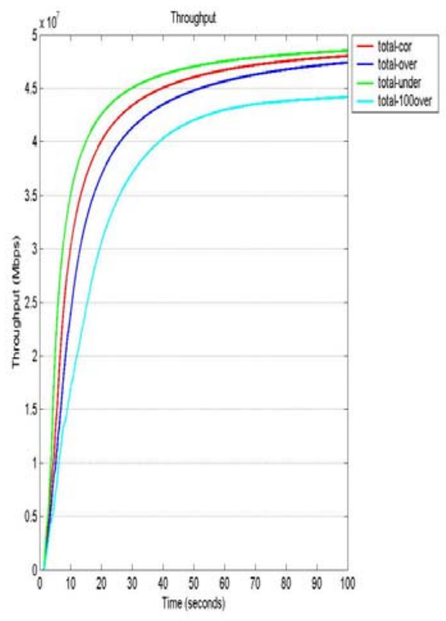
Then using some of the previous configurations we created some scenarios for testing the response of the congestion control scheme after the stop and restart of some sources. The experiments were tested in the cases of 20 and 50 sources for 1000 packets buffer sizes, where the preferred ordinary buffer size was 500 packets. In the case of the 50 sources we make a test to observe the response of the queue in different delays (1 ms and 20 ms) under the same traffic conditions.

In the first case a simpler scenario is used, where a stop occurs every 10 seconds, from the 20th up to the 70th second, that is six stops in total. As can be seen from figure 4.17, if we exclude the case of the 100% overestimation, there is not obvious difference in the link utilization due to the stop of some ordinary sources. The IDCC scheme reacts very quickly to each stop calculating the appropriate rate in order to keep the buffer size as close to the preferred buffer size (figure 4.19). Furthermore there are no losses of ordinary packets (figure 4.18) due to the stop of the sources. All the packet drops happen at a very small amount of time in the beginning of the simulation and once the rate has converged to a stable state, we do not have any more losses. In the case of the 100% overestimation of the maximum number of sources, the scheme does not react at all and this is reasonable if we take in account that it believes that much higher number of nodes are sending packets. The other estimations respond very well when a stop occurs and the increase in the rate is the same for all the cases.

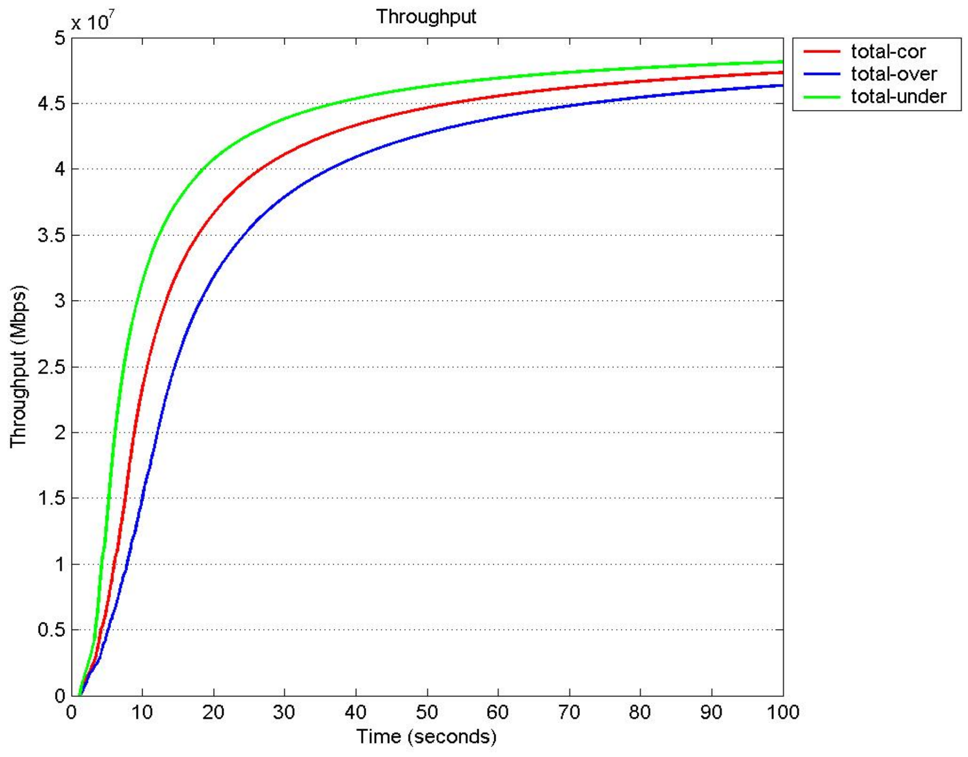
Next a more complicated scenario was used, where stops could occur simultaneously. We also had some premium sources starting during the simulation time in order to observe if the scheme would decrease the rate that ordinary packets are send to let the premium traffic pass. Like the previous "stops" scenario, the throughput is very high and it does not show to be influenced at all by the behavior of the sources (figure 4.20). The losses for one more time are happening in the first few seconds in the beginning of the simulation, where the rate was not converged to the proper one (figure 4.21). The only event that witnesses the simulation scenario is the accents shown in the buffer size graph (figure 4.23) and the difference in the rate during the simulation scenario time (figure 4.22). In the 25th second two premium sources request for QoS and the rate of the ordinary class is reduced, whereas at the 60th second the two sources are stopped and the rate goes immediately at the same value as it was before the 25th second. Then at the 80th, 110th and 130th second a number of ordinary sources stopped for sending packets in the network so the rate of the rest has increase (the increase was proportional of the number of sources that has been stopped), whereas at the 150th second three ordinary sources have been started and so on.



50 sources 1ms link

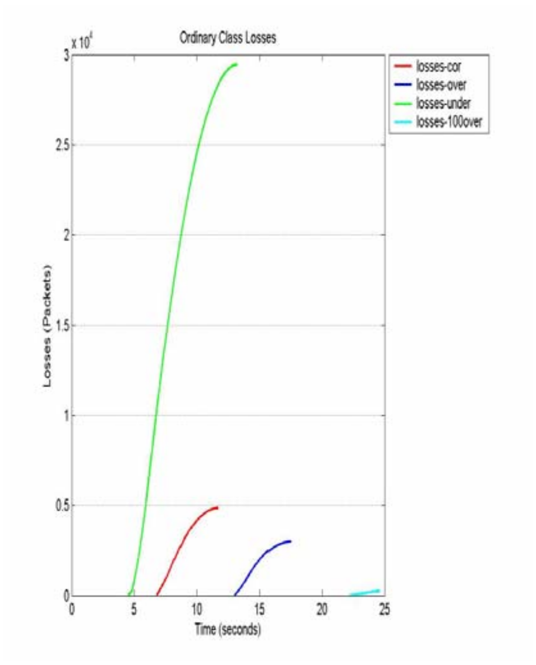


50 sources 20ms link

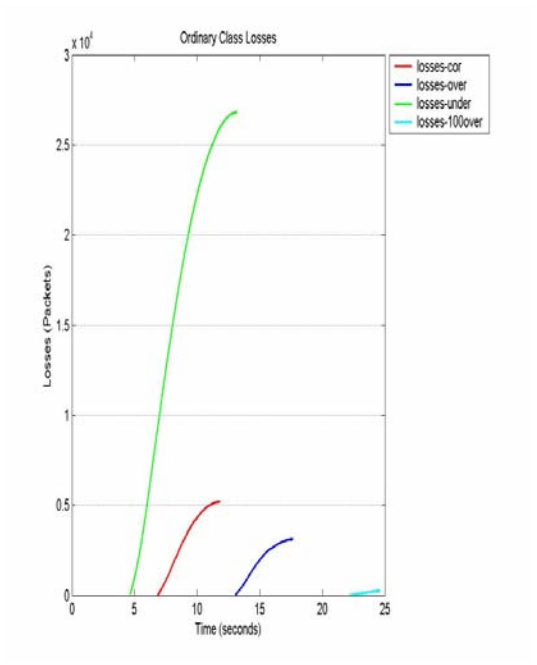


20 sources 1ms link

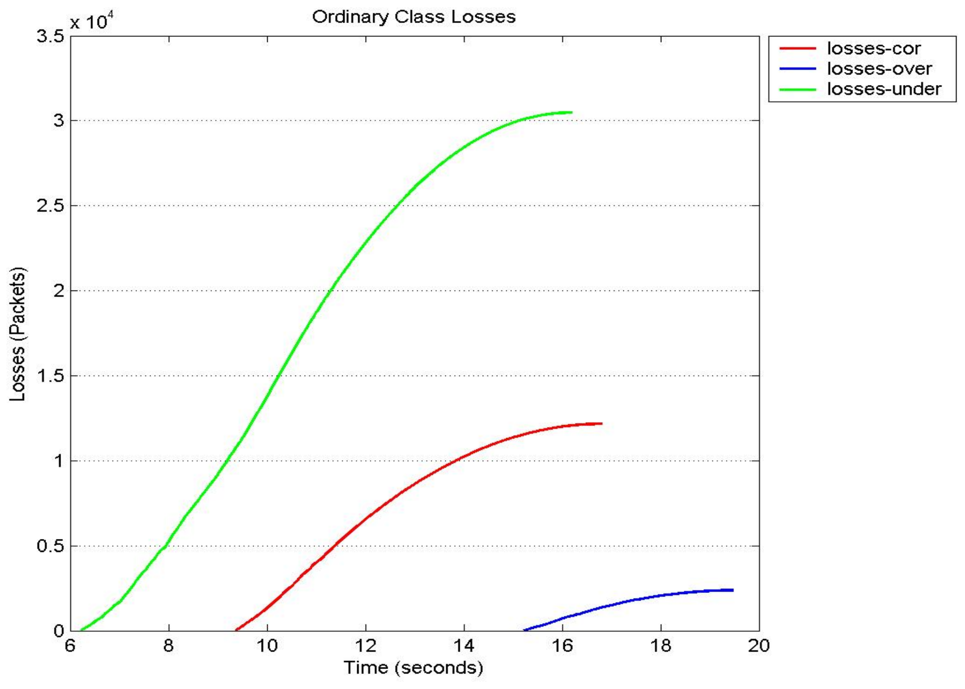
Figure 4.17 Throughput with sources being stopped



50 sources 1ms link

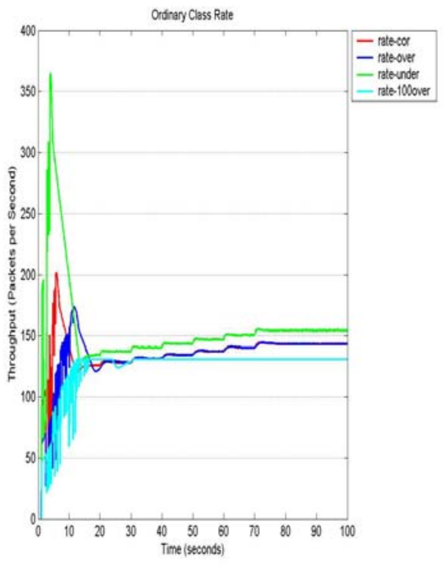


50 sources 20ms link

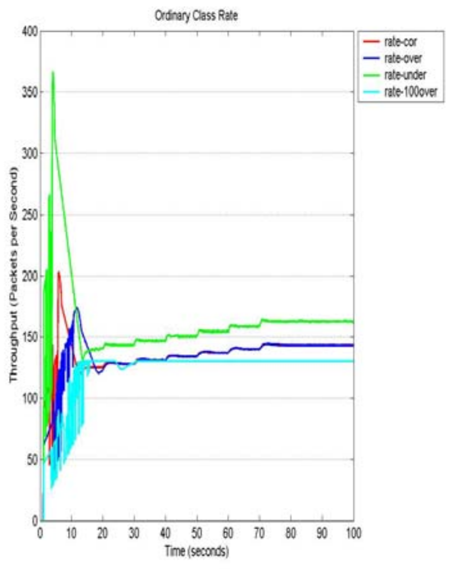


20 sources 1ms link

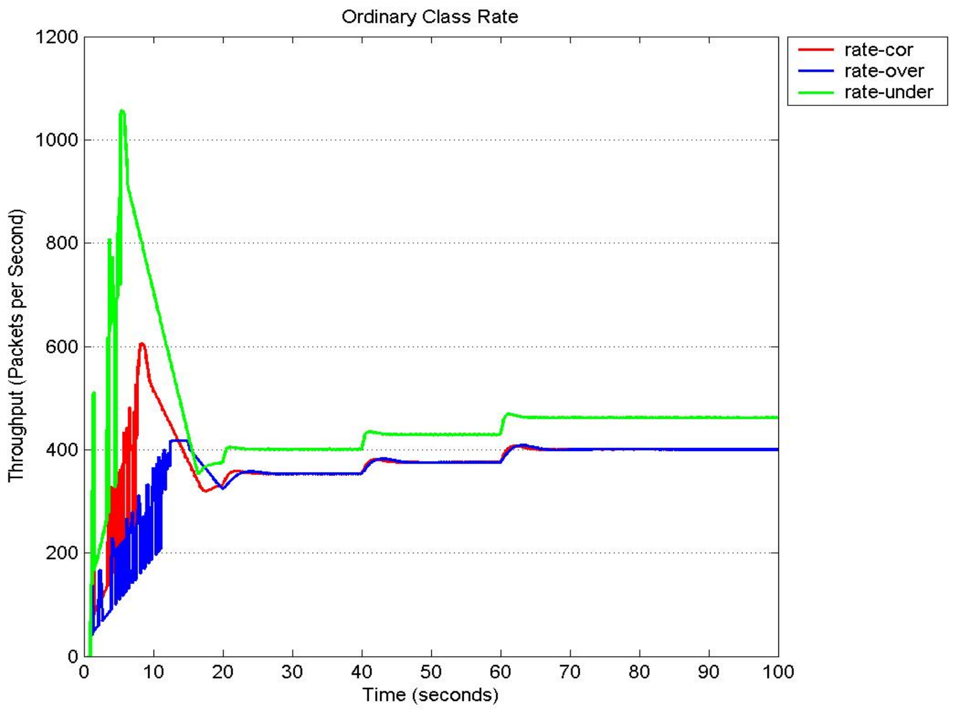
Figure 4.18 Losses of ordinary sources with stops



50 sources 1ms link



50 sources 20ms link



20 sources 1ms link

Figure 4.19 Rate of ordinary sources with stops

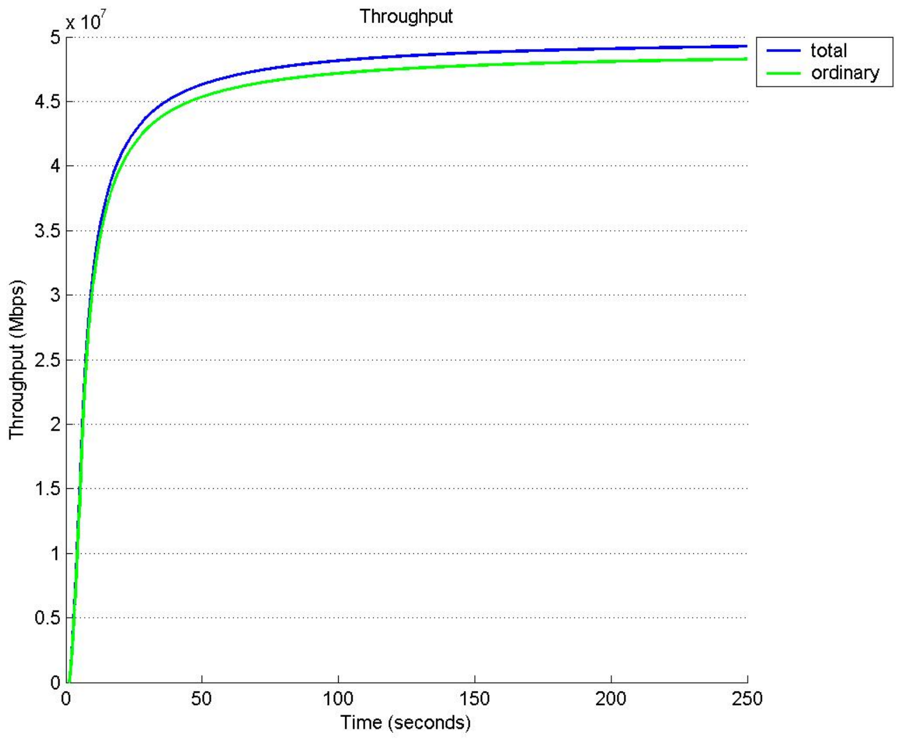


Figure 4.20 Throughput of ordinary sources with stops

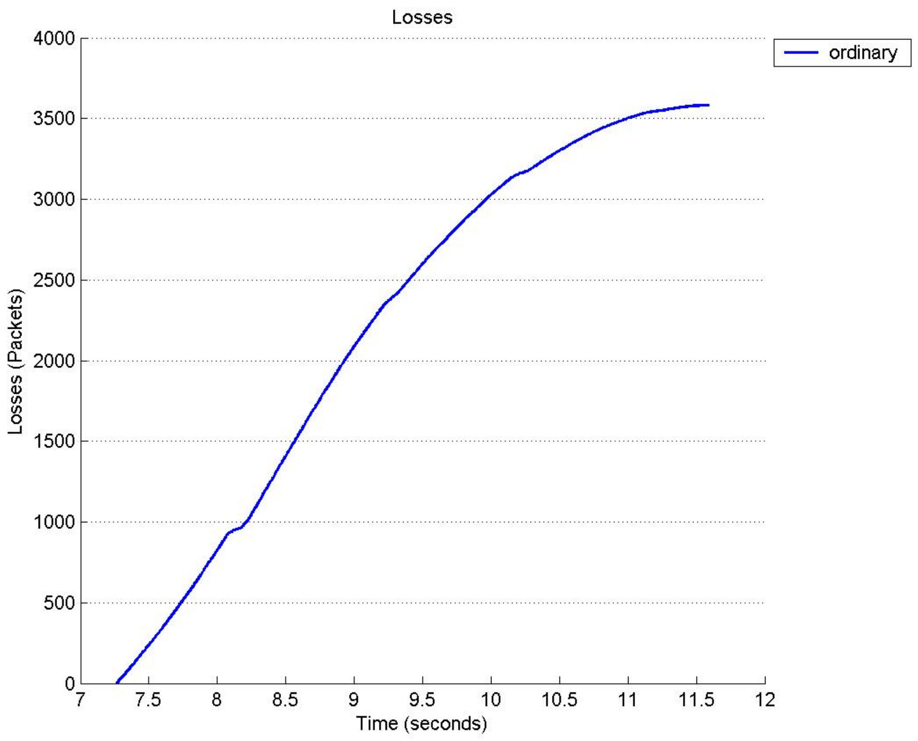


Figure 4.21 Losses of ordinary sources with stops

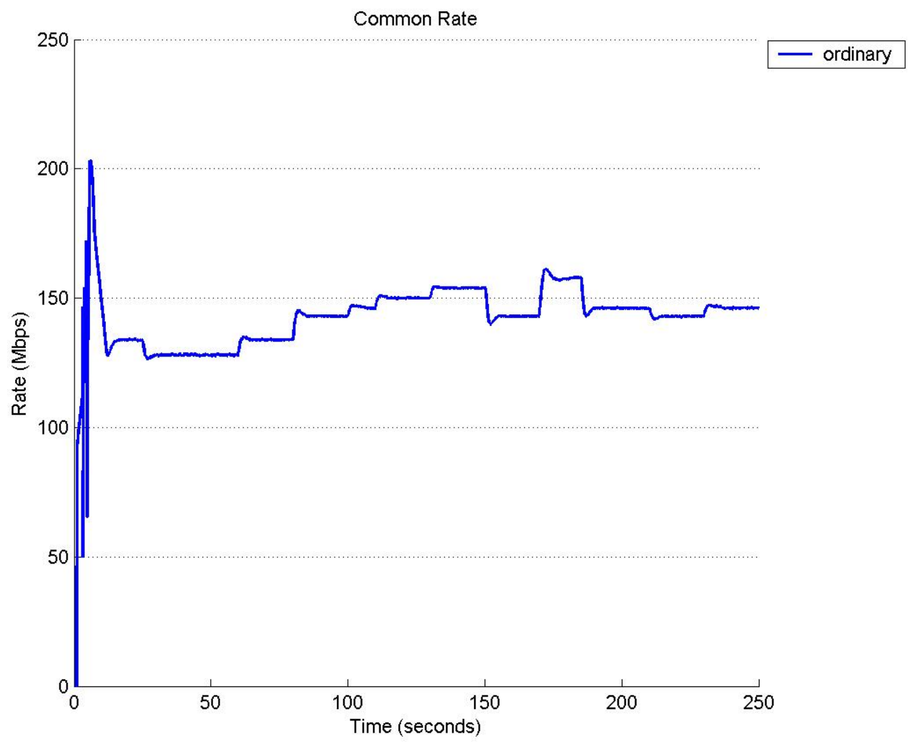


Figure 4.22 Rate of ordinary sources with stops

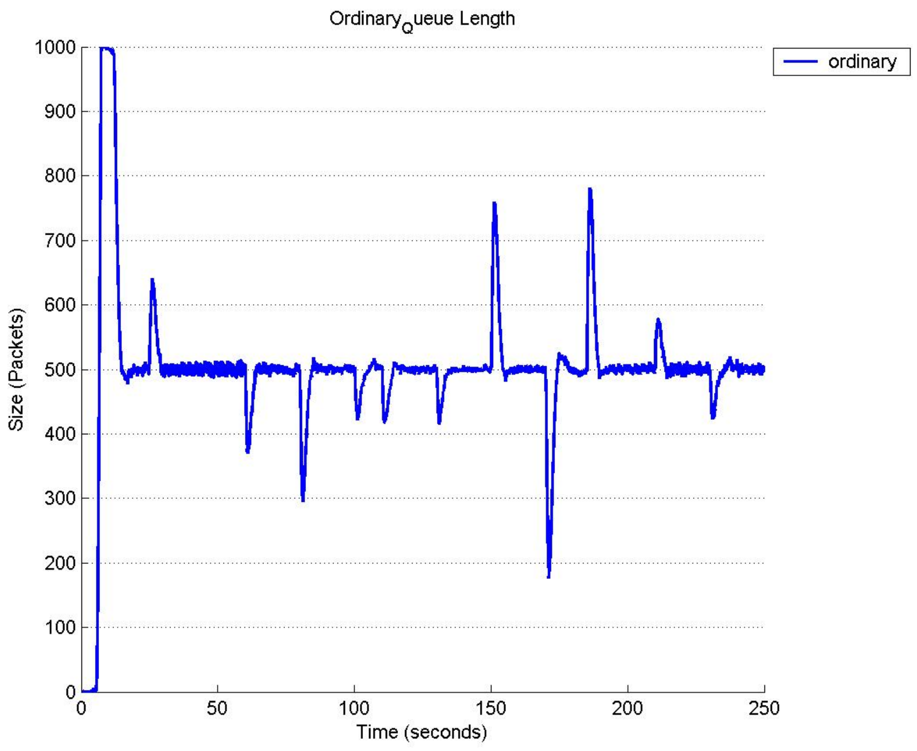


Figure 4.23 Ordinary Buffer Size when stops occur

The last measurement done in this scenario was about the fairness of the scheme. In order to do that we compared the performance of four, random chosen ordinary sources (the 1st, 2nd, 42nd, and 43rd source). The first two sources had begun sending packets in the network the first second of the simulation, whereas the latter two at the sixth second. From figure 4.24 we can see that the sources beginning at the same time get the same throughput. Also, the sources that begun later in the simulation scenario achieved less throughput than the ones started in the beginning. This is logical, because since they send with the same rate for less time they should send less packets in the network. The losses graph (figure 4.25) comes to contribute to the previous observation. A difference in the number of losses per flow exists, but it is very small that does not have any effect in the overall performance of each flow.

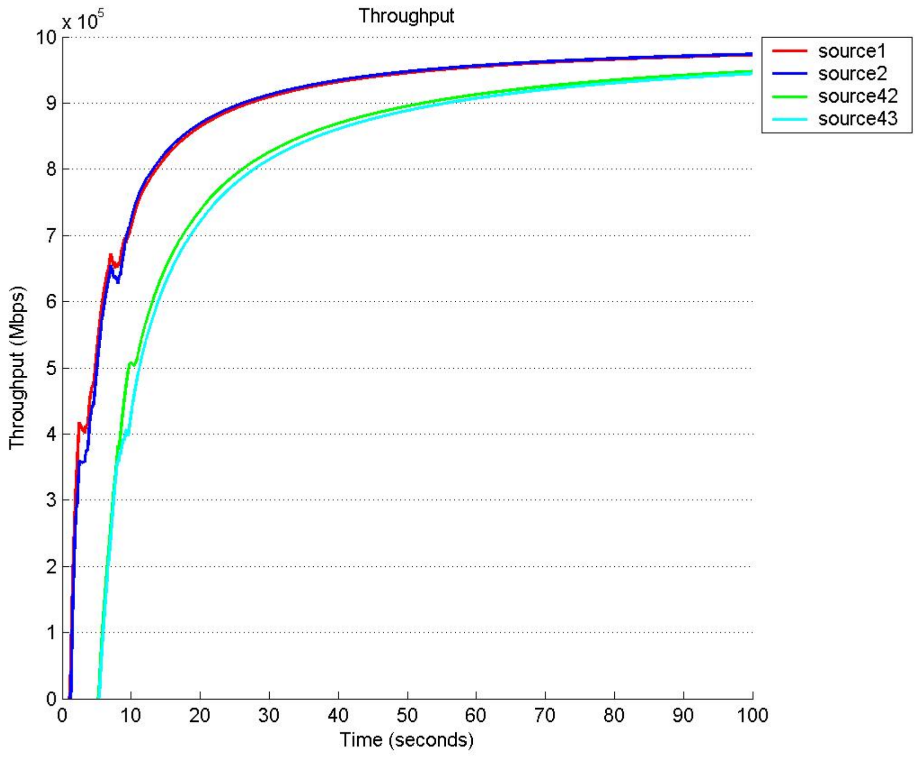


Figure 4.24 Throughput of different ordinary sources

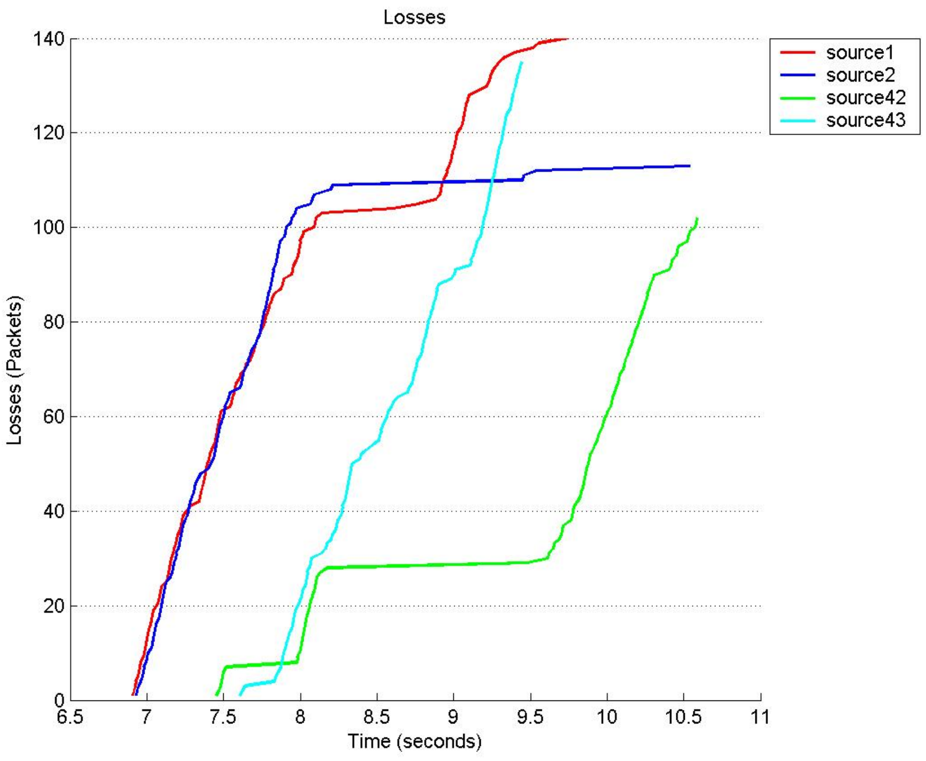


Figure 4.25 Losses of different ordinary sources

4.2.2 Scenario 2

Scenario 2 (figure 4.26) is used in order to see the response of the queue (using different buffer sizes) when the sources are gradually starting and stopping and the maximum number of flows is estimated. In this network topology we have fifty-one sources (all have the same link delay of 1ms) sending traffic to fifty-one other destinations through a link connecting two routers. The first source traffic is tagged as premium class traffic. The next forty-nine sources are IDCC sources sending ordinary traffic. The bandwidth of the link between the two routers is set to 50 Mbps in all cases and the propagation delay is 1ms. This scenario is tested with 200 and 1000 buffer sizes having preferred buffer size to be set at 150 and 500 packets respectively. Each source starts with 2 seconds difference from its previous one and the first one starting at the beginning of the simulation. Thus, at time 100 seconds all the sources are sending data. At time 150 seconds and for each interval of 1 second a source stops sending data. The simulation time is 200 seconds.

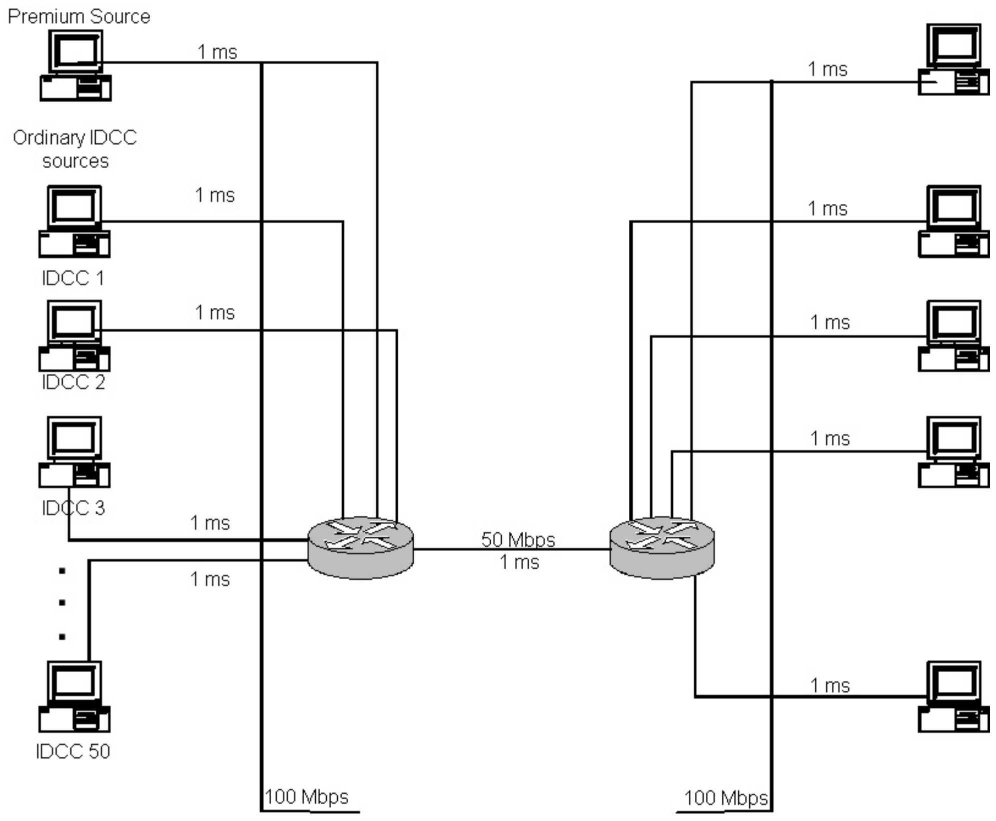


Figure 4.26 Network Topology Scheme for Scenario 4

Each of the following figures contains three graphs. The first and the second refer to the 1000 and 200 buffer sizes respectively and they show the results of the different estimations in the number of ordinary flows. The third represents the comparison of the two cases, when the scheme knows the correct maximum number of the flows.

The throughput (figure 4.27) is increasing, as the estimated number of the flows is smaller. We can see about 0,4% increase for each case, except the difference between the underestimation and correct estimation when the buffer is 200, which has about 8% increase. In the case of the 1000 buffer size IDCC performs better achieving better utilization of the link, because of the fact that the ordinary queue size is set to a higher value (scenario 1). The losses (figure 4.28) are again not accepted in the case of the 200 buffer sizes, which are multiples of the 1000 buffer size. In the case of 1000 buffer size we can observe that the losses of the underestimation in the number of flows is very smaller than the correct and 50% overestimation. This is reasonable because the sources are gradually starting and the underestimation in the number of flows can adapt the rate more easily. Around the 40th second (the 20th source has already started sending traffic) of the simulation the rate of the underestimation scenario is dropped to the half. At that time the number of flows traversing the link is very close to the one that the estimation knows. The rest of the estimations achieve the correct rate (and also stop losing packets) at the time the number of flows estimated is close to the real number of flows traversing the link. As explained the underestimation case finds a lot faster the rate needed (figure 4.29) to regulate the buffer size in the target size of 500 packets (figure 4.30).

Once again, IDCC proved incapable to perform well under small buffer sizes since it wasn’t possible to regulate its rate and get the target ordinary buffer size. As explained in the previous scenario the very large amount of packet losses do not permit the scheme to receive in time the correct feedback and thus the scheme can not be used with small buffer sizes.

Better performance should be achieved when not only the maximum number of flows can be estimated, but also the number of flows at any time during the simulation.

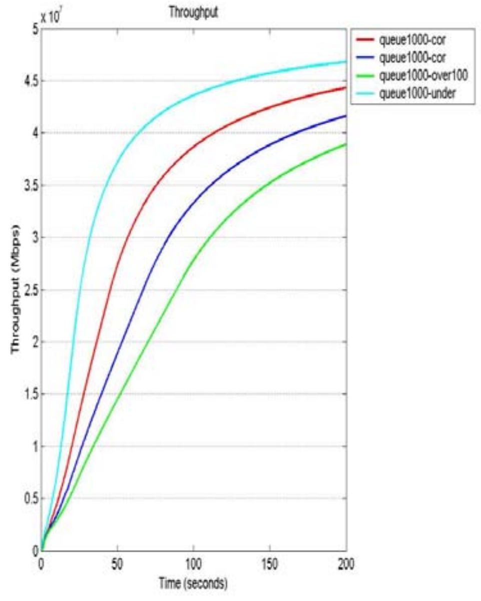
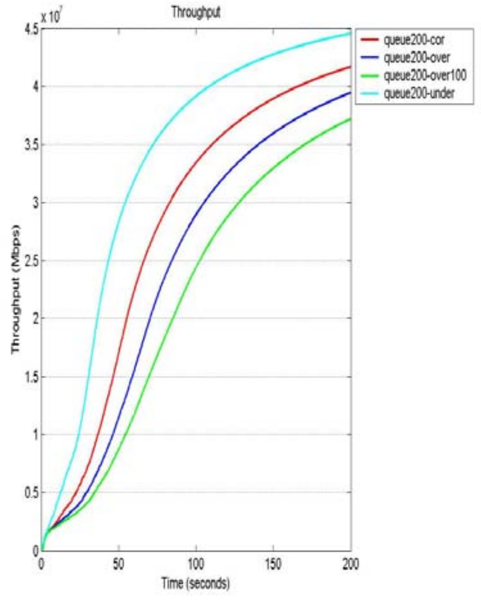
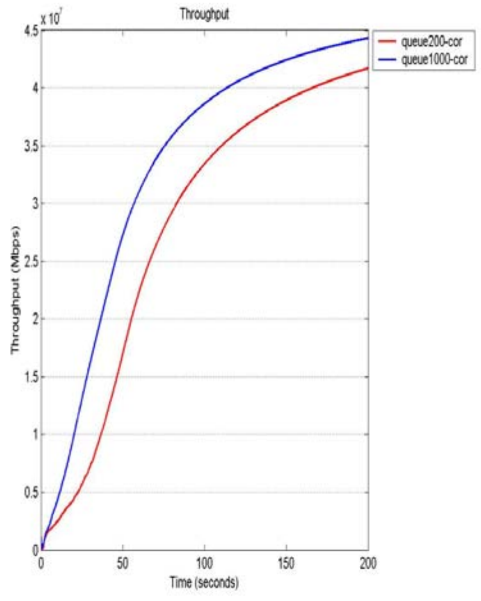
 Throughput buffer size 1000	 Throughput buffer size 200
 Combination of the correct estimations	

Figure 4.27 Throughput

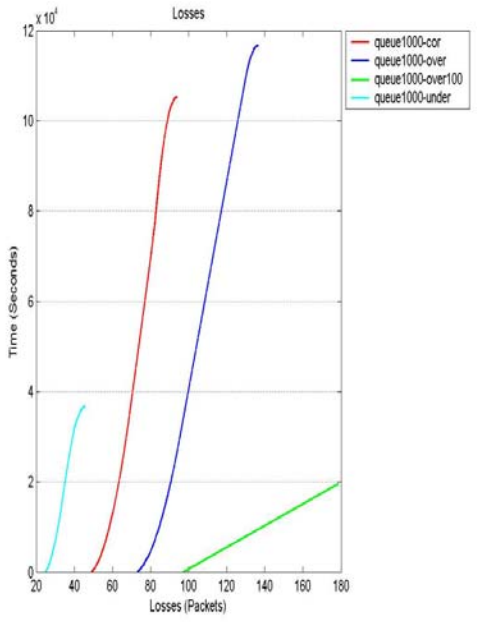
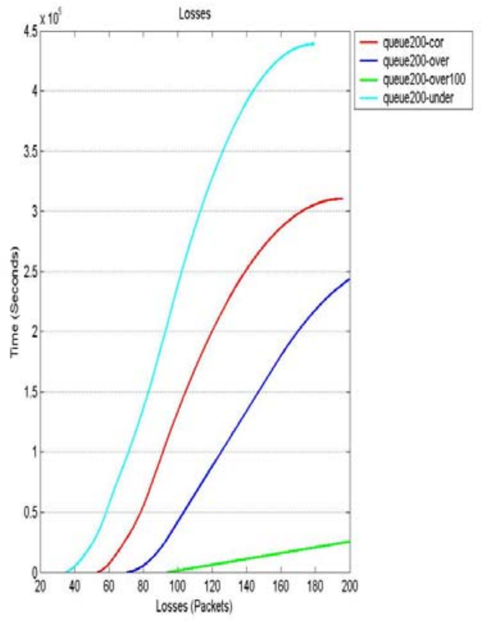
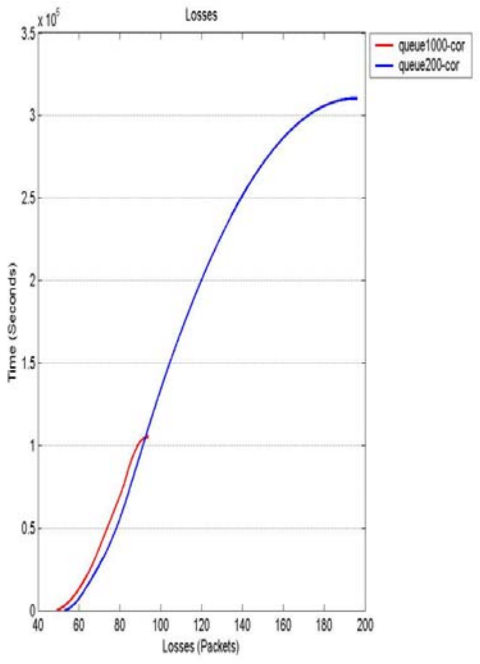
 Losses buffer size 1000	 Losses buffer size 200
 Combination of the correct estimations	

Figure 4.28 Losses

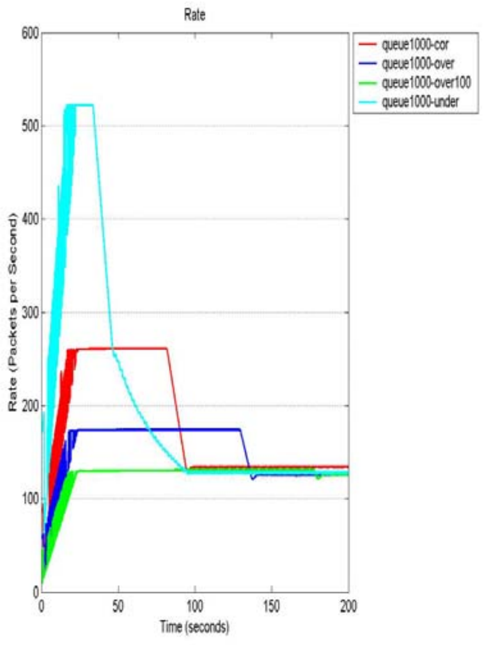
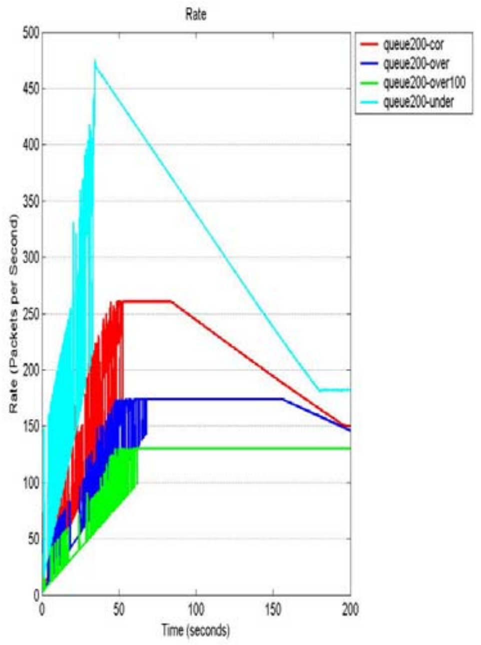
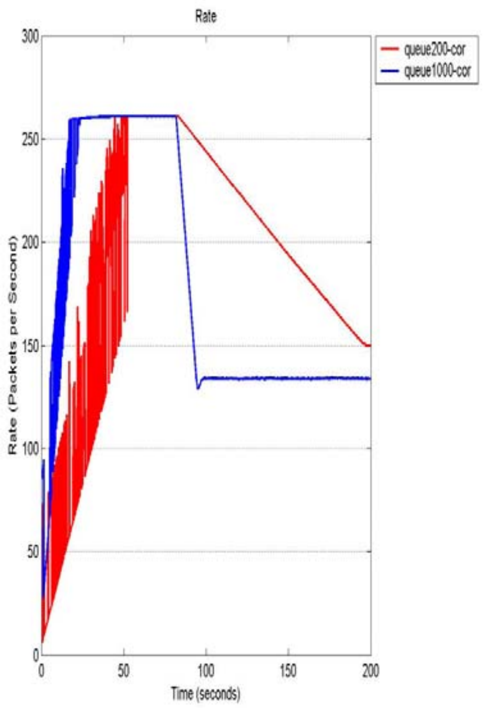
 Rate buffer size 1000	 Rate buffer size 200
 Combination of the correct estimations	

Figure 4.29 Rate

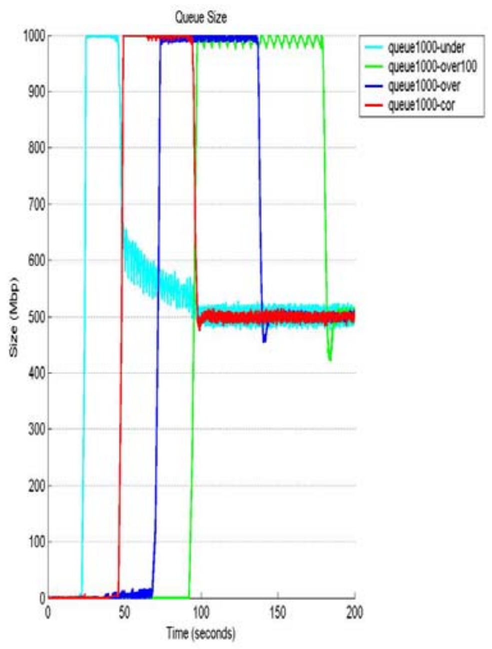
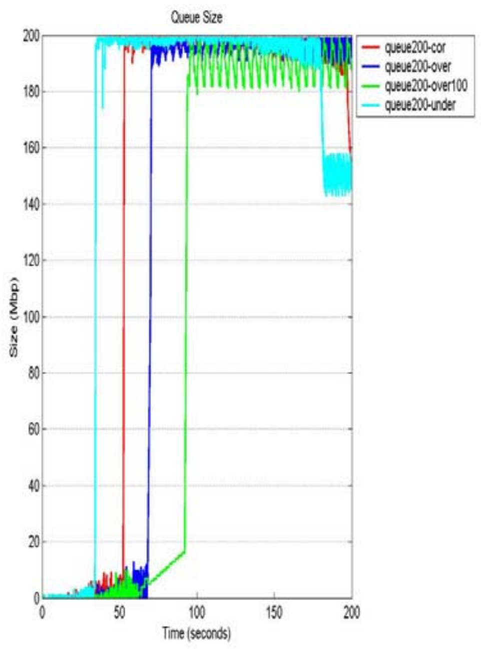
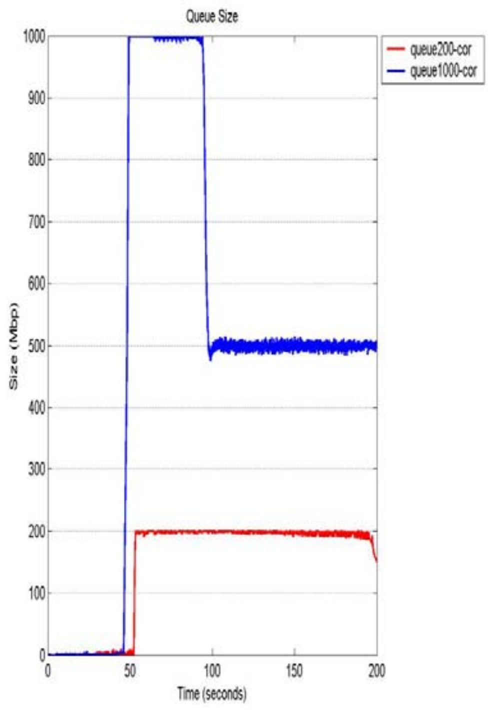
 Ordinary Queue Size buffer size 1000	 Ordinary Queue Size buffer size 200
 Combination of the correct estimations	

Figure 4.30 Ordinary Queue Size

4.2.3 Scenario 3

Scenario 3 is more complicated than the previous ones and it is mainly used in order to study the performance of the scheme under different end-to-end link delays as well as the performance of each individual source when the distance between the sources and the first common link varies are under investigation. Also the QoS guarantees offered to the premium traffic and the fairness of the scheme are also examined. Finally the effect of the value of parameter alphar is examined.

The topology used for this scenario is presented if figure 4.31. For the purposes of the simulation, we have one premium source and nine ordinary sources sending packets to ten destinations. The ordinary traffic service is represented again by saturated IDCC sources. Between the sources and the destinations exist four routers. The routers in the middle constitute the congested link. Its bandwidth is set to 10 Mbps in all cases and the propagation delay is varies from 5 to 50ms. Again tests were made for 200 and 1000 packet buffer sizes. In order to examine the QoS guarantees offered to the premium traffic service end-to-end delay and jitter measurements have been taken for different network parameters.

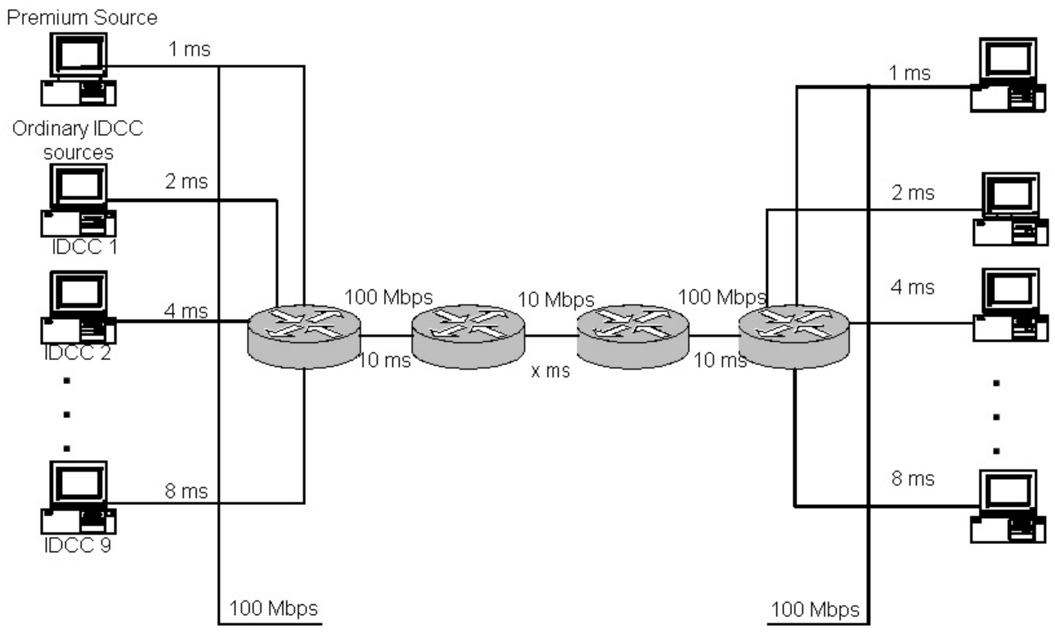


Figure 4.31 Network Topology Scheme for Scenario 3

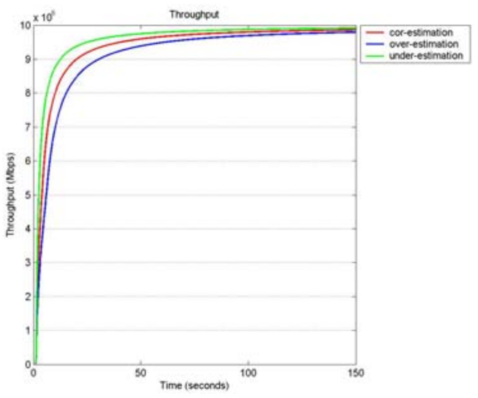
The simulations have been run for various delay times for the congested link (between 1 and 50ms) for both buffer sizes. At the same time, we also examine the behaviour of the scheme, having wrong estimations of the flows traversing the link.

The total throughput for buffer sizes 200 and 1000 is shown in figures 4.32 and 4.33 respectively. We can observe that the utilization of the link in the case of the 200 packet buffer is very high for all the estimations. In the case of the 1000 packet buffer the underestimation of the number of flows leads to an undesirable state where there is no stabilization. The higher the delay is, the higher the problem. This is the first time we witness a good performance of the scheme under small buffer sizes. Most probably this is due to the smaller bandwidth size of the congested link. Fewer packets are needed in order to achieve high utilization, so the rate that the ordinary sources are sending traffic is lower and that leads to smaller number of packet losses and quick stabilization of the ordinary rate.

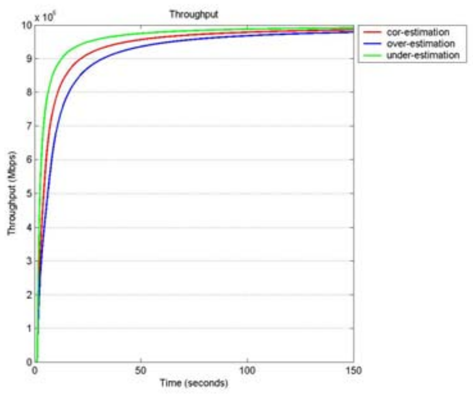
Although the performance of the small buffer is much better than the previous scenarios, the drops (figure 4.34) are more compared to the 1000 packet buffer (figure 4.35), ranging from 4000 to 5800 according to the delay (table 4.2). The higher the delay, the higher the drops, probably because of the time needed for the feedback to reach the sources. Also the smaller the estimation of the flows is, the higher the number of drops. In the case of the 1000 packet queue and the correct estimation (figure 4.35), the number of drops is very low, ranging between 600 and 760. In this case the losses are independent from the delay (the test with the 10ms link had the most losses). The case of the underestimation of the flows is disappointing. There are losses in all the simulation time and as the figures 4.36 and 4.37 show it cannot converge to a stable rate in order to achieve the target buffer length (which is set to 500 packets). The correct and the over estimation converge at the same rate, with the overestimation having better performance (showing less oscillations). The higher the delay is, more time is needed for the correct estimation to attain the correct rate and queue size. The results for the 200 buffer (figures 4.38 and 4.39) are as expected, with the rate to stabilize in all cases in less than 20 seconds of simulation time. All the estimations lead to the same value of rate for the same reasons explained in scenario 1.

As it would be expected, the next step was to test the behaviour of IDCC when some sources were stopped. The case of the 30ms delay link was selected, and we demonstrate two scenarios one for each buffer size. In each case we have two stops, happening at the 45th and 90th second of the simulation respectively. Except the case of the underestimate of flows when the buffer size is 1000 packets (which was already problematic), the rest of the cases have a very quick respond to the stops of the two sources. There is no indication from the throughput graphs (figures 4.40 and 4.43) that some sources were stopped. As we can see IDCC take actions very quickly (in a few seconds) and regulate the rates (figures 4.42 and 4.45) in order to attain the proper queue sizes (figures 4.41 and 4.44). The results comply with the ones taken from the first scenario. IDCC has very quick response to the individual sources stops, although it operates with the wrong instantaneous number of flows (as mentioned earlier only the maximum number of flows is known to the scheme).

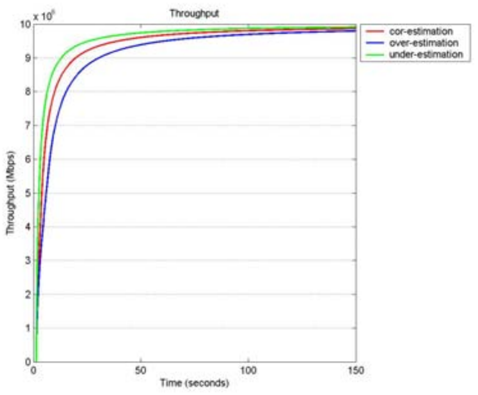
The next test was to study the QoS offered by IDCC to the premium service (figure 4.46). Firstly we made four measurements for the end-to-end delay. The selected cases were the 10 and 30ms delays with combination of the 200 and 1000 packet buffer sizes. Once again the end-to-end delay was very high, so we decrease the target queue size to 15 in the cases of the 10ms delay and we observe very big difference from the previous measurements. The decrease was expected due to the way the premium controller works. In order to keep the buffer size in lower levels, the packets suffer less queuing delays. Table 4.3 shows the exact values of those measurements. The end-to-end delay is affected by the link delay (propagation delay) and the preferred buffer size. The smaller that size is, the smaller the end-to-end delay. Once again the end-to-end delay values are not satisfactory. In order to have accepted values of the end-to-end delay we need to have very small target queue sizes (which leads to worse performance) and a big number of sources sending traffic through the congested link.



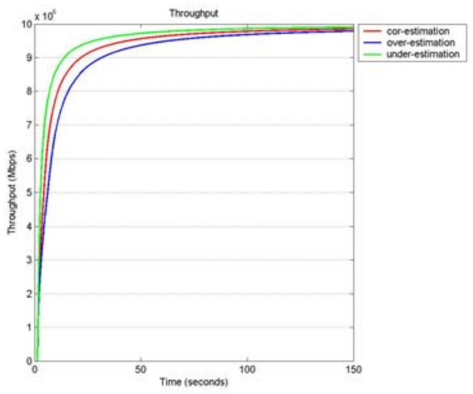
5 ms delay



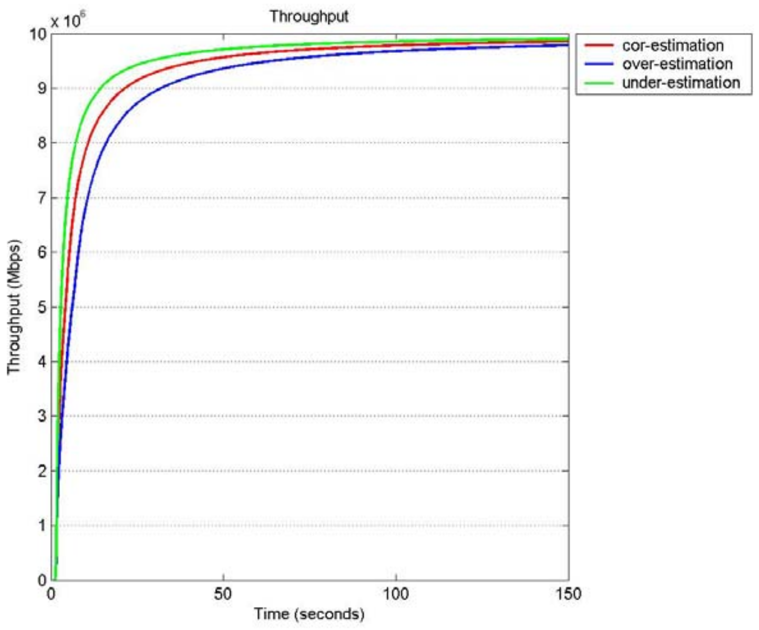
10 ms delay



15 ms delay

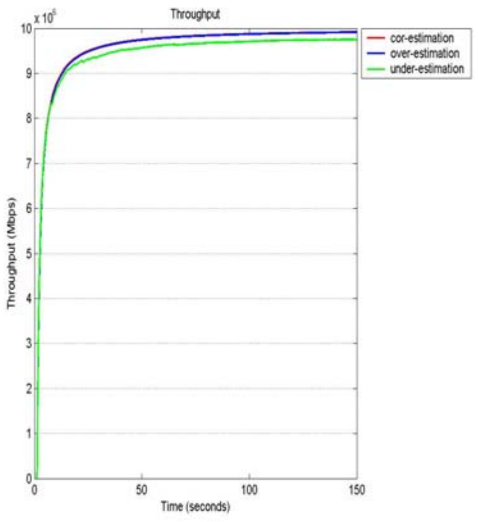


30 ms delay

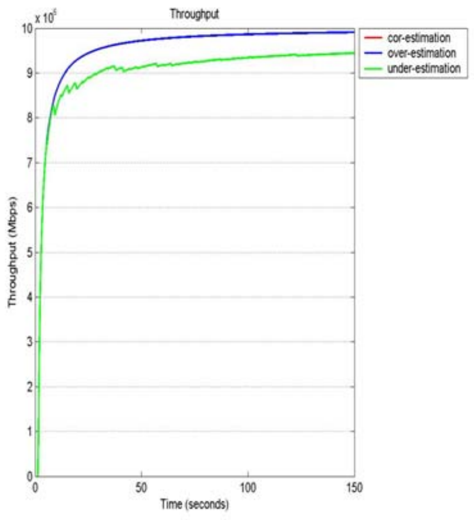


50 ms delay

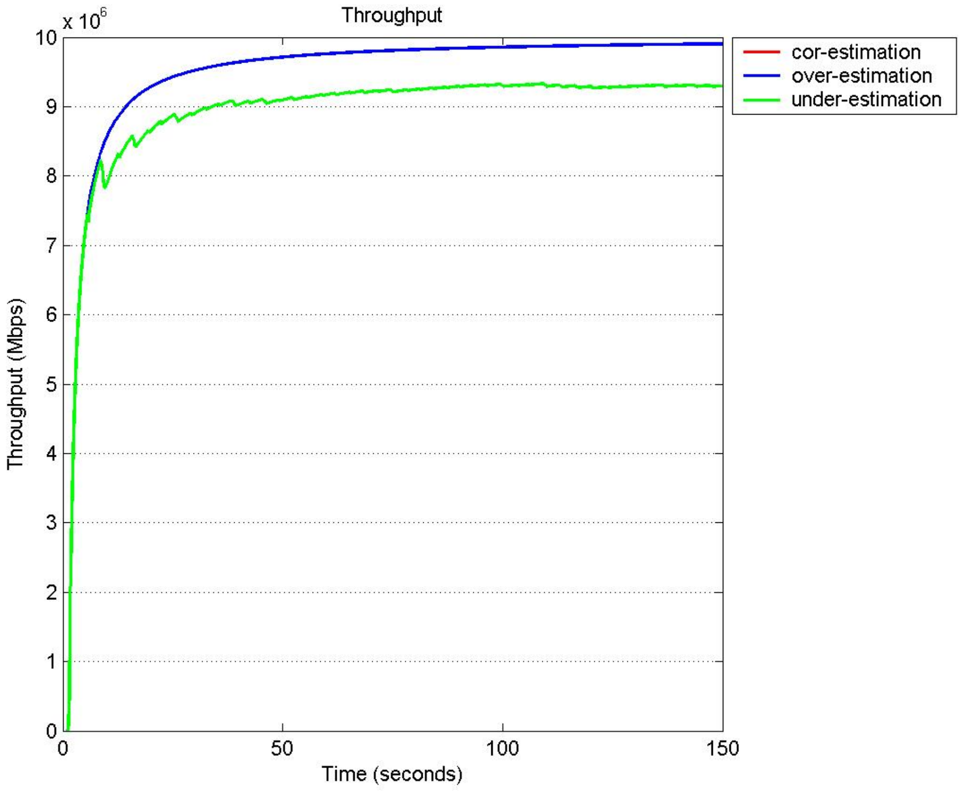
Figure 4.32 Throughput for various delays and estimations. Buffer size 200



5ms delay

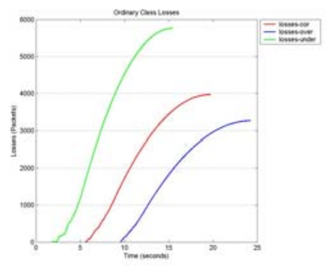


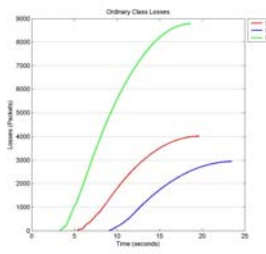
30ms delay

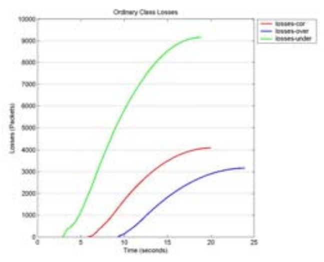


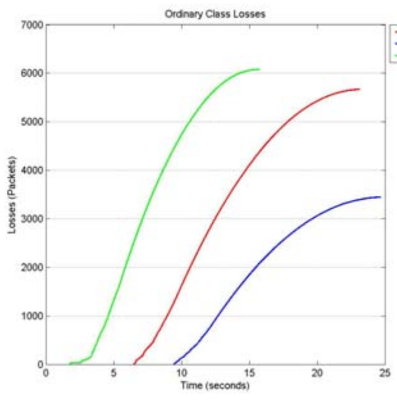
50 ms delay

Figure 4.33 Throughput for various delays and estimations. Buffer size 1000

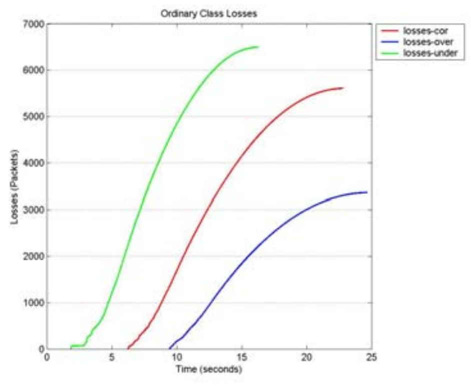

5ms delay


10 ms delay

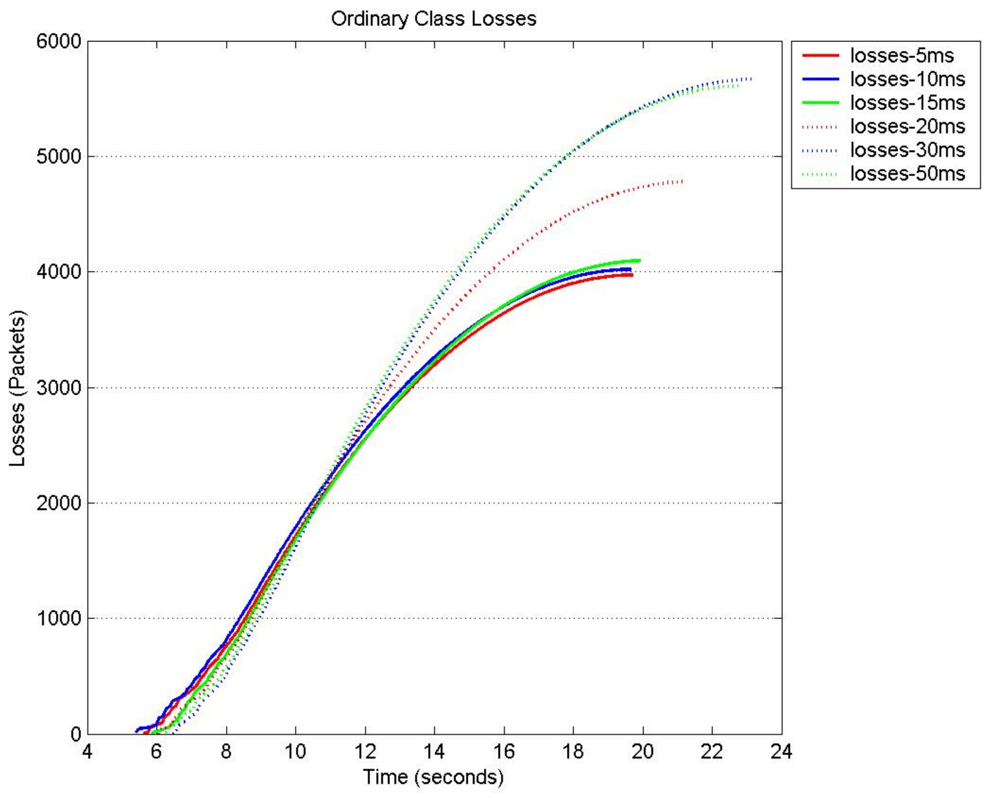

15 ms delay



30 ms delay

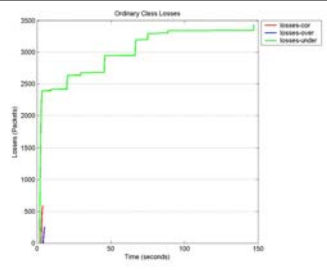


50 ms delay

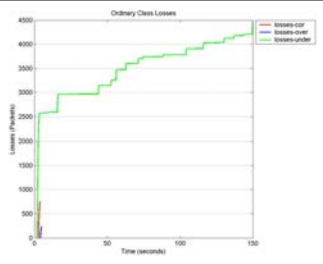


combination of the delays for the correct estimations

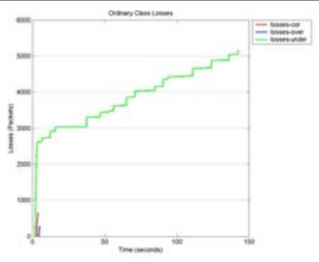
Figure 4.34 Losses for various delays and estimations. Buffer size 200



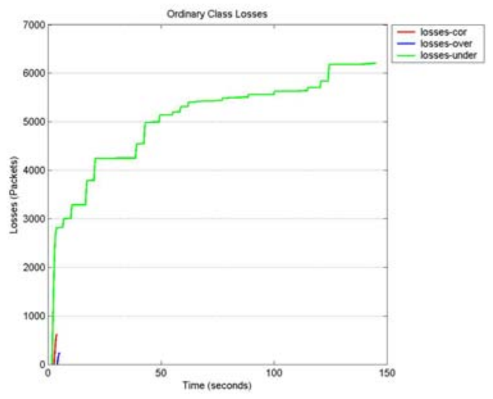
5 ms delay



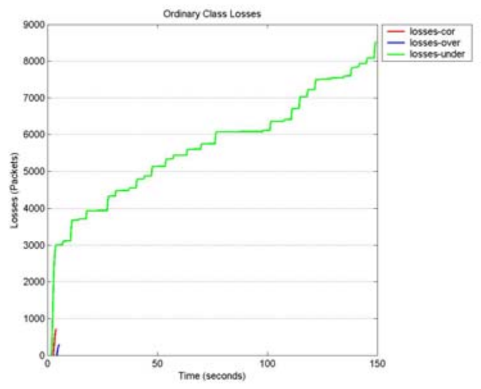
10 ms delay



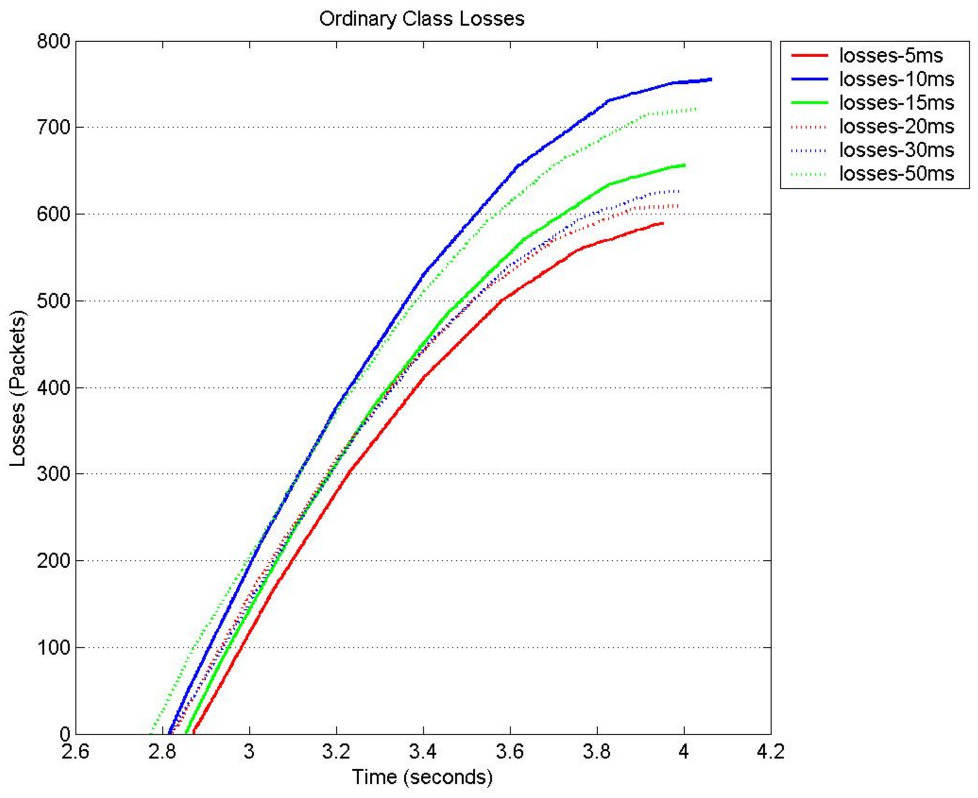
15 ms delay



30 ms delay



50 ms delay

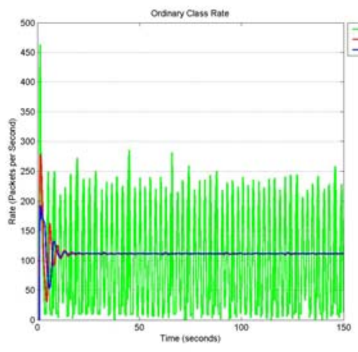


combination of the delays for the correct estimations

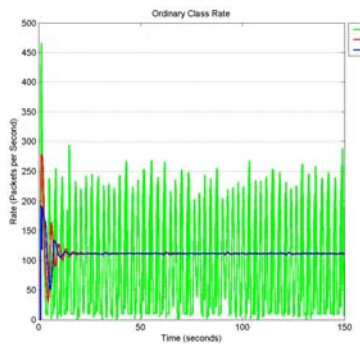
Figure 4.35 Losses for various delays and estimations. Buffer size 1000

		5 ms	10 ms	15 ms	30ms	50ms
buffer size = 1000	over estimation	254	239	281	235	289
	under estimation	3435	4466	5169	6217	8525
	correct estimation	589	755	656	626	721
buffer size = 200	over estimation	3266	2944	3166	3445	3362
	under estimation	5766	8782	9161	6078	6498
	correct estimation	3972	4021	4096	5668	5609

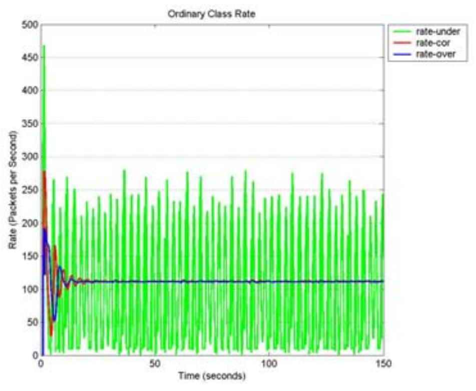
Table 4.2 Losses for buffer sizes 200 and 1000 under different estimations



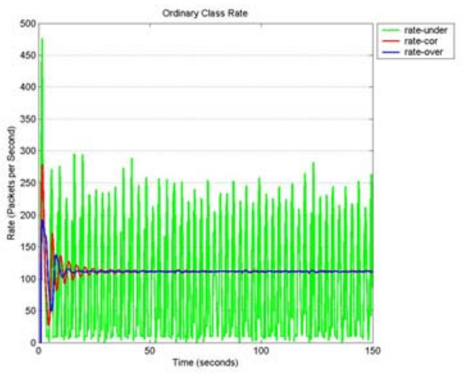
5ms delay



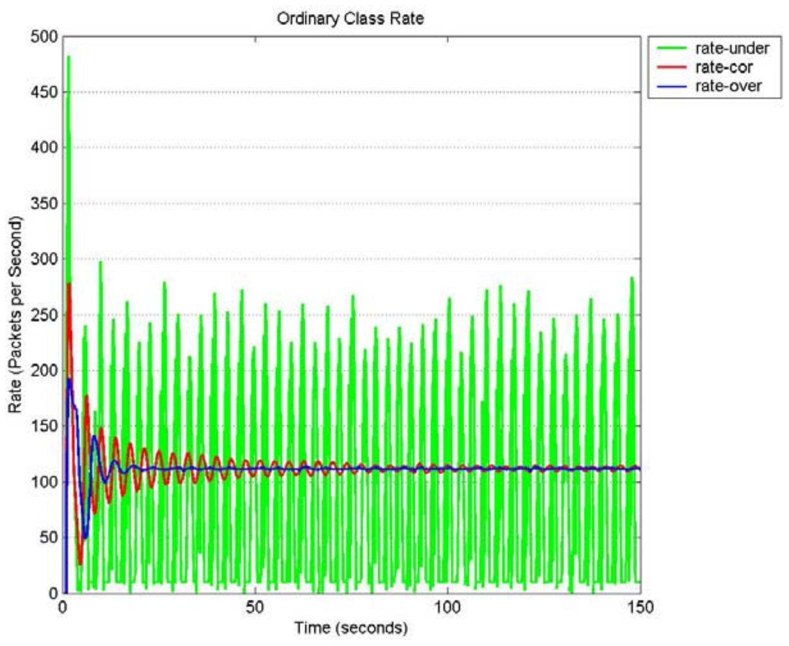
10ms delay



15ms delay

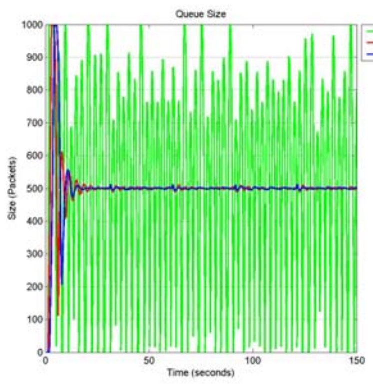


30ms delay

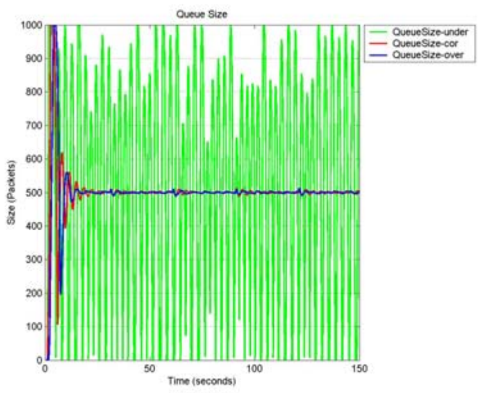


50ms delay

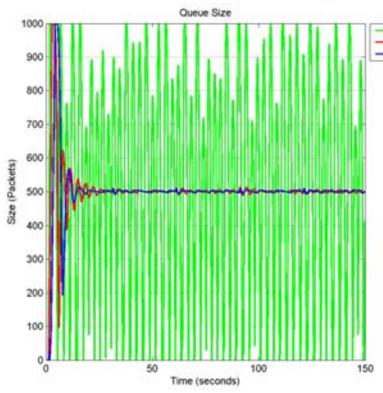
Figure 4.36 Rate. Buffer size 1000



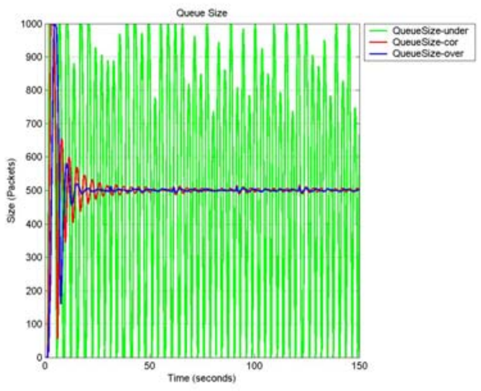
5ms delay



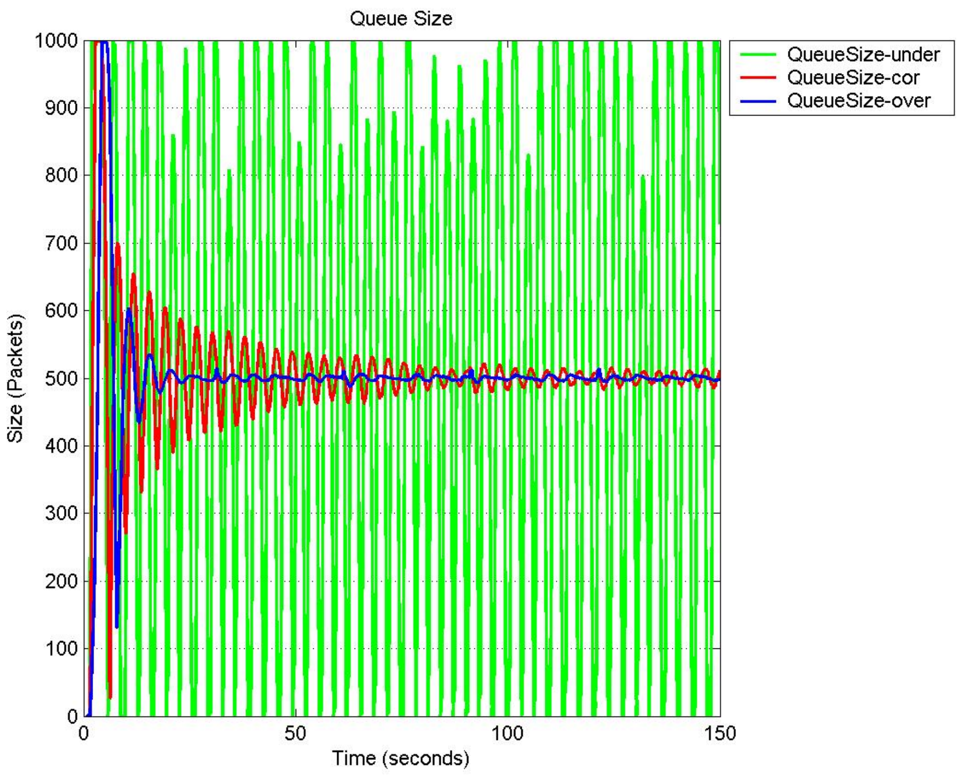
10ms delay



15ms delay

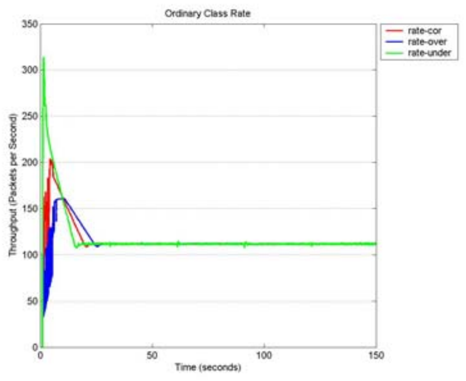


30ms delay

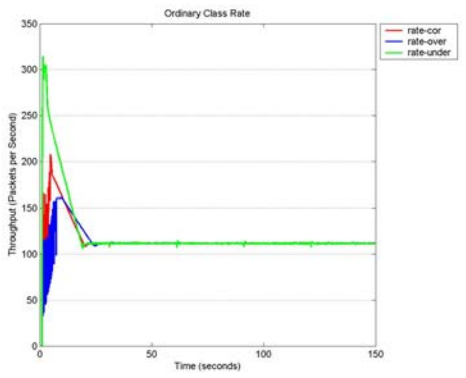


50ms delay

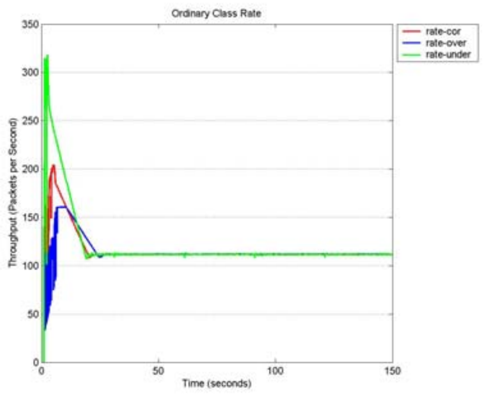
Figure 4.37 Ordinary Queue Size. Buffer size 1000



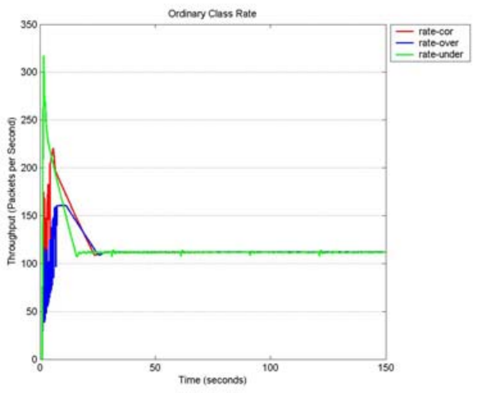
5ms delay



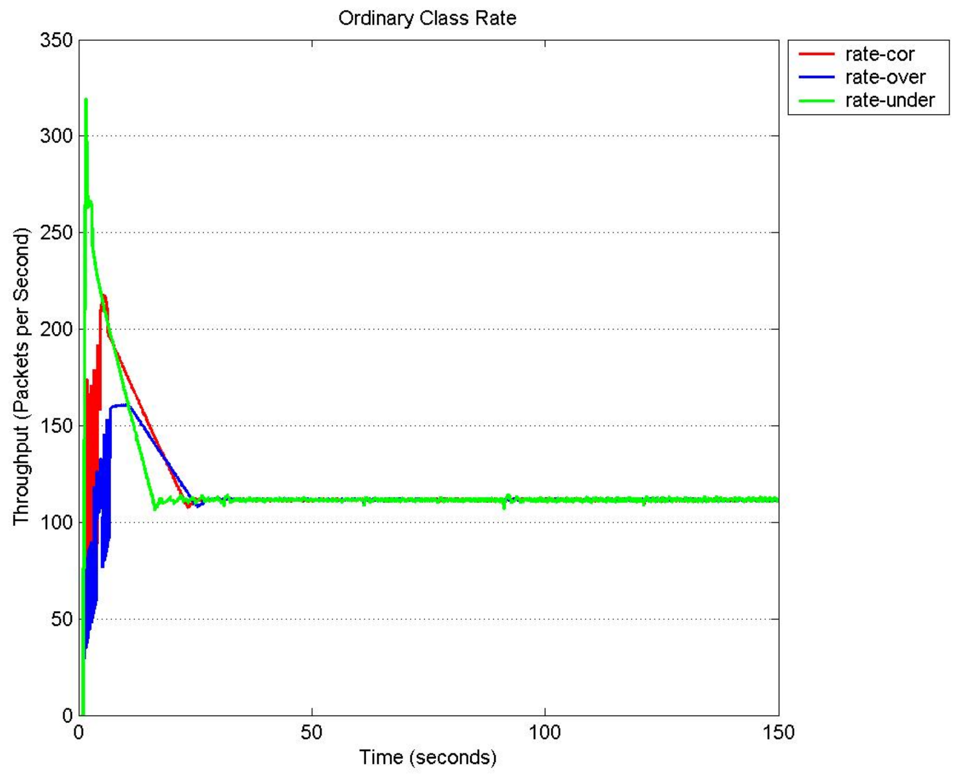
10ms delay



15ms delay

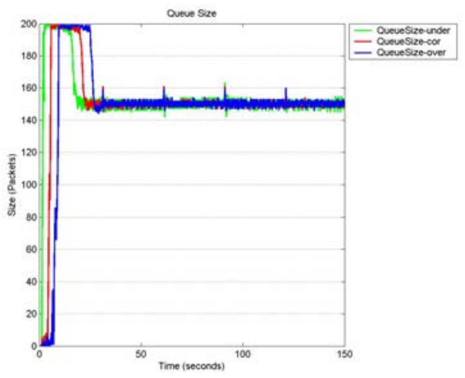


30ms delay

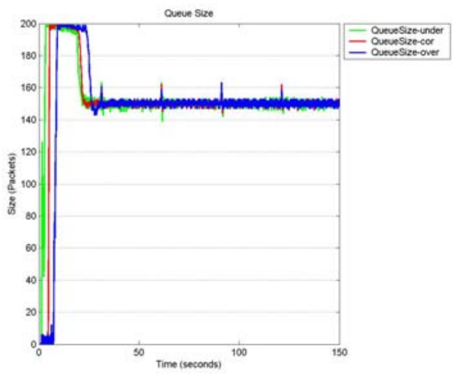


50ms delay

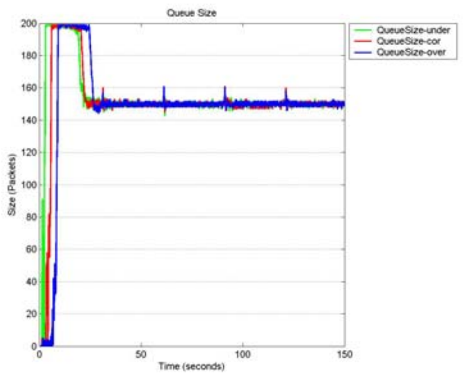
Figure 4.38 Rate. Buffer size 200



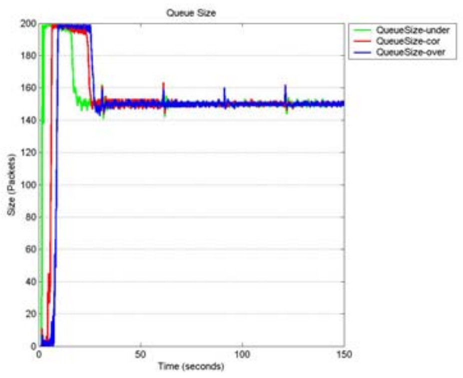
5ms delay



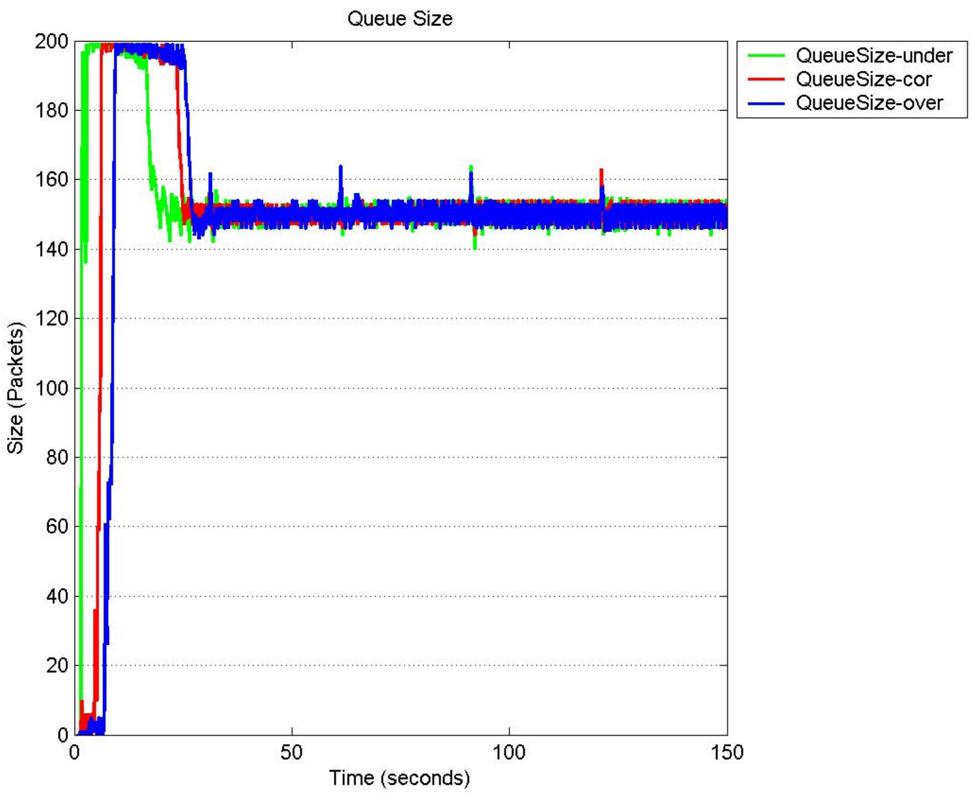
10ms delay



15ms delay



30ms delay



50ms delay

Figure 4.39 Ordinary Queue Size. Buffer size 200

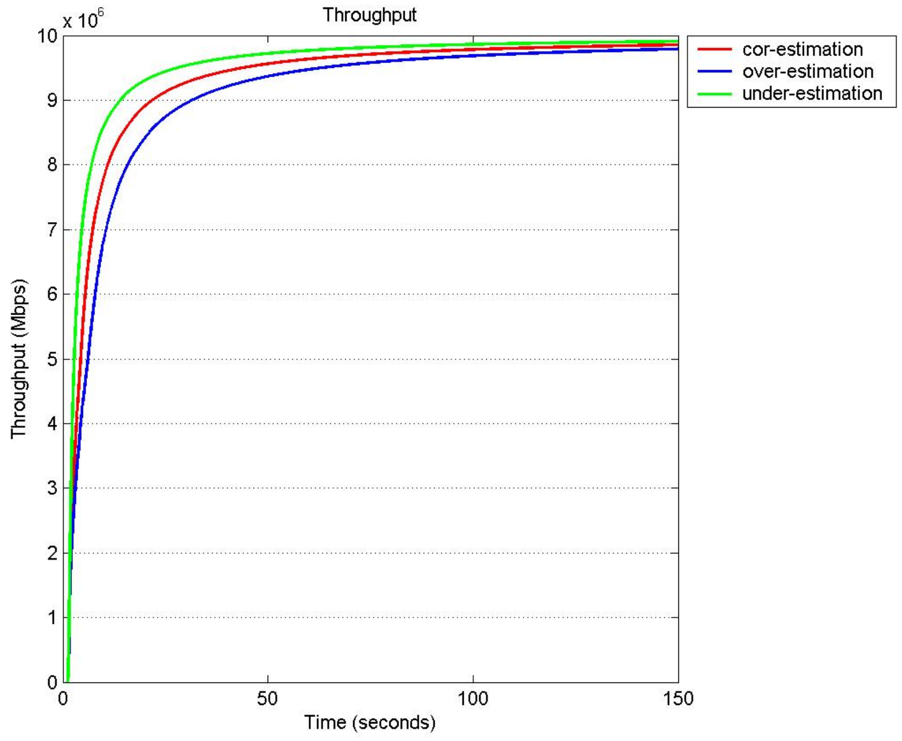


Figure 4.40 Throughput. Buffer size 200

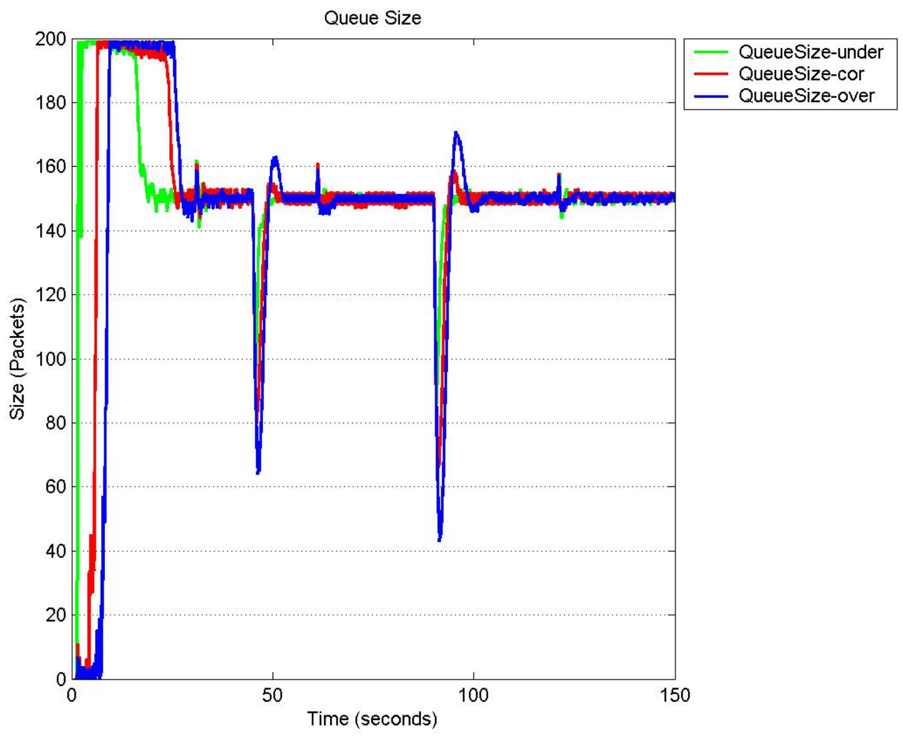


Figure 4.41 Ordinary Queue Size. Buffer size 200

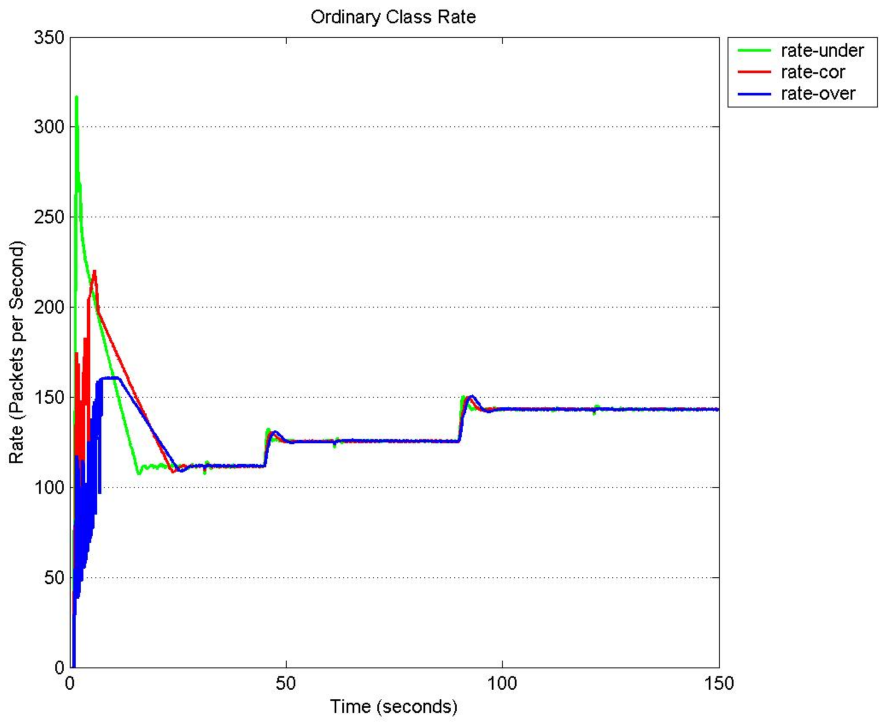


Figure 4.42 Rate. Buffer size 200

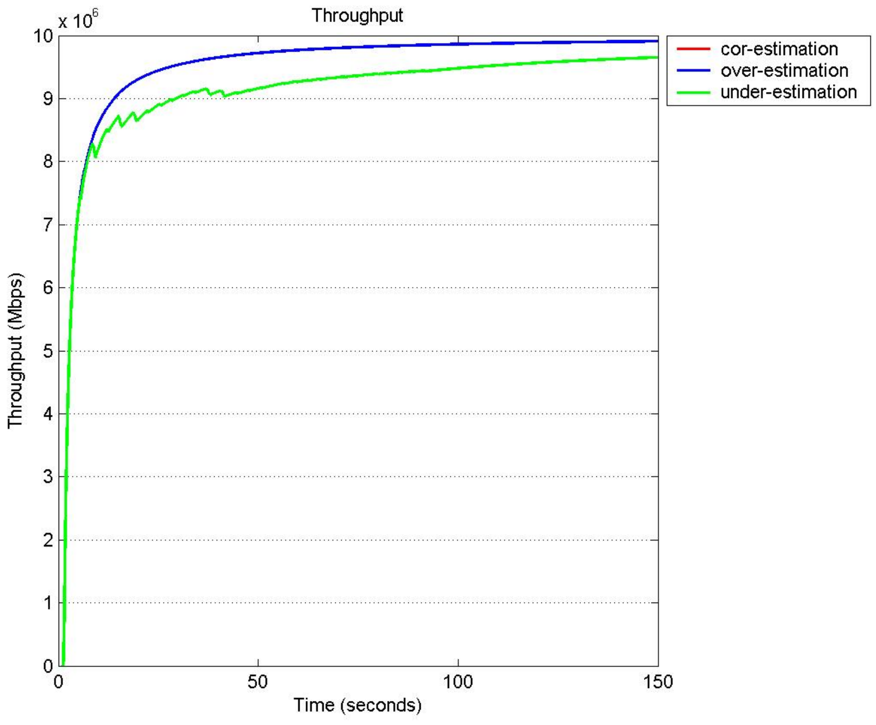


Figure 4.43 Throughput. Buffer size 1000

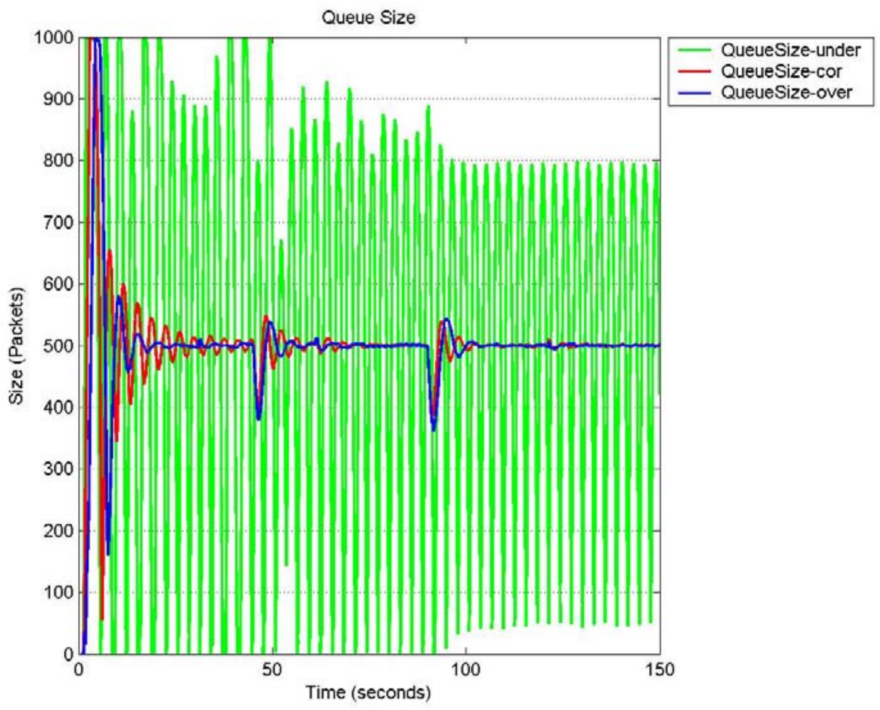


Figure 4.44 Ordinary Queue Size. Buffer size 1000

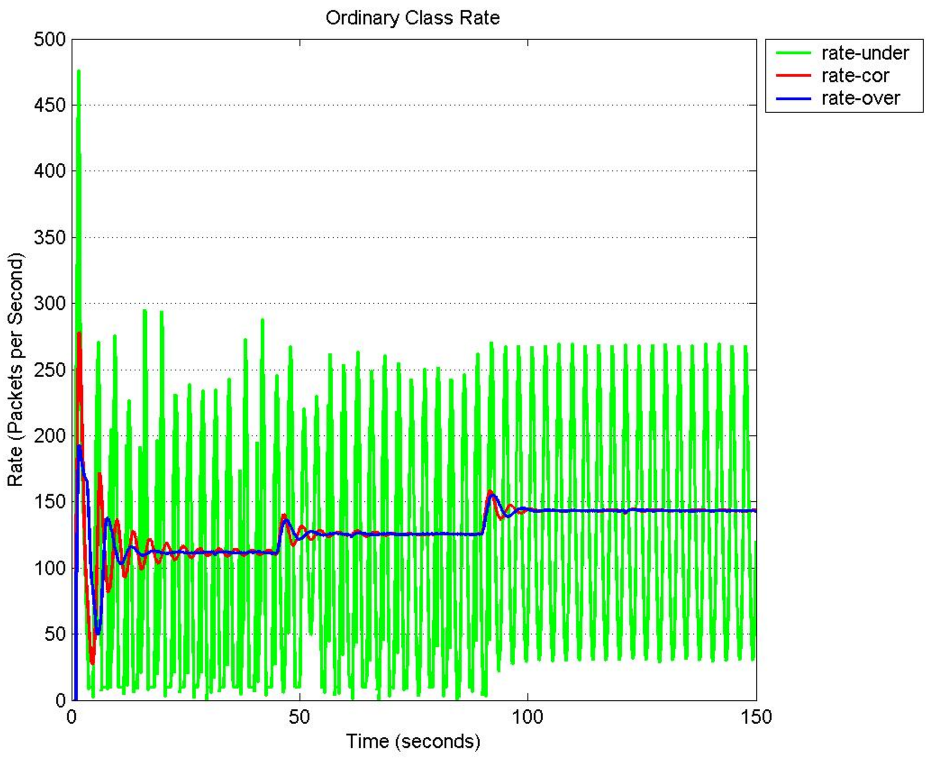


Figure 4.45 Rate. Buffer size 1000

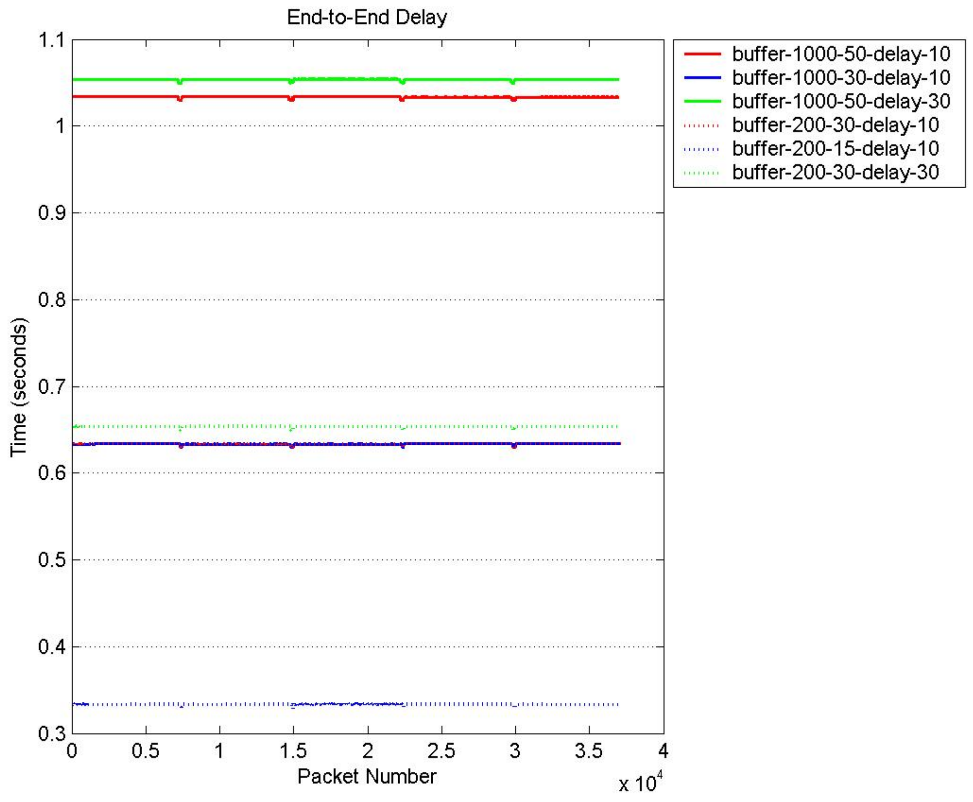


Figure 4.46 End-to-End delay for the Premium Traffic Service

	delay	preferred buffer size	mean delay	jitter
buffer size = 1000	10ms	50 packets	1033.48 ms	0.6784 ms
	10ms	30 packets	633.509 ms	0.5607 ms
	30ms	50 packets	1053.48 ms	0.6786 ms
buffer size = 200	10ms	30 packets	633.466 ms	0.5614 ms
	10ms	15 packets	333.424 ms	0.4197 ms
	30ms	30 packets	653.469 ms	0.5495 ms

Table 4.3 End-to-end delay and jitter for different network conditions

The next measurements were made in order to study the fairness of IDCC that is all the sources have the same treatment by the scheme (have the same link utilization, same number of packet losses etc). Figure 4.47 shows the throughput of five individual sources, which are connected with different link delays with the first router. The link delays are relatively small and we notice that the sources at get the same throughput. Even the small difference in the first 20 seconds is rather randomly than based on the delays. The losses of each individual source are presented in figure 4.48. Although there is a difference in the packet losses, there is also a random distribution of the losses. We can observe that the 2ms delay source has higher number of loses whereas the 4ms smaller number of losses. The numbers of losses are very small, so we can state that the scheme is fair.

Although the scheme seems to be fair, it did not have the expected behavior. Someone would expect the closest source to send more packets and probably have more losses. So another case was tested, where the link delays had more difference. The results were the same as before and are presented in figures 4.49 and 4.50. It seems that the sources get the same feedback independently of their link delays (and the more precisely their distance from the first router).

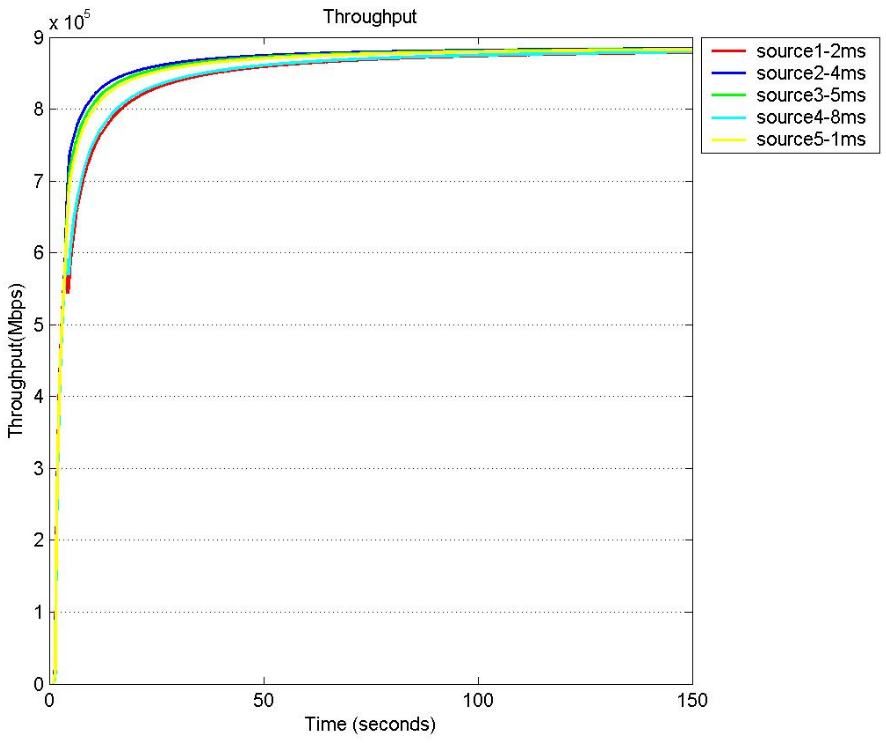


Figure 4.47 Throughput of individual sources with small delays

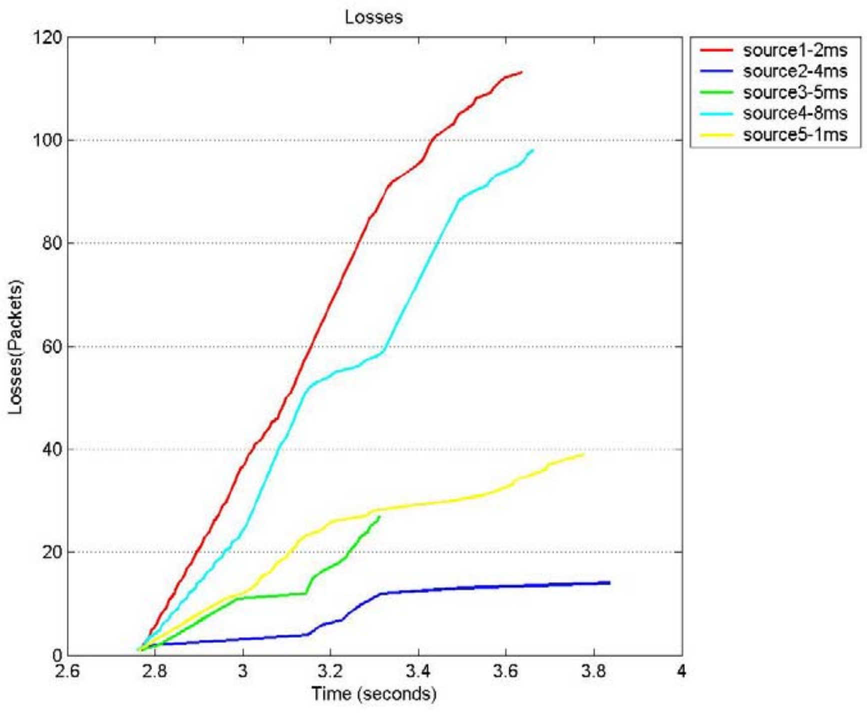


Figure 4.48 Packet losses of individual sources with small delays

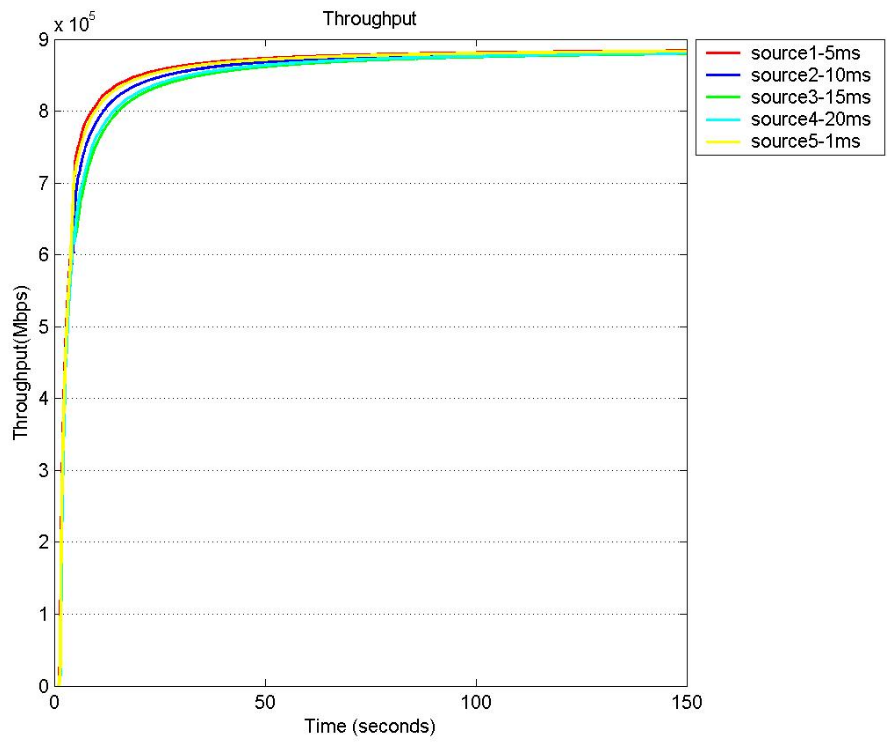


Figure 4.49 Throughput of individual sources with large delays

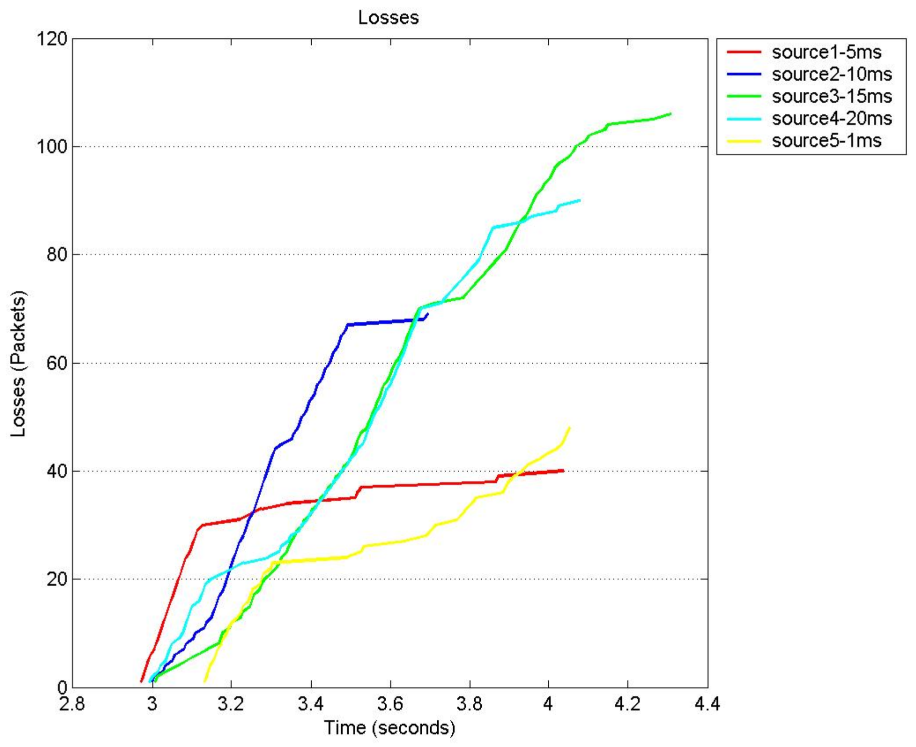


Figure 4.50 Packet losses of individual sources with large delays

The last measurement for this scenario was to test the effect of the value of a_r in the performance of the ordinary service controller (and generally in the whole scheme). Different values for a_r were selected and the throughput achieved by each one is presented in figure 4.51. With a close look at the figure, we can distinguish four different categories of performances. The categories are based on the throughput achieved by the parameter and each one is drawn with different type of line. Immediately the case where a_r is 0.7 is discarded, because both of the low performance and the oscillations it presents

From figure 4.52 we can observe that the lowest the throughput of the category, the more packet losses are detected. Now for each individual category a difference in the drops is presented. The next figures 4.53 – 4.60 show the buffer size and the rate the sources are sending traffic during the simulation time. Using these results we can say that only the values of category 2 could be used for the a_r value in the scheme (category 2 is presented with solid line in figure 4.51 and achieves the highest throughput). From that category it seems that the most effective value is the $a_r = 1.5$ because by using this value the oscillations are minimized and the queue stabilizes and responds to the stops of sources faster.

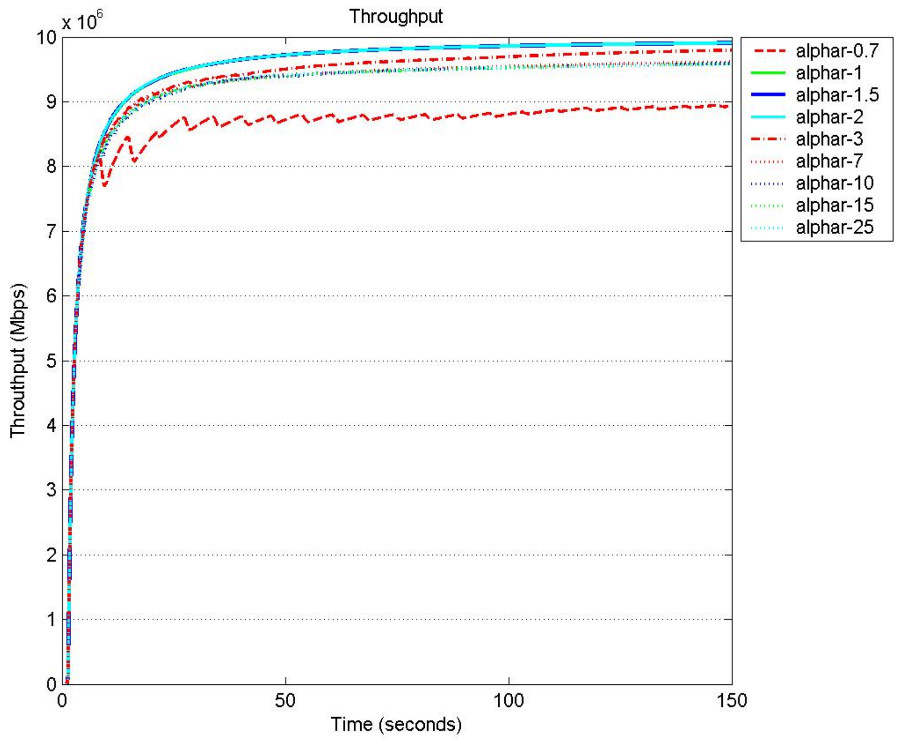


Figure 4.51 Throughput using different a_r

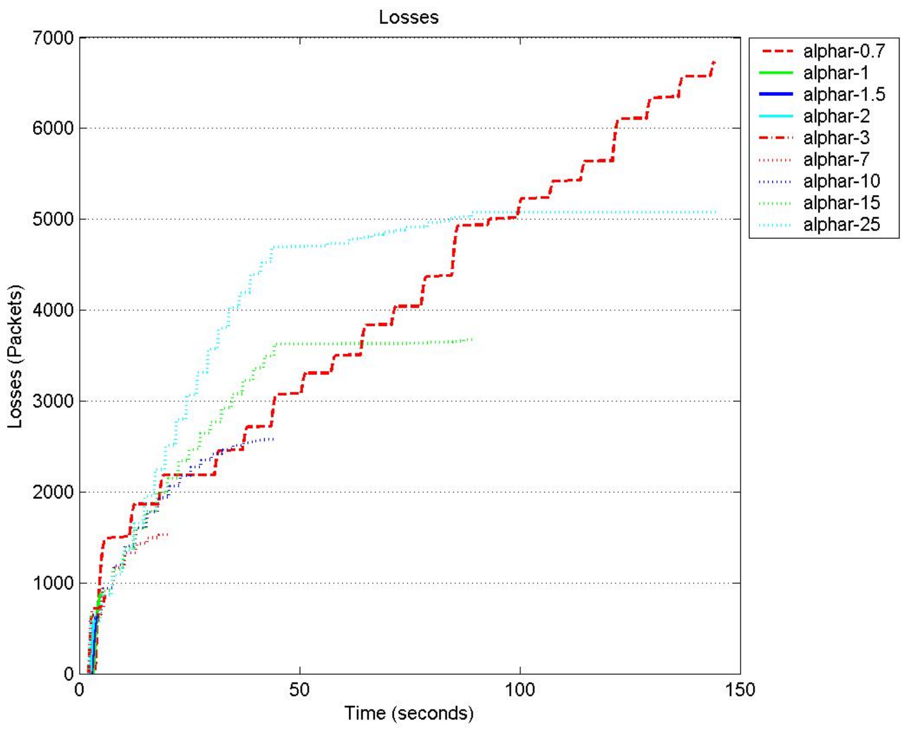


Figure 4.52 Losses using different a_r

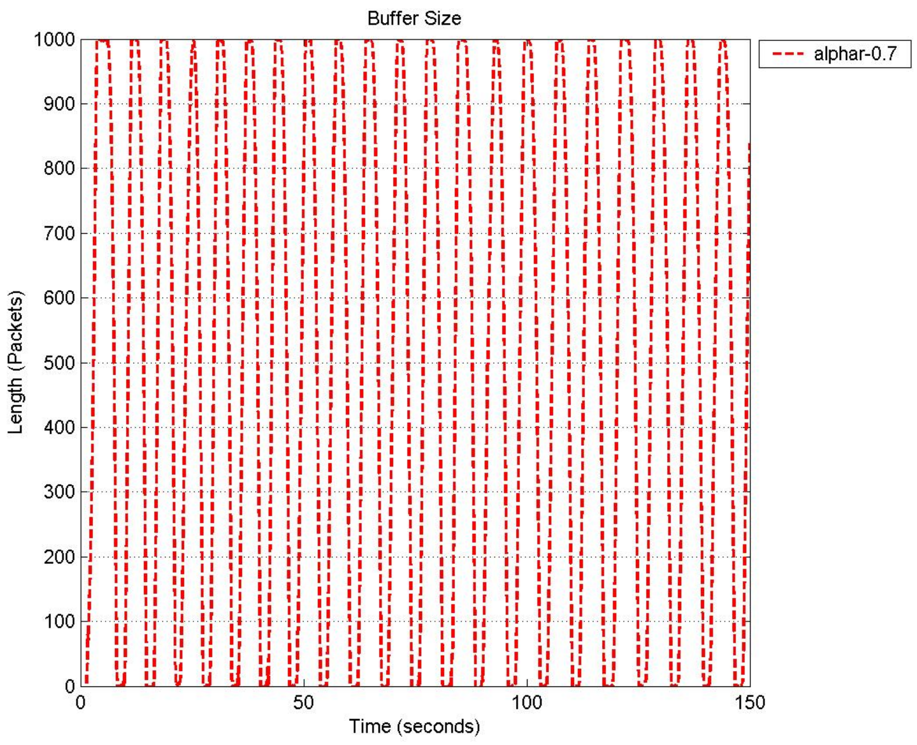


Figure 4.53 Buffer Size for category 1

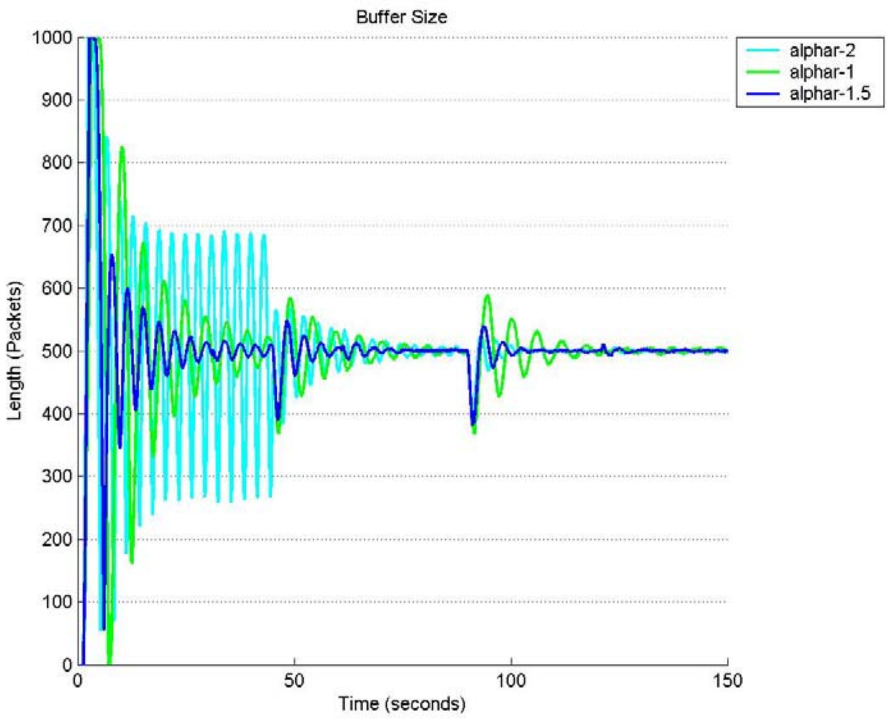


Figure 4.54 Buffer Size for category 2

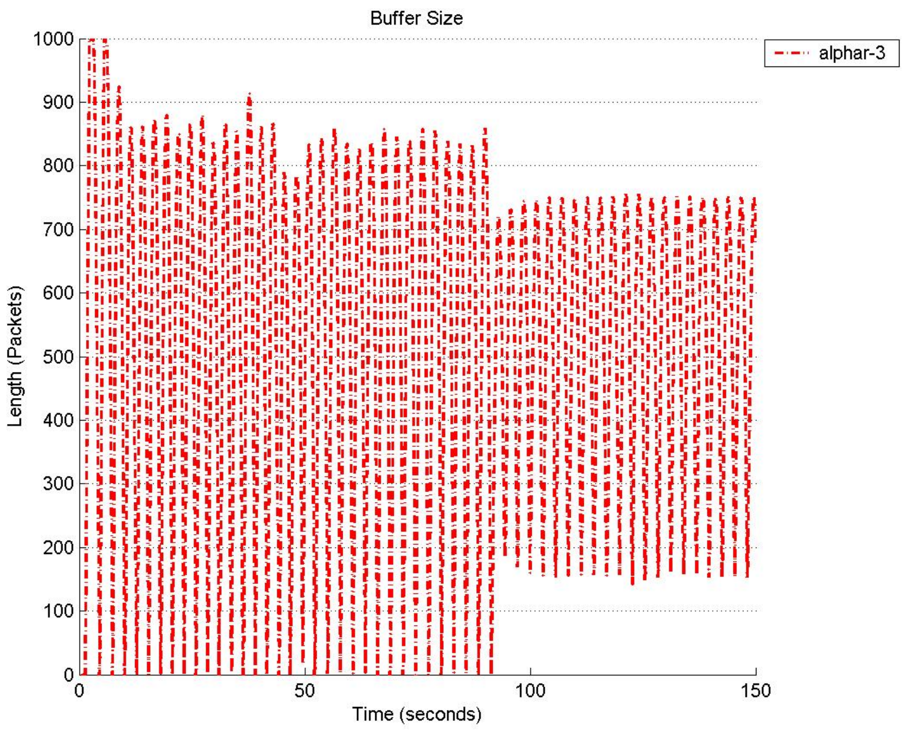


Figure 4.55 Buffer Size for category 3

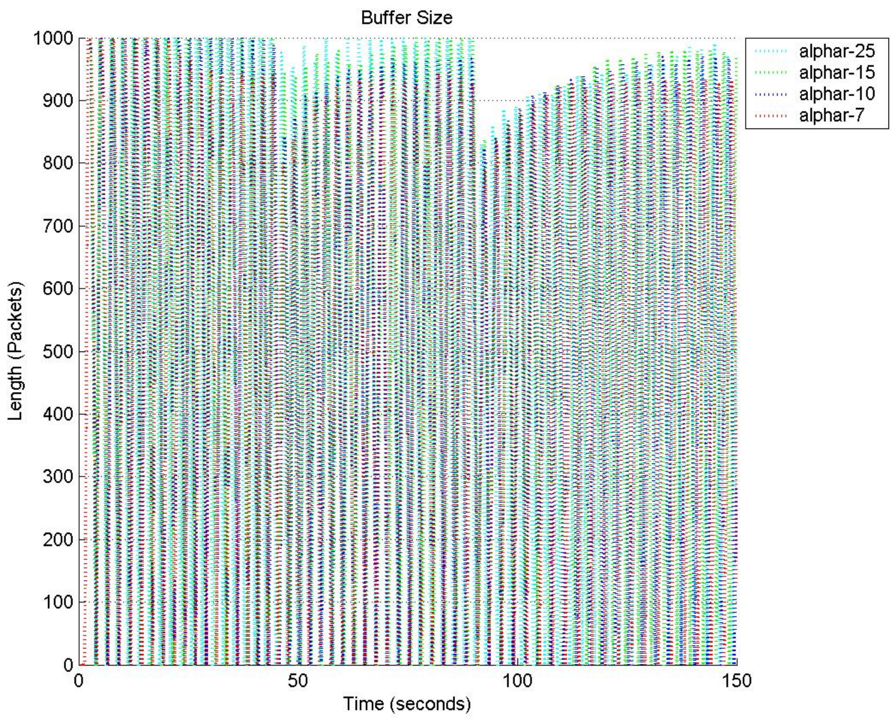


Figure 4.56 Buffer Size for category 4

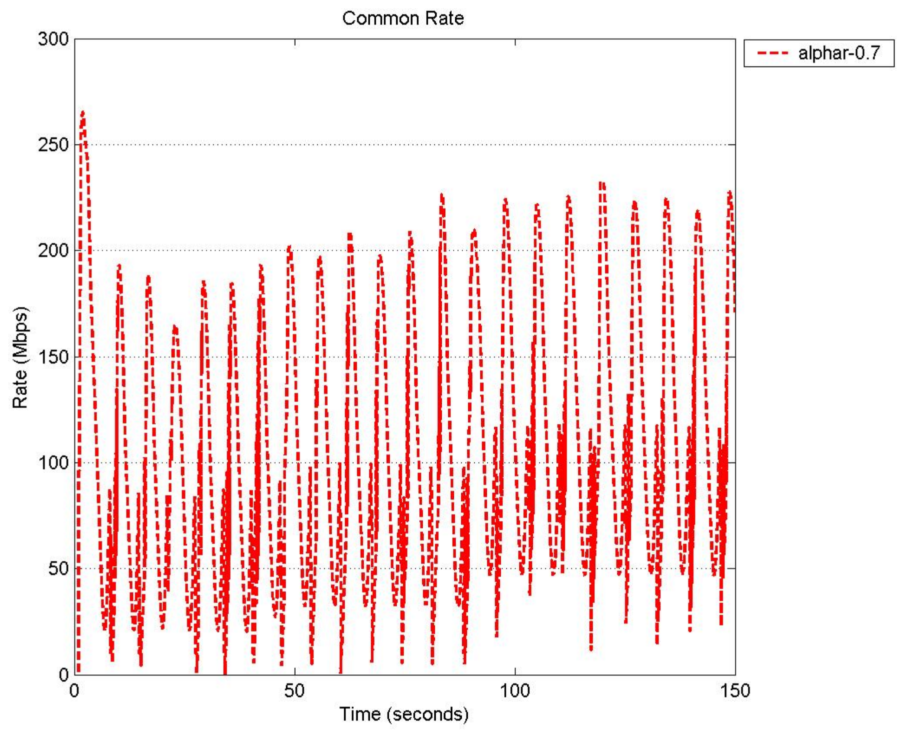


Figure 4.57 Rate for category 1

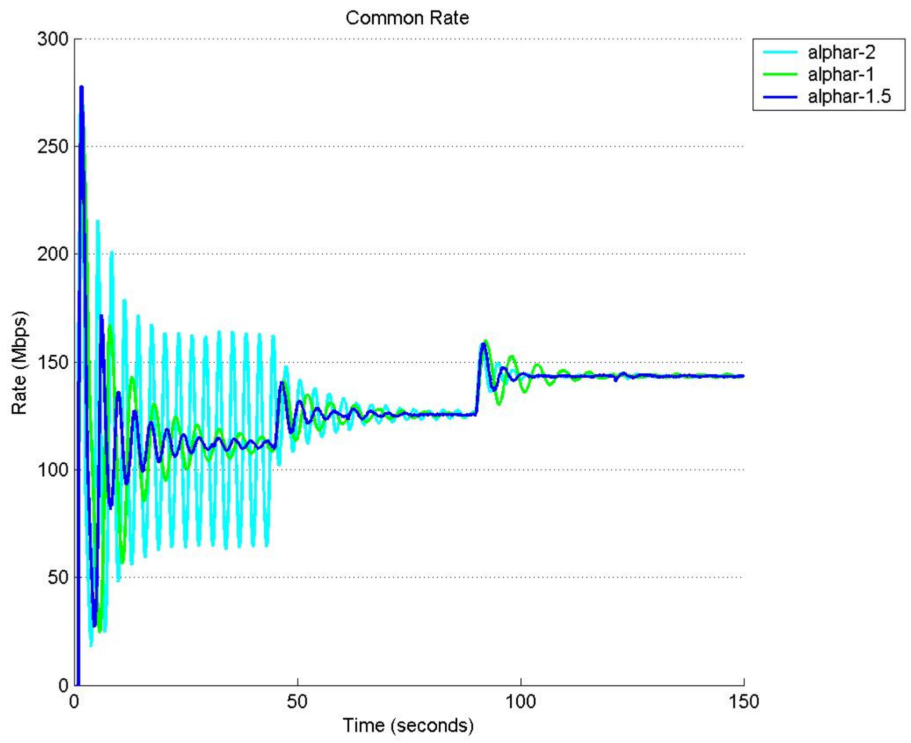


Figure 4.58 Rate for category 2

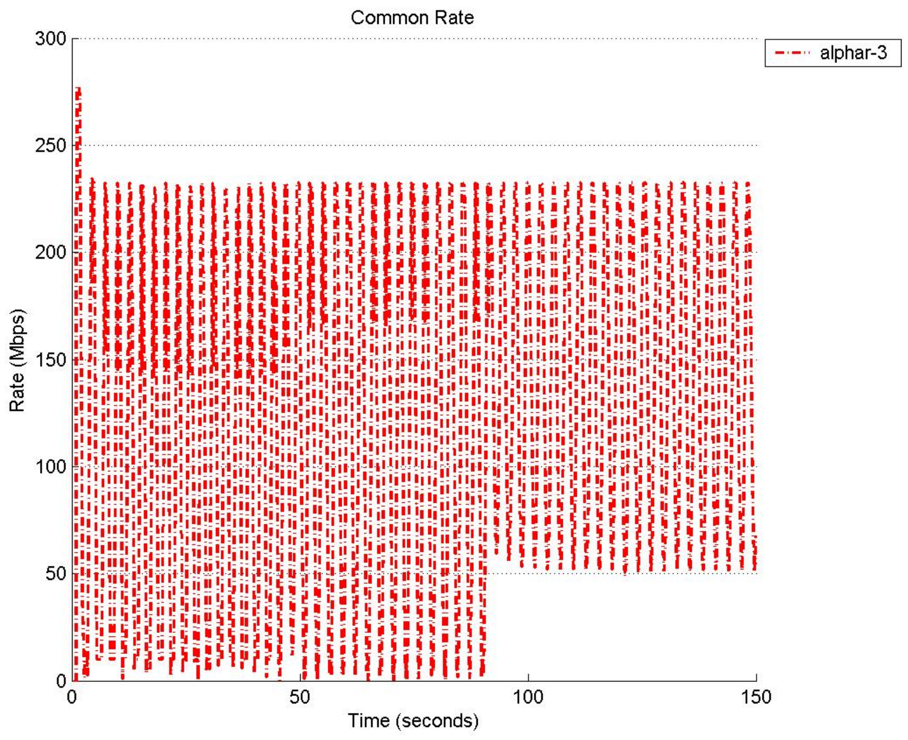


Figure 4.59 Rate for category 3

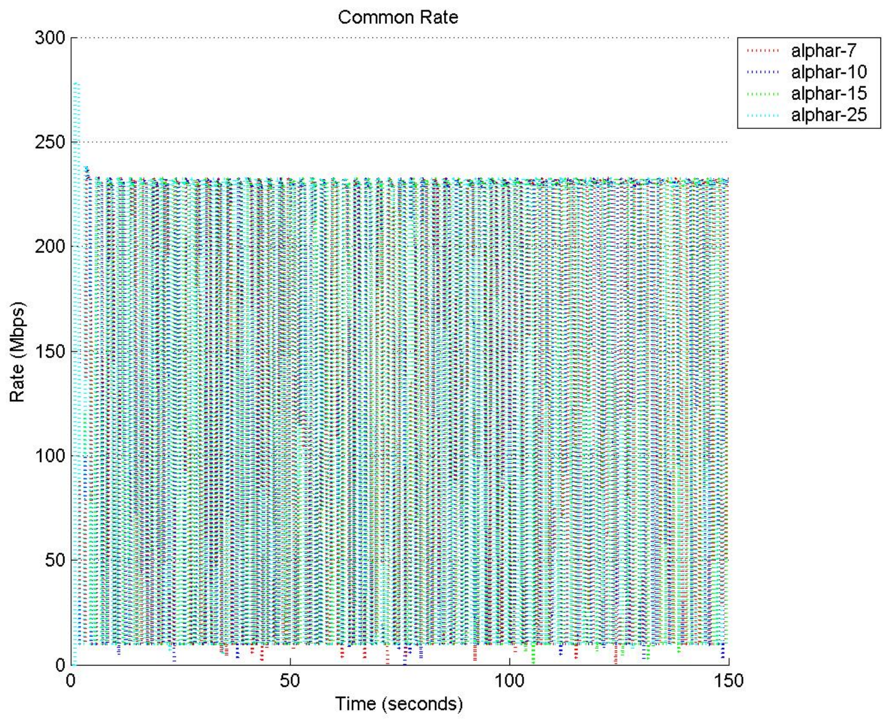


Figure 4.60 Rate for category 4

Chapter 5

Conclusion

Current TCP/IP congestion control algorithms cannot efficiently support new and emerging services needed by the Internet community. In this M.Sc Thesis we analyze the performance analysis of a new congestion control mechanism, called Integrated Dynamic Congestion Control, used in a Resource Management Differential Services Environment.

Following the spirit adopted by the IETF DiffServ working group for the Internet, IDCC contains definition for different classes of aggregated behavior: Premium Traffic Service, Ordinary Traffic Service and Best Effort Traffic Service. The Premium class will be used for real time applications, thus very strict QoS guarantees are needed. For this reason RMD is used as an admission control mechanism, in order to secure the desired bandwidth. The Ordinary class will be used for elastic applications, where the regulation of the rate is possible. In order to avoid congestion in the network and achieve high utilization, IDCC is responsible for regulating its rate. The Best Effort will be used for applications that can afford losses and very big delays.

The results of the simulations in Chapter 5 were very encouraged. The proposed scheme was tested under various environments and found to be able to be used for the needs of the new Internet. The scheme found to be very robust and reliable under all the tests done.

Its performance is not affected neither from the number of flows traversing the link (as long as this information is identified) nor from the propagation delays of the network. The first remark was expected due to the way the algorithm of the scheme is designed. As soon as the capacity needed for the premium class is calculated, the scheme calculates the rate of the ordinary class based on the available capacity and the preferred buffer size. That means that the rate calculation is not influenced by the number of nodes traversing the link. Then this rate is divided by the number of flows and all the sources send equal amount of traffic. The latter remark was unexpected. I expect that the higher the propagation delay is, the longer time should be needed to converge to a steady rate.

In the case where the congested link had large bandwidth (>50Mbps) the scheme was performing very well only with the use of big buffer sizes, whilst in the case of small bandwidth (10Mbps) good performance could be achieved by small buffer sizes too. That is because in order to utilize the links with smaller bandwidths fewer packets are needed, so the rate the sources send packets is less and can be controlled more easily. Furthermore, after the calculation of the correct rate, IDCC reacted very quickly and precisely on any change on the number of flows traversing the router (i.e a premium source starting sending traffic or some ordinary sources stop sending traffic through the network).

The IDCC scheme was found to be fair to all the sources using it. At the case were the propagation delays were the same, the sources starting at the same time were getting the same throughput over the links and the losses were shared with only a small variant among them. The same observation happened when we change the propagation delays so each source has a different one. Unrepentantly the sources shared equally the bandwidth and the losses were not proportional to the delays.

The performance was tested also by using estimation on the maximum number of flows traversing the congested link. In the cases where all the sources were starting almost at the same time, I observed that the scheme can perform very well if it has the correct or an overestimation in the number of flows. In the case where the flows were starting gradually the best performance was achieved by the underestimation on the maximum number of flows.

As it was expected the premium traffic service had no loss in anyone of the scenarios, due to the use of resource management and admission control. The jitter of the network delay was very small in order to satisfy the QoS requirements of premium service class.

The only drawback for IDCC was the end-to-end delay measurements for the premium traffic service. According to the algorithm of the scheme, we have a target buffer size for each of the premium and ordinary buffer sizes. In order to have accepted values of the end-to-end delay we need to have very small target buffer sizes and a big amount of premium traffic to be send through the network. Thus IDCC scheme cannot be used in the edges of the network, where the number of traffic sources is small. In contrary it can be used possibly with very good performance in the core network where the number of sources is very high.

After the completion of the performance analysis, some open issues have risen. Firstly the scheme must be tested under real traffic conditions, which means that the ordinary sources should not be saturated. A real application behavior could be simulated in NS-2 so to achieve the goal mentioned above. By doing that, we will also be able to test the contribution of the best effort traffic in the overall performance of the scheme. Since the ordinary class sources will not be saturated, the best effort class sources would send packets in order to achieve higher utilization.

Also, an estimation function for the number of ordinary flows traversing a link can be created. This function (as concluded from the performance analysis) should be conservative, which means that it should try to overestimate a little the number of flows traversing the link. If the estimation function performs well, most probably the oscillations presented in the beginning of each simulation will disappear. Those oscillations were due to the fact that the scheme had knowledge only for the maximum number of ordinary flows, something that was not precise for the number of flows in the first few seconds (i.e the scheme knew about fifty sources at the 4th simulation second, but only twelve existed).

Furthermore, we can test the one (or a small number) bit feedback in order to calculate the rate of ordinary sources. The routers could mark the ECN bit, which is already available in the IP header of the packet (mark according to a congestion function). Then the source could use this information and regulate its rate. If the performance of using one (or a small number) bit feedback is very close to the one presented in this thesis, then there is no reason to load the network with more data (using full feedback in the packet headers).

Moreover, we must investigate the way this scheme interacts with TCP. TCP applications use different congestion control methods and we should test how IDCC influence their performance. We could also change the window based TCP agents in a way that the window size could be regulated by IDCC.

Bibliography

[1] R. Braden, D. Clark and S. Shenker, “Integrated Services in the Internet Architecture: an Overview”, IETF RFC-1633, Jun. 1994

[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Whang. W. Weiss, “An Architecture for Differentiated Services”, IETF RFC-2475, 1998

[3] Jacobson, Congestion Avoidance and Control, ACM SIGCOMM88, 1988.

[4] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC 2001, Jan. 1997.

[5] Pitsillides A., Chrysostomou, C., Ioannou, P., Lestas, M., “Non-linear Controllers for Congestion Control in Differentiated Services Networks”, Submitted, July 2002.

[6] A. Pitsillides, Petros Ioannou, Marios Lestas, “Congestion Control in Differentiated Services Networks: A Non-linear Control Theory Approach”, Submitted, April 2002.

[7] A. Pitsillides, P. Ioannou, L. Rossides, “Congestion Control for Differentiated-Services using Non-Linear Control Theory”, in Proceedings of the Sixth IEEE Symposium on Computers & Communications, ISCC 2001, Hammamet, Tynisia, 3-5 July 2001, pp. 726-733.

[8] Braden et al, Recommendations on Queue Management and Congestion Avoidance in the Internet, RFC2309, April 1998. circuit, and ATM networks”, Kluwer Academic, 1995.

[9] S. Floyd, V. Jacobson, “Random Early Detection gateways for congestion avoidance”, IEEE/ACM Trans.on Networking, Aug. 1993.

[10] K. Ramakrishnan, and S. Floyd, “The Addition of Explicit Congestion Notification (ECN) to IP”, RFC 3168, September 2001.

[11] S. Floyd, “ TCP and Explicit Congestion Notification”, ACM Computer Communication Review, 24(5), pp. 8-23, Oct. 1994

[12] W. Feng, D. Kandlur, D. Saha, and K. Shin, “A self-configuring RED gateway,” IEEE INFOCOM’99, New York, Mar. 1999.

[13] D. Clark, W. Fang “Explicit Allocation of Best Effort Packet Delivery Service”, IEEE/ACM Transactions on Networking, Vol. 6, No. 4, pp. 362-373, August 1998.

[14] Wu-chang Feng, “Improving Internet Congestion Control and Queue Management Algorithms”, PhD Dissertation, University of Michigan, 1999.

[15] W. Feng, D. Kandlur, D. Saha, and K. Shin, “Blue: A New Class of Active Queue Management Algorithms,” tech. rep., UM CSE-TR-387-99, 1999.

[16] C. Chrysostomou, A. Pitsillides, G. Hadjipollas, A. Sekercioglu, M. Polycarpou, “Fuzzy Explicit Marking for Congestion Control in Differentiated Services Networks”, 2003.

[17] R. Jain, S. Kalyanaraman, R. Goyal, S. Fahmy, R. Viswanathan, ERICA switch algorithm; a complete description, ATM FORUM, AF/96-1172, August 1996.

[18] C.E. Rohrs and R.A. Berry and S.J. O'Halek, A Control Engineer's Look at ATM Congestion Avoidance, IEEE GLOBECOM'95, Singapore, 1995.

[19] A. Pitsillides, P. Ioannou, L. Rossides, “Congestion Control for Differentiated-Services using Non-Linear Control Theory”, in Proceedings of the Sixth IEEE Symposium on Computers & Communications, ISCC 2001, Hammamet, Tynisia, 3-5 July 2001, pp. 726-733.

[20] A. Pitsillides, P. Ioannou, D. Tipper, “Integrated control of connection admission, flow rate, and bandwidth for ATM based networks”, IEEE INFOCOM'96, 15th Conference on Computer Communications, San Francisco, USA, March 1996, pp. 785-793.

[21] A. Pitsillides, J. Lambert, "Adaptive congestion control in ATM based networks: quality of service with high utilization", Journal of Computer Communications, 20, 1997, pp. 1239-1258.

[22] Westberg, L., Jacobsson, M., Karagiannis, G., Oosthoek, S., Partain, D., Rexhepi, V., Szabo, R., Wallentin, P., "Resource Management in Diffserv Framework", Internet Draft, Work in Progress, 2001.

[23] Westberg, L., Karagiannis, G., Partain, D., Oosthoek, S., Jacobsson, M., Rexhepi, V., "Resource Management in Diffserv On DemAnd (RODA) PHR", Internet Draft, Work in progress

[24] Jacobsson, M., “Resource Management in Differentiated Services – A Prototype Implementation”, M.Sc. Thesis, Computer Science/TSS, University of Twente, June 2001.

[25] Heijenk, G., Karagiannis, G., Rexhepi, Westberg, L., "DiffServ Resource Management in IP-based Radio Access Networks", Wireless Personal Multimedia Communications (WPMC'01), Aalborg, Denmark, September 2001.

[26] Marquetant, A., Pop, O., Szabo, R., Dinnyes, G., Turanyi, Z., "Novel enhancements to load control a soft state, lightweight admission control protocol", QofIS'2000 - 2nd International Workshop on Quality of future Internet Services, September 2001.

[27] Csaszar, A., Takacs, A., Szabo, R., Rexhepi, V., Karagiannis, G., "Severe Congestion Handling with Resource Management in Diffserv On Demand", submitted to Networking 2002, May 19-24 2002, Pisa – Italy.

[28] D. Black, S. Blake, M. Carlson, E. Davies, Z.Wang, andW. Weiss, "An Architecture for Differentiated Services", RFC 2475, December 1998. (also see IETF Differentiated Services Working Group http://www.ietf.org/html.charters/diffserv-charter.html). Forwarding PHB Group, RFC 2597, June 1999.

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

[30] J. Heinamen, F. Baker, W. Weiss, J. Wroclawski, Assured Forwarding PHB Group, RFC 2597, June 1999.

[31] Nichols K et al. (1998) Definition of the differentiated Services Field in the Ipv4 and Ipv6 Headers. RFC 2474.

[32] Braden, R., Zhang, L., Berson, S., Herzog, A., Jamin, S., "Resource ReSerVation Protocol (RSVP)-- Version 1 Functional Specification", IETF RFC 2205, 1997.

[33] Baker, F., Iturralde, C. Le Faucher, F., Davie, B., "Aggregation of RSVP for IPv4 and IPv6 Reservations", IETF RFC 3175, 2001.

[34] G. Fehér, K. Németh et al., “Boomerang - A Simple Protocol for Resource Reservation in IP Networks”, IEEE Workshop on QoS Support for Real-Time Internet Applications, Vancouver - Canada, Jun. 1999.

[35] El Allali, H., Heijenk, G., "Resource management in IP-based Radio Access Networks", Proceedings CTIT Workshop on Mobile Communications, ISBN 90-3651-546-7, February 2001.

[36] L. Westberg et al, “Resource Management in Diffserv (RMD) Framework”, IETF’s I-D: draft-westberg-rmdframework-00.txt, April, 2001 (work in progress).

[37] L. Westberg et al, “Resource Management in Diffserv On DemAnd (RODA) PHR”, IETF’s I-D: draftwestberg- rmd-od-phr-00.txt, April, 2001 (work in progress).

[38] C.E. Rohrs and R.A. Berry and S.J. O'Halek, A Control Engineer's Look at ATM Congestion Avoidance, IEEE GLOBECOM'95, Singapore, 1995

[39] A. Pitsillides, P. Ioannou, L. Rossides, Congestion Control for Differentiated-Services using Non-linear Control Theory, TR-01-11, Dept. Computer Science, University of Cyprus, 2001.

[40] Costas Djouvas, M.Sc Thesis, “Extending DiffServ Architecture: Integration of IDCC and RMD Frameworks”.

[41] Network Simulator, NS-2, Homepage, http://www.isi.edu/nsnam/ns/

Appendix A

A.1 Sample Scenario Code written in TCL script language

The following code is a TCL script that produces the topology of the scenario 5 topology where the buffer size is 1000 packets, the delay is 30ms and the ordinary traffic service is represented by TCP sources.

```
##############################
# Create a simulator object #
##############################
set ns [new Simulator]
set FINISH_TIME 150.0

########################
# Simulation variables #
########################
$ns set numRMDFIDs_ 2        ;#How many flowIDs will be used for RMD
signaling messages
$ns set RMDFIDOffset_ 0      ;#Where these flowIDs (DSCPs) start
$ns set RODARefreshPeriod_ 30 ;#in case of RODA PHR group is used

RODAMonitor set RMDBWUnit_ 2000           ;#BytesPerSec - one resource
unit corresponds to this BW value

RODAMonitor set dump_enabled_ 0           ;#Enable dumping in user
readable format
RODAMonitor set auto_dump_enabled_ 0      ;#Non-user readable format for
processing scripts
RODAMonitor set debuglevel_ 0      ;#Debug info

# PerEdge variables
Agent/RMDEdge set dump_enabled_ 0
Agent/RMDEdge set auto_dump_enabled_ 0
Agent/RMDEdge set debuglevel_ 0

# IDCC variables
Queue set limit_ 1000
Queue/IDCCQ set bandw_ 1250
```

```
Queue/IDCCQ set alphar_ 1.5
Queue/IDCCQ set alphap_ 1000
Queue/IDCCQ set xrref_  500
Queue/IDCCQ set xpref_ 50
Queue/IDCCQ set noflows_ 11
Queue/IDCCQ set cq 0
Queue/IDCCQ set lambda_rd 0
Queue/IDCCQ set len 0
Queue/IDCCQ set sampling_time 0.1
Queue/IDCCQ set before_rate_ 0
Queue/IDCCQ set Cpt 0
Queue/IDCCQ set cstat 0
Queue/IDCCQ set intcon 1
Queue/IDCCQ set gamma 1
Queue/IDCCQ set alpha 0.1

Agent/IDCC set packetSize_ 1000
Agent/IDCC set InitRate_ 10
Agent/IDCC set filterpar_ 3
Agent/IdccSink set packetSize_ 30

# Create the files in order to get the traces results
set f [open trace.tr w]
set f5 [open Common_Rate.data w]
set f8 [open Ordinary_Queue.data w]
set nofn_ 24                        ;# Number of nodes
set sources [expr ($nofn_-4)/2]     ;# Number of sources
set destinations [expr $sources + 4]
set router1 [expr $sources]         ;# router 1 id
set router2 [expr $sources+1]       ;# router 2 id
set router3 [expr $sources+2]       ;# router 3 id
set router4 [expr $sources+3]       ;# router 4 id
set idcc_sources 1                  ;# Number of idcc sources

##########################
# The 'finish' procedure #
##########################
proc finish {} {
    global ns f  f5 f8
    $ns flush-trace
    close $f5
    close $f8
    close $f

     exit 0
}

################
# Create nodes #
################
puts nodes
for {set i 0} {$i < $nofn_} {incr i} {
      puts $i
      set n($i) [$ns node]
}

############################
# Trace the congested link #
############################
$ns at 0.0 "$ns trace-queue $n($router2) $n($router3) $f"
```

```
#############################
# Create the RMD-IDCC links #
#############################
$ns duplex-rodalink $n(0) $n($router1) 500Mb 1ms IDCCQ
#$ns duplex-rodalink $n(1) $n($router1) 500Mb 1ms IDCCQ
$ns duplex-rodalink $n($router1) $n($router2) 100Mb 10ms IDCCQ
$ns duplex-rodalink $n($router2) $n($router3) 10Mb 30ms IDCCQ
$ns duplex-rodalink $n($router3) $n($router4) 100Mb 10ms IDCCQ
$ns duplex-rodalink $n($router4) $n(14) 100Mb 1ms IDCCQ
#$ns duplex-rodalink $n($router4) $n(29) 100Mb 1ms IDCCQ

for {set i 0} {$i < $idcc_sources} {incr i} {
      [[$ns link $n($i) $n($router1)] get-roda-mon] init-thresholds-by-
percent 10 90
      [[$ns link $n($router1) $n($i)] get-roda-mon] init-thresholds-by-
percent 10 90
}

    [[$ns link $n($router1) $n($router2)] get-roda-mon] init-thresholds-
by-percent 10 90
    [[$ns link $n($router2) $n($router1)] get-roda-mon] init-thresholds-
by-percent 10 90
    [[$ns link $n($router2) $n($router3)] get-roda-mon] init-thresholds-
by-percent 10 90
    [[$ns link $n($router3) $n($router2)] get-roda-mon] init-thresholds-
by-percent 10 90
    [[$ns link $n($router3) $n($router4)] get-roda-mon] init-thresholds-
by-percent 10 90
    [[$ns link $n($router4) $n($router3)] get-roda-mon] init-thresholds-
by-percent 10 90

for {set i $destinations} {$i < [expr $destinations+$idcc_sources]} {incr
i} {
puts "$i $router4"
    [[$ns link $n($i) $n($router4)] get-roda-mon] init-thresholds-by-
percent 10 90
    [[$ns link $n($router4) $n($i)] get-roda-mon] init-thresholds-by-
percent 10 90
}

set qmon [$ns monitor-queue $n($router2) $n($router3) 0]
set q [[$ns link $n($router2) $n($router3)] queue]

puts " "
puts idcc
for {set i 0} {$i < $idcc_sources} {incr i} {
      set s [expr $i+$destinations]
      puts "$i $s"
      set ingress_agent($i) [new Agent/RMDEdge/RODA]
      set egress_agent($i) [new Agent/RMDEdge/RODA]
      $ns attach-agent $n($i) $ingress_agent($i)
      $ns attach-agent $n($s) $egress_agent($i)
      $ns connect $ingress_agent($i) $egress_agent($i)
      set idcc($i) [new Agent/IDCC]
      set idccsink($i) [new Agent/IdccSink]
      $ingress_agent($i) attach $idcc($i)
      $egress_agent($i) attach $idccsink($i)
}
```

```
######################################################
# Create TCP and CBR Agents and the remaining links #
######################################################

puts " "
puts links

for {set i $idcc_sources} {$i < $sources} {incr i} {
puts "$i $router1"
      if { [expr $i % 5] == 0 } {
         $ns duplex-link $n($i) $n($router1) 500Mb 1ms DropTail
}
      if ([expr $i % 5 == 1]) {
         $ns duplex-link $n($i) $n($router1) 500Mb 2ms DropTail
         }
      if ([expr $i % 5 == 2]) {
         $ns duplex-link $n($i) $n($router1) 500Mb 4ms DropTail
         }
      if ([expr $i % 5 == 3]) {
         $ns duplex-link $n($i) $n($router1) 500Mb 5ms DropTail
         }
      if ([expr $i % 5 == 4]) {
         $ns duplex-link $n($i) $n($router1) 500Mb 8ms DropTail
         }
}

for {set i [expr $destinations+$idcc_sources]} {$i < $nofn_} {incr i} {
puts "$router4 $i"
      if ([expr $i % 5 == 0]) {
         $ns duplex-link $n($router4) $n($i) 500Mb 1ms DropTail
         }
      if ([expr $i % 5 == 1]) {
         $ns duplex-link $n($router4) $n($i) 500Mb 2ms DropTail
         }
      if ([expr $i % 5 == 2]) {
         $ns duplex-link $n($router4) $n($i) 500Mb 4ms DropTail
         }
      if ([expr $i % 5 == 3]) {
         $ns duplex-link $n($router4) $n($i) 500Mb 5ms DropTail
         }
      if ([expr $i % 5 == 4]) {
         $ns duplex-link $n($router4) $n($i) 500Mb 8ms DropTail
         }
}

set t 1.0
puts " "
puts ftp
for {set i $idcc_sources} {$i < $sources-2} {incr i} {
      set s [expr $i+$destinations]
      puts "$i $s"
      set tcp($i) [new Agent/TCP/Reno]
      set sink($i) [new Agent/TCPSink]
      $ns attach-agent $n($i) $tcp($i)
      $ns attach-agent $n($s) $sink($i)
      $ns connect $tcp($i) $sink($i)
      set ftp($i) [new Application/FTP]
      $ftp($i) attach-agent $tcp($i)
      $ns at $t "$ftp($i) start"
      $ns at $FINISH_TIME "$ftp($i) stop"
      $ftp($i) set fid_ 0
      set $t [expr $t + 0.4]
}

puts " "
puts cbr
for {set i [expr $sources-2]} {$i < $sources} {incr i} {
      set s [expr $i+$destinations]
      puts "$i $s"
      set cbr($i) [new Agent/CBR]
      set null($i) [new Agent/Null]
      $ns attach-agent $n($i) $cbr($i)
      $ns attach-agent $n($s) $null($i)
      $ns connect $cbr($i) $null($i)
      $cbr($i) set packetSize_ 500
      $cbr($i) set interval_ 0.0005
      $ns at 1.1 "$cbr($i) start"
      $ns at $FINISH_TIME "$cbr($i) stop"
      $cbr($i) set fid_ 2
}

$ingress_agent(0) set fid_ 1
$idcc(0) set fid_ 1
#$ingress_agent(1) set fid_ 1
#$idcc(1) set fid_ 1

$idcc(0) set InitRate_ 250
#$idcc(1) set InitRate_ 250

    #This automatically calls "Admitted $ingress_agent" upon admission
    #Otherwise, auto calls "Refused $ingress_agent"

      $ns at 1.0 "$ingress_agent(0) QoSRequest 125" ;# in units!!!

#    $ns at 2.0 "$ingress_agent(1) QoSRequest 250"
#    $ns at 2.2 "$ingress_agent(2) QoSRequest 0"
#    $ns at 2.3 "$ingress_agent(3) QoSRequest 0"
#    $ns at 2.4 "$ingress_agent(4) QoSRequest 0"

#I can self define what needs to be done upon admittance, maybe I don't
want to start traffic at all because I just
#care about signaling traffic...
proc Admitted {ingress_agent} {
      global ns FINISH_TIME
      [$ingress_agent get-agent] start

      $ns at $FINISH_TIME "[$ingress_agent get-agent] stop"
      $ns at $FINISH_TIME "$ingress_agent QoSRelease"
      #May send explicit release request message, and will definetly call
"Released $ingress_agent"
}

#Self define what happens when the connection is refused
#Maybe in that case I can change my DSCP to a best-effort DSCP and attach
my regular agent directly on the node...
proc Refused {ingress_agent} {
      global ns n
      set egress_agent [$n([$ingress_agent set dst_addr_]) agent
[$ingress_agent set dst_port_]]
      delete [$ingress_agent get-agent]
      $ns detach-agent [$ingress_agent set node_] $ingress_agent
      $ns detach-agent [$egress_agent set node_] $egress_agent
      $ns at [expr [$ns now] + 0.0000000001] "delete $ingress_agent"
      $ns at [expr [$ns now] + 0.0000000001] "delete $egress_agent"
}

proc Released {ingress_agent} {
      global ns n
      set egress_agent [$n([$ingress_agent dst_addr_]) agent
[$ingress_agent dst_port_]]
      delete [$ingress_agent get-traffic-app]
      delete [$ingress_agent get-agent]

      #The following need to be manually done just like in case of any
regular agents, like TCP or UDP...
      $ns at-now  "$ns detach-agent [$ingress_agent set node_]
$ingress_agent"
      $ns at-now  "$ns detach-agent [$egress_agent set node_]
$egress_agent"
      $ns at-now  "delete $ingress_agent"
      $ns at-now  "delete $egress_agent"
}

proc record2 {qs} {
    global f5 f8 ns q
    set ns [Simulator instance]
    set rate [$q set lambda_rd]

    set time 0.01
    set now [$ns now]
    set curqueue [$q set cq]

    puts $f8 "$now $curqueue"
    puts $f5 "$now $rate"
 $ns at [expr $now+$time] "record2 $qs"
}

proc record1 {so} {
    global f4 ns
    set ns [Simulator instance]
    set qrate [$so set rate_]
    set time 0.01
    set now [$ns now]
    puts $f4 "$now $qrate"

    $ns at [expr $now+$time] "record1 $so"
}

$ns at 0.5 "record2 $qmon"

#Call the finish procedure after x seconds of simulation time
$ns at $FINISH_TIME "finish"

#Run the simulation
$ns run
```

A.2 C++ code used for the end-to-end delay and jitter

In order to compute the end-to-end delay and jitter C++ code was used. The program was split in two functions. The first function reads all the lines of the trace file, selects the records that refer to flow packets that we are interested and are sent from source node to sink node and that either are just entering the queue at node s or are just arriving at node t. The selected lines are placed in the file an output file. The second function reads that output file, and records the delays of all the packets belonging to flow f that start from a source at node and are received at a sink at node. It also records the percentage of the lost packets and the jitter. It prints all the statistics to the standard output and the delay of each packet is written to a file specified by the user.

Function 1

```
#include <iostream.h>
#include <string.h>
#include <fstream.h>
#include <stdlib.h>
#include <iomanip.h>
#include <math.h>

int main(int argc, char * argv[])
{
  // Packet type to be excluded - e.g., ack in case of TCP, etc.:
  char *Type;
  Type = "rtProtoLS";

  cout << "\n" << "The packet type that will be excluded: " << Type << "\n\n";

  // The constants s, t, and f must be set via STANDARD INPUT:
  int s = atoi(argv[1]); //12; // Source node.
  int t = atoi(argv[2]); //63; // Sink node.
  int f = atoi(argv[3]); //0;  // Flow id.

  // Here we get the number N of rows in the input data file:

  ifstream count_file(argv[4]);//"outidcc.tr");
  int i = 0;
  char buff[100];
  while (count_file) {
    count_file.getline(buff, 100);
    i++;
  }
  int N = i-1;
  count_file.close();

  ifstream in_file(argv[4]);
  ofstream out_file(argv[5]); //"selected");

  char line[100];
  char *vector[12];
  char *tokenPtr;
  char buffer[100];

  int n = 0;

//while (in_file) {
  while (n < N)
  {
    n++;
    in_file.getline(line, 100);

    // Each "line" is a row of characters in the original file out.tr.
    // These characters form 12 words separated by 11 blank delimiters.
    // Thus, each character array "line" yields a 12-word array "vector".
    // The following parses ("tokenizes") "line" into "vector":
//  buffer = line;
    strcpy (buffer, line);
    // It is necessary to use a _copy_ of "line", "buffer", instead of "line"
    // itself, since an application of "strtok" function modifies its first
    // argument!
    tokenPtr = strtok(buffer, " ");
    for (int j=0; j < 11; j++) {
      vector[j] = tokenPtr;
      tokenPtr = strtok(NULL, " ");
    }
    vector[11] = tokenPtr;

    // Now we are ready to examine each line of our out.tr file. We first need
    // some labels:
    char *sign;
    sign = vector[0];
    float time_stamp = atof(vector[1]);
    int from_node = atoi(vector[2]);
    int to_node = atoi(vector[3]);
    char *type;
    type = vector[4];
    int flow  = atoi(vector[7]);
    int source_node = atoi(vector[8]);
    int sink_node = atoi(vector[9]);
    int packet_num = atoi(vector[10]);
    int packet_id = atoi(vector[11]);
    // Finally, here is the criterion for finding the desirable lines of the
    // file out.tr and placing them into a new file called "selected".
    if ((((*sign == '+') && (from_node == s) && (flow == f) && (source_node
== s) && (sink_node == t)) || ((*sign == 'r') && (to_node == t) && (flow ==
f) && (source_node == s) && (sink_node == t))) && (strcmp(type,Type)!=0)) {
      out_file << line << "\n";
    }
  }
  in_file.close();
  out_file.close();

 return 0;
}
```

Function 2

```
#include <iostream.h>
#include <string.h>
#include <fstream.h>
#include <stdlib.h>
#include <iomanip.h>
#include <math.h>

int main(int argc, char * argv[])
{
  // The constants s, t, and f must be set at the beginning.
  int s = atoi(argv[1]); //12; // Source node.
  int t = atoi(argv[2]); //63; // Sink node.
  int f = atoi(argv[3]); //0;  // Flow id.

  ifstream in_file(argv[4]);//"selected");
  int i = 0;
  char buffer[100];
  while (in_file) {
    in_file.getline(buffer, 100);
    i++;
  }
  // From here we get the number M of rows in the input data file
  // (i.e., the size of the character array "data" defined below):
  int M = i-1;
  in_file.close();

  char sign[M][2];
  float time_stamp[M];
  int packet_id[M];
  ifstream inn_file(argv[4]);//"selected");
  i = 0;
  while (i < M){
    char data[100];
    inn_file.getline(data, 100);
    strcpy(buffer, data);
    char *tokenPtr;
    tokenPtr = strtok(buffer, " ");
    strcpy(sign[i], tokenPtr);
    tokenPtr = strtok(NULL, " ");
    time_stamp[i] = atof(tokenPtr);
    for (int j=1; j < 11; j++) {
      tokenPtr = strtok(NULL, " ");
    }
    packet_id[i] = atoi(tokenPtr);
    i++;
  }
  inn_file.close();

  // The array "delay" shall have plenty of zeros at the tail...
  float delay[M];
  int k = 0;
  int l = 0;
  for (int i=0; i < M; i++) {
    if ((*sign[i] == '+')) {
      k++;
      for (int j=0; j < M; j++) {
           if ((*sign[j] == 'r') && (packet_id[j] == packet_id[i])) {
             delay[l] = time_stamp[j] - time_stamp[i];
             l++;
           }
      }
    }
  }

  int T = k; // The number of packets transmitted.
  int R = l; // The number of packets received.
  int L = T - R; // The number of packets lost.
  cout << "\n\n";
  cout << "The number of packets transmitted T = " << T << ".\n";
  cout << "The number of packets received R = " << R << ".\n";
  cout << "The number of packets lost L = " << L << ".\n";

  //////////////////////////////////////////////////////////////////
  // The following computes the percentage, truncated at two decimals,
  // of the lost packets:
  double loss_ratio = floor(10000*L/T)/100;
  cout << "\n" << loss_ratio << "% of all packets of flow " << f << " sent
from node " << s << " to node " << t << " were lost." << "\n\n";
```

```
  ////////////////////////////////////////////////////////////////////////
  // This computes delay sample mean and prints it to the standard output:
  double sigma = 0;
  for (l=0; l < R; l++){
    sigma = sigma + delay[l];
  }
  double delay_mean = sigma/R;
  cout << "Delay mean for flow " << f << " = " << 1000*delay_mean << " ms"
<< "\n\n";

  ///////////////////////////////
  // This computes delay variance:
  double quad = 0;
  for (l=0; l < R; l++){
    quad = quad + pow(delay[l] - delay_mean,2);
  }
  double delay_variance = quad/R;

  ////////////////////////////////////////////////////////////
  // This computes delay sample standard variation and prints it
  // to the standard output:
  double delay_standard_deviation = sqrt(delay_variance);
  cout << "Jitter for flow " << f << " = " << 1000*delay_standard_deviation
<< " ms" << "\n\n";

  ofstream out_file(argv[5]);

  for (l=0; l < R; l++){
    out_file << l << "  " << delay[l] << endl;
  }

  out_file.close();
  return 0;
}
```

A.3 Awk code used for the class throughput differentiation and drops

In order to record the throughput at a selected link and the drops for each flow, some simple awk scripts were written that used as input file the trace file generated by ns-2 after the simulation end.

```
    exec awk {
        {
        if($1 == "-")
        {
            old_data = old_data + $6
            print $2, old_data*8.0/$2
        }
        }
    } trace.queue > throughput.cor

    exec awk {
        {
        if($1 == "d" && $8 == "1")
        {
            old_data = old_data + 1
            print $2, old_data
        }
        }
    } trace.queue > lossesPremiun.cor

    exec awk {
        {
        if($1 == "d" && $8 == "0")
        {
            old_data = old_data + 1
            print $2, old_data
        }
        }
    } trace.queue > lossesOrdinary.cor
```

A.4 Sample Matlab Code for the creation or the graphs

```
load ('E:\thesis\simulations\scenario3\idcc\1000-buffer\no-stops\10ms\correct-estimation\delay_0_cor101000.data');
load ('E:\thesis\simulations\scenario3\idcc\1000-buffer\no-stops\10ms\correct-estimation\15_premium_300_ordinary\delay_0_cor151000.data');
load ('E:\thesis\simulations\scenario3\idcc\1000-buffer\no-stops\30ms\correct-estimation\delay_0_cor301000.data');
load ('E:\thesis\simulations\scenario3\idcc\200-buffer\no-stops\10ms\correct-estimation\delay_0_cor10200.data');
load ('E:\thesis\simulations\scenario3\idcc\200-buffer\no-stops\10ms\correct-estimation\15_premium_100_ordinary\delay_0_cor15200.data');
load ('E:\thesis\simulations\scenario3\idcc\200-buffer\no-stops\30ms\correct-estimation\delay_0_cor30200.data');
%load ('E:\thesis\simulations\scenario2\no-stops\queue-200\100-nodes\delay_4_55_100over.data');

time = delay_0_cor101000(:,1);
time1 = delay_0_cor151000(:,1);
time2 = delay_0_cor301000(:,1);
time3 = delay_0_cor10200(:,1);
time4 = delay_0_cor15200(:,1);
time5 = delay_0_cor30200(:,1);
%time6 = delay_0_51_100over(:,1);
delay_cor = delay_0_cor101000(:,2);
delay_over = delay_0_cor151000(:,2);
delay_under = delay_0_cor301000(:,2);
```

```
delay_200 = delay_0_cor10200(:,2);
delay_1000 = delay_0_cor15200(:,2);
losses_100 = delay_0_cor30200(:,2);
%delay_100over = delay_0_51_100over(:,2);

figure
plot (time, delay_cor, 'Color', 'red', 'LineWidth', 1.8, 'LineStyle', '-')
hold
plot (time1, delay_over, 'Color', 'blue', 'LineWidth', 1.8, 'LineStyle', '-')
plot (time2, delay_under, 'Color', 'green', 'LineWidth', 1.8, 'LineStyle', '-')
plot (time3, delay_200, 'Color', 'red', 'LineWidth', 1.8, 'LineStyle', ':')
plot (time4, delay_1000, 'Color', 'blue', 'LineWidth', 1.8, 'LineStyle', ':')
plot (time5, losses_100, 'Color', 'green', 'LineWidth', 1.8, 'LineStyle', ':')
%plot (time6, delay_100over, 'Color', 'cyan', 'LineWidth', 1.8, 'LineStyle', '-')

%axis([0 100 0 50]);
title('End-to-End Delay');
xlabel('Packet Number');
ylabel('Time (seconds)');
set(gca,'YGrid','on');
%legend('delay-cor', 'delay-over', 'delay-under', -1);
%legend('delay-10sources', 'delay-50sources', -1);
%legend('delay-cor', 'delay-over', 'delay-under', 'delay-100over', -1);
legend('buffer-1000-50-delay-10', 'buffer-1000-30-delay-10', 'buffer-1000-50-delay-30',
'buffer-200-30-delay-10', 'buffer-200-15-delay-10', 'buffer-200-30-delay-30', -1);

cd scenario3
print -djpeg Delay_premium
cd ..
```