Analysis of Heterogeneous IP Traffic in Wireless Environment

Ilpo Järvinen

Helsinki February 2006

Master's Thesis

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta/Osasto — Fakultet/Sektion — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Ilpo Järvinen

Työn nimi — Arbetets titel — Title

Analysis of Heterogeneous IP Traffic in Wireless Environment

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
M.Sc. Thesis	February 2006	124 p. + Appx.

Tiivistelmä — Referat — Abstract

Wireless access is expected to play a crucial role in the future of the Internet. The demands of the wireless environment are not always compatible with the assumptions that were made on the era of the wired links. At the same time, new services that take advantage of the advances in many areas of technology are invented. These services include delivery of mass media like television and radio, Internet phone calls, and video conferencing. The network must be able to deliver these services with acceptable performance and quality to the end user.

This thesis presents an experimental study to measure the performance of bulk data TCP transfers, streaming audio flows, and HTTP transfers which compete the limited bandwidth of the GPRS/UMTS-like wireless link. The wireless link characteristics are modeled with a wireless network emulator. We analyze how different competing workload types behave with regular TPC and how the active queue management, the Differentiated services (DiffServ), and a combination of TCP enhancements affect the performance and the quality of service. We test on four link types including an error-free link and the links with different Automatic Repeat reQuest (ARQ) persistency.

The analysis consists of comparing the resulting performance in different configurations based on defined metrics. We observed that DiffServ and Random Early Detection (RED) with Explicit Congestion Notification (ECN) are useful, and in some conditions necessary, for quality of service and fairness because a long queuing delay and congestion related packet losses cause problems without DiffServ and RED. However, we observed situations, where there is still room for significant improvements if the link-level is aware of the quality of service. Only very error-prone link diminishes the benefits to nil. The combination of TCP enhancements improves performance. These include initial window of four, Control Block Interdependence (CBI) and Forward RTO recovery (F-RTO). The initial window of four helps a later starting TCP flow to start faster but generates congestion under some conditions. CBI prevents slow-start overshoot and balances slow start in the presence of error drops, and F-RTO reduces unnecessary retransmissions successfully.

ACM Computing Classification System (CCS):
C.2.2 (Network Protocols)
C.4 (Performance of Systems)

Avainsanat — Nyckelord — Keywords
Performance, streaming, HTTP, TCP enhancements, competing flows, RED, DiffServ

Säilytyspaikka — Förvaringsställe — Where deposited
Kumpula Science Library, serial number C-2006-13

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction

From the early days, the Internet with very limited number of users and hosts has evolved into heterogeneous and complex combination of world-widely networked hosts available for massive number of users. Wide variety of services has been deployed. Many of them have requirements for the transmission of the data in the network. Some of these requirements are contradicting with each other. Therefore it is necessary to classify the traffic somehow and to treat different classes according to their preferences when competing traffic classes are present.

An on-going trend in the Internet expansion is a wireless access. The access devices include laptops and hand-held devices such as personal digital assistants or mobile phones having wireless LAN connections or using General Packet Radio Service (GPRS) [BW97, ETS05] or Universal Mobile Telecommunications System (UMTS) [ETS05, KAL+05] (or their counterparts) as an access method. This shift towards mobile access is a great challenge for the existing Internet protocols that were designed in the era of wired links. The origin of the Transmission Control Protocol (TCP) [Pos81b] dates back multiple decades. Without multiple amendments TCP is not able to efficiently meet the ever increasing requirements that arise from a vast diversity of the communication mediums and methods. One very important protocol built on the top of TCP is the Hypertext Transfer Protocol (HTTP) [BLFF96] that backbones the World Wide Web (WWW). The World Wide Web was and still is a key application in popularizing the Internet for a large number of end users. Therefore the performance of HTTP is vital.

In the meantime, the increase in the network bandwidth and in the computing power of the end-user devices together with the development of powerful compression algorithms has opened a way for video and audio streaming services. Both of these require a continuous data supply from the streaming source. To provide this, a certain amount of bandwidth must be available with a precedence for the streaming traffic whenever a stream is transmitted. Otherwise it is hard for the network to provide end-to-end delay that is low and stable enough for the delay sensitive streaming traffic.

Whenever more than one transfer is present in the network, a harmful interference between the transfers may occur. In a lightly loaded network the interference might not be significant, but when the network is congested, the interference can ruin the performance of the transfer, displeasing the end user. A congested router must decide the order in which the packets are served and which packets are discarded. Because of the induced losses and delays, the queuing at the intermediate routers that may extend even up to a congestion is a serious threat for many applications, including multiple streaming applications.

Traditionally the routers have used a single queue, which drops packets only when the queue is full. This behavior is known as tail-drop. To provide a better control over the traffic, new methods for controlling traffic in the routers are emerging. These include active queue management [BCC+98] and quality-of-service techniques.

The active queue management tries to proactively prevent congestion from occurring. Regardless of the congestion, the quality-of-service techniques prioritize packets at routers, which is required to fulfill quality of service demands of the delay sensitive traffic. Especially to provide a low delay, prioritization must be performed even with moderate queue lengths.

In this thesis, we measure and analyze interaction in traffic mixtures. The traffic mixtures include bulk data TCP transfers, audio streaming traffic and HTTP transfers. Different workloads are transferred over a wireless link with multiple repetitions to generate a statistical view of the behavior. The wireless link configuration resembles a GPRS/UMTS-type wireless link. In the measurements real-time emulator Seawind [KGM+01] is used to model the characteristics of the wireless link and to control the workloads. Between the wireless link and the Internet is a last-hop router, which is subject to congestion by the transferred traffic. Each workload is tested with multiple queue management and quality of service configurations. The last-hop router employs Random Early Detection (RED) [FJ93] with Explicit Congestion Notification (ECN) [RFB01, Flo94], Differentiated services (DiffServ) [BBC+98], and their combination. The results with these configurations are compared to the best-effort packet treatment with tail-drop queuing policy. The end host TCP implementation has two alternatives. Baseline TCP is compared with Enhanced TCP that includes combination of selected TCP enhancements that are standardized or documented by the Internet Engineering Task Force (IETF).

The rest of this thesis is organized as follows. In Section 2 we discuss the characteristics of wireless environment and challenges that Internet protocols encounter in environment with a wireless link. In Section 3 we discuss solutions that are used to overcome the problems. In Section 4 we describe our arrangements for the measurements and define the metrics that are used in the analysis of the results. In Section 5 we report the results of the performed measurements and provide a deep analysis of the measured performance. Finally, in Section 6 we conclude our work.

2 Internet Traffic in Wireless Environments

2.1 Wireless Link Characteristics

There are two main categories of the wireless links. A Wireless Local Area Network (WLAN) covers a compact area, e.g., a building, with data rates above 10 Mbit/s and latencies of a few milliseconds. Another category are Wireless Wide Area Networks (WWAN), which are divided into satellite and terrestrial networks. This study covers only the terrestrial WWAN such as General Packet Radio Service (GPRS) or Universal Mobile Telecommunications System (UMTS) [ETS05]. The area of coverage in the GPRS/UMTS cellular network extends to square kilometers, and the data rate is lower than in the WLAN environments.

Communication from a mobile host to a wired host in the Internet is presented

in Figure 1. The wireless environment which provides Internet access, in spite of many differences in the wireless technologies, has some basic building blocks. The mobile host is communicating with a stationary peer over a wireless link. For now on we refer to the peer as a base station (term used in the GPRS/UMTS cellular networks). A terrestrial WWAN is constructed from many base stations. A mobile host is connected to a single base station at a time. From the Internet side, a last-hop router is the last node at the edge of WWAN. It is technology dependent how firmly the base station and the last-hop router are bound together, nevertheless that pair serves as a gateway for Internet traffic of the mobile host. In a typical environment, between the pair is a Radio Access Network (RAN).

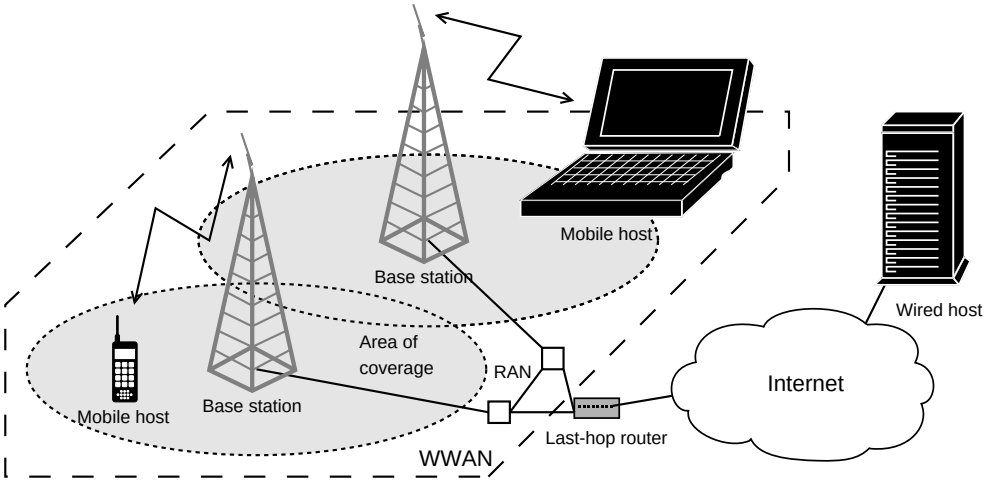


Figure 1: Wireless clients connected to the Internet

The wireless link has usually very different characteristics from the wired counterparts. A large share of the wireless links has a relatively low bandwidth, a high bit-error rate, and a long propagation delay. Therefore long round-trip times are experienced due to the propagation delay and a increase in the transmission delay because of a serialization of the bits to the lower bandwidth. In addition, with wireless access, variability of the network conditions is high because users are moving [IML+03].

Because the bit-error rates are several magnitudes higher than on the wired link, it is no longer true that the packet losses happen only due to congestion. Two different kinds of sources for the high bit-error rate can be identified. A blackout occurs when the mobile device moves to a location that is not within the coverage area of the radio waves, and a corruption of the frame occurs because of a disruption of the transmitted radio wave. Many blackouts are temporary lasting only for a short duration because the receiver moves past a blocking obstacle (e.g., a tunnel, a building, or a blocking terrain). Similarly, the probability of the corruption depends on the quality of the radio wave at the current position of the receiver. Because the quality does not fluctuate rapidly between the frames at the current position, corruption frequently occurs in a burst for many frames when the receiver is in a

place with a poor quality. An Automated Repeat reQuest (ARQ) [FW02] may be used by the link level to trade off the large bit-error rate for a longer delay through retransmissions.

When a mobile host moves, it needs to switch between base stations. Switching between the base stations is called a hand-off. Hand-off is necessary when the quality of the radio wave becomes poor and there is another base station that could provide a better quality in range. The hand-off introduces delay spikes because time-consuming operations must be performed before the transmission can continue through the new base station. Furthermore, because of the mobility, the number of users connected to a single base station varies, which may affect a bandwidth share of a single user.

2.2 TCP Traffic

The Transmission Control Protocol (TCP) [Pos81b] with which a large share of Internet communication is performed has been used as a base protocol in the Internet communication for ages. On the top of TCP multiple upper-level protocols with associated services have been developed. These include an interactive, low bandwidth terminal access (e.g., Telnet [PR83]), bulk data transfers (e.g., FTP [PR85]), and the popular World Wide Web (WWW) with the associated HTTP protocol [BLFF96]. Those protocols rely on the flow and congestion control and the reliability of TCP.

The congestion control is very important for the stability of the Internet because it is a counter-measure to a dreaded congestion collapse [Nag84]. TCP uses slow start and congestion avoidance algorithms to maintain the stability [Jac88, APS99]. Key thought behind the congestion control is to limit the number of unacknowledged segments a TCP flow is permitted to have inflight simultaneously. For this purpose, each TCP flow maintains a congestion window. The slow start probes the available bandwidth of the transmission path by doubling the congestion window on each round-trip, which prevents injecting a large, sudden burst of packets into the network. A transition from the slow start to the congestion avoidance occurs when the current congestion window is larger than a slow-start threshold (*ssthresh*), which is initially set to a large value. We refer to the first slow start that is performed by a flow as *an initial slow start*. The congestion avoidance linearly increments the congestion window per round-trip. TCP detects lost segments through three duplicate acknowledgements [APS99] and Retransmission Timeouts (RTO) [PA00]. Whenever a loss is detected, the sender performs a loss recovery. In the loss recovery, the congestion window is reduced as a response to the detected congestion and the slow-start threshold is adjusted according to the discovered bandwidth. As these congestion control algorithms were designed on the era of wired links, the early TCP specifications have multiple shortcomings when used over the wireless link because TCP makes assumptions that are not valid with the wireless links. A leading assumption is that all losses indicate congestion. Besides that, many other phenomena occur that are not present in the wired environment.

In a wireless environment, the sender may falsely consider all losses as an indication of a congestion [DMK+01]. Losses, however, occur frequently on a wireless link without congestion. In such a case, the retransmission should be performed because the segment was lost due to corruption, but at the same time, the network could accept more packets because no congestion occurred. With information available to the sender, it cannot know that the congestion window should not be reduced. Besides, the specifications mandate reduction of the congestion window after a loss is detected [APS99]. In general, a congestion window that is too small causes sub-optimal performance. Even worse, a loss that occurs during the initial slow start near the beginning of a connection prevents the rapid increase of the congestion window because the loss causes a transition to the congestion avoidance, which increases the congestion window linearly. If multiple flows are transferred simultaneously, the affected congestion window becomes unfairly small because of the loss while other flows can open congestion windows to unfairly large sizes.

A small congestion window may cause domino effect because an efficient initiation of the TCP's loss recovery depends on the number of duplicate ACKs. When the sender receives less than three duplicate ACKs, it does not activate the loss recovery through them. The number of generated duplicate ACKs depends on the number of packets-in-flight that reach the receiver. When the congestion window is small and some of the packets are lost due to corruption, the receiver generates very few duplicate ACKs that can also suffer same kind of losses. Consequently, not enough duplicate ACKs always reach the sender, which is not yet able to decide whether the duplicate ACKs are caused by a loss or a reordering in the network. Because the sender is unsure, it does not initiate the loss recovery. To resolve this situation, the sender must wait for RTO. When RTO is triggered, the congestion window is reset to one.

In a WWAN environment, as the traffic is going from a fast wired network to a slow wireless link the last-hop router buffers are filled by the incoming packets that await transmission. The last-hop router buffers become almost full for long periods of time, which is a full-queue problem [BCC+98, DMKM01]. If the bottleneck is at the last-hop router, the initial slow start eventually fulfills the last-hop router buffer no matter how large the buffer is unless the transfer ends first or something else prevents the slow start from continuing (e.g., the receiver window limits the congestion window, a packet is lost due to corruption, etc.). It takes approximately a round trip until the sender detects the congestion through three duplicate ACKs. During this period, the slow start still increases the congestion window exponentially even through the path is already congested. We refer to this period of congestion as *slow-start overshoot* and to the loss recovery following the initial slow-start overshoot as *overshoot recovery*. Also when TCP uses congestion avoidance after the slow start, the length of the buffer must again reach the maximum length before any congestion drops can occur. With congestion avoidance the length of the buffer grows slowly towards the maximum, and therefore the queue is almost full for a long period. For TCP performance, it is necessary that the buffering capacity at the bottleneck is at least twice the size of the end-to-end Delay-Bandwidth Product (DBP) because then

the sender halves the congestion window close to the actual DBP in the loss recovery. Since end-to-end round-trip time, in general, is not known, the correct sizing of the buffer can only be estimated using the DBP of the link as a reference. Because it is likely that the wireless link is the bottleneck of the transfer, the last-hop router should have this buffer for the traffic to keep the link fully utilized.

When competing flows are transferred, the severity of the full-queue problem becomes more intense because a flow that starts when the queue is nearly full is penalized. The later starting flow cannot open its congestion window freely but encounters immediately congestion because the other flows occupy all buffer space. This phenomenon is called a lock-out [BCC+98]. For example, when a user downloads a large file on the background while browsing the Internet, many HTTP objects are typically sent simultaneously in a single Web page request due to images. Because many HTTP objects perform slow start at the same time, the last-hop router becomes quickly congested. The HTTP objects switch to congestion avoidance with a small congestion window and therefore complete slowly. These delays are very easily noticed.

Several TCP variants exist. TCP Reno [APS99] performs badly if multiple losses hit a single TCP window. Such losses delay TCP's loss recovery by multiple round-trips as RTO is almost always necessary. With a wireless link, this problem is more frequent than with a wired link due to the error bursts. When a loss is detected, the sender halves the congestion window and retransmits the first missing segment. Arriving ACKs tell the sender only the missing segment, which is the first in a sequence, at a time. The sender does not know which segments above the first missing segment were delivered and proceeds to transmit new data if the congestion window allows it. For the next lost segment, three new duplicate ACKs indicating the lost segment must arrive before it is retransmitted but very like RTO is triggered earlier because RTO is not reset during the recovery to a later point.

TCP NewReno [FH99] was designed to handle multiple losses within a single window without RTO. NewReno introduces a concept of partial ACK that acknowledges less than to the highest sequence number sent before the entry to the loss recovery. If the sender receives a partial ACK in the loss recovery, NewReno retransmits the segment indicated by the partial ACK immediately. For the next-sequenced partial ACK to arrive, the acknowledgement of the retransmission must arrive, which takes one round-trip. Hence, if multiple losses occurred, the partial ACKs indicate the lost segments, one per round-trip, which is NewReno's rate of recovery.

Selective Acknowledgements (SACK) [MMFR96, BAFW03] enable a receiver to report to the sender detailed information of the segments it has received instead of the bare duplicate ACK. With SACK, the sender knows which segments do not require retransmission and can guess with a high accuracy the lost segments from the holes in the SACK information. Therefore the recovery can proceed with a faster rate than NewReno's one lost segment per round-trip time. The SACK-capability is already widely deployed on the Internet hosts [All00].

The congestion control algorithms cannot respond well to bandwidth variations

[IML+03], which are common in wireless environment. A transfer rate is limited by the periods of a lower bandwidth and by the growth rate of the congestion window. Besides the initial slow start, the sender probes for a larger congestion window only in the congestion avoidance because the other slow starts are limited by the slow-start threshold. Therefore the growth rate is constrained by the congestion avoidance that increases the congestion window only by one segment per a round-trip. This problem is not limited just to the bandwidth variations of the wireless link but whenever the congestion window and the slow-start threshold are set to a smaller value, the restoration of the larger congestion window takes a long time because of the long round-trip time.

Because of the long round-trip time, opening a new TCP connection is time-consuming [DMK+01]. The slow start requires at least one round-trip to double the size of the congestion window. Short connections spend most or all their time in the slow start probing the available bandwidth, but they are not able to ever reach an optimal level. Therefore it would be useful to use the status information of an earlier connection if such a connection is available because in it the probing of the bandwidth has already been done. This is not an universal solution because slow start might also be required after an idle period longer than the current retransmission timeout [APS99, HPF00] but can improve many cases where the new transfer can begin immediately when the earlier completes. Not all protocols take advantage of the earlier connections, for example, HTTP/1.0 opens a new connection for every HTTP object. In addition, when there is competition, the opening of a new connection might not succeed fairly because of the lock-out.

A sudden long delay that may occur in wireless environment triggers the retransmission timer spuriously, which causes unnecessary retransmissions [IML+03]. On RTO, the sender is forced as congestion control measures to reset the congestion window to one segment, to discard the gathered SACK information, and to halve the slow-start threshold. In addition, the sender performs the slow start, which typically retransmits all segments unnecessarily because they are already delivered after the delay. If the retransmission timer would have been configured to a larger timeout value, the unnecessary retransmissions and the congestion control measures would have been avoided completely. However, the sender cannot enlarge the RTO timer at will without at the same time delaying the loss recovery when a loss has indeed happened and less than three duplicate ACKs arrive to the sender.

2.3 Streaming Traffic

An emerging Internet traffic type is streaming traffic. Typically these streams contain multimedia, like streaming audio, that is presented to a human viewer. The viewer expects a fluent presentation of the media. For the stream, series of mathematical operations are performed by a codec to shrink the size in bytes without human noticeable flaws [WHZ+01] because an uncompressed stream is immensely large for transporting over many of the Internet-access links. The streaming services can be categorized into real-time and archived streams. Both categories require a

delivery of a certain bit-rate, but the time-scale of this requirement is different between the categories. Transmission across the Internet is subject to many factors that cause variability among packets (e.g., queuing delay, fragmentation, different routes, etc.). Variations in the bit-rate can be expressed with an end-to-end delay and a jitter. The jitter measures the variations of the end-to-end delay.

The archived streams are easier for the network to cope with as the receiver can buffer data tens of seconds before presenting it to the user. This buffer can absorb short-term variations of the delivered bit-rate as long as each packets of a stream is delivered before the payload of the packet is passed on from the receiver buffer to the codec. Therefore the receiver has a window during which it can re-request the missing parts of the stream. The upper-bound for the end-to-end delay is quite large and the jitter is almost irrelevant. Typical real-time radio or television broadcasts also fall into this category because no interactivity is required and buffering tens of seconds is not an issue for the user (many will not even know). However, in a case where the user demands a full-blast live presentation without any delay, also this type of streaming qualifies with real-time stream requirements.

The real-time streams include services like video conferencing or voice phone calls, which require bidirectional interactivity. Even half a second is a long time for the human user where a rapid response is expected [Shn98]. Contrary to the archived streams, this category requires a short end-to-end delay, which implies that the receiver performs no buffering or minimal buffering that can be used only to handle moderate reordering. Therefore short-term variations in the delivered bit-rate must be minimized, which means that the packets of the stream must flow fluently through the network to the receiver. Small short-term variations translate to a tight upper-bound for the end-to-end delay and jitter. If a packet from the stream is dropped, there is no way for the recovery by retransmission before the moment of presentation is at hand because too long a delay would be introduced. Equally, if a packet in the stream is delayed excessively, it is discarded by the receiver or the codec. Such discarded packets only waste link bandwidth. Whenever an underlying network fails to deliver the required bit-rate for a real-time stream, the viewer quickly notices it from the unnatural part that occurs in the presentation at the point of the drop or the delay. The codecs are able to conceal a drop or a delay of couple packets. Even though the codecs evolve, it is important to minimize the network's rate of failure because the codec approach has a limit in extent to which it is applicable.

With a long propagation delay, the wireless link constrains the delay bound of the other components that is necessary to meet the requirements of the real-time streaming. With propagation delays of 200-300 ms, it is clear that one round-trip takes too long to perform any retransmissions, otherwise the stream is no longer a real-time stream. Because of the additional delay, the real-time streaming cannot benefit from TCP's retransmissions that are intended to provide reliability. Such retransmissions can even be harmful because the segments with a higher sequence number are not delivered to the receiving application until the missing segment arrives. This blocking introduces an unwanted delay and jitter. Therefore User Datagram Protocol (UDP) [Pos80] is a viable alternative for real-time streaming but nothing restricts

streaming applications to use UDP only. UDP, however, does not offer any tools to session management.

Real-time transport protocol (RTP) [SCFJ03] provides a framework for streaming traffic with session management. An RTP message includes a header and a payload and is built upon a transport protocol (e.g., UDP). An additional framework for sharing streaming metadata includes RTP control protocol (RTCP) [SCFJ03] and Real-time streaming protocol (RTSP) [SRL98]. RTCP specifies how the end hosts share information about stream quality. RTSP provides a "remote controller" for the RTP streams, but it is not limited to RTP as a stream carrier. RTSP uses HTTP-like requests but instead of receiving the requested stream through the same connection as HTTP does, in a typical case, another protocol is used (typically RTP).

With real-time streaming the messages are sent frequently to maintain the liveliness of the presentation. Assuming that the messages are constant length, we can calculate the payload size of each message from the inter-packet sending delay and the media bit-rate that is the output of the codec. For example, with 20 ms inter-packet delay and 16 kbps bit-rate, the payload per message is only 40 bytes. If the message size within the stream varies, this value is the average payload of the messages.

At a bottleneck, the preference of the real-time streaming for a short queuing delay conflicts with the buffering requirements of TCP. Obviously the relatively long buffer required by TCP, when it is full, takes a long time to transmit. When competing TCP flows are present while a stream is transmitted, the packets of the stream get stuck into the queue of the bottleneck router for long periods of time because of the full-queue problem [BCC+98]. A streaming packet cannot just wait that long. Instead, the service must be differentiated so that the streaming packets can bypass the TCP packets awaiting in the buffer.

A later starting real-time stream is vulnerable in its beginning to the full queue problem but succeeds in the competition better than a later starting TCP flow. In the beginning, a large part of the stream is dropped because of the full queue. For the receiver, a large drop rate means a poor quality of the presentation, which the user unlikely wants. To counter the full queue related drops, the stream should be protected from the full queue that is caused by TCP flows. Contrary to TCP flows, many streaming flows are not TCP friendly [BCC+98, FF99], which basically means that they do not apply a congestion control that is similar to the TCP's congestion control. No matter how many packets of the streaming flow are dropped by the intermediate routers, its transfer rate does not adapt downward. Nor performs the stream a slow start. The packets of the stream arriving to the bottleneck have a short inter-arrival interval. Therefore they likely push out some TCP packets, which would fit to the buffer without the stream. Then the dropped TCP packets cause a congestion window reduction for the competing TCP flows, which makes room for the stream. Hence, the drop rate of the stream tends to decrease after the beginning.

3 Traffic Controlling Mechanisms

Three different kind of solutions for the problems of TCP occurring in the wireless environment have been suggested: link-level, end-to-end TCP, and split-connection schemes [BPSK96]. We discuss here some link-level techniques and enhanced TCP implementations. In addition, advanced router algorithms are discussed. The router algorithms discussed, in general, are applicable to any network regardless of the presence of the wireless link.

3.1 Link-Level Techniques

Many link-level techniques target at a lower residual bit-error rate. Firstly, redundant error-correction information, a *forward error correction* [KBF+04], can be used by a wireless link protocol to encode redundant information into a frame, which the receiver uses to reconstruct the original frame when a part of the frame is corrupted. The redundant information is useful only if the frame is corrupted, otherwise it only wastes bandwidth. An advantage of redundancy is that the original frame can be reconstructed immediately without waiting for the retransmission.

Another solution is a link-level retransmission of corrupted frames, an Automated Repeat reQuest (ARQ) [FW02]. It does not require adding redundant information into the frames except the checksum for detecting frame corruption. The corrupted frames are resent, causing additional round-trip delay. Because of the delay, the receiver cannot reconstruct the original frame until the retransmission arrives. Before the retransmitted frame arrives, multiple frames that were sent prior to the retransmission, arrive at the receiver. The receiver has three options on how these frames are handed to the upper level of the protocol stack: the receiver hands the uncorrupted frames immediately, the receiver buffers the frames until an in-order delivery can be performed, or all out-of-order frames are discarded. Discarding is a very poor solution because the wireless link is likely to be the most expensive resource of the system. The order of delivery is very significant for Internet protocol in general. With TCP reordering results in duplicate ACKs. TCP can accept only a few out-of-order segments without triggering a loss recovery, because a duplicate ACK should be sent after an arrival of every out-of-order segment [APS99]. Therefore most ARQ schemes choose in-order delivery, which in turn introduces delay and burstiness because the out-of-order frames, which are held back by the link level, are delivered back-to-back when the delivery is finally possible. To compensate this burstiness, spacing ACKs with an artificial delay is possible [BKG+01].

3.2 Router Active Queue Management

The router Active Queue Management (AQM) is a proactive method reacting to a developing congestion and to a resulting full-queue problem. Because of the TCP's default behavior, the queue of the bottleneck router remains full for long periods

of time with the traditional tail-drop algorithm [BCC+98, DMKM01]. With a full queue, queuing delays are longer and short bursts cannot fit to the queue, which is one intention of the queuing. The router using AQM tries to keep the queue short except during short bursts.

With the tail-drop algorithm, packets are only dropped when the queue is full. On the contrary, a router with AQM aims at smoothly dropping packets before the queue is full. As every drop is an indication of a congestion for the sending host, the sender reduces the congestion window. Thereby the router forces senders to reduce the rate of the incoming traffic before the queue overflows. Because a legitimate TCP implementation is forced to react to every drop, AQM does not require any modification to the TCP implementation. A smaller number of packets-in-flight is reflected to the queue length of the router, which becomes shorter. The shorter queue reduces queuing delays and prevents starvation of the flow that is unlucky with the tail-drop. The lock-out problem is prevented and later starting flows can open the congestion window more rapidly because round-trip times are shorter. In addition, the shorter queuing delay is very important for delay sensitive applications such as real-time streaming.

With AQM the tail-drop approach is left only for the case of extreme congestion when the router queue is really exhausted during a load spike. As a correctly configured AQM router is reacting to incipient congestion beforehand, tail dropping is only an infrequent event.

Random Early Detection

The most widely used AQM algorithm, the Random Early Detection (RED) [FJ93], is recommended to be deployed on the Internet routers [BCC+98]. RED estimates an average queue length based on the history of the instant queue length (see Figure 2). The estimation is an exponentially weighted moving average with a configurable weight (w_q). The average queue length with an appropriate weight is slow to rise during short bursts but it gets larger when congestion is more persistent. Then it is necessary to halt the growth of the queue length by signaling congestion with a drop.

if the queue is non-empty
$$queue_{avg} = (1 - w_q) \cdot queue_{avg} + w_q \cdot queue_{instant}$$
else
$$queue_{avg} = (1 - w_q)^{f_{linear}(idletime)} \cdot queue_{avg}$$

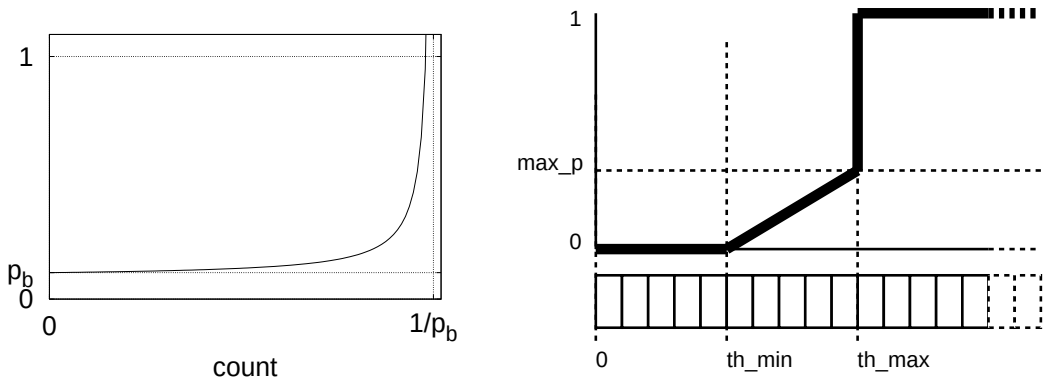
Figure 2: RED formula for the exponentially weighted moving average calculation from the instant queue size

The queue average can be calculated in two ways. The first way updates the average only when a new packet arrives. When the packets do not arrive back-to-back, the

idle period that occurs between the packets must be taken account. Another way is to calculate the average in a periodic background process. The latter solution becomes more attractive when the number of packets that must be handled per second increases.

The drops with the RED algorithm are probabilistic. There are three configurable parameters which define the dropping behavior of the algorithm: a *minimum threshold* (th_{min}), a *maximum threshold* (th_{max}), and a *maximum-initial packet marking probability* (max_p). When the average queue length is below the minimum threshold, no packets are dropped, and when it is above the maximum threshold, all incoming packets are dropped. On the region between the minimum and maximum threshold, an incoming packet is dropped with the probability p_a, which depends on the number of packets received since the last drop (*count*) and the current average queue length (see Figure 3.2).

$$p_a = \frac{p_b}{1 - count \cdot p_b}, \text{ in which } p_b = max_p \cdot \frac{queue_{avg} - th_{min}}{th_{max} - th_{min}}$$



(a) Final drop probability p_a (b) Intermediate probability p_b

Figure 3: RED dropping probabilities

Many RED variants have been developed (list is not extensive): *Gentle RED* [RBL99] has another region of a probabilistic drop between the maximum threshold and twice the maximum threshold. The drop probability increases from max_p to 1 in that region. *Weighted RED* [Cisco] is using different RED parameters for each class of traffic in the same queue. *Stabilized RED* [OLW99] estimates the number of active flows, which is used for determining RED dropping probability. *Adaptive RED* [FKSS97] varies max_p based on the traffic pattern.

3.3 Differentiated Services

The Differentiated services (DiffServ or DS) mechanism [BBC+98, Hus00] is a divide-and-conquer approach for providing Internet-wide quality of service (QoS). A net-

work with DiffServ is presented in Figure 4. A DiffServ-capable network, which is called a DS domain, is divided into the interior nodes and the boundary nodes. Multiple contiguous DS domains form a DS region. Thus the emphasized path in Figure 4 is within the same DS region.

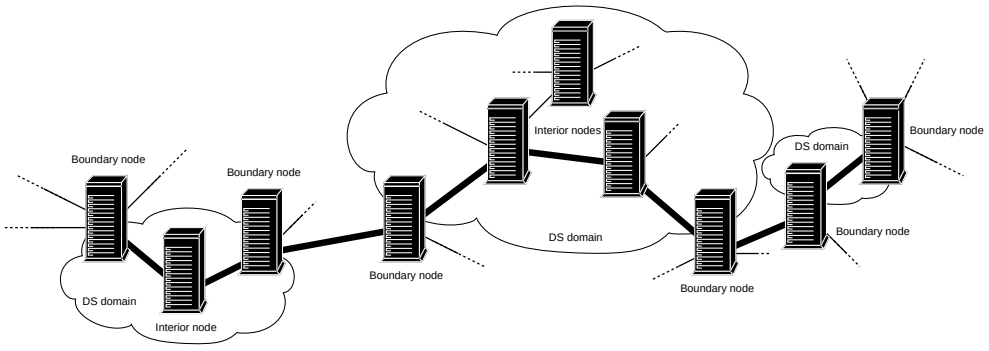


Figure 4: A network with DiffServ

For traffic controlling, DiffServ has allocated a DS field from the IP header [NBBB98]. The field is eight bits long and replaces the Type-of-Service (TOS) field of the IP header. Only six bits of the DS field are used by DiffServ as a DS Codepoint (DSCP) and the remaining two bits are reserved for other use [NBBB98, RFB01]. The DSCP is used for selecting a Per-Hop Behavior (PHB), which defines how the interior nodes handle the packet. DSCP defines the quality-of-service request of the packet. As the DSCP field is only six bits long, a DS domain is limited to the maximum of 64 simultaneously available PHBs. Universally, however, there can exist more than 64 PHBs. Therefore the DSCP space is mapped from the PHB identification code space [BBCF01]. It is recommended that half of the DSCP space is allocated for the standard action PHBs, which are defined by the Internet engineering task force. DSCP mappings are DS-domain specific but common mappings are expected to appear rather than to have no convergence. In different DS domains conflicting DSCP mappings can be defined because the network administrators are allowed to define their own PHB mapping for the DSCPs. These conflicts are addressed in a Service Level Agreement (SLA) by introducing appropriate re-marking rules, which change DSCP of the packet, to the communication between DS domains. A node claiming DiffServ compliance, however, must meet the minimal PHB requirements [NBBB98]. The goal of the requirements is to provide a smooth transition from the current routing technology to DiffServ. This restricts the use of the DSCP space as the transition is achieved by offering compatibility with the precedence field of the TOS field [Pos81a, Bak95].

The boundary node communicates with the hosts that do not belong to the same DS domain. Such exterior hosts could even be non-DS-capable. The boundary nodes can be split into two subcategories. The host handling the incoming traffic from outside of the DS domain are the ingress nodes. While the egress nodes handle the packets leaving the DS domain and are often similar with the interior nodes. An

additional task of the egress node is to re-mark the packets by modifying the DS field if required by the SLA.

An ingress node decides with which DSCP the packet is allowed to proceed into the interior nodes. For each packet entering a DS domain the ingress node selects a class into which the traffic belongs. Two types of the classification exist. The behavior aggregate classifier looks only at DSCP. The multi-field classifier considers also the other fields of the packet header and non-packet information such as the incoming interface. The resulting classification can therefore extend also to the flow or host granularity.

The interior nodes of a DS domain receive traffic only from the same DS domain. They forward packets only to another node within the same DS domain. The packet management in the inner network depends on DSCP that selects PHB with the corresponding class. All per-flow state information can therefore be allocated only on the boundary node while the interior nodes are left stateless. The removal of this massive amount of state information is a serious advantage for scalability as the other QoS mechanisms such as the Integrated services [BCS94] require per-flow state on each host of the transfer path. In a real environment the node categorization is not strict, one node can act in multiple categories at the same time.

Any DiffServ node may perform a traffic conditioning but it is typically happening on the ingress nodes only. The node with the traffic conditioner checks each packet against the traffic profile of the class. These profiles define the characteristics of the traffic that the class enforces. Then the router can apply different conditioning actions if the packet does not fit into the profile (e.g., the bandwidth limit of the class is violated). These actions include:

- **Shaping**: The packet is delayed.
- **Policing**: The packet is dropped.
- **Re-marked**: The DSCP of the packet is changed. This operation changes the QoS request of the packet (e.g., to lower priority or to best-effort).
- **Accounting**: The packet is allowed to continue but the account of the sender is charged.

A rate based profile can be implemented with a Token Bucket Filter (TBF) [Hus00]. TBF has a bucket of tokens. Tokens are generated to the bucket at configurable rate. Each packet consumes a number of tokens according to its size in bytes or one token per packet. If sending the packet requires more tokens than is available in the bucket, the corresponding conditioning action is performed. If shaping is used, the packet must wait until enough tokens have been regenerated. The size of the bucket determines the maximum length of a burst that the TBF can transmit at the full rate of the out-bound interface. When tokens are exhausted, the configured rate of the token generation determines the maximum rate of transfer.

A DiffServ-capable router has one or multiple queues into which the packets are placed depending on the classification. For QoS it is necessary to prioritize one traffic class above the other somehow (e.g., in latency or in drop probability). To perform prioritization, the DiffServ router must schedule the queues and during the times of heavy load also apply congestion control. The DiffServ architecture does not specify the scheduling and congestion control algorithms for the routers. This freedom leaves room for deployment of the different algorithms on the DiffServ routers. Each queue has its own queue management algorithm that is configured to perform the functionality of the specific class. Queue management algorithms were discussed in Section 3.2. When the outbound interface is able to send more data, it is necessary to select the next packet to transmit over the link. The selection of the next packet to be sent is a scheduling decision. Many scheduling solutions are suggested:

Priority Queuing (PQ) always schedules from a queue with the highest priority that has a packet available. Without other means of bandwidth limitation (e.g., a token bucket), the queues with a low priority starve if a higher priority queues have always a packet to schedule, or they experience large, unbounded delays.

Round-Robin (RR) [Kes97] scans cyclically multiple queues and schedules one packet from each queue. All queues receive a guaranteed minimum bandwidth without starvation.

Weighted Round-Robin (WRR) [Kes97] is a generalization of RR having different weight for each queue.

Deficit Round-Robin (DRR) [SV95] drops the requirement of WRR to know the mean packet size to fairly share.

Weighted Fair Queuing (WFQ) [DKS89] combines the benefits of PQ and WRR. It approximates a bit-by-bit based decision of the Generalized Processor Sharing (GPS) [Kes97]. Thus it is called also with other name: the Packet-by-packet Generalized Processor Sharing (PGPS). GPS guarantees a fair sharing without starvation in a theoretical, bit-precision environment. WFQ assigns each arriving packet a finish time, which is the time the bit-based GPS would have sent the last bit of the packet. The packets are then served in the smallest finish-time-first order.

Worst-case Fair Weighted Fair Queuing (WF^2Q) [BZ96] improves WFQ to differ even less from GPS. With WFQ it was possible that a packet could start the service much before the same event would happen in GPS. In a large time-scale this unbalance must be compensated with a period of smaller than average service, which increases burstiness. WF^2Q removes this discrepancy by considering only the smallest finish-time of the packets which GPS would be serving or would have already served when the next packet is scheduled.

WF^2Q+ is a WF^2Q variant which reduces complexity of this scheduling algorithm from linear to O(log n) [BZ97].

Class Based Queuing (CBQ) [FJ95] offers, in general, fair sharing without starvation as WFQ does. CBQ is less complex to implement and more flexible than WFQ allowing unused bandwidth to be channeled with a precision to a subset of the other queues, rather than making it available for all other queues with an equal share.

Even though we have listed some of the scheduling algorithms currently used, it should be noted that scheduling is a complex topic under an active research and that consensus about the best algorithms does not exist.

At the moment the Internet Engineering Task Force has the PHBs specified for an Explicit Forwarding (EF) [DCB+02] and an Assured Forwarding (AF) [HBWW99]. The EF PHB provides a low latency and drop service for traffic whose maximum rate is negotiated in SLA. It is ideal for streaming services. The AF PHB creates multiple classes with an increasing discard probability that is relative to the other classes. An application may indicate a relative importance of the packets by assigning them to different classes.

3.4 TCP Enhancements

It is possible to modify a TCP implementation so that the special needs of the wireless links are taken into account. We call such a modification *a TCP variant.* In a common case, the sending-side TCP implementation should be changed to enable a variant because most of the variants are sending side only modifications or rely on the cooperation of the end hosts. Therefore the variant affects all Internet servers not just the mobile devices of the end users. Changing Internet-wide TCP implementation is not a trivial task because the deployment takes some time. Nevertheless, the variant must be widely deployed in the Internet hosts to be effective from the perspective of the end user, which requires standardization. Although there are plenty of research proposals, we introduce only some TCP variants that are standardized or documented by the Internet Engineering Task Force:

Increased initial window [AFP02] allows TCP streams to open the congestion windows more rapidly. Instead of one segment, the slow start begins with an initial window of two or more segments. Less round-trips are required to reach the congestion window of the size of the delay-bandwidth product, which is necessary to fully utilize the link. A larger initial window increases probability that a flow has attained a congestion window that is large enough to trigger three duplicate ACKs after a loss.

Limited transmit [ABF01] allows a sender to respond with a new segment also to the first two duplicate ACKs that arrive prior to the third one triggering the loss recovery. Such a response increases probability that the receiver is able to generate the third duplicate ACK also when the congestion window is small or a large number

of segments are lost in a single window. A fast recovery that follows the third duplicate ACK occurs sooner than in the case where the sender waits for the RTO timer to expire. In addition, the congestion window is not necessarily reset to one.

Forward RTO recovery (F-RTO) [SKR03, SK05] detects spurious retransmission timeouts. The detection enables the sender to discern the congestion response that was triggered falsely. The sender with F-RTO does not perform the unnecessary retransmission of the whole window in the case of a spurious timeout. After detection of the spurious RTO, the sender can undo some effects of the congestion response. The F-RTO detection algorithm [SK05], however, does not specify the response, which can be freely chosen. Possible response algorithms include in a conservative response suggested by Sarolahti et al. [SKR03], which halves the congestion window to the level of ssthresh, and Eifel response algorithm [LG05], which restores the congestion window and ssthresh to the level prior the spurious RTO.

Control Block Interdependence (CBI) [Tou97] enables a TCP connection to use information acquired by the earlier flows about the route to the peer end host. The congestion control state information of the earlier flows can help the later flows, e.g., to avoid the slow-start overshoot, to estimate the round-trip time more accurately, and to share other information that is useful for the specific TCP implementation. CBI uses the slow-start threshold of the earlier connections, which is very likely smaller than the default value. With the pre-probed initial slow-start threshold, the overshoot in the slow start is less severe or does not occur at all. However, also false control information is propagated. For example, when an error burst takes place near the end of a preceding flow. The following connection starts with a low slow-start threshold, and because of that, it proceeds most of its time slowly in the congestion avoidance.

Explicit Congestion Notification

Having a non-full queue when a new packet arrives, a router has two choices because it no longer has to drop the packet that AQM urges to drop. The router can select whether the packet should be queued or dropped. A new option is made possible: signaling congestion without the drop that the sender interprets as an indication of congestion. Explicit Congestion Notification (ECN) [RFB01, Flo94] takes advantage of this option. It enables an inter-mediate router to notify the flow about congestion without drops, which is called *marking*. Instead of the drop the router adds into the packet a *would-have-been-dropped* mark and queues it normally. ECN specifies a two-bit field into the IP header for marking purposes. This field is mutable by an intermediate router that signals congestion to an end host. The field values are interpreted as:

- **ECN-Capable Transport (ECT)** indicates for an intermediate router that the end hosts are ECN-capable. Only when both end hosts have agreed to use ECN, the end hosts are allowed to use ECT-flagged packets.

- **Congestion Experienced (CE)** replaces ECT when the intermediate router notices that a congestion condition is developing. The router can only mark the packets having ECT or the packets that have already been CE marked by another router.

- **Non-ECT** packet must not be marked with CE by the intermediate router. This flag indicates that the flow does not support ECN or a certain kind of packet in a flow which must be marked with Non-ECT by the end hosts even if ECN is enabled (e.g., TCP retransmission or pure ACK). If AQM urges a packet with this flag to be dropped, it cannot be marked but it is dropped as without ECN.

The flow that receives the mark may be able to reduce its sending rate before the router queue overflows. This spares the sender from the retransmission because the original packet was delivered despite of the marking. For ECN to function correctly, both end hosts have to support ECN and cooperate with the router as the mark is sent with the flow to the receiving end. As the sender should reduce its sending rate, the receiver of the marked packet must communicate the mark back to the sender. It is achieved in TCP with additional flags, which are set by the end hosts only. These flags are:

- **ECN Echo (ECE)** is set in the TCP ACK when the receiver receives a CE marked packet from the network.

- **Congestion Window Reduced (CWR)** is sent with new data segment when the sender has reduced its congestion window. The ECN-capable sender must set this flag whether the reduction was of ECN origin or not.

To support incremental deployment of ECN, the ECN capability is first negotiated in a TCP connection between the end hosts on the TCP level. If both hosts agree to be ECN-capable, ECN is enabled for the flow. When a CE marked packet is received, the TCP receiver starts to set ECE-flag in every ACK. The ECE-flag is sent until a CWR-flagged TCP segment, which "acknowledges" the ECE-flag, arrives at the receiver. When the sender receives the ECE-flagged packet, it acts as if a packet would have been dropped but does not activate the loss recovery. This behavior leads to reduction of the transmission rate of the marked flow as if the marked packet would have been dropped. CWR-flag is sent only once with the first new segment after the window reduction because the "acknowledgement" signal must be reliably delivered to the receiver [RFB01].

Multiple ECE-flagged packets are typically sent before the sender receives information about congestion and can respond with CWR which then must reach the receiver of the marked packet. Assuming no packets are dropped, it takes one round-trip before the first non-ECE packet arrives to the sender. The sender must avoid multiple congestion window reductions during this round-trip but still respond to a loss that belongs to the next window of data. Therefore if the end hosts have agreed

to use ECN, they must respond to every CE marked packet and to ECE-flagged packets only once per round-trip time.

4 Test Arrangements

4.1 Objectives

We study the benefits of different mechanisms on traffic mixtures, whether the mechanisms still provide the same benefits when the link behavior is changed, and interaction of the mechanisms with different link behaviors. The mechanisms include DiffServ, RED with ECN, a couple of TCP variants, and their combinations. We employ DiffServ and RED with ECN at the last-hop router of the wireless environment while the end hosts apply the TCP variants. The traffic mixtures include bulk data TCP transfers, streaming traffic and HTTP transfers. The study concentrates on the behavior in environment with a GPRS/UMTS-type wireless link. In the tests, an emulated wireless link, which loosely resembles the parameters of the link using GPRS/UMTS-technology, is set up. The quality of service for the traffic mixtures is evaluated, and the performance of the different configurations is compared with each other.

We seek answers to these general questions:

1. Can the technologies that are intended for Internet-wide quality of service (QoS) and congestion management be applied successfully into an environment with the wireless link?

2. Does a combination of TCP enhancements improve performance? And what are the effects when the combination is used with the QoS and congestion management mechanisms?

3. Can we improve the HTTP response times?

4. Does the performance of the streaming flow meet the requirements of the real-time streaming?

5. How does the tested router techniques affect the behavior of the bulk TCP flows? Do they suffer considerably because of them?

6. How do the different congestion control phases of the bulk TCP flows affect the competing traffic and vice-versa?

We study the performance in detail when suboptimal performance is noticed, and we identify the cause of the sub-optimal performance and check if an alternative configuration improves the performance.

4.2 Methods and Model

In the topmost part of Figure 5 we see the target environment we want to model. We use a real-time software emulator, the Seawind wireless network emulator [KGM+01], to emulate the properties of the target environment. In the middle of Figure 5 is the host configuration used in the test network and below it are the emulation components of Seawind and how they are distributed to the hosts participating in the tests. For accurate emulation, it is necessary to avoid interfering network traffic from other sources, which are not under the control of the testing software. We achieve this by isolating the test network. Similarly, the interfering, unnecessary processes on the hosts of the test network are shutdown before the tests because the processes of Seawind must be scheduled whenever they demand.

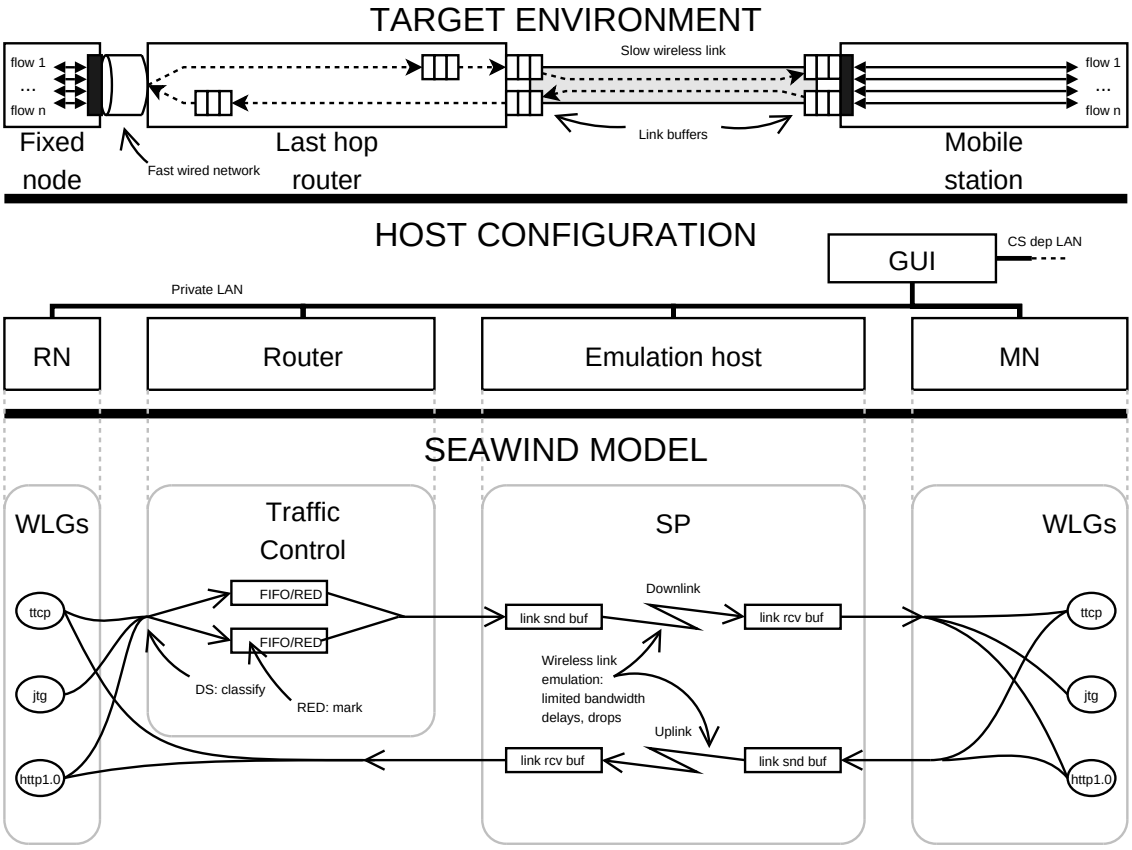


Figure 5: The Target and Emulation Environments

Using Seawind emulation instead of a physically wireless medium gives us fine grain control over the characteristics of the link and an ability to repeat tests with similar characteristics, which in turn allows result comparison. For user convenience, Seawind allows running multiple test cases in a predefined test set automatically. In order to obtain more accurate information for statistics each configuration is tested with 120 repetitions. Each repetition runs in real-time, even though the Seawind

system was automated to perform the tests round the clock without user intervention, it took nearly two months to complete the tests. Therefore we are not able to have more repetitions.

The test network has five hosts that are connected through a 100 Mbit/s switch. Four machines are needed for the emulation and the fifth is running the Seawind control tool. Two of the machines are the end hosts and the third machine acts as a last-hop router. The fourth one is an emulation host that runs the Seawind Simulation Process (SP) that emulates a wireless link. The end hosts communicate with each other through the last-hop router and the emulated wireless link. The operating system in each host participating in the test is Linux.

We assume that the end-to-end packet delay is dominated by the delays that are introduced in the queuing and in the transfer over the wireless link. Therefore we ignore the effect of internal processing and transmission between Seawind components over the 100 Mbit/s network, which are of a negligible magnitude.

While a test is running, Seawind is emulating the events that would be happening on a real wireless link. If one of the emulation events is delayed, the model and the reality will no longer match. Consequently, the emulation events require a sub-millisecond precision and are time critical operations, but Linux is not a real-time operating system. Therefore we have to use extreme caution to get accurate results as Linux cannot guarantee the scheduling of the process when a time critical operation happens. We remove the effect of the other processes that run in a default Linux system by shutting unnecessary background services down and by avoiding any other use of the hosts during a test. Off the shelf Linux kernel of 2.4 line for i386 has a 100 Hz timer, which causes a ten milliseconds sleep granularity that conflicts with the required precision. Therefore Seawind has chosen to use a busy-wait for the last ten milliseconds before a event to avoid oversleeping due to the sleep granularity. In addition, Seawind provides a mechanism that can be used to detect the inconsistent test cases. The timer oversleeps are logged by the emulator automatically. When significant delays occur, the specific test case is ignored.

The data for each workload is generated by the Seawind system using a component called Workload Generator (WLG). These components are run on a normal Linux system having a normal implementation of the TCP/IP stack. Thus our system has a realistic TCP/IP implementation of the Linux system instead of abstract models used by the simulators such as the Network simulator [ISI05]. The end hosts are running Linux kernel version 2.4.22 with modifications [IIPa] that allow tuning of the TCP implementation while the system is running and enable testing of the TCP enhancements that are not included in the standard Linux kernel.

In Figure 5 the Seawind model shows how traffic is transferred within the test network. Traffic from the fixed node WLGs is transmitted to the last-hop router. DiffServ is employed for packets by the Linux traffic control [HGM+04]. In part of our test cases DiffServ is disabled. For those cases, we use a similar network setup but the DiffServ router has only one queue, which effectively disables DiffServ. In the other cases, there are multiple queues based on the DiffServ configuration. These

queues are configured to tail-drop or RED depending on the configuration of the particular test case. From the router, the packets enter to the wireless link on SP and then proceed onward to the mobile node.

The path for traffic in an opposite direction has a simplified path because the much faster wired link with a low latency should always be able to immediately transfer the packet onward when it arrives from the wireless link. Therefore we consider the effect of the last-hop router uplink buffers negligible for the tests. Furthermore, streaming traffic used in the tests is unidirectional without an uplink transfer and small-sized HTTP requests and TCP ACKs generate only a little load for the uplink send buffer so that no packet should be lost due to congestion. To guarantee that no losses happen, we configure the uplink send buffer for unlimited buffer space. Thus, for now on, *link send buffer* refers to the downlink send buffer.

We calculate an end-to-end delay of the streaming packets. To keep the end-to-end delays accurately, we need to keep the clocks synchronized over the hosts participating in the test. At the same time, we do not want to do that uncontrollably on the background by a Network time protocol [Mil92] daemon as it could create inaccuracies to our measurements. Instead, we disable the daemon while we run the tests and do an instant synchronization of the clocks between each basic test. The testing system logs the offset that the clock was moved between basic tests. It was verified that in the tests containing the streaming flow the offset did not exceed 0.2 ms.

4.3 Configuration of the Wireless Link

The used bandwidths of the wireless link are 64 kbps and 384 kbps. The propagation delays are 300 ms and 200 ms respectively. Maximum Transmission Unit (MTU) used in the tests is 576 bytes. The link send and receive buffer sizes are set approximately to two times the delay-bandwidth product. In the Seawind system three packets fit between the last-hop router and the link send buffer. These packets reside in neither buffer and are required by the Seawind system. Table 1 summarizes the parameters of the link configuration.

Characteristics	**Low-speed link (GPRS-type link)**	**High-speed link (UMTS-type link)**
Transmission rate	64000 bps	384000 bps
Transmission delay	Frame size / Bandwidth	
Propagation delay	300 ms	200 ms
MTU	576 bytes	
Link send buffer size	12000 bytes	42000 bytes
Link receive buffer size	12000 bytes	42000 bytes

Table 1: The link characteristics for the wireless link emulation

For the wireless link, we use a two-state Markov error model. Table 2 specifies the states of the error model. The error model has two states, a good state and a bad state. The good state represents a good quality, error-free wireless environment and the bad state represents a bad-quality environment with a high error probability, which is configured to 60%. We use an exponential distribution for the lengths of the good and bad state with the mean of 10 seconds for the good state and of 2.5 seconds for the bad state. Every time the good state ends, the bad state begins and vice versa. To have exactly the same quality of the wireless link for each test case, one distribution of the state lengths was generated and it is used in every test case.

Parameter	**Good state**	**Bad state**
Error probability	0%	60%
Type of state length function	Exponential distribution	
State length, seconds (min/mean/max)	5.0 / 10.0 / 15.0	1.4 / 2.5 / 4.0
Maximum number of ARQ retransmissions	0/1/2/4	1/2/4
Retransmission delay	700 ms	

Table 2: States of the error model of the wireless link

In the bad state, the link-layer retransmissions of ARQ are emulated. If a packet is dropped, the link layer retransmission is emulated by delaying the packet for about one RTT instead of dropping it. The one-RTT additional delay is defined as 700 ms. If the link is still in the bad state, also the retransmission may experience a drop. If so, the packet requires another emulated retransmission, which takes additional 700 ms. The amount of link persistency defines the maximum number of times that the emulated retransmission is attempted. If none of the retransmissions succeed before the maximum is exceeded, the packet is counted as lost. We define four link types with different ARQ persistency:

- **Error-free link** does not cause any error-related drops to traffic that is transferred over the link. This link type uses only the good state of the error model. No link-level retransmissions are required to hide packet losses as all the packets are reliably delivered without them.
- **Link with low ARQ persistency** is a lossy link on which the link layer tries to retransmit a missing packet once to hide a drop that occurs on the link.
- **Link with medium ARQ persistency** is a lossy link on which the link layer tries to retransmit a missing packet at most twice.
- **Link with high ARQ persistency** is a lossy link on which the link layer tries to retransmit a missing packet at most four times. The link layer is persistent enough to recover from most of the packet losses during the bad state. Therefore the upper protocol layers experience only a small number of error drops, but the packet delays are increased when retransmissions occur.

For TCP's retransmission timer to trigger spuriously, a relatively long delay spike is needed. Because the maximum packet delay increases in proportion to the retransmission count, spurious TCP RTOs are excepted with this link type.

Packets are handed onward from the link receive buffer only in-order. The link-level of the receiver buffers all out-of-order packets, which are delivered only when the preceding packets have been successfully received or counted as lost. Therefore every 700 ms delay affects not only the packet in question but also many successive packets that cannot be handed to the receiver immediately.

The complete Seawind process configuration is listed in Appendix B.2.

4.4 Workloads

In this section we specify the traffic models and the configuration of the workload generators. The bulk data TCP transfer models a long lasting, unidirectional transfer, the HTTP transfer models a single Web page retrieval, and the streaming flow models an unidirectional streaming traffic on the top of UDP/IP. In every workload, there are two bulk data TCP transfers, which start at zero seconds (we refer to them as *zero-starting* TCP flows), and one or two competing transfers, which start after a delay. First we run two bulk data TCP transfers with only one competing workload type at a time. To match closer with real usage patterns we also run a combined workload with both HTTP transfer and the streaming traffic competing with the two bulk data TCP transfers. In all, there are five workloads with different competing transfer setting:

- Two bulk data TCP transfers and a competing short TCP transfer
- Two bulk data TCP transfers and a later starting, competing bulk data TCP transfer
- Two bulk data TCP transfers and a competing HTTP transfer
- Two bulk data TCP transfers and a competing streaming flow
- Two bulk data TCP transfers and a competing HTTP transfer and a competing streaming flow with two bursts

The delayed starting times target the different congestion-control phases of the zero-starting bulk data TCP transfers. By comparing the cases with different starting times we can estimate the effect of the phase of the background load. Exact times cannot be used as the timing of the phases varies between the configurations. This variation happens because of packet timing, different link configurations, error drops, and buffer size, which affect the duration of the slow start. Therefore the starting point is distributed between 0 and 36 seconds. For the higher link speed, the starting

time distribution is 0-13.5 seconds. All starting times are offset by the sum of the maximum lengths of the good and bad states because the zero-starting TCPs must be able to begin in either a bad or a good state.

The HTTP transfer consists of a HTML body and eight inline objects (e.g., images). The body is 11792 bytes long and each inline object is 8576 bytes long, which is 80400 bytes in total. The HTTP object sizes include the response header. The length of the socket write buffers is 4288 bytes, which is eight times the maximum segment size. A request size of the body is 351 bytes and a request size of the inline objects is 350 bytes.

The streaming flow is constant bit-rate type with 16000 bps media-rate, which resembles an output of G.726 [ITU90] codec. The packets are sent with 20 millisecond intervals. The stream consist 1147 packets, which are 76 bytes long with the UDP/IP headers. Hence, the bit-rate with the headers is 30400 bps. The size of the stream (UDP-m) is summarized in Table 3. The packets of the stream are non-ECN capable. In the workload with both the HTTP transfer and the streaming flow, the streaming flow includes two bursts, and the bursts are spaced with idle time that has an uniform distribution between 5-15 seconds.

Label	**Length**	**Duration**	**Total size**
UDP-m	1147 packets	22.94 sec	55056 bytes

Table 3: Streaming flow size

The bulk data TCP transfers are 343040 bytes, 1029120 bytes, or 2572800 bytes long. A mapping of sizes to different workloads is listed in Table 4. The two bulk data TCP transfers competing with an HTTP transfer and a streaming flow require long bulk data TCP transfers because of the burst spacing. The size of the short competing TCP flow is 21440 bytes, which is 40 Maximum Segment Size (MSS) sized segments. The length of TCP write buffers is eight times the MSS.

Link speed	**Workload**	**Size**	**Number of MSS sized segments**
64000 bps	All	343040 bytes	640
384000 bps	2xBulk TCP + Stream	1029120 bytes	1920
384000 bps	2xBulk TCP + HTTP + Stream	2572800 bytes	4800

Table 4: Bulk TCP sizes

The competing transfers are specified so that they complete before two zero-starting TCP transfers do, except in the workload that has three equal-sized bulk data TCP transfers. The earlier ending time ensures that the competing transfer has always two bulk TCP flows as a background load during the whole transfer regardless of the starting time. This allows comparing the results with different starting times.

The streaming flow is generated with *jtg* version 1.90 [JTG05]. The HTTP transfer is generated by Seawind's http 1.0 WLG version 1.01. The TCP flows are generated with modified *ttcp*. The http WLG and the modified ttcp are distributed with the standard Seawind distribution version 4.0.1 [KGM+01, IIPb].

The workloads are summarized in Appendix A.

4.5 TCP Variants

We define two TCP variants, Baseline TCP and Enhanced TCP, which are used in the tests. Baseline TCP is modified from the standard Linux kernel. The purpose of the modification is to remove some non-standard features of the Linux kernel and to add features that are expected to be soon included in "the standard TCP". The standard Linux kernel includes a dynamically adjusted Delayed ACK [Cla82] timer, delayed ACKs after slow start (known as quick acknowledgements in Linux) [All98, ADG+00, MDK+00], a rate halving [Hoe95, MSML99, MSM99], and a CBI reusing [Tou97] for duplicate ACK threshold, for round-trip estimator, for initial window, and for slow-start threshold. In addition, the standard kernel does not allow specifying the size of the initial window. To disable and control these features modifications to the kernel were made [IIPa]. The static timer value for the delayed ACK is set to 200 milliseconds. In Linux the CBI reusing stores values to per host entries when a TCP connection ends. Therefore values from an ongoing flow have no effect on a starting flow. Enhanced TCP is built on the top of the Baseline TCP and represents a combination of the selected TCP features. The variants are summarized in Table 5. Appendix B.1 lists the used kernel parameters.

Baseline TCP	**Enhanced TCP**
Limited transmit	Baseline TCP +
Initial window of two	Initial window of four
No CBI	CBI (reuse only ssthresh)
Selective acknowledgements (SACK)	F-RTO (uses conservative
Conservative SACK-based loss recovery	response [SKR03])
Delayed ACKs (200 ms)	
No rate halving	
No quick ACKs	

Table 5: TCP Variants

Our test kernel has a couple of noteworthy features that affect the functionality of CBI and F-RTO. A standard Linux kernel includes a garbage collector for the destination host entries that contain the data for CBI reusing. After the collector has disposed an initial ssthresh, the next connection gets the default value of ssthresh, which is 65535. The garbage collector was not disabled, which causes a slow-start overshoot to happen like without CBI when the destination entry is

disposed. The F-RTO algorithm counted erroneously an opposite direction retransmission as duplicate ACK. Therefore F-RTO fails to detect spurious RTO when an HTTP request is retransmitted. A retransmission of SYN into which the peer responds with SYN-ACK was handled as duplicate ACK by our F-RTO implementation. These SYN-ACKs are a corner case that F-RTO could ignore during detection of a spurious timeout.

4.6 Network Configuration

We employ DiffServ on the last-hop router using the Linux Traffic control [HGM+04]. The DiffServ configuration is presented in Figure 6. The configuration includes an individual class for bulk TCP, HTTP, and audio. In addition, an intermediate class, which is the parent of the HTTP and Bulk TCP classes, is needed. In the workload which consist of all three traffic types, both the separate and shared queue for bulk TCP and HTTP are tested. Depending on the configuration of the test case, the audio, HTTP, and bulk TCP classes use tail-drop or RED queue. The traffic control is configured to prioritize streaming flow over TCP traffic up to the bandwidth limit of the audio class, which is 32000 bit/s. A Token Bucket Filter (TBF) generates the bandwidth limit. The bucket size of the audio class is configured to MTU, which prevents audio from dominating the link for a long period of time. The configuration of the audio class follows closely the principles of the expedited forwarding PHB [DCB+02]. The intermediate class can use the remaining share of the link bandwidth because the bandwidth limit of the audio class is lower than the nominal link bandwidth. The available bandwidth varies because the streaming traffic does not last over the duration of the bulk TCP transfers and because drops due to corruption occur during the bad state, which effectively consume a part of the link bandwidth. The intermediate class uses a Hierarchical Token Bucket (HTB), which implements the sharing goals of the Class based queuing [Dev02]. The bulk TCP and HTTP classes have an equal priority, which allocates an equal bandwidth share for both. After an idle period, HTB is able to feed the whole link send buffer because the sizes of the HTB tokens are configured to match the size of the link send buffer. This configuration of scheduling algorithms is adequate for our workloads.

We use the recommended values for the RED configuration [FJ93, Flo97] and diverge from them as little as possible. The configuration is listed in Table 6. In earlier tests [Kul03], RED with ECN performed slightly better than without ECN. The number of test cases is reduced by enabling ECN with RED for all the test cases that have RED because it is expected to perform better than RED alone. The standard Traffic control tool (`tc`) uses also *burst* parameter, but only internally in calculation of the EWMA exponent that is an input of the kernel. This unnecessary step was bypassed with a small modification that makes possible to specify the exponent directly. The average packet size varies between the workloads because the percentage of the streaming IP packets varies in the different workloads. Therefore the average packet size is set constantly to the Maximum Transmission Unit (MTU). The average packet size is meaningful for an idle damping of the queue average, which

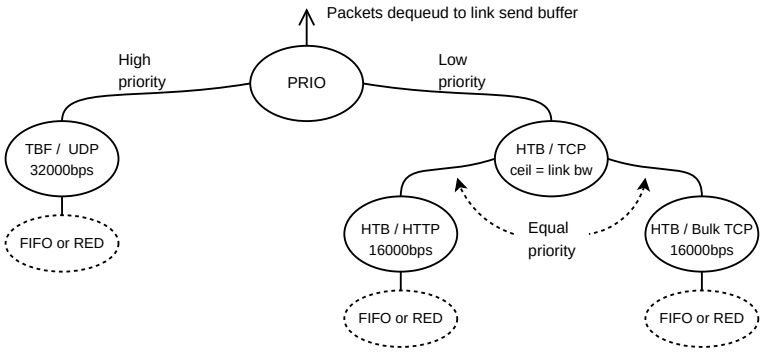


Figure 6: The DiffServ configuration

is precalculated to an array by the Traffic control tool and passed to the kernel. The relation between the average packet size and the idle damping is linear, and the difference of the real average packet size to MTU is up to 11% in the workload specified in Section 4.4. Thus, the queue average falls down up to 11% slower than in an ideally configured environment. However, the difference affects only during an idle period.

Preliminary tests were run to adjust the maximum threshold and the buffer_size, which are not specified by the recommendations. The selected values are based on the knowledge gathered during the preliminary tests. The standard RED algorithm of the Linux kernel contains an overflow bug, which was discovered and fixed (see Appendix C). During the analysis of the actual results, the loss recovery of the TCP implementation was noticed to be broken, and the actual tests were rerun with a corrected implementation. The broken recovery may have altered the results of the preliminary tests for determining the RED parameters.

Parameter	**Description**	**Suggested Value**	**Configured Value**
buffer_size	Physical queue size	free	11520 bytes (20 pkts)
$\min_{th}$	Lower threshold below which no packet is marked	5 pkts	1728 bytes (3 pkts)
$\max_{th}$	Upper threshold above which all incoming packets are marked	$3 \cdot min_{th}$	5184 bytes (9 pkts)
w_q	Weight in the low-pass filter used to compute the average queue length	0.002	$\frac{1}{2^9} \approx 0.00195$
$\max_p$	Maximum drop probability for marking	0.1	0.1
avpkt	The average number of bytes in a packet	free	576 bytes

Table 6: The configuration of last-hop router's RED

When the last-hop router employs tail-drop, FIFO queue is used. The size of the FIFO queue is set to the same value as the RED buffer_size parameter. With the higher link-speed two queue sizes are tested. First one is equal to the size with the lower link speed and the other is 80 packets.

4.7 Metrics

In this section we define the metrics that are used in the analysis. We group similar flows from individual repetitions. This grouping is done by classifying the flows within each repetition. The third flow, which is starting with a varying delay, has its own class. For this purpose, the multi-flow HTTP transfer (see Section 4.4) is considered as a single entity, and hence the metrics such as number drops, spurious RTOs, and unnecessary retransmissions are reported over all nine TCP connections of a HTTP transfer. In the workload which has both HTTP transfer and streaming flow, the streaming flow has a class of its own. The two TCP flows starting at time zero are classified by the elapsed time as the faster and slower flow in each repetition based on the performance data. The statistical indices of the metrics are calculated for each group. However, when necessary, a single test case or a flow is presented to point out a specific phenomenon.

The Seawind emulator system collects information from the components into multiple logs. We combine the information available in the logs to get more detailed view of the events which took place for each transferred packet. The main source of information are the *tcpdump* [TDUMP] logs of the end hosts, which store all traffic generated by the test case. The events which occurred on the wireless link are gathered from the Seawind process log.

Total elapsed time is the elapsed time of the entire workload transfer. The first and last packet do not necessarily belong to the same flow. The SYN and FIN-ACK packets are included. As the streaming flow is configured to end always before the slower TCP flow in all cases, the last packet cannot be a streaming packet.

Overall throughput of the workload is the total size of the transferred objects divided by the total elapsed time of the transfer. The sizes of the transferred objects include the transport protocol and IP headers. For streaming flows, the size of the transferred object is measured from the receiving end because of drops.

Link utilization tells how effectively the bandwidth of the wireless channel was used during the transfer of the entire workload. It is the amount of received data that includes transport protocol and IP headers divided by the total elapsed time and normalized by the link speed.

Elapsed time of a connection is the time needed to complete the entire transfer of the connection, including the time to establish and to close the connection. With TCP the SYN and FIN-ACK packets are included in the elapsed time.

Throughput of a transfer is the size of the transferred object divided by the elapsed time of the transfer. The size of the object includes the transport protocol and IP headers. For streaming flows, this value is calculated by dividing the amount of received data with the time between the first and the last packet sent. We have chosen the sending party because it is impossible to determinate the exact start and end time in the receiving end when a loss happens for either the first or last packet because queuing delay varies. The value typically overestimates the real throughput because it ignores the length of the extra delay that occurred for the last packet compared to the delay of the first packet

Simultaneous throughput is a throughput over the period during which all configured workload generators of the test case are transmitting packets simultaneously. The period starts when the latest Workload Generator (WLG) sends its first packet and ends when the first WLG to finish sends its last packet. For TCP, the SYN and FIN-ACK packets are included and only original transmissions of segments are counted. Thus a packet that is a retransmission of a segment which was first transmitted before the period started is not counted in the size of the transfer.

Number of dropped packets is the number of packets that were sent but on the way to the receiver the packets were dropped. There are two kinds of drops present in the test network:

- **Error** drops happen in the wireless channel because of data corruption.
- **Congestion** drops are caused by the queue management drops of the last-hop router.

Number of spurious RTOs is the subset of RTOs that could have been avoided by having a larger RTT estimate. We count as normal RTO only the cases where no ACK is received by the sender because the Limited transmit can help to recover from the cases with less than three duplicate ACKs.

Number of unnecessary retransmissions is the subset of the retransmissions which was unnecessarily retransmitted by the flow.

HTTP specific metrics:

Response time is the time from opening the TCP connection for the first object (HTML page) to the arrival of the last data segment of the last object.

Streaming specific metrics:

End-to-end delay measures the delay which was accumulated for the packet in a flow. This metric is very important for the interactivity of the real-time streaming applications. We are very interested on the high-end percentiles as they indicate the delay of a nearly full bit-rate stream.

IP-packet delay variation (IPDV) measures the difference between the end-to-end delay of the consecutive packets [DC02]. IPDV of a packet pair is calculated by subtracting the end-to-end delay of the earlier packet from the delay of the later packet. If either of the packets is dropped, the value is undefined. All pairs with undefined value are ignored in the calculation of the indices.

Consecutive packets dropped is the number of streaming packets that are lost in a contiguous sequence.

Network metrics:

Router queue length is the instant length of the last hop router queue. Our test system does not gather the exact value, but it can be estimated from the experienced queuing delay. The value is used in discussion.

Average queue length is used by the RED algorithm. It is an exponentially weighted moving average of the router queue length.

Indices used in the analysis:

Interquartile range is the difference between the upper and lower quartile (75th and 25th percentile).

Stability is derived from the interquartile range. It is used only in discussion. When the interquartile range is smaller, the stability is better and vice versa.

Fairness indicates how similar transfers in a test set cluster on one of the metrics. If the performance of one transfer is considerably lower than the performance of another, the fairness is low. Besides the approximate definition, to measure fairness a Jain fairness index [Jai91] is used:

$$f = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n\sum_{i=1}^{n} {x_i}^2}$$ where $x_1, x_2, \ldots, x_n$ are n instances of the metric of interest.

5 Test Results and Analysis

In this section we analyze the results. Analysis is performed workload by workload. Analysis of each workload is organized by the link type. First, we analyze the results over the error-free link, then the links with ARQ, from high ARQ persistency to low ARQ persistency. The analysis in grouped further by the last-hop router and end-host TCP configurations, which vary between the workloads and are explained in detail per workload. After all link types are analyzed, we include a short summary of

the main findings with the workload. Finally, we discuss the results to see whether the findings agree with each other.

The storage space required to store the uncompressed, raw log files of the Seawind system is over 150 gigabytes. A collection of scripts were used to generate intermediate data files from the log files of the Seawind system. These data files categorize the immense amount of data so that trends can easily be extracted and presented in figures or tables. Full tables of the performance data are reported in supplementary [Jär06]. We analyze the macroscopic behavior based on figures that show statistical indices of metrics. We compare the performance of different configurations with each other. The comparison is mainly performed with Baseline TCP. When an interesting phenomena is found, we explain its microscopic behavior.

5.1 Two Bulk TCP Flows Competing with a Short TCP Flow

In the workload with short competing TCP flow, there are three flows. Two of them are bulk data TCP transfers that start at zero seconds. We refer to them as faster or slower flow. As these flows start at the same time, we can directly derive from the elapsed times which one is the faster flow. The competing TCP flow that starts later is labeled as the third flow.

We report the performance of the whole workload using the overall throughput. The value includes headers and therefore its theoretical maximum is 8000 B/s. We use elapsed time as the basic metric for the performance of the third flow. As the third flow is short, it is not able to reach the same level of throughput as the bulk TCP connections because the slow-start phase has a large weight in the throughput of the third flow. Therefore we will not compare its throughput with the throughputs of the bulk TCP flows.

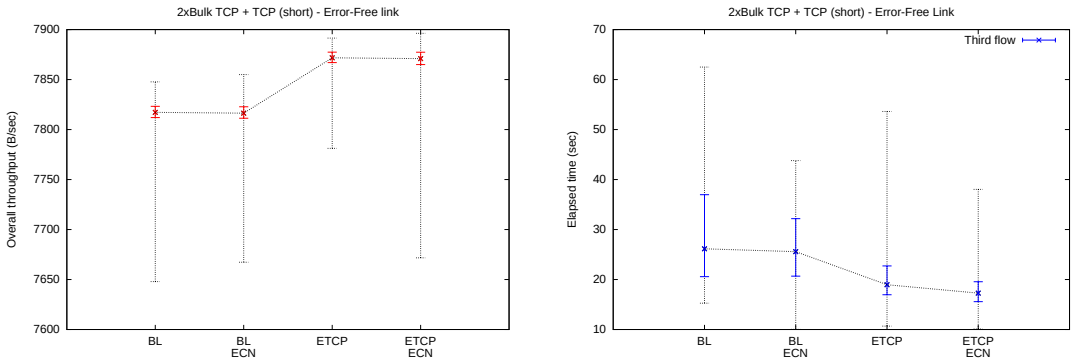
For each link type we use figures that compare the results with Baseline TCP (BL), Baseline TCP with ECN (BL+ECN), Enhanced TCP (ETCP), and Enhanced TCP with ECN (ETCP+ECN). We analyze the cases in that order. The figures show the median and the quartiles with a solid errorbar for the given metrics. When present, a dotted errorbar shows the minimum and the maximum.

5.1.1 Error-Free Link

The overall throughput over the error-free link is shown in Figure 7a. With Enhanced TCP, the overall throughput is higher than with Baseline TCP.

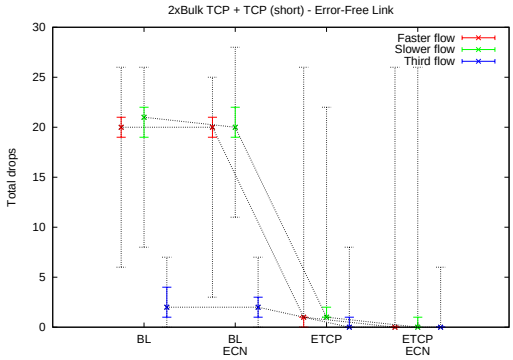
The elapsed time of the third flow is shown in Figure 7b. With Enhanced TCP, the elapsed time of the third flow is shorter and more stable than with Baseline TCP. The the elapsed time of the third flow is more stable with ECN than without it. Also a slight improvement in the median elapsed time can be observed with ECN.

The drops for each flow are shown in Figure 7c. With Baseline TCP, the slow-start overshoot happens causing many drops. With Enhanced TCP, the slow-start

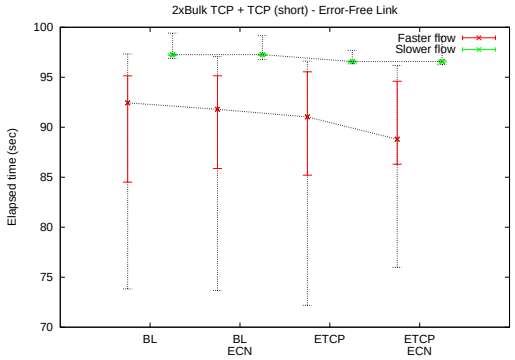


(a) Overall throughput

(b) Elapsed time of the third flow



(c) Packets dropped



(d) Elapsed time of the bulk TCP flows

Figure 7: Summary of results over the error-free link in the workload of a short competing TCP flow (120 repetitions)

overshoot is avoided.

The elapsed times of the bulk TCP flows are shown in Figure 7d. The slower flow has a very stable elapsed time in all error-free link cases, ending roughly at the same time in most of the repetitions. With ECN, the fairness between the elapsed time of the faster and the slower flow is worse than without. The elapsed time of the faster flow is slightly more stable with ECN than without it.

Baseline TCP

When Baseline TCP is used, the overall throughput is quite stable, the median being 7817 B/s. The median elapsed time of the third flow is 26.1 seconds and the interquartile range is 16.40 seconds. The median elapsed time of the faster flow is 92.4 seconds and the interquartile range of the elapsed time is 10.6 seconds. The median elapsed time of the slower flow is 97.3 seconds and the interquartile range of the elapsed time is 0.14 seconds. The downlink utilization is very high with the median of 97%.

The link is less than fully utilized only during SYN phase, the slow-start phase, and

at the end of the repetition starting from the last FIN-ACK. Most of the missing 3% utilization accumulates during the start of the connection because only one round-trip is required to close the last connection. In the slow start the current congestion window, which requires a round-trip to multiply itself, limits the transfer rate to less than the link bandwidth for several round-trips.

The elapsed time of the slower flow is stable because it takes almost the same time to transfer the workloads data in each case over the wireless link having unchanging bandwidth. Thus the slower flow completes almost at the same time regardless of the completion time of the other flows.

Each repetition includes a slow-start overshoot for the faster and slower flow but it does not reduce the overall throughput despite the large number of packets dropped. However, the drops that occur during congestion cause the difference that is observed between the elapsed times of the bulk TCP flows. In many repetitions the progress of the bulk TCP flows starts to apart, usually slowly but firmly. This unbalance begins to grow immediately after the slow-start overshoot or after congestion drops that affect only one bulk TCP flow.

In Figure 8 we see time sequence graphs for two bulk data TCP flows and for a short competing TCP flow. In this case, the first bulk TCP flow ends slow-start overshoot at 11.7 seconds by retransmitting the first lost segment, which reduces queuing delay from that point on. The slow-start overshoot of the other bulk TCP flow ends at 12.8 seconds. Many losses occur during the round-trip prior to the end of the slow-start overshoot starting roughly at 7.5 seconds. The presented times of the slow start vary very little as long as the the third flow does not start sending data before the drops begin to occur. When the starting time of the third flow is early enough to cause the overlap, the end of the slow-start overshoot occurs slightly earlier than the presented case. We next compare these bulk TCP flow phases to the elapsed time of the third flow that is shown in Figure 9 for individual repetitions. We observe that classes can be extracted based on the starting time, which are presented in Table 7.

Starting time	Min	25%	Median	75%	Max
- 2.5 s	15.27	15.30	17.55	20.24	20.34
2.5 s - 8.2 s	17.53	35.82	37.61	40.04	50.84
8.2 s - 15.5 s	15.37	20.20	20.88	25.12	33.39
15.5 s -	15.72	21.69	30.00	38.69	62.50

Table 7: Elapsed time of the third flow in classes based on the starting time of the third flow

The repetitions that start before 2.5 seconds complete very fast because the third flow competes a fair share of the link bandwidth as the bulk TCP flows are also still in the slow start and there is no congestion yet. A few losses, which occur during the fourth or later round-trip, cannot delay the completion of the transfer more than a

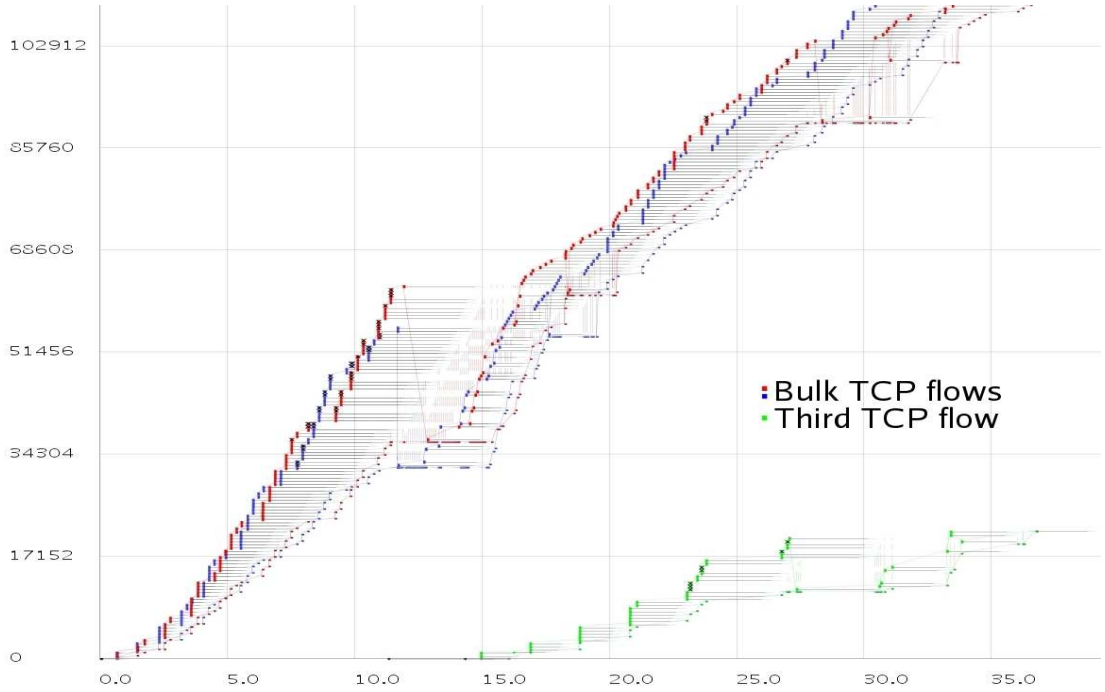


Figure 8: Time sequence graphs for bulk TCP flows and for short competing flow that is starting at 11.3 seconds

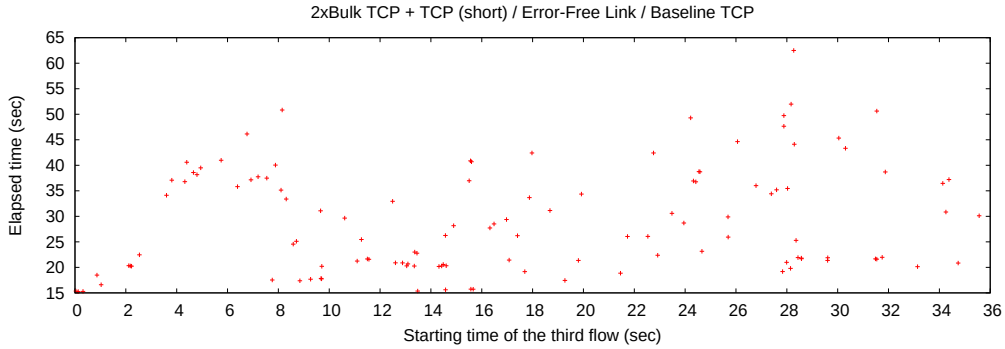


Figure 9: The elapsed time of the third flow with Baseline TCP in all individual repetitions over error-free link (120 repetitions)

few round-trips because they hit the large congestion window that triggers the fast retransmit quickly.

If the third flow starting time is later than 2.5 seconds, it is largely affected by the slow-start overshoot. Multiple packets are dropped from the third or earlier round-trip. The recovery slows down the completion of the transfer because not enough duplicate ACKs are generated from one window of data. The limited transmit helps but at least one additional round-trip is required to trigger the fast retransmit. The aggressiveness of the slow-start phase increases round-trip times rapidly. Because

of that, the third flow can experience a spurious RTO in the initial window but its unnecessary retransmissions are likely to be lost due to the congestion or if not lost they help the loss recovery by generating additional duplicate ACKs.

Next there is a notch during which the third flow is able to complete quickly when the starting time of the third flow is later than 8.2 seconds. The period of the notch starts very close to the point where end of the slow-start overshoot of the bulk TCP flows is less than round-trip away. First the third flow performs handshake with SYNs. SYN-ACK arrives while the bulk TCP flows are in overshoot recovery. If SYN of the third flow is lost because of the losses that occur during the slow-start overshoot of the bulk TCP flow, it is retransmitted later during the overshoot recovery and because the queuing delay is the low, the SYN-ACK for the third flow arrives quickly. The segments of the third flow experience low queuing delay because the bulk TCP flows are recovering during the early round-trips of the third flow. Thus, the third flow does not experience any drops until its congestion window is large enough to handle congestion quickly as can be observed in Figure 8. When starting time of the third flow is slightly below 8.2 seconds, the third flow may complete quickly if the drops caused by the slow-start overshoot of the bulk TCP flows do not hit the initial window of the third flow. On the other hand, if the retransmission of SYN dropped, the third flow suffers due to exponential back-off of RTO.

During the recovery of the bulk TCP flows, the queuing delay first shortens quickly, then it starts to increase again when cumulative ACKs arrive. Thus, the elapsed time climbs back to a higher level. In the repetitions with the starting time later than 15.5 seconds, the performance of the third flow depends greatly on the events that take place during its first round-trips. In the other extreme there are the repetitions were no congestion drops occur for the third flow but only the bulk TCP flows are hit by the drops. In such a repetition, the third flow completes reasonably quickly because the bulk TCP flows slow down. When the congestion drops happen and hit the third flow near its beginning, an early transition to congestion avoidance with a small congestion window is performed and the bulk TCP flows maintain the unfair, large congestion windows. The congestion avoidance prevents the fast growth of the congestion window of the third flow. The small congestion window does not contain enough packets to trigger three duplicate ACKs. Instead, additional round-trips are needed before limited transmit is able to trigger the third duplicate ACK. One such a repetition is shown in Figure 10. With limited transmit, new segments are sent when the duplicate ACKs are received. These new segments maintain the ACK clock of TCP and RTO is not triggered. The congestion window does not increase during these round trips. When the loss recovery finally begins, the already small congestion window is reduced even more. At the same time, the bulk TCP flows transmit with a large congestion window that can recover from a loss faster than the third flow, and then continue with much larger congestion windows than the third flow.

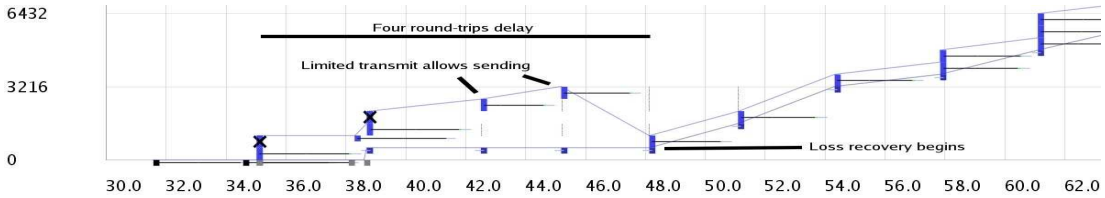


Figure 10: Time sequence graph for the third flow with a small congestion window that delays the start of loss recovery by multiple RTTs

Baseline TCP with ECN

When Baseline TCP is used with ECN, the overall throughput remains roughly the same as without ECN. The median elapsed time of the third flow is 2% shorter, the upper quartile is 13% shorter, and the maximum is 30% shorter than with the Baseline TCP. The interquartile range of the elapsed time of the third flow is 30% smaller because the lower quartile of the elapsed time remained roughly the same. The median elapsed time of the faster flow is 0.7% shorter and the interquartile range of the elapsed time is 13% smaller than with Baseline TCP.

For all flows, the number of packets dropped is almost the same as with Baseline TCP because the third flow is short and the dominating number of drops happen during the slow-start overshoot. It is known beforehand that RED reacts too late to the slow-start overshoot [Kul03]. However, the elapsed times of the third flow are shorter because the third flow experiences less drops and shorter queuing delay. RED prevents the bulk TCP flows from monopolizing the queue of the last-hop router. Therefore the upper quartile and maximum of the elapsed time are clearly shorter than without ECN while the lower quartile and minimum of the elapsed time remain almost the same.

The difference between the median elapsed times of the faster flows and the slower flow is slightly smaller than with Baseline TCP because ECN marks can affect only a single flow at a time. As the affected flow reduces congestion window, the other flow has advantage until ECN mark happens for it. If no ECN marks occur later on, the advantaged flow becomes very likely the faster flow, completing much faster than the slower flow. On the other hand, the elapsed time of the faster flow is more stable than with Baseline TCP because ECN marks reduce a probability of a very short elapsed time for the faster flow.

Next we look why the median elapsed time of the third flow is only 2% shorter than with Baseline TCP. The oscillation of the RED queue average that happens because of the varying router instant queue length easily leads to an additional queue overflow after a slow-start overshoot. Often the second overshoot does not trigger ECN marks or the ECN marks occur after the first drop because the oscillation has lowered the queue average much below the level at which it will settle but the instant queue length has already lengthened. The third flow starting times that are late enough benefit most from ECN because the oscillations of the queue average

attenuate. With the starting times of the third flow above 15 seconds the elapsed time of the third flow is shorter with ECN. However, the aggressiveness of the slow start of the third flow is still able to cause queue overflows, which are not prevented by ECN. When the third flow starting time is near 8 seconds, its transfer is unluckily in its final part when the second overflow takes place. In such a repetition, the third flow cannot use duplicate ACKs for loss detection but is forced to wait for RTO. This increase is visible in the median elapsed time of the third flow, with ECN it is in 25% of the repetitions. Without ECN only 13% of the repetitions are affected. The last window RTO affects significantly the median elapsed time. If the repetitions with a starting time less than 15 seconds are ignored, the median elapsed time of the third flow is 9% shorter than with Baseline TCP.

Enhanced TCP

When Enhanced TCP is used, the median overall throughput is 7872 B/s. The median elapsed time of the third flow is 27% shorter and the interquartile range is 65% smaller than with Baseline TCP. The median elapsed time of the faster flow is 1.5% shorter and the interquartile range of the elapsed time is 2.7% smaller than with Baseline TCP. The median elapsed time of the slower flow is 96.6 seconds. The median link utilization is 98%.

The overall throughput is 0.7% higher than with Baseline TCP because the initial window of four is aggressive in the slow-start phase. Therefore also the link utilization is slightly higher and the slower flow is able to complete slightly faster than with Baseline TCP.

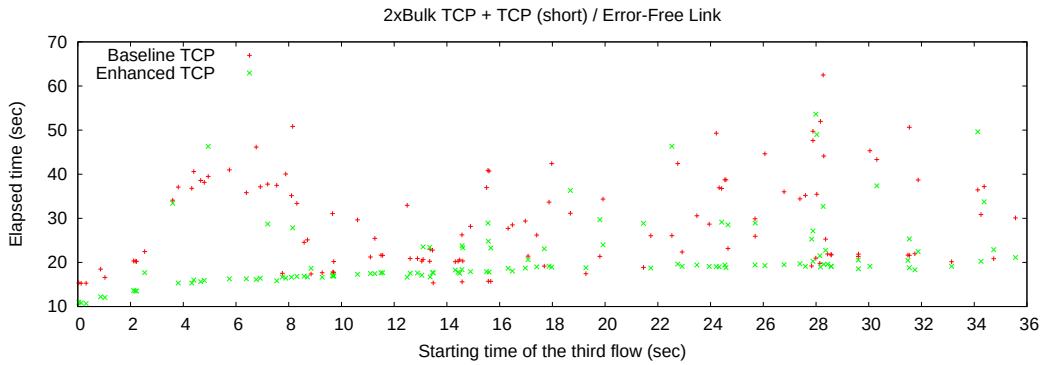


Figure 11: Comparison of the elapsed time of the third flow with Baseline TCP and Enhanced TCP in individual repetitions over error-free link (120 repetitions)

The elapsed time of the third flow against Baseline TCP in individual repetitions is shown in Figure 11. The elapsed time with Enhanced TCP is a smoothly and slowly increasing line that is quite stable below 15 seconds. With CBI, the bulk TCPs smoothly switch from the slow start to the congestion avoidance with ssthresh that is conservatively below the value necessary for the slow start to overshoot. The

difference between the number of total drops for Baseline TCP and for Enhanced TCP is roughly the number of losses that occur during the slow start overshoot because CBI prevents the overshoot related drops. Congestion is very limited and does not occur during the third flow, and therefore the third flow is able to proceed without drops. The spike in the elapsed time of the third flow, which occurs during the slow start of the bulk TCP flows with Baseline TCP, is flattened except in a few repetitions that include the slow-start overshoot because the Linux kernel garbage collects the memory of the initial ssthresh values (see Section 4.5). When the starting time of the third flow later than 15 seconds congestion starts to occur during the third flow because the bulk TCP flows have reached larger congestion windows. After that point the elapsed time of the third flow starts to get dispersed three to ten seconds above the smooth line on which the elapsed times still settle when the third flow is not hit by any of the occurring congestion drops but only the bulk TCP flows are affected.

The congestion drops are few and therefore occur in many repetitions only for one bulk TCP flow. When the other bulk TCP flow is able to proceed without drops, it becomes the faster flow because the congestion window of the slower flow is smaller than the congestion window of the faster flow due to the drops, which increase the difference between the median elapsed times of the bulk TCP flows.

In the repetitions with a starting time above 28 seconds, there is a spurious RTO phenomenon that is explained in greater detail in the workload with a competing bulk TCP flow.

Enhanced TCP with ECN

When Enhanced TCP is used with ECN, the median overall throughput is roughly the same as with Enhanced TCP. The median elapsed time of the third flow is 33% shorter and the interquartile range of the elapsed time is 76% smaller than with Baseline TCP. As with the Baseline TCP with ECN case, ECN is able to reduce the queuing delay and the number of drops of the third flow, which shortens the elapsed times of the upper end. Also in this case the ECN benefits diminish "towards" the minimum elapsed time repetition as was discussed in the Baseline TCP with ECN case.

The median elapsed time of the faster flow is 3.9% shorter and the interquartile range of the elapsed time is 22% smaller than with Baseline TCP. The difference between the elapsed time of the faster flow and the elapsed time of the slower flow is larger than with Enhanced TCP because ECN marks only one of the bulk flows earlier than congestion drops would occur. Therefore the congestion windows are unbalanced for longer duration than with Enhanced TCP. When ECN marks occur for both bulk TCP flows, the difference between the elapsed times of the bulk TCP flows is smaller than with Enhanced TCP, which reduces the interquartile range of the elapsed time of the faster flow.

The elapsed time of the third flow is plotted in Figure 12 for all individual repetitions.

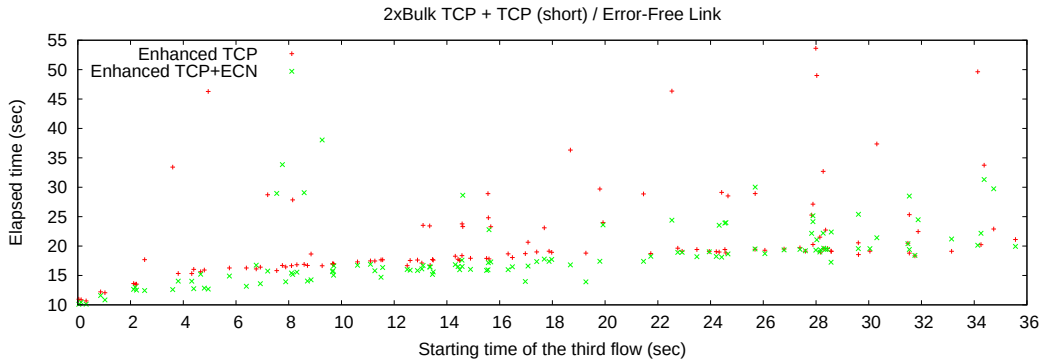


Figure 12: Comparison of the elapsed time of the third flow in the cases of Enhanced TCP with and without ECN in individual repetitions over error-free link (120 repetitions)

Only the repetitions that have a starting time near the end of 36 seconds include ECN marks that shorten the elapsed time of the third flow slightly. In the earlier starting repetitions there is an absence of congestion whose severity RED could reduce. However, there is a small inter-repetition effect which is indirectly caused by ECN. CBI sets smaller initial ssthresh because of the ECN mark that happens near the total elapsed time of the previous repetition. Therefore the amount of other traffic during the third flow is lighter and the third flow is able to complete faster.

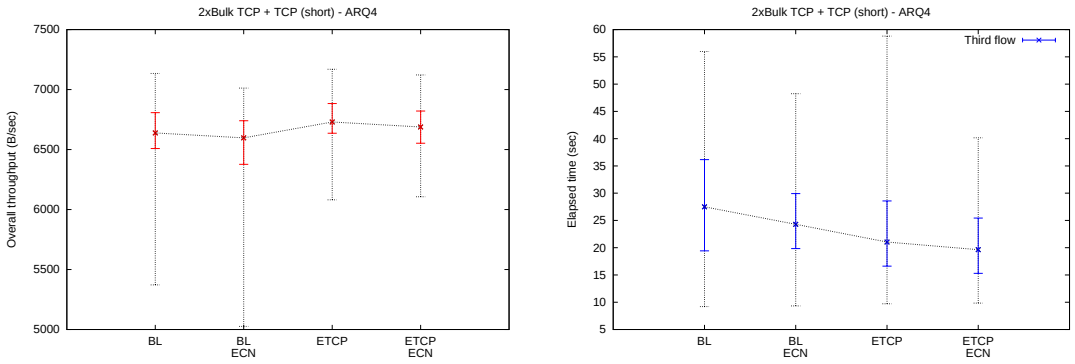
5.1.2 Link with High ARQ Persistency

The overall throughput over the link with high ARQ persistency is shown in Figure 13a. With a lossy link the theoretical maximum of the overall throughput is affected by the error model. Therefore the nominal link bandwidth of 8000 B/s cannot be reached. As with the error-free link, Enhanced TCP has a higher overall throughput than Baseline TCP. With ECN the median of the overall throughput is very slightly lower than without ECN.

The elapsed time of the third flow is shown in Figure 13b. With Enhanced TCP, the elapsed time of the third flow is shorter and more stable than with Baseline TCP. The elapsed time of the third flow is more stable with ECN than without it, and also a slightly improvement in the median elapsed time can be observed.

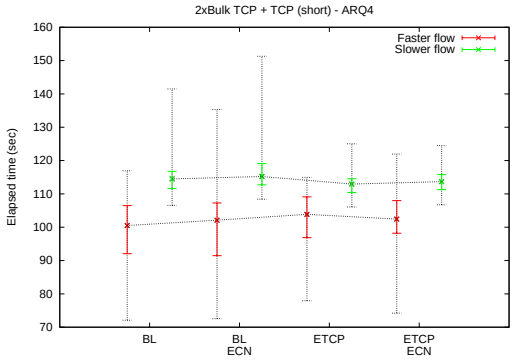
The elapsed times of the two bulk TCP flows are shown in Figure 13c. With Enhanced TCP, the difference between the slower and faster is shorter than with with Baseline TCP. The elapsed time of the slower flow is less stable than with the error-free link.

Figure 13d shows the number of the spurious RTOs and the unnecessary retransmissions. Unnecessary retransmissions occur mainly because of the spurious RTOs. The number of unnecessary retransmission with Enhanced TCP is smaller than with Baseline TCP.

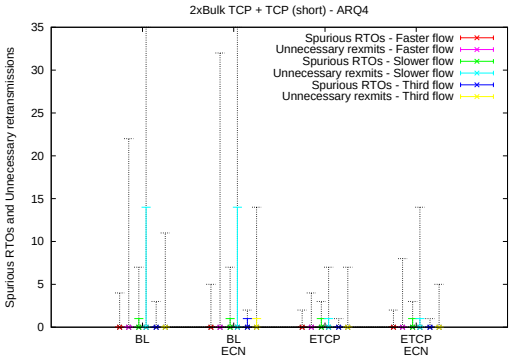


(a) Overall throughput

(b) Elapsed time of the third flow



(c) Elapsed time of the bulk TCP flows



(d) Spurious RTOs and Unnecessary retransmissions (the largest maximums are cropped)

Figure 13: Summary of results over the link with high ARQ persistency in the workload of a short competing TCP flow (120 repetitions)

Baseline TCP

When Baseline TCP is used, the median overall throughput is 6639 B/s. The interquartile range of the overall throughput is 301 B/s that is 27 times larger than with the error-free link. The median elapsed time of the third flow is 5% longer and the interquartile range of the elapsed time is 2% larger than with error-free link. The median elapsed time of the faster flow is 100.5 seconds. The median elapsed time of the slower flow is 114.5 seconds.

The slow-start overshoot is happening earlier because of an earlier queue overflow. There are two reasons for this earlier queue overflow. The link-level retransmission delays group multiple ACKs into a single burst that is delivered back-to-back. On the sender side, the delivery of an ACK burst triggers a transmission of multiple segments, which builds up the last-hop router queue. Secondly, the accumulation of the new packets during the link-level retransmissions increases the last-hop router queue length.

If one of the bulk TCP flows is hit by an spurious RTO near the beginning, the

other bulk TCP flow is able to slow start to a large congestion window because there is much more available space in the last-hop router buffer. When spurious RTO occur, they cause a number of unnecessary retransmissions, which lower the overall throughput.

With a lossy link the total elapsed time varies. Main cause for the variation are the bad state retransmissions at the link layer that stall the link for a moment. In addition, the link may become underutilized when congestion windows are reduced because of congestion that is caused by the burstiness after link-level retransmissions, which lengthens the total elapsed time and reduces overall throughput. Each stall delays the completion of the whole transfer, and even though the link would be fully utilized during link ARQ, TCP makes no progress during the stall. The longer median elapsed times for all flows happen mainly because of the stalls. A similar notch in the elapsed time of the third flow as with the error-free link case happens, but the link ARQ related delays increase variability of the elapsed time regardless of the starting time.

Baseline TCP with ECN

When Baseline TCP is used with ECN, the median overall throughput is 6598 B/s. The interquartile range of the overall throughput is 20% larger than with Baseline TCP. The median elapsed time of the third flow is 11% shorter, the upper quartile is 17% shorter, and the interquartile range of the elapsed time is 40% smaller than with Baseline TCP.

The median overall throughput is slightly lower than with Baseline TCP, but only 0.6%. With ECN the congestion window is reduced earlier than with Baseline TCP due to the ECN mark. Therefore the congestion window tends to be smaller than with Baseline TCP. With such a small congestion window TCP is not always able to fully utilize the link if other flows have just completed. Consequently, the link has nothing to send when the packets are depleted from the link send buffer, which lowers the overall throughput.

The elapsed time of the third flow is reduced similarly as with the error-free link. In addition to the error-free link results, ECN clearly shortens the median elapsed time as the last window drops that occurred with the error-free link do not occur. ECN is able to control the TCP congestion window after the initial overshoot as with the error-free link, but the back-to-back ACK bursts lead to a sudden overflow of the router queue. In the cases where the ACK burst does not occur, ECN shortens the elapsed time of the third flow by preventing the monopolization of the router queue. The improvement is present in the repetitions where the starting time of the third flow is later than the point where the bulk TCPs are about to complete the overshoot recovery. In such case, the bulk TCP flows are forced to make room for the third flow by ECN marks. In addition, when the third flow starts during the slow start of the bulk TCP flows and has a very long elapsed time, ECN marks force bulk TCPs to make room.

When the third flow starts close to the beginning of the overshoot recovery of the bulk TCP flows, the third flow is able to gain a larger sized congestion window right from the beginning as the bulk TCP flows have a reduced congestion window because of a loss recovery. With the larger congestion window, the third flow has a higher throughput and is able to react faster to drops as enough immediate duplicate ACKs are generated. Therefore ECN cannot improve the balance between flows any further from the performance of Baseline TCP.

Enhanced TCP

When Enhanced TCP is used, the median overall throughput is 6730 B/s that is 1.4% higher and the interquartile range of the overall throughput is 17% smaller than with Baseline TCP. The overall throughput is higher because the larger initial window reduces the weight of the slow-start phase in the overall throughput and because F-RTO reduces the number of unnecessary retransmissions.

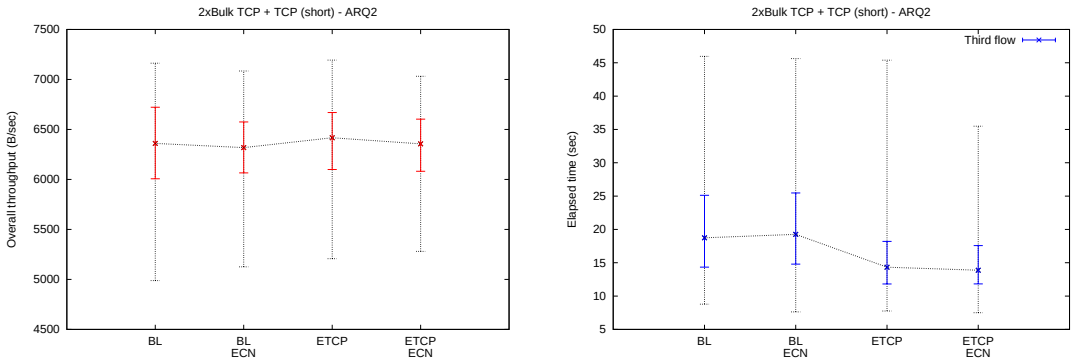
The median elapsed time of the third flow is 24% shorter and the interquartile range of the elapsed time is 29% smaller than with Baseline TCP. As with the error-free link, the median elapsed time of the third flow is shorter because the slow-start overshoot is avoided with CBI. CBI also prevents the unbalanced congestion windows between the faster and slower flow when an spurious RTO occurs during a bad state for one of the bulk TCP flows in the initial slow-start phase. The CBI's ssthresh reusing causes a transition to congestion avoidance also when there is available buffer space at the last-hop router. In addition, with the larger initial window the third flow recovers faster in the cases where congestion occurs near the beginning of the third flow.

Enhanced TCP with ECN

When Enhanced TCP is used with ECN, the median overall throughput is 6688 B/s and the interquartile range of the overall throughput is 11% smaller than with Baseline TCP. The median overall throughput is 0.7% higher than with Baseline TCP but it is less than in the Enhanced TCP case. The median elapsed time of the third flow is 29% shorter and the interquartile range of the elapsed time is 40% smaller than with Baseline TCP.

Over the whole starting time range, ECN avoids very long elapsed times for the third flow because the bulk TCP flows cannot monopolize the last-hop router queue for a long periods of time. An ACK burst is harmful to RED's queue average and hence the overflows exist as is explained with Baseline TCP and ECN. As the third flow is short, congestion happens occasionally near the end of the transfer of the third flow. This causes an ECN mark after the third flow has completed while the congestion is already decreasing. If the third flow would be larger we expect ECN to perform better against a tail-drop.

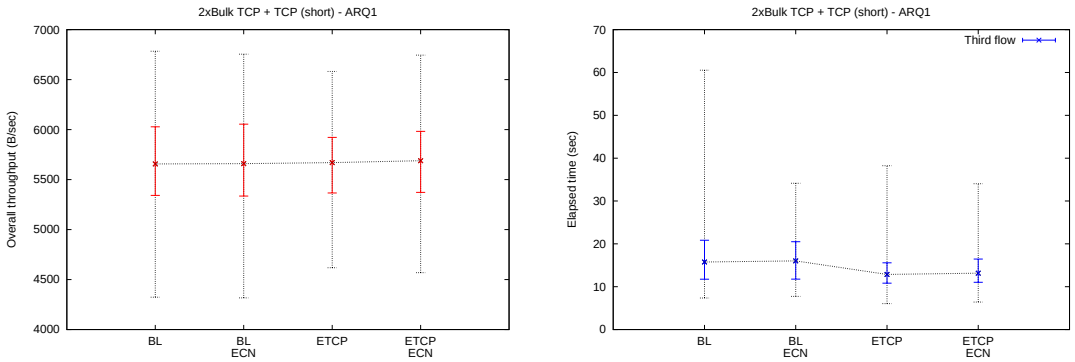
5.1.3 Links with Medium and Low ARQ Persistency



(a) Overall throughput (b) Elapsed time of the third flow

Figure 14: Summary of results over the link with medium ARQ persistency in the workload of a short competing TCP flow (120 repetitions)

The overall throughput over the link with medium ARQ persistency is shown in Figure 14a. The median overall throughput varies as with high ARQ persistency, with ECN the overall throughput is slightly lower than without it and with Enhanced TCP the median overall throughput is higher than with Baseline TCP. Error drops or a spurious RTO in the slow-start phase frequently prevent the overshoot of a single bulk TCP flow from occurring, however, the other bulk TCP flow usually continues to the overshoot. In Figure 14b we see the elapsed time of the third flow. When Enhanced TCP is used, the elapsed time of the third flow is shorter and more stable than with Baseline TCP because the slow-start overshoots of the both bulk TCP flows are avoided with CBI and because the third flow has a larger initial window resisting more drops that are now more frequent due to corruption.



(a) Overall throughput (b) Elapsed time of the third flow

Figure 15: Summary of results over the link with low ARQ persistency in the workload of a short competing TCP flow (120 repetitions)

The overall throughput over the link with low ARQ persistency is shown in Figure 15a. The median overall throughput has a smaller increase with Enhanced TCP than with the other link types. The ECN marks do not occur because the large number of error drops keeps the average queue length low. CBI's reuse ssthresh does not lower the overall throughput despite the error drops that affect the congestion windows. In Figure 15b we see the elapsed time of the third flow. As with medium ARQ persistency, the Enhanced TCP enables the third flow to complete faster and its elapsed time is more stable than with Baseline TCP.

5.1.4 Summary of Results

The slow start of the bulk TCP flows overshoots with Baseline TCP, when there are only few or no error drops. When the residual error rate increases, the slow-start overshoot becomes less frequent. With the initial window of two it takes couple of round-trips to fully utilize the wireless link. When the third flow starts during the later phases of the slow start of the bulk TCP flows, its elapsed time is likely to be long because of congestion drops. The queuing delay during the slow-start overshoot recovery is small, which enables the third flow to complete quickly if it is starting during the recovery. After the slow-start overshoot of the bulk TCP flows, there is large difference in the elapsed times of third flow between individual repetitions because of congestion. In the worst case the third flow is unlucky and both packets from the initial window are dropped resulting in very slow completion of the third flow. When the congestion drops touch only the bulk TCP flows, the elapsed time of the third flow is good.

With ECN the third flow is able to open its congestion window faster because the last-hop router queue length is shorter. Therefore ECN helps the third flow to complete faster and more stably. The fairness between the two bulk TCPs is reduced due to ECN marks that tend to mark only one of the bulk TCP flows. The link-level retransmissions group ACKs into back-to-back bursts that increase last-hop queue overflows, which ECN cannot prevent. When the residual bit-error rate increases, the ECN marks become rare because of the error drops that cause early congestion window reductions.

The bulk TCP flows of the Enhanced TCP with larger initial window start faster and reach quicker the state where the wireless link is fully utilized. The additional packets in the initial window also help the third flow to recover faster. CBI provides two main advantages. First, the slow-start overshoot, which is very damaging to the third flow, is avoided. Secondly, the lower slow-start threshold prevents the another bulk TCP flow from slow starting above a "fair share" when an error drop or a spurious RTO forces the other bulk TCP flow to the congestion avoidance in an early phase. CBI's ssthresh reuse does not ruin the performance of TCP when error drops occur. F-RTO reduces the number of unnecessary retransmission when spurious RTOs are triggered, but such cases do not occur often. In summary, Enhanced TCP performs well.

All enhancements offer a better performance for the third flow compared with Baseline TCP. The best performance for the third flow is achieved in the Enhanced TCP with ECN case. A single ECN mark or drop is always unilateral affecting only one of the flows, which reduces the fairness of the bulk TCP elapsed times in all configurations having ECN compared with Baseline TCP.

5.2 Two Bulk TCP Flows Competing with a Bulk TCP Flow

In the workload of the two bulk TCP flows competing with a bulk TCP flow, two bulk TCP flows are started at zero seconds. We refer to them as faster and slower flow. The competing bulk TCP flow that starts later is labeled as the third flow.

We report the performance of the whole workload using the overall throughput. The value includes headers and therefore its theoretical maximum is 8000 B/s. In the analysis of individual flows we use the simultaneous throughput (see Section 4.7) as the basic metric because it includes only the phase of the transfer during which all three flows are transferring. The ignored phases have varying amount of competition that is not equal between repetitions and therefore we ignore those phases. We also calculate the fairness index over the simultaneous throughputs.

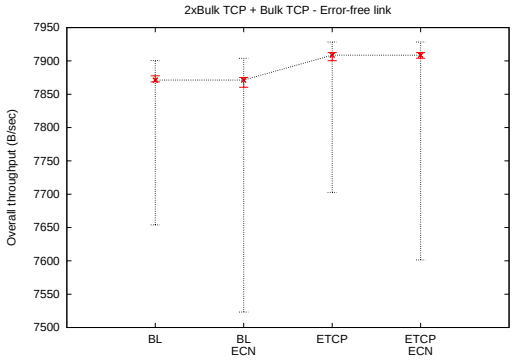
For each link type we use figures that compare the results with Baseline TCP (BL), Baseline TCP with ECN (BL+ECN), Enhanced TCP (ETCP), and Enhanced TCP with ECN (ETCP+ECN). We analyze the cases in that order. The figures show the median and the quartiles with a solid errorbar for the given metrics. When present, a dotted errorbar shows the minimum and the maximum.

5.2.1 Error-Free Link

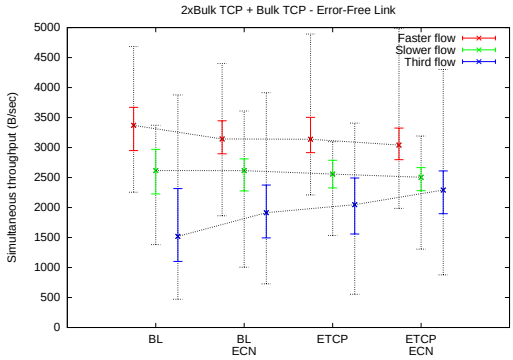
In Figure 16a we see the overall throughput over the error-free link. In all cases the overall throughput is very stable. As with the short competing TCP flow, the Enhanced TCP cases have a higher overall throughput than the Baseline TCP cases.

The simultaneous throughputs of the individual TCP flows over the error-free link are shown in Figure 16b and the fairness index for the simultaneous throughputs in Figure 16c. In the Enhanced TCP cases the simultaneous throughput is more fair than in the Baseline TCP cases. Also ECN improves fairness between the flows. The simultaneous throughput of the third flow is higher both with Enhanced TCP and ECN.

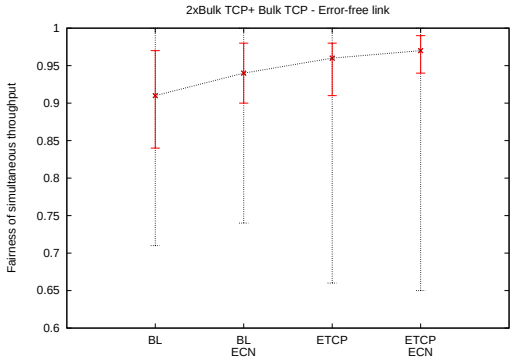
In Figure 16d we see the number of packet drops for each flow. The slow-start overshoot is avoided with Enhanced TCP and therefore the number of drops is very low. ECN avoids one to two drops from every flow.



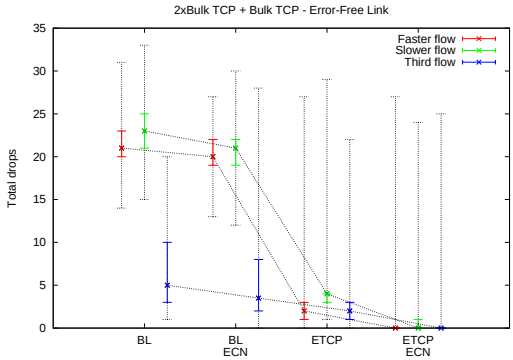
(a) Overall throughput



(b) Simultaneous throughput for individual flows



(c) Fairness index of the simultaneous throughput



(d) Packets dropped

Figure 16: Summary of results over the error-free link in the workload of competing bulk TCP flow (120 repetitions)

Baseline TCP

When Baseline TCP is used and the size of a competing TCP flow is increased from short to long, the median overall throughput increases to 7871 B/s. The median simultaneous throughput is 3370 B/s for the faster flow, 2617 B/s for the slower flow, and 1520 B/s for the third flow. The interquartile range of the third flow is 1214 B/s. The median fairness index of the simultaneous throughput is 0.91, and the interquartile range of the index is 0.97-0.84. The link utilization is 98%.

The total elapsed time is longer than with the short competing TCP flow because the amount of data that is transferred over the link is larger. A longer total elapsed time reduces the weight of the slow-start phase and therefore the median overall throughput is higher than in the workload with the short competing TCP flow.

Because of the similar congestion as with the workload of the short competing TCP flow, the third flow is very often penalized near to its start. This penalty prevents the third flow from opening the congestion window to the size equal to the size of the other flows. In the repetitions where one of the zero-starting flows suffers

because of congestion drops, the third flow attains simultaneous throughput equal to the simultaneous throughput of the slower flow. The faster flow, however, is almost always served unfairly well. Figure 17 plots the simultaneous throughput of the third flow for individual repetitions. As with the short competing TCP flow, the worst starting times for the third flow are during the slow-start overshoot and after the last-hop router queue has grown again. In the latter case, the zero-starting flows have completed their overshoot recovery and are firmly in the congestion avoidance.

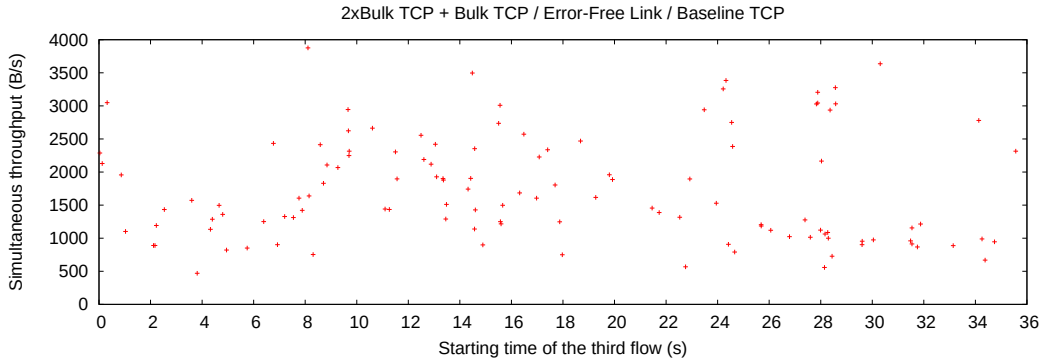


Figure 17: The simultaneous throughput of the third flow with Baseline TCP in individual repetitions over error-free link (120 repetitions)

Next we look at the repetitions where the third flow starts while the other flows are in the congestion avoidance in greater detail. Those repetitions have two categories that are based the round-trip on which the slow-start overshoot for the third flow occurs. In many repetitions the slow-start overshoot occurs during the third or earlier round-trip and the congestion window of the third flow is left small. When the slow start is able to continue to the fifth or later round-trip, the third flow reaches a higher simultaneous throughput because it is able to inflate the congestion window to a large enough size to even with the zero-starting bulk TCP flows. These two starting patterns cover almost all repetitions. The large difference in those two categories reduces the stability of the simultaneous throughput of the third flow.

Baseline TCP with ECN

When Baseline TCP is used with ECN, the median overall throughput is roughly the same as with Baseline TCP. The median simultaneous throughput is 3144 B/s for the faster flow, 2615 B/s for the slower flow, and 1915 B/s for the third flow. The slower flow yields roughly the same throughput as with Baseline TCP while the faster flow has 7% lower and the third flow has 26% higher simultaneous throughput than with Baseline TCP. The interquartile ranges of the simultaneous throughputs are smaller than with Baseline TCP, the faster flow by 24%, the slower flow by 28%, and the third flow by 28%. The median fairness index of the simultaneous throughput is 0.94, and the interquartile range of the index is 0.98-0.90.

ECN slows the flows that try to monopolize the last-hop router queue by marking them. Therefore fairness between the flows increases. The lower and upper quartiles of the simultaneous throughput of the individual flows are packed closer together and also the medians of the different flows draw closer each other. The simultaneous throughput of the third flow increases with ECN regardless of the third flow's starting time.

We look now at the repetitions where the third flow is started during the congestion avoidance of the other flows, which was problematic with Baseline TCP. The simultaneous throughput of the third flow is higher in the repetitions where the slow-start overshoots during an early round-trip because ECN marks prevent the other flows from keeping the larger bandwidth share, which they steal from the third flow's fair share, for a long period of time. When ECN has marked before the entry of the third flow, the shorter queue length increases probability that the third flow overshoots on the fifth or later round-trip. In such a repetition, the third flow has a very good simultaneous throughput as with Baseline TCP.

Enhanced TCP

When Enhanced TCP is used, the median overall throughput increases to 7909 B/s because of the larger initial window. The median simultaneous throughput is 3138 B/s for the faster flow, 2558 B/s for the slower flow, and 2048 B/s for the third flow. The interquartile ranges of the simultaneous throughputs are smaller than with Baseline TCP, the faster flow by 18%, the slower flow by 38%, and the third flow by 23%. The median fairness index of the simultaneous throughput is 0.96, and the interquartile range of the index is 0.98-0.91. The median downlink utilization is 99%.

The absence of the slow-start overshoot helps the third flow that is started during the slow-start phase of the other flows with Baseline TCP. The third flow is able to increase its congestion window to a value that is large enough to handle the losses, which occur later, effectively with the fast recovery. As a result, the fairness of the simultaneous throughput is also better.

With Enhanced TCP, a slow-start overshoot caused by a CBI garbage collection (see Section 4.5) is present in 33% of the repetitions having the starting time of the third flow between 24-36 seconds. In cases where CBI reusing are not garbage collected, the round-trip time increases smoothly as the zero-starting flows switch to congestion avoidance before the slow-start overshoot occurs. When the round-trip time has grown above three seconds before the entry of the third flow, a spurious RTO happens in the initial window for the third flow because the initial RTO for a starting TCP flow is three seconds. As the number of packets in flight increases only by the initial window of the third flow, the other flows do not experience congestion and continue with the large congestion windows. In such a repetition, the fairness of the simultaneous throughput is poor because the spurious RTO limits the growth of the congestion window of the third flow. This limit is set by the selected F-RTO

response which halves the slow-start threshold as a precaution [SKR03].

Enhanced TCP with ECN

When Enhanced TCP is used with ECN, the overall throughput remains roughly the same as with Enhanced TCP. The median simultaneous throughput is 3042 B/s for the faster flow, 2505 B/s for the slower flow, and 2292 B/s for the third flow. The interquartile ranges of the simultaneous throughputs are smaller than with Baseline TCP, the faster flow by 27%, the slower flow by 48%, and the third flow by 41%. The median fairness index of the simultaneous throughput is 0.97. The interquartile range of the index is 0.99-0.94.

ECN improves the simultaneous throughput of the third flow by marking unfairly well served flows with a high probability which reduces the unbalance of the congestion windows as in the case of Baseline TCP with ECN. The phenomenon that occurred with Enhanced TCP when the round-trip time grew above three seconds is countered by the ECN mark that reduces the last-hop router queue length before the round-trip time rises above three seconds.

5.2.2 Link with High ARQ Persistency

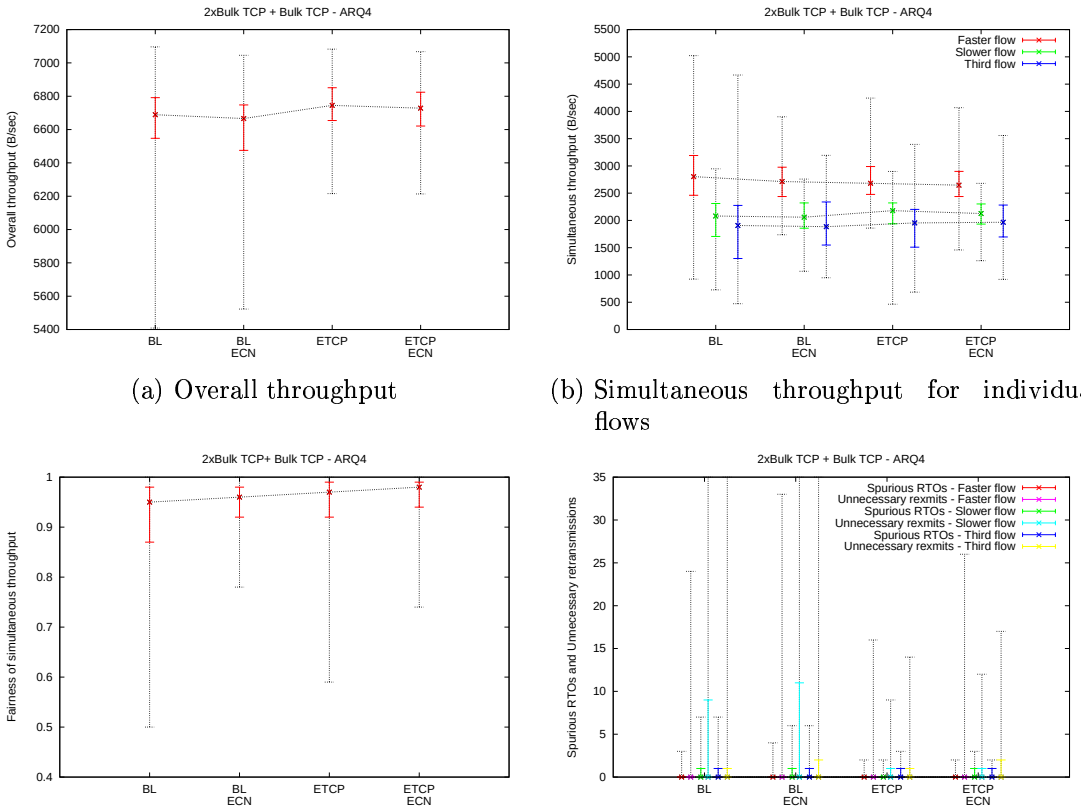
In Figure 18a we see the overall throughput over the link with high ARQ persistency. As with the error-free link, the overall throughput is higher with Enhanced TCP than with Baseline TCP. The interquartile ranges are larger than with the error-free link.

In Figure 18b is the simultaneous throughput for the bulk TCP flows and in Figure 18c is the fairness index for the simultaneous throughputs. ECN improved fairness between the flows, especially very unfair repetitions are affected. When Enhanced TCP is used, the median and the lower quartile of the fairness index are improved against Baseline TCP.

The number of spurious RTOs and unnecessary retransmissions are shown in Figure 18d. In Enhanced TCP, F-RTO reduces the number of unnecessary retransmissions.

Baseline TCP

When Baseline TCP is used, the median overall throughput is 6689 B/s. The interquartile range of the overall throughput is 244 B/s. The median simultaneous throughput is 2804 B/s for the faster flow, 2081 B/s for the slower flow, and 1907 B/s for the third flow. When compared with the error-free link results, the median simultaneous throughput of the faster flow is 19% lower, the slower flow is 21% lower, and the third flow is 26% higher. The interquartile ranges of the simultaneous throughputs of the slower and third flow are 20% smaller than with the error-free link, while the faster flow has 1% larger interquartile range. The median fairness



(a) Overall throughput

(b) Simultaneous throughput for individual flows

(c) Fairness index of the simultaneous throughput

(d) Spurious RTOs and unnecessary retransmissions (the largest maximums are cropped)

Figure 18: Summary of results over the link with high ARQ persistency in the workload of competing bulk TCP flow (120 repetitions)

index of the simultaneous throughput is 0.95, and the interquartile range of the index is 0.98-0.87.

The median overall throughput is lower and the interquartile range of the overall throughput is larger than with the error-free link because of ARQ-related delays. In addition, unnecessary retransmissions lower the overall throughput very slightly compared with the reduction caused by the ARQ-related delays.

The median of the fairness index over the simultaneous throughput is better and the simultaneous throughputs for slower and for third flow are more stable than with the error-free link because of error drops and periodic congestion. Two main reasons explain the cause of this congestion. First, during ARQ-related delays arriving segments increase the queue length of the last-hop router. Secondly, an ARQ-related delay groups many cumulative ACKs, which triggers a burst of new TCP segments when delivered. With three bulk TCP flows, congestion occurs more frequently than in the workload of the short competing TCP flow and two bulk TCP flows due to larger amount of traffic. After a long, uninterrupted period of successful transfer, the leading flow is likely to experience either kind of drop that slows it

down. Because of additive increase, multiplicative decrease behavior of TCP, the flow that has the largest congestion window is slowed down more than the other flows with high probability. The slowdown increases fairness within the quartiles of the simultaneous throughput. The cases near the maximum and minimum simultaneous throughputs, however, include repetitions with unfair performance.

Spurious RTOs happen for the slower and third flow. Most of the spurious RTOs happen because of the link-level retransmissions during a bad state. However, in a few of the repetitions with a later starting time of the third flow, the first round-trip is longer than three seconds and spurious RTOs are triggered because three seconds is the initial RTO of a starting TCP flow. The spurious RTOs cause unnecessary retransmissions. The upper quartile of the slower flow has nine, and the upper quartile of the third flow has one unnecessary retransmission. The maximum number of unnecessary retransmissions is considerably larger than the upper-quartile, 24 packet for the faster flow, 53 packets for the slower flow, and 57 packets for the third flow. With 64000 bps link transmitting 57 TCP segments that are 576 bytes each takes 4.1 seconds. All unnecessary retransmissions reduce overall throughput.

Baseline TCP with ECN

When Baseline TCP is used with ECN, the median overall throughput is 6666 B/s and the interquartile range is 273 B/s that is 12% larger than with Baseline TCP. The median simultaneous throughput is 2714 B/s for the faster flow, 2059 B/s for the slower flow, and 1886 B/s for the third flow. The median simultaneous throughputs are lower than with Baseline TCP, the faster flow by 3% and the slower and third flow by 1%. The interquartile range of the simultaneous throughput of the third flow is 19% smaller than with Baseline TCP. The median fairness index of the simultaneous throughput is 0.96, and the interquartile range of the index is 0.98-0.92.

The improvement against Baseline TCP in the simultaneous throughput of the third flow is only a fraction of what was observed with the error-free link. However, when we look at the numbers, the median simultaneous throughput of the third flow is very close to the value that was measured with the same configuration over error-free link. We see that the fairness index of the Baseline TCP case is already better than with the error-free link because of the link conditions. Therefore ECN cannot improve fairness more with this link type.

The upper quartile of the unnecessary retransmissions for the slower flow is 11 packets and for the third flow is two packets. The shorter queuing delay, which RED provides, reduces the round-trip time with which a flow enlarges the congestion window to a larger value than with Baseline TCP. The larger window is then unnecessarily retransmitted.

Enhanced TCP

When Enhanced TCP is used, the median overall throughput is 6745 B/s and the interquartile range of the overall throughput is 19% smaller than with Baseline TCP. The median of the simultaneous throughput is 2681 B/s for the faster flow, 2179 B/s for the slower flow, and 1955 B/s for the third flow. The interquartile range of the simultaneous throughput of the third flow is 29% smaller than with Baseline TCP. The median fairness index of the simultaneous throughput is 0.97, and the interquartile range of the index is 0.99-0.92.

As with the error-free link, the slow-start overshoot is not happening which improves the fairness index of the simultaneous throughput.

F-RTO eliminates the effects of the spurious RTOs. The upper quartiles of the slower and third flow have both only one unnecessarily retransmitted packet, and the maximum of the unnecessary retransmissions is 16 packets for the faster flow, nine packets for the slower flow, and 14 packets for the third flow. The smaller number of unnecessary retransmissions increases the median overall throughput together with the larger initial window.

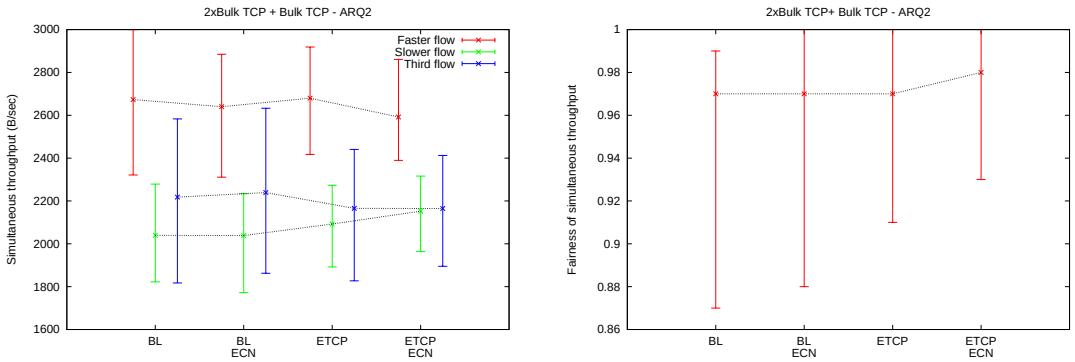
Enhanced TCP with ECN

When Enhanced TCP is used with ECN, the median overall throughput is 6728 B/s, and the interquartile range is 17% smaller than with Baseline TCP. The median simultaneous throughput is 2648 B/s for the faster flow, 2128 B/s for the slower flow, and 1967 B/s for the third flow. The interquartile range of the third flow is 40% smaller than with Baseline TCP. The median fairness index of the simultaneous throughput is 0.98, and the interquartile range of the index is 0.99-0.94. The upper quartile of the unnecessary retransmissions is one packet for the slower flow and two packets for the third flow.

With ECN marks, the queue length of the last-hop router is under the control of RED. A single flow cannot monopolize the queue for a long period of time. Therefore the fairness index of the simultaneous throughput is better.

5.2.3 Links with Medium and Low ARQ Persistency

Figure 19a shows the simultaneous throughputs of the individual flows. As the lossy link with medium ARQ persistency is on the edge where the dominance of the congestion and error-related drops is not strongly on either side, ECN is not able to improve all repetitions. The largest change in the simultaneous throughput can be observed in the fastest flow because it loosely relates to the severity of congestion. The ECN marks constrain the high-end of its simultaneous throughput. The Figure 19b that shows 10th percentile, the median, and 90th percentile of the fairness index over the simultaneous throughputs. The lower end of the index is slightly better in the cases with ECN. These benefits in the fairness index and



(a) Simultaneous throughput for individual flows

(b) Fairness index of the simultaneous throughput (10th and 90th percentiles)

Figure 19: Summary of results over the link with medium ARQ persistency in the workload of competing bulk TCP flow (120 repetition)

the reduction in the simultaneous throughputs high-end are likely related because when the faster flow has a high simultaneous throughput, the fairness index is likely poor. The quartiles of the fairness index over simultaneous throughputs are roughly the same regardless of ECN (See [Jär06] pages 4–6). Similarly as ECN, Enhanced TCP improves only the repetitions with poor fairness. On the lossy link with low ARQ persistency the ECN marks can happen only shortly after the slow start when prior error drops do occur. Even though a few ECN marks occur, the simultaneous throughputs stay roughly the same. CBI reusing ssthresh prevents the slow-start overshoot and therefore with Enhanced TCP ECN marks occur only in two cases that both have slow-start overshoot due to CBI garbage collection.

With the higher residual error drop rate, F-RTO cannot always discern a spurious RTO from a necessary one. If the link drops the packet which is then next in sequence after the segment on which the RTO was triggered, the F-RTO algorithm is not able to use its heuristics.

5.2.4 Summary of Results

The large amount of data increases the total elapsed time, which in turn reduces the weight of the slow-start phase in the overall throughput. Therefore the overall throughput is higher than with the workload of a short competing TCP flow.

As expected based on the workload of a short competing TCP flow, ECN improves fairness among the flows because without slow-start overshoot being present in Enhanced TCP, the third flow is able to open congestion window fairly. Only when the round-trip time is longer than three seconds during the first round-trip of the third flow because of the other flows, a spurious RTO is trigger for the third flow. An ECN mark solves also that spurious RTO issue. The error drops of the lossy link increase fairness of the Baseline TCP case so that ECN has smaller room for

improvement than with the error-free link. ECN is still beneficial with medium ARQ persistency but its effect is limited. With low ARQ persistency the error drops are a dominant factor in the performance of TCP, and ECN does not benefit either of the TCP variants. Spurious RTO that occur cause slight reduction in the overall throughput. Enhanced TCP is useful against them due to F-RTO that prevents the unnecessary retransmission of the whole window.

5.3 Two Bulk TCP Flows Competing with an HTTP Transfer

In the workload with a competing HTTP transfer, there are two bulk data TCP transfers and one HTTP transfer that has a body and eight inline objects. The bulk TCP flows are started at zero seconds. We refer to them as the faster and slower flow.

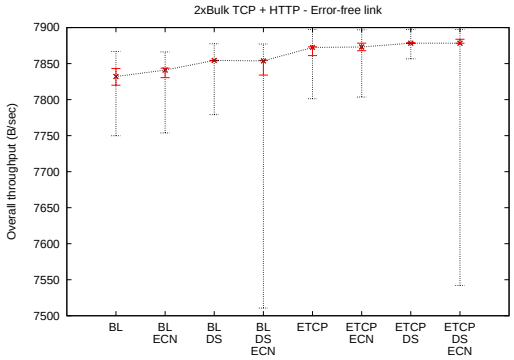
We report the performance of the whole workload using the overall throughput. It is calculated only for the downlink direction. Because the HTTP request are transferred to uplink direction, they are not taken into account. The value includes headers and therefore its theoretical maximum is 8000 B/s. With the HTTP transfer, the main focus is on the response time that is very important from the user perspective. The other TCP metrics, which are not visible to the user, are used in the analysis to explain occurring phenomenon. The number of drops, spurious RTOs and unnecessary retransmissions are reported over all nine TCP connections of the HTTP transfer. For the faster and slower flows, elapsed times are used in the analysis. In addition, the simultaneous throughputs are reported.

For each link type we use figures that compare the results with Baseline TCP (BL), Baseline TCP with ECN (BL+ECN), Baseline TCP with DiffServ prioritization for HTTP (BL+DS), Baseline TCP with ECN and DiffServ prioritization for HTTP (BL+DS+ECN), Enhanced TCP (ETCP), Enhanced TCP with ECN (ETCP+ECN), Enhanced TCP with DiffServ prioritization for HTTP (ETCP+DS), and Enhanced TCP with ECN and DiffServ prioritization for HTTP (ETCP+DS+ECN). We analyze the cases in that order. Since prioritization with this workload is only for HTTP, we refer to it simply as prioritization. The figures show the median and the quartiles with a solid errorbar for the given metrics. When present, a dotted errorbar shows the minimum and the maximum.

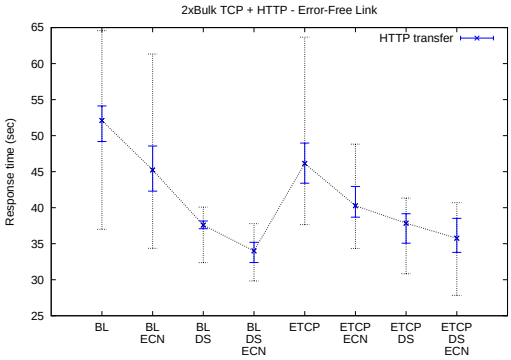
5.3.1 Error-Free Link

The overall throughput over the error-free link is shown in Figure 20a. The overall throughput is stable in all cases, but the minimums of the prioritization with ECN cases have a lower throughput than the other cases. The Enhanced TCP cases have a higher overall throughput than the Baseline TCP cases.

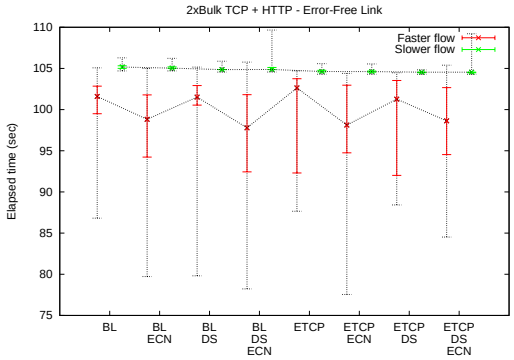
The response times of the HTTP transfer are shown in Figure 20b. Both prioritization and ECN reduce the response time. The shortest response times are achieved when both of them are combined, however; the stability is not as good as with prior-



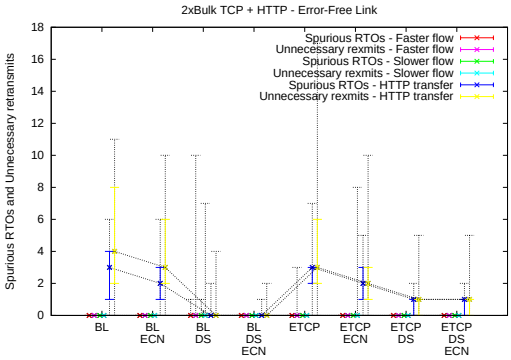
(a) Overall throughput



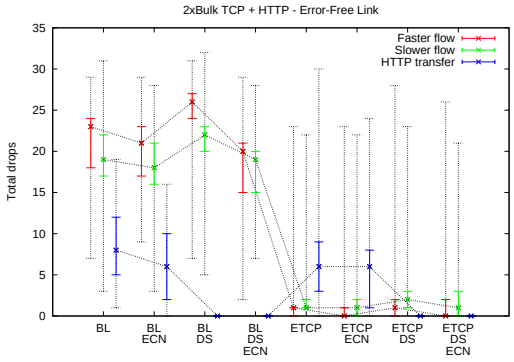
(b) HTTP Response time



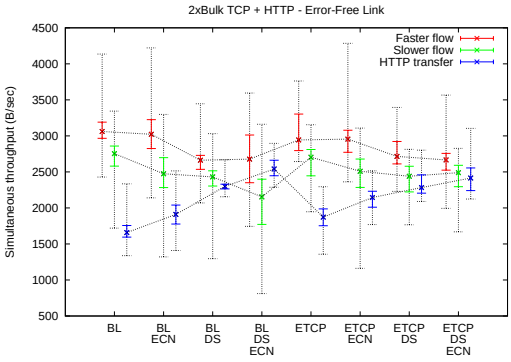
(c) Elapsed time of the bulk TCP flows



(d) Spurious RTOs and unnecessary retransmissions (for HTTP total over 9 connections)



(e) Packets dropped (for HTTP total over 9 connections)



(f) Simultaneous throughputs

Figure 20: Summary of results over the error-free link in the workload of a competing HTTP transfer (120 repetitions)

itization without ECN. Without prioritization the response times are shorter in the Enhanced TCP cases than in the Baseline TCP cases, whereas with prioritization the Baseline TCP cases have a shorter response time.

The elapsed times of the bulk TCP flows are shown in Figure 20c. As with the competing TCP flow workloads, the elapsed time of the slower flow is very stable

because of the error-free link that is mostly fully utilized. The median elapsed time of the faster flow is shorter with ECN than without ECN, which reduces fairness between the bulk TCP flows. ECN affects the stability of the elapsed time of the faster flow, but the direction is controversial. With Baseline TCP, the ECN cases are less stable than the non-ECN cases, whereas with Enhanced TCP an opposite happens.

The number of spurious RTOs and unnecessary retransmissions are shown in Figure 20d and the number of drops are shown in Figure 20e for the bulk TCP flows and for the HTTP transfer. ECN reduces drops in all cases against the same configuration without ECN. Also the number of spurious RTOs, and hence the number of unnecessary retransmissions are reduced. As with the workloads of a competing TCP flow, the median of drops in the Baseline TCP cases is larger than in the Enhanced TCP cases. In the Enhanced TCP cases without prioritization, F-RTO reduces the number of unnecessary retransmissions. With prioritization, there are not drops for the HTTP transfer and spurious RTOs occur only when Enhanced TCP is used.

The simultaneous throughputs of all flows are shown in Figure 20f. Without prioritization the simultaneous throughput of the HTTP transfer is below the simultaneous throughput of the slower flow. With prioritization the simultaneous throughput of the slower flow and the HTTP transfer are close to each other but the faster flow is still served better than the HTTP transfer. When prioritization is used with ECN, the simultaneous throughput of the HTTP transfer is almost leveled with the faster flow.

Baseline TCP

When Baseline TCP is used, the median overall throughput is 7832 B/s. The median HTTP response time is 52.1 seconds. The median elapsed time of the slower flow is 105.2 seconds. The median elapsed time of the faster flow is 101.6 seconds, and the interquartile range of the elapsed time is 3.4 seconds. The median simultaneous throughput is 3062 B/s for the faster flow, 2754 B/s for the slower flow, and 1658 B/s for the HTTP transfer. For the HTTP transfer, the median spurious RTOs is three, and the interquartile range of the spurious RTOs is three. The median unnecessary retransmissions is four packets, and the interquartile range of the unnecessary retransmissions is six packets.

A theoretical HTTP transfer without drops is as follows. First the body is requested. When the first segment of the body has arrived, the WWW browser starts three inline objects because the maximum number of concurrent HTTP objects is limited to four. The first segment of the body is followed by the rest of the initial window segments shortly. When SYN-ACKs of the three inline objects arrive, the transfer of the body has started its second round-trip (SYN round-trip is not included). Because the SYN-ACKs arrive close to each other, the HTTP inline objects send their initial windows also very close to each other, and at the same time, the body

has the number of segments inflight that is determined by its current congestion window. This packet burst increases round-trip times and causes congestion at the last-hop router. We refer to it as *inline object burst*. After each round-trip, this same burst occurs again because the ACKs of the HTTP objects arrive with short intervals and the slow start rapidly inflates the congestion windows of the HTTP objects. When any of the HTTP objects completes, the transfer of a new inline is started until all eight inline objects have been started.

The early round-trip problems that occur with the workload of a short competing TCP flow affect also the HTTP objects. Especially SYN-ACK losses or the losses in the initial window slow down the HTTP objects. These losses happen mainly because of the congestion that is generated by the HTTP transfer itself or by the slow-start overshoot of the bulk TCP flows. In the repetitions where drops do not occur, queuing at the last-hop router increases the round-trip times to the level that triggers spurious RTOs. 93% of the repetitions have at least one spurious RTO. Each spurious RTO requires retransmission of the whole window, very often unnecessarily.

	Body	Inline objects								All
		1	2	3	4	5	6	7	8	objects
Spurious RTOs	63	57	51	58	3	3	4	39	31	309
Packets dropped	243	139	206	206	75	60	48	42	6	1025

Table 8: Total number of spurious RTOs and total number of congestion drops for each HTTP object over individual repetitions (120 repetitions)

The total number of spurious RTOs and congestion drops for the HTTP objects over all repetitions are shown in Table 8. In greater detail, in the repetitions where the starting time of the HTTP transfer is 4-9 seconds, many repetitions have a spurious RTO for the initial window of the HTTP body because of the round-trip time that is long during the slow-start overshoot of the bulk TCP flows. When the starting time is after 20 seconds, all repetitions include one spurious RTO for the initial window of the HTTP body because the queuing delay has grown long enough after the overshoot. The spurious RTOs are very common also for the inline objects that are starting in the first inline object burst because the round-trip times easily exceed the initial RTO value of three seconds. Only the repetitions where the first inline objects send their initial window during the overshoot recovery of the bulk TCP flows have a short enough round-trip time to not trigger spurious RTOs. The bulk TCP flows reduce their congestion windows because of the congestion that occurs during the inline object burst. At the same time, typically at least some of the HTTP objects do not hit the full queue during the congestion and complete, which triggers new HTTP objects. As the congestion windows of the bulk TCP flows are now smaller, these following HTTP objects experience a shorter queue length, and therefore they experience spurious RTOs only in a few repetitions and less likely proceed to the slow-start overshoot before the transfer of the object ends. The number of spurious RTOs increases again for the last two inline objects because the

queuing delay grows again. Only few drops occur for the later inline objects because the bulk TCP flows have adjusted their congestion windows.

Baseline TCP with ECN

When Baseline TCP is used with ECN, the median overall throughput is 7841 B/s and the interquartile range of the overall throughput is 44% smaller than with Baseline TCP. The median HTTP response time is 13% shorter but the interquartile range of the response time is 27% larger than with Baseline TCP. The median elapsed time of the faster flow is 2.7% shorter and the interquartile range of the elapsed time is 125% larger than with Baseline TCP. The elapsed time of the slower flow is roughly the same as with Baseline TCP. For HTTP transfer, the median spurious RTOs is two and the median unnecessary retransmissions is three packets. The median simultaneous throughput is 3023 B/s for the faster flow, 2475 B/s for the slower flow, and 1910 B/s for the HTTP transfer.

The slightly higher and more stable overall throughput compared with Baseline TCP happens due to changes in the number of unnecessary retransmissions. Because less unnecessary retransmissions are sent, the overall throughput is higher and the smaller interquartile range of the unnecessary retransmissions makes the overall throughput more stable.

The median HTTP response time is shorter because RED constrains the queuing delay. With shorter queuing delay round-trips are shorter, less packets are dropped, and less spurious RTOs occur. The reduction of the spurious RTOs is reflected to the number of unnecessary retransmissions, which is also reduced. However, the drops and the spurious RTOs are not radically reduced. The segments of the inline burst cause sudden variation in the last-hop router instant queue length, and ECN marks that are based on the averaged queue length do not occur before congestion in over half of the repetitions. The TCPs respond to the congestion by reducing the congestion windows while almost at the same time all three inline objects complete. The smaller congestion windows of the bulk TCP flows and ending transfers shorten the queue of the last-hop router. The next three new inline objects produce another congestion spike shortly. This oscillation keeps the queue average low enough to prevent ECN marks, but queue overflows occur during the spikes. An adjustment to the exponential weight of the queue average or larger physical buffer size could help, but they affect not only to this issue.

ECN moves the median elapsed times of the bulk TCP flows away from each other by marking the bulk TCP flows unfairly because the marking process is random and can affect only a single flow at a time. After the ECN mark, there is a period during which the congestion windows are unbalanced for the advantage of a non-marked flow. Also the interquartile range of the elapsed time of the faster flow is larger because of the ECN marks. We can think these two effects as a quantification error of the RED algorithm because only one flow is marked at a time unless the queue average is above the maximum threshold. Therefore it is expectable that this

unfairness does not increase boundlessly.

Baseline TCP with Prioritization for HTTP

When Baseline TCP is used with prioritization, the median overall throughput is 7854 B/s and the stability of the overall throughput is very good. The median HTTP response time is 28% shorter and the maximum response time is 38% shorter than with Baseline TCP. The stability of the response time is very good, the interquartile range is only 1.1 seconds, which is 78% smaller than with Baseline TCP. There are no drops for the HTTP objects. The median elapsed time of the faster flow is not altered when compared with Baseline TCP but the interquartile range of the elapsed time is 29% smaller than with Baseline TCP. The elapsed time of the slower flow is less than 1% shorter than with Baseline TCP. Even though the spurious RTOs rarely occur for the HTTP transfer with this configuration, the upper quartiles of the spurious RTOs and unnecessary retransmissions are zero. The median simultaneous throughput is 2662 B/s for the faster flow, 2430 B/s for the slower flow, and 2299 B/s for the HTTP transfer.

The overall throughput is higher than with Baseline TCP because there are less unnecessary retransmissions. The effect, however, is insignificantly small from the bulk transfer perspective. The good stability of the overall throughput happens because the bulk TCP queue of the last-hop router remains long when the HTTP transfer ends. Also the elapsed time of the slower flow is negligibly shorter because of the reduced number of unnecessary retransmissions.

We see a significant reduction in the HTTP response time because the last-hop router is prioritizing HTTP packets. The router is able to queue the incoming HTTP packets even when the bulk TCP queue is congested. Therefore HTTP packets are not dropped at all. The queuing delay of the HTTP packets is reduced because the HTTP packets intervene the bulk TCP packets from its own queue. The shorter queuing delay reduces the round-trip time resulting in shorter elapsed times of the HTTP objects and less spurious RTOs. The smaller number of spurious RTOs cuts down the number of unnecessary retransmissions close to zero. Spurious RTOs occur only for the initial window of the third inline object, and with one exception, also the seventh inline object experiences a spurious RTO in the initial window in addition to the one for the third inline object.

Prioritization improves the stability of the elapsed time of the faster flow and provides more fairness between the bulk TCP flows because congestion in the bulk TCP queue affect both the flows quite fairly. For the bulk TCP flows, a spurious RTO occurs in the repetitions where the HTTP transfer begins after 32 seconds. Figure 21 shows the time-sequence graph for the bulk TCP flow in a repetition with HTTP transfer starting time at 35.6 seconds (HTTP transfer itself is not shown). The entry of the HTTP transfer causes congestion after 40 seconds. A loss recovery starts with the fast retransmit at 45 seconds. A spurious RTO for the first retransmitted packet is triggered at 50 seconds because of the increased round-trip time due to the

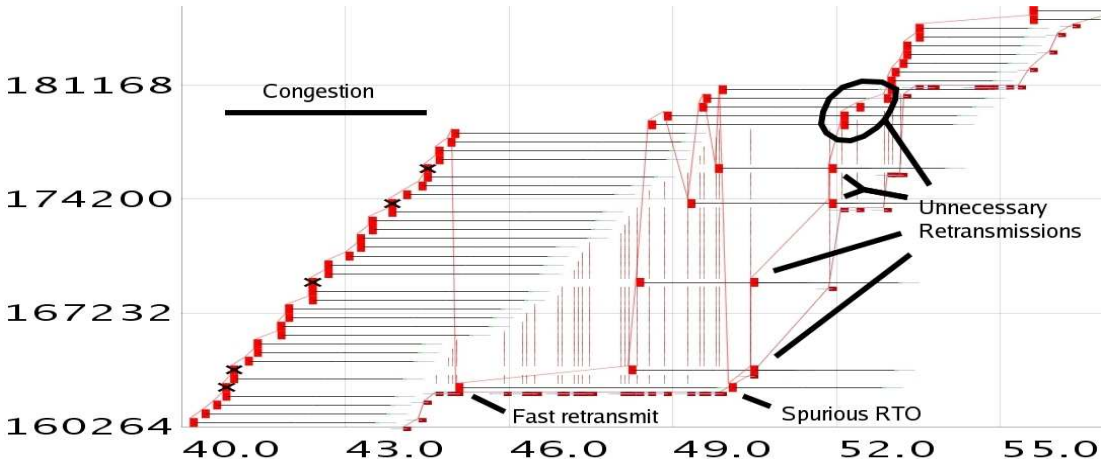


Figure 21: Time sequence graph for a bulk TCP having a spurious RTO after the entry of the HTTP transfer at 35.6 seconds

intervening HTTP packets. SACK information constrains the resulting unnecessary retransmissions to segments that were lost during the congestion and to the new segments sent during the fast recovery. These unnecessary retransmissions lengthen the total elapsed time by 0.4-1.0 second depending on the number of unnecessary retransmissions occurring in the repetition. However, SACK scoreboard handling of Linux TCP is optimistic, a conforming TCP implementation would have to retransmit also the segments SACK indicates as received [MMFR96] resulting in a larger increase than with Linux TCP. If a congestion drop hits also the first retransmitted packet, obviously an RTO that occurs is not spurious.

Baseline TCP with ECN and Prioritization for HTTP

When Baseline TCP is used with ECN and prioritization, the median overall throughput is roughly the same as without ECN. The interquartile range of the overall throughput is 12% smaller than with Baseline TCP. The median response time of the HTTP transfer is 35% shorter and the interquartile range of the response time is 43% smaller than with Baseline TCP. The response time of Baseline TCP with prioritization was very stable, but this case has 160% larger interquartile range than it. The median elapsed time of the faster flow is 3.8% shorter and the interquartile range of the elapsed time is 180% larger than with Baseline TCP. The lower end of the elapsed time of the slower flow is equal to the case with Baseline TCP and prioritization but the upper end rises above the other Baseline TCP configurations. The upper quartiles of the spurious RTO and unnecessary retransmissions for the HTTP transfer are zero as with Baseline TCP and prioritization without ECN. The median simultaneous throughput is 2677 B/s for the faster flow, 2153 B/s for the slower flow, and 2541 B/s for the HTTP transfer.

The overall throughput is affect by the smaller number of the unnecessary retrans-

missions as was discussed with Baseline TCP and ECN. ECN marks occurring for the slower flow near the end of the faster flow cause reduction to the congestion window of the slower flow. When the faster flow completes, the slower flow does not have enough packet in flight and thereby the link becomes underutilized.

When compared with prioritization without ECN, the queuing delay for the HTTP packets is reduced only during the moments when the bulk TCP queue is emptied by an ECN mark or by a congestion drop. In such a repetition, the performance of the bulk TCP flows is not ruined because the link send buffer still has bulk TCP segments awaiting.

Similarly to the Baseline TCP with ECN case, the number of the spurious RTOs for the HTTP transfer increase slightly compared to the Baseline TCP with prioritization. As with Baseline TCP and prioritization, all spurious RTOs hit the third inline object. However, ECN marks solve the spurious RTO problem that occurred for the bulk TCP flow with Baseline TCP and prioritization when the HTTP transfer started after 32 seconds.

Next we take a closer look at the repetitions where the elapsed time of the slower flow is clearly longer than the median. ECN marks for the slower flow empty the link send buffer after the faster flow has completed. With prioritization the bulk-TCP queue length is long when the HTTP transfer completes. Therefore ECN marks are more probable with prioritization than without it to cause a window reduction for the slower flow near the end of the faster flow. This results in such a small congestion window for the slower flow that is not large enough to fill the link for the duration of the whole round-trip. Therefore, the link send buffer is empty for a part of the round-trip and the elapsed time of the slower flow increases.

Enhanced TCP

When Enhanced TCP is used, the median overall throughput is 7872 B/s, which is 0.5% higher than with Baseline TCP. The interquartile range of the overall throughput is 48% smaller than with Baseline TCP. The response time of the HTTP transfer is 11% shorter but the interquartile range is 0.6 seconds (13%) larger than with Baseline TCP. The median elapsed time of the faster flow is 1.0% longer and the interquartile range of the elapsed time is 242% larger than with Baseline TCP. The median elapsed time of the slower flow is 1% shorter than with Baseline TCP. For the HTTP transfer, the median spurious RTOs is three and the median unnecessary retransmissions is four segments. The median simultaneous throughput is 2944 B/s for the faster flow, 2705 B/s for the slower flow, and 1872 B/s for the HTTP transfer.

The overall throughput is higher and the elapsed time of the slower flow is shorter because of the larger initial window. With the larger initial window, the WWW browser injects in initial window twice as much HTTP segments as Baseline TCP to the network for each HTTP object. Therefore the round-trip time during the inline object burst is longer than with Baseline TCP, resulting in spurious RTOs

systematically for the HTTP inline objects.

The bulk TCP flows pass on very conservative ssthresh with CBI reusing. As a result, the congestion that occurs after the start of the HTTP transfer is very limited or non-existing. When the HTTP starting times are before 20 seconds, the elapsed times of the bulk TCP flows have a large difference in some repetitions compared with the other other repetitions. In the unfair repetitions, only one bulk TCP flow is hit by a drop when the HTTP transfer starts. The congestion windows of the bulk TCP flows have grown, but the queue overflow of the last-hop router has not yet happened. The congestion that happens solely because of the bulk TCP flows hits very likely both flows, impacting both TCP flows fairly. If only one of the flows is affected, the progress of the transfers starts to slowly diverge from each other ending up with a large difference between the elapsed times as can be observed from Table 9 and from the fact that the elapsed time of the slower flow is very stable.

Starting time	Min	25%	Median	75%	Max
- 20 s	89.06	91.53	96.63	103.60	104.33
20 s -	87.65	101.18	103.48	104.11	104.70

Table 9: Elapsed time of the faster flow in classes based on the starting time of the HTTP transfer

Four factors shorten the HTTP response time. First, the queue length is shorter because CBI reduces the initial ssthresh of the bulk TCP flows. The queue length is shorter especially in the repetitions where the starting time of the HTTP transfer is prior the beginning of the overshoot recovery of the bulk TCP flows in the Baseline TCP case, which occurs near 12 seconds as was discussed with the workload of the short competing TCP flow (see Section 5.1). The second factor is the larger initial window that reduces the number of round-trips required to complete the transfer. Also, the larger initial window loses less likely all its packets, requiring an RTO. Third, some drops of the HTTP objects benefit the response time as the inline object burst is spread. Lastly, the spurious RTOs affect less with F-RTO because a part of them is detected, thus the corresponding unnecessary retransmissions are avoided. F-RTO reduces the number of unnecessarily retransmissions occurring for the HTTP transfer despite of the F-RTO feature in the test kernel which counts retransmissions of the peer as duplicate ACKs (see Section 4.5). The upper quartile has 50% less unnecessary retransmissions. In the case leading to the maximum number of the unnecessary retransmissions, SYN-ACK due to retransmitted SYN triggers additional duplicate ACK which arrives after the spurious RTO preventing F-RTO recovery of four HTTP objects, but in many other cases that are close to the maximum of unnecessary retransmissions, the retransmissions of the HTTP request prevent the F-RTO recovery. Some of the tests were rerun with a improved F-RTO implementation, which ignores the segment that is a retransmission of the opposite side. In this case, also the maximum of the unnecessary retransmissions is cut down with same magnitude as the upper-quartile.

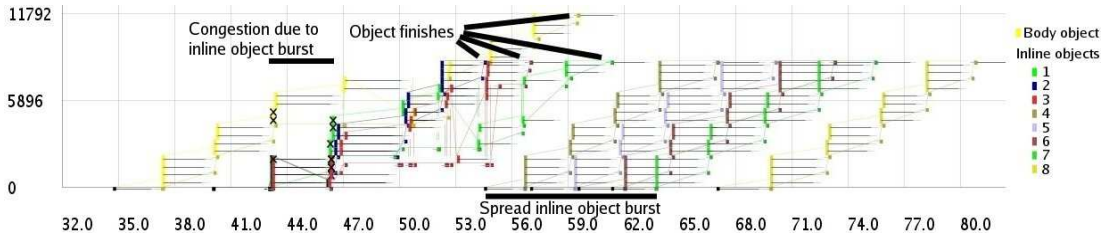


Figure 22: Time sequence graph for the HTTP objects in which congestion balances queuing delay of the HTTP transfer

With larger initial window congestion can also result in shorter response time. Figure 22 shows an example time-sequence graph for the HTTP objects from the server's point of view. The HTTP body request arrives slightly before 35 seconds. The inline objects burst causes congestion at 43 seconds. Multiple round-trips are necessary to trigger the three duplicate ACKs for those objects, but the transfers of the objects still make some progress during these round-trips. When the loss recovery begins, only a few segments are required to complete the transfer and because of the larger initial window the congestion window remains larger than with Baseline TCP. When an HTTP object completes, a new inline object begins SYN exchange. The objects complete at different time due to loss recovery caused variations. Therefore the inline object burst is spread, which reduces round-trip time of an individual segment and probability of congestion later on. In addition, no time is wasted on the SYN round-trips because the other objects are transferring data segments during the SYN exchange.

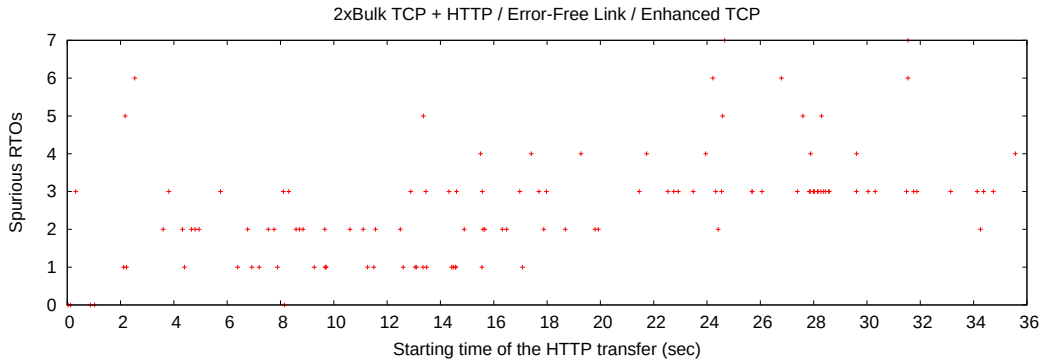


Figure 23: Spurious RTOs for the HTTP transfer with Enhanced TCP in individual repetitions over error-free link (120 repetitions)

The total number of spurious RTOs in individual repetitions for the HTTP transfer are shown in Figure 23. It is raising as a function of the starting time. The increase is caused by the queuing delay growth. When the starting time is less than two seconds, there are some repetitions without spurious RTOs. After two seconds the number of spurious RTOs is increasing as a function of the HTTP starting time.

After 20 seconds, the median is three spurious RTOs but there are some repetitions that have more of them. The majority of the spurious RTOs occur for the objects that are in the first inline burst. Only few spurious RTOs occur for the body, which then completes quickly. The spurious RTOs that with Baseline TCP occurred for the two latest inline objects occur now for the sixth and seventh inline objects because of the larger initial window.

Enhanced TCP with ECN

When Enhanced TCP is used with ECN, the median overall throughput is 7873 B/s. The median HTTP response time is 23% shorter and the interquartile range is 14% smaller than with Baseline TCP. The median elapsed time of the faster flow is 3.4% shorter and the interquartile range of the elapsed time is 145% larger than with Baseline TCP. For the HTTP transfer, the median spurious RTOs is two, the interquartile range of the spurious RTOs is two, the median unnecessary retransmissions is two, and the interquartile range of the unnecessary retransmissions is two packets. The median simultaneous throughput is 2955 B/s for the faster flow, 2509 B/s for the slower flow, and 2145 B/s for the HTTP transfer.

As was discussed in the Baseline TCP with ECN case, the HTTP response time is shorter, the number of drops for the HTTP transfer is smaller, and the number of spurious RTOs for the HTTP transfer are smaller than with Enhanced TCP because of the shorter queuing delay due to the ECN.

The unfairness between the bulk TCP flows that happens with Enhanced TCP is mitigated with a high probability by the ECN marks that prevent unfair scenarios. If we compare the elapsed time of the faster flow with Baseline TCP, the elapsed time seems instable. However, the interquartile range is almost equal to the Baseline TCP with ECN case. Hence, the instability that Enhanced TCP introduced into the elapsed time is slightly reduced by ECN.

Enhanced TCP with Prioritization for HTTP

When Enhanced TCP is used with prioritization, the median overall throughput is 7879 B/s and the stability of the overall throughput is good. The median HTTP response time is 27% shorter and the interquartile range of the response time is 17% smaller than with Baseline TCP. The median elapsed time of the faster flow is 0.3% shorter and the interquartile range of the elapsed time is 244% larger than with Baseline TCP. The median and the upper quartile of the spurious RTOs and of the unnecessary retransmissions for the HTTP transfer are one. The median simultaneous throughput is 2716 B/s for the faster flow, 2440 B/s for the slower flow, and 2282 B/s for the HTTP transfer.

The elapsed time of the slower flow is very slightly shorter than with Baseline TCP because of the smaller number of unnecessary retransmissions as was observed with Baseline TCP and prioritization. Also the larger initial window shortens the elapsed

time of the slower flow slightly. The HTTP response time is shorter because of the separate HTTP class queue that shortens queuing delay of the HTTP segments as with Baseline TCP and prioritization. However, the round-trip time is still long enough in 75% of the repetitions to cause a spurious RTO for the initial window of the latest inline object in the first inline object burst because of the larger initial window. In addition, 12% of the repetitions have another spurious RTO for a later inline object.

The median response time is inferior to Baseline TCP with prioritization case because bulk TCP flows have larger effect on the HTTP transfer. In the lower quartile, the response time is shorter where as in the upper quartile this case has longer response time than with Baseline TCP and prioritization, and in the median case the Baseline TCP and prioritization wins slightly. The longer end of the response times include cases where only one of the bulk TCP flows receives congestion drops or congestion does not occur at all during the HTTP transfer. The shorter end includes cases where both bulk TCP flows are slowed down by congestion drops during the HTTP transfer. The drops lead to a period during which the link send buffer is slightly less than full, which enables the HTTP segments to experience shorter queuing delay than with Baseline TCP and prioritization. In addition, there could be other factors that have slight effect on the difference.

Enhanced TCP with ECN and Prioritization for HTTP

When Enhanced TCP is used with ECN and prioritization, the median overall throughput is 7878 B/s. The interquartile range of the overall throughput is 77% higher than with Baseline TCP. The median HTTP response time is 31% shorter and the interquartile range of the response time is 17% smaller than with Baseline TCP. The median elapsed time of the faster flow is 2.9% shorter and the interquartile range of the elapsed time is 143% larger than with Baseline TCP. ECN is stabilizing the elapsed time of the faster flow similarly as also happened in the Enhanced TCP with ECN case. The median and the upper quartile of the spurious RTOs and unnecessary retransmissions for the HTTP transfer are one. The median simultaneous throughput is 2667 B/s for the faster flow, 2490 B/s for the slower flow, and 2416 B/s for the HTTP transfer.

The overall throughput remains high, but as discussed with Baseline TCP, ECN, and prioritization, there are repetitions that have an ECN mark for the slower flow leading to underutilization. The median overall throughput is not changed from the case with Enhanced TCP and prioritization. As with Enhanced TCP and prioritization, spurious RTOs occur.

The response time is longer than with Baseline TCP and the same configuration. With Baseline TCP, the average queue length of RED grows quickly because the slow-start overshoot of the bulk TCP flows keeps queue almost full for one round-trip. As CBI prevents the slow-start overshoot from occurring with Enhanced TCP, the queue length grows slowly and ECN marks occur much later than with Baseline

TCP, usually very near the end of the HTTP transfer. Therefore link send buffer is almost full most of the time during the HTTP transfer with Enhanced TCP whereas with Baseline TCP the bulk TCP flows slow down and the link send buffer is occasionally slightly shorter than full, which reduces queuing delay of the HTTP segments.

5.3.2 Link with High ARQ Persistency

In Figure 24a is the overall throughput over the link with high ARQ persistency. The Enhanced TCP cases yield slightly higher overall throughput than the Baseline TCP cases. The ECN cases have a slightly lower median overall throughput.

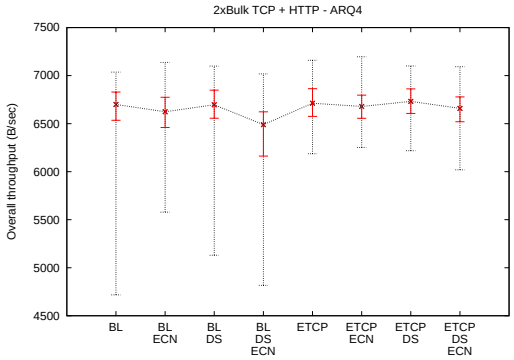
The response times for the HTTP transfer is presented in Figure 24b. The pattern of the response time is very close to the one with the error-free link, and also the reasons are similar. The response times are not as stable as with the error-free link.

In Figure 24c are the elapsed times of the bulk TCP flows. The stability of the elapsed time of the slower flow is not significantly affected by the configuration of the last-hop router, except in the case of Baseline TCP with ECN and prioritization. In Figure 24d are the spurious RTO and the unnecessary retransmissions for the bulk TCP flows and for the HTTP transfer. The number of unnecessary retransmissions for the slower flow is clearly higher in Baseline TCP with ECN and prioritization than with Baseline TCP. Also with Baseline TCP and ECN has more unnecessary retransmissions for the slower flow than Baseline TCP but not as radically as with ECN and prioritization. With Enhanced TCP the number of unnecessary retransmissions for the HTTP transfer is slightly larger than with Baseline TCP.

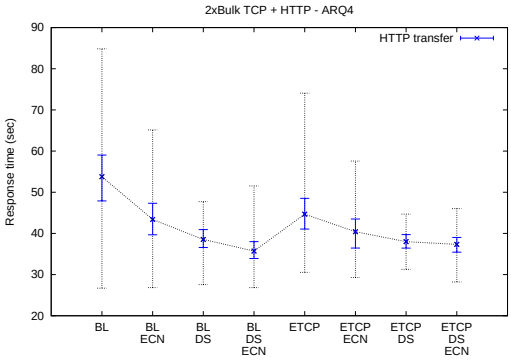
In Figure 24e are the total drops for the bulk TCP flows and for the HTTP transfer. Prioritization eliminates most of the drops that occur for the HTTP transfer. The numbers of drops are smaller in the ECN cases than without ECN. In Figure 24f are the simultaneous throughput of the transfers. ECN and prioritization clearly increase the simultaneous throughput of the HTTP transfer.

Baseline TCP

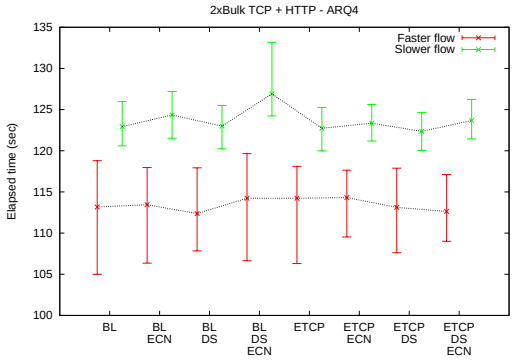
When Baseline TCP is used, the median overall throughput is 6700 B/s. The median HTTP response time is 53.8 seconds, which is 3% longer than with the error-free link. The interquartile range of the response time is 11.2 seconds, which is 126% larger than with the error-free link. The median congestion drops of the HTTP transfer is ten, and the interquartile range of the congestion drops is six packets. The median elapsed time of the faster flow is 113.2 seconds, and the interquartile range of the elapsed time is 13.8 seconds. For the slower flow, the median elapsed time is 122.9 seconds, and the interquartile range of the elapsed time is 5.4 seconds. The median simultaneous throughput is 2716 B/s for the faster flow, 2183 B/s for the slower flow, and 1603 B/s for the HTTP transfer. The median unnecessary retransmissions of the slower flow is zero.



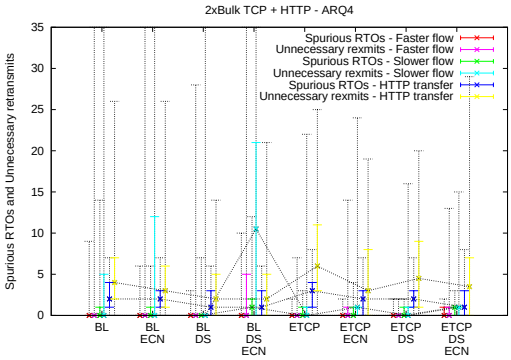
(a) Overall throughput



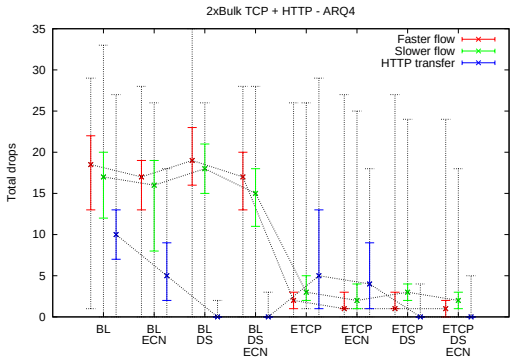
(b) HTTP Response time



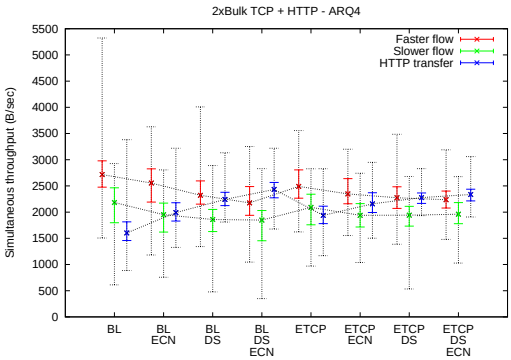
(c) Elapsed time of the bulk TCP flows



(d) Spurious RTOs and unnecessary retransmissions (the largest maximums are cropped; for HTTP total over 9 connections)



(e) Packets dropped (for HTTP total over 9 connections)



(f) Simultaneous throughputs

Figure 24: Summary of results over the link with high ARQ persistency in the workload of a competing HTTP transfer (120 repetitions)

When the HTTP response time is compared with the error-free link results, the lower quartile is slightly shorter because few repetitions have an error drop for the bulk TCP flows, while the upper quartile of the response time increases due to the delays from the link-level retransmission during the bad states. This duality explains the poor stability of the response time. During a HTTP transfer the link experiences

multiple bad states. Each of them may accumulate additional delay to the total response time. The repetitions that accumulate a long delay lengthen the upper quartile.

Queue overflows affect the transfer of the HTTP objects. First, there is the slow-start overshoot of the bulk TCP flows during which many losses occur for the HTTP transfer as well. When the HTTP starting time is clearly below 20 second, the queue of the last-hop router overflows second time between 20-30 seconds. If a bad state begins during the overflow round-trip, the congestion drops occur on the both sides of the bad state. When the starting time of the HTTP transfer is approaching 20 seconds, the second overflow shifts slowly to a later point. When a HTTP object is transmitted during the slow-start overshoot of the bulk TCP flows or during the second overflow, the HTTP object often loses most of the window because of the congestion. Such a loss burst delays the completion of the HTTP transfer because the congestion window of the HTTP object is set to a small value. A smaller congestion window requires more round-trips to transfer the object and is more sensitive to additional losses. When the drops affect the initial window but the other segment is delivered, the start of the recovery is delayed by multiple round-trips, which are required to generate three duplicate ACKs.

	Body	Inline objects								All
		1	2	3	4	5	6	7	8	objects
Spurious RTOs	36	37	45	49	28	27	26	31	29	308

Table 10: Total number of spurious RTOs for each HTTP object over individual repetitions (120 repetitions)

Table 10 shows the total number of spurious RTOs for the HTTP objects over all repetitions. Even though the total number of the spurious RTOs is roughly the same as with the error-free link, the spurious RTOs are spread across the HTTP objects more evenly than with the error-free link because of the spurious RTOs that are triggered during the link-level retransmissions in the bad state. Yet, the inline objects in the first inline burst have slightly more spurious RTOs.

Baseline TCP with ECN

When Baseline TCP is used with ECN, the median overall throughput is 6624 B/s. The median HTTP response time is 19% shorter and the interquartile range of the elapsed time is 29% smaller than with Baseline TCP. For the HTTP transfer, the congestion drops median is five, and the interquartile range of the congestion drops is seven packets. The median elapsed time of the faster flow is 113.5 seconds and the interquartile range of the elapsed time is 16% smaller than with Baseline TCP. The median elapsed time of the slower flow is 124.4 seconds and the interquartile range of the elapsed time is 5.8% larger than with Baseline TCP. The median simultaneous

throughput is 2555 B/s for the faster flow, 1952 B/s for the slower flow, and 1991 B/s for the HTTP transfer.

The HTTP response time is shorter and the probability of congestion is smaller than with Baseline TCP because ECN marks shorten the queue of the last-hop router, which reduces round-trip times. The number of drops is almost halved for the HTTP transfer. The HTTP objects benefit largely from the reduction of the congestion drops. ECN is reducing also the drops that occur for the bulk TCP flows. However, the second buffer overflow is not avoided except in a few repetitions.

There is a phenomenon that increases the number of spurious RTOs for the bulk TCP flows but we postpone a detailed analysis to the case of Baseline TCP with ECN and prioritization because in that case the probability of the phenomenon is much higher.

Baseline TCP with Prioritization for HTTP

When Baseline TCP is used with prioritization, the median overall throughput is 6696 B/s. The median HTTP response time is 28% shorter and the interquartile range of the response time 61% smaller than with Baseline TCP. The upper quartile of the HTTP packets dropped is zero. The median elapsed time of the faster flow is 112.4 seconds and the interquartile range of the elapsed time is 27% smaller than with Baseline TCP. The median elapsed time of the slower flow is 123.0 seconds and the interquartile range of the elapsed time is 2.0% smaller than with Baseline TCP. The median simultaneous throughput is 2320 B/s for the faster flow, 1858 B/s for the slower flow, and 2242 B/s for the HTTP transfer.

Prioritization of the HTTP packets work as with the error-free link. Therefore the HTTP transfer does not experience congestion drops and the response time is shorter because the round-trip time for an HTTP packet is reduced. The simultaneous throughput of the HTTP transfer is almost as good as the simultaneous throughput of the faster flow.

Most of the spurious RTOs for the HTTP objects occur due to the bad state link-level retransmissions. Therefore the number of spurious RTOs is almost equal with each HTTP object.

Baseline TCP with ECN and Prioritization for HTTP

When Baseline TCP is used with ECN and prioritization, the median overall throughput is 6489 B/s. The median HTTP response time is 34% shorter and the interquartile range of the response time is 63% smaller than with Baseline TCP. The upper quartile of the HTTP packets dropped is zero. The median elapsed time of the faster flow is 114.2 seconds and the interquartile range of the elapsed time is 6% smaller than with Baseline TCP. The median elapsed time of the slower flow is 126.9 seconds, and the interquartile range of the elapsed time is 9.0 seconds, which is 66% larger than with Baseline TCP. The median simultaneous throughput is 2176

B/s for the faster flow, 1848 B/s for the slower flow, and 2433 B/s for the HTTP transfer. The median unnecessary retransmissions of the slower flow is 10.5 packets.

As in Baseline TCP with prioritization, the HTTP response time is shorter and the number of HTTP packet drops is smaller because of prioritization. ECN slightly lowers the overall throughput because occasionally the link becomes underutilized as discussed with the error-free link when Baseline TCP is used with ECN and prioritization. Unlike with the error-free link, it is possible that the link becomes underutilized even when the bulk TCP flows are still both active because occasional error drops reduce congestion windows.

A phenomenon that increases spurious RTOs for the bulk TCP flows appears because of RED. RED is able to control the queue lengths but because of the exponentially weighted moving average, it has a memory that lasts beyond the end of the HTTP transfer. Immediately after the end of the HTTP transfer the queuing delay shrinks because the HTTP transfer ceases, but the bulk TCP flows still receive ECN marks that force bulk TCP flows to reduce the transfer rate and thereby further reduce the queuing delay. With the low queue delay, the RTO value shrinks. When a bad state with the link-level retransmissions begins, the delay exceeds the current RTO value, which triggers a spurious RTO. After the spurious RTO, the bulk TCP flow reduces its congestion window, which reduces the round-trip time, and hence the RTO value shrinks again. During the next bad state similar events are repeated. The poor stability of the elapsed time of the slower flow is caused by those spurious RTOs.

Enhanced TCP

When Enhanced TCP is used, the median overall throughput is 6713 B/s. The median HTTP response time is 17% shorter and the interquartile range of the response time is 33% smaller than with Baseline TCP. The median simultaneous throughput is 2490 B/s for the faster flow, 2090 B/s for the slower flow, and 1937 B/s for the HTTP transfer.

The HTTP response time is shorter than with the Baseline TCP. Two main reasons explain it. The slow-start overshoot of the bulk TCP flows is avoided because of CBI, and a smaller number of round-trips is required to complete the transfer with the initial window of four. The larger initial window is also less sensitive to losses that are now slightly more frequent due to corruption. In addition, the spurious RTOs for the bulk TCP flows reduce queuing delay because the used response algorithm reduces the congestion windows of the bulk TCP flows. The drops for the HTTP objects positively balance the queuing delay of the HTTP objects as was discussed with the error-free link. In the repetitions with error drops or spurious RTOs during the slow start of the bulk TCP flows, unbalanced congestion windows are prevented by CBI as in the workloads of a short competing TCP flow (see Section 5.1).

Enhanced TCP with ECN

When Enhanced TCP is used with ECN, the median overall throughput is 6678 B/s. The median HTTP response time is 25% shorter and the interquartile range of the response time is 37% smaller than with Baseline TCP. The median simultaneous throughput is 2347 B/s for the faster flow, 1939 B/s for the slower flow, and 2158 B/s for the HTTP transfer. The response times are shorter because CBI prevents the slow-start overshoot from occurring for the bulk TCP flows, ECN constrains the queuing delay, and the larger initial window allows faster completion of the transfer.

Enhanced TCP with Prioritization for HTTP

When Enhanced TCP is used with prioritization, the median overall throughput is 6732 B/s. The median HTTP response time is 29% shorter and the interquartile range of the response time is 71% smaller than with Baseline TCP. The median simultaneous throughput is 2275 B/s for the faster flow, 1942 B/s for the slower flow, and 2278 B/s for the HTTP transfer. Besides what happens happens with Baseline TCP and prioritization, the queuing delay is shorter because of CBI. Therefore the response time is shorter than with Baseline TCP and prioritization.

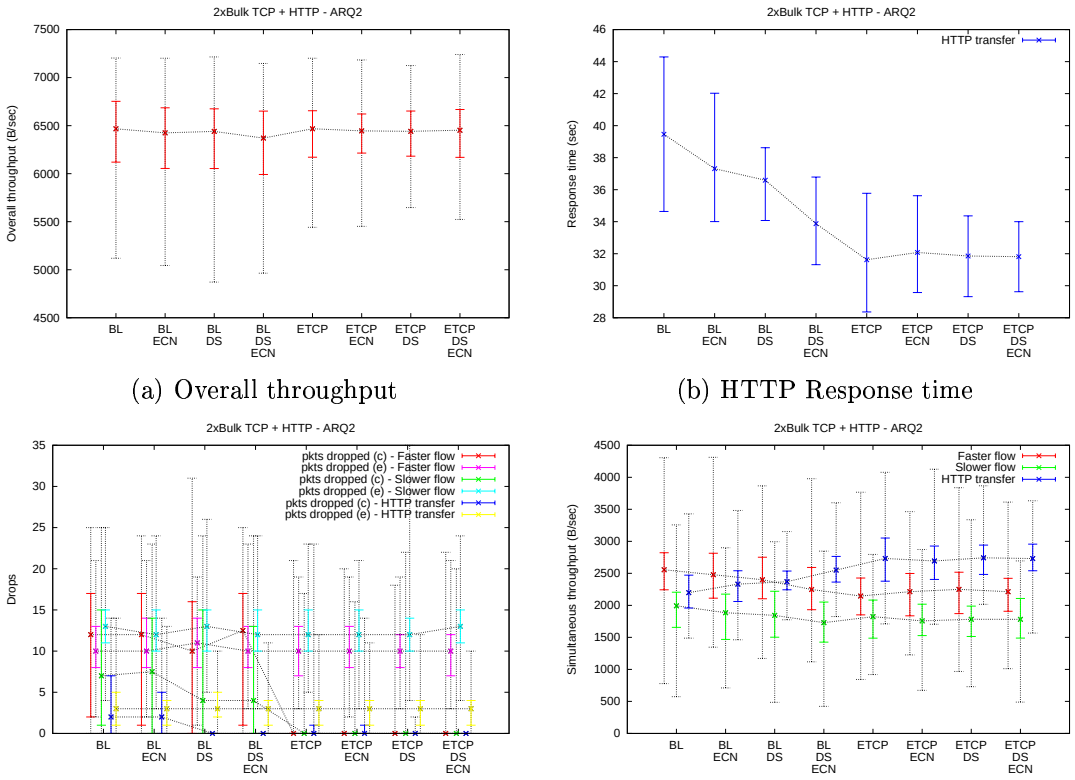
Enhanced TCP with ECN and Prioritization for HTTP

When Enhanced TCP is used with ECN and prioritization, the median overall throughput is 6659 B/s. The median HTTP response time is 30% shorter than with Baseline TCP, which is only slightly shorter than with prioritization without ECN. As with error-free link, CBI prevents the slow-start overshoot from occurring for the bulk TCP flows, which moves ECN marks to a later point causing ECN to benefit the HTTP response time less than with Baseline TCP and prioritization. The interquartile range of the response time is 68% smaller than with Baseline TCP. In numbers, the difference of the interquartile range between this and the prioritization only case is just 0.3 seconds. The median simultaneous throughput is 2236 B/s for the faster flow, 1960 B/s for the slower flow, and 2336 B/s for the HTTP transfer. The median unnecessary retransmissions of the slower flow is one packet.

It is expected that ECN cannot improve the response time much, as the CBI and prioritization limit congestion. Therefore only the repetitions that can reach a reasonable level of congestion can slightly benefit from ECN. F-RTO in Enhanced TCP eliminates all but one of the unnecessary retransmissions caused by the phenomenon that happened in Baseline TCP with ECN and prioritization.

5.3.3 Links with Medium and Low ARQ Persistency

The overall throughput over the link with medium ARQ persistency is shown in Figure 25a.



(a) Overall throughput

(b) HTTP Response time

(c) Packets dropped (for HTTP total over 9 connections)

(d) Simultaneous throughputs

Figure 25: Summary of results over the link with medium ARQ persistency in the workload of a competing HTTP transfer (120 repetitions)

In Figure 25b is the response time of the HTTP transfer. ECN helps with Baseline TCP. With Enhanced TCP the response time is shorter than with Baseline TCP and the enhancements only make the response time slightly more stable.

In Figure 25c is the total number of congestion and error drops for the bulk TCP flows and for the HTTP transfer. With Baseline TCP congestion occurs but with Enhanced TCP congestion drops are very few. With prioritization, only error drops occur for the HTTP transfer.

Figure 25d shows the simultaneous throughputs for the bulk TCP flows and for the HTTP transfer.

When Baseline TCP is used, the median overall throughput is 6467 B/s. The median response time of the HTTP transfer is 39.5 seconds, which is 24% shorter than with the error-free link. The interquartile range of the response time is 9.7 seconds, which is 9.6% larger than with the error-free link. The response time is shorter than with the error-free link because of the error-related drops slow down the other traffic, which reduces the length of the shared queue.

For the bulk TCP flows, the median of the error drops is 10 for the faster flow

and 13 for the slower flow. When compared with the error-free link results, the HTTP response times that are longer than the median have a smaller reduction in the response time and the shorter ones have a larger reduction than the median case. This is an indicator of the error drop dominance in the cases having a short response time and of the congestion drop dominance in the opposite end. The low-end response times consist of repetitions where both bulk TCP flows suffer from error drops, and therefore they cannot disturb enough the HTTP transfer, and where the HTTP transfer avoids all drops that happen during the first round trips. On the contrary, the high-end response times include RTO that delays the completion of the whole HTTP transfer because few of the HTTP segments are delivered after a long delay, or the unlucky repetitions where multiple error drops hit at least one of the HTTP object and the effect is then cascaded to the transfer of the later objects because the number of concurrent HTTP connections is limited.

When Baseline TCP is used, the congestion is experienced due to the slow-start overshoot whenever an error drop does not cause early transition to the congestion avoidance. Congestion drops happen during the slow-start overshoot of the bulk TCP flows and near the entry of the HTTP transfer. When the HTTP starting time increases, a probability of congestion at the entry of the HTTP transfer decreases because the bulk TCP flows have already quite small congestion windows because the probability of an earlier error drop is high.

When Baseline TCP is used with ECN, the median overall throughput is 6425 B/s. ECN is still effective, the median HTTP response time 5% shorter and the interquartile range of the response time is 17% smaller than with Baseline TCP. When the HTTP transfer starts during the initial slow start of the bulk TCP flows, there are a few repetitions where ECN is beneficial, but the benefits are small. In the later starting repetitions, RED controls the queue before the entry of the HTTP transfer, which makes room for the HTTP transfer to complete faster.

When Baseline TCP is used with prioritization, the median overall throughput is 6440 B/s. The median HTTP response time is 7% shorter and the interquartile range of the response time is 53% smaller than with Baseline TCP. The separate HTTP class queue shortens response time because the HTTP packets bypass the bulk TCP packets. As there is a separate queue, a simultaneous HTTP transfer does not cause congestion after entry. Instead, the bulk TCP queue absorbs the packets of the bulk TCP flows that are only delayed because of the prioritization. Therefore the faster TCP flow drops few packets less and completes faster, which reduces fairness between the bulk TCP flows.

When Baseline TCP is used with ECN and prioritization, the median overall throughput is 6371 B/s. The HTTP response time is still pushed downward, the median is 14% shorter than with Baseline TCP. As with link with high ARQ persistency, the response time is more stable than with Baseline TCP, but not as much as in the case with prioritization without ECN.

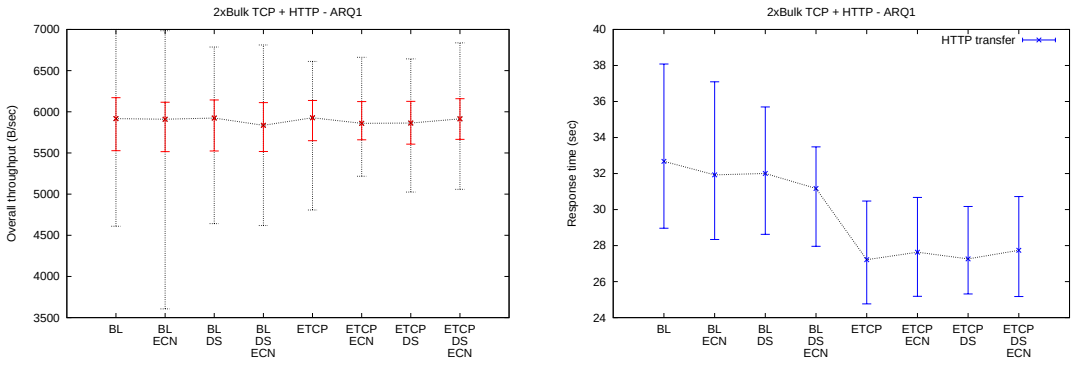
When Enhanced TCP is used, the median overall throughput is 6468 B/s. The slow-start overshoot does not occur because of CBI's ssthresh reusing. Because

of error-related drops, the bulk TCP flows are starting with an even lower initial ssthresh, and they often get hit by an error drop before the congestion window is large enough to congest the last-hop router. The median HTTP response time is 20% shorter and the interquartile range of the response time is 23% smaller than with Baseline TCP. Only the upper-quartile has one congestion drop for the HTTP transfer, which shortens response times. With the larger initial window the HTTP objects open the congestion window faster, complete with less round-trips, and are less vulnerable to error drops.

When Enhanced TCP is used with ECN, the median overall throughput is 6446 B/s. RED can do very little with Enhanced TCP because there is no congestion is most cases. Therefore only very few repetitions have any ECN marks. The median HTTP response time is 19% shorter and the interquartile range of the response time is 37% smaller than with Baseline TCP.

When Enhanced TCP is used with prioritization, the median overall throughput is 6441 B/s. Prioritization cannot shorten the median response time of the HTTP transfer any further from the Enhanced TCP case because congestion drops are very few with Enhanced TCP, but stability of the response time is still better than with Enhanced TCP. Compared with Baseline TCP, the interquartile range of the response time is 48% smaller.

When Enhanced TCP is used with ECN and prioritization, the median overall throughput is 6452 B/s. The interquartile range of the HTTP response time is 55% smaller than with Baseline TCP. Besides that, the results are similar to Enhanced TCP with prioritization without ECN.



(a) Overall throughput

(b) HTTP Response time

Figure 26: Summary of results over the link with low ARQ persistency in the workload of a competing HTTP transfer (120 repetitions)

In Figure 26a is the overall throughput over the link with low ARQ persistency, and in Figure 26b is the response time of the HTTP transfer. A lossy link with low ARQ persistency has many error drops. These drops reduce the load of the last-hop router in most cases, which allows the competing HTTP transfer to start much faster without congestion. From the median response time with Baseline TCP we

deduce that the HTTP transfer got one third of the bandwidth during its transfer even without ECN or prioritization.

When Baseline TCP is used with prioritization, a half of the link bandwidth is allocated for the HTTP transfer but part of the allocated bandwidth is not used by the HTTP transfer because the slow-start round-trips limit the congestion window of the HTTP objects. Therefore the HTTP response time is only 0.6 seconds shorter than with Baseline TCP.

The median HTTP response time with Enhanced TCP is 15%-17% shorter than with Baseline TCP, depending on the configuration of the last-hop router. The interquartile range of the response time is 37%-47% smaller than with Baseline TCP. With Enhanced TCP 40% of the link bandwidth is used by the HTTP transfer as the HTTP objects have larger initial windows allowing them to open congestion window faster than in Baseline TCP. Even though the HTTP transfer is hit by the error-related drops, multiple concurrent transfers enable it to complete reasonably quickly because it is not very probable that the transfer of all four objects is disturbed simultaneously by the error drops. The bulk TCP flows do not have this advantage of multiple connection and thereby slow down significantly when an error occurs. The delays for an individual HTTP object are covered by the transfer of the other objects. The repetitions where the last-completing object is hit by the error drops are rare enough having little significance in the quartiles or the median. As ECN nor prioritization cannot affect the link errors and there is no congestion, the HTTP response time of the Enhanced TCP case is not shortened by the enhancements.

5.3.4 Summary of Results

When Baseline TCP is used, the response time of the HTTP transfer suffers because of slow-start overshoot of the bulk TCP flows, congestion, spurious RTOs and error drops. These problems have the most severe effect when they hit the initial window of an HTTP object. ECN reduces severity of these problems except for the slow-start overshoot. However, the problems are not solved completely. When Baseline TCP is used with prioritization, the response time of the HTTP transfer is shortened dramatically. The slow-start overshoot and congestion for the bulk TCP flows cannot delay HTTP packets, which are served from the HTTP-class queue. Spurious RTOs for the HTTP transfer become rare and its drops are completely removed. When prioritization is used together with ECN, congestion is controlled by ECN, which reduces queuing delay of the HTTP packets during the times when the bulk TCP queue becomes empty. Therefore the HTTP response time is slightly shorter than with prioritization alone. With all Baseline TCP configurations these effects become almost non-existing only over very error prone link. On the links that perform ARQ retransmissions, the response time grows because of stalls of the link. Those stalls also generate congestion.

When Enhanced TCP is used, CBI, F-RTO, and the initial window of four reduce the HTTP response time. CBI reduces congestion, F-RTO prevents unnecessary

retransmissions, and initial window of four allows faster opening of the congestion window. However, with larger initial window the spurious RTOs and drops become more frequent as the HTTP transfer generates congestion for itself. Therefore the results, when Enhanced TCP and prioritization is used, are not as good as in Baseline TCP with prioritization. Enhanced TCP with ECN has similar ECN mark benefits as the Baseline TCP with ECN case. Because slow-start overshoot of the bulk TCP flows is avoided, the HTTP transfer takes a large share of the bandwidth on the link with medium or low ARQ persistency. Therefore neither ECN nor prioritization has no significant effect on the HTTP response time with them.

With ECN fairness between the bulk TCP flows is inferior to the Baseline TCP case. With prioritization the fairness is better than in the Baseline TCP case. With error-free link the bulk TCP flows experience due to prioritization a queuing delay that is long enough to trigger spurious RTOs for the bulk TCP flows when the HTTP transfer begins after 32 seconds. The spurious RTO causes many unnecessary retransmissions. With prioritization and ECN, such spurious RTOs are prevented by ECN marks that reduce the queuing delay beforehand. However, with high ARQ persistency the small queuing delay can lead to loop which together with the link-level retransmission caused delay triggers a spurious RTO per each bad state for a bulk TCP flow. As with the workloads of a competing TCP flow, the slow-start overshoot is prevented by CBI in Enhanced TCP case. On the error-free link, unfairness increases between the bulk TCP flows when there is no slow-start overshoot because of drops affecting only one bulk TCP flow. However, ECN is able to improve fairness again to level with the Baseline TCP with ECN case. With other link types, no concrete trends for the bulk TCPs can be drawn out of the results.

5.4 Two Bulk TCP Flows Competing with a Streaming Flow

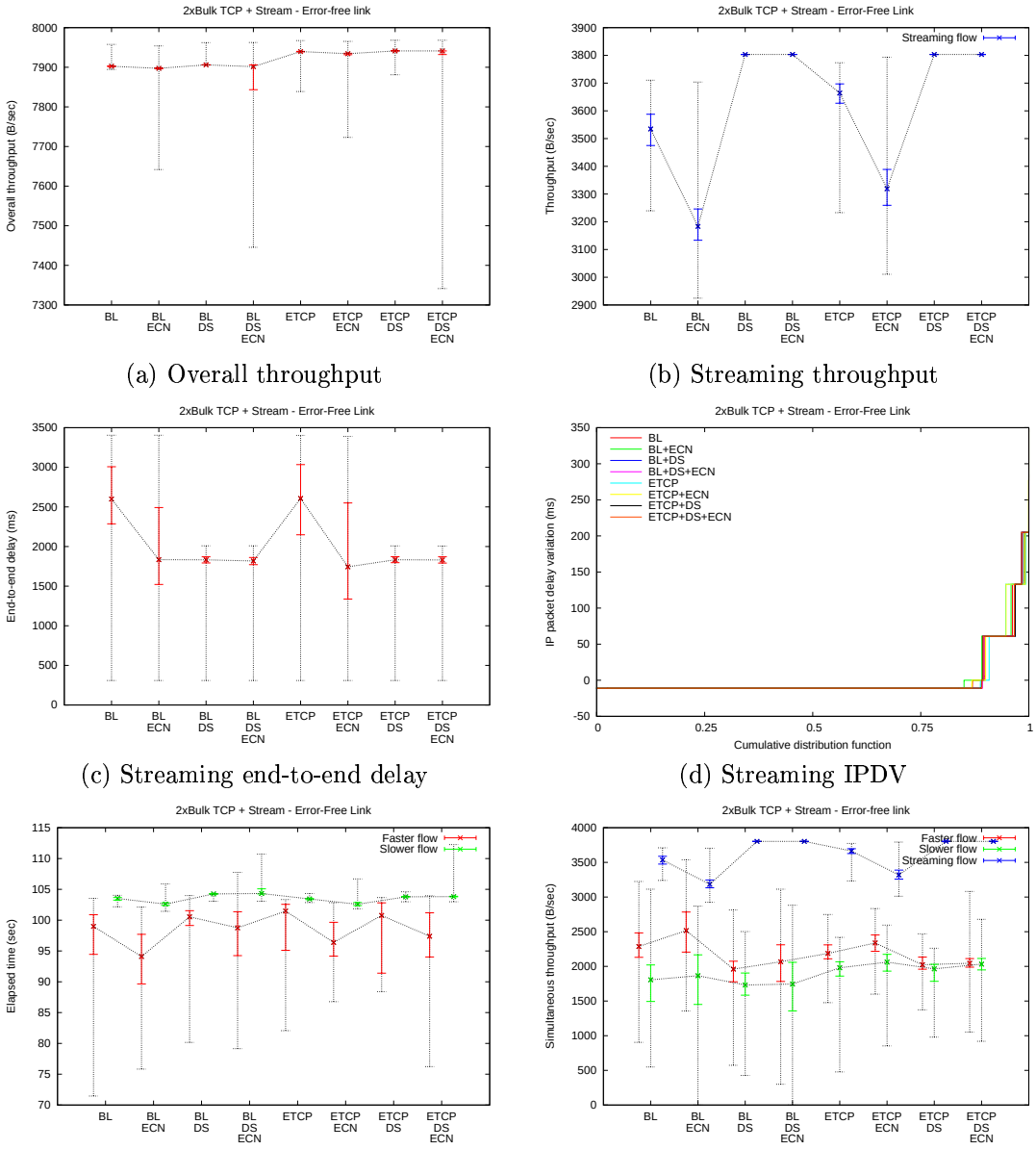
In the workload with competing streaming flow, there are two bulk data TCP transfers and one streaming UDP flow. The bulk TCP flows are started at zero seconds. We refer to them as the faster and slower flow.

We report the performance of the whole workload using the overall throughput. The value includes headers, and therefore its theoretical maximum is 8000 B/s. In the analysis of the streaming flow we use the throughput, the end-to-end delay, the IP Packet Delay Variation (IPDV), and the number of packet drops. The number of streaming packets dropped is inversely proportional to the throughput of the streaming flow. For analysis of the faster and slower flow we use elapsed time. In addition, the simultaneous throughputs are reported.

For each link type we use figures that compare the results with Baseline TCP (BL), Baseline TCP with ECN (BL+ECN), Baseline TCP with DiffServ prioritization for streaming (BL+DS), Baseline TCP with ECN and DiffServ prioritization for streaming (BL+DS+ECN), Enhanced TCP (ETCP), Enhanced TCP with ECN (ETCP+ECN), Enhanced TCP with DiffServ prioritization for streaming (ETCP+DS), and Enhanced TCP with ECN and DiffServ prioritization for

streaming (ETCP+DS+ECN). We analyze the cases in that order. Since only prioritization with this workload is for streaming, we refer to it simply as prioritization. The figures show the median and the quartiles with a solid errorbar for the given metrics. When present, a dotted errorbar shows the minimum and the maximum.

5.4.1 Error-Free Link



(a) Overall throughput

(b) Streaming throughput

(c) Streaming end-to-end delay

(d) Streaming IPDV

(e) Elapsed time of the bulk TCPs

(f) Simultaneous throughput

Figure 27: Summary of results over the error-free link in the workload of the competing streaming flow (120 repetitions)

The overall throughput over the error-free link is shown in Figure 27a. The stability of the overall throughput is good in all configurations except in Baseline TCP with ECN and prioritization. With Enhanced TCP the overall throughput is slightly higher than with Baseline TCP. In the cases with ECN, there is a negligible decrease in the overall throughput.

The streaming throughput is shown in Figure 27b. Without prioritization the throughput is lower than the sending rate. With ECN the throughput is clearly lower than in the other cases. With prioritization the streaming flow is served at the full rate in every case. The throughput of the streaming flow is higher with Enhanced TCP than with Baseline TCP.

The streaming end-to-end delay is shown in Figure 27c. The theoretical minimum of the end-to-end delay is the sum of the transmission and propagation delays. Baseline TCP has very long end-to-end delay. Without prioritization, ECN reduces the end-to-end delay of the streaming packets slightly. With prioritization the end-to-end delay remains stable but relatively long. In the ECN with prioritization case, the end-to-end delay is not improved against the prioritization only case. The end-to-end delay is only slightly affected by the TCP variant.

IPDV of the streaming flow is shown in Figure 27d. It is dominated by the bursts that occur when many streaming packets are delivered as consecutive packets and relatively large delays.

In Figure 27e, the elapsed times of the bulk TCP flows are shown. With ECN the median elapsed time of the faster flow is shorter than without ECN. With Enhanced TCP, the elapsed time of the faster flow is more stable with ECN, but with Baseline TCP the opposite happens. The variations of the streaming throughput are visible in the elapsed time of the slower flow because they affect the total size of the transmitted data.

The simultaneous throughputs are shown in Figure 27f. The streaming flow is able to gain a larger share of the bandwidth with prioritization, and therefore the bulk TCP flows have lower simultaneous throughputs. When Baseline TCP is used with prioritization, the simultaneous throughputs of the bulk TCP flows are more stable than without prioritization.

Baseline TCP

When Baseline TCP is used, the median overall throughput is 7903 B/s, and the stability of the overall throughput is good. The median streaming throughput is 3535 B/s. The interquartile range of the streaming throughput is 113 B/s. The median of the dropped streaming packets is 81, which is 7% of the streaming flow. For the streaming packets, the minimum end-to-end delay is 309 ms, the lower quartile is 2287 ms, the median is 2599 ms, and the maximum is 3404 ms. The quartiles of stream IPDV are -11 ms and the maximum is 349 ms. The median elapsed time of the faster flow is 99.0 seconds, and the interquartile range of the elapsed time is 6.3 seconds. The median elapsed time of the slower flow is 103.53 seconds. Spurious

RTOs do not occur.

The streaming throughput varies because of a varying number of streaming drops that occur because of congestion. In the end-to-end delay of streaming packets there is a large difference between the minimum and the lower quartile, which exists because the total delay is dominated by the queuing delay. The minimum end-to-end delay is on the edge where it could still be useful for a real-time streaming application. The value of the lower quartile, however, is way beyond of being useful in such an application.

IPDV is dominated by the bursts of streaming packets and the bursts of TCP packets. Without competition each streaming packet should arrive with 20 ms delay that is the sending interval. The competition complicates the arrival pattern because multiple TCP packets can intervene consecutive streaming packets in the queue. The wireless link transmits a single TCP packet in 72 ms, which is longer than 20 ms that is the sending interval of the streaming packets. Therefore multiple streaming packets arrive to the last-hop router during the transmission delay of the TCP packet. As new TCP segments are generated only after ACK or RTO, many of these streaming packets reside back-to-back in the queue, which causes them to be transmitted in a row. This burst of streaming packets is delivered to the receiver with intervals of 9 ms, which is the transmission delay of the streaming packet. When there is not enough buffer space, consecutive streaming packets are, of course, dropped by the last-hop router. As TCP uses delayed ACKs, each ACK typically acknowledges two segments, which allows sending at least two new TCP segments. These two TCP segments arrive almost at the same time to the last-hop router and are very probably queued in a row. When the link transmits these two or more segments, the transmission delay between two consecutive streaming packets is multiple of 72 ms. From the maximum IPDV of the streaming flow we can calculate that five TCP segments were transferred between two streaming packets.

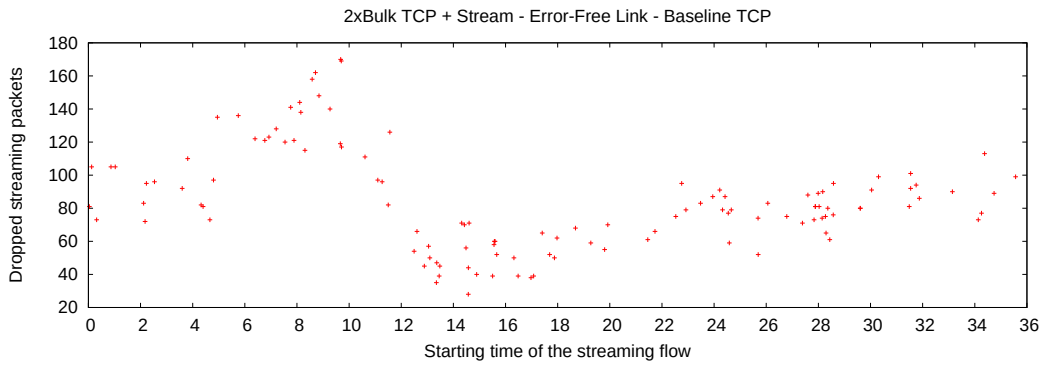


Figure 28: Drops for the streaming flow with Baseline TCP in individual repetitions over error-free link (120 repetitions)

In closer detail, the number of dropped streaming packets and the elapsed time of the faster flow depend on the starting time of the streaming flow. In Figure 28 is

the number of packet drops for the streaming flow in individual repetitions. The streaming drops are largely affected by the slow-start overshoot when the streaming flow starts between 5-12 seconds. During the overshoot, the streaming flow loses many packets in bursts during a transmission of the TCP packet. With starting time before 5 seconds, the streaming flow serves as a bandwidth limiter to the bulk TCP flows that overshoot earlier and less streaming packets are dropped than when the streaming flow starts during the overshoot of the bulk TCP flows because the congestion after the slow-start overshoot is almost non-existing. When the starting time is between 12 and 22 seconds, the number of streaming drops is smaller than elsewhere because the bulk TCP flows halve their congestion windows when the overshoot is detected, which makes room in the queue for the streaming packets. The streaming drops form approximately a right angle triangle whose base is upward and the right-angle at the 12 seconds because the queue length is again increasing towards later starting times. The number of drops in that region is less than half of the drops that took place in the repetitions that were largely affected by the slow-start overshoot. The number of streaming drops does not increase further when starting time is after 22 seconds because the last-hop router queue is nearly full when the streaming flow enters.

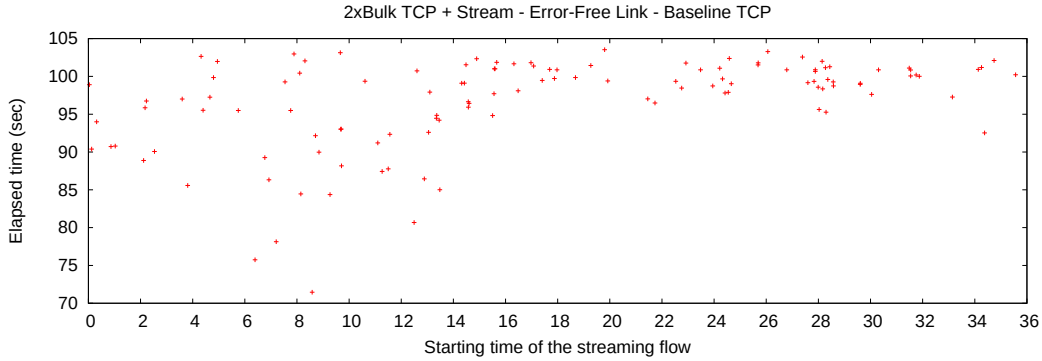


Figure 29: Elapsed time of the faster flow with Baseline TCP in individual repetitions over error-free link (120 repetitions)

In Figure 29 is the elapsed time of the faster flow in individual repetitions. The elapsed time of the slower flow is almost constant. In the repetitions with the streaming flow starting times before 14 seconds, the faster flow can gain an advantage that allows it to complete with a large margin to the slower flow, which reduces fairness between the bulk TCP flows. There are multiple reasons to this increase of the unfairness, and they cancel each other out in part of the repetitions. The most obvious reason are congestion drops that occur only for one of the bulk TCP flows after the transition to the congestion avoidance. The second reason is a lost retransmission that slows only the affected flow. The third reason is a slow overshoot recovery for the bulk TCP flow that loses more packets than the another bulk TCP flow during the slow-start overshoot. Because of the losses, less packets-in-flight estimate shrinking SACK blocks arrive during the recovery and more segments need

to be retransmitted than for the other bulk TCP flow. Therefore the congestion windows become unbalanced between the TCP flows, which is a condition that feeds itself. The bulk TCP flow that loses less packets completes the overshoot recovery quickly, and it starts to increase the congestion window in the congestion avoidance. The other bulk TCP flow cannot increase the congestion window until its overshoot recovery is complete, which takes a long time because of the larger amount of the losses and the smaller number of packets-in-flight. In the worst-case, the faster flow has completed the entire transfer when the slower flow is less than halfway through.

When starting time of the streaming flow is later than 14 seconds, the last-hop router queue length is short but increases rapidly because of the streaming packets. The growth leads to a congestion, which is present in every repetition. The congestion does not happen before 16 seconds because the queue length is short near the 14 seconds. Both bulk TCP flows reduce the congestion window because the congestion drops hit both of them. The equal treatment of the congestion windows is good for fairness. Part of the repetitions with the congestion drop burst include a drop of the first retransmission, which is prone to reduce the fairness of the elapsed times between the bulk TCP flows, but only slightly. When the starting time of the streaming flow is close to 36 seconds, there is another drop burst of few packets in few repetitions when streaming flow is near its end because the queue length grows until a new overflow occurs or some of the flows complete.

Baseline TCP with ECN

When Baseline TCP is used with ECN, the median overall throughput is 7897 B/s. The median streaming throughput is 10% lower than with Baseline TCP. The median of the dropped streaming packets is 187 packets, which is 16% of the stream. The interquartile range of the dropped streaming packets is almost the same as with Baseline TCP. For the streaming packets, the minimum and maximum values of the end-to-end delay are roughly the same as with Baseline TCP, and the median is 29% shorter than with Baseline TCP. IPDV is roughly the same as with Baseline TCP.

The median end-to-end delay of the streaming packets is shorter because ECN shortens the queuing delay. The minimum and maximum end-to-end delay, however, are not shorter because the link send buffer size is constraining the usefulness of ECN in controlling the total queuing delay. This happens because RED does not measure the length of the link send buffer, but averages only the last-hop router queue length. Thus the end-to-end delay is large also in this case.

After the entry of the streaming flow, the queue length is raising rapidly because of the streaming flow that does not perform a slow start. This growth often leads to a situation in which the queue average is firmly above the maximum threshold of RED. Since all streaming packets are non-ECN-capable, all of them are dropped when the maximum threshold is exceeded. Many repetitions that have the starting time of the streaming flow after 5 seconds are affected by this phenomenon with more than

100 drops in a row. However, there is a window of the starting times between 11-15 seconds in which some of the repetitions avoid the phenomenon and have less than 100 total drops. If the streaming flow begins before 10 seconds the number of drops is larger than in the repetition having a later streaming flow starting time because of the drops that occur during the slow-start overshoot.

As with Baseline TCP, it is very likely that the entry of the streaming flow causes a congestion very soon if the starting time is past the overshoot of the bulk TCP flows. The bulk TCP flows experience losses, and when they start to retransmit, they also use non-ECN-capable packets that are required by the specification [RFB01]. If the queue average is above the maximum threshold, every non-ECN-capable packet is dropped. When the streaming flow starts before 15 seconds, ECN marks with a higher probability before a drop can happen because the instant queue length is shorter. When the starting time of the streaming flow is after 15 seconds, the entry of the streaming flow causes a congestion in almost every repetition, which is avoided very rarely by an ECN mark. Due to the congestion, the bulk TCP flows reduce their congestion windows, which adjusts the incoming rate. With the new rate, the last-hop router is not suddenly overloaded and packets are ECN marked when the queue is getting longer. Thus additional congestion is prevented.

The stability of the elapsed time of the faster flow is affected by ECN marks that touch only the slower flow. Because of the different sized congestion windows, a gap between the progress of the bulk TCP flows starts to grow. When they complete, there is a large difference.

The minimum simultaneous throughput of the slower flow is 0 B/s, which is caused by the recovery that lasts the duration of the streaming flow. As the simultaneous throughput counts only new data segments, the simultaneous throughput becomes zero.

Baseline TCP with Prioritization for Streaming

When Baseline TCP is used with prioritization, the median overall throughput is 7906 B/s and even the minimum has roughly the same value. The streaming throughput is 3803 B/s without any variations. There are no drops for the streaming flow. For streaming packets, the minimum end-to-end delay is the same as with Baseline TCP, the median is 30% shorter and the maximum is 41% shorter than with Baseline TCP. IPDV is roughly the same as with Baseline TCP.

The end-to-end delay of streaming packets is stable because the audio class dequeues the packets at the same rate as they arrive and because of the link send buffer that is full almost all the time. The median is determined by the length of the link send buffer that is out of reach of DiffServ and full almost all the time. The queuing occurs in audio class only because of transmission delays of TCP packet, which is longer than the arrival interval of the streaming packets. The interquartile range of the end-to-end delay is caused by the amount of IP-layer buffering.

Let us consider why IPDV is not shorter with prioritization. The link is able to re-

lease up to three packets from the link send buffer before the sender of the streaming flow generates a new packet. The first of them is close to the end of its transmission delay and the transmission of the two following streaming packets takes 18 ms. Together the delay is below 20 ms that is the sending interval. These three packets make room for three new packets, which are then transferred over the link one after another. When the audio-class queue is empty, those three packets are TCP packets from the bulk TCP queue. Their transmission takes three times 72 ms. That delay then accumulates to the delay of the next streaming packet that cannot bypass those three TCP packets in the link send buffer. When the link send buffer is not full, a TCP burst can push even more than three intervening TCP packets between two streaming packets because all TCP packets go directly to the link send buffer. Due to the limitations of Seawind, the transfer between the last-hop router and the link send buffer is always in units of packets. Therefore dequeuing from the bulk TCP queue cannot be prevented when the whole TCP packet does not immediately fit to the link send buffer.

Despite of prioritization, the end-to-end delay remains still too high for a real-time use and IPDV is not improved. Since the delay median is caused by the full link send buffer, a shorter end-to-end delay can be obtained by configuring the link send buffer size to a smaller value. IPDV, on the other hand, is pushed down by limiting the size of the TCP class bursts from the last-hop router to the link send buffer. Both of these modifications are done at the expense of an ability to use the link-level retransmissions efficiently. With a link send buffer that can hold only a single packet and with the TCP class that is speed-limited using the nominal link bandwidth, the maximum end-to-end delay falls down to 596 ms, the upper quartile to 446 ms, and the median to 410 ms, and the maximum IPDV is reduced to 141 ms.

The start of the streaming flow causes an immediate bandwidth variation for the bulk TCP flows because the last-hop router allocates the required bandwidth for the streaming packets. If the streaming flow begins before the slow-start overshoot of the bulk TCP flows, it effectively reduces the bandwidth that is visible for the bulk TCP flows. The effect of the bandwidth reduction is very stable as the audio class has a higher priority, and therefore the streaming flow does not experience any congestion that occurs with Baseline TCP. With the reduced bandwidth, the bulk TCP queue of the last-hop router overflows earlier. The entry of the streaming flow during the slow start of the bulk TCP flows causes similar unfairness in the overshoot recovery as with Baseline TCP. With later starting times, the entry of the streaming flow immediately lengthens the queuing delay of the TCP packets and finally last-hop router's bulk TCP queue overflows. This congestion fairly forces both bulk TCP flows to reduce their congestion windows. Therefore the fairness between the elapsed times of the bulk TCP flows is good when the starting time of the streaming flow is later than 13 seconds.

Baseline TCP with ECN and Prioritization for Streaming

When Baseline TCP is used with ECN and prioritization, the median overall throughput is 7902 B/s. The streaming throughput is 3803 B/s without any variation as with Baseline TCP and prioritization. For streaming packets, the minimum and maximum values of the end-to-end delay remain roughly the same as with Baseline TCP and prioritization. The median end-to-end delay is 30% shorter than with Baseline TCP. The IPDV quartiles are the same as with Baseline TCP and the maximum is 277 ms.

RED is able to control the queue length of the bulk TCP class because with prioritization it increases slowly after the slow-start overshoot of the bulk TCP flows. Drops of the bulk TCP packets occurring near the end of the streaming flow are avoided with ECN marks, but the entry of the streaming flow still causes congestion. ECN marks for the bulk TCP flows after the end of the streaming flow reduce the congestion window that can cause underutilization of the link, which reduces the stability of the overall throughput. The underutilization happens only in the repetitions when the starting time of the streaming flow is after 8 seconds, and it is very likely to happen when the starting time is between 16-24 seconds.

The maximum IPDV is shorter than in the Baseline TCP case. The difference is exactly the size of the transmission delay of a single TCP packet. In this case, the TCP segments that intervene consecutive streaming packets have one less in a row than with Baseline TCP because of normal variation in the queuing pattern of the last-hop router.

The minimum of the simultaneous throughput for the slower flow is 0 B/s because slow-start overshoot recovery lasts the duration of the streaming flow as was discussed with Baseline TCP and ECN.

Enhanced TCP

With Enhanced TCP, the median overall throughput is 7940 B/s. The median streaming throughput is 3664 B/s. The interquartile range of the streaming throughput is 38% smaller than with Baseline TCP. The median of the dropped streaming packets is 42 packets. The end-to-end delay of the streaming flow is almost the same as with Baseline TCP, only the lower quartile is 6% shorter while the other indices vary less than one percent. Spurious RTOs do not occur.

The median overall throughput is higher than with Baseline TCP because of larger initial window. With the larger initial window, the bulk TCP flows utilize the wireless link fully sooner than with the smaller initial window of Baseline TCP.

In Figure 30 is the number of streaming drops compared with Baseline TCP in individual repetitions. CBI prevents the slow-start overshoot of the bulk TCP flows from occurring. The number of streaming drops is on the same level with Baseline TCP only while the starting time of the streaming flow is between 12-18 seconds. Before 12 seconds the slow-start overshoot related streaming drops are present only in those

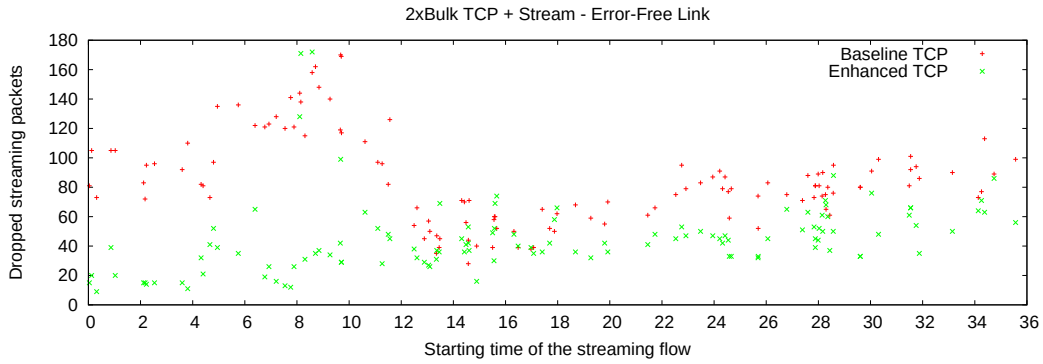


Figure 30: Comparison of streaming drops with Baseline TCP and Enhanced TCP in individual repetitions over error-free link (120 repetitions)

few repetitions that include the slow-start overshoot because of CBI's garbage collector (see Section 4.5). Other repetitions before 12 seconds have slightly less drops than the repetitions in the 12-18 seconds region because the streaming flow acts as a bandwidth limiter for the bulk TCP flows, which causes less severe congestion. The repetitions after 18 seconds have less drops because with Baseline TCP the queuing delay after the overshoot recovery is longer than with Enhanced TCP. A shorter queuing delay means that there are less bulk TCP packets in the last-hop router queue. A larger share of the streaming packets fit into the last-hop router buffer than with Baseline TCP and therefore the entry of the streaming flow causes less congestion drops for the streaming flow.

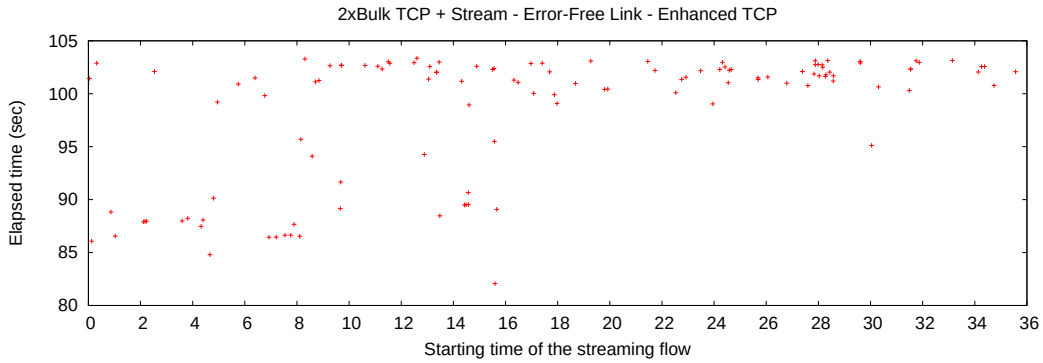


Figure 31: Elapsed time of the faster flow with Enhanced TCP in individual repetitions over error-free link (120 repetitions)

We see that in Figure 27e the median elapsed time has good fairness between the bulk TCP flows while the lower quartile is far away from the median. The elapsed time of the faster flow in individual repetitions is shown in Figure 31. With CBI, the congestion is almost non-existing when the streaming flow starting is times before 12 seconds. Only few bulk TCP drops take place near the end of the streaming

flow. With small number of bulk TCP drops, there is a large probability for unfairness because the drops can hit only one bulk TCP flow that loses a share of the bandwidth. The stability of the elapsed time of the faster flow is not as good as with Baseline TCP because the lower quartile is affected by the larger share of the bandwidth. Probability of a drop pattern that affects both flows increases when the starting time of the streaming flow is later. Such drops fairly affect the congestion windows of the bulk TCP flows.

Enhanced TCP with ECN

When Enhanced TCP is used with ECN, the median overall throughput is 7934 B/s. The median streaming throughput is 3319 B/s. The interquartile range of the streaming throughput is 15% larger than with Baseline TCP. The median of the dropped streaming packets is 146 packets. The end-to end delay median is 33% shorter than with Baseline TCP, but the maximum is almost as high as with Baseline TCP. IPDV is the same as in the Baseline TCP case.

ECN is able to control the progress of the bulk TCP flows from the starting times that are before 8 seconds, because CBI prevents the slow-start overshoot from occurring for the bulk TCP flows. In the repetitions with starting time after 8 seconds, congestion occurs once. As with Enhanced TCP, the probability of affecting both bulk TCP flows fairly increases when the starting time grows. When the starting time is after 15 seconds, the first retransmission in the loss recovery is dropped in every repetition by RED because the queue average is above the maximum threshold and the retransmitted packet is non-ECN-capable. As in Baseline TCP with ECN, consecutive streaming packets are also lost while the RED queue average remains above the maximum threshold.

Enhanced TCP with Prioritization for Streaming

When Enhanced TCP is used with prioritization, the median overall throughput is 7941 B/s. The streaming throughput is 3803 B/s without variation as with Baseline TCP and prioritization. The end-to-end delay of the streaming flow is the same as with Baseline TCP and prioritization. IPDV is roughly the same as with the Baseline TCP except for the maximum that is 286 ms.

The elapsed time of the faster flow in individual repetitions is shown in Figure 32. There are less bulk TCP drops during the occurring congestion than with Enhanced TCP, which increases unfairness of the elapsed times of the bulk TCP flows because the congestion drops more likely affect only one of the bulk TCP flows. These drops start to happen when the starting time of the streaming flow is more than 11.5 seconds. When the starting time is between 11.5 and 16 seconds, the presence of drops depends on the initial ssthresh that is determined by CBI reusing. When the starting time is after 16 seconds, the congestion drops are present in every repetition. In the repetitions having the starting time of the streaming flow after 30 seconds,

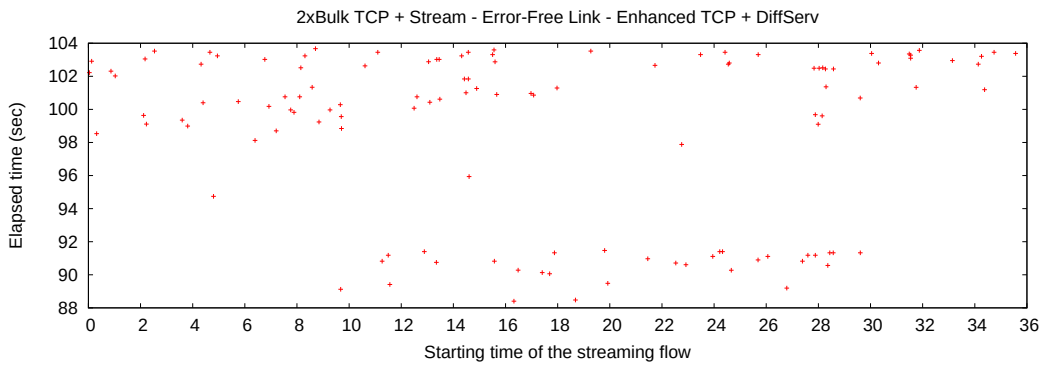


Figure 32: Elapsed time of the faster flow when Enhanced TCP is used with prioritization in individual repetitions over error-free link (120 repetitions)

both flows are affected by the congestion drops that occur, and therefore the fairness of the elapsed times between them is good.

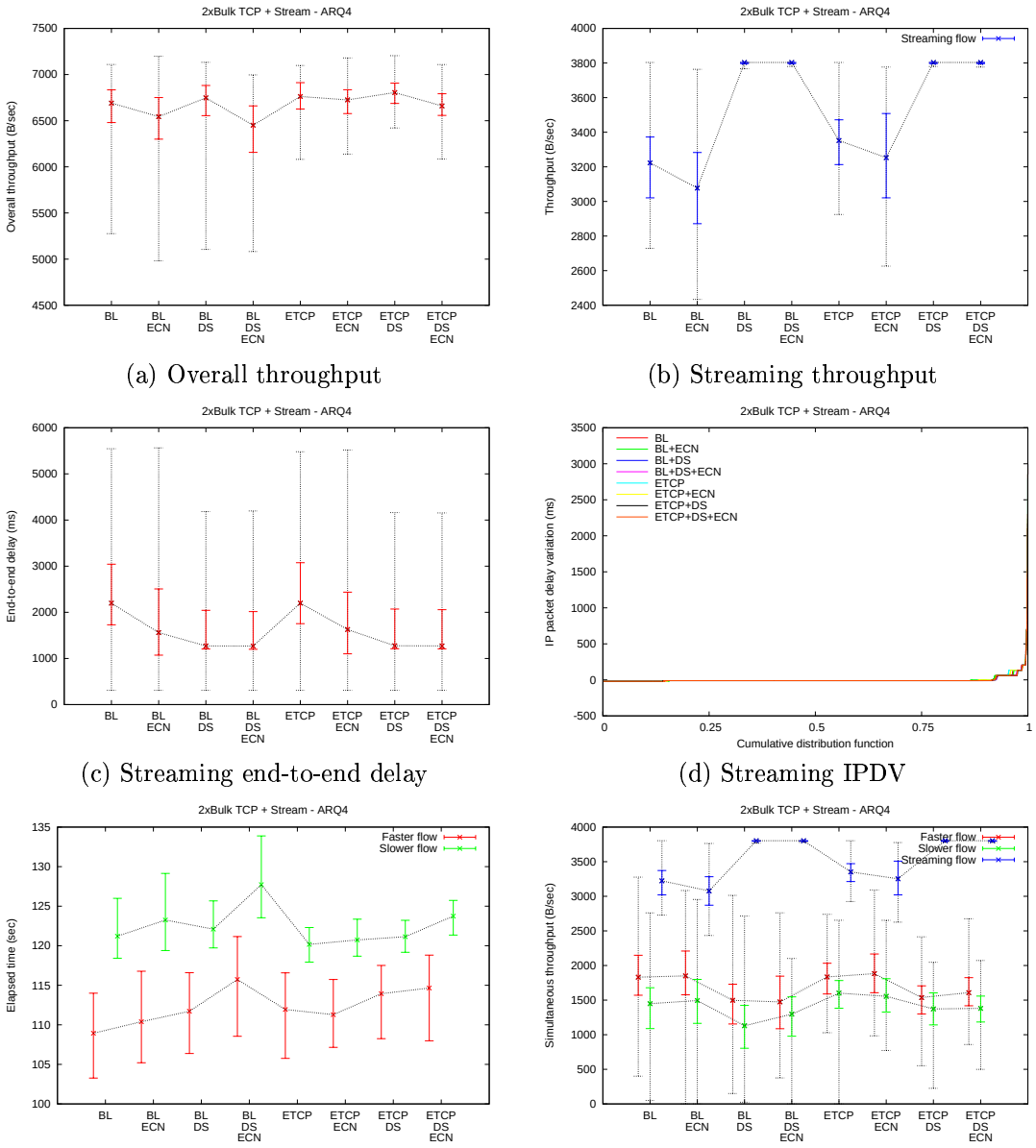
The maximum IPDV is caused by the streaming packets that are transferred when the initial window of a bulk TCP flow is sent between two streaming packets. Because of a link idle period before the arrival of the initial window, the maximum is not calculable directly from the multiples of the transmission delay of the full sized TCP segment.

Enhanced TCP with ECN and Prioritization for Streaming

When Enhanced TCP is used with ECN and prioritization, the median overall throughput is the same as with Enhanced TCP and prioritization. The throughput is constant 3803 B/s as with Baseline TCP and prioritization. The end-to-end delay is not shorter than with Enhanced TCP and prioritization.

As was discussed with Baseline TCP with ECN and prioritization, the stability of the overall throughput is reduced, but less dramatically, because ECN marks occasionally cause underutilization of the link.

In the repetitions where the streaming flow is started before 14 seconds there is not enough congestion that would cause ECN to mark anything during the streaming flow. With later starting times, the ECN marks start to show up, but even then, they appear for the last round-trip during the streaming flow, or show up after the end of the streaming flow altogether. An ECN mark that occurs too late increase unfairness between the bulk TCP flows unless the bulk TCP flows have already unbalanced congestion windows because of congestion that touches only one of the flows.



(a) Overall throughput

(b) Streaming throughput

(c) Streaming end-to-end delay

(d) Streaming IPDV

(e) Elapsed time of the bulk TCPs

(f) Simultaneous throughput

Figure 33: Summary of results over the link with high ARQ persistency in the workload of the competing streaming flow (120 repetitions)

5.4.2 Link with High ARQ Persistency

The overall throughput over the link with high ARQ persistency is shown in Figure 33a. As with the error-free link, the overall throughput is less in the cases with ECN.

In Figure 33b is the streaming throughput. The pattern with the enhancements is similar as with the error-free link. With prioritization, the high and stable through-

put is maintained. With ECN there are more drops for the streaming flow than without ECN. The end-to-end delay is presented in Figure 33c. With the four bad state retransmissions the delay maximum is very long, even with prioritization. There is no relatively no improvement with any enhancement in IPDV shown in Figure 33d.

In Figure 33e are the elapsed times of the faster and slower flow. When Baseline TCP is used with ECN, the elapsed time of the slower flow is less stable than without ECN. When Baseline TCP is used with ECN and prioritization, the elapsed times are longer than in other cases. The spurious RTOs and the unnecessary retransmissions for the bulk TCP flows are almost equal to the case in which the workload of a competing HTTP transfer is transferred over the link with high ARQ persistency (See [Jär06] pages 64–68). In Figure 33f are the simultaneous throughputs. The stability of the simultaneous throughputs and the difference between the medians of the faster and slower flow remain almost the same within the TCP variant. With Enhanced TCP the stability is slightly better than with Baseline TCP.

Baseline TCP

When Baseline TCP is used, the median overall throughput is 6691 B/s. The median streaming throughput is 3223 B/s. The interquartile range of the streaming throughput is 351 B/s. The median of the dropped streaming packets is 174.5 packets. The minimum end-to-end delay is 309 ms, the median is 2197 ms, and the maximum is 5546 ms.

Besides very few error drops, the bulk TCP flows have four sources of drops. First, there are the drops that happen during the slow-start overshoot. Secondly, drops happen when the streaming flow enters, which is a reduction in the available bandwidth from bulk TCP's point of view. The other two are caused by the stalls of the wireless link that happen during the link-level retransmissions. Depending on the events which follow the start of the streaming flow one of these drop sources occurs first. A downlink stall happens when the link send buffer is full while the link-level retransmissions occur. During the retransmissions, the link send buffer cannot receive any packets from the last-hop router that is forced to queue all incoming packets. Another kind of stall happens for the uplink direction, which carries the ACKs. After the uplink stall, the link releases a burst of cumulative ACKs that were waiting at the base-station end for the delivery during the link-level retransmissions. This cumulative ACK burst causes a burst of new TCP segments. The queuing during the downlink stall and the burst of new TCP segments increase the queue length of the last-hop router, which easily overflows because the streaming flow sends new packets with the constant bit-rate. After the ACK burst, a typical loss pattern includes a large consecutive block of the TCP segments, which are all dropped. In the case of the downlink burst or the entry of the streaming flow, the drops are spread over the time of a single round-trip. After the first instance of the congestion, both bulk TCP flows have reduced their congestion windows and cannot regain it before the streaming flow completes. Therefore congestion does not occur

later on. In addition to drops, during the stall, a spurious RTOs can be triggered. These spurious RTOs cause unnecessary retransmissions and unbalance between the bulk TCP flows as was discussed with the workload of a short competing TCP flow (see Section 5.1).

As with the error-free link, the number of streaming drops is very high, the median is 15% of the streaming flow. However, the minimum of the dropped streaming packets is less because of the poor quality periods that prevent bulk TCP flows from opening the congestion window to full bandwidth usage in some of the repetitions. The maximum number of dropped packets on the other hand is much larger because of consecutive losses that happen during and after the stalls.

Baseline TCP with ECN

When Baseline TCP is used with ECN, the median overall throughput is 6544 B/s. The median streaming throughput is 5% lower and the interquartile range of the streaming throughput is 17% larger than with Baseline TCP. The median of the dropped streaming packets is 221 packets. For the streaming packets, the minimum and maximum values of the end-to-end delay are the same as with Baseline TCP, and the median is 29% shorter than with Baseline TCP. The interquartile range of the end-to-end delay is 9% larger than with Baseline TCP.

Since the congestion windows are smaller because of the error drops, the overall throughput is often affect by underutilization of the link. The bulk TCP flows reduce sending rate too much because of ECN marks that happen after one of the flows has slowed down by a recovery or has completed. The events that take place are explained with the competing HTTP transfer in Baseline TCP with ECN and prioritization.

The maximum of the streaming packets dropped is 413, which is 36% of the streaming flow, while the median settles on 19%. As with the error-free link, the state where the queue average is above the maximum threshold causes of the larger number of drops. The difference between this link type and the error-free link is not as large as with Baseline TCP because the drops caused by the queue average phenomenon dominate with both link types.

Luck determines whether one of the four drop sources that are described in the Baseline TCP case occurs before the queue average rises enough for ECN to mark. The shorter queue delay that ECN marks cause is visible in the end-to-end delay whose median is shorter than with Baseline TCP.

Baseline TCP with Prioritization for Streaming

When Baseline TCP is used with prioritization, the median overall throughput is 6747 B/s. The median streaming throughput is 3803 B/s, which is equal to the error-free link results. The median streaming drops is zero and the upper quartile includes two error-related drops. The lower quartile of the throughput falls to 3796

B/s because of the drops. For the streaming packets, the minimum end-to-end delay is the same as with Baseline TCP, the median is 42% shorter and the maximum is 25% shorter than with Baseline TCP. The interquartile range of the end-to-end delay is 36% smaller than with Baseline TCP.

Prioritization of the streaming packets helps in the number of packet drops. Only a few error-related drops happen for the streaming flow. As with the error-free link, the median end-to-end delay of the streaming flow is determined by the size of the link send buffer. It is longer with this link type because of the ARQ protocol which was disabled with the error-free link. The ARQ protocol holds a frame in the link send buffer longer than with the error-free link. The frame is held until the link-level sender has received a confirmation of the frame delivery, which means an additional propagation-delay sized delay.

Baseline TCP with ECN and Prioritization for Streaming

When Baseline TCP is used with ECN and prioritization, the median overall throughput is 6451 B/s. The streaming throughput is the same as with Baseline TCP and prioritization. The median and upper quartile of the streaming drops remain the same as without ECN. The minimum end-to-end delay of the streaming flow is the same as with Baseline TCP, the median is 42% shorter and the maximum is 24% shorter than with Baseline TCP. The interquartile range of the end-to-end delay is 38% smaller than with Baseline TCP.

The overall throughput is lower than with Baseline TCP due to the underutilization of the link. This phenomenon is described in greater detail with the competing HTTP transfer. The streaming flow performance is equal to the case with Baseline TCP and prioritization.

Enhanced TCP

When Enhanced TCP is used, the median overall throughput is 6762 B/s. The median streaming throughput is 4% higher and the interquartile range of the streaming throughput is 26% smaller than with Baseline TCP. The median of the dropped streaming packets is 135.5 packets. The end-to-end delay of the streaming flow is the same as with Baseline TCP.

As with the error-free link, the absence of the slow-start overshoot for the bulk TCP flows increases the throughput of the streaming flow. Most of the streaming drops are caused by the link stalls that are explained in the Baseline TCP case. The fairness of the elapsed time between the TCP flows is better than with Baseline TCP because CBI limits the slow start of the not suffering flow in the repetitions with an error drop or spurious RTO for only the other bulk TCP flow.

Enhanced TCP with ECN

When Enhanced TCP is used with ECN, the median overall throughput is 6725 B/s. The median streaming throughput is 1% higher and the interquartile range of the streaming throughput is 39% larger than with Baseline TCP. The median of the dropped streaming packets is 164 packets. For the streaming packets, the minimum end-to-end delay is the same as with Baseline TCP, the median is 26% shorter, and the maximum is 1% shorter than with Baseline TCP.

As when Baseline TCP is used with ECN, RED causes consecutive drops for the streaming flow but congestion happens less likely because the slow-start overshoot for the bulk TCP flows is avoided by CBI ssthresh reusing.

Enhanced TCP with Prioritization for Streaming

When Enhanced TCP is used with prioritization, the median overall throughput is 6805 B/s. The streaming throughput and the packets dropped are equal to the case with Baseline TCP and prioritization. For the streaming packets, the minimum end-to-end delay is the same as with Baseline TCP, the median is 42% shorter, and the maximum is 25% shorter than with Baseline TCP.

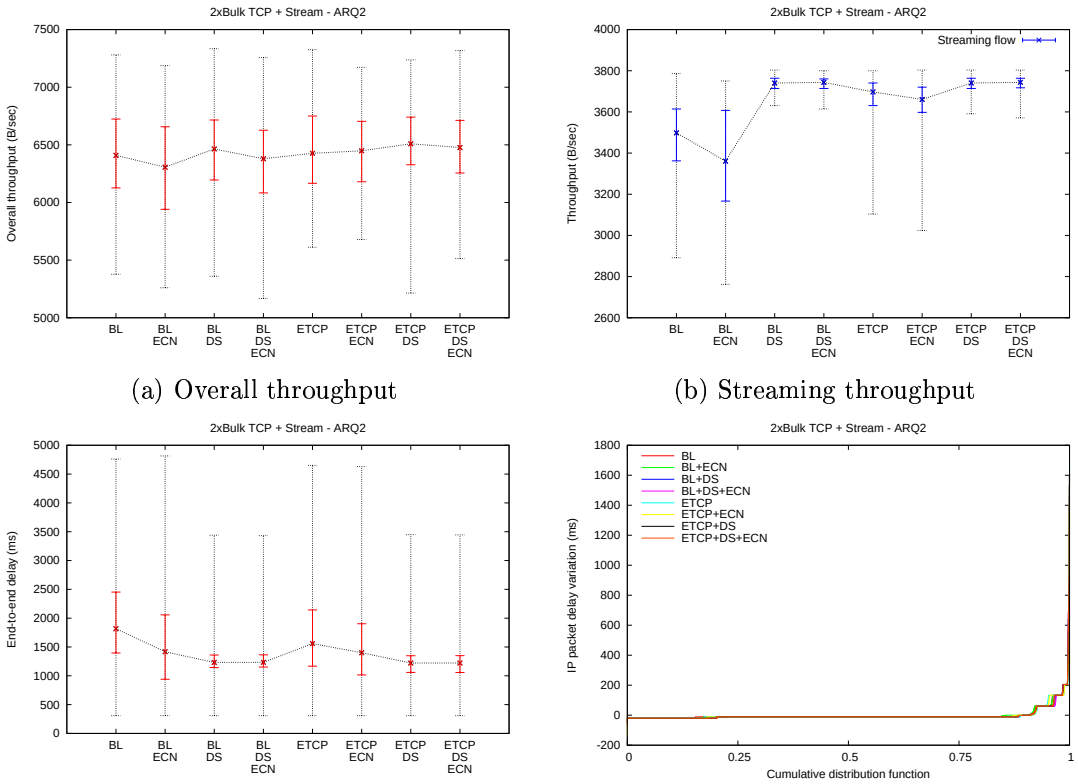
Enhanced TCP with ECN and Prioritization for Streaming

When Enhanced TCP is used with ECN and prioritization, the median overall throughput is 6660 B/s. The streaming throughput, the dropped streaming packets, and the end-to-end delay of the streaming flow are the same as in the case with Enhanced TCP and prioritization.

5.4.3 Link with Medium and Low ARQ Persistency

In Figure 34 is the overall throughput, the streaming throughput, the end-to-end delay, and IPDV over the link with medium ARQ persistency. ECN affects slightly with Baseline TCP. As with high ARQ persistency, the overall throughput is lower, the streaming throughput is lower, the end-to-end delay is shorter, and IPDV is not changed with ECN. The maximum IPDV and the maximum end-to-end delay for the streaming flow are now shorter than with the high ARQ persistency because the link performs a smaller number of link-level retransmissions. With prioritization the overall throughput is slightly higher than without prioritization because of the additional buffer space at the last-hop router.

The streaming throughput varies depending on the congestion drops. With Baseline TCP, the congestion drops during the slow-start overshoot for the bulk TCP flows can be observed. In the median, 8% of the streaming packets are dropped. In Baseline TCP with ECN, the drops of the non-ECN-capable streaming packets are visible but with a lower severity and a smaller probability than with the error-free



(a) Overall throughput

(b) Streaming throughput

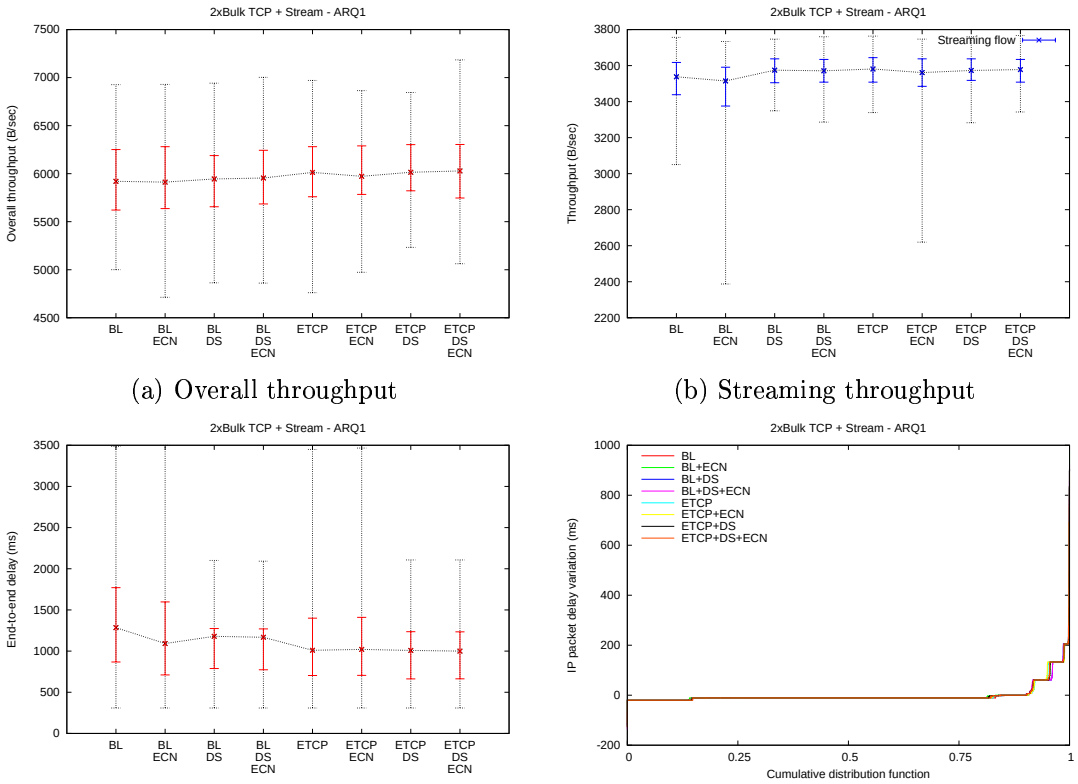
(c) Streaming end-to-end delay

(d) Streaming IPDV

Figure 34: Summary of results over the link with medium ARQ persistency in the workload of the competing streaming flow (120 repetitions)

link. In all 11.6% of the streaming packets are dropped in the median case. With all prioritization cases, the streaming throughput is affected by the error drops. The median error drops is 1.6-1.7% of the packets of the streaming flow. In the Enhanced TCP case, the absence of congestion reduces the median of the drops to 2.8% of the streaming packets. In Enhanced TCP with ECN, the queue average less likely exceeds the maximum threshold of RED. Therefore the probability of a drop for a streaming packet by RED is lower than in Baseline TCP with ECN.

In Figure 35 we see the overall throughput, the streaming throughput, the end-to-end delay, and IPDV over the link with low ARQ persistency. Error drops dominate and they control the last-hop router queue length so that congestion is a rare event. An ECN mark can happen only in the repetitions where the streaming flow starting time is early because in the later starting repetitions the bulk TCP flows have already received error-related drops that reduce the congestion windows. These rare repetitions with ECN marks accumulate to the minimum streaming throughput and very slightly reduce the median streaming drops.



(a) Overall throughput

(b) Streaming throughput

(c) Streaming end-to-end delay

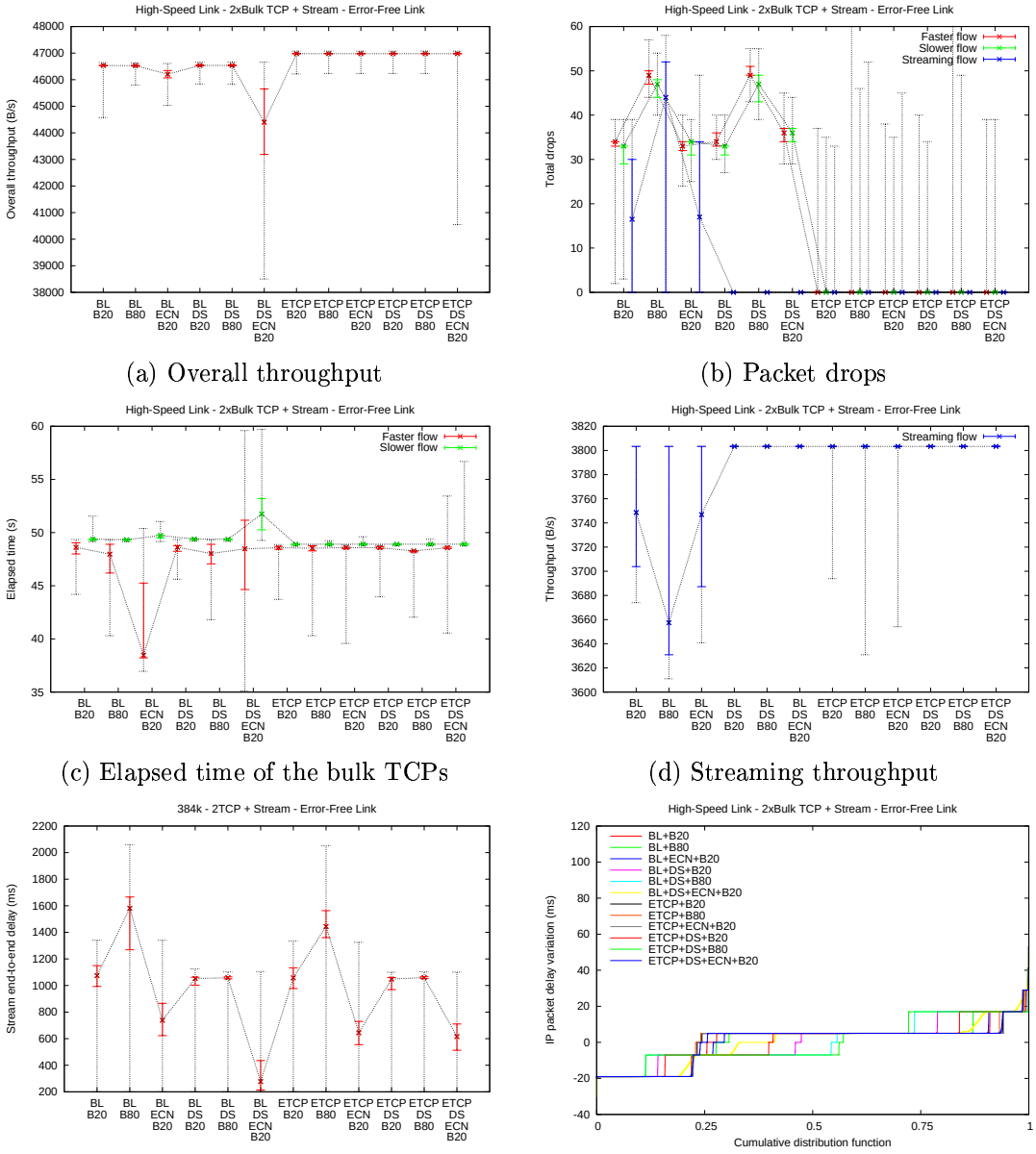
(d) Streaming IPDV

Figure 35: Summary of results over the link with low ARQ persistency in the workload of the competing streaming flow (120 repetitions)

5.4.4 High-Speed Error-Free Link

With the high-speed error-free link, we test with two last-hop router queue sizes: a smaller size is 20 packets (B20) and a larger size is 80 packets (B80). TCP flows did not work with ECN when the larger queue size was used because non-ECN-capable TCP retransmissions in the overshoot recovery were all dropped until the TCP throughput is very poor or the sender gives up. There are other anomalies as well due to rare inaccuracies in Seawind. These anomalies are visible in some minimums and maximums but we do not explain them in detail.

The overall throughput is shown in Figure 36a. The pattern is similar to the results with the slower link speed. The median of the Baseline TCP case is 46535 B/s and the median of all Enhanced TCP cases is 46978 B/s. Enhanced TCP has higher throughput than Baseline TCP because of the larger initial window that allows TCP flows to open congestion window faster as was discussed in Section 5.1. ECN marks cause very serious underutilization of the link in the case of Baseline TCP with ECN and prioritization. The underutilization happens close after the overshoot recovery, which indicates a poor RED configuration. The RED configuration is likely to be suboptimal because it was not tuned for the higher speed link that has a quite



(a) Overall throughput

(b) Packet drops

(c) Elapsed time of the bulk TCPs

(d) Streaming throughput

(e) Streaming end-to-end delay

(f) Streaming IPDV

Figure 36: Summary of results over the error-free high-speed link in the workload of the competing streaming flow (120 repetitions)

different delay-bandwidth product. When Enhanced TCP is used with ECN and prioritization, a similar underutilization of the link occurs causing the minimum overall throughput.

In Figure 36b are the packets dropped for each flow. With Baseline TCP the slow-start overshoot of the bulk TCP flows causes all the drops. The minimums for the bulk TCP flows in the Baseline TCP case are anomalous. In Enhanced TCP, CBI prevents the slow-start overshoot for the bulk TCP flows from occurring except

when CBI garbage collector has disposed the initial ssthresh. Thus, also Enhanced TCP rarely overshoots. With the larger queue size, the slow-start overshoot is more severe causing a larger number of drops. As with the slower link speed, spurious RTOs do not occur.

The elapsed times of the bulk TCP flows are shown in Figure 36c. The slow-start overshoot and its recovery cause slight unfairness between the bulk TCP flows. When Baseline TCP is used with ECN, the progress of the bulk TCP flows starts often to diverge right after the overshoot recovery because of unbalanced ECN marks. Because ECN marks do not occur from that time forward at all or they occur slightly before the faster flow completes, the unfairness is never recovered. So that unfairness is a short of a quantification error of the RED algorithm. If transfers would last longer, the median case could not be as bad as more ECN marks would occur later, which would balance the elapsed times with high probability. With Baseline TCP and the larger queue size the unfairness after the overshoot recovery is greater than with the smaller queue size. Hence the stability of the elapsed time is inferior to the smaller queue size case. The poor results with prioritization and ECN are a result of the serious underutilization. Since there are no drops in the Enhanced TCP cases, the bulk TCP flows advance equally, which explains the good stability of the elapsed times.

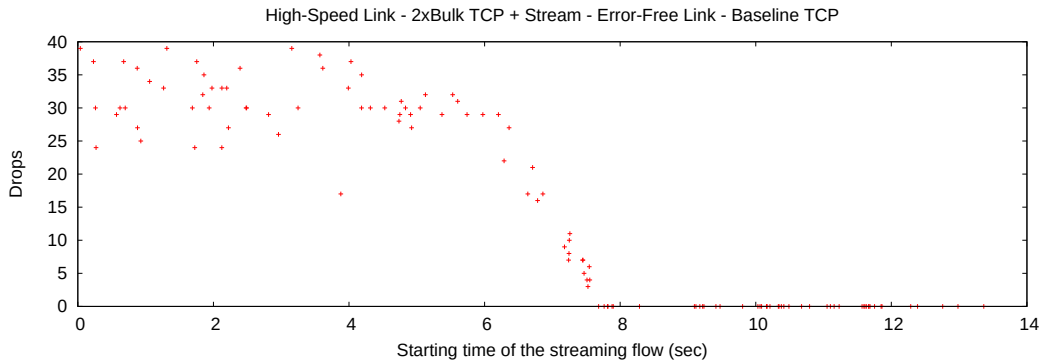


Figure 37: Drops for the streaming flow with Baseline TCP in individual repetitions over high-speed error-free link (120 repetitions)

The bulk TCP flows disturb the streaming flow less than with the slower link speed, even without prioritization. As we can see from Figure 37, which shows the streaming drops in individual repetitions, with Baseline TCP up to 39 streaming packets are lost during the slow-start overshoot. The number of the drops decreases with the later streaming flow starting times because an overlap with the overshoot is shorter. The median and the upper quartile of the consecutive packets dropped are two packets with Baseline TCP, and the maximum is six packets. With ECN, RED's queue average does not rise above the maximum threshold as with the slower link speed except very rarely causing up to 11 consecutive losses. With the larger queue size, the number of streaming drops is larger because of a longer slow-start overshoot but otherwise performance is very similar. With prioritization the slow-start overshoot

does not affect the streaming flow. With Enhanced TCP without prioritization the slow-start overshoot is happening rarely. Hence, no drops occur in the median case.

The streaming throughput is shown in Figure 36d. As with the slower link speed, it is stable with the prioritization. Without prioritization, the drops occurring during the slow-start overshoot reduce the throughput.

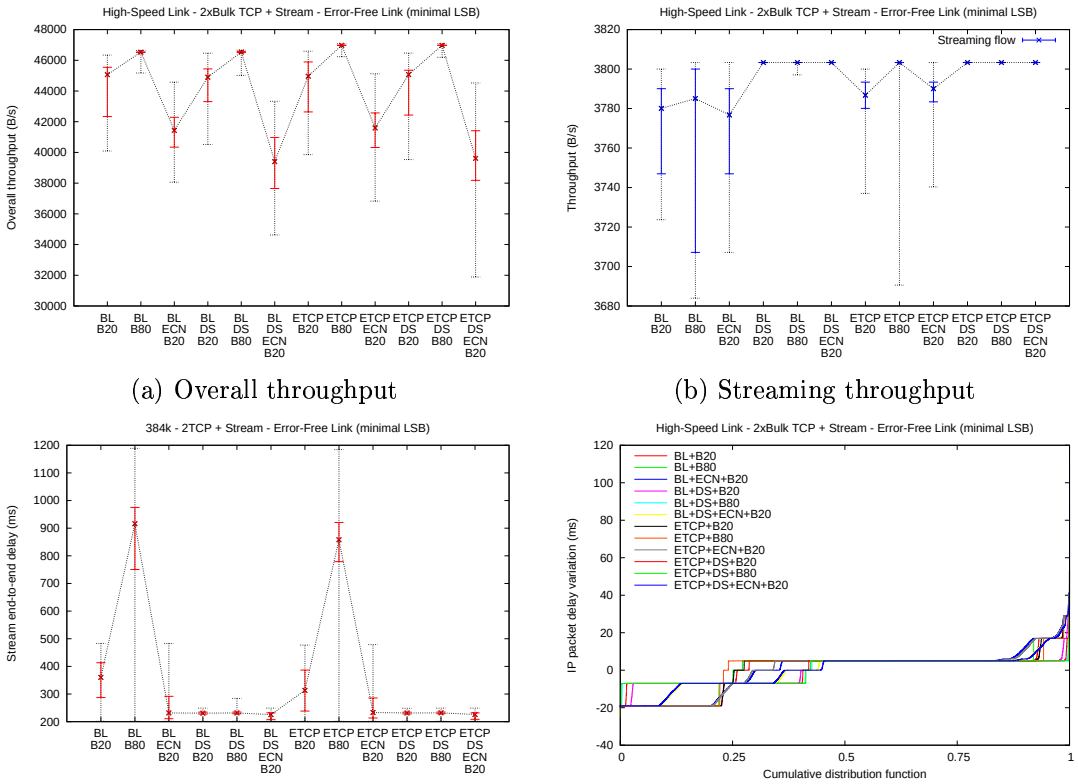
The end-to-end delay is shown in Figure 36e. The queuing delay in the link send buffer is observable from the case of Baseline TCP with prioritization and the larger queue size as discussed with the slower link speed. Because the last-hop router queue size is inadequate for optimal TCP performance, the end-to-end delay with the smaller queue size without prioritization is not considerably longer as it was with the lower link speed. ECN works as with the slower link speed. When Baseline TCP is used with the smaller queue size, ECN, and prioritization, the median end-to-end delay is very low because the duration of the underutilization is long. Therefore the queuing delay during the streaming flow is very low.

IPDV is shown in Figure 36f. Most configurations have the larger median (5 ms) with equal quartiles, which indicates two intervening TCP segments between the streaming packets as discussed with the slower link speed. These packets are a result of a delayed ACK that acknowledges two segments, which allow at least two new segments. The shorter median (-7 ms) has just one intervening TCP segment, but the upper quartile is 17 ms. It happens in the case where prioritization is used with the larger queue size. Because of adequate buffering, the link send buffer remains full during whole transfer. Therefore less bulk TCP packets are moved from the last-hop router to the link send buffer before the next streaming packet arrives to the last-hop router. The minimum of the Baseline TCP with ECN and prioritization is anomalous.

Minimal Link Send Buffer

Now we set the link send buffer to one MTU because the end-to-end delay of the streaming flow is shorter with a lower link send buffer as was discussed with the slower link speed. The overall throughput with the minimal link send buffer is shown in Figure 38a. The best overall throughput is achieved with the larger router queue size. With the smaller queue size, the link starves often, which reduces the overall throughput. With ECN the last-hop router queue does not overflow except during the slow-start overshoot, but often underutilization of the link happens after an ECN mark. Also the elapsed times of the bulk TCP flows vary because of the underutilization.

In Figure 38b is the streaming throughput. The streaming throughput with Baseline TCP is higher than with the larger link send buffer because the number of drops is smaller during the slow-start overshoot. When the link send buffer is downsized, the total buffering capacity is smaller, which reduces the number of drops. Again, prioritization guarantees a good throughput. Contrary to the case with a larger link send buffer, when Enhanced TCP is used without prioritization, there is a con-



(a) Overall throughput

(b) Streaming throughput

(c) Streaming end-to-end delay

(d) Streaming IPDV

Figure 38: Summary of results over the error-free high-speed link with the minimal link send buffer in the workload of the competing streaming flow (120 repetitions)

gestion during which some streaming packet are lost because the buffering capacity is smaller. The streaming end-to-end delay is shown in Figure 38c. It follows the guidelines that were discovered with the lower link speed. When prioritization is not used, the queuing delay increases the end-to-end delay. ECN reduces it as with the larger link send buffer. With prioritization the end-to-end delay is low and very stable. IPDV is shown in Figure 38d. The median IPDV is 5 ms. With prioritization the end-to-end and IPDV values are good enough for the real-time streaming. Without prioritization the end-to-end delay depends on the queuing delay, which is varies based on the last-hop router queue length. Therefore when the queue is long, many consecutive streaming packets have an end-to-end delay that does not meet the requirements of the real-time streaming. Thus the results with ECN do not actually meet the requirements of the real-time streaming in a short time-scale.

5.4.5 Summary of Results

Without prioritization the streaming flow suffers from congestion. A large number of congestion drops occur during the slow-start overshoot of the bulk TCP flows. With lower link speed, drops occur also when the last-hop router is congested later. With

ECN the RED algorithm drops many streaming packets when the queue average is above the maximum threshold. When prioritization is used, the stream end-to-end delay, the streaming throughput, and stream IPDV are equal regardless of the TCP configuration because the end-to-end delay is dominated by the size of the link send buffer, no congestion drops occur for the streaming flow, and IPDV is dominated by the occurring back-to-back bursts. Only when the link send buffer is shorter than almost full, the end-to-end delay is also shorter. Therefore prioritization cannot provide drastic improvement, but another solution is necessary when the transmission delays are too long for the requirements of the real-time streaming. When the link is not fully utilized, the overall throughput suffers, which occurs especially with the high-speed link when the queue sizes are inadequate for TCP's performance.

Because of congestion drops, the Baseline TCP cases have good fairness between the bulk TCP flows regardless of prioritization when the error-free link is used. The congestion drops affect both flows quite equally. When ECN is employed in those cases, the fairness is inferior because of ECN marks that cause inequality that is not always recovered later. When Enhanced TCP is used, fairness without ECN is inferior to the Baseline TCP case with the slower link speed because congestion drop can unfairly touch only one bulk TCP flow. Therefore ECN improves fairness between the bulk TCP flows. With the high-speed link and Enhanced TCP, the fairness between the bulk TCP flows is very good because no drops occur. These effects are visible only on the error-free link because when the link is lossy, the effects of the link-level retransmissions and error drops dominate. With the high-speed link, the larger queue size, and ECN completing the test runs was impossible because of a high RED queue average.

5.5 Two Bulk TCP Flows Competing with an HTTP Transfer and a Streaming Flow

In the workload with a competing HTTP transfer and a competing streaming flow, there are two bulk data TCP transfers, one HTTP transfer, and one streaming UDP flow. The bulk TCP flows are started at zero seconds. We refer to them as the faster and slower flow. These tests were ran with only 40 repetitions.

We report the performance of the whole workload using the overall throughput. Only downlink direction is considered as was discussed in Section 5.3. The value includes headers and therefore its theoretical maximum is 8000 B/s. With the HTTP transfer, the main focus is on the response time that is very important from the user perspective. Other TCP metrics are used, but only for a technical view. In the analysis of the streaming flow, we use the streaming throughput, the end-to-end delay, IPDV (IP packet delay variation), and the number of packets dropped. The number of streaming packets dropped is inversely proportional to the streaming throughput. In analysis of the faster and slower flow, we use elapsed time, spurious RTOs, and unnecessary retransmissions. In addition, the simultaneous throughputs

of all flows are shown for a comparison.

For each link type we use figures that compare the results with Baseline TCP (BL), Baseline TCP with ECN (BL+ECN), Baseline TCP with DiffServ prioritization for streaming (BL+DS-U), Baseline TCP with ECN and DiffServ prioritization for streaming (BL+DS-U+ECN), Baseline TCP with DiffServ prioritization for HTTP and streaming (BL+DS), Baseline TCP with ECN and DiffServ prioritization for HTTP and streaming (BL+DS+ECN), Enhanced TCP (ETCP), Enhanced TCP with ECN (ETCP+ECN), Enhanced TCP with DiffServ prioritization for streaming (ETCP+DS-U), Enhanced TCP with ECN and DiffServ prioritization for streaming (ETCP+DS-U+ECN), Enhanced TCP with DiffServ prioritization for HTTP and streaming (ETCP+DS), and Enhanced TCP with ECN and DiffServ prioritization for HTTP and streaming (ETCP+DS+ECN). We analyze the cases in that order. The figures show the median and the quartiles with a solid errorbar for the given metrics. When present, a dotted errorbar shows the minimum and the maximum.

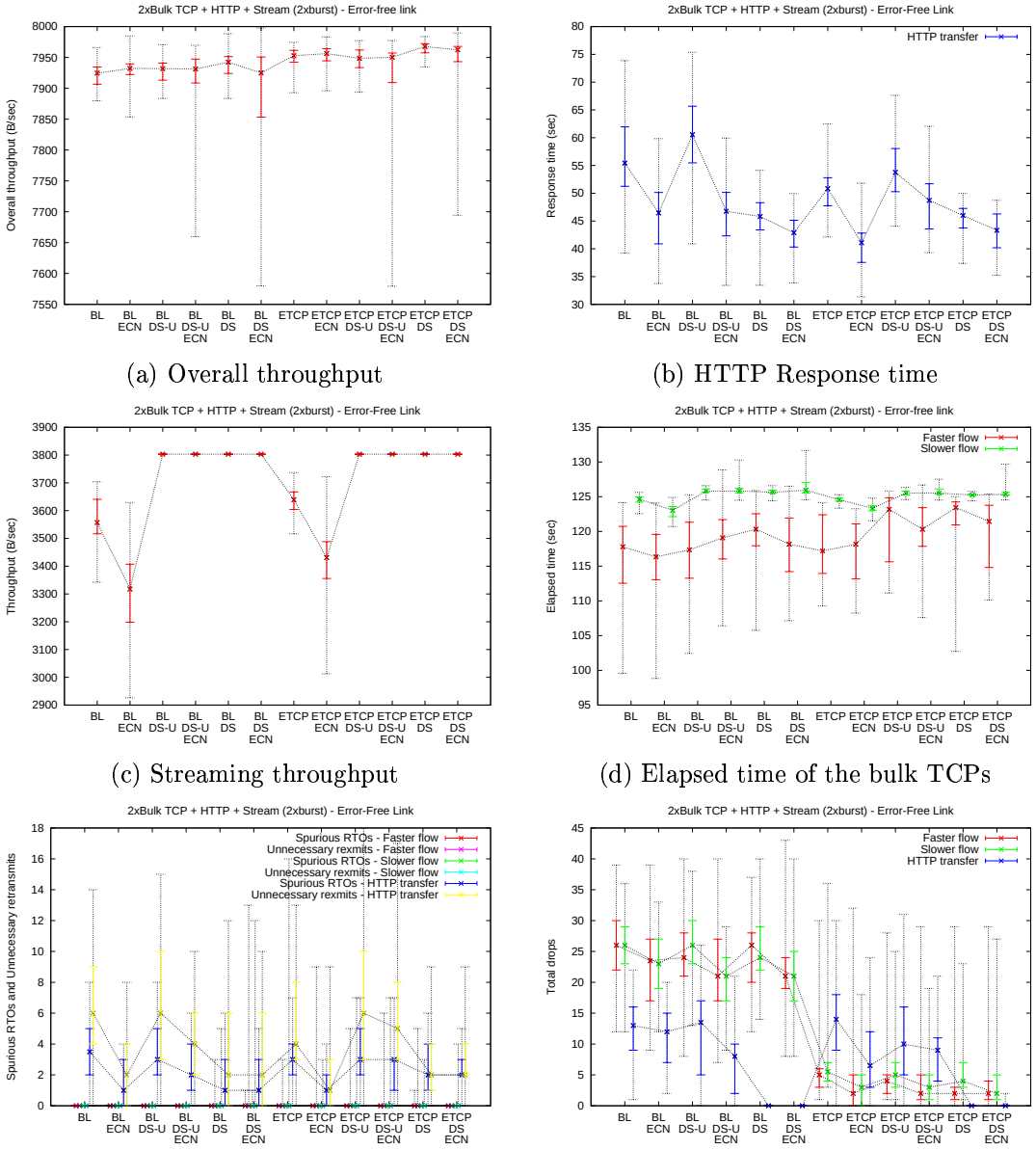
5.5.1 Error-Free Link

In Figure 39a is the overall throughput over the error-free link. As with the other workloads, the throughput with Enhanced TCP is higher than with Baseline TCP because of the larger initial window. Also similarly, in the case of Baseline TCP with DiffServ and ECN the throughput suffers because of the depletion of the link send buffer.

In Figure 39b is the HTTP response time. With Enhanced TCP shorter response times are achieved than with the same configuration with Baseline TCP. With ECN a larger reduction is achieved than in the workload of a competing HTTP transfer. Besides that, the performance with prioritization for HTTP and streaming is what is expected based on the results obtained with the workload of a competing HTTP transfer.

In Figure 39c is the streaming throughput. The pattern of different configurations is similar to the one that is seen with the competing streaming flow. As expected, with prioritization for streaming the throughput is constantly 3803 B/s because no streaming packets are dropped. Without prioritization many drops occur for the streaming packets during the slow-start overshoot when Baseline TCP is used. When Enhanced TCP is employed, CBI prevent the slow-start overshoot from occurring and therefore the throughput is higher than with Baseline TCP. In the cases with ECN, the throughput suffers because the non-ECN-capable streaming packets are dropped when the queue average is above the maximum threshold. The end-to-end delay and IPDV are not shown because they are similar to the values that are obtained with the competing streaming flow.

In Figure 39d are the elapsed times of the bulk TCP flows. The elapsed time of the slow flow depends on the number of streaming drops as with the workload of a competing streaming flow. The stability of the elapsed time of the faster flow and fairness between the elapsed times of the bulk TCP flows is best in the cases



(a) Overall throughput

(b) HTTP Response time

(c) Streaming throughput

(d) Elapsed time of the bulk TCPs

(e) Spurious RTOs and unnecessary retransmissions (for HTTP total over 9 connections)

(f) Packets dropped (for HTTP total over 9 connections)

Figure 39: Summary of results over the error-free link in the workload of competing HTTP transfer and streaming flow (40 repetitions)

that have prioritization for both HTTP and streaming but do not employ ECN. The other cases do not show clear trends. The number of spurious RTOs and unnecessary retransmits is shown in Figure 39e. Without prioritization, ECN reduces the number of spurious RTOs as with the workload of a competing HTTP transfer. Hence, also the number of the unnecessary retransmissions are reduced with ECN. With prioritization for streaming, the number of spurious RTOs are smaller but less dramatically than without prioritization. With prioritization for HTTP and stream-

ing, the number of spurious RTOs and unnecessary retransmissions is not affected by ECN.

The packets dropped by the bulk TCP flows and the HTTP transfer are shown in Figure 39f. ECN marks prevent some instances of congestion, which reduces the number of drops in all configurations.

Baseline TCP Without Prioritization

When Baseline TCP is used, the median overall throughput is 7925 B/s. The interquartile range of the overall throughput is 27 B/s. The median HTTP response time is 55.4 seconds, and the interquartile range of the response time is 10.7 seconds. For the HTTP transfer, the median packets dropped is 13 packets, the median spurious RTOs is 3.5, and the median unnecessary retransmissions is six packets. The median streaming throughput is 3557 B/s.

The largest delaying factor that affects the HTTP transfer is a long queuing delay experienced by most of the packets. In addition to the slow-start overshoot, there are one to three instances of the congestion that take place during the repetition. These congestions affect the HTTP response time, but the effect is large only when the last object is considerably delayed. Such a delay occurs when a congestion drop happens for the last inline object or through a cascade effect from a congestion drop delaying an earlier object so that the last object is started later. If one of the three last packets of the HTTP object is dropped, not enough duplicate ACKs are generated, and the object waits for RTO as with short competing TCP flow.

When Baseline TCP is used with ECN, the median overall throughput is 7932 B/s. The median HTTP response time is 16% shorter and the interquartile range of the response time is 17% smaller than with Baseline TCP. For the HTTP transfer, the median spurious RTOs is one and the median unnecessary retransmissions is two packets. The median streaming throughput is 3318 B/s.

The load caused by the streaming flow is very stable from RED's point of view, and it raises the queue average rapidly to the point where ECN starts to mark. Because the bulk TCP flows have large congestion windows, they are more likely to get marked and the small windowed HTTP objects proceed with less marks. Therefore the HTTP response times are shorter at the expense of the throughputs of the bulk TCP flows.

The first instance of the congestion after the slow-start overshoot is too quick for ECN that does not mark until on the round-trip that includes the congestion drops. Good ECN marks, however, successfully remove the second or third instance of the congestion. The absence of congestion helps the small HTTP objects to complete without drops in the last packets because during the transfer of the latest object the queue length is then tightly in the hand of ECN. Therefore they do not need to wait for RTO that was necessary with Baseline TCP.

A large number of streaming packet drops happen during the period on which the

last-hop router queue average is above the maximum threshold as with the competing streaming flow.

Baseline TCP with Prioritization for Streaming

When Baseline TCP is used with prioritization for streaming, the median overall throughput is 7932 B/s. The median HTTP response time is 9% longer and the interquartile range of the response time is 5% smaller than with Baseline TCP. The response times are longer because DiffServ at the last-hop router prioritizes streaming packets, which delays all TCP and HTTP segments.

When Baseline TCP is used with ECN and prioritization for streaming, the median overall throughput is 7931 B/s. The median HTTP response time is 16% shorter and the interquartile range of the response time is 27% smaller than with Baseline TCP. With ECN the bulk TCP queue is kept short by ECN marks. Therefore the response times are almost equal to the Baseline TCP with ECN case, only the lower quartile is slightly longer.

We see in Figure 39f that in this case there are less drops than in the cases we have seen so far. In over 25% of the repetitions, the drops occurring during the slow-start overshoot are the only drops for the bulk TCP flows because ECN marks prevent congestion. The remaining repetitions are limited to a single instance of the congestion except in a single repetition where the congestion occurs twice after the slow-start overshoot.

Baseline TCP with Prioritization for HTTP and Streaming

When Baseline TCP is used with prioritization for HTTP and streaming, the median overall throughput is 7942 B/s. The median HTTP response time is 17% shorter and the interquartile range of the response time is 54% smaller than with Baseline TCP. The separation of the HTTP and bulk TCP packets in DiffServ eliminates all HTTP drops, shortens the median HTTP response time, and improves the stability of the response time. The separation allocates part of the bandwidth for the HTTP transfer. Therefore it does not have to compete in the bulk TCP queue, which is almost full whole the time.

In some repetitions the bulk TCP flows experience congestion twice after the slow-start overshoot, but mostly congestion occurs once. The bulk TCP-class queue absorbs well the variation of load that happens when the streaming flow starts, and only the queuing delay becomes longer.

When Baseline TCP is used with ECN and prioritization for HTTP and streaming, the median overall throughput is 7924 B/s. The stability of the overall throughput is inferior to any other last-hop router configuration. The median HTTP response time is 23% shorter and the interquartile range of the response time is 55% smaller than with Baseline TCP.

The link is occasionally underutilized, which reduces the stability of the overall throughput as was discussed in greater detail with the workloads of a competing HTTP transfer (see Section 5.3).

ECN restricts the bulk TCP flows, which now congest their queue at the last-hop router only once, except in some repetitions where ECN prevents all drops after the slow-start overshoot.

Enhanced TCP Without Prioritization

When Enhanced TCP is used, the median overall throughput is 7953 B/s. The overall throughput is more stable than with Baseline TCP. The median HTTP response time is 8% shorter and the interquartile range of the response time is 53% smaller than with Baseline TCP. For HTTP transfer the median spurious RTOs is three, the median packets dropped is 14 packets, and the median unnecessary retransmissions is six packets. The median streaming throughput is 3639 B/s, and the median of the streaming packets dropped is 99.

The HTTP response time is shortened most when the starting time of the HTTP transfer is close to zero seconds. The larger initial window allows HTTP objects to open congestion window faster, and the CBI prevents the bulk TCP flows from overshooting. When the starting time gets larger, the larger initial window starts to cause congestion. Because of that, the number of HTTP packets dropped increases.

The F-RTO prevents some of the unnecessary retransmissions as with the competing HTTP transfer.

The slow-start overshoot is not happening due to CBI, and ECN shortens the queuing delay. Because of these reasons, the last-hop router drops less streaming packets. These reasons are discussed in greater detail in the analysis of the workload of a competing streaming flow.

When Enhanced TCP is used with ECN, the median overall throughput is 7956 B/s. The median HTTP response time is 26% shorter and the interquartile range of the response time is 51% smaller than with Baseline TCP. The streaming throughput is 3431 B/s, and the median of the streaming packets dropped is 224.5 packets.

The ECN marks work well with the shared queue. The entry of the streaming flow increases rapidly the queue average, which is low because CBI sets a conservative initial ssthresh. Therefore an ECN response comes quite early and is well distributed between the bulk TCP flows, which kindly slow down. As a result, the HTTP response time is very good.

Enhanced TCP with Prioritization for Streaming

When Enhanced TCP is used with prioritization for streaming, the median overall throughput is 7949 B/s. The median HTTP response time is 3% shorter and the interquartile range of the response time is 28% smaller than with Baseline TCP. The

prioritization for streaming raises the response time against the Enhanced TCP case as in Baseline TCP with prioritization for streaming.

When Enhanced TCP is used with ECN and prioritization for streaming, the median overall throughput is 7950 B/s. The median HTTP response time is 12% shorter and the interquartile range of the response time is 24% smaller than with Baseline TCP.

The separation of the streaming flow into own queue affects the load of the TCP-class queue. The queue average rises slowly because CBI's ssthresh reusing prevents the slow-start overshoot. When all transfers shared a single queue, there was the constant load of the streaming flow. The HTTP transfer being very bursty only lengthens queuing delay of the bulk TCP packets but does not sum up to the average queue length of RED. Therefore ECN fails to mark the bulk TCP flows efficiently, and the HTTP transfer suffers from longer queuing delay and cannot reach as good response time as in the case with Baseline TCP of the same configuration.

Enhanced TCP with Prioritization for HTTP and Streaming

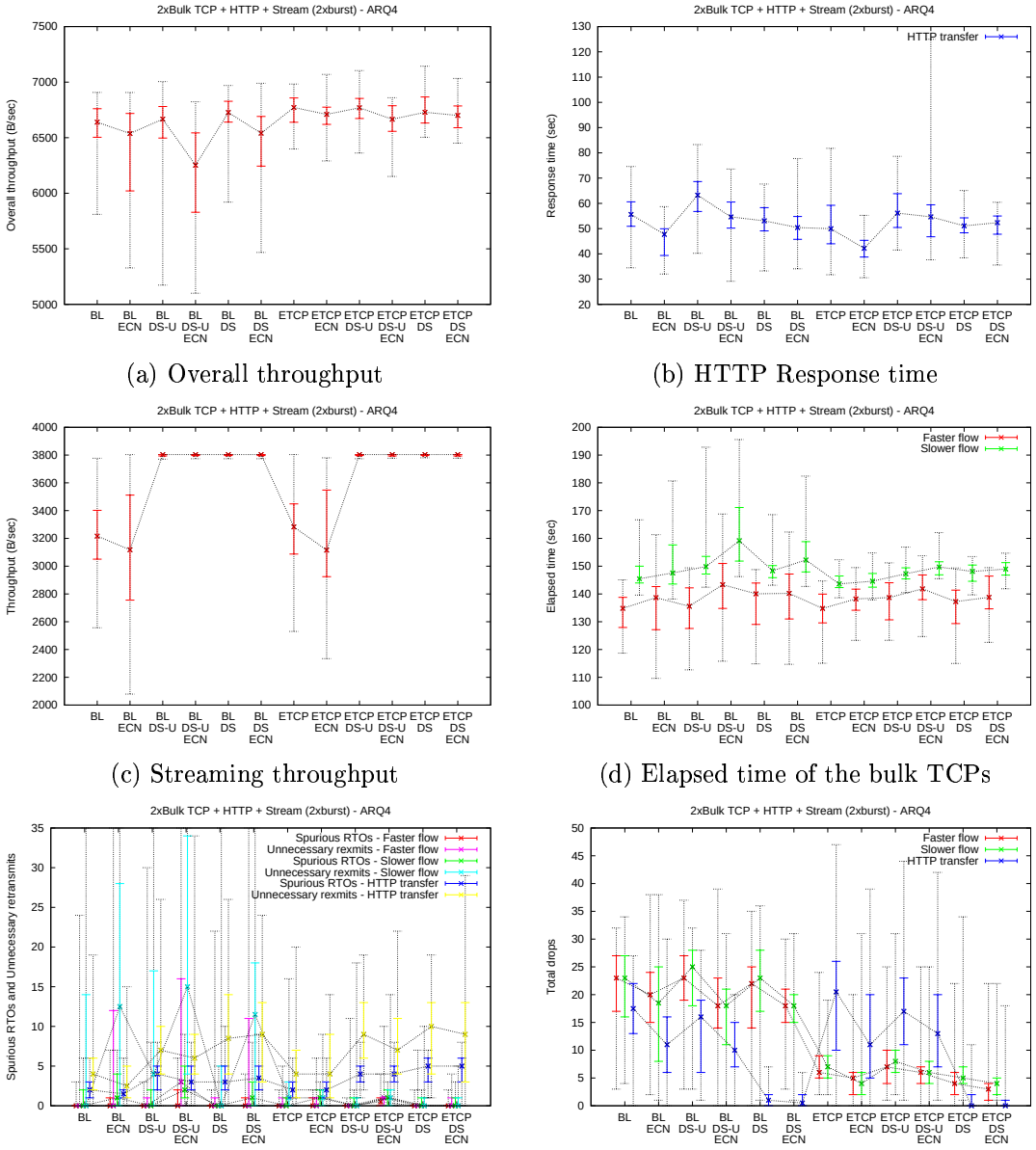
When Enhanced TCP is used with prioritization for HTTP and streaming, the median overall throughput is 7968 B/s. The median HTTP response time is 17% shorter and the interquartile range of the response time is 67% larger than with Baseline TCP. The response time of the HTTP transfer is already very good in Baseline TCP with prioritization for HTTP and streaming. Therefore Enhanced TCP cannot improve the response time without congestion.

When Enhanced TCP is used with ECN and prioritization for HTTP and streaming, the median overall throughput is 7962 B/s. The median HTTP response time is 22% shorter and the interquartile range of the response time is 43% larger than with Baseline TCP. As without ECN, the response time is not shortened from the Baseline TCP case with the same last-hop router configuration.

5.5.2 Link with High ARQ Persistency

In Figure 40a is the overall throughput over the link with high ARQ persistency. As we have seen with the high ARQ persistency link in the other workloads, the Enhanced TCP case have a higher throughput than the Baseline TCP case, and the ECN cases have a slightly lower overall throughput than the cases without ECN.

In Figure 40b is the HTTP response time. The pattern is similar to the error-free link cases. As with the error-free link, the response time varies largely because of the varying queuing delay, the number of drops and the number of spurious RTOs. However, the response time with prioritization for HTTP and streaming is inferior to the case where ECN is employed alone, regardless of the TCP variant. A shared queue causes larger number of ECN marks for the bulk TCP flows. With smaller congestion windows, the queue of the last-hop router is occasionally non-full, which does not happen with prioritization for HTTP and streaming. Thus, the bulk TCP



(a) Overall throughput

(b) HTTP Response time

(c) Streaming throughput

(d) Elapsed time of the bulk TCPs

(e) Spurious RTOs and unnecessary retransmissions (the largest maximums are cropped, for HTTP total over 9 connection)

(f) Packets dropped (for HTTP total over 9 connections)

Figure 40: Summary of results over the link with high ARQ persistency in the workload of competing HTTP transfer and streaming flow (40 repetitions)

flows interfere less with the HTTP transfer, which encounters shorter queuing delay and is therefore able to complete faster.

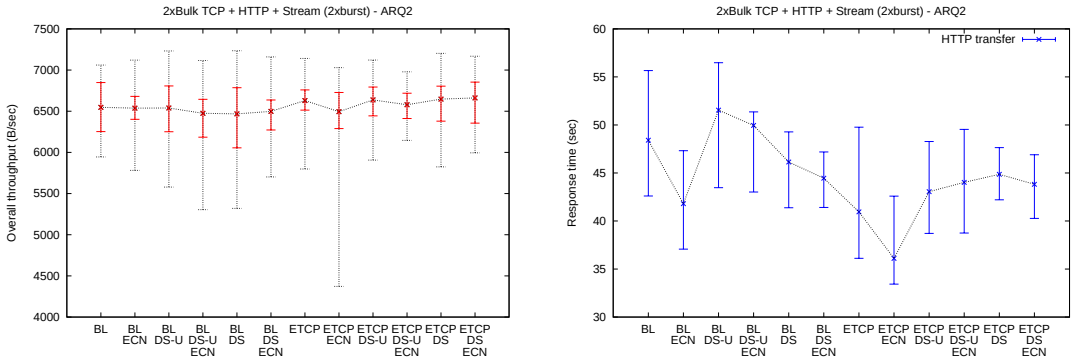
In Figure 40c is the streaming throughput. The throughput is very close to the values obtained with the workload of a competing streaming flow. This is especially true with prioritization, which makes the streaming flow almost completely immune

to the changes in the type of the TCP flow. Compared with the workload of a competing streaming flow, the additional competing traffic does not affect much to the streaming throughput in the cases without prioritization. The cases with prioritization are not affected at all.

In Figure 40d are the elapsed times of the faster and slower flow. With ECN the elapsed times are longer that without it because of the lower overall throughput. In Figure 40e is the number of spurious RTOs and unnecessary retransmissions for the bulk TCP flows and for the HTTP transfer. The spurious RTOs are common because of the link-level retransmissions. The numbers are larger than with the error-free link in all configurations except in the Baseline TCP case. With the error-free link the Baseline TCP case has a long queue, which causes multiple spurious RTOs for the HTTP transfer, but now the queue overflow forces the flows to reduce their congestion windows, which reduces the length of the queue.

In Figure 40f are the drops for each flow. When we compare them with the error-free link cases, the Enhanced TCP cases have more drops for the bulk TCP flows because of the link-level retransmissions during the bad state that lengthen the last-hop router queue which then overflows more often. With Baseline TCP the numbers are slightly smaller because error drops limit the severity of the slow-start overshoot in some cases.

5.5.3 Links with Medium and Low ARQ Persistency



(a) Overall throughput

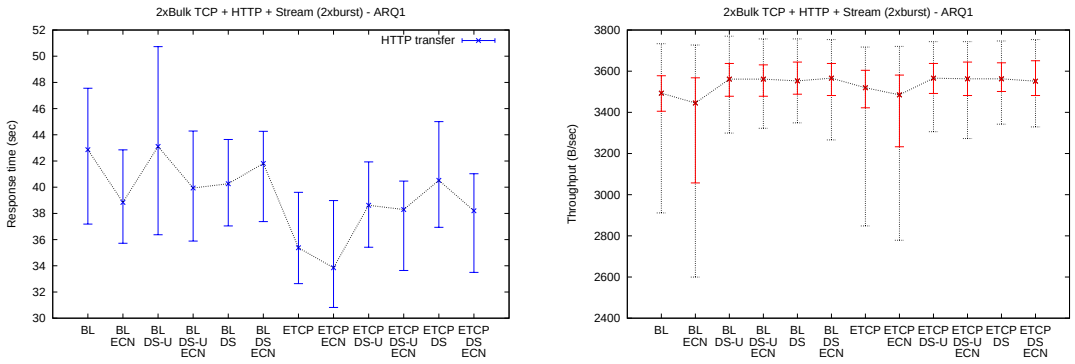
(b) HTTP Response time

Figure 41: Summary of results over the link with medium ARQ persistency in the workload of competing HTTP transfer and streaming flow (40 repetitions)

The overall throughput over the link with medium ARQ persistency is shown in Figure 41a. The overall throughput is affected by the larger initial window of Enhanced TCP and by the depletion of the link send buffer after an ECN mark in the cases with ECN.

In Figure 41b are the HTTP response times. The bare Baseline TCP and Enhanced TCP cases still benefit from ECN because congestion happens in the shared queue.

The superior performance of the ECN case compared with prioritization for HTTP and streaming happens for similar reasons as on the link with high ARQ persistency. With prioritization for HTTP and streaming, the bulk TCP flows are not ECN marked because the queue average does not rise enough between the error drops, only the repetitions with very unfair response time are shortened.



(a) HTTP Response time

(b) Streaming throughput

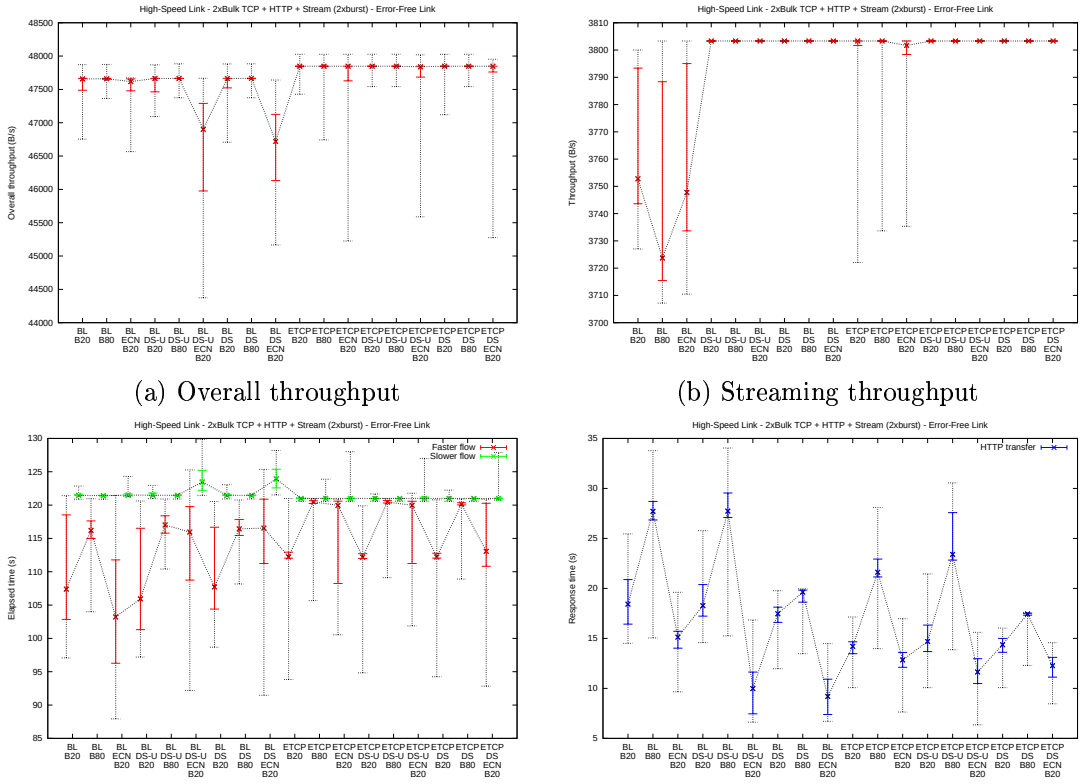
Figure 42: Summary of results over the link with low ARQ persistency in the workload of competing HTTP transfer and streaming flow (40 repetitions)

In Figure 42a are the HTTP response times over the link with low ARQ persistency. In the case of Baseline TCP with ECN, the response time is shorter than in the Baseline TCP cases because of the ECN marks that reduce the queue length. In the cases of Enhanced TCP with prioritization, the response time is longer than without DiffServ because of bulk TCP class's queuing space, which increases the number of the bulk TCP packets transferred during the HTTP transfer.

The streaming throughput is presented in Figure 42b. The values without prioritization are slightly lower than in the workload of a competing streaming flow.

5.5.4 High-Speed Error-Free Link

With the high-speed error-free link, we test with two last-hop router queue sizes: a smaller size is 20 packets (B20) and a larger size is 80 packets (B80). ECN did not work with the larger queue size because non-ECN-capable TCP retransmissions in the overshoot recovery were all dropped as with the workload of a streaming flow. Each repetition contains one oversleep that is up to 20 ms long. This oversleep occurs because the log size configuration of Seawind process was not reflected to the larger amount of data of this test case, which causes a single five megabyte memory copy during the repetition. The effect is minor because affect packets are very few. Since the starting times of the competing traffic vary, the oversleep does not systematically occur at the same place. It is known based on the preliminary tests that such oversleeps are insignificant for TCP. The affected streaming packets cannot affect indices.



(a) Overall throughput

(b) Streaming throughput

(c) Elapsed time of the bulk TCPs

(d) HTTP response time

Figure 43: Summary of results over the error-free high-speed link in the workload of competing HTTP transfer and streaming flow (40 repetitions)

The overall throughput is shown in Figure 43a. Baseline TCP has slightly lower (47657.70 B/s) median throughput than Enhanced TCP (47848.35 B/sec) as with the smaller link speed because of larger initial window. The depletions of the link send buffer cause the lower medians in two Baseline TCP cases with ECN. The depletions of the link send buffer that occur in cases of Enhanced TCP with ECN are less severe.

The streaming throughput is shown in Figure 43b. The results follow what is observed with the workload of a competing streaming flow over the high-speed link. Baseline TCP is affected by the slow-start overshoot. With Enhanced TCP no overshoot occur. With prioritization the streaming flow does no experience any drops.

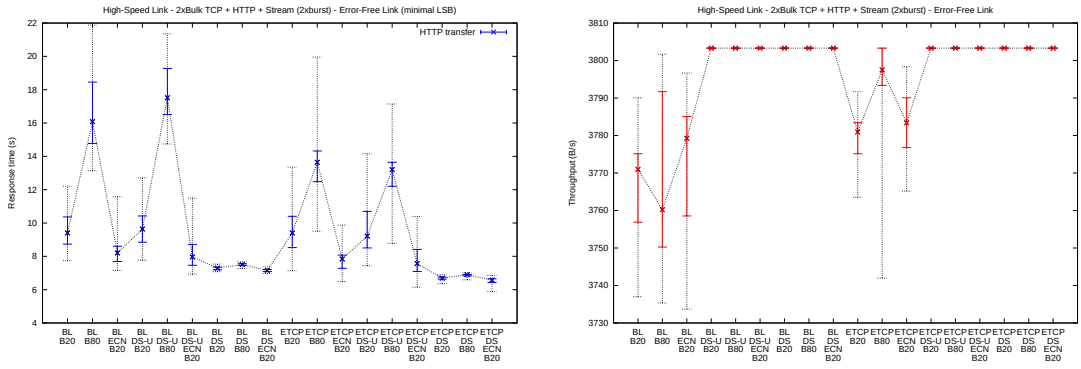
The elapsed times of the bulk TCP flows are shown in Figure 43c. When Baseline TCP without prioritization is used, congestion that occurs after the slow-start overshoot causes only few drops. These drops can affect just one flow, which increases unfairness between the elapsed times of the bulk TCP flows. For the same reason, the stability of the elapsed time of the faster flow is not good. With ECN the probability of the second instance of congestion is smaller but it still occurs occasionally. Besides the slow-start overshoot, no other congestion occurs with the larger queue size. Because the streaming bit-rate is small compared with the total bandwidth

of the link, with prioritization the elapsed times of the bulk TCP flows are almost equal to the no prioritization cases except when the depletion of the link send buffer occurs.

With Enhanced TCP there is no slow-start overshoot. Therefore packets are dropped only rarely with the smaller queue size without ECN. When the drops occur, the touch only the slower flow, which is the only bulk TCP flow that slows down. Because the bulk TCP transfer progress very similarly in the individual repetitions, the drops occur nearly at the same time in each repetition regardless of prioritization. Thus the stability of the elapsed time of the faster flow is good, but the fairness between the bulk TCP flows is not good. With ECN the drops are avoided, but unfair ECN marks occur and spread the quartiles. With the larger queue size, no congestion occurs. Therefore fairness between the bulk TCP flows looks too promising to last if the length of the bulk TCP flows would be increased.

The response time of the HTTP transfer is shown in Figure 43d. Most of the HTTP packet drops occur during the slow-start overshoot. The later congestion is not severe, and therefore it does not always cause HTTP packet drops. Even when they occur, the other objects cover the delay in the response time. With ECN the queuing delay is shorter, and therefore the response time is shorter. With the smaller queue size, the inadequate queuing space is again visible as in the workload of a competing streaming flow. Therefore prioritization for HTTP does not shorten the response time almost at all. With the larger queue size, however, 29% shorter response time is achieved with prioritization than without it. The depletions of the link send buffer reduce the response time in the corresponding cases as the queuing delay during the depletion is very low. No spurious RTO occur for the HTTP transfer.

Minimal Link Send Buffer



(a) HTTP response time

(b) Streaming throughput

Figure 44: Summary of results over the error-free high-speed link with minimal link send buffer in the workload of competing HTTP transfer and streaming flow (40 repetitions)

The overall throughput is similar to the workload of a competing streaming flow over the high-speed link. As with it, the depletion of the link send buffer causes suboptimal performance with the smaller queue size. Good overall throughput is achieved only with the larger queue size. The Enhanced TCP cases have a slightly higher overall throughput than the Baseline TCP cases because of the larger initial window. Similarly, the elapsed times of the bulk TCP flows are inversely proportional to the overall throughput. They resemble the results that were obtained in the workload of a competing streaming flow.

The HTTP response times are shown in Figure 44a. Compared with the larger link send buffer results, also the response time seems to benefit from the minimal link send buffer. With larger queue size, a drastic more than 50% cut is achieved in the median, and the stability of the response time is very good. As with larger link send buffer, the inadequate buffering is visible with the smaller queue size.

The streaming throughput is shown in Figure 44b. With prioritization for streaming, the performance is stable as with the workload of a competing streaming flow. The cases without prioritization suffer from a small number of congestion drops. With the larger queue size, the slow-start overshoot of the bulk TCP flows is more severe and therefore the throughput of the Baseline TCP case is lower than in the other cases. With CBI in Enhanced TCP the overshoot is avoided, but as with the workload of a competing streaming flow, the traffic congests the link once because of the smaller total buffering capacity, which causes streaming drops. The end-to-end delay and IPDV are almost the same as with the competing streaming flow over the high-speed link.

5.5.5 Summary of Results

The performance of the streaming flow is equal to that of the streaming flow with two bulk TCP flows. Because of shared queue, the HTTP response time is better in ECN cases without prioritization than expected from the results of the workload of competing HTTP transfer. With 40 repetitions only, the elapsed time of the faster flow does not stabilize well, but it seems that with the error-free link the fairness between the bulk TCP flows is best with prioritization for HTTP and streaming when ECN is not employed. With high-speed link, the larger buffer size and ECN, completing the test runs was impossible because of a high RED queue average.

The HTTP response time varies because of queuing and congestion. Prioritization for streaming only increases an already poor median response time above one minute. With Enhanced TCP the HTTP transfer adds to the congestion because of the larger initial window. When ECN is employed without prioritization, the response time is unexpectedly good compared with the case of prioritization for HTTP because ECN marks reduce the congestion windows of the bulk TCP flows more than with prioritization for HTTP that protects the bulk TCP flows from the HTTP segments as well. Similarly, the response time in the case of Baseline TCP with ECN and prioritization for streaming over the error-free link is equal to the case of Baseline

TCP with ECN. However, that level cannot be maintained when error drops occur on the link. With higher link speed and smaller queue size, the inadequate buffering obfuscates the response times.

5.6 Discussion of Results

Regardless of workload, the performance of the Baseline TCP with the tail-drop dropping policy is not good, as expected, mainly because of congestion. A competing transfer that starts during the slow-start overshoot or while the other bulk TCP flows are in the congestion avoidance is often hit by a congestion drop in the beginning, which penalizes the competing transfer unfairly because its congestion window becomes small compared with the other flows. In addition with the low link speed, the spurious RTOs for the initial window of a later starting TCP flow affect the progress of the flow in multiple repetitions. Especially with usage that is common in Web browsing, a spurious RTO occurs in almost every test case at least for a single HTTP object but often during an HTTP transaction more than one HTTP object is affected by a spurious RTO for the initial window. The spurious RTO triggered unnecessary retransmissions that a more advanced TCP sending logic could avoid but main problem with an initial window spurious RTO is the congestion window that is left very small, which slows an later starting transfer heavily relative to the bulk TCP flows starting at time zero. Because of congestion and spurious RTOs, it is clear that we need to find an enhancement for better performance.

The slow-start overshoot is avoidable with CBI that was included to the Enhanced TCP. The initial slow-start threshold that is passed to a new flow is small enough to cause transition to the congestion avoidance before the overshoot begins. The later starting transfers that were suffering in all workloads because of the slow-start overshoot have much better performance when CBI is enabled. In addition, when one of the zero-starting bulk TCP flows does not suffer from error drop, CBI prevents it from slow starting its congestion window to an unfairly large size as the transition to the congestion avoidance is performed whenever the slow-start threshold is crossed. The transition happens even when other flows suffer from the error drops and cannot yet congest the wireless link. Contrary to what was expected, the performance of TCP was not ruined by error drops which cause CBI to hand on a too small initial slow-start threshold to a following flow. Most likely the performance is maintained because two bulk TCP flows need quite small initial ssthresh to fully utilize the link. With the delay-bandwidth product of the lower link speed only five segments per flow are necessary to fully utilize to wireless link, and also with an initial ssthresh smaller than five segments, the five segments threshold is crossed after few round-trips. The initial ssthresh seems to be at least four segments in over 90% of the cases.

During the congestion avoidance the last-hop router queue remains long for considerable duration. We need RED or other active queue management algorithm to solve this problem. With RED the ECN successfully attacks this problem but introduces instability and occasionally occurring suboptimal phenomena. The RED algorithm

introduces some unfairness, however, this unfairness is less severe than what congestion causes for a later starting flow without RED. Besides, the unfairness in a statistical sense seems to be upper bounded. Since finding good RED parameters was not a task of this study, the results achieved with ECN could possibly be improved more. With careful RED parametrization many of the observed negative phenomena could be less severe or disappear altogether.

The initial window of four increased considerably the probability that a flow recovers quickly from an error burst or congestion occurring close to the beginning of the flow. Especially the cases that avoid RTO because of the larger initial window benefit much. With the error-free link, the effect is, of course, limited to congestion as no error drops occur. On the negative side is the possibility of more severe congestion if many flows are started close to each other and they overload the capacity of the bottleneck router considerably.

The results obtained with the workload of two bulk TCP flows competing with a short TCP flow and with the workload of two bulk TCP flows competing with a bulk TCP flow agree with each other. The fairest performance is achieved by using Enhanced TCP with ECN. Many phenomena that occur in these cases are also visible in the other workloads, usually more frequently.

As regards the streaming flow, the workload of two bulk TCP flows competing with a streaming flow and the workload of two bulk TCP flows competing with a streaming flow and an HTTP transfer have similar characteristics. In both cases DiffServ prioritization is necessary to achieve a good throughput for the streaming flow, however, the observed performance does not meet the end-to-end delay and IPDV requirements of the real-time streaming. With prioritization the bit-rate of the streaming flow is protected from the interference of the other flows on a long time-scale. On a short time-scale, however, the IP-level quality of service does not meet the requirements of the real-time streaming because of buffering required at the link level and because of the transmissions delays of the intervening packets that cause jitter. Other means are required to improve the end-to-end delay and IPDV. Some buffering at the base station is often necessary because the wireless link might use specific techniques, e.g., resource allocation or interleaving, which require certain amount of data to be buffered before the data is actually transmitted over the wireless channel. We observed that with such a buffer below the IP-level, the median end-to-end delay of the streaming flow is increased by the queuing delay due to the link-level buffering, easily preventing the use of the real-time streaming. The transmission delays can be reduced by using a smaller MTU, a smaller frame size on the link-level, or preemption. The smaller MTU and frame size increases overhead. The preemption could, for example, stop sending the current lower priority frame and discard its remaining part when a higher priority frame becomes available, which reduces the effective throughput of the link because resources that were used to transmit the start of the lower priority frame are wasted.

As regards the HTTP transfer, prioritization for HTTP using DiffServ is very useful for the response time. When error drops do not occur, the bulk TCP flows hinder

the TCP transfer considerably without prioritization. A more controversial issue is the TCP variant. It seems that Enhanced TCP benefits most when the link errors are frequent, but on the error-free link the large initial window backfires by generating more troubles than it solves. However, it could be possible to eliminate those problems but still preserve the benefits. The congestion caused by the large initial window could be avoided with a larger HTTP-class queue that could absorb bursts better. Another issue are the spurious RTOs. As discussed earlier, the HTTP WLG plugin of Seawind starts many inline objects simultaneously. The resulting burst builds up a long queue on the last-hop router that is the bottleneck. The packets that arrive at the tail of the queue experience long queuing delay, which is long enough to lead to spurious RTOs. Arguably, starting of the HTTP inline objects could be more sophisticated. As we observed, the segments of the inline objects tend to increase burstiness because they arrive roughly at the same time to the bottleneck router. The sender could try to compensate this piling by pacing inline objects or packets over the whole round-trip. Then the end-to-end delay of an individual segment is shorter, and thus it less likely triggers a spurious RTO. A better pacing of the HTTP packets could also reduce the severity of the congestion problem or remove it altogether. The pacing of the TCP packets has been studied earlier [ASA00]. The study came up with an intuition that pacing performs well in some scenarios (including high delay-bandwidth product links) but has some drawbacks in number of cases. Therefore, pacing in general case can degrade performance. However, pacing of the HTTP inline objects could also be performed by the mobile host by pacing SYNs, instead of the fixed host that does not know when the access link has high delay-bandwidth product.

As the small HTTP objects spend most of their time in the slow start, ECN cannot equalize the throughputs because HTTP traffic has a different sending pattern compared with the bulk TCP flows. The bulk TCP flows have higher throughput during the HTTP transfer as the individual TCP flows carrying HTTP are not able to gain an equal share of the bandwidth before the transfer completes. Even though the larger initial window helps the HTTP flows to open congestion window faster in some repetitions, the other repetitions are penalized when spurious RTOs start to happen. Therefore prioritization for HTTP flows is necessary in reducing round-trips of the HTTP objects because with prioritization the HTTP flows can bypass the bulk TCP queue that can be long due to the full-queue problem [BCC+98]. Otherwise, the acceleration of the HTTP objects is constrained by the long round-trip time.

With the initial window of four the effect of the unnecessary retransmissions would be more severe if a TCP variant would not include F-RTO because all segments of the initial window are retransmitted after a spurious RTO of the first segment. Another frequent case that F-RTO could not resolve is the initial window of two segments as it is in Baseline TCP. If a spurious RTO happens before any segment from initial window is acknowledged by cumulative ACK or SACK information, F-RTO cannot distinguish a loss from spurious RTO because the second ACK will always either be a duplicate acknowledgement or an acknowledgement for all pending data,

that is, initial-window segments. Such acknowledgements indicate a potential loss for the F-RTO. If these segments are acknowledged with two individual ACKs, the SACK-enhanced F-RTO version [SK05] detects RTO as spurious, but if only a single ACK is received, also the SACK-enhanced version claims that the loss recovery was necessary, which causes TCP to skip the response algorithm. If response algorithm is aggressive to restore the pre-RTO congestion control state, this step cannot be performed, which can cause substantial decrease in performance. Thus we conclude that he initial window should be at least three segments for F-RTO to work reliably in a case of the initial-window spurious RTO.

The HTTP objects experience most of the spurious RTOs in the initial window. When a spurious RTO is detected by F-RTO, the congestion window is reduced as a precaution and the previous slow-start threshold value is not restored by the used F-RTO implementation. This reduction aborts the aggressive slow-start phase. A resulting small congestion window is vulnerable to drops and force the HTTP objects to do more round-trips to complete the transfer. A delay for an individual HTTP object cascade to the transfer of the next HTTP objects. It would require further study, to find out whether the response time of the HTTP transfer would be shorter if F-RTO would do restoring or would congestion just become more severe.

The HTTP response time pattern with the different configurations is similar regardless of the presence of the streaming flow, except with a shared queue RED that has a very good performance with the streaming flow. The response times are, of course, longer with the streaming flow as the wireless link transfers more non-HTTP packets. With prioritization for HTTP, the difference in the median HTTP response time between the workload with and without streaming flow is around eight seconds in all configurations. Without prioritization, especially when Enhanced TCP is used with ECN, the ECN works very well. Hence, the same difference in the response time of the Enhanced TCP with ECN case is less than one second. When Baseline TCP is used with ECN, similar thing happens, but the presence of the slow-start overshoot makes the difference inferior to the Enhanced TCP case. Over the links with errors, the benefits of the shared queue RED are more evident. Without prioritization the RED algorithm marks bulk TCP flows more than with prioritization for HTTP, which shortens the queuing delay of the HTTP transfer. The queue of the last-hop router becomes occasionally empty. Therefore the HTTP packets can go directly to the non-full link send buffer, which makes the response time even shorter than with DiffServ.

We observed that not only the streaming flow is improved with link-level quality of service but also the response time of the HTTP transfer can be improved drastically.

6 Conclusions and Future Work

This thesis covers GPRS/UMTS-like wireless environments. It discusses problems encountered, suggested solutions, and provides an experimental performance study in such environment. The problems that are encountered in wireless environment

include congestion, error bursts due to corruption or black-outs, long propagation delay, spurious RTOs because of latency spikes, and underutilization due to varying link bandwidth. Since some services cannot survive acceptably in such conditions, these problems must be addressed. In this study, we test the suitability of DiffServ, RED with ECN, and link-level retransmissions for providing quality of service, fairness, and robustness.

In this study we analyzed workloads with bulk data TCP transfers, streaming traffic, and HTTP transfer. We observed that Baseline TCP with tail-drop fails to deliver fairness and quality of service mainly because of congestion. RED with ECN helps later starting transfers by reducing queuing delay and by preventing congestion. Enhanced TCP includes three TCP enhancements: initial window of four, CBI, and F-RTO. Initial window of four helps short flows to start faster and reduces probability that RTO is necessary to trigger the loss recovery when the flow encounters drops in the initial window. CBI prevent the slow-start overshoot and balances congestion windows in case a error drop occurs only for a single flow. F-RTO prevents unnecessary retransmissions that are very common especially with the HTTP transfer. Both the streaming flow and the HTTP transfer benefit from quality of service provided by DiffServ, which reorders the arriving packets so that the effect of bulk TCP packets is less severe because the streaming and HTTP packets are prioritized among all the packets, which reduces their queuing delay and the number of congestion drops.

As regards the general questions raised in Section 4.1. To the first question, the experiments show that we can successfully apply the techniques intended for Internet-wide deployment also when a wireless link is present. The second question on whether the TCP enhancements improve performance and whether the performance gain can be maintained when advanced quality-of-service and congestion management techniques are used is more controversial. The improvements with TCP enhancements are clear for later starting transfers but performance may suffer when DiffServ prioritization is enabled compared with Baseline TCP and prioritization. To the third question on whether the HTTP response time can be improved the experiments show that both DiffServ and RED with ECN can be used to reduce the response time. The fourth question on whether the requirements of the real-time streaming can be met proves to be hard even with quality of service. With lower link-speed, it is clear that without quality of service support, the requirements are not met. Also with quality of service support, the improvements to the end-to-end delay of the stream are limited by the buffering required by the wireless link, and stream IPDV is affected by the long queuing delay. The long queuing delay occurs especially during link-level retransmissions but the transmission delay of the full-sized TCP segment is also relatively long compared with the sending interval of the streaming flow. Again with the higher link speed and adequate buffering, the buffering prevents streaming flow from achieving a low end-to-end delay. However, the high end of IPDV is considerably shorter than with the lower link speed because the faster link transmits a packet in shorter time, but the maximum end-to-end delay still exceed the 20 ms sending interval almost threefold also with quality of

service. The fifth question concerns the effect of the quality-of-service and congestion management mechanisms for the bulk TCPs. With DiffServ the elapsed time of the bulk TCP flows are slightly more stable than without it. ECN introduces some unfairness but the introduced unfairness seems to be under control of RED in statistical sense. The effect of the congestion control phase of the background load, which is covered in the sixth question, is very significant because an occurrence of congestion is almost directly related to it. Thus, a later starting transfer encounters different conditions depending on the starting time relative to the background load. However, we can reduce significance of the starting time with CBI, RED with ECN, or DiffServ.

In the future our intention is to add the quality of service support also below the IP level, which can be used to provide quality of service that is necessary for the real-time streaming also when link-level ARQ for TCP traffic is present. With such a configuration the performance should resemble the results that were obtained with the minimal link send buffer work-around. Based on this study, we expect end-to-end delay and jitter that are low enough for real-time streaming, and also a drastic improvement in the HTTP response time.

The parametrization of RED can possibly be improved. The configuration for this study was adjusted using a limited amount of testing. Thus, the configuration in use caused unnecessary ECN marks and often did not respond before drops had already occurred. Finding out a better combination of RED parameters in an environment with a GPRS/UMTS-type wireless link remains a topic for further research.

References

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

ADG+00 M. Allman, S. Dawkins, D. Glover, J. Griner, D. Tran, T. Henderson, J. Heidemann, J. Touch, H. Kruse, S. Ostermann, K. Scott and J. Semke, Ongoing TCP Research Related to Satellites. RFC 2760, Internet Society, February 2000.

AFP02 M. Allman, S. Floyd and C. Partridge, Increasing TCP's Initial Window. RFC 3390, Internet Society, October 2002.

All98 M. Allman, On the Generation and Use of TCP Acknowledgements. *ACM SIGCOMM Computer Communication Review*, 28,5(1998), pages 4–21.

All00 M. Allman, A Web Server's View of the Transport Layer. *ACM SIGCOMM Computer Communication Review*, 30,5(2000), pages 10–20.

APS99 M. Allman, V. Paxson and W. Stevens, TCP Congestion Control. RFC 2581, Internet Society, April 1999.

ASA00 A. Aggarwal, S. Savage and T. Anderson, Understanding the Performance of TCP Pacing. *Proceedings of the Conference on Computer Communications (INFOCOM 2000)*, volume 3, Tel Aviv, Israel, March 2000, pages 1157–1165.

BAFW03 E. Blanton, M. Allman, K. Fall and L. Wang, A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP. RFC 3517, Internet Society, April 2003.

Bak95 F. Baker, Requirements for IP Version 4 Routers. RFC 1812, Internet Society, June 1995.

BBC+98 S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang and W. Weiss, An Architecture for Differentiated Service. RFC 2475, Internet Society, December 1998.

BBCF01 D. Black, S. Brim, B. Carpenter and F. L. Faucheur, Per Hop Behavior Identification Codes. RFC 3140, Internet Society, June 2001.

BCC+98 B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski and L. Zhang, Recommendations on Queue Management and Congestion Avoidance in the Internet. RFC 2309, Internet Society, April 1998.

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

BKG+01 J. Border, M. Kojo, J. Griner, G. Montenegro and Z. Shelby, Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations. RFC 3135, Internet Society, June 2001.

BLFF96 T. Berners-Lee, R. Fielding and H. Frystyk, Hypertext Transfer Protocol – HTTP/1.0. RFC 1945, Internet Society, May 1996.

BPSK96 H. Balakrishnan, V. Padmanabhan, S. Seshan and R. Katz, A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (ACM SIGCOMM 1996)*, Palo Alto, California, United States, August 1996, pages 256–269.

BW97 G. Brasche and B. Walke, Concepts, Services, and Protocols of the New GSM Phase 2+ General Packet Radio Service. *IEEE Communications Magazine*, 35,8(1997), pages 94–104.

BZ96 J. Bennett and H. Zhang, WF^{2}Q: Worst-Case Fair Weighted Fair Queueing. *Proceedings of the Conference on Computer Communications (INFOCOM 1996)*, San Francisco, California, United States, March 1996, pages 120–128.

BZ97 J. Bennett and H. Zhang, Hierarchical Packet Fair Queueing Algorithms. *IEEE/ACM Transactions on Networking*, 5,5(1997), pages 675–689.

Cisco Cisco Systems, *Distributed Weighted Random Early Detection.* Technical Specification. `http://www.cisco.com/univercd/cc/td/doc/product/software/ios111/cc111/wred.pdf`. [15.11.2005]

Cla82 D. Clark, Window and Acknowledgement Strategy in TCP. RFC 813, Internet Society, July 1982.

DC02 C. Demichelis and P. Chimento, IP Packet Delay Variation Metric for IP Performance Metrics (IPPM). RFC 3393, Internet Society, November 2002.

DCB$^+$02 B. Davie, A. Charny, J. Bennet, K. Benson, J. L. Boudec, W. Courtney, S. Davari, V. Firoiu and D. Stiliadis, An Expedited Forwarding PHB (Per-Hop Behavior). RFC 3246, Internet Society, March 2002.

Dev02 M. Devera, Hierachical token bucket theory, May 2002. `http://luxik.cdi.cz/~devik/qos/htb/manual/theory.htm`. [15.11.2005]

DKS89 A. Demers, S. Keshav and S. Shenker, Analysis and Simulation of a Fair Queueing Algorithm. *Proceedings of the Symposium on Communications Architectures & Protocols (ACM SIGCOMM 1989)*, Austin, Texas, United States, August 1989, pages 1–12.

DMK$^+$01 S. Dawkins, G. Montenegro, M. Kojo, V. Magret and N. Vaidya, End-to-end Performance Implications of Links with Errors. RFC 3155, Internet Society, August 2001.

DMKM01 S. Dawkins, G. Montenegro, M. Kojo and V. Magret, End-to-end Performance Implications of Slow Links. RFC 3150, Internet Society, July 2001.

ETS05 ETSI, *Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); General Packet Radio Service (GPRS); Service description; Stage 2.* TS 23.060, version 6.10.0. September 2005.

FF99 S. Floyd and K. Fall, Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, 7,4(1999), pages 458–472.

FH99 S. Floyd and T. Henderson, The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 2582, Internet Society, April 1999.

FJ93 S. Floyd and V. Jacobson, Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1,4(1993), pages 397–413.

FJ95 S. Floyd and V. Jacobson, Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3,4(1995), pages 365–386.

FKSS97 W. Feng, D. Kandlur, D. Saha and K. Shin, Techniques for Eliminating Packet Loss in Congested TCP/IP Networks. Technical Report CSE-TR-349-97, U. Michigan, November 1997.

Flo94 S. Floyd, TCP and Explicit Congestion Notification. *ACM SIGCOMM Computer Communication Review*, 24,5(1994), pages 10–23.

Flo97 S. Floyd, *RED: Discussions of Setting Parameters*, November 1997. `http://www.icir.org/floyd/REDparameters.txt`. [15.11.2005]

FW02 G. Fairhurst and L. Wood, Advice to link designers on link Automatic Repeat reQuest (ARQ). RFC 3366, Internet Society, August 2002.

HBWW99 J. Heinanen, F. Baker, W. Weiss and J. Wroclawski, Assured Forwarding PHB Group. RFC 2597, Internet Society, June 1999.

HGM+04 B. Hubert, T. Graf, G. Maxwell, van R. Mook, van M. Oosterhout, P. Schroeder, J. Spaans and P. Larroy, *Linux Advanced Routing & Traffic Control HOWTO*, March 2004. `http://lartc.org/lartc.html`. [24.11.2005]

Hoe95 J. Hoe, Startup Dynamics of TCP's Congestion Control and Avoidance Schemes. Master's thesis, MIT, June 1995.

HPF00 M. Handley, J. Padhye and S. Floyd, TCP Congestion Window Validation. RFC 2861, Internet Society, June 2000.

Hus00 G. Huston, *Internet Performance Survival Guide: QoS Strategies for Multiservice Networks*. Wiley, 2000.

IIPa IIP Mixture Project, Patches to Linux kernels. `http://www.cs.helsinki.fi/research/iwtcp/kernel-patch/`. [06.10.2005]

IIPb IIP Mixture Project, Seawind home page. `http://www.cs.helsinki.fi/research/iwtcp/seawind/`. [24.11.2005]

IML+03 H. Inamura, G. Montenegro, R. Ludwig, A. Gurtov and F. Khafizov, TCP over Second (2.5G) and Third (3G) Generation Wireless Networks. RFC 3481, Internet Society, February 2003.

ISI05 ISI at University of South Carolina, The Network Simulator – ns-2. `http://www.isi.edu/nsnam/ns`. [19.09.2005]

ITU90 ITU, *40, 32, 24, 16 kbit/s Adaptive Differential Pulse Code Modulation (ADPCM)*. Recommendation G.726. December 1990.

Jac88 V. Jacobson, Congestion Avoidance and Control. *ACM SIGCOMM Computer Communication Review*, 18,4(1988), pages 314–329.

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

Jär06 I. Järvinen, Analysis of Heterogeneous IP Traffic in Wireless Environment: Full Statistics of the Results, Master's Thesis Supplementary, University of Helsinki, Department of Computer Science, February 2006. `http://www.cs.helsinki.fi/research/iwtcp/papers/`

KAL+05 H. Kaaranen, A. Ahtiainen, L. Laitinen, S. Naghian and V. Niemi, *UMTS Networks – Architecture, Mobility and Services*. Wiley, 2005.

KBF+04 P. Karn, C. Bormann, G. Fairhurst, D. Grossman, R. Ludwig, J. Mahdavi, G. Montenegro, J. Touch and L. Wood, Advice for Internet Subnetwork Designers. RFC 3819, Internet Society, July 2004.

Kes97 S. Keshav, *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network*. Addison-Wesley, 1997.

KGM+01 M. Kojo, A. Gurtov, J. Manner, P. Sarolahti, T. Alanko and K. Raatikainen, Seawind: a Wireless Network Emulator. *Proceedings of 11th GI/ITG Conference on Measurement, Modelling and Analysis (MMB 2001)*, Aachen, Germany, September 2001.

Kul03 T. Kulve, Analysis of Concurrent TCP and Streaming Traffic Over a Wireless Link. Master's thesis, University of Helsinki, Department of Computer Science, Series of Publications C, No. C-2003-50, October 2003.

LG05 R. Ludwig and A. Gurtov, The Eifel Response Algorithm for TCP. RFC 4015, Internet Society, February 2005.

JTG05 J. Manner, Jugi's Traffic Generator (JTG), November 2005. `http://www.cs.helsinki.fi/u/jmanner/software/jtg/`. [15.11.2005]

MDK+00 G. Montenegro, S. Dawkins, M. Kojo, V. Magret and N. Vaidya, Long Thin Networks. RFC 2757, Internet Society, January 2000.

Mil92 D. Mills, Network Time Protocol (Version 3) Specification, Implementation and Analysis. RFC 1305, Internet Society, March 1992.

MMFR96 M. Mathis, J. Mahdavi, S. Floyd and A. Romanow, TCP Selective Acknowledgement Options. RFC 2018, Internet Society, October 1996.

MSM99 M. Mathis, J. Semke and J. Mahdavi, The Rate-Halving Algorithm for TCP Congestion Control, August 1999. `http://ietfreport.isoc.org/idref/draft-mathis-tcp-ratehalving/`. [18.10.2005]

MSML99 M. Mathis, J. Semke, J. Mahdavi and K. Lahey, The Rate-Halving Algorithm for TCP Congestion Control, June 1999. `http://www.psc.edu/networking/ftp/papers/draft-ratehalving.txt`. [17.10.2005]

Nag84 J. Nagle, Congestion control in IP/TCP internetworks. RFC 896, Internet Society, January 1984.

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

OLW99 T. Ott, T. Lakshman and L. Wong, SRED: Stabilized RED. *Proceedings of the Conference on Computer Communications (INFOCOM 1999)*, volume 3, New York, United States, 1999, pages 1346–1355.

PA00 V. Paxson and M. Allman, Computing TCP's Retransmission Timer. RFC 2988, Internet Society, November 2000.

Pos80 J. Postel, User Datagram Protocol. RFC 768, Internet Society, August 1980.

Pos81a J. Postel, Internet Protocol. RFC 791, Internet Society, September 1981.

Pos81b J. Postel, Transmission Control Protocol. RFC 793, Internet Society, September 1981.

PR83 J. Postel and J. Reynolds, Telnet Protocol Specification. RFC 854, Internet Society, May 1983.

PR85 J. Postel and J. Reynolds, File Transfer Protocol. RFC 959, Internet Society, October 1985.

RBL99 V. Rosolen, O. Bonaventure and G. Leduc, A RED discard strategy for ATM networks and its performance evaluation with TCP/IP traffic. *ACM SIGCOMM Computer Communication Review*, 29,3(1999).

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

SCFJ03 H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, RTP: A Transport Protocol for Real-Time Applications. RFC 3550, Internet Society, July 2003.

Shn98 B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, 1998.

SK05 P. Sarolahti and M. Kojo, Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP and the Stream Control Transmission Protocol (SCTP). RFC 4138, Internet Society, August 2005.

SKR03 P. Sarolahti, M. Kojo and K. Raatikainen, F-RTO: An Enhanced Recovery Algorithm for TCP Retransmission Timeouts. *ACM SIGCOMM Computer Communication Review*, 33,2(2003), pages 51–63.

SRL98 H. Schulzrinne, A. Rao and R. Lanphier, Real Time Streaming Protocol (RTSP). RFC 2326, Internet Society, April 1998.

SV95 M. Shreedhar and G. Varghese, Efficient Fair Queueing Using Deficit Round Robin. *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (ACM SIGCOMM 1995)*, Cambridge, Massachusetts, United States, September 1995, pages 231–242.

TDUMP TCPDUMP public repository. `http://www.tcpdump.org/`. [21.11.2005]

Tou97 J. Touch, TCP Control Block Interdependence. RFC 2140, Internet Society, April 1997.

WHZ+01 D. Wu, Y. Hou, W. Zhu, Y. Zhang and J. Peha, Streaming Video over the Internet: Approaches and Directions. *IEEE Transactions on Circuits and Systems for Video Technology*, 11,1(2001), pages 1–20.

A Summary of the Workloads

Table 11 summarizes the measurement workloads. DiffServ class is TCP (T), HTTP (H), or UDP (U). All starting times are offset by the sum of the maximum state length of the good and bad state with the lower link speed, that is 19 seconds.

Mixture workload	Link bw (bps)	Used link types	Data sent per flow (B)	Delay of the flow (s)	DiffServ classes	Last-hop router queue size size (pkts)	RED+ ECN
2xBulk TCP + TCP (short)	64000	e0, r4, r2, r1	2x343040 21440	19 19 + 0-36	No DS	20	no
2xBulk TCP + Bulk TCP	64000	e0, r4, r2, r1	2x343040 343040	19 19 + 0-36	No DS	20	no
2xBulk TCP + HTTP	64000	e0, r4, r2, r1	2x343040 80400	19 19 + 0-36	1) No DS 2) T, H	20	no/ yes
2xBulk TCP + Stream	64000	e0, r4, r2, r1 e0 (LSB1)	2x343040 55056	19 19 + 0-36	1) No DS 2) T, U	20	no/ yes
2xBulk TCP + HTTP + Stream (2xburst)	64000	e0, r4, r2, r1	2x343040 80400 2 x 55056	19 19 + 0-36 19 + 0-36	1) No DS 2) T&H, U 3) T, H, U	20	no/ yes
2xBulk TCP + Stream	384000	e0, e0 (LSB1)	2x1029120 55056	19 19 + 0-13.5	1) No DS 2) T, U	1) 20 2) 80	no/ yes
2xBulk TCP + HTTP + Stream (2xburst)	384000	e0, e0 (LSB1)	2x2572800 80400 2 x 55056	19 19 + 0-13.5 19 + 0-13.5	1) No DS 2) T&H, U 3) T, H, U	1) 20 2) 80	no/ yes

Table 11: Workloads

B Configuration Parameters

In this section we list the used end-host kernel and Seawind configuration.

B.1 End-Host Kernel Configuration

In Table 12 is the configuration of the additional kernel parameters provided by the IIP patch [IIPa] and Table 13 lists the configuration of the standard kernel parameters. The boldfaced values depend on the configuration of the test case. The larger `tcp_rmem` and `tcp_wmem` where used with the higher link speed.

Option	Value
iip_iw	**2 / 4**
iip_quickacks	0
iip_delack	1
iip_limitedxmit	1
iip_ratehalving	0
iip_srwnd_max	16384
iip_srwnd_min	1024
iip_srwnd_size	16384
iip_srwnd_addr	0
iip_cbi_reuse_reorder	0
iip_cbi_reuse_ssthresh	**0 / 1**
iip_cbi_reuse_rtt	0
iip_cbi_reuse_iw	0

Table 12: IIP patch sysctls

Option	Value
tcp_low_latency	0
tcp_frto	**0 / 1**
tcp_tw_reuse	0
tcp_adv_win_scale	2
tcp_app_win	31
tcp_rmem	4096 **87380 174760** / 4096 **1398080 2796160**
tcp_wmem	4096 **32768 131072** / 4096 **4194304 8388608**
tcp_mem	97280 97792 98304
tcp_dsack	0
tcp_ecn	**0 / 1**
tcp_reordering	3
tcp_fack	0
tcp_orphan_retries	0
tcp_max_syn_backlog	1024
tcp_rfc1337	0
tcp_stdurg	0
tcp_abort_on_overflow	0
tcp_tw_recycle	0
tcp_fin_timeout	60
tcp_retries2	15
tcp_retries1	10
tcp_keepalive_intvl	75
tcp_keepalive_probes	9
tcp_keepalive_time	300
tcp_max_tw_buckets	180000
tcp_max_orphans	16384
tcp_synack_retries	5
tcp_syn_retries	50
tcp_retrans_collapse	1
tcp_sack	1
tcp_window_scaling	**0 / 1**
tcp_timestamps	0

Table 13: Kernel's standard TCP sysctls

B.2 Seawind Configuration

The parameters of Seawind process depend on the state. Common parameters are listed in Table 14, error-free link specific parameters in Table 15, good state specific parameters in Table 16, and bad state specific parameters in Table 17. The latter two relate only the links that have ARQ and not to the error-free link. Since the maximum number of ARQ retransmissions is one, two, or four, the `delay_drop_threshold` is 701 ms, 1401 ms, or 2801 ms respectively.

Option	Uplink Value	Downlink Value
link_receive_buffer_size	12000 bytes	12000 bytes
bgl_channel_allocation_algorithm	BASIC	BASIC
reordering	FALSE	FALSE
propagation_delay	300 ms	300 ms
rate_base	64000 bits/s	64000 bits/s
reassembly	FALSE	FALSE
user_max_rate	1	1
max_packet_size	2048 bytes	2048 bytes
link_send_buffer_size	12000 bytes	12000 bytes
queue_max_length	Disabled	1
queue_overflow_handling	Disabled	FLOW_CONTROL
queue_max_delay	Disabled	Disabled
bgl_output_handling	Disabled	Disabled
bgl_max_users_per_channel	Disabled	Disabled
link_overhead	Disabled	Disabled
bgl_user_priority	Disabled	Disabled
protocol_overhead_per_packet	Disabled	Disabled
available_rate	Disabled	Disabled
allocation_delay	Disabled	Disabled
rate_change_interval	Disabled	Disabled
fragment_size	Disabled	Disabled
random_delay_probability	Disabled	Disabled
fragmentation	Disabled	Disabled
queue_drop_policy	Disabled	Disabled
random_delay_length	Disabled	Disabled
link_idle_timeout	Disabled	Disabled
bgl_average_rate	Disabled	Disabled
queue_max_size	Disabled	Disabled
output_dropper	Disabled	Disabled
bgl_load_type	Disabled	Disabled
background_load	Disabled	Disabled

Table 14: Common Seawind process parameters

Option	Uplink Value	Downlink Value
link_layer_ack_emulation	FALSE	FALSE
error_delay_function	Disabled	Disabled
error_handling	Disabled	Disabled
error_rate_type	Disabled	Disabled
error_probability	Disabled	Disabled
delay_drop_threshold	Disabled	Disabled

Table 15: Additional Seawind process parameters for the error-free link

Option	Uplink Value	Downlink Value
link_layer_ack_emulation	TRUE	TRUE
error_delay_function	static, 700 ms	static, 700 ms
error_handling	DELAY_ITERATE	DELAY_ITERATE
error_rate_type	UNIT	UNIT
error_probability	static, 0.0	static, 0.0
delay_drop_threshold	701, 1401, 2801 ms	701, 1401, 2801 ms

Table 16: Additional Seawind process parameters for the good state of a link with ARQ

Option	Uplink Value	Downlink Value
link_layer_ack_emulation	TRUE	TRUE
error_delay_function	static, 700 ms	static, 700 ms
error_handling	DELAY_ITERATE	DELAY_ITERATE
error_rate_type	UNIT	UNIT
error_probability	static, 0.60	static, 0.60
delay_drop_threshold	701, 1401, 2801 ms	701, 1401, 2801 ms

Table 17: Additional Seawind process parameters for the bad state of a link with ARQ

C Description of Linux RED Algorithm Bug

During the preliminary tests a bug in the Linux RED algorithm was discovered. This bug causes too slow decrease of queue average when the idle period branch of the RED algorithm is used, that is, no packets are received for a short duration. The code ignores limitations of 32-bit multiplies which leads to an overflow. The offending line is in net/sched/sch_red.c:

```
us_idle = (q->qave * us_idle)>>q->Scell_log;
```

The final result after shifting fits always to `us_idle` but the intermediate result of the 32x32-bit multiplication very likely overflows with typical configuration. Neither of the types are defined so that they would have to be longer than 32-bits, `qave` is unsigned long and `us_idle` is long. RED stores the current queue average left shifted by the EWMA exponent to `qave` (i.e., EWMA is 9), and `us_idle` is the duration of the idle period in microseconds. Thus with one second idle period, `us_idle` is almost 2^{20}. To keep the result of 32x32-bit multiplication within 32-bits, the average queue length in bytes can be only very small:

$$2^{32} > (q_{len} \cdot 2^9) \cdot 2^{20} \Rightarrow q_{len} < 2^{(32-9-20)} = 2^3 = 8$$

So with a conservative idle period duration and recommended EWMA, the average queue length could be only 8 bytes long. With a longer idle period the bug easily occurs regardless of the queue length. To fix this bug, the result of the multiplication must be 64-bits long, which is then shifted back to 32-bits by `Scell_log`, and after that, the high 32-bits can be safely discarded. Casting either multiplicant to long long is enough to fix the bug. In the time of writing, the bug is still present in the latest kernel (2.6.15.3).