Performance Analysis on HTTP Traffic and Traffic Mixtures with Competing TCP and UDP Flows

IIP Mixture Project

Oriana Riva Jarno Saarto

Markku Kojo

May 2004

University of Helsinki - Department of Computer Science

Contents

1 Introduction

In this document, we report the results of the performance experiments executed in the IIP Mixture project during Phase I. We first describe the test methodology and arrangements, including a short description of the test environment that has been used in the tests in terms of link characteristics, workload models, metrics and TCP enhancements for the performance measurements. The test cases under the study can be classified in two main categories. The first one studies HTTP 1.0/HTTP 1.1 traffic performance in wireless environment and the second one traffic mixtures with competing TCP and UDP traffic. RED [FJ93] and DiffServ mechanism are applied in part of the traffic mixture tests.

2 Objectives and methods

Our objective is to evaluate TCP protocol performance with different types of competing traffic in a wireless environment with great variation on prevalent network conditions. The link may experience unexpected black-outs increasing the packet error rate dramatically. With ARQ, the link may recover some of the packet losses, but the link level retransmissions add additional delays to delivering of packets. There are several RFCs, and drafts that consider means to ameliorate the performance of TCP environments of this kind. From these we will select a reduced set that represents options that nowadays are included in many TCP implementations or which are likely to be included in the near future. We call this set Baseline TCP. Other TCP variants will be compared against it.

The workload includes mixtures of TCP, UDP and HTTP traffic. DiffServ will be used to divide traffic types to classes, which can be prioritized. By prioritizing we can assure for example that the UDP traffic gets enough bandwidth and thus reduce the amount of jitter in the stream while mandating a decent speed on other concurrent transfers. To have better understanding of the behaviours in transfers with each workload, we first run tests with HTTP and other traffic types separately.

The wireless link is emulated using Seawind Network Emulator, which allows us to run real TCP implementations in the end hosts. The last-hop router before the wireless link is emulated with Traffic Control (tc) of Linux Kernel. The traffic control will be applied in the TUN interface created by Seawind and the last-hop router buffers are set by means of tc parameters. Each scenario is usually tested with 20 or 40 replications in order to obtain more accurate statistics.

To define the network parameters, we first run a number of preliminary tests with a limited amount of replications. The results indicate which TCP variants and combinations should be tested more. By running preliminary tests we minimize the work that has to be repeated while fine-tuning the network environment and link characteristics.

The types of tests run can be classified in two categories according to the traffic workload employed: HTTP tests and Traffic Mixture tests.

HTTP tests

In HTTP 1.0 the browser transfers each downloaded object in its own TCP connection. This causes each object's transfer to experience slow start phase. To speed up the download process, the browser may open a number of parallel TCP. The maximum number of connections is limited by the browser. Despite all open connections still spend almost all the transfer time in slow start, the cumulative transfer rate can be good with the use of several concurrent connections, since the capacity of the line is better in use than with one connection. Current browsers use typically four to six concurrent connections. HTTP 1.0 is still widely in use in the Internet today.

In HTTP 1.1, the requests are pipelined to maximize the use of the persistent connections. The persistent connections themselves are very similar to the keep-alive connection modification in HTTP 1.0, with the exception of being for a proxy. As in HTTP 1.0 with multiple connections, the browser limits the amount of concurrent connections. The maximum number of simultaneous connections to a server in HTTP 1.1 should be two [FIE99]

In HTTP 1.1, a client may request a new object before the previous connection is finished and the request is placed in queue if the maximum number of simultaneous connections to a server is in use. A queued request will be started as soon as a connection becomes available. Pipelined requests make the use of persistent connections far more efficient since a same connection can often actually be used. The pipelining makes the server able to start sending a new file immediately without a lag in the transmission, which might cause the drawback of the multiple slow start phases. HTTP 1.1 is widely supported by the web servers in the Internet today. The support for the protocol is disabled, however, in many web servers to save server resources [Saa03].

HTTP 1.1 may improve the performance of TCP transfers notably with workload typical in World Wide Web compared to HTTP 1.0 by introducing persistent connections and request pipelining. These techniques may help avoiding multiple slow-start phases in a web requests consisting of more than one objects.

The objective is to test the performance of HTTP 1.1 in wireless environment with great variation on prevalent network conditions. The link may experience unexpected black-outs increasing the packet error

rate dramatically. The link layer is capable of retransmissions, which reduce the residual frame error rate, but add additional delays to the TCP transmission of packets. The workloads selected are similar to the HTTP 1.0 workloads tested in [Saa03]. The experiments are follow-up to the HTTP 1.0 experiments and are to be compared to them. To make sure no false conclusions about HTTP 1.0 performance are made because of slightly different test environment, also a selected set of HTTP 1.0 tests are run.

To emulate the wireless link and the last-hop router we use the *Seawind Wireless Network Emulator* [KGMS01], which allows us to test with different network and link types and to repeat tests with exactly same network conditions with different workload or TCP variant. We make use of the results of tests with HTTP 1.0 [Saa03]. This saves both time and allows us to compare the results to those tests. To make sure the changes in Seawind parameters does not influence the comparison we rerun some of the HTTP 1.0 tests also with the parameters that are used in HTTP 1.1 tests.

Traffic Mixture tests

The main result of the study conducted in [Riva03] was a comparative study of TCP (and UDP) performance when RED or the traditional Tail Drop mechanism was employed in the last-hop router in front of the bottleneck link. We used a small testbed of hosts handling packets and the last-hop router employed the DiffServ architecture to classify the UDP and TCP traffic. The intent was to model a wired network connected to a wireless network through a *well-behaving* wireless link. Indeed the last-hop link did not present any other wireless characteristic than the limited bandwidth, fixed by means of Linux Traffic Control [HMVV02] to 140kbit/s. This allowed us to analyse how different queueing disciplines, such as Tail Drop and RED, perform when they are employed in a DiffServ router. We pass now to analyse how the same mechanisms behave in a real wireless environment, where the last hop link is a wireless link emulated with Seawind emulator.

As regards the RED algorithm, its behaviour is strongly dependent on several parameters. The default values are shown in Table 1. Some variations will be considered in order to address specific issues, such Slow Start Overshoot or high throughput or minimization of the dropped packets, and in order to provide a better study of the tuning of RED parameters and determine how the TCP performance achieved may vary according to them. The max_{th} is usually set enough high to keep the percentage of discarded packets low and to have a wide range max_{th}-min_{th}. w_q and max_p are generally set by default. With such a max_p, the RED gateway marks around 1/5th of the incoming packets when the average queue size is close to max_{th}.

Parameter	Description	Suggested Value
buffer-size (bytes)	Physical buffer queue size	2* max_{th}
min_{th} (bytes)	Lower threshold below which no packet is dropped	5 packets or 5 times *avpkt*
max_{th} (bytes)	Upper threshold above which all incoming packets are dropped	3 times min_{th}
avpkt (bytes)	The average number of bytes in a packet	free
w_q	Weight in the low-pass filter used to compute the *avg*	0,002
max_p (float)	Maximum drop probability	0.1

Table 1. The RED Parameters

In most of the cases, RED will be used in combination with *Explicit Congestion Notification (ECN)* [RFB01]. This mechanism can signal the router of incipient congestion without discarding packets, but simply setting a bit in the packet header. In this way the TCP sender is able to understand if it must reduce its transmission rate because of incipient congestion.

3 Experimentation Environment

To reduce the amount of traffic not related to the selected workload, the test environment is located in a private network consisting of three machines running the Linux Operating System (kernel version 2.4.22). The test environment consists of two end hosts communicating with each other through a third machine, which runs the Seawind Simulation Process (SP) and models the DiffServ capable router. Pehessari-23 is one of end hosts and it is directly connected to the emulation host with a cross-wired 100Mbit/s cable. Pehesaari-23, Kajaa, and Tainio are connected through a 100Mbit/s switch. Between the emulation host and the other end host (Tainio) is the emulation host (Kajaa) in which Seawind emulates a wireless link. The bandwidth is reduced to 64kbit/s. The router can employ the Tail Drop discipline or the RED algorithm as mechanism of router queue management.

In Figure 1, we illustrate the emulation environment and our approach to emulate it. The DiffServ router is configured using a HTB (Hierarchical Token Bucket) [HMVV02] queueing discipline. We create two or three classes of services depending on the considered workload. The WLGs for the sender and the receiver are in Pehesaari-23 and Tainio respectively. The Simulation Process (SP) is located in Kajaa and the two Network Protocol Adapter (NPAs) are in Kajaa and Tainio.

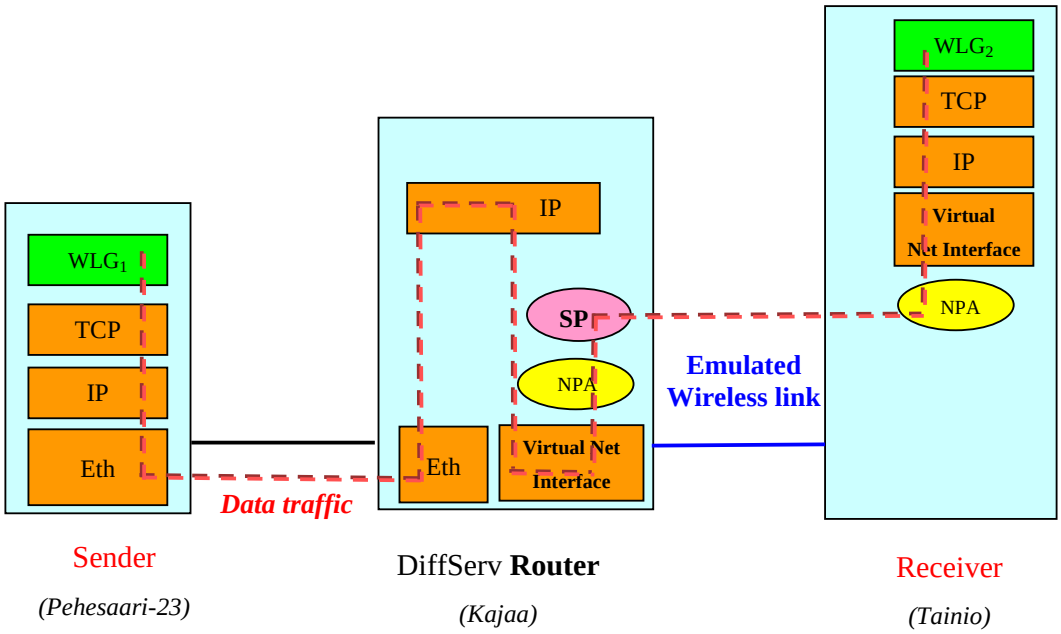

Figure 1. The topology of the testbed network

As regards the link characteristics, we will use a configuration similar to the one used in HTTP 1.0 tests (see [Saa03] for details). In addition to the bandwidth of 64kbps used in the previous tests, we will test with a higher bandwidth. We will also test with larger MTU size in addition to the size of 576 bytes used in HTTP 1.0 tests and a broader set of error models. The error-model is based on a two-state Markov model. To have exactly the same network behaviour in each test, we will use a fixed distribution for lengths of good and bad states. The good state represents a good error free wireless environment and the other state represents a bad environment with high error probability. Each time a good state ends the bad state starts and vice versa. In bad states, ARQ link layer retransmissions with several amount of persistency are emulated. If a packet is dropped, the link layer retransmission is emulated by delaying the packet for about one RTT for each retransmission instead of actually dropping it. The one RTT additional delay is defined as 700ms. After the emulated retransmission, the packet may experience a drop again and might need additional retransmissions. We will also test with and without link layer ARQ persistency. Table 2 describes the parameters of the link configuration and Table 3 specifies the link error model and the number or retransmissions when link layer ARQ is used.

Characteristics	Value
Propagation delay	300 ms
Transmission delay	MTU / Bandwidth
Transmission rate	64 & 384 kbps
MTU	576 & 1400 bytes
Link send buffer size	2*Bandwidth*RTT
Link receive buffer size	2*Bandwidth*RTT
Link error model	2-state Markov model with and without link layer ARQ

Table 2. Link characteristics

Parameter	Good state	Bad state
Error probability	0 %	83 %
Type of state length function	user defined	user defined
Retransmission delay	700 ms	700 ms
Number of retransmissions	0/2/4	0/2/4

Table 3. Error model

4 TCP options and enhancements

The TCP enhancements that will be used in the experiments are described in Table 4. The table is divided between *Baseline TCP* and several *TCP variants*, including different enhancements to be tested. The baseline TCP has includes many TCP enhancements suggested for wireless links. In addition to those, we have disabled some TCP functionalities in order to make the TCP less Linux specific and to be able to analyze the TCP behaviours better. In baseline TCP, we have disabled Rate Halving and Quick ACKs, and modified the Delayed Acknowledgment time limit to 200 ms.

	TCP feature
Baseline TCP	Initial Window of 2 segments (IW2) Limited transmit Selective Acknowledgments (SACK) TCP Timestamps Delayed ACKs (200 ms) No Rate halving No Quick ACKs
TCP Variants	D-SACK Forward-RTO v1 (F-RTO 1) Forward-RTO v2 (F-RTO 2) Eifel (TCP Timestamps) CBI (Control Block Interdependence for the reuse of ssthresh value) IW4 (Increased Initial Window up to 4 segments) F-RTO 2 Early Retransmit (optional) Quick ACKs (optional) F-RTO 2 + CBI + IW4 F-RTO 1 + CBI + IW4

Table 4. TCP enhancements used in tests

The tests will be run first for the baseline TCP. Then each option listed in enhanced TCP will be changed one more at a time and compared to the baseline TCP. Some TCP enhancements that will be considered are the followings.

- *Increased Initial Window* [AFP98] was found to have a great effect in performance with workload typical in World Wide Web. Increasing the initial window from 2 up to 4 segments will reduce the amount of time required by the slow start phase. Therefore, an initial window of four segments will be combined with selected enhancements. To be able to compare the results of HTTP 1.0 tests to HTTP 1.1 results, the initial window is, however, set to two segments in baseline TCP. With longer bulk-type transfers, the effect of increased initial window is less significant.

- *D-SACK* [FMMP00] did not improve performance in tests with HTTP 1.0 in general [Saa03] nor with tests using concurrent TCP and streaming traffic flows [kulve03]. The main reason was that the transferred objects were so small that the readjustment of the congestion window parameters did not have time to make effect to the performance of the TCP transfers. Since with D-SACK algorithm the number of retransmitted packets during a recovery from an RTO is identical to the number when using

conventional RTO, the performance was not improved. With HTTP 1.1, the same connection is used for many of the transferred objects, which gives more time to the TCP to have benefit of the congestion widow readjustment.

- *F-RTO* [SKR02] may improve the performance in cases of unnecessary retransmissions caused by spurious retransmission timeouts. In tests with HTTP 1.0, there were some cases in which the F-RTO algorithm was not able to conclude a retransmission of a packet unnecessary. In F-RTO 2, the detection of unnecessary retransmissions has been improved by making use of SACK information when making the conclusion.

- *CBI* [Touch97]*:* each TCP connection maintains state in the TCP Control Block (TCB) structure. The TCB contains information about connection state. This contains information of RTT estimator, Initial window size, ssthresh and reordering in previous connection. When the option is enabled, these states can be shared across concurrent connections to the same host. We have run some tests disabling this option in order to study phenomena such as Slow Start Overshoot related to the starting of the TCP connection.

- *Eifel (Timestamps option)* [LM03] allows a TCP sender to detect a posteriori whether it has entered loss recovery unnecessarily. It requires that the TCP Timestamps option defined in [JBB92] be enabled for a connection.

- *Random Early Detection (*RED) [FJ93] allows the sender to prevent congestion collapse by signalling to the sender of the incipient congestion before the queue overflows. RED congestion control mechanisms monitor the average queue size for the output queue and choose connections to notify of the incoming congestion. It discards packets that arrive at the router selectively, hence TCP connections, after they have detected lost packets, reduce their transmission rate and congestion can be prevented. Moreover, RED drops packets in a probabilistic manner and such probability grows with the estimated average size of the queue.

- *Explicit Congestion Notification* (ECN) [RFB01] is a mechanism that allows signalling the router of incipient congestion without discarding packets, but simply setting a bit in the packets header. In this way, the TCP sender is able to understand whether it must reduce its transmission rate because of incipient congestion (ECN bit set).

5 Workload Models

5.1 Workload for HTTP tests

The HTTP client is on the mobile host, and the server is on the fixed network. The requested objects are transmitted using downlink of the client. Although there are situations in WWW traffic in which the uplink is more in use than the downlink (error situations, forwarding, uploads), they are not common and not very interesting either. We assume that the transfers of requests and acknowledgments from the HTTP client to the HTTP server will not cause congestion at any point, because the server resides in a fixed network remarkably faster than the wireless link between the last-hop router and the client. Transfers from the server to the client will most probably experience congestion in some cases, however.

The HTTP 1.0 traffic is generated using an HTTP traffic generator developed earlier in the University of Helsinki. We made a few modifications to the tool to allow us to have full control over the sizes of requests and responses and incorporated the tool into Seawind to have the benefits of automated test runs and Graphical User Interface. With the tool we can simulate customizable HTTP requests using HTTP 1.0 like traffic. The tool makes a HTTP request for each object in a new connection. The same tool was used in the HTTP 1.0 test in [Saa03].

The HTTP 1.1 traffic will be generated using a WWW Client based on W3C sample libraries together with Apache HTTP server for HTTP traffic. Both client and server will be configured to use persistent connections and request pipelining with HTTP 1.1 protocol. The Apache server will be compiled for maximum performance using the most effective system calls and threading methods available for the Operating System and processor family used. The Apache configuration file will be configured for high performance. The server will not launch new child processes while test is taking place, but forks processes before a test is started.

Table 5 describes the sizes of small, medium and large objects for main objects and small and medium object sizes for in-line objects. All requests are 350 bytes of size. Table 6 describes the selected types of workload.

object type	**small object size**	**medium object size**	**large object size**
main object	6288 bytes / 6740 bytes	12576 bytes / 13480 bytes	71788 bytes / 72792 bytes
in-line object	2096 bytes / 2696 bytes	8384 bytes / 9436 bytes	-

Table 5. Sizes of object types

Label	main object size	in-line object size	# of in-line objects
s+2s	Small	Small	2
s+8m	Small	Medium	8
m+8m	Medium	Medium	8
l+8m	Large	Medium	8

Table 6. Workloads used in tests

5.2 Workload for Traffic Mixture tests

We have chosen to run tests in a dynamic environment with competing traffic flows. Generally the type of workload used is classified on the basis of the individual connections involved, the number of contemporary flows and their position in time. As regards the start time for traffic mixture flows, it may be either the same for all the flows or different. For instance, it may be interesting to delay some traffic in order to study how their arrival could impact on the connections already going on and how the performance itself may change according to the start time. The exact starting times will be decided based on the preliminary tests. The workload source and sink computers use the proposed Baseline TCP or some variants, UDP or HTTP. The traffic is generated using three different types of traffic generator: JTG (Jugi's Traffic Generator) for the UDP flows, TTCP (TCP test utility) for TCP flows [TTCP98].

The Nagle algorithm in TCP says that a TCP connection can have only one outstanding small segment that has not yet been acknowledged ("small" means less than the segment size). In fact, to send 1 byte of data we need to build a 41-byte packets (20 bytes for the IP header and 20 bytes for the TCP header). These small packets, called tynigrams, do not represent a problem on LANs, but they can add congestion on wide area networks. Referring to our tests on a small testbed network this algorithm could disturb our results. Thus in order to prevent this, we have set the length of buffers written to the network to four times the MSS of the connection (4*524 bytes). The basic traffic characteristics are the following:

- TCP traffic: length of buffers=2096 bytes, number of buffers=90,100
- UDP traffic: packet size =132 bytes, constant bit rate 32kbit/s
- HTTP traffic: small object size (see Table 5)

Accordingly to the workload in use the router will be configured consequently. With TCP and UDP traffic two service classes may be considered. When combining HTTP traffic with TCP bulk data transfers and UDP streaming traffic, three service classes may be employed where the streaming traffic has the highest priority. The combination of traffic considered can be classified in 4 categories:

1. **TCP competing flows**

The three TCP flows have different loads and several starting times; one flow is delayed to the respect of the first ones. The delay of the third flow is varied in order to gain knowledge about the various problems it encounters to start, when two other connections are already transmitting, and to determine the level of performance it manages to achieve. In this test case the DiffServ architecture does not have a strong relevance since we create just one common service class. The main aim of this workload is to compare the performance achieved, especially with regard to the fairness behaviour, when the Tail Drop or the RED discipline (in combination with ECN) is deployed, and further analyze several TCP enhancements.

2. **TCP + UDP**

The main aim in using this workload is to go through several traffic combinations with higher priority UDP traffic and lower priority TCP traffic, in order to reach an advanced analysis of the DiffServ architecture. The DiffServ configuration will be the key aspect to consider in the network configuration as well as the workload features.

3. **HTTP + TCP competing traffic (1 or 2 flows)**

This further combination will consider HTTP traffic of higher priority and one or two TCP flows. This is a preliminary step to the last case.

4. **HTTP + UDP + TCP**

This is the final configuration and the most challenging. Two or three classes of service will be employed. UDP traffic has the highest priority and it is mapped in the EF class. TCP traffic has the lowest priority and it is mapped in the BE class. Finally, HTTP traffic is classified in the AF class.

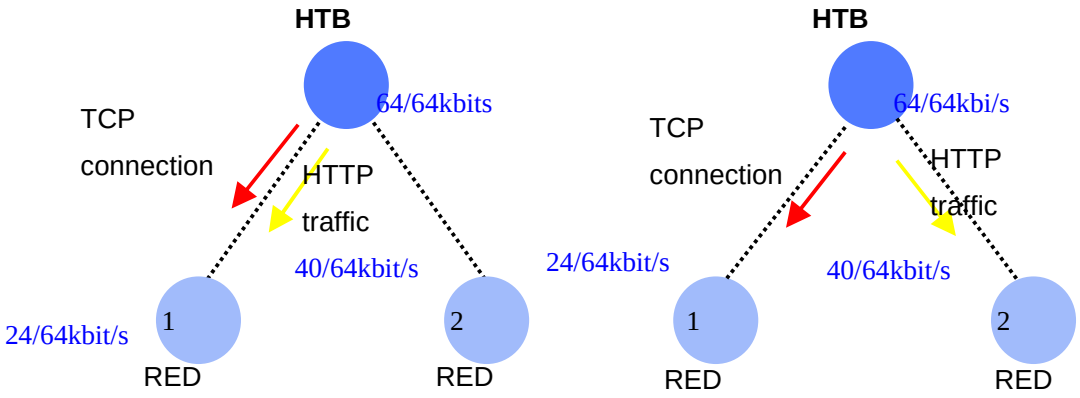

Figure 2. Example of class hierarchy of two HTB configurations for the DiffServ router.

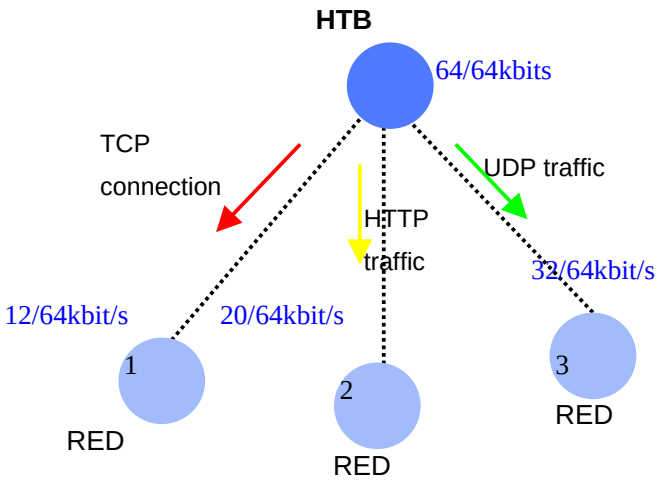

Figure 3. Example of class hierarchy of the HTB configuration of the DiffServ router

Just to give a couple of examples of how the DiffServ router may be configured we consider the third and fourth workload. Figure 2 refers to the third workload consisting of HTTP and TCP traffic. It illustrates how the available bandwidth is shared in a way to give higher priority to the HTTP traffic. Two service classes are employed. We can consider two situations. In the former the TCP and HTTP traffic are served by the same service class, sharing a bandwidth of 64kbit/s. In the latter two service classes are employed and the HTTP traffic has available a larger bandwidth than the TCP traffic. It may be interesting to consider the comparison between these two configurations and study how the performance differs. In Figure 3, we give an example related to the last workload. We consider three classes of service and a possible bandwidth share. The UDP traffic has the highest priority and gets the largest amount of bandwidth. Naturally, it is not always true that the highest priority traffic gets the highest share of bandwidth. Furthermore, we must also consider the amount of data carried by each traffic flow.

In this first phase, only the first two workloads have been studied. The last two are left for further analysis. In the result sections more details are given as regards the employed workload.

6 Metrics

The Seawind emulator logs every action that has been made to each packet. It also reports queue status and other events that help analyzing the results and adds an exact time stamp and packet ID to the events. The remote and mobile hosts collect tcpdump logs of the network traffic. With both the tcpdump logs and the Seawind log we can figure out exactly how each packet was affected during the transmission. To be able to get the applications view of the transfers we have also the log created by the traffic generator. Because our main interest is in the TCP protocol's behaviour, we do not use this log as a main resource, but as a reference to get a deeper understanding of each test run.

For both type of tests we use quite standard metrics, except for some particular tests where additional metrics specific for the type of traffic analysed, TCP, UDP or HTTP, may be considered. In the HTTP tests, we have a number of *classes* that is equal to the total number of connections in a test. Main objects have their own class, which gives us an easy way to compare the characteristics of TCP transfers of the main objects in different tests. Each in-line object is classified by its throughput: the in-line object with the highest throughput within each HTTP transaction is put to the first class and the in-line object with the lowest throughput to the last class. This is repeated for all test runs in a replication set. In the end we have all transfers divided to classes based on their throughput and if needed we can make calculations independently in the classes. All metrics, except the response time and the router queue length, will be collected from the HTTP server to get accurate results. We do collect all information both ways, but include information about the client to server traffic only if it is relevant.
Finally, in some Mixture tests, TCP and UDP connections can be started with different starting times. With such a workload the TCP data transfer may be classified as the *fastest* and *slowest* one, based on the elapsed time required to complete the transfer.

The performance metrics we have used in our evaluation, in order to compare the performance of different experiments include some common metrics, and some metric specific to the traffic.

Common metrics

Elapsed time of a TCP connection is the time needed to complete the entire transfer of a given size, including the time to establish and close the connection. The calculation of the elapsed time is done in the sender's end of the TCP transfer and it is measured as the time from the first TCP SYN segment sent till the receiving ACK for the FIN segment. Since the transferred data are not of the always of the same size, we cannot always compare the elapsed times with each other. For HTTP, the elapsed time is the transmission time of main or inline object. It does not include the request, but only the response from WWW server to WWW client.

Throughput of a TCP transfer is calculated in the sender of a TCP transfer. Measuring it is the most straightforward way to evaluate the TCP performance. The throughput is calculated by dividing the size in bytes of the transferred data with the elapsed time of a TCP connection. Opposite to elapsed times, throughput can be compared for all TCP transfers regardless of the size of the transferred data. For concurrent bulk data TCP transfers of same size throughput is one of the most important metrics. For HTTP tests with variable object sizes throughput gives us extra information to be used in the evaluation of stability of TCP transfers. Since HTTP 1.1 uses persistent connections, the throughput of the transfer of an individual object cannot be compared to the transfer of the same object with HTTP 1.0. Many or all of the

transferred objects are sent in the same TCP connection with HTTP 1.1, while each object was transferred in its own connection in tests with HTTP 1.0. UDP throughput is calculated by dividing the amount of received data with the time between the first and the last received packet.

Number of dropped packets affects directly to the number of TCP retransmissions and has remarkable effect on TCP behaviour. With UDP data flows the number of packet drops are counted, as the packet losses affects the quality of streaming traffic. We present the median drops for both congestion and error related drops as well as the sum of maximum packet drops in connection classes. The sum gives a comparison value of worst cases between test results with a workload. The total number of drops in all repetitions could also be used, but the sum value is calculated from values that actually exist in one particular connection and is not smoothed over all repetitions. Together with the median the sum can be used to conclude the distribution of the drops in connections. Additionally the value can be used despite some of the tests would be run with unequal number of repetitions.

Number of retransmitted packets: it can be due to a packet drop or to an excessive delay and in the second case they are unnecessary retransmissions. The parameter has to be compared to the loss rate one. This is an indication of how aggressive the TCP is.

Number of unnecessary retransmissions is laborious to be counted reliably. We can count the number of retransmitted packets and subtract the number of packet drops from it, but we fail to take in to account the unnecessarily resent packets that are dropped during the retransmission. However, with the help of Seawind logs we can look if a retransmitted packet is dropped and see if the original packet will be delivered, which makes the retransmission of that packet unnecessary. In conventional TCP, retransmission timeouts are sometimes followed by unnecessary retransmitted packets. D-SACK, Linux undo mechanism and F-RTO enhancements may improve the situation notably, which also is being evaluated also in our tests.

Number of retransmission timeouts cannot be measured automatically with our current tools. With tracelook and Seawind we can however study it manually. In our tests there can be quite long periods of time with no activity as seen in the application level due to the link layer retransmissions. These delays can be seen in TCP trace quite easily, and may be followed by retransmission timeouts. Also lost acknowledgments and packets may cause an RTO.

Fairness of TCP and UDP transfers tells how close the metrics of interest of different transfers (e.g., the throughput) inside a test run are to each other and it shows the impact a connection has on a competing traffic sharing the network path. If the performance of one transfer is considerably lower than another's, the fairness is considered low. Especially the fairness of TCP transfers is an important metric because in most of our test cases we have several concurrent TCP transfers. Besides this approximate definition, in some cases

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

Stability of a data flow tells how close the values of a certain metric in separate replications are to each other. If for instance a throughput of one TCP transfer is considerably lower than another's, the stability is considered bad. Stability of transfers is important because in all of our test cases we have several concurrent transfers. It is not automatically calculated, but only discussed.

Router queue length can be observed in any given time by extracting the information from DiffServ router or from Seawind logs. The length of the queue is an indication of the congestion in the last-hop router. Even if the queue is not full causing packets to be dropped, the queuing delays the delivery of the packets and may cause the TCP sender's retransmission timer to expire. The progression of the queue length is observed to see it effects to the TCP sender.

For the most important metrics, such as elapsed time, throughput and number of retransmitted packets, we calculate the first quartile (25% percentile), median (50% percentile), third quartile (75% percentile), and the average regarding all the test replications.

HTTP - specific metrics

Response time of a HTTP transaction is very important, because it can be considered the time that the user experiences when opening a page with a WWW browser. A HTTP transaction begins by requesting a main object and ends when the last in-line object has been transferred. The calculation of the response time is done in the client end of the TCP transfer and starts when the first SYN segment is sent and ends when the server's FIN segment is received. Note that response time does not include the time caused by the transmission of the last ACK segment from client to server that is typically close to a half of a round trip time.

TCP - specific metrics

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

UDP - specific metrics

Jitter tells how much the spacing between two incoming packets has changed compared to the spacing they were sent. The jitter is very significant with interactive streaming data as it tells the delay variation of the packets and generally this kind of traffic requires stable data flows. If a packet is lost and the two received packets are not therefore consecutive, the spacing is ignored. The jitter is expressed in terms of the range (min, max) over the full replications set for each test case.

7 Tests Results

This section presents the results for the test run. We first specify better the network characteristics and TCP variants considered. Then the results are organized in two sections: HTTP tests and Mixture tests.

7.1 Network Characteristics

The experimentation environment is thus characterized. The transmission rate is 64kbps and the MTU is set to 576 bytes. The link send buffer size and the link receive buffer size are fixed to 12000 bytes.

The different wireless link types considered are:

- **e0** is an optimal link. There are no errors and therefore no link layer retransmissions
- **r0** is a lossy link with 2-state Markov model and without ARQ persistency.
- **r2** is a lossy link with 2-state Markov model and with medium ARQ persistency. This means that the link layer tries to retransmit a missing packet at most once. If a packet is dropped, the link layer retransmission is emulated by delaying the packet for about one RTT for each retransmission instead of actually dropping it. The one RTT additional delay is set to 700ms.
- **r4** is a lossy link with 2-state Markov model and with high ARQ persistency. The link layer tries to retransmit a missing packet at most twice.

	e0 (optimal link)	**r0/ r2/ r4**
TCP variants	Baseline TCP IW4+FRTO 1 CBI+FRTO 1 CBI + F-RTO 1 + IW4	Baseline TCP IW4+FRTO 1 CBI+FRTO 1 F-RTO 1 CBI + F-RTO 1 + IW4 DSACK RED

Table 7. TCP variants and network configurations tested

Table 7 describes the TCP variants tested with different link types. With optimal link we used Baseline TCP, TCP with Initial Window of 4 segments, TCP with Control Block Interdependence and a combination that includes CBI, F-RTOv1 and IW4. With other link types we also tested with DSACK and F-RTO v1. RED is employed instead of Tail Drop only in Traffic Mixture Tests. The MTU size was 576 bytes in all tests.

More details about the network configuration for both categories of tests are given in the following sections.

7.2 HTTP tests

This subsection describes the results of the HTTP tests. The subsection is divided to optimal link results, lossy link with no ARQ results, lossy link with medium ARQ results and lossy link with high ARQ results. The tests were run with workload s+2s, m+8m and l+8m with both HTTP 1.1 and HTTP 1.0 protocols. The state lengths are calculated as a uniform distribution whose parameters are described in Table 8.

	s+2s	m+8m	l+8m
Bad state length	1.4 – 3 seconds	1.4 – 3 seconds	1.4 – 3 seconds
Good state length	2.8 – 3.5 seconds	8 – 12 seconds	8 – 12 seconds

Table 8. Lengths of states with each workload

Optimal link results

HTTP 1.1

Table 9 and Table 10 summarize the results of tests with optimal link and http 1.1 protocol. The former describes the results of Baseline TCP and TCP with IW4 and FRTO. The latter describes the results of CBI+FRTO enhanced TCP and TCP with F-RTO, CBI and Initial Window of four packets.

The tables have a column for each workload tested. In each row we have a metric and the corresponding value. *Time* is the total time of a test. *Med_thro, Min_thro* and *Max_thro* are median, minimum and maximum throughputs in test runs. *Drops_c* means congestion related drops, and *Drops*_e error related drops. *Time, Drops_e* and *Drops*_q are given as median values.

Because the TCP transfers were short, the throughputs were not very close to the theoretical maximum throughput of 8000 bps. We can see, however, that with workload *l+8m* the throughput was already more than 6000 bps with all of the TCP variants tested. This results from the persistent connections and pipelining of http 1.1 which makes the transfers closer to bulk TCP transfers.

Due to the optimal link we did not experience any error related drops. With workloads *m+8m* and *l+8m* there was already congestion in the last hop router with Baseline TCP and with IW4+FRTO enhanced TCP. The congestion resulted from slow start overshoots in TCP transfers. When CBI was applied to the TCP, the slow start overshoots were avoided. This did not affect to the performance however, because the transfers started slower than in transfers where CBI was not used. Increasing Initial Window to four segments improved performance of all tests like it did in the earlier HTTP 1.0 tests. Because only one TCP was used, this effect was not as big as it was with HTTP 1.0. The improvement was highest with the smallest workload because the effect of the initial window becomes irrelevant when the amount of transferred data grows.
With workload *s+2s* there was no queue at any time at the last-hop router, and the TCP transfer stayed in slow start for the whole test, which was the case with the HTTP 1.0 tests made earlier also. With workload *m+8m* the behaviour is different to the HTTP 1.0 tests however, since also with this workload the TCP was in the slow start phase the whole TCP transfer. This is also a result from the persistent connections together with pipelining in HTTP 1.1.

OPTIMAL LINK (HTTP 1.1)

Name	BASELINE			IW4+FRTO		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	5.5	15.9	22.9	4.9	20.6	22.3
Med_thro	2034.0	5144.0	6152.0	2301.0	3980.0	6324.0
Min_thro	2032.0	4948.0	6151.0	2290.0	3975.0	6324.0
Max_thro	2152.0	5218.0	6212.0	2415.0	4052.0	6387.0
Drops_c	0.0	7.0	16.0	0.0	9.0	16.0
Drops_e	0.0	0.0	0.0	0.0	0.0	0.0

Table 9. Results of optimal link tests with Baseline TCP and IW4+FRTO

With Baseline TCP and IW4+FRTO the transfers proceeded similarly. In IW4+FRTO tests the last-hop router queue filled up 1.5 seconds earlier after 6.5 seconds resulting in an extra packet drop with workload m+8m. To analyze more of this subject, we also run some tests workload s+8m. These tests indicated that also in that workload congestion was present. With both *m+8m* and *l+8m* the lost segments were

retransmitted using fast recovery. With workload m+8m the transfer completed after the lost packets were retransmitted, but with l+8m the TCP transfer continued in congestion avoidance. No packets were lost after the slow start overshoots.

OPTIMAL LINK (HTTP 1.1)

Name	CBI+FRTO			FRTO+CBI+IW4		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	5.5	15.9	22.9	4.9	20.6	22.3
Med_thro	2033.0	5143.0	6151.0	2300.0	3978.0	6324.0
Min_thro	2030.0	4945.0	6149.0	2288.0	3976.0	6324.0
Max_thro	2121.0	5215.0	6211.0	2401.0	4051.0	6389.0
Drops_c	0.0	7.0	0.0	0.0	9.0	0.0
Drops_e	0.0	0.0	0.0	0.0	0.0	0.0

Table 10. Results of optimal link tests with CBI+FRTO and FRTO+CBI+IW4 enhanced TCP

With *s+2s* the results were practically the same with CBI+FRTO and Baseline TCP, because there were no slow start overshoots in the TCP transfers. With workload *m+8m* CBI did not improve performance, but the slow start overshoot happens in each test run. With workload *l+8m* the TCP transfer in the first test run experienced a slow start overshoot, but with CBI they were avoided in subsequent test runs. In these tests the slow start overshoot happens in different position. In *m+8m* the TCP never reaches congestion avoidance (slow start overshoot happens in the end of the tests) but in *l+8m* it does. In tests with CBI+FRTO the maximum or median throughputs were not improved, but stability of minimum and maximum performances was.

In FRTO+CBI+IW4 tests the results were similar than the previous cases. The throughputs were therefore similar than the results with Initial Window of four segments.

HTTP 1.0

Table 11 and Table 12 describe the results of HTTP 1.0 with four concurrent connections. In these tables the rows are equivalent to the HTTP 1.1 with the following exceptions:

1) The median throughput is not for all connections but for main objects' transfer.
2) The congestion and error related drops are the highest median values of connection groups in test case. Connection groups are created like in the HTTP 1.0 tests in [Saa03].

OPTIMAL LINK (HTTP 1.0)

Name	BASELINE			IW4+FRTO		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	3.9	15.1	22.7	3.5	13.7	22.2
Med_thro	1809.0	2381.0	3288.0	2446.0	2801.0	3009.0
Min_thro	845.0	1406.0	793.0	962.0	1475.0	659.0
Max_thro	1825.0	2391.0	3308.0	2450.0	2805.0	3532.0
Drops_c	0.0	0.0	10.0	0.0	0.0	14.0
Drops_e	0.0	0.0	0.0	0.0	0.0	0.0

Table 11. Results of optimal link tests with Baseline TCP and IW4+FRTO

OPTIMAL LINK (HTTP 1.0)

Name	CBI+FRTO			FRTO+CBI+IW4		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	3.9	15.0	21.2	3.5	13.7	21.0
Med_thro	1810.0	2380.0	3361.0	2443.0	2803.0	3397.0
Min_thro	845.0	1404.0	808.0	962.0	1475.0	834.0
Max_thro	1820.0	2390.0	3391.0	2449.0	2807.0	3430.0
Drops_c	0.0	0.0	0.0	0.0	0.0	0.0
Drops_e	0.0	0.0	0.0	0.0	0.0	0.0

Table 12. Results of optimal link tests with CBI and F-RTO+CBI+IW4

With optimal link, HTTP 1.0 outperforms HTTP 1.1 with workload *s+2s* and *m+8m. With l+8m* the performance was nearly same with HTTP 1.1 and 1.0. With workload *m+8m* there are many congestion related drops with HTTP 1.1, but not with HTTP 1.0. Because there were no congestion related drops with workloads besides *l+8m* with HTTP 1.0, CBI had effect to the performance only with that. CBI improved the median of response times with this workload about 4.5%.

Lossy link without ARQ results

Results of HTTP 1.1 tests with lossy link with no ARQ persistency are described in Table 13, Table 14 and Table 15**Error! Reference source not found.**. Table rows and columns are similar than in previous cases. These results are extremely unstable. Many of the test runs did not complete at all, but ended with RESET when either client or server gave up with the transfer. The HTTP 1.0 plugin was not able to recover from such scenarios: in some cases the server did not exit but continued waiting for connections. This prevented the test from completing. Baseline and DSACK tests are run with only 15 replications due to this. With HTTP 1.1 the plugin was able to handle this situation correctly, so all tests with it are run with 30 replications as specified. However several connections were terminated before all data had been sent successfully.

HTTP 1.1

In these tests the network conditions caused a lot of packet losses. This test case is an extreme case: in some tests the test timed out before all data was sent. With workload s+2s there were 7 error related drops in the median. The median throughput has decreased from 2034 to 282 bps compared to the optimal case tests. The response time was high, almost as high as with workload m+8m, which has a lot larger objects. The network conditions were worse because of different distribution parameters used in these two tests (see Table 8).

Due to packet losses the TCP retransmitted several packets, some of the unnecessarily. Unnecessary retransmissions occurred in the beginning of TCP transfers. For example, in one test the sender first sent two packets to the receiver. The receiver quickly acknowledged the first one. When the second packet arrived, the receiver started to wait for new data before sending another acknowledgment (delayed acknowledgments were in use). The sender transmitted 2 new packets which were lost due to error. Because no acknowledgment for packet two was received, RTO was triggered and the packet was unnecessarily resent. The extremely bad network conditions caused some connections to time out in apache server. This caused the minimum throughout to decrease to 0 with workload *s+2s*.

LOSSY LINK (HTTP 1.1)

Name	BASELINE			IW4+FRTO		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	33.8	33.1	51.3	28.9	30.1	45.9
Med_thro	282.0	2354.0	2626.0	280.0	2607.0	2706.0
Min_thro	0.0	1264.0	1269.0	0.0	694.0	1177.0
Max_thro	920.0	4183.0	3877.0	1206.0	4188.0	4400.0
Drops_c	0.0	0.0	0.0	0.0	0.0	0.0
Drops_e	7.0	26.0	36.0	6.0	26.0	35.0

Table 13. Lossy link with no ARQ persistency – Baseline TCP and IW4+FRTO

Increasing Initial Window to four segments improved the response time with all workloads. CBI did not have effect on performance, because no congestion developed at the last-hop router. The FRTO results were somewhat worse than baseline or CBI results especially with *s+2s*. Studying the traces showed, however, that in many test cases the last packet or the last acknowledgment had been lost, which increased the response time considerably since the TCP had to recover with RTO in those cases.

LOSSY LINK (HTTP 1.1)

Name	CBI+FRTO			FRTO		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	31.7	30.8	49.8	38.7	32.9	49.0
Med_thro	334.0	2606.0	2824.0	276.0	2417.0	2784.0
Min_thro	59.0	1703.0	2169.0	0.0	939.0	1005.0
Max_thro	1097.0	3416.0	3371.0	1152.0	4498.0	4434.0
Drops_c	0.0	0.0	0.0	0.0	0.0	0.0
Drops_e	7.0	12.0	20.0	6.0	26.0	34.0

Table 14. Lossy link no ARQ persistency – CBI+FRTO and F-RTO

DSACK did not improve the performance of TCP in our tests. The response times are in fact decreased notably from baseline test results. This shows how much problems the TCP has with this link type making the variation of the results very unstable. FRTO-CBI-IW4 tests are close to results of IW4 tests. Only IW4 had some effect on performance compared to the baseline, mostly with the shortest workload.

LOSSY LINK (HTTP 1.1)

Name	DSACK			FRTO+CBI+IW4		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	43.5	41.4	63.4	26.9	29.6	48.1
Med_thro	202.0	1901.0	1962.0	302.0	2764.0	2868.0
Min_thro	93.0	612.0	772.0	109.0	1471.0	1734.0
Max_thro	711.0	3873.0	3554.0	1151.0	4299.0	4446.0
Drops_c	0.0	0.0	0.0	0.0	0.0	0.0
Drops_e	7.0	24.0	37.0	7.0	15.0	20.0

Table 15. Lossy link with no ARQ persistency - DSACK and F-RTO+CBI+IW4

HTTP 1.0

HTTP 1.0 outperformed HTTP 1.1 with all workloads and all TCP variants. For example with workload *s+2s* the response time was 43.1% lower with http 1.0 than with http 1.1 in baseline tests. In IW4 tests the improvement was 50.9 %. Because more connections were in use, not as many packet losses occurred to a single connection. This made recovering easier for TCP.

LOSSY LINK (HTTP 1.0)

Name	BASELINE			IW4+FRTO		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	11.1	21.7	45.6	14.2	18.5	37.4
Med_thro	668.0	1685.0	1573.0	441.0	1694.0	1926.0
Min_thro	92.0	363.0	334.0	32.0	142.0	44.0
Max_thro	1013.0	2383.0	2475.0	2446.0	2806.0	3118.0
Drops_c	0.0	0.0	0.0	0.0	0.0	0.0
Drops_e	3.0	4.0	25.0	3.0	4.0	21.0

Table 16. Lossy link with no ARQ persistency – Baseline TCP and IW4+FRTO

In tests with IW4+FRTO the performance was decreased in tests with workload *s+2s* compared to the Baseline TCP. In FRTO+CBI+IW4 tests the response times in tests with *s+2s* and *l*+8m were also higher than in CBI+FRTO tests. This indicates that applying IW4 might decrease the performance. However, the results are highly unstable for extremely bad network conditions. The minimum and 25% quartile of response time are lower with IW4. The medians are close to each other especially with *s+2s* and *m+8m*, but 75% quartile and maximum response time are noticeably higher with IW4, where some test runs have experienced really bad luck with lost packets. Lost acknowledgments together with consecutive lost data packets make the TCP to recover quite often with RTO from multiple packet losses in IW4 tests.

LOSSY LINK (HTTP 1.0)

Name	CBI+FRTO			FRTO		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	21.9	22.4	33.1	13.3	21.6	36.3
Med_thro	336.0	1347.0	2092.0	603.0	1692.0	1961.0
Min_thro	44.0	582.0	563.0	22.0	494.0	296.0
Max_thro	1039.0	2382.0	2519.0	1122.0	2392.0	2635.0
Drops_c	0.0	0.0	0.0	0.0	0.0	0.0
Drops_e	3.0	3.0	13.0	3.0	4.0	21.0

Table 17. Lossy link no ARQ persistency – CBI+FRTO and F-RTO

CBI did not have effect to the results. In general, there was no congestion present during the TCP transfers. FRTO results are close to the baseline results with workloads *m+8m* and *l+8m*, but remarkably better with *s+2s*. In FRTO+CBI+IW4 tests, the results with workload *s+2s* were, however, not as good as with IW4+FRTO or Baseline TCP. The maximum value for response times is 2.8 times higher than with baseline. Also the TCP traces indicate that the difference between the results is not caused by FRTO.

The higher median response time in tests with DSACK was caused by different luck with lost packets. It depicts the unstability between test runs, since DSACK did not have effect on performance. FRTO+CBI+IW4 combination results were similar to IW4+FRTO results.

LOSSY LINK (HTTP 1.0)

Name	DSACK			FRTO+CBI+IW4		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	29.5	23.1	55.1	25.9	20.6	34.9
Med_thro	287.0	1412.0	1340.0	242.0	1429.0	2042.0
Min_thro	22.0	41.0	157.0	29.0	518.0	428.0
Max_thro	1120.0	2386.0	2254.0	2390.0	2806.0	2734.0
Drops_c	0.0	0.0	0.0	0.0	0.0	0.0
Drops_e	5.0	4.0	23.0	3.0	3.0	14.0

Table 18. Lossy link with no ARQ persistency - DSACK and F-RTO+CBI+IW4

Lossy link with medium ARQ persistency results

Results of tests with lossy link with medium ARQ persistency are described in Table 20, Table 21 and Table 21 for HTTP 1.1 and in Table 22, Table 23 and Table 24 for HTTP 1.0. Table rows and columns are similar than in previous cases.

While the effective error rate is reduced notably for medium ARQ persistency, quite many packets are lost for errors in TCP transfers. The median values for error related losses are about one third compared to the tests with no ARQ persistency. This improves the performance and eliminates the timeouts during TCP transfers from the server. The delays caused by retransmissions of TCP packets did not cause RTOs.

Medium ARQ persistency improved the response times of all workloads, in particular *s+2s*. With medium ARQ persistency the response times correspond to the amount of transferred data, which was not the case with out ARQ persistency. All packet losses were due to errors.

HTTP 1.1

The HTTP 1.1 results were not very stable, but varied a lot depending on which packets were lost or delayed. Also the median values differ significantly from each other. Baseline, CBI+FRTO, DSACK and FRTO results should be close to each other, since no congestion or extensive delays causing spurious retransmissions were present.

LOSSY LINK WITH MEDIUM ARQ PERSISTENCY (HTTP 1.1)

Name	BASELINE			IW4+FRTO		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	10.0	18.2	31.6	8.1	17.6	31.5
Med_thro	1045.0	4388.0	4280.0	1371.0	3883.0	4426.0
Min_thro	525.0	1738.0	2638.0	339.0	2050.0	2408.0
Max_thro	1857.0	5361.0	5477.0	2364.0	5607.0	6073.0
Drops_c	0.0	0.0	0.0	0.0	0.0	0.0
Drops_e	2.0	6.0	10.0	2.0	6.0	11.0

Table 19. Lossy link with medium ARQ persistency - Baseline TCP and IW4+FRTO

Increasing Initial Window improved performance in all tests. In tests IW4+FRTO with workload *m+8m*, the minimum throughput was notably lower than with others. The lowest throughput occurred in the second test run. The TCP recovered two times from lost packets using fast recovery, which caused the TCP sender to halve its congestion window. This resulted in relatively long pause in transfer. When the sender's congestion window allowed sending of new packets another bad state had already begun, which caused more packets to be dropped or delayed. This individual test did not affect to the median values.

The second test run was problematic with other workloads also. With *s*+2s the second test run suffered for several errors targeted to the last packets in TCP transfer. The last FIN segment was lost after two link level retransmissions. It was resent when RTO was triggered, but was delayed for 1400 milliseconds because of two attempts of link layer retransmission.

CBI+FRTO results were not as good as baseline results. The response time was lower with all workloads, while the median of dropped packets was nearly the same. FRTO performed well, even though it was not helping the recoveries and CBI was not affecting to the performance. This demonstrates the unstability of the TCP transfers with this link type.

LOSSY LINK WITH MEDIUM ARQ PERSISTENCY (HTTP 1.1)

Name	CBI+FRTO			FRTO		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	11.8	27.5	45.1	10.8	20.0	31.2
Med_thro	951.0	2872.0	3114.0	1024.0	3767.0	4372.0
Min_thro	380.0	1798.0	2290.0	215.0	2062.0	617.0
Max_thro	1546.0	5151.0	5060.0	1827.0	5487.0	5643.0
Drops_c	0.0	0.0	0.0	0.0	0.0	0.0
Drops_e	1.0	5.0	10.0	2.0	7.0	10.0

Table 20. Lossy link with medium ARQ persistency – CBI+FRTO and F-RTO

LOSSY LINK WITH MEDIUM ARQ PERSISTENCY (HTTP 1.1)

Name	DSACK			FRTO+CBI+IW4		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	10.2	18.0	34.3	8.2	27.1	45.1
Med_thro	1055.0	3615.0	3812.0	1265.0	2988.0	3119.0
Min_thro	573.0	1816.0	2476.0	431.0	2129.0	2407.0
Max_thro	1895.0	5437.0	5599.0	1960.0	4301.0	4648.0
Drops_c	0.0	0.0	0.0	0.0	0.0	0.0
Drops_e	2.0	7.0	13.0	1.0	5.0	11.0

Table 21. Lossy link with medium ARQ persistency - DSACK and F-RTO+CBI+IW4

HTTP 1.0

HTTP 1.0 results are more stabile than HTTP 1.1 results. HTTP 1.0 also outperforms HTTP 1.1 in all tests, but the difference is lower than in tests with no ARQ persistency. Dividing the data transfer to several TCP connections allows data to be transferred all the time regardless if one TCP transfer is recovering from packet losses. On the other hand inside a test the stability between different TCP transfers may be very bad. This can be seen also in minimum and maximum throughputs in the tables.

LOSSY LINK WITH MEDIUM ARQ PERSISTENCY (HTTP 1.0)

Name	BASELINE			IW4+FRTO		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	7.6	18.1	28.2	7.4	16.6	25.6
Med_thro	988.0	1684.0	2668.0	1040.0	2150.0	3105.0
Min_thro	145.0	420.0	265.0	66.0	481.0	226.0
Max_thro	1314.0	2390.0	4066.0	2441.0	2808.0	4353.0
Drops_c	0.0	0.0	0.0	0.0	0.0	0.0
Drops_e	1.0	2.0	5.0	1.0	3.0	6.0

Table 22. Lossy link with medium ARQ persistency - Baseline TCP and IW4+FRTO

Though increasing Initial Window theoretically improves the performance in all cases, the actual improvement was quite small. The additional delays caused by link layer retransmissions and the packet losses had more effect on performance. In tests with IW4+FRTO the response times were improved with workloads *m+8m* and *l+8m*. With *s+2s* the response time was equal to the baseline TCP results. With *l+8m* the medium and maximum throughputs were higher, and minimum throughput lower than in baseline TCP tests. This means that the stability between TCP transfers in a test was worse.

CBI+FRTO, FRTO and DSACK results were close to the baseline results. FRTO+CBI+IW4 results were close to the IW4+FRTO results, but the median response times with workloads *m+8m* and *l+8m* were somewhat higher. With *m+8m* they were actually higher than in baseline tests.

LOSSY LINK WITH MEDIUM ARQ PERSISTENCY (HTTP 1.0)

Name	CBI+FRTO			FRTO		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	7.5	20.3	30.2	7.5	18.2	27.7
Med_thro	880.0	1505.0	2384.0	987.0	1553.0	2678.0
Min_thro	142.0	517.0	369.0	146.0	474.0	380.0
Max_thro	1308.0	2299.0	3681.0	1292.0	2385.0	4183.0
Drops_c	0.0	0.0	0.0	0.0	0.0	0.0
Drops_e	1.0	1.0	6.0	1.0	2.0	6.0

Table 23. Lossy link with medium ARQ persistency – CBI+FRTO and F-RTO

LOSSY LINK WITH MEDIUM ARQ PERSISTENCY (HTTP 1.0)

Name	DSACK			FRTO+CBI+IW4		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	11.2	17.8	28.1	7.2	18.5	28.6
Med_thro	900.0	1573.0	2605.0	1157.0	1579.0	2512.0
Min_thro	68.0	278.0	482.0	65.0	406.0	90.0
Max_thro	1269.0	2391.0	4067.0	2446.0	2800.0	4263.0
Drops_c	0.0	0.0	0.0	0.0	0.0	0.0
Drops_e	1.0	1.0	7.0	1.0	2.0	4.0

Table 24. Lossy link with medium ARQ persistency - DSACK and F-RTO+CBI+IW4

Lossy link with high ARQ persistency results

Results of tests with lossy link with medium ARQ persistency are described in Table 25, Table 26 and Table 27 for HTTP 1.1 and in Table 28, Table 29 and Table 30 for HTTP 1.0. Table rows and columns are similar than in previous cases.

In these tests the link has high ARQ persistency and is able to retransmit a lost frame up to four times. This avoids almost all error related packet losses. For example with workload *s+2s* only two packets are dropped for error in the 30 replications. Link level retransmissions add up to 2.8 seconds additional delay to transmissions of packets. This delay is enough to cause retransmission timeouts in TCP.

HTTP 1.1

Compared to medium ARQ persistency results, the results are notably more stable. With workload *s+2s* in baseline tests the median is very close to the average of throughputs. With other workloads the median throughput becomes closer to the maximum throughput.

Like in optimal tests, packets are lost due to congestion with *m+8m* and *l+8m*. Compared to baseline TCP, IW4+FRTO improved performance with the smallest workload, but not with others. This is due to increased congestion in the last-hop router. Similar behaviours were noticed in optimal link test results.

In many test runs with *s*+2s retransmission timeouts occurred due to the link level retransmissions. With Baseline TCP and DSACK this caused unnecessary retransmission of all in-flight packets. With the smallest workloads, however, this meant unnecessary retransmission of only a few packets. With the largest workload the effect was notably higher.

LOSSY LINK WITH HIGH ARQ PERSISTENCY (HTTP 1.1)

Name	BASELINE			IW4+FRTO		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	8.4	20.1	26.9	6.5	23.9	26.3
Med_thro	1331.0	3075.0	5242.0	1693.0	3341.0	5354.0
Min_thro	841.0	2015.0	3726.0	1282.0	2687.0	2732.0
Max_thro	1892.0	5018.0	5675.0	1865.0	5220.0	6172.0
Drops_c	0.0	8.0	16.0	0.0	9.0	16.0
Drops_e	0.0	0.0	0.0	0.0	0.0	0.0

Table 25. Lossy link with high ARQ persistency - Baseline TCP and IW4+FRTO

There were also many congestion related losses with *m+8m* and *l+8m* workloads. Most of the spurious RTOs occurred before the congestion related losses. Most of the error related packet losses were avoided before the fourth link level retransmission. The response time in FRTO tests was higher than with baseline TCP in some tests, especially with workload *m+8m*. However, the minimum and maximum values as well as

the lower and upper quartiles were very close to each other. Inspecting the TCP traces showed, that the difference in performance was due to different timings of dropped packets, not for FRTO or Baseline TCP's behaviours while recovering from RTOs. In most of the test cases FRTO did not improve the response time of the tests, but it improved the stability between different test cases.

With CBI+FRTO the TCP was able to avoid the slow start overshoots after the first test run with *l+8m*. With *m+8m* the results were same than before, however: slow start overshoots were not avoided because the TCP did not reach congestion avoidance. CBI did not improve the median response time with *l+8m*, but made the TCP transfers more stabile.

DSACK did not improve performance. In general, the RTOs occurred after half of the data was already transferred. Since they were seldom unnecessary, the DSACK algorithm did not change the way the TCP recovered. Even if they were unnecessary, all data was still retransmitted like with baseline TCP, and the only benefit is the unadjustment of congestion window. This did not affect to the results.

In the traffic mixtures tests the TCP transfers proceeded nicely. In the first test run with *l*+8m the slow start overshoot caused multiple packet losses like in baseline tests. In the second test run this was avoided for CBI, but smaller slow start overshoot occurred later. This was avoided in the rest of the test runs. FRTO handled spurious retransmissions like in FRTO tests, and IW4 improved the performance in the beginning of TCP transfers. Other packet losses, if any, were generally recovered with fast recovery. The median response times are close to each other with all TCP variants. However, the stability in TCP transfers was best with FRTO+CBI+IW4 combination.

LOSSY LINK WITH HIGH ARQ PERSISTENCY (HTTP 1.1)

Name	CBI+FRTO			FRTO		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	7.9	21.7	27.4	7.9	25.4	26.8
Med_thro	1379.0	3498.0	4989.0	1406.0	3217.0	5235.0
Min_thro	923.0	2460.0	3601.0	1003.0	2520.0	3643.0
Max_thro	1778.0	4978.0	5644.0	1733.0	5014.0	5651.0
Drops_c	0.0	7.0	0.0	0.0	8.0	16.0
Drops_e	0.0	0.0	0.0	0.0	0.0	0.0

Table 26. Lossy link with high ARQ persistency – CBI+FRTO and F-RTO

LOSSY LINK WITH HIGH ARQ PERSISTENCY (HTTP 1.1)

Name	DSACK			FRTO+CBI+IW4		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	8.1	20.5	27.0	7.6	22.0	27.6
Med_thro	1379.0	3718.0	5179.0	1401.0	3708.0	5104.0
Min_thro	880.0	2588.0	2570.0	803.0	2694.0	3213.0
Max_thro	1872.0	4865.0	6009.0	1930.0	5098.0	6190.0
Drops_c	0.0	8.0	16.0	0.0	8.0	0.0
Drops_e	0.0	0.0	0.0	0.0	0.0	0.0

Table 27. Lossy link with high ARQ persistency - DSACK and F-RTO+CBI+IW4

HTTP 1.0

HTTP 1.0 outperformed HTTP 1.1 also with this link configuration. Congestion was present only with the largest workload, but with the help of CBI this was avoided. CBI did not improve the median response time like similarly than in the previous cases, but improved the stability between TCP transfers. In tests with IW4+FRTO the performance was improved with all workloads, but also congestion was increased at the last-hop router. DSACK did not improve the performance. The reasons were similar to those in HTTP 1.1 tests and in [Saa03].

Spurious retransmission timeouts decreased the response times of tests quite lot, even more than in tests with HTTP 1.1. Because there was less data to be transferred in single connection, RTO estimator was smaller in HTTP 1.0 tests than in HTTP 1.1 tests and RTO occurred more often. In some cases, especially with the largest workload, FRTO improved the performance significantly.

The FRTO+CBI+IW4 results were close in performance with the other TCP variants. The stability between test runs was however better with CBI, and performance was improved with IW4. FRTO helped in cases of spurious retransmission timeouts.

LOSSY LINK WITH HIGH ARQ PERSISTENCY (HTTP 1.0)

Name	BASELINE			IW4+FRTO		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	6.8	18.2	29.0	6.2	16.1	26.0
Med_thro	948.0	1429.0	2479.0	1108.0	2215.0	2779.0
Min_thro	368.0	391.0	401.0	364.0	795.0	296.0
Max_thro	1307.0	2390.0	3978.0	2450.0	2806.0	3461.0
Drops_c	0.0	0.0	8.0	0.0	0.0	13.0
Drops_e	0.0	0.0	0.0	0.0	0.0	0.0

Table 28. Lossy link with high ARQ persistency - Baseline TCP and IW4+FRTO

LOSSY LINK WITH HIGH ARQ PERSISTENCY (HTTP 1.0)

Name	CBI+FRTO			FRTO		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	6.5	20.2	26.9	6.6	17.5	27.0
Med_thro	993.0	1482.0	2662.0	1019.0	1453.0	2714.0
Min_thro	362.0	868.0	722.0	255.0	881.0	562.0
Max_thro	1145.0	2378.0	3151.0	1317.0	2385.0	3804.0
Drops_c	0.0	0.0	0.0	0.0	0.0	8.0
Drops_e	0.0	0.0	0.0	0.0	0.0	0.0

Table 29. Lossy link with high ARQ persistency – CBI+FRTO and F-RTO

LOSSY LINK WITH HIGH ARQ PERSISTENCY (HTTP 1.0)

Name	DSACK			FRTO+CBI+IW4		
Workload	s+2s	m+8m	l+8m	s+2s	m+8m	l+8m
Time	6.6	18.2	28.6	6.2	18.1	25.4
Med_thro	1006.0	1655.0	2512.0	1038.0	1491.0	2945.0
Min_thro	357.0	472.0	229.0	370.0	800.0	368.0
Max_thro	1348.0	2386.0	3967.0	2446.0	2808.0	3716.0
Drops_c	0.0	0.0	7.0	0.0	0.0	0.0
Drops_e	0.0	0.0	0.0	0.0	0.0	0.0

Table 30. Lossy link with high ARQ persistency - DSACK and F-RTO+CBI+IW4

7.3 Traffic Mixture Tests

As regards the type of lossy link (r0, r2, r4) of the network configuration in use, the length of the good state is specified by a uniform distribution between 10 and 25 seconds. The length of the bad state is specified by a uniform distribution between 1.8 and 4 seconds and the error probability is 83%.

The traffic workload used in the tests is explained in Table 31. All data flows are directed downlink from the fixed host to the mobile host. The TCP and UDP workloads considered are of four different types:

- 3 simultaneous TCP connections;
- 2 TCP connections start at time 0 seconds and the third start after 5 seconds;
- 2 TCP connections and 1 UDP flow that starts at the same time and ends after the TCP connections have completed; and
- 2 TCP connections and 1 UDP flow that starts after 5 seconds and ends after the TCP connections have completed.

In the cases in which the connections are started at the same time, they should share the bandwidth fairly. In the cases with different starting times, the last starting flow is influenced by the behaviour of the initial connections. On one hand, the last started flow can be negatively affected if the initial connections have already increased their sending rate and gain bandwidth for their transfer (e.g. they are in slow start phase). On the other hand, in some cases the last flow can take advantage from starting later whether the initial connections after an intense transmitting phase have experiences dropped or received duplicated acknowledgment (e.g. congestion avoidance). Thereby, with traffic starting at different times it is possible to observe unfair behaviour and for this reason such study is of more interest when RED is employed. Since we consider only the case of 5 seconds of delay in all tests the third TCP connection or UDP flow starts when the other flows are in slow start phase.

Workload	**File size (bytes)**	**Single flow data sent (kbytes)**	**Total data sent (kbytes)**	**Delay of the 3rd flow**	**HTB Configuration Class of service**	**Class Bandw**	**Link type**	**Link Bandw**
3 x TCP	2096	3 x 209.6	628.8	0 sec	-Without DS 1 class	1 Mbit/s	e0, r0, r2, r4	64kbit/s
3 x TCP 5sec between	2096	3 x 209.6	628.8	5 sec	-Without DS 1 class	1 Mbit/s	e0, r0, r2, r4	64kbit/s
2 x TCP + 1 x UDP	2096 140	2 x 188.64 1 x 720	1097.28	0 sec	-Without DS -With DS: TCP class UDP class	 500kbit/s 500kbit/s	e0, r0, r2, r4	64kbit/s
2 x TCP + 1 x UDP 5sec between	2096 140	2 x 188.64 1 x 720	1097.28	5 sec	With DS: TCP class UDP class	 500kbit/s 500kbit/s	e0, r0, r2, r4	64kbit/s

Table 31. Traffic Mixture Tests Workloads

In the first two cases indicated in the above table, all traffic is TCP and of the same priority. Only one class of service is in use. When the scheduling discipline is Tail Drop, the buffer size is of 22 packets. When RED is employed, the buffer limit is 11000 bytes. The RED parameters follow the guidelines given in Section 2. The parameters not before specified are set as follows:

- the min_{th} is 2000 bytes,
- the max_{th} is 6000 bytes, and
- the avpkt is 500 bytes.

Passing to the third workload, we study both the case with and without DiffServ. With the last type of workload, DiffServ is always employed. The configuration of DiffServ, as explained previously, is based on a HTB configuration with two classes of service: one for higher priority UDP traffic and one for lower priority TCP traffic. The aim in employing such DiffServ configuration was to classify the traffic in distinct queueing class. More complex DiffServ configurations may be considered in the future.

For all types of workload, Baseline TCP, Baseline TCP +RED, and the combination IW4-CBI-FRTO have been studied.

In the following sections, we present a detailed analysis of the results. We first give a general overview on all the tests run by showing some summarizing result tables and graphs. Then the analysis is organized in four subsections, one for each type of link considered. In each subsection, every workload is analysed and the performance achieved by using different TCP Variants are compared.

Overview on the final results

The following tables summarize the results of all Traffic Mixture Tests run. All the tables are organized in the same way. Each table is divided in several parts according to the TCP Variants studied. For each TCP Variant, tests have been run for 4 types of link. The statistics reported depend on the workload considered. Table 32 and Table 33 refer to 3TCP 0s-0s-0s and 3 TCP 0s-0s-5s workloads. Table 34, Table 35, and Table 36 regard 2TCP+1UDP 0s-0s-0s (with and without DiffServ) and 2TCP+1UDP 0s-0s-5s workloads. Additionally, Figure 4 and Figure 5 report a comparison of how the TCP variants behave for different link types.

TCP connections are analysed based on elapsed time (min, max, median, average, percentile 25%, percentile 75%), average throughput, median number of retransmitted packets, median number of congestion related drops, and median number of error related drops. UDP traffic is studied based on median throughput, range min-max of the average jitter, median number of congestion related drops, and median number of error related drops.

3 TCP connections 0s-0s-0s													
				Elapsed Time (sec)						Through (bytes/s)	error drops	conges drops
TCP Var	link type	queue disc.	TCP flow	min	perc25	median	perc75	max	avg	median	medium	median
Baseline	e0	fifo	fastest	53,73	61,56	66,86	66,89	68,88	64,54	3135,00	0.00	8.00
			medium	68,79	74,48	86,58	86,73	86,80	80,34	2421,00	0.00	10.00
			slowest	86,74	86,76	86,79	86,87	87,96	85,27	2415,00	0.00	16.00
	r0	fifo	fastest	61,65	81,61	95,69	99,52	110,38	91,89	2190,50	34.50	7.50
			medium	79,65	104,01	108,98	115,89	131,93	109,73	1923,00	34.00	7.50
			slowest	107,80	112,69	119,19	125,25	162,75	121,17	1758,50	37.00	5.00
	r2	fifo	fastest	50,37	77,99	80,29	85,02	104,35	79,22	2610,50	9.00	8.00
			medium	81,99	88,24	96,07	99,79	110,36	95,07	2039,50	14.50	7.50
			slowest	95,74	101,52	102,77	110,41	117,53	104,44	2181,50	16.50	7.50
	r4	fifo	fastest	42,10	74,04	77,82	85,38	97,90	76,44	2693,50	2.00	10.00
			medium	78,58	89,39	93,40	98,08	108,86	93,91	2244,00	1.50	10.50
			slowest	95,05	96,31	99,16	102,80	112,42	99,82	2113,50	1.50	9.50
baseline+RED	e0	red	fastest	42,10	74,04	77,82	85,38	97,90	76,44	2693,50	0.00	9.50
			medium	78,58	89,39	93,40	98,08	108,86	93,91	2244,00	0.00	11.50
			slowest	95,05	96,31	99,16	102,80	112,42	99,82	2113,50	0.00	13.50
	r0	red	fastest	75,90	90,45	101,56	111,47	123,12	101,55	2063,50	33.00	8.00
			medium	96,25	106,06	110,76	118,27	132,41	111,96	1892,50	33.50	7.50
			slowest	107,21	111,03	117,26	119,40	137,12	116,46	1787,50	35.50	6.50
	r2	red	fastest	66,11	82,04	88,71	94,76	105,58	88,18	2363,50	13.00	9.00
			medium	89,10	94,32	97,05	101,99	109,31	98,62	2160,00	12.50	8.00
			slowest	98,09	101,81	104,67	108,33	122,50	104,52	2002,50	17.00	6.50
	r4	red	fastest	60,13	80,67	90,12	95,95	110,23	87,82	2326,00	1.00	14.00
			medium	89,28	93,57	98,75	103,45	113,37	99,31	2123,00	2.00	12.00
			slowest	99,64	102,48	105,78	108,73	121,53	105,66	1981,50	1.00	12.50
FRTO-CBI-IW4	e0	fifo	fastest	74,22	76,30	76,32	84,40	84,41	79,76	2746,00	0.00	1.50
			medium	82,63	82,65	84,46	86,27	86,31	84,46	2482,50	0.00	1.00
			slowest	86,41	86,43	86,72	86,99	87,03	86,61	2417,00	0.00	1.50
	r0	fifo	fastest	83,77	90,96	97,18	105,95	129,99	100,69	2157,00	37.00	0.00
			medium	89,05	102,39	104,57	109,17	130,74	106,51	2004,50	37.00	0.00
			slowest	101,24	107,89	112,41	118,81	135,66	114,05	1864,50	38.50	0.00
	r2	fifo	fastest	64,01	79,39	84,67	93,38	99,09	84,91	2476,00	13.00	0.00
			medium	84,92	89,16	96,44	97,69	103,80	94,47	2173,00	15.00	0.00
			slowest	90,48	99,49	100,76	103,00	106,52	99,67	2080,00	14.50	0.00
	r4	fifo	fastest	71,35	78,99	82,27	84,50	98,47	82,62	2548,00	0.50	0.50
			medium	78,88	86,63	91,72	95,09	98,99	90,70	2285,50	2.00	0.50
			slowest	90,86	95,26	97,12	100,08	104,93	96,36	2158,50	2.50	4.0

Table 32. Results for 3 TCP connections 0s-0s-0s with different link types and TCP variants in use

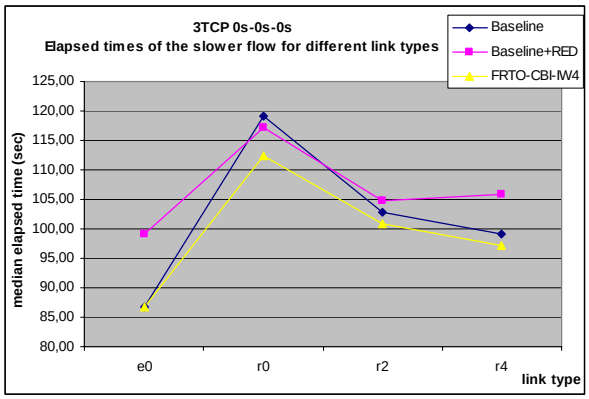

Figure 4. 3TCP 0s-0s-0s, comparison of TCP variants for different link types

3 TCP connections 0s-0s-5s												
				Elapsed Time (sec)						Through (bytes/s)	error drops	conges drops
TCP Var	link type	queue disc.	TCP flow	min	perc25	median	perc75	max	avg	median	median	median
Baseline	e0	fifo	faster	43,00	43,78	57,67	57,67	60,41	55,15	3635,00	0.00	12.00
			slower	70,38	74,11	74,12	79,09	80,31	75,41	2828,00	0.00	15.00
			3rd flow	81,91	81,93	81,93	81,95	82,04	80,87	2558,00	0.00	2.00
	r0	fifo	faster	58,92	75,34	88,59	100,48	131,71	89,44	2367,00	36.00	9.00
			slower	90,28	103,91	112,76	122,47	191,69	114,86	1858,50	36.50	5.00
			3rd flow	52,13	103,74	110,71	122,59	145,71	108,40	1893,50	34.50	2.00
	r2	fifo	faster	53,73	64,77	78,33	91,10	109,80	79,61	2676,50	11.50	9.00
			slower	79,48	92,25	99,28	106,17	120,54	98,56	2111,00	16.00	7.50
			3rd flow	54,69	80,85	93,80	98,27	104,84	87,21	2234,50	12.50	2.00
	r4	fifo	faster	47,86	51,36	55,72	71,27	78,22	61,68	3764,00	1.00	14.50
			slower	74,19	80,05	90,76	105,57	115,54	91,51	2310,50	1.50	13.00
			3rd flow	81,39	90,83	95,50	99,37	104,29	94,84	2195,00	1.00	4.00
baseline+RED	e0	red	faster	49,94	56,55	67,06	73,41	85,74	67,11	3128,00	0.00	14.50
			slower	67,91	74,62	83,73	87,72	90,06	79,90	2504,50	0.00	15.00
			3rd flow	57,75	78,66	82,77	83,85	85,72	79,57	2532,50	0.00	5.00
	r0	red	faster	54,39	83,56	96,25	106,02	124,06	93,71	2178,00	31.50	5.50
			slower	81,01	100,01	111,14	123,18	221,96	117,06	1886,00	32.00	9.00
			3rd flow	41,53	94,32	106,45	113,89	140,31	103,40	1969,00	33.00	3.50
	r2	red	faster	54,23	71,90	85,83	94,53	103,35	83,40	2442,00	13.50	11.00
			slower	85,12	95,81	103,09	105,84	113,34	100,11	2033,00	15.50	10.50
			3rd flow	50,51	81,41	97,79	102,30	111,17	91,91	2144,00	12.00	4.00
	r4	red	faster	56,98	70,74	80,47	90,49	100,18	81,22	2605,50	2.00	15.00
			slower	78,30	93,68	99,15	104,81	116,54	99,07	2114,50	1.00	17.00
			3rd flow	65,59	94,79	100,07	105,74	120,19	98,99	2094,50	2.00	8.00
FRT0-CBI-IW4	e0	fifo	faster	69,45	79,54	79,55	79,56	79,57	79,05	2618,00	0.00	1.00
			slower	80,03	80,05	80,06	80,07	86,43	80,37	2635,00	0.00	1.00
			3rd flow	80,90	81,41	81,41	81,42	81,44	81,39	2575,00	0.00	1.00
	r0	fifo	faster	79,39	87,12	97,84	105,56	132,90	99,86	2142,50	33.00	0.00
			slower	94,92	105,53	109,54	116,40	138,52	111,77	1914,00	36.50	0.00
			3rd flow	91,87	99,28	103,38	112,95	129,03	105,72	2027,50	37.00	0.00
	r2	fifo	faster	72,21	80,07	86,73	93,37	99,26	87,37	2417,00	11.00	0.00
			slower	89,10	95,92	100,75	103,48	115,95	100,22	2080,50	17.00	0.00
			3rd flow	51,48	86,17	91,86	95,32	98,48	88,52	2281,50	13.00	0.00
	r4	fifo	faster	66,21	80,12	85,80	101,20	142,45	88,50	2443,00	1.00	1.00
			slower	80,64	88,40	98,11	110,61	145,99	102,23	2136,50	2.00	2.00
			3rd flow	72,06	84,55	90,04	95,09	162,96	94,66	2327,50	2.00	1.00

Table 33. Results for 3TCP connections 0s-0s-5s; Baseline TCP, Baseline TCP + RED, and FRTO-CBI-IW4 and different link types

2TCP - 1IDP 0s-0s-0s																			
TCP connections															UDP flow				
				Elapsed Time (sec)							Through (bytes/s)	Rext pkts	conges drops	error drop	conges drop	error drop	Through (bytes/s)	Jitter (msec)	
TCP Var	link type	queue disc.	TCP flow	min	perc25	median	perc75	max	avg		median	median	median	median	median	median	median	(min,max)	
Baseline	e0	fifo	fastest	83,93	95,12	104,57	112,42	121,78	103,31		1722,00	13,00	12.00	0.00	0,00	0,00	4000,00	(34,63-34,64)	
			slow est	122,79	122,86	122,91	122,96	123,34	122,74		1535,00	15,00	14.00	0.00					
	r0	fifo	fastest	103,56	127,65	132,99	147,36	158,33	134,65		1314,00	43,00	2.50	34.50	0,00	665,00	3501,00	(34,67-35,91)	
			slow est	142,18	152,28	155,61	163,59	224,38	160,73		1224,50	46,00	1.00	36.50					
	r2	fifo	fastest	81,75	117,64	130,29	136,79	157,49	128,33		1366,50	21,50	9.00	11.50	0,00	174,00	3864,00	(31,59-33,32)	
			slow est	140,92	153,13	156,71	161,44	168,01	157,06		1233,50	26,50	8.50	15.50					
	r4	fifo	fastest	102,52	129,10	140,56	159,68	179,78	143,33		1287,50	15,50	12.00	1.00	0,00	21,50	3972,50	(32,96-34,63)	
			slow est	157,95	171,28	174,81	179,76	187,68	174,03		1091,00	21,00	13.50	3.00					
baseline+RED	e0	red	fastest	93,31	106,53	110,32	114,00	123,80	110,11		1652,00	16,00	15.00	0.00	0,00	0,00	4000,00	(34,63-34,64)	
			slow est	122,87	124,09	125,67	127,46	128,36	124,16		1530,00	17,00	16.00	0.00					
	r0	red	fastest	88,66	125,68	134,83	142,01	166,44	132,84		1368,00	43,50	6.00	34.50	0,00	668,00	3480,00	(34,78-36,41)	
			slow est	144,38	148,91	155,05	163,34	195,80	158,51		1263,00	47,00	8.50	37.00					
	r2	red	fastest	100,76	125,92	137,87	143,79	159,82	135,28		1304,50	25,50	11.00	12.00	0,00	179,50	3861,50	(31,66-33,40	
			slow est	141,90	153,34	155,58	159,89	175,28	156,15		1222,50	27,50	11.00	15.50					
	r4	red	fastest	118,54	141,47	153,72	161,59	181,20	152,69		1165,50	16,50	12.00	2.00	0,00	22,00	3970,00	(32,06-34,57)	
			slow est	166,16	172,50	178,56	181,97	190,84	176,65		1097,00	20,00	15.00	2.50					
FRTO-CBI-IW4	e0	fifo	fastest	80,37	107,99	109,94	119,62	121,92	108,83		1549,50	2,00	1.00	0.00	0,00	0,00	4000,00	(34,63-34,74)	
			slow est	121,95	122,11	122,13	122,20	123,01	121,59		1546,50	3,00	2.00	0.00					
	r0	fifo	fastest	91,07	130,47	133,48	139,37	149,68	131,86		1320,00	42,00	0.00	38.00	0,00	660,50	3500,00	(34,83-35,50)	
			slow est	138,22	144,89	148,53	154,27	206,62	151,38		1337,50	42,50	0.00	37.00					
	r2	fifo	fastest	115,02	125,87	133,69	136,88	144,47	131,35		1377,00	18,00	0.00	13.00	0,00	174,50	3860,00	(31,78-33,27)	
			slow est	142,44	148,68	150,33	155,08	190,50	152,43		1268,00	20,00	0.00	17.00					
	r4	fifo	fastest	124,58	140,38	144,53	154,24	165,32	146,73		1220,50	4,50	1.00	1.00	0,00	22,00	3982,00	(32,54-34,74)	
			slow est	154,78	160,47	165,84	169,28	176,35	163,90		1150,50	6,00	2.50	3.00					

Table 34. Results for 2TCP + 1UDP 0s-0s-0s. DiffServ is in use; Baseline TCP, Baseline+RED, and FRTO-CBI-IW4 and different link types

2TCP - 1IDP 0s-0s-0s without DS																			
TCP connections															UDP flow				
				Elapsed Time (sec)							Through (bytes/s)	Rext pkts	cong drops	error drops	cong drop	error drops	Through (bytes/s)	Jitter (msec)	
TCP Var	link type	queue disc.	TCP flow	min	perc25	median	perc75	max	avg		median	median	median	median	median	median	median	(min,max)	
Baseline	e0	fifo	fastest	76,62	90,05	97,72	105,34	119,42	97,88		1629,50	20,50	20.50	0.00	61,50	0,00	3952,00	(34,67-35,61)	
			slowest	117,39	118,91	119,69	120,71	123,48	119,58		1600,00	23,50	23.50	0.00					
	r0	fifo	fastest	102,69	120,39	126,92	137,06	159,57	127,79		1353,00	46,50	0.00	37.50	0,00	655,00	3472,50	(34,83-35,98)	
			slowest	132,57	143,52	146,66	156,50	309,58	158,22		1326,50	45,50	0.00	38.50					
	r2	fifo	fastest	96,99	129,01	136,12	144,98	162,89	135,86		1321,00	25,50	6.00	15.00	58,50	174,00	3828,00	(31,27-33,93)	
			slowest	133,00	145,05	153,27	156,97	165,16	149,91		1291,00	25,50	5.00	14.00					
	r4	fifo	fastest	98,70	130,03	141,52	167,57	184,78	145,27		1185,50	32,50	29.50	2.00	195,50	20,00	3813,00	(32,80-36,62)	
			slowest	139,79	158,90	176,52	181,20	189,87	170,16		1139,50	35,00	27.00	2.00					
baseline+RED	e0	red	fastest	85,20	96,54	106,36	111,91	115,58	103,89		1640,50	12,00	12.00	0.00	83,00	0,00	3935,00	(34,83-35,44)	
			slowest	116,28	118,00	118,65	119,32	120,35	117,24		1621,50	13,50	13.50	0.00					
	r0	red	fastest	101,37	120,01	128,71	132,60	144,96	125,14		1382,50	42,50	3.50	35.00	28,00	666,00	3441,00	(35,19-36,63)	
			slowest	134,63	139,06	142,85	146,21	211,40	144,55		1361,00	41,50	3.00	34.00					
	r2	red	fastest	90,98	123,32	129,59	140,07	154,41	129,26		1380,00	21,50	4.50	13.50	88,50	172.5	3798,50	(32,39-34,13)	
			slowest	138,06	145,77	150,31	153,27	162,02	147,95		1286,00	25,00	4.50	16.00					
	r4	red	fastest	113,04	139,98	151,07	165,18	183,02	150,80		1154,00	28,00	23.50	2.00	246,00	21,50	3773,00	(33,66-36,02)	
			slowest	152,19	167,01	175,91	182,62	189,27	173,08		1100,50	33,50	25.50	2.00					
FRTO-CBI-IW4	e0	fifo	fastest	88,78	100,21	109,60	113,03	118,79	107,07		1661,00	3,00	3.00	0.00	11,50	0,00	3990,50	(34,49-34,72)	
			slowest	117,68	120,71	121,12	121,44	122,24	120,88		1566,50	4,00	4.00	0.00					
	r0	fifo	fastest	84,91	117,40	130,57	136,19	148,52	126,54		1373,50	43,00	0.00	40.50	0,00	641,50	3511,00	(34,69-35,98)	
			slowest	135,73	139,95	142,97	147,88	399,15	156,18		1350,50	42,50	0.00	37.50					
	r2	fifo	fastest	112,30	126,26	137,41	145,29	151,60	134,33		1309,00	22,00	1.50	14.00	13,00	174,00	3851,50	(31,24-33,34)	
			slowest	137,52	145,26	149,47	152,23	192,13	149,00		1284,50	24,50	1.50	16.50					
	r4	fifo	fastest	104,02	131,22	141,04	151,34	158,71	140,10		1248,50	16,00	12.00	1.50	126,50	20.50	3876,50	(32,54-35,57)	
			slowest	141,48	157,67	161,24	168,92	180,78	161,05		1228,50	19,50	16.00	1.50					

Table 35. Results for 2TCP connections + 1UDP flow 0s-0s-0s. DiffServ is not employed; Baseline TCP, Baseline+RED, and FRTO-CBI-IW4 and different link types

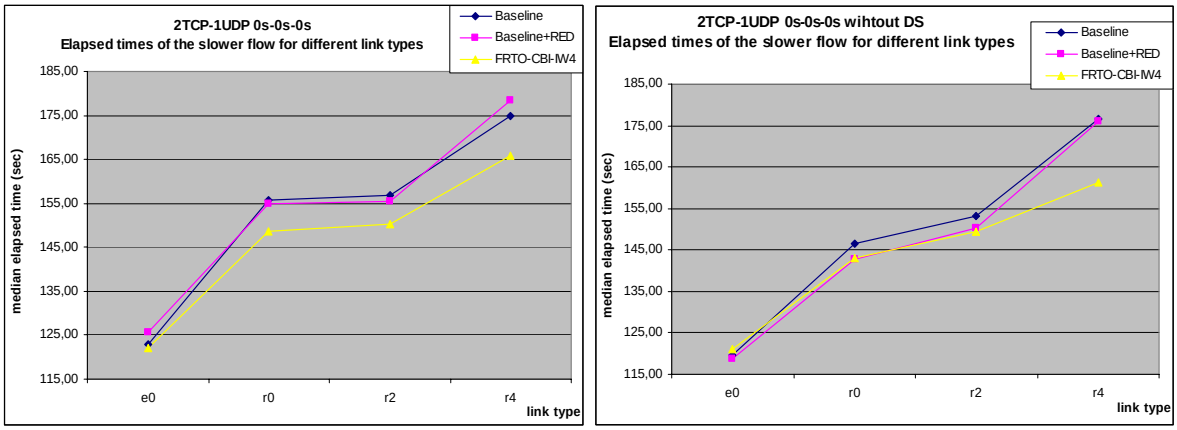

Figure 5. 2TCP-1UDP 0s-0s-0s, comparison of TCP variants for different link types, with (on the left) and without DS (on the right)

2TCP - 1IDP 0s-0s-5s																		
TCP connections														UDP flow				
				Elapsed Time (sec)						Through (bytes/s)	Rext pkts	cong drops	error drops	cong drops	error drops	Through (bytes/s)	Jitter (msec)	
TCP Var	link type	queue disc.	TCP flow	min	perc25	median	perc75	max	avg	median	median	median	median	median	median	median	(min,max)	
Baseline	e0	fifo	fastest	82,53	84,48	98,36	108,35	113,27	97,13	1703,50	10,00	10.00	0.00	0,00	0,00	4000,00	(34,89-34,90)	
			slowest	116,31	116,39	116,46	116,52	117,60	116,17	1621,00	14,00	14.00	0.00					
	r0	fifo	fastest	90,74	111,70	121,31	129,74	146,80	120,78	1414,50	45,00	7.50	36.50	0,00	642,00	3503,50	(35,09-36,57)	
			slowest	141,88	147,54	157,45	176,69	270,08	164,75	1258,50	42,00	5.50	34.50					
	r2	fifo	fastest	103,73	113,87	126,46	136,14	144,86	124,83	1375,00	21,00	8.50	11.50	0,00	181,50	3872,00	(31,48-33,82)	
			slowest	140,22	147,67	149,95	155,24	167,70	150,64	1269,00	26,00	9.00	16.00					
	r4	fifo	fastest	101,07	135,22	142,35	154,59	179,43	143,07	1216,50	16,00	13.50	1.00	0,00	24,50	3988,50	(31,74-34,62)	
			slowest	152,06	158,94	166,90	170,83	188,96	165,90	1179,00	15,50	13.00	2.00					
baseline+RED	e0	red	fastest	83,30	101,69	107,03	111,27	116,63	105,16	1646,00	18,00	18.00	0.00	0,00	0,00	4009,00	(34,98-34,90)	
			slowest	116,62	117,68	120,14	120,81	122,62	119,02	1617,50	19,50	19.50	0.00					
	r0	red	fastest	85,61	120,78	130,73	133,58	144,74	124,76	1414,50	45,00	11.50	34.50	0,00	670,00	3482,00	(35,33-36,14)	
			slowest	136,07	141,71	144,37	150,65	185,12	146,93	1328,50	47,00	7.50	37.00					
	r2	red	fastest	101,05	126,28	131,52	139,81	146,26	129,97	1314,00	24,00	12.00	11.50	0,00	185,50	3866,00	(32,18-33,38)	
			slowest	137,17	144,23	148,76	151,87	163,04	148,52	1313,50	26,00	11.00	16.00					
	r4	red	fastest	118,65	138,05	149,45	159,70	188,97	148,79	1159,00	19,00	15.00	2.00	0,00	21,00	3992,00	(32,65-34,59)	
			slowest	158,88	164,58	170,36	171,48	190,95	168,86	1142,50	24,00	17.00	2.50					
FRTO-CBI-IW4	e0	fifo	fastest	84,30	99,52	101,97	105,07	114,33	101,10	1830,50	1,5	1.50	0.00	0,00	0,00	4016,50	(34,81-35,06)	
			slowest	114,93	115,37	115,40	115,42	117,29	114,82	1635,00	3,50	3.50	0.00					
	r0	fifo	fastest	95,32	121,56	127,26	133,31	146,77	126,45	1376,50	42,00	0.00	37.50	0,00	649,00	3506,50	(35,18-36,36)	
			slowest	133,51	137,54	141,66	145,42	170,33	143,19	1381,50	40,50	0.00	38.00					
	r2	fifo	fastest	107,27	118,57	128,15	136,08	144,47	127,34	1376,50	18,00	0.00	10.50	0,00	185,50	3865,50	(32,09-33,32)	
			slowest	135,77	140,36	144,57	145,89	155,90	144,43	1352,50	19,00	0.00	15.50					
	r4	fifo	fastest	112,59	137,35	144,42	151,14	166,33	143,65	1238,00	4,00	1.00	1.50	0,00	21,50	3994,50	(32,44-34,92)	
			slowest	145,42	153,28	159,01	162,77	168,39	157,57	1246,50	6,00	2.00	3.00					

Table 36. Results for 2TCP -1UDP 0s-0s-5s. DiffServ is in use. Baseline, Baseline+RED, and FRTO-CBI-IW4 and different link types

Optimal link results

The results of the optimal case are analyzed based on the type of workload considered. Four subsections follow.

3 TCP connection 0s-0s-0s and 3 TCP connections 0s-0s-5s

The reference table for the following result analysis is Table 32. Figure 6 shows the elapsed time achieved for several TCP enhancements for slowest, medium, and fastest TCP connection (classified based on the elapsed time).

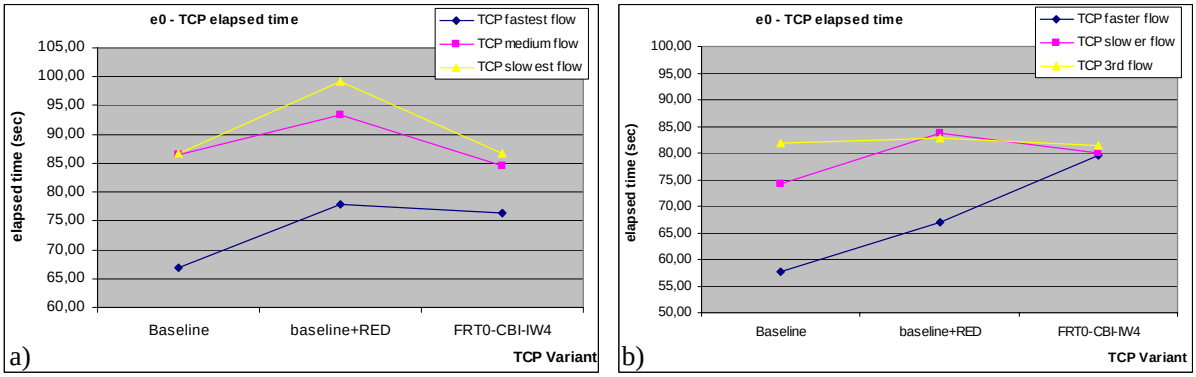

Figure 6. Optimal link - 3 TCP 0s-0s-0s (left side) and 3 TCP 0s-0s-5s (right side) median elapsed time for several TCP enhancements

Among all TCP variants considered, for both workloads the combination FRTO-CBI-IW4 performed better in terms of overall elapsed times, congestion drops, and stability. Note that with 3TCP 0s-0s-0s, the overall elapsed time needed to complete the transfer of all three connections is represented by the elapsed time of the slowest flow, since all TCP flows establish the connection at the same time.

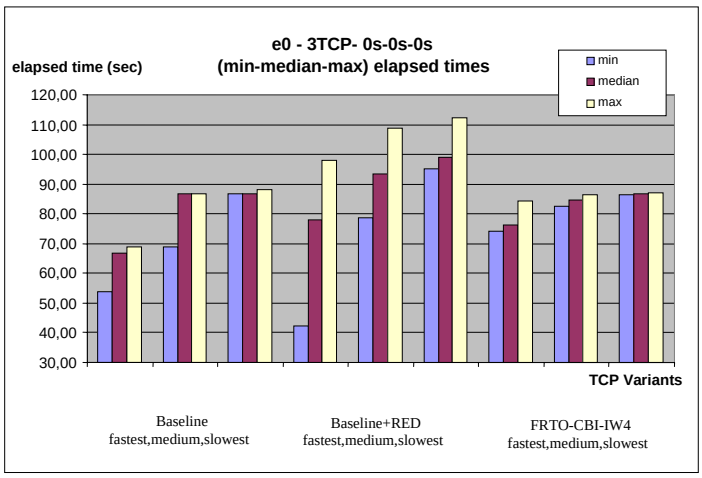

Figure 7. 3TCP 0s-0s-0s - Comparison of (min, median, max) elapsed times for Baseline, Baseline+RED, and FRTO-CBI-IW4

As regards the 3TCP-0s-0s-0s workload, Figure 6a shows that while Baseline and FRTO-CBI-IW4 complete within almost the same time (86.79 sec for Baseline and 86.72 sec for FRTO-CBI-IW4), the performance is quite unsatisfactory when RED is in use (99.16 sec). Furthermore, FRTO-CBI-IW4 guarantees more stability since (min-max) ranges of the elapsed times are reduced, and more fairness since fastest, medium, and slowest flows achieve closer results. Such behaviour for 3TCP 0s-0s-0s is shown in Figure 7.

With Baseline, one of the connections manages to take advantage over the others and completes the transfer even 20 sec before. Usually the reason is due to longer recovery times due to more congestion drops for the other two flows. One or two connections did not manage to recover efficiently from slow start overshoot and transferred data very slowly for a relatively long period of time. Meanwhile, the other connection that recovered efficiently from the overshoot was increasing its transfer rate. As it utilized more and more of the router queue, the queueing delays got longer and the recovery of the slower connections got even slower. Even after the recovery the slower connections still suffered from high queueing delays and managed to start transferring data at normal rate only after the faster connection was finished. In one case at the conclusion of the slow start overshoot the fastest connection managed to start transferring new packets even 20 seconds before the slower flows did and without experiencing any other drops till the end of the transfer.

When RED is employed the results can vary a lot. There are cases in which RED guarantees fairness: connections complete all together, drops are evenly distributed, and non synchronization among the connections occurs, albeit the overall duration of the transfer is anyway larger than without RED. There are cases in which the fastest flow experiences half of the drops of the other flows and naturally completes the transfer before. However, compared to Baseline congestion drops are more evenly distributed among all TCP connections.

Finally, the combination FRTO-CBI-IW4 performed very well. The Slow Start overshoot was totally avoided thanks to the use of CBI and very few congestion drops occurred (1 or 2 for connection). The stability of the results is good.

With 3TCP 0s-0s-5s, the overall duration of the entire TCP traffic is quite the same for all cases. The third flow is in most cases the last one to complete the transfer. However, with FRTO-CBI-IW4 the fairness is extremely good: faster and slower initial flows complete the transfer almost simultaneously and experience the same amount of congestion drops (the median is 1 packet for connection). The third flow usually completes after 5 seconds. Note that in Figure 6b it is reported the elapsed time of the third flow, which means that the actual time of completion is 5 seconds later. However, the results of FRTO-CBI-IW4 with this type of workload were highly stable. For example, the statistics for the elapsed times (see Table 33) show perc25, median, perc75 that differ only for 0.01 sec.

With 3TCP 0s-0s-5s, Baseline did not behave fairly. When the third flow starts, the initial flows are in slow start phase, thus its transfer is at the beginning very slow. Subsequently the initial flows experience slow start overshoot at the same time, but usually one of the initial flow (the faster) recovers much faster than the others and starts to transmit new packets even 25 seconds before than what the other initial flow (the slower) does. The third flow experiences few packets drops since it does not enter in heavy congestion states, but at the same time it usually completes after the slower initial flow.

With this workload, RED performed better than before in the sense that it guaranteed more fairness, especially as regards the third flow that in most cases even managed to complete the transfer before the slower initial flow. This is due to the fact that the initial flows have shorter slow start phase and there are slightly more drops and evenly distributed. This behaviour gives the third flow the possibility to reach better results than with Baseline.

2TCP-1UDP 0s-0s-0s and 2TCP-1UDP 0s-0s-5s

Figure 8 shows statistics concerning median elapsed time of faster and slower TCP connections for 2TCP-1UDP 0s-0s-0s workload (left side) and for 2TCP-1UDP 0s-0s-5s workload (right side).

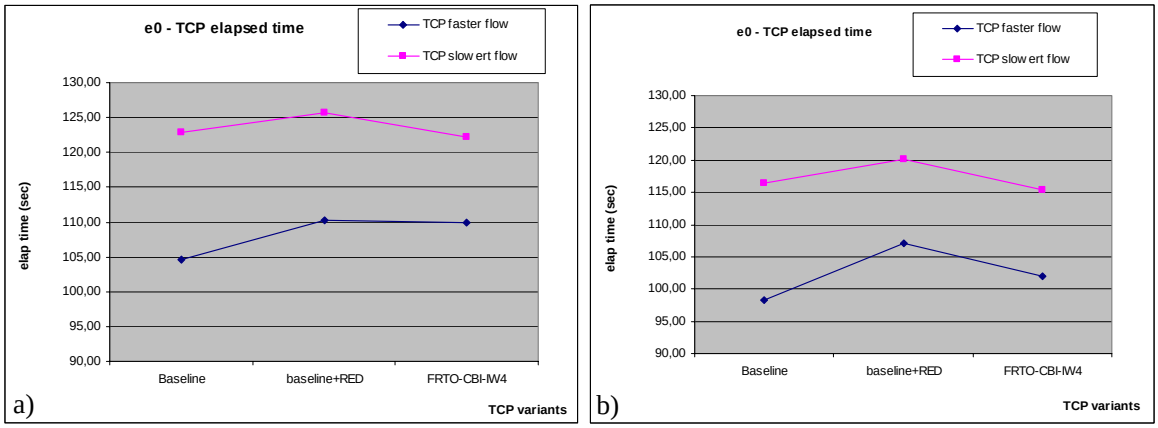

Figure 8. Optimal link - Statistics for 2TCP-1UDP 0s-0s-0s (left side) and for 2TCP-1UDP 0s-0s-5s (right side) with several TCP variants

As regards the higher priority UDP traffic, there are no significant differences in its performance with different TCP variants. As regards the TCP traffic, the best performance is achieved again with FRTO-CBI-IW4, especially for the small number of congestion drops (1 or 2 packets for each connection compared to even 16 packets with RED).

With 2TCP-1UDP 0s-0s-0s, all three TCP variants need almost the same overall time to complete the transfer. For all cases, the same behaviour occurs. The connections need to retransmit the SYN packet to establish the connection because the higher priority UDP flow starts the transmission at the same time and

fills the buffers. It follows slow start overshoot and the difference in performance between TCP flows mostly depends on the duration of the recovery phase. With RED the performance is better than with the two previous workloads. It provides more stability, drops are evenly distributed and shorter recovery times are guaranteed. Also the slow start overshoot is of less entity than with simple Baseline.

With 2TCP-1UDP 0s-0s-5s nothing relevant was observed. The connections are now established without the need of retransmitting the SYN packet. The slower elapsed times compared to the previous workload are due to the fact that the UDP flow starts after 5 seconds and more bandwidth is available for the TCP to start the transfer. Again with RED drops are evenly distributed over the entire transfer thus guaranteeing more fairness than with normal Baseline: the difference between the duration of slower and faster flow is around 18 seconds with Baseline and 13 seconds with Baseline+RED.

2TCP-1UDP 0s-0s-0s without DiffServ

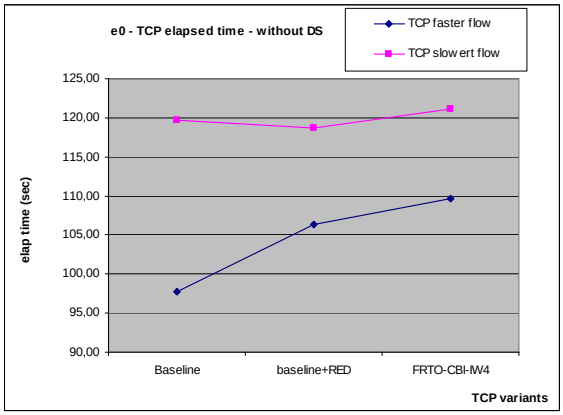

Figure 9. Optimal link - 2TCP-1UDP-0s-0s-0s without DiffServ

Without employing DiffServ (see Figure 9), the TCP elapsed times are slightly reduced to the detriment of the UDP flow that experiences some drops: 61.5 packets are lost with Baseline, 83 packets with Baseline+RED, and 11.5 packets with FRTO-CBI- IW4. TCP drops increase with Baseline and FRTO-IW4-CBI, but decrease with Baseline+RED. The overall performance was again the best with FRTO-IW4-CBI: slow start overshoot was avoided, fewer drops occurred and more stability was guaranteed.

Lossy link without ARQ results

The results analysis is carried on in three subsections, for each type of workload. These results are quite unstable, thus even though most of the graphs are based on median values they should be compared to min, max, average, perc25%, and perc75% values reported in the corresponding tables (Table 32 - Table 36).

3 TCP connection 0s-0s-0s and 3 TCP connections 0s-0s-5s

The statistics for 3TCP 0s-0s-0s and 3TCP 0s-0s-5s are shown in Figure 10. Although with this type of link the median values are not very reliable since for all type of TCP variants the results show a high variance. Each TCP connection experiences in median from 31.5 up to 37 packets lost due to errors on the link.

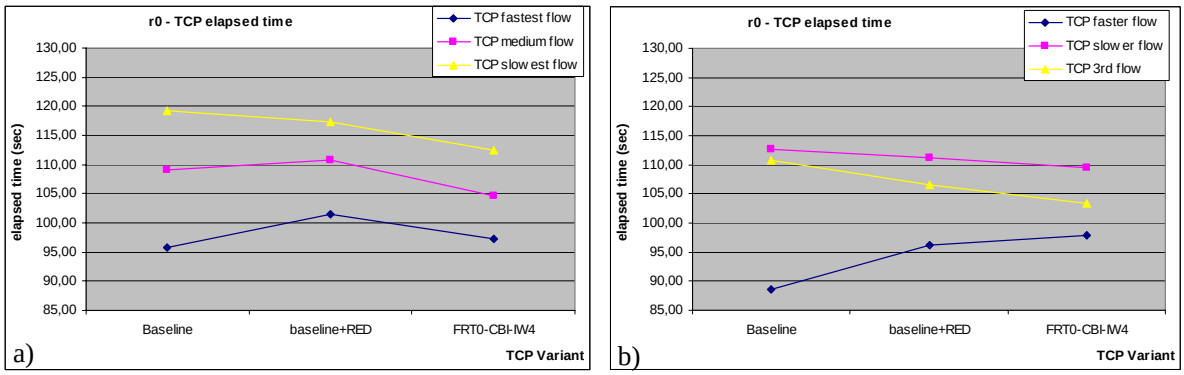

Figure 10. Lossy link r0 - Statistics for 3TCP 0s-0s-0s (left side) and for 3TCP 0s-0s-5s (right side) with several TCP variants

With 3TCP 0s-0s-0s, if we consider the duration of the total transfer, FRTO-CBI-IW4 provides the shortest transfer times. The median elapsed time is 112.4 sec for FRTO-CBI-IW4, 119 sec for Baseline, and 117.26 sec for Baseline+RED. Due to the lossy link, congestion drops are obviously decreased to the respect of the optimal case. Baseline and Baseline+RED had in median from 5 up to 8 congestion drops, whereas none occurred with FRTO-CBI-IW4. Furthermore, due to the error link, the differences among fastest, medium, and slowest flows were attenuated. Finally, note that compared to the optimal case, the global performance with error link is obviously diminished in terms of elapsed times. Elapsed times are generally one third longer.

With 3TCP 0s-0s-5s, we note that the third flow achieves better elapsed times than the slower initial flow. Actually since the third flow starts 5 seconds later, the two connections complete almost at the same time. With Baseline, by looking at the traces we see that the behaviour can vary a lot from case to case. To understand the true reasons of every type of behaviour, each trace should be compared to the distribution of bad and good states employed. There are cases in which the third flow performs very poorly and the initial flows have elapsed times that differ for no more than 10 seconds. In one case the faster flow completed in 83 sec, the slower in 104 sec, and the third flow in 121 sec. From the trace we see that the third flow starts its

transfer very slowly and sees its first transmitted packets lost. The other two flows have long slow start phase and subsequently enters in slow start overshoot. The faster flow is the one that recovers faster and starts to retransmit new packets 31 seconds before than what the slower one does. Such an unfair behaviour is similar to what notice in the optimal case with the same workload. Indeed, the distribution of bad and good states for this case starts with a good state of 18.28 sec followed by a bad state of 3.50 sec. If we take another case where the initial good state is much shorter (7.41sec) the performance of the third flow is completely different. The third flow completes the transfer in 71sec (almost half of before), the faster flow completes in 111sec and the slower one in 113 sec. This behaviour is quite obvious and confirms the fact that more error drops there are, especially in the beginning of the transfer and more the three flows achieve similar results.

As regards 3TCP 0s-0s-5s, the same observations can be applied when RED is in use. However, with RED the unfairness is slightly reduced due to random congestion drops, slow start phases are shorter for all flows, and drops are evenly distributed. In many cases the third flow completes before than the slower flow. With FRTO-CBI-IW4, there are no congestion drops and more fairness in the way the three connections share the bandwidth, albeit the variance of the results is still very high.

2TCP-1UDP 0s-0s-0s and 2TCP-1UDP 0s-0s-5s

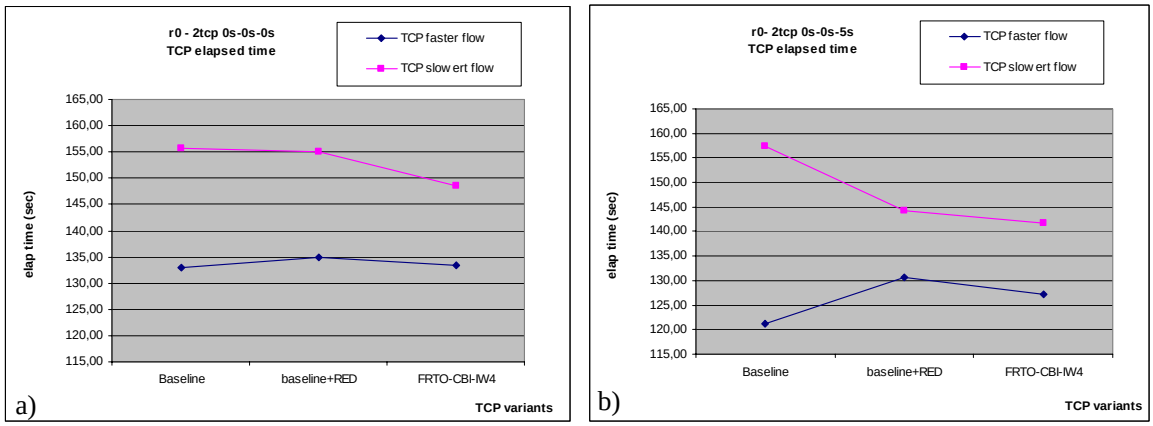

Figure 11. Lossy link r0 - Statistics for 2TCP-1UDP 0s-0s-0s (left side) and for 2TCP-1UDP 0s-0s-5s (right side) with several TCP variants

As regards the workload 2TCP-1UDP 0s-0s-0s (see Figure 11a), with Baseline and Baseline+RED the TCP flows achieved the same elapsed times, experienced the same amount of retransmitted packets (from 43 to 47 packets), but RED caused more congestion drops. The UDP traffic performs in both cases in the same way: the median of error drops is 665 packets for Baseline and 668 packets for Baseline+RED. FRTO-CBI-IW4 achieved better results. The median of UDP drops is 660.5 packets. The overall duration of the TCP transfer is shorter of 7.08 seconds to the respect of Baseline. However, like with the previous workload the variation range (min,max) of the elapsed times was very large. Both TCP connections are established only

after the SYN has been retransmitted one or two times. Moreover, with all TCP variants there was one case out of the 20 replications in which the TCP performance was extremely poor. With Baseline in one case a TCP flow completed the transfer in 224.38 seconds. For two times a packet lost was retransmitted three times thus causing the first time an idle period of about 82 seconds and the second time an idle period of 130 seconds. With FRTO-CBI-IW4, in one case the TCP completed in 206.62 seconds. Both TCP connections had to retransmit two times the SYN packets to establish the connection and one connection experienced an idle period of 50 seconds for a lost packet.

With 2TCP-1UDP 0s-0s-5s (see Figure 11b), RED performed much better than before. On the contrary, Baseline performed very poorly by achieving the same median elapsed times of 2TCP-1UDP 0s-0s-0s. The results of FRTO-CBI-IW4 were slightly better than the ones of Baseline+RED. With Baseline there was high unfairness. In most of cases, one of the TCP flow (the slower) transmits more than half of its traffic after the other connection has completed and experiences long idle periods of even 128 seconds. The difference between median values of the elapsed times of faster and slower flows is 36.41 seconds for Baseline, 13.64 seconds for Baseline+RED, and 14.4 seconds for FRTO-CBI-IW4. With FRTO-CBI-IW4, no slow-start overshoots and no long recovery times occurred. Furthermore, note that the worst overall TCP elapsed time with Baseline was 270.08 seconds, with Baseline+RED was 185.12 seconds, and with FRTO-CBI-IW4 was 170.33 seconds. Compared to the case 2TCP-1UDP 0s-0s-0s, the UDP performance is the same with the only difference that the minimum jitter is lightly increased.

2TCP-1UDP 0s-0s-0s without DiffServ

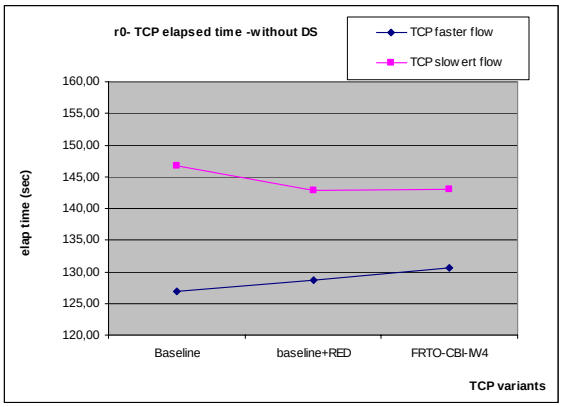

Figure 12. Lossy link - 2TCP-1UDP-0s-0s-0s without DiffServ

Even though DiffServ was not in used the performance of the UDP traffic was not much affected. Only when RED was in use there were some congestion drops in the router (the median is 28 packets). The median UDP throughput decreased of 0.81% with Baseline and 1.12% with Baseline+RED, but it increases of 0.31% with FRTO-CBI-IW4. If we compare Figure 12 to Figure 11, the median elapsed times show a light improvement in performance for the TCP traffic even though the UDP traffic performs in the same

manner. The explanation is rather obvious. For example, comparing the traces of the Baseline case with and without DiffServ, we see that the main difference, which mostly affects the slightly better performance of the case without DiffServ, is the starting phase of the TCP connections. Without DiffServ the TCP connections can compete for an unavailable bandwidth that is the double so that they can establish the connection very quickly. With DiffServ the connections usually need to retransmit the SYN packet at least once and at least five seconds more are needed to establish the connection.

Lossy link with medium ARQ persistency results

In this section, we introduce the results for lossy link and medium ARQ persistency. The link layer tried twice to retransmit the lost packet, which introduced a delay up to 1.4 sec, and dropped the packet if after the delay it was lost again. The number of error related drops is obviously reduced to the respect of the previous case, but still we have many error drops. The analysis for each type of workload follows.

3 TCP connection 0s-0s-0s and 3 TCP connections 0s-0s-5s

The results for the workload 3TCP 0s-0s-0s and 3TCP 0s-0s-5s are shown in Figure 13a and Figure 13b respectively. The number of error related losses is roughly reduced from 26% up to 48% (considering the median values). In the case of 3TCP 0s-0s-0s, the overall elapsed time to complete the transfer of all TCP traffic is quite the same for all TCP variants. The Baseline is the case where the performance is mostly ameliorated by introducing ARQ persistency. The median overall elapsed time achieved with r0 is decreased of about 10-15%. However, slightly better performance is achieved with FRTO-CBI-IW4. Some unnecessary retransmissions occurred with Baseline and Baseline+RED, but no more than one for each connection. The number of congestion drops increase to the respect of the case with r0 except in the case of FRTO-CBI-IW4 where none occurred. Furthermore, the variation ranges (min, max) are much reduced with r2 to the respect of r0. With Baseline the average variation range of the elapsed times is reduced of about 35-45%.

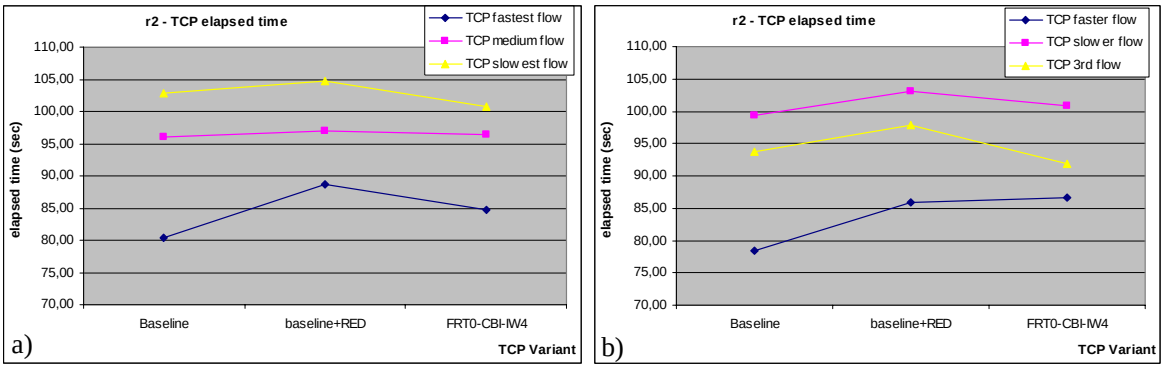

Figure 13. Lossy link with medium ARQ persistency, r2 - 3 TCP connections 0s-0s-0s (left side) and 3 TCP connection 0s-0s-5s (right side)

With 3TCP 0s-0s-5s, it is worthwhile to note the TCP flow started after 5 seconds achieve better elapsed times than one of the initial flow, even though the difference among their elapsed times is in many cases roughly about 5 seconds, which means that they often complete at the same time. The overall duration of the transfer is rather the same for all TCP variants. To the respect of the r0 case with this workload, the median number of congestion drops increase with RED and with the Baseline only as regards the slower flow. None congestion drops were observed with FRTO-CBI-IW4.

2TCP-1UDP 0s-0s-0s and 2TCP-1UDP 0s-0s-5s

As regards the TCP performance, the results for 2TCP-1UDP 0s-0s-0s and 2TCP-1UDP 0s-0s-5s show similar trends, as illustrated by the graphs in Figure 14. Obviously the elapsed times for 2TCP-1UDP 0s-0s-5s are reduced since the TCP traffic has the full link bandwidth available in the first 5 seconds of transmission before the UDP starts. Likewise, the results for the UDP traffic are similar.

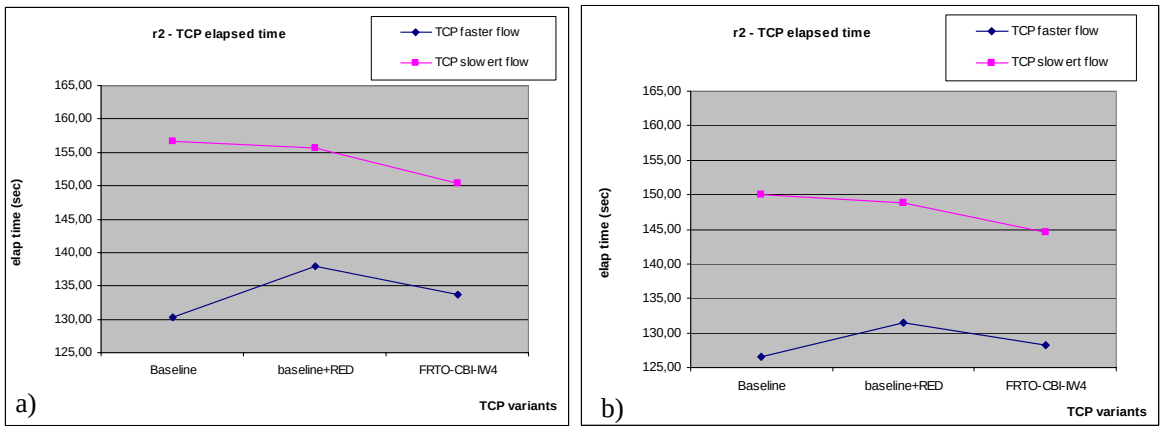

Figure 14. Lossy link r2, Statistics for 2TCP-1UDP 0s-0s-0s (on the left) and 2TCP-1UDP 0s-0s-5s (on the right) with several TCP variants

To the respect of r0 link case, these results are much more stable, as shown by the variation range (min,max) that are largely reduced. Furthermore, the max elapsed time is quite reduced. With 2TCP-1UDP 0s-0s-0s, the worse elapsed time occurred with FRTO-CBI-IW4 and was of 190.5 sec, much lower than what observed with r0 (224.38 sec). With 2TCP-1UDP 0s-0s-5s the maximum elapsed time was 167.7 sec with Baseline (with r0 it was 270.08 sec). The UDP jitter is reduced of about 3 msec and the UDP throughput is increased of 10%. TCP error drops are reduced for both workloads to one third, thus to have median values varying from 10.50 up to 17 packets. UDP error drops are also reduced to one third or even one fourth. TCP congestion drops increase with Baseline in 2TCP-1UDP 0s-0s-0s by passing from 2.5 and 1 for faster and slower flow respectively to 9 and 8.5 and in 2TCP-1UDP 0s-0s-5s by passing from 7.5 and 5.5 to 8.5 and 9. With Baseline+RED, the increase is less: with 2TCP-1UDP 0s-0s-0s, TCP congestion drops pass from 6 and 8.5 for faster and slower flows respectively to 11 for both flows; with 2TCP-1UDP 0s-0s-5s, TCP congestion drops pass from 11.5 and 7.5 for faster and slower flows respectively to 12 and 11. As regards

TCP elapsed times, due to the delays introduced by medium ARQ, in some cases the median TCP elapsed times slightly increase or decrease.

As already noted in r0 with 2TCP-1UDP 0s-0s-0s, the TCP connections need to retransmit the SYN packet before establishing the connection. With 2TCP-1UDP 0s-0s-5s the connection is established without any SYN retransmitted.

2TCP-1UDP 0s-0s-0s without DiffServ

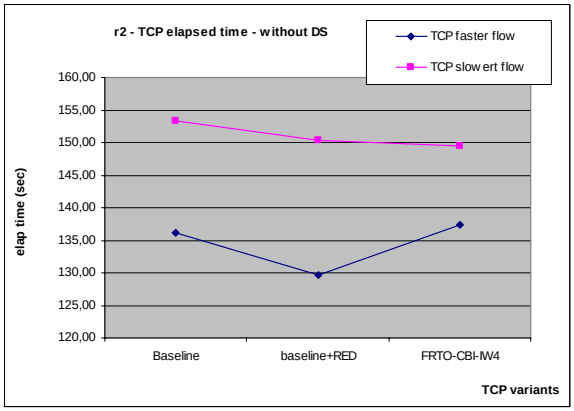

Figure 15. Lossy link r2 - 2TCP-1UDP 0s-0s-0s without DiffServ

The statistics for 2TCP-1UDP 0s-0s-0s when DiffServ is not employed are shown in Figure 15. Also these results are more stable compared to the r0 case. The median number of UDP error drops is about 174 packets (with r0 it was about 654 packets). The congestion drops increase to the respect of r0 case: 58.5 for Baseline, 28 for Baseline+RED, and 13 for FRTO-CBI-IW4. Likewise, also the TCP elapsed times increase to the respect of r0 case. If we compare this case to the correspondent case with DiffServ, the performance of UDP and TCP is only slightly affected. The overall duration of the TCP transfer is a little shorter. However, the best performance was still with FRTO-CBI-IW4.

Lossy link with high ARQ persistency results

Finally, we discuss the results of the tests run over lossy link with high ARQ persistency. In these tests the link has high ARQ persistency and is able to retransmit a lost frame up to four times. This avoids almost all error related packet losses. In the tests only a median of 0.5 up to 2 packets were dropped due to error. The link level retransmissions add up to 2.8 seconds additional delay to transmissions of packets.

3 TCP connection 0s-0s-0s and 3 TCP connections 0s-0s-5s

Figure 16 presents the results of each TCP variant when all TCP connections start concurrently and when one of them starts 5 seconds later than the others. Having up to four link level retransmissions increased

congestion at the last-hop router. Compared to the r2 case, the increase on the median number of congestion drops varies from 30% up to 85%, thus to reach higher values than in the optimal case. With FRTO-CBI-IW4 where the median congestion drops was 0 with r2, it passes to 0.5 up to 2 for each TCP connection with r4. As shown by Figure 4, in 3TCP 0s-0s-0s with FRTO-CBI-IW4 and Baseline+RED, the median elapsed time of the slower flow decreases compared to the r2 case. With Baseline it slightly increases. Instead with 3TCP 0s-0s-5s the overall duration is reduced in all cases.

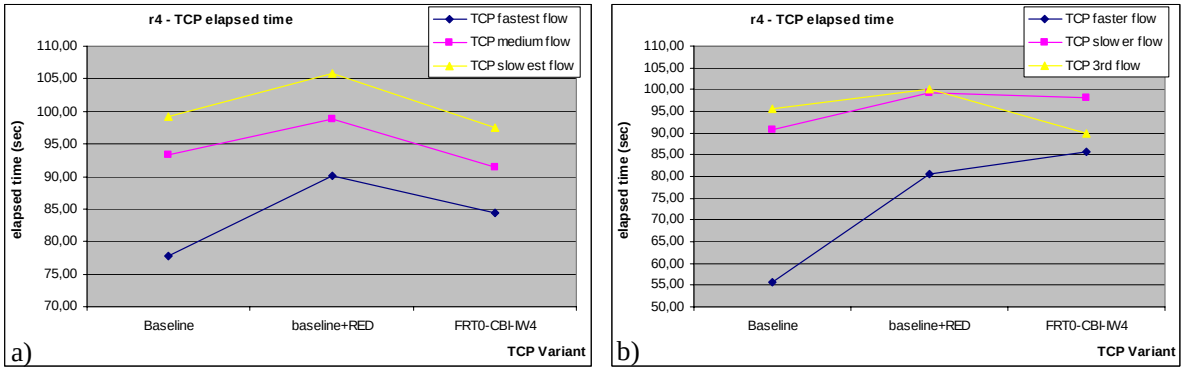

Figure 16. Lossy link with high ARQ persistency - 3 TCP 0s-0s-0s and 3TCP 0s-0s-5s, median elapsed time for different TCP variants

RED performed very poorly with both workloads and achieved the longest elapsed times. With 3TCP 0s-0s-0s, FRTO-CBI-IW4 and Baseline provided closer elapsed times, but with 3TCP 0s-0s-5s FRTO-CBI-IW4 performed much better. Indeed, in 3TCP 0s-0s-0s tests the elapsed times of the three flows were already close to each other, but with 0s-0s-5s the benefit was even higher, even though the (min, max) range was even 6 times larger. Increasing initial window improved the performance only a bit, since the highest benefit of it is in the beginning of a TCP transfer. Since there were only a few error related losses and several congestion related losses, CBI improved the performance quite remarkably. As regards FRTO, some spurious retransmission timeouts occurred in the Baseline tests. For example, in one test run with Baseline TCP the TCP unnecessarily retransmitted 17 packets after a spurious RTO. This took more than 3 seconds of time. In 0s-0s-5s tests, the stability was not good when Baseline TCP was used. This was caused by the high number of congestion related losses the slower flow experienced.

2TCP-1UDP 0s-0s-0s and 2TCP-1UDP 0s-0s-5s

Figure 17 depicts the results of 2TCP-1UDP tests when all flows start concurrently, and when the UDP flow starts 5 seconds later than the TCP flows. In the tests the results are similar to the results with three TCP flows. Applying RED to the Baseline TCP generally decreases the performance. The flows are not stable in the Baseline tests or in Baseline+RED tests. The stability between TCP flows in tests is higher in the FRTO-CBI-IW4 tests, in which the elapsed time of the lowest flow is also better than in the others.

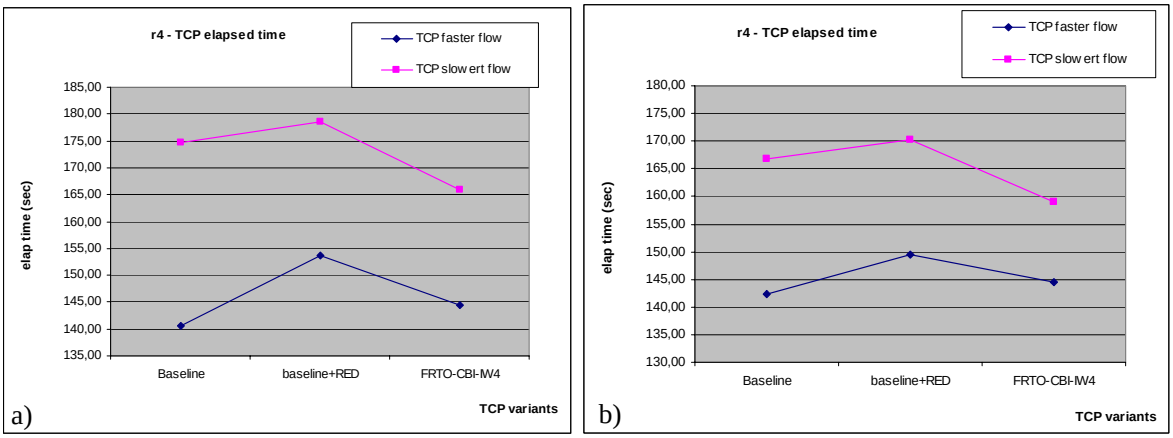

Figure 17. Lossy link r4, Statistics for 2TCP-1UDP 0s-0s-0s and 2TCP-1UDP 0s-0s-5s with several TCP variants

2TCP-1UDP 0s-0s-0s without DiffServ

Results of 2TCP-1UDP 0s-0s-0s without DiffServ tests are depicted in Figure 18. To the respect of r2 case the median number of error related drops were reduced of at least 8 times: from 172.5-174 packets to 20-21.5 packets. The median number of congestion related drops is 7 times higher and with FRTO-CBI-IW4 even 10 times. Disabling DiffServ increased the instability of the two TCP flows when using Baseline TCP. For example, the (min,max) range is (98.7,184.78) for the faster flow and (139.79,189.87) for the slower flow. This was because the UDP flow now competed with the TCP flows from the same link bandwidth, and unlike the TCP flows did not slow down if congestion occurred. The stability slightly improved when RED was applied. As shown in Figure 5, by comparing with the other link types, in this case FRTO-CBI-IW4 provided much lower elapsed times than the other TCP variants.

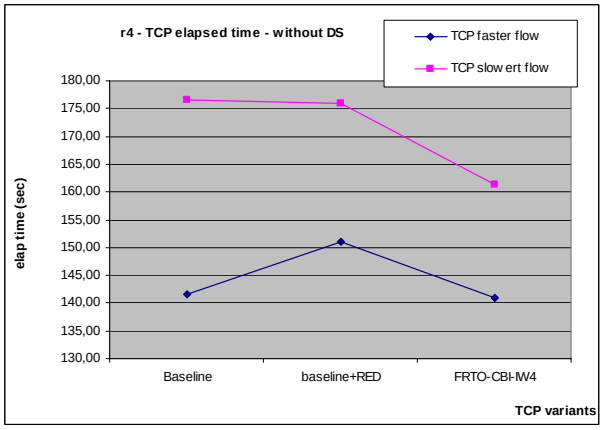

Figure 18. Lossy link r4, Statistics for 2TCP-1UDP 0s-0s-0s without DiffServ

In tests with TCP with FRTO, CBI and IW4 the results were better than with the other variants also with this case. The number of congestion related losses in the UDP flow was decreased, the elapsed time of the slowest flow was better and the stability was higher in these tests than with the other variants.

8 Concluding Remarks and Future Work

In HTTP tests, the HTTP 1.0 protocol outperformed HTTP 1.1 protocol. There were two main reasons for this. Firstly, the total amount of transferred octets was too small for TCP to make full use of the bandwidth available. Only with workload *l+8m* the TCP reached congestion avoidance. With this workload the performance was also more close to each other with the protocol versions. Secondly, HTTP 1.1 used only one concurrent connection even though two concurrent connections could have been used. With the small transfers divided to multiple objects HTTP 1.0 got advantage from its four concurrent connections.

Baseline TCP performed quite well in most scenarios. However, when relatively long delays were present, it suffered from spurious retransmission timeouts. Increasing initial window to four segments generally improved the performance. This is quite clear since doubling the initial window allowed the TCP to save at least one RTT when transferring the smallest objects. In tests with four link level retransmissions it also increased congestion, which neglected the benefits described above since more lost packets had to be resent.

CBI affected more in HTTP 1.1 tests than in HTTP 1.0 tests. It allowed the TCP to avoid slow-start overshoots with workload *l+8m*, but also slowed the TCP transfer down, which made the response time sometimes lower than in baseline results. When both congestion and error related packets were present, CBI seemed to lower the performance. Stability was generally improved, when CBI was in use. In tests with HTTP 1.1 CBI was not able to prevent slow-start overshoots with workload *m+8m*, but they occurred in all 30 replication sets. This was because the TCP never reached congestion avoidance phase.

In tests with CBI+FRTO+IW4 the results were typically caused by one or two of the three enhancements. In many tests, the results were close to IW4 results, for example. Best results with the combined TCP variants were with HTTP 1.0 using four times retransmitting link. The spurious retransmission timeouts caused by extensive delays and packet losses caused by congestion caused scenarios in which TCP had benefit from CBI, FRTO and IW4 all together.

In the traffic mixture tests, Baseline TCP performed poorly with most of the workloads. The behaviour was highly unfair mainly due to serious slow start overshoots and long recovery phases. Many unnecessary retransmissions have been observed with the r4 link. RED did not provide fairness in all workloads and generally helped more when both UDP and TCP traffic were considered. However, it guaranteed evenly distributed congestion drops and less serious slow start overshoots than what was observed with Baseline. The combination of IW4, CBI, and FRTO provided the best results in all cases and with every type of link. The fairness was high, stability good and the overall duration of the transfer usually shorter than with Baseline or Baseline+RED.

The Future work will concern modifications of several aspects of the network configuration and workload types. As regards the TCP Baseline considered, TCP Timestamps will be separated from Linux Eifel algorithm and added to TCP Baseline. This will make the Baseline TCP to correspond to the latest recommendations on TCP implementation over wireless links. Additional tests will be run keeping the same workloads, but with higher bandwidth and considering different starting times among concurrent traffics. In particular cases in which TCP or UDP traffic is started when other TCP connections are in congestion avoidance or recovery phase (and not only Slow Start phase) will be analysed. Furthermore, two additional workloads for which no tests have been run in this first phase will be studied: concurrent HTTP and TCP transfers, and UDP, HTTP and TCP transfer with prioritized DiffServ class. These workloads have already been described in the first part of the document. The DiffServ configuration will be further extended to provide a more complex classification of different priority traffics.

9 References

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

[FIE99] Fielding R. Gettys J., Mogul J., Frystyk H. Masinter L., Leach P. and Bainers-Lee T. *Hypertext Transfer Protocol -- HTTP/1.1*. RFC 2616, June 1999.

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

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

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

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

[JBB92] Jacobson V., Braden R., and Borman D., *TCP Extensions for High Performance*, RFC 1323, May 1992.

[KGMS01] Kojo M., Gurtov A., Manner J., Sarolahti P., Alanko T., and Raatikainen K., Seawind: a Wireless Network Emulator. In proceeding of 11th GI/ITG Conference on Meaurement, Modelling and Analysis (MMB 2001), pages 151-166, Sept. 2001.

[Kulve03] Kulve T., *Analysis of concurrent TCP and streaming traffic over a wireless link*. University of Helsinki, Department of Computer Science. Series of Publications C, No. C-2003-. October 2003.

[LM03] Ludwig R., and Meyer M., *The Eifel Detection Algorithm for TCP*, RFC 3522, April 2003.

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

[Riva03] Riva O., *Analysis of Internet Transport Service Performance with Active Queue Management in a QoS-enabled network,* Politecnico di Milano, April 2003.

[Saa03] Saarto J., *WWW traffic performance in wireless environment*. University of Helsinki, Department of Computer Science. Series of Publications C, No. C-2003-35. May 2003.

[SKR02] Sarolahti P., Kojo M., and Raatikainen K., F-RTO: An Enhanced Recovery Algorithm for TCP Retransmission Timeouts, ACM Computer Communication Review, Vol 33, No 2, 2003.

[Touch97] Touch J., *TCP Control Block Interdependence*, RFC 2140, April 1997.

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