# Analysis of Concurrent TCP and Streaming Traffic Over a Wireless Link

Tuomas Kulve

Master's Thesis
Helsinki, 22nd September 2003

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

| | |
|---|---|
| Tiedekunta/Osasto — Fakultet/Sektion — Faculty<br>Faculty of Science | Laitos — Institution — Department<br>Department of Computer Science |

Tekijä — Författare — Author
Tuomas Kulve

Työn nimi — Arbetets titel — Title
Analysis of Concurrent TCP and Streaming Traffic Over a Wireless Link

Oppiaine — Läroämne — Subject
Computer Science

| | | |
|---|---|---|
| Työn laji — Arbetets art — Level<br>M.Sc. Thesis | Aika — Datum — Month and year<br>October 2003 | Sivumäärä — Sidoantal — Number of pages<br>73 p. + Appx. |

Tiivistelmä — Referat — Abstract

In this thesis we analyze the behavior of concurrent TCP connections and UDP flows over an emulated wireless link with real end hosts. For the emulation of the link and a last-hop router a real-time software emulator is used. With this software we are able to model the different link and network element characteristics. We run series of experiments and analyze the behavior of different TCP enhancements – *Duplicate Selective Acknowledgement* (D-SACK), *Eifel*, *Forward RTO-Recovery* (F-RTO), *Control Block Interdependence* (CBI), *Random Early Drop* (RED) and *Explicit Congestion Notification* (ECN) – and compare them to each other. We describe the main problems, which are the unfair use of the bandwidth and the additional delays caused by the wireless link. We present a detailed analysis of the results. The CBI option yielded good results in general and RED algorithm improved the fairness among the TCP connections. The Eifel and the F-RTO enhancements worked well with the presence of long delays.

Computing Reviews Classification:
C.2.2 (Network protocols)
C.4 (Performance of Systems)

Avainsanat — Nyckelord — Keywords
Wireless communication, mobile computing, performance, TCP, streaming

Säilytyspaikka — Förvaringsställe — Where deposited
Library of the Dept. of Computer Science, Report C–2003–

Muita tietoja — Övriga uppgifter — Additional information

# Contents

# 1 Introduction

Nowadays laptops and different PDA-devices with wireless network interfaces are joining the Internet. Mobile devices can access the Internet using for example satellite link, wireless LAN or cellular network, which all have different characteristics. In this thesis we concentrate on Internet transport protocol behavior over *wireless wide area networks* (W-WANs). We try to roughly mimic characteristics of 2.5G and 3G cellular network type links like *General Packet Radio Service* (GPRS) [BW98] and *Universal Mobile Telecommunications System* (UMTS). GPRS is an extension to *Global System for Mobile Communications* (GSM) [MM92]. It uses packet based communication while GSM is based on circuit switched connections. UMTS is a third generation communication (3G) system, which is supposed to offer fast wireless connections. 3G systems offer both packet data flows and circuit switched connections to meet the growing need of different types of quality requirements for interactive and streaming multimedia transfer applications.

The most widely used transport protocol in the Internet, *Transmission Control Protocol* (TCP) [Pos81], was designed for fixed networks and is known to have suboptimal performance on wireless networks [BPS+96, KRL+97]. The low performance is mainly due to high packet error rate or packets delayed and wrongly interpreted as lost. TCP slows down the sending rate when a packet loss occurs because it is normally interpreted as a sign of congestion in the network. In wireless networks delays are long and unpredictable and error rate is high. Therefore a missing packet does not always indicate congestion. In traditional fixed networks delays are small and quite steady and error rate is close to zero.

Another type of traffic has become more common as the bandwidth of the Internet has grown. Video conferencing or phone calls over IP networks use *streaming traffic* in which the data is sent as continuous data stream between the participants as long as the session lasts.

There are different approaches with number of proposals to enhance the performance of TCP in wireless environments [BPS+96]. Some of them propose changes to link layer and some are enhancements to TCP protocol itself. In this thesis we will analyze the influence of concurrent TCP connections and a streaming traffic flow to each other over a wireless link. We selected certain TCP enhancements that we expect to improve TCP performance over wireless networks and study their effect on TCP performance.

The performance analysis is done by measuring data flows originating from a fixed network and going through a last-hop router and a W-WAN link to a mobile host. The last-hop router is the last router in the fixed network before entering the W-WAN environment. The W-WAN link and a last-hop router are emulated but the end hosts are real computers with their own operating system and TCP/IP stack. We use a real-time emulator software called Seawind Network Emulator [KGM+01] for the emulation. The analysis will focus on TCP performance over a lossy link with GPRS-like bandwidth. With Seawind software we limit the bandwidth of the link, drop packets using different kind of distributions for the error rate, and emulate link layer retransmissions. With Seawind we can also use an active queue management algorithm in the emulated last-hop router in addition to a traditional tail-drop algorithm.

End host PCs in the analysis use the real TCP/IP stack implementation of the Linux operating system. Linux was chosen because its TCP/IP stack supports many of the new features that are standardized by the *Internet engineering task force* (IETF) and therefore supposed to spread to other implementations as well. Linux is also used in many servers throughout the world and it is freely available[1], which makes it ideal for research purposes.

The rest of the thesis is organized as follows. In section 2 we will introduce main characteristics of wireless environments. In section 3 we will discuss TCP behavior in wireless environment and what enhancements exist for improving TCP performance with various network characteristics. We also introduce shortly streaming data transfers. Test arrangements are described in section 4. We define what kind of tests we run and what TCP enhancements we measure. Results of the analysis are detailed in section 5 and finally a conclusion is given in section 6.

[1]Sources can be obtained from **http://www.kernel.org**

# 2 Wireless Environment

There are several different types of wireless environments. For communicating between countries or continentals there are satellite links with relatively high bandwidth and very high latency because of the great distance from a source point to a destination point through a satellite far in the orbit. For very short ranges, for example for networks inside buildings, there are wireless LANs which nowadays have bandwidth of over 10 Mbit/s and a latency of few milliseconds. The third type of the wireless environments is a W-WAN in which we will focus on in this study.

## 2.1 Wireless Wide Area Networks

Nowadays a typical way for a mobile client to connect to Internet is by using GSM or GRPS. These both belong to W-WAN category in which the range in one cell is roughly few tens of kilometers.

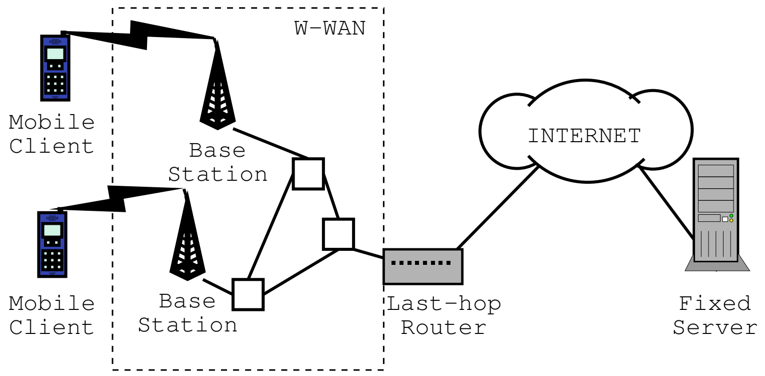

Figure 1: Wireless WAN environment

The main components between a fixed server host and a mobile host in a W-WAN environment are shown in Figure 1. A mobile host is usually used by an end user who accesses a mail or WWW -server in the Internet by using his phone or a laptop with GSM/GPRS capabilities. The W-WAN provides for the mobile client a logical access link to Internet via a last-hop router. The server that the mobile client communicates with is usually in a fixed wired network, which has much higher bandwidth and lower delays than the wireless link. Therefore the logical wireless link between a last-hop router and a mobile client is usually a severe bottleneck with the current bandwidths of W-WAN connections.

## 2.2 Characteristics of a GPRS Link

GPRS is a mobile packet radio communication system which makes use of the same radio architecture as GSM. GSM uses physical frequency channels, each divided into eight *time slots*. In GSM one user consumes one time slot and therefore there can be up to eight users in one channel. The bandwidth of a GPRS user can be dynamically modified by using multiple time slots simultaneously. This multislotting capability allows the GPRS network to give more bandwidth to a single GPRS user when there is bandwidth available and to level the bandwidth among the users when the overall requirement for bandwidth increases or new active GPRS users join the network.

Using the very popular GSM for data connections the available bandwidth is 9.6 kbit/s with basic GSM data service. It is sufficient only for reading email or for very basic WWW-browsing. With high speed circuit switched data (HSCSD), which is an enhancement to GSM data services, multiple data channels can be given to a single user and this way higher speeds are achievable. The downside of the HSCSD is that the data channels are reserved for one terminal even if they are not used all the time. In GPRS networks the data channels can be reserved on demand and the theoretical maximum bandwidth is 171.2 kbit/s, although the maximum bandwidth in practice is around 40kbit/s nowadays and is sufficient for WWW-surfing and very basic streaming. With third generation systems like UMTS the bandwidth grows up to hundreds of kilobits per second or even to megabits per second allowing streaming multimedia transfers such as video calls and conferences and network radios.

| Scheme | Data rate (kbit/s) |
|---|---|
| CS-1 | 9.05 |
| CS-2 | 13.4 |
| CS-3 | 15.6 |
| CS-4 | 21.4 |

Table 1: GPRS channel protection schemes and data rates [BW98]

GPRS data is transmitted over data channels and is protected by one of four possible channel protection schemes described in Table 1. These schemes can be chosen according to current channel quality. The CS-1 scheme provides the best forward error correction and the CS-4 provides the lowest forward error correction capability. By allocating multiple data channels for one terminal, data rates are multiplied. For example by using two CS-2 scheme channels, a data rate of 26.8 kbit/s can be achieved.

GPRS protocol layers are shown in Figure 2. The physical layer is responsible for forward error correction and detection. The radio link control / medium access control (RLC/MAC) layer arbitrates the access to the shared medium and performs the QoS control. The logical link control (LLC) provides a logical link for upper levels. The subnetwork-dependent convergence (SNDC) layer maps network-level protocol characteristics onto the underlying logical link control. Ciphering, segmentation and compression are done at this level.

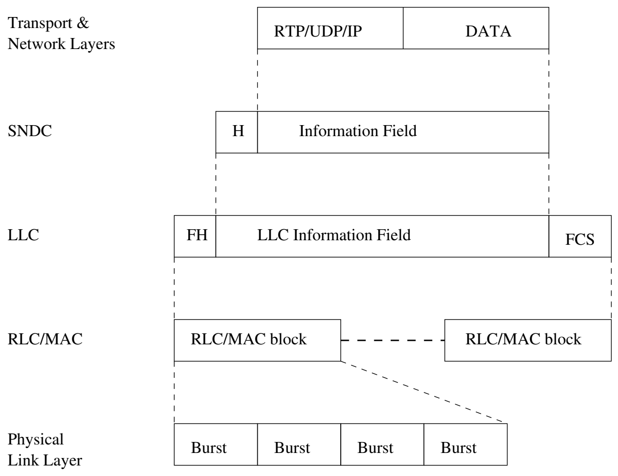

Figure 2: GPRS data flow

A GPRS network can efficiently hide packet losses from upper protocol layers by providing *automatic repeat requests* (ARQ) retransmissions. ARQ is a mechanism to retransmit a missing data segment and there are different ways to implement it like *Stop-And-Wait ARQ* and *Sliding-Window ARQ* [FW02]. ARQ retransmissions on link layer can increase the data throughput because link layer acknowledgements over a single link can be faster compared to for example TCP acknowledgements over longer end-to-end path and the sender detects the missing data sooner. Another reason is data frames used in link layer. They are often smaller than for example upper level TCP packets and retransmitting a single frame is faster than retransmitting all frames corresponding to a TCP segment.

The number of error related packet losses on wireless link detected by upper layers depends heavily on the ARQ persistency. ARQ persistency tells how many times or how long the missing frame is retransmitted. With low ARQ persistency the missing segment can be retransmitted for example once and with high persistency several times. If the ARQ persistency is very high and the link layer provides in-order delivery the upper layers can experience very long periods of time without receiving any data as the link layer is retransmitting a lost packet several times and queuing the rest of the incoming packets. This kind of phenomenon can happen when a mobile client is momentarily totally without a network coverage or the coverage is very poor.

# 3 Internet Traffic

In this section we discuss data transfers based on TCP and the behavior and problems of those over wireless links. Streaming data transfer is also discussed shortly. We explain what enhancements there are to improve TCP performance in general and especially with wireless links. The behavior of active queue management and quality of service algorithms are described.

## 3.1 Transmission Control Protocol

TCP [Pos81] was designed in early 80's for wired connections. While the Internet grew, congestion became a problem. Packet losses occurred mainly because of congestion when buffers of a congested router overflowed. Because the bit error rate is very low, packets corrupt rarely in a wired network. Therefore interpreting a packet loss as a sign of congestion was the basis for the congestion control algorithms [Jac88, APS99], *slow start* and *congestion avoidance*, which were designed to prevent congestion. To efficiently recover from single packet losses, two new algorithms, *fast retransmit* and *fast recovery*, were designed also.

*Initial window*, the initial size of the *congestion window* (`cwnd`), was originally one segment, but RFC 2581 [APS99] allows to use two segments. Although this part of the RFC is now updated [AFP02], many TCP implementations still use one or two segments. The `cwnd` tells how many packets can be transmitted without being acknowledged. In the beginning of a connection a TCP sender is in slow start phase and increases its `cwnd` by one segment for each new *acknowledgement* (ACK), thus sending two new segments[2] per ACK and increasing transmission rate rapidly. At some point the sender is sending data faster than a bottleneck router on the path can forward and therefore the router buffer becomes full. When a router buffer is full and a new packet arrives, it must drop the packet. When it can send packets onward, some new packets fit again in the buffer until it is full again. This leads to a behavior in which the router drops one packet out of every few packets. The sender will continue overflowing the router buffer until it detects a packet loss which is interpreted as congestion. This phenomenon is very common with TCP connections and is called *slow start overshoot*.

[2]Three, if delayed ACKs are used.

When the receiver receives an out-of-order packet, it immediately sends a *duplicate acknowledgement* (dupack), which is an ACK that acknowledges the same sequence number as the previous ACK. A TCP sender assumes a segment is lost after receiving three consecutive dupacks and interprets it as a sign of congestion. The sender retransmits the missing segment, slows down by setting the slow-start threshold (`ssthresh`) to one half of the number of packets outstanding in the network and setting `cwnd` to `ssthresh` plus the three segments that left the network according to the three received dupacks. This is called the fast retransmit. The fast recovery algorithm is followed after the fast retransmit. In the fast recovery the rest of possible lost segments are retransmitted. The standard variant of the fast recovery algorithm exits when the first ACK advancing the window arrives. This, however, is inefficient when more than one segment are lost in the same window, which is a common case with slow start overshoot. Another fast recovery variant, *NewReno* [FH99], exits only after all packets in the last window have been acknowledged. Another significant improvement in NewReno variant is that it interprets an ACK that acknowledges some but not all of the packets transmitted before the fast retransmit as an indication of a lost packet and retransmits the packet that triggered this acknowledgement.

After the `cwnd` reaches `ssthresh` and also always after fast recovery, a *congestion avoidance* takes place. In this phase sender increases `cwnd` by one segment per each *round trip time* (RTT) therefore slowly increasing the transmission rate. This continues till the end of the connection or until a segment loss is detected.

An example of slow start overshoot is shown in Figure 3. Data packets are marked as *seqlow* which tells the lower sequence number of the segment. The buffer overflows are shown as a cross over data packets. The sender detects congestion roughly at time 10 seconds, which triggers fast retransmit and fast recovery. At time 15 seconds the slow start overshoot is recovered and the sender continues sending data in congestion avoidance.

A TCP sender maintains a timer which expires if an acknowledgement to a TCP segment is not received in certain time period after the segment is sent. This timer is called *retransmission timeout* (RTO). This ensures that the TCP sender can retransmit a packet if the three dupacks sent by the receiver are not received by the sender for some reason or all the packets sent are lost. When RTO expires sender assumes the packet is lost due to congestion in the network and slows down

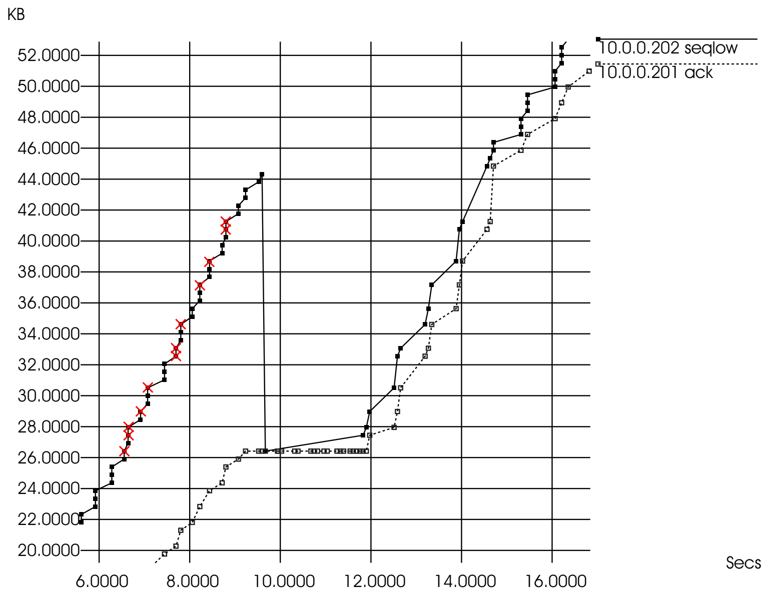

Figure 3: Slow start overshoot

by setting **`cwnd`** to one segment and **`ssthresh`** to half of the number of outstanding segments and starts sending data in slow start. Because of the smaller **`ssthresh`** the congestion avoidance will start sooner next time and the transmission rate increases slower than before.

To calculate an RTO timer according to RFC 2998 [PA00] a TCP sender has to maintain two state variables, *smoothed round-trip time* (SRTT) and *round-trip time variation* (RTTVAR). Every time a round-trip time measurement (R) is made, the SRTT value is updated by summing the old SRTT value with a high weight and the new measurement with a low weight. The RTT variation is calculated similarly by summing the old RTTVAR with high weight and the new variation with a low weight. The exact equations for the calculations are as follows and must be calculated in the given order.

$$RTTVAR \leftarrow (1 - beta) \cdot RTTVAR + beta \cdot \mid SRTT - R \mid$$
$$SRTT \leftarrow (1 - alpha) \cdot SRTT + alpha \cdot R$$

where the recommended values for *alpha* and *beta* are $\frac{1}{8}$ and $\frac{1}{4}$, respectively. After updating these variables the RTO can be calculated as well. It is calculated by summing the SRTT and the RTTVAR multiplied by 4 according to the equation

$$RTO \leftarrow SRTT + max\,(G, K \cdot RTTVAR)$$

where K is 4 and the G is clock granularity of the TCP sender. If the calculated RTO value is below one second it should be set to one second.

Traditional TCP has suboptimal performance over wireless links because of the high bit error rate and highly variable and long delays. Because of the high bit error rate many TCP segments are lost and as the TCP sender interprets the losses as a sign of congestion, it slows down and retransmits the lost segment. If the sender knew that the packet was lost because of an error on the lower protocol layer and not because of congestion, it could just retransmit the packet and continue sending new data in congestion avoidance without slowing down. The long delays can cause a TCP sender's RTO to expire. In W-WAN environment long delays can occur without any data loss seen by upper protocol layers as described in section 2. In this case the RTO is *spurious*, the delayed data segments will be eventually received and acknowledged. A spurious RTO decreases the performance because of two reasons. First, the sender will have to retransmit in slow start all data after and including the segment that triggered the RTO. Secondly, it will lower the value of `sstresh` and `cwnd`. In the case of real RTO these actions are required, but in the case of spurious RTO at least the retransmissions are needless, but some slowdown may be justified as there was a some sort of block out.

There has been a lot of research to improve the performance of TCP over wireless links with different approaches. These approaches can be divided roughly to three categories [BPS+96]: split connections, link layer solutions and end-to-end solutions.

Split connections hide the wireless link from the sender by terminating the TCP connection before the wireless link. The second connection over the wireless link can use different protocol or algorithms compared to first part of the connection to perform well over the lossy link. This approach can however violate the idea of TCP being a reliable end-to-end protocol. Link layer solutions try to hide link-related losses from higher layers. This can be achieved for example by error correction algorithms or link layer retransmissions and these algorithms can be tuned for specific links to get optimal results. TCP sender is however prone to suffer from highly variable delays because of link layer retransmits. In end-to-end approach the TCP sender tries to handle variable delays and packet losses by using different algorithms.

In next section we introduce several algorithms and TCP enhancements belonging to this category.

## 3.2 TCP Options and Enhancements

We will introduce several TCP options and algorithms most of which we expect to be useful in wireless environment. Many of the introduced options and their values are proposed in RFC 3481: "TCP over Second (2.5G) and Third (3G) Generation Wireless Networks" [IML+02].

**Increased initial window** can be up to four segments according to RFC 3390 [AFP02], which specifies the following equation for the initial window

$$min(4 * MSS, max(2 * MSS, 4380 bytes))$$

where *MSS* is maximum segment size of the TCP packet. This equation results to an initial window of two to four segments, which is recommended for 2.5G and 3G networks by the RFC 3481. Size of the initial window is important with high latency links and especially effective for short transmissions with a few TCP segments' worth of data. For the bulk transfers the gain is less significant.

**Control Block Interdependence** (CBI) [Tou97] is a mechanism to store current values for certain TCP variables concerning a connection to a specific host and use these values to determine initial values for a new connection to that same host.

After a connection has successfully closed, values for the selected variables of the connection are saved for future connections. These variables are SRTT, RTT variance, `cwnd`, `ssthresh`, and reordering. Reordering metric is the maximal distance which a packet was displaced in packet stream. When a new connection to that same host is opened these variables are used to determine initial values for the new connection.

Because the `ssthresh` is saved for future connections, the slow start overshoot can be avoided after the first connection to that same host as shown in Figure

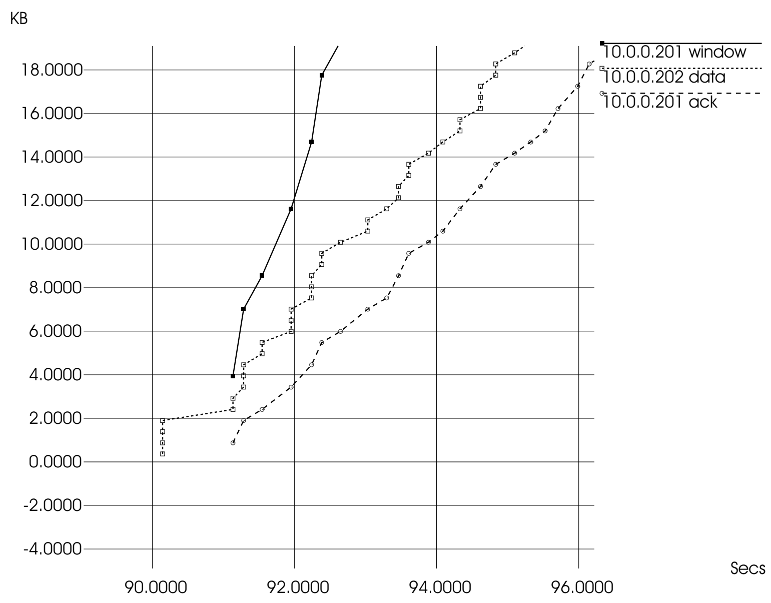

Figure 4: Initial window of 4 packets and absence of slow start overshoot with CBI option

4. This leads to a better utilization of the link since slow recovery from the overshoot is not needed.

Storing the values of SRTT and RTT variance leads to better calculation of the RTO in the beginning of the connection. Without these values, the first RTT sample for the RTO calculation is got from SYN - SYNACK pair and there are not enough RTT samples to have a proper RTT variance value.

**Ratehalving** [MSM+99] is a different way to decrease `cwnd` during the fast recovery. With the reqular fast recovery [APS99], the `ssthresh` is halved and the `cwnd` should be set to `ssthresh` plus three segments. With ratehalving the `cwnd` is not set immediately but decreased by one segment for every other incoming ACK during the fast recovery, until the `cwnd` has the right value.

**Limited transmit** [ABF01] allows a sender to send a new data segment for each of the first two dupacks if the receiver's advertised window allows and if the amount of outstanding data would remain less than or equal to the congestion window plus two segments. This is useful for small amounts of data to be transmitted or with small congestion windows that are not likely to generate the three duplicate ACKs required to trigger fast retransmit.

**Delayed ACKs** [Bra89] algorithm delays sending of ACKs. An ACK packet is delayed at most 500 ms and if another data packet arrives within that period, they are both ACKed with a single cumulative ACK. This behavior leads to a lower sending rate of ACK packets since in normal case only every other data segment is acknowledged. Althought the maximum delay allowed by the standard is 500 ms, many TCP implementations use a maximum delay of 200 ms.

If a delay between two data packets is always more than the mentioned threshold, the receiver will always wait the maximum time and only after that will it send the ACK. Therefore the maximum allowed size of a data segment should not be too large on slow links, because otherwise the transmission delay of a data packet is larger than the threshold and this would result in acknowledging every data segment and increase the RTT needlessly.

**Quick acknowledgements** [MDK+00] algorithm is used in the beginning of a TCP connection to disable delayed ACKs since the delayed ACKs needlessly slow down the slow start phase by acknowledging only every other segment. With quick acknowledgements every packet is ACKed and therefore the sender increases `cwnd` faster.

**Timestamps** option enables use of time stamps in TCP header. A TCP sender puts a time stamp in the TCP header and the receiver echoes the same time stamp back to the sender in the acknowledgement. The timestamps option can be used to calculate RTT also from retransmissions and for detecting spurious retransmissions. The timestamps option is recommended for 2.5G and 3G networks by the RFC 3481 [IML+02] although it introduces additional 12 bytes overhead per packet and current TCP header compression algorithms do not support it.

**TCP header compression** [Jac90] compresses TCP/IP headers and is very useful on low bandwidth links. It does not, however, perform well when packet losses are encountered or different TCP options, like timestamps or SACK, are in use. RFC 3481 [IML+02] recommends that the header compression should be disabled on 2.5G and 3G networks.

***Selective acknowledgement*** (SACK) [MMF+96] was designed to compensate the high error rate. Traditionally TCP uses cumulative ACKs and the receiver can therefore inform the sender only about the last segment of the continuous data block. With cumulative ACKs the sender can retransmit only one segment per RTT which is inefficient if there are several segment losses in the same TCP window. Using the SACK option the receiver can inform in one ACK packet up to four blocks[3] of data it has received properly and the sender can send the missing parts immediately, if `cwnd` allows.

***Duplicate SACK*** (D-SACK) [FMM00] makes it possible for a receiver to inform the sender that a duplicate packet was received. D-SACK uses the first SACK block to inform which data segment triggered this ACK. The rest of the SACK blocks specify in a normal manner the continuous data blocks that the receiver has received. D-SACK can be used to detect spurious retransmissions as the receiver informs the sender about every duplicate segment it has received [BA03].

***Forward RTO-recovery*** (F-RTO) [SKR02] is an alternative algorithm for the traditional RTO recovery. F-RTO was designed to avoid unnecessary retransmissions after a longer delay that causes an RTO to expire but when no packets are lost. These spurious RTOs can be common in wireless environments, because link layer retransmissions often hide the actual errors and higher protocol layers experience only longer delays. F-RTO efficiently avoids unnecessary retransmissions after a spurious RTO and it performs equally to the traditional RTO recovery when there are real packet losses. Main F-RTO behavior is the following:

- After an RTO the sender retransmits the packet that triggered the RTO and sets `ssthresh` to one half of the number of packets outstanding in the network.
- If the first ACK after the RTO advances the window, sender sends two new packets and sets `cwnd` to `ssthresh`.
- If also the second ACK advances the window, the RTO was most likely spurious and the sender continues sending new data in congestion avoidance.
- If either one of the ACKs was duplicate ACK, the sender continues retransmissions similarly to the regular RTO recovery algorithm.

[3]Three if timestamps option is in use

If the retransmission is detected as spurious, the sender continues sending data in congestion avoidance but because of the modifications to `ssthresh` and `cwnd` it has halved the transmission rate. F-RTO does not need any TCP options or bits in the header for distinguishing spurious retransmissions and the support for it is required only in the sending side of the TCP connection.

**The Eifel Algorithm** [LK00] is basically a quite simple algorithm to avoid spurious retransmissions. It uses TCP timestamps option to distinguish whether the first incoming ACK after a retransmission is acknowledgement to the original packet or to the retransmitted packet. If the retransmission is detected as spurious by Eifel, modifications to `cwnd` and `ssthresh` can be undone.

## 3.3 Active Queue Management and Explicit Congestion Notification

A router has a certain amount of buffer space where packets will be stored if they can not be forwarded immediately. If a router is between a faster link and a slower link these buffers will overflow from time to time as TCP senders try to increase the sending rate. The traditional router queue overflow policy is *tail-drop*, packets are put to the router queue until it is full. After this new packets are dropped. The tail-drop policy has two important drawbacks [BCC+98]. One or a few flows can fill up the router queue and thus monopolize it because other flows do not have space in the queue for their packets. This phenomenon is called a *lock-out* and is often the result of synchronization or other timing effects. Another problem is the filled router queue. Because a sender will slow down only after it has detected the congestion, the router queue has been full for a relatively long period by that time. As the packets often arrive at routers in bursts the full or almost full router queue will result to several packet drops. A sender experiences also longer queuing delays because of the heavily utilized router queue.

*Active queue management* (AQM) mechanism drops incoming packets at a router before the buffer becomes full. This way an AQM capable router can choose when and how many packets to drop and therefore keep the queue utilization at a low level. There are several advantages of the low router queue utilization. The number of dropped packets can be reduced as there is space for short bursts, a sender experiences lower queuing delays and the lock-out behaviour can be avoided. The fairness among concurrent separate flows should also be better as all flows have space in the router queue.

The most widely used AQM mechanism, *Random Early Detection* (RED) [FJ93], is an algorithm for a router to signal congestion when its queue starts to fill. There are different possibilities to provide the congestion signal but a common method is just to drop the packet. If TCP protocol is used the sender will slow down when it experiences a packet drop since a packet drop is considered, correctly in this case, as a congestion notification.

The RED algorithm calculates the average queue size of a router using a low-pass filter with exponential weighted moving average. If the average queue size $avg$ is less than the minimum threshold $min_{th}$, a packet is not dropped and if it is more than the maximum threshold $max_{th}$ a packet is dropped. Between the thresholds a packet is dropped with probability $p_a$. The algorithm in general is the following:

```
for each packet arrival
  calculate the average queue size avg
  if min_th <= avg < max_th
    calculate probability p_a
    with probability p_a:
      drop the arriving packet
  else if max_th <= avg
    drop the arriving packet
```

and the average queue size is calculated as follows:

```
if the queue is nonempty
    $avg \leftarrow (1 - w_q) \cdot avg + w_q \cdot q$
else
    $m \leftarrow f\,(time - q\_time)$
    $avg \leftarrow (q \cdot w_q)^m \cdot avg$
```

where $q\_time$ is start of the queue idle time, $time$ is current time, function $f(t)$ grows linearly as time $t$ grows, $w_q$ is queue weight and $q$ is current queue size. The marking probability $p_a$ increases while the average queue size increases. $p_a$ is calculated according to following equations:

$$p_b \leftarrow max_p \cdot \frac{avg - min_{th}}{max_{th} - min_{th}}$$

$$p_a \leftarrow \frac{p_b}{1 - count \cdot p_b}$$

where $max_p$ is the maximum probability. $p_b$ is the probability that increases from zero to $max_p$ as the average queue size increases toward to $max_{th}$. *count* is the number of packets sent since the last drop. Therefore the probability $p_a$ increases with the average queue size. The *count* value also increases marking probability to avoid biases and global synchronization and to mark packets frequently enough to control the average queue size.

To avoid using a drop alone as a congestion signal, the *explicit congestion notification* (ECN) [RFB01] was specified. With active queue management and ECN the router can actually mark the packets instead of dropping them. This way the sender may be able to slow down before router buffers overflow and therefore retransmissions can be avoided. ECN uses two bit ECN-field in the IP header which was originally part of the *Type of Service* (TOS) field [IP81] and two originally reserved bits from the TCP header. ECN requires support from both the sending and the receiving end and also from routers between them. The four ECN bits are used as follows

- Sender sets *ECN-capable transport* (ECT) flag in the IP header to inform routers that ECN is understood.
- When a congested router's RED algorithm chooses a packet for marking the router marks the packet by setting the *congestion experienced* (CE) flag in the IP header.
- The TCP receiver that receives the congestion experienced mark, sets *ECN congestion echo* (ECE) flag in the TCP header back to the TCP sender.
- When a TCP sender receives a congestion echo mark it will slow down by reducing the `cwnd` and `ssthresh`. The sender will also set *congestion window reduced* (CWR) flag in the next packet's TCP header after receiving the ECE flag to inform the it has reacted to the CWR-flag.

## 3.4 Streaming Traffic

Besides the bulk data transfer, where the only important metric for the user is the finishing time, there are also other types of data transfers, like streaming data transfer. It is a transfer of continuous stream of data, and the data is consumed more or less immediately when it arrives and then discarded. The data can be for example endless stream of a radio broadcast or a phone call.

Streaming transfers require steady flow of bits, otherwise video or audio will not be fluent. Even the smallest breaks in audio stream are annoying to human ear. For a video stream we are a bit more tolerant and the missing packets can be compensated with more unnoticeable methods. When downloading for example a movie trailer the stream can be buffered for a few seconds and a wider jitter can be tolerated and short breaks avoided with retransmissions. But with interactive streams buffering is impossible because of the high latency.

*Real-time transport protocol* (RTP) [SCF+96] provides end-to-end network transport functions suitable for applications transmitting real-time data, such as audio, video or simulation data, over multicast or unicast network services. It is often run over *user datagram protocol* (UDP) [Pos80] although it is not tied to it. Use of reliable flow like TCP is unnecessary for data with real time requirements, for example interactive streaming, because there is no time to wait for lost and re-transmitted packets. Because of the possibly non-reliable transport layer, video and audio specifications are usually designed to tolerate lost packets using encoding methods that minimize the effect of missing data. A lot of work has been done to reduce bandwidth demands of streaming video. ITU-T specifies H.263 [H263] which is a low-bit-rate video compression algorithm and ISO/IEC has specified MPEG-4 [MPEG4]. Compression rates as high as 80% can be achieved using algorithms based on these specifications.

Streaming transfers are established and controlled with for example *real time streaming protocol* (RTSP) [SRL98]. RTSP does not deliver the data itself, instead it acts like "network remote control". The data is delivered using for example RTP but the operation of RTSP does not depend on the transport mechanism used to carry continuous media.

*RTP control protocol* (RTCP) [SCF+96] specifies report packets exchanged between sources and destinations. Reports contain statistics such as the number of packets sent, number of packets lost, and inter-arrival jitter. RTCP packets are sent from each of the receivers to the sender and therefore the RTCP adjusts the interval between reports based on the number of participating receivers. Typically the limit for the RTCP bandwidth is 5% of the session bandwidth, divided between the sender reports (25%) and the receiver reports (75%).

## 3.5 Quality of Service in the Internet

Original purpose of the Internet was to deliver information using email or copying files among the few machines that were connected in those days. Speed of the delivery was not important and the only service provided was *best effort service*; the network does its best to deliver the data packets, but no guarantees can be given for example about the delay. With interactive services like WWW the speed increases usability, but high speed is still not critical. High speed and stable connections have become necessary along with different types of streaming media. For a fluent stream of audio or video the data flow must not experience congestion. This can not be easily guaranteed in today's Internet. IPv4 has always had type-of-service field in the header but it has not been used widely. Many companies do business in the Internet and are willing to pay for use of a good-quality network. There are many other reasons as well why different *Quality of Service* (QoS) types are required.

*Integrated services* (IntServ) [Wro98] with for example *resource reservation protocol* (RSVP) [BZB+98] is one solution for QoS architecture. IntServ has, however, some problems which are the reason why it did not succeed well. It requires a lot from routers and the amount of state information increases proportionally with the number of flows and does not therefore scale well. It is also too complicated. *Differentiated services* (DiffServ) [BBC98] was designed to outcome these problems. With DiffServ packets are classified by certain rules when they enter a DiffServ capable network and then treated according to that class. These different classes can for example specify an upper limit for the transmission rate and allow certain amount of burstiness for the traffic.

# 4 Test Arrangement

In this section we introduce the methods used in performance study, including emulation of the target environment. We describe the workload and the network characteristics used in the measurements. The difference between the Linux TCP used and the normal behavior according to various RFCs is explained. Finally test sets and metrics are listed.

## 4.1 Modeling the Target Environment

The target environment consists of a mobile host which is connected to a fixed host through a last-hop router. We use Seawind [KGM+01] to emulate a W-WAN environment. The target environment is shown in the upper section of the Figure 5 and the lower section describes our emulation in Seawind. The upper section shows a server in the fast fixed network sending data using multiple flows to a mobile host through a last-hop router and slow wireless link with variable delays and packet losses. In the lower section the main component is the emulation host. It has an input queue which emulates a router queue and link send and receive buffers (LSB and LRB, respectively) which emulates the link layer buffers. Delays and packet losses are done between the send and receive buffers.

The hardware consists of three PCs; one for mobile host, one for fixed host and one for the host that does the actual emulation of the wireless link and a last-hop router; the emulation host. These three PCs are connected together using private LAN with 100 Mbit/s of bandwidth. Emulation host is connected also to the Computer Science department's 100 Mbit/s LAN. The fourth PC is used just like a remote terminal to run the graphical user interface remotely.

Last-hop router has typically one router queue per outbound link and therefore mobile hosts will not interfere with each other. Router queue size is not relevant for uplink because the much faster fixed link should always be ready to deliver data from the last-hop router. In the emulation environment the router queue corresponds to input queue of the Seawind emulation host.

Link send and receive buffers are needed for the link layer to provide reliable transmission to upper level protocols. If a packet corrupts or does not arrive to

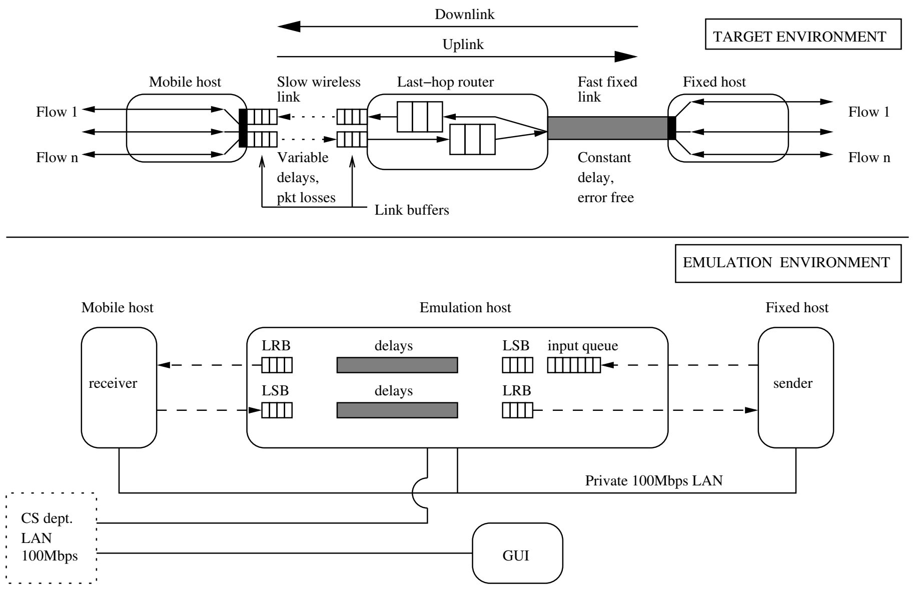

Figure 5: Target and emulation environments

destination for some other reason the link layer can retransmit the missing packet. While the packet is being retransmitted the other already arrived data packets, which should be delivered after the missing packet, wait at the link receive buffer. When the missing packet arrives to the destination, all the pending packets are delivered onward at once.

Operating system in each of the computers is Linux. Sending and receiving computers have *workload generator* processes (WLGs) which create and receive the data. The end hosts use the real TCP/IP stack of the Linux kernel. The kernel used in our tests was the newest at the moment, 2.4.20, with some modifications [IIP]. With these modification it is possible to disable or tune some TCP features.

Because Seawind is a real-time emulator, there should not be any other CPU or network intensive processes running on the emulation host. Detailed TCP dump logs are gathered from each of the hosts and a log is created in the emulation host which tells every event that a packet was affected by. It is very important that the delays created by the emulation host are accurate and therefore both the requested and the actual delay are logged. If the actual delay exceeds the requested delay by more than 10 ms a warning is logged and the effect of the oversleep can be easily detected and test run can be discarded if necessary.

## 4.2 Workloads

Our workloads consist of different types of data transfers listed in Table 2. All data flows are directed downlink from the fixed host to the mobile host. There were 10 replications for the baseline TCP with optimal link and 20 for rest of the test cases.

| Workload | Data amount (bytes) |
|---|---|
| 2x TCP | 2x 188640 |
| 2x TCP, 5sec between | 2x 188640 |
| 2x TCP + 1x UDP | 2x 188640 + 32000 bits/s |

Table 2: Workload types

The test runs contain the following three different workloads:

**2 TCP connections**. Both connections start at the same time.

**2 TCP connections, 5 second delay between start times**. The first connection starts at time 0 seconds and the second connection at time 5 seconds.

**2 TCP connections and 1 UDP flow**. All data flows start at time 0 seconds. The length of UDP transmissions are 180 seconds which is enough for the TCP connections to finish before the UDP flow.

The first two workloads are pure TCP workloads with different starting times and one workload contains one additional UDP flow. With the first workload we can analyze how two TCP connections react to each other when they are started at the same time. The optimal case would be no reaction at all, they should both proceed similarly taking half of the bandwidth. In the second case the first TCP sender has already increased its sending rate and therefore the second TCP sender can more easily experience the lock-out situation. When a UDP flow is running in addition to two TCP data flows the available bandwidth for TCP connections is lower as the UDP requires its own share of it and the router queue can be more utilized.

In our tests the total amount of data transferred by TCP is always 377280 bytes. With two concurrent connections both connections will transfer 188640 bytes in 360 full size packets when the TCP timestamps option is used. Without the timestamps option the amount of packets is 352 of which the last one is not full sized. The amount of data was chosen so that it can be divided between one and two connections and it should be large enough for different TCP data transfer behaviors to emerge.

The UDP flow is a *constant bit rate* (CBR) data flow consisting of packets with 512 bytes of payload. The constant payload bit rate is 32000 bit/s, so a UDP packet is sent every 128 ms. The *tcptrace* [Ost] program used to calculate the throughput in analysis includes UDP headers in results. IP payload, including the UDP header, is 520 bytes and the CBR is 32500 bit/s (4062.5 bytes/s) with the UDP headers.

The traffic generator used in the measurements for both TCP and UDP traffic is *Jugi's Traffic Generator* (JTG) [Man] which is based on *Test TCP* [PS98] program. Using JTG we can specify for example the buffer size used when writing to TCP socket. By selecting a proper buffer size and the amount of data according to link characteristics we can be ensured that the TCP packets sent to the network are always the same size. For UDP flows we can specify for example constant bitrate and the packet size.

## 4.3 Network Characteristics

Link and network characteristics are listed in Table 3. Router queue length is 20 packets toward mobile host which is about two times *bandwidth delay product* (BDP) as are link buffers although queue length is defined as packets and link buffers as bytes. Propagation delay is 300 ms. Transmission rate is symmetric 64000 bit/s. *Maximum transfer unit* (MTU) of the wireless link is 576 bytes.

For the RED router the values listed in Table 4 were used in the tests. The values are based on recommendations [Flo]. The threshold values differ slightly from the recommended values, because they are the values of the router queue and they do not take into account the packets in the link buffers.

We use a two-state error model for the link layer error probability where one state represents a good error free wireless environment and the other state represents a bad environment with high error probability. Every time a good state ends the bad state starts and vice versa. In bad states ARQ link layer retransmissions with

| Parameter name | Downlink value | Uplink value |
|---|---|---|
| Router queue length | 20 packets | 1 packet |
| Queue overflow handling | drop, RED | drop, RED |
| Link send buffer size | 9600 bytes ( 2x BDP) | 9600 bytes ( 2x BDP) |
| Link receive buffer size | 9600 bytes ( 2x BDP) | 9600 bytes ( 2x BDP) |
| Propagation delay | 300 ms | 300 ms |
| Transmission rate | 64000 bit/s | 64 000 bit/s |
| MTU | 576 bytes | 576 bytes |

Table 3: Link and network characteristics

| Name | Value |
|---|---|
| Minimum threshold | 2 packets |
| Maximum threshold | 7 packets |
| Maximum probability | 0.1 |
| Queue weight | 0.002 |

Table 4: Parameters of the RED algorithm

different amount of persistency are emulated. If a packet is dropped, the link layer retransmission is emulated by delaying the packet for about one RTT for each retransmission instead of actually dropping it. The one RTT additional delay is defined as 700 ms in our tests. After the emulated retransmission the packet may experience a drop again and might need additional retransmissions. The different wireless link types are the following:

- **e0** is an optimal link. There are no errors and therefore no link layer retransmissions.
- **r1** is a lossy link with low ARQ persistency. The link layer tries to retransmit a missing packet at most once. The length of a good state is specified by an exponential distribution with the mean value of 15 seconds, minimum value of one second, and maximum value of 20 seconds. In the bad state the packet error probability is 63% and the state length is uniformly distributed between 0.2 and 1.5 seconds. There will be error related drops as this link is able to hide packet drops only for short bad states with error bursts.

- **r3** is a lossy link with medium ARQ persistency. The link layer tries to retransmit a missing packet at most three times. Length of the states and error probabilities are the same as with *r1* link. This link layer is persistent enough to recover from all packet losses during a bad state and upper protocol layers should not therefore experience any error-related drops.

- **r6** is a lossy link with high ARQ persistency. The link layer tries to retransmit a missing packet at most six times. The bad state is longer and packet error rate higher than with *r1* and *r3* links. With this link the length of the bad state is defined by an uniform distribution between 0.5 seconds and 4.0 seconds and the packet loss probability in the bad state is increased to 95%. Six link layer retransmissions are enough to hide all error related packet drops and upper protocol layers should experience only delays.

## 4.4 Linux TCP Features

Linux mainly follows the principles of the congestion control algorithms and different enhancements as specified by the RFCs, but all TCP implementations have some specific features of their own. In this section we describe those features that were used in this study and implemented differently in our Linux TCP version compared to TCP as specified in RFCs or TCP behavior in other widely used TCP implementations.

Because for example CBI and ratehalving algorithms cannot be disabled in vanilla Linux kernel we used some modifications [IIP] to alter the default behavior of the kernel. The ratehalving algorithm was disabled in every test, so the `cwnd` was reduced immediately. Linux does not use a static value for delay ACKs, but adjusts it to be two times the estimated packet interarrival time. In our tests we used a static value of 200ms for delayed ACKs and quick acknowledgements were disabled.

### Control Block Interdependence and Initial Window

In our analysis the CBI algorithm was measured as one of the enhancements. When the CBI algorithm was disabled, no stored variables were used for determining initial values for new TCP connections and a static value of two segments was used for the initial value of `cwnd`. With the CBI algorithm the initial window value was set according to the normal Linux kernel behavior, which sets it to one to four segments by the following rules:

```
if MSS > 1460 bytes
  IW = 2
else
  if MSS > 1095
    IW = 3
  else
    IW = 4
```

Next, if a stored value of `ssthresh` exists, the value of IW is compared to it and if IW is higher the IW is set to `ssthresh`.

### SACK Based Error Recovery and Congestion Control

Linux TCP uses a *scoreboard* to keep track of the sent packets. There are separate flags for every packet. These flags are *sacked*, *retransmitted* and *lost* and a packet can have several flags on at the same time. Using this scoreboard the TCP knows which packets are lost, received by a receiver, SACKed or retransmitted. A packet is marked as lost in the following events; if the sender is in fast recovery and an RTO expires, if there are a certain[4] amount of SACKed packets above a non-SACKed packet [BAF+03]. Slow start overshoot recovery with the SACK option is shown in Figure 6. Because every dupack has SACK information, only non-SACKed packets are retransmitted. The packet at the recovery point is not retransmitted until the ACK that acknowledges up to the recovery point is received since it carries the SACK information of the third segment above the lost, non-SACKed, segment.

The Linux TCP uses the equations [SK02]:

```
left_out <- sacked_out + lost_out
in_flight <- packets_out - left_out + retrans_out
```

[4]Normally three

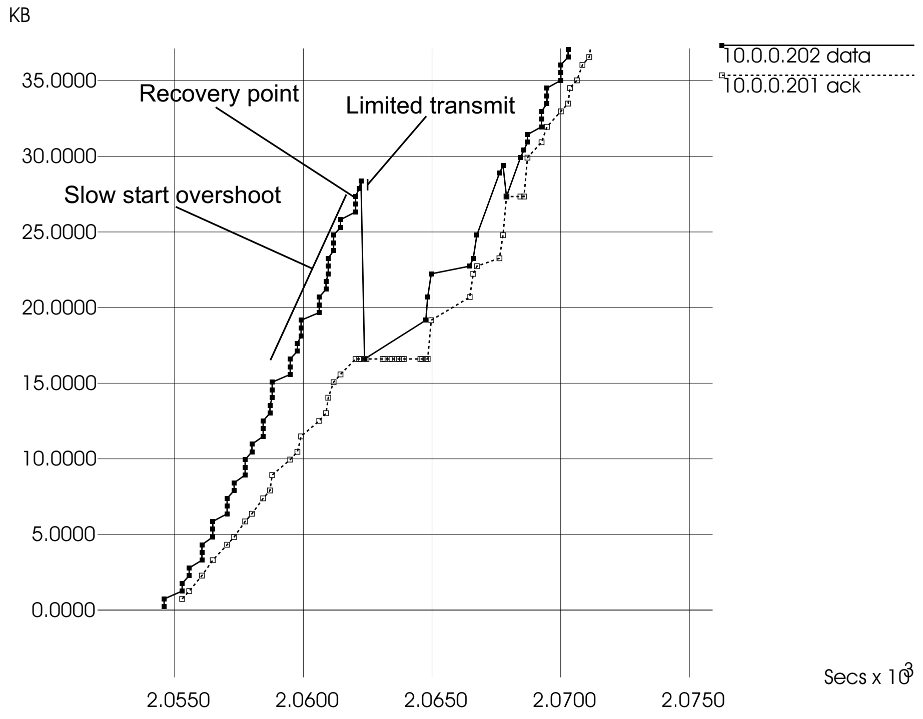

Figure 6: Slow start overshoot recovery with the SACK option

in calculating the number of segments outstanding in the network and the calculated value is stored in variable `in_flight`. The number of packets acknowledged by SACK blocks is stored in variable `sacked_out`. In variable `lost_out` is an estimation of the number of packets lost in the network, `left_out` is the number of packets that have left the network, `packets_out` is the number of packets sent but not acknowledged by a cumulative ACK, `retrans_out` is the number of retransmitted packets.

Linux TCP compares the `in_flight` number to `cwnd` and if lower, a packet can be sent. Using the scoreboard the Linux TCP decides what packets to transmit. If there are packets in the scoreboard marked as lost but not retransmitted, they are retransmitted, otherwise new packets can be transmitted.

### Linux RTO calculations

The current retransmission timer specification [PA00] requires that RTO timer should not be less than one second. Linux's RTO timer has a minimum value of 200 ms and a timer granularity of 10 ms, where in many other implementations the granularity can be as high as 500 ms. Because of these differences the Linux RTO calculations can achieve better accuracy.

The RTO equations stated in the specification have two problems [SK02]. If the RTT decreases suddenly the RTO value will be overestimated because the RTT variance grows momentarily. The second problem is that the RTT variance can be very low when the RTT is taken from every packet and if the window is big.

For the first problem Linux RTO introduces a new variable `MDEV` for the measured mean deviance. The following code is used to calculate MDEV:

```
if ($R < SRTT$ and $|SRTT - R| > MDEV$)
then
    $MDEV \leftarrow \frac{31}{32} \cdot MDEV + \frac{1}{32} \cdot |SRTT - R|$
else
    $MDEV \leftarrow \frac{3}{4} \cdot MDEV + \frac{1}{4} \cdot |SRTT - R|$
```

where R is the newest RTT measurement and SRTT is a smoothed RTT value. If the newest RTT measurement is lower than the smoothed RTT and if the difference is bigger than the mean deviance, then a very light weight is given to the newest RTT measurement. Otherwise the weight is given as recommended in the specification [PA00].

If the new MDEV is higher than before, the RTTVAR variable is set to MDEV immediately. If not the RTTVAR is updated only once per one round trip time. The following equations are used to calculate the actual RTO timer:

$$SRTT \leftarrow \frac{7}{8} \cdot SRTT + \frac{1}{8} \cdot R$$

$$RTO \leftarrow max\,(SRTT + 4 \cdot RTTVAR, 200ms)$$

### Detecting Spurious Retransmissions

The RTO can expire spuriously if there is a longer delay although no packets are lost. Fast retransmit can also be triggered spuriously as a result of reordering in the network. A Linux sender has three ways to detect a spurious retransmission; F-RTO[5], D-SACK and TCP timestamps. With timestamps Linux uses an Eifel like undo mechanism [SK02] to revert the `ssthresh` and `cwnd` values. If a spurious RTO is detected with Eifel the sender removes the lost flags from the scoreboard and sets the `ssthresh` to the saved value and continues sending new data in the congestion avoidance. If the sender is in fast recovery it increases the `cwnd` to the maximum of its current value and two times `ssthresh` and sets `ssthresh` to its saved value. These modifications lead automatically to the end of the fast recovery, since new data can be sent and no packets are flagged as lost. The main difference between D-SACK and Eifel is the retransmissions, with Eifel the sender can stop when retransmission are detected as spurious but with D-SACK the conclusion can be done only after all retransmissions are done and acknowledged as duplicates.

## 4.5 Test sets

The combinations of different link layer characteristics and TCP enhancements are listed in Table 5. In the table the marking *BL* means our baseline TCP. The mark • states that the tests will be run for every workload and ○ means only the workload where two TCP connections start simultaneously. The RTO variants are run with the baseline, but with other TCP enhancements the Eifel acts as the baseline TCP. To limit the amount of different test cases, the different RTO variants are tested with all ARQ persistences, but only with the workload in which the two TCP connections start simultaneously. Other TCP enhancements are tested with all workloads but only with optimal and low ARQ links. With the medium ARQ persistency the other TCP enhancements are tested with the workload in which the two TCP connections start simultaneously.

| Enhancement<br>Link | BL | BL +<br>FRTO | BL+<br>DSACK | BL+<br>EIFEL | EIFEL+<br>RED | EIFEL+<br>ECN | EIFEL+<br>CBI | EIFEL+<br>CBI+ECN |
|---|---|---|---|---|---|---|---|---|
| e0 | ○ | ○ | ○ | • | • | • | • | • |
| r1 | ○ | ○ | ○ | • | • | • | • | • |
| r3 | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| r6 | ○ | ○ | ○ | ○ | - | - | - | - |

Table 5: Combinations of link layer characteristics and TCP enhancements. • TCP + UDP, ○ TCP only, - not run.

[5]Included in Linux kernels starting from 2.4.21

Table 6 lists TCP options and algorithms and their use in tests. Our baseline TCP has limited transmit and SACK always on, delayed ACKs set to static 200 ms, initial window is set to 2 segments and ratehalving is always off. The static initial window size is not used if the CBI option is in use. The baseline TCP has timestamps option disabled. With timestamps option enabled the Linux TCP uses Eifel algorithm and it is run as one of the enhancements when comparing different RTO recovery variants, and as the baseline TCP when comparing other TCP enhancements.

| Baseline | Value |
|---|---|
| Limited transmit | yes |
| SACK | yes |
| Delayed ACKs | 200 ms |
| Initial Window | 2 |
| Quick ACKs | no |
| Ratehalving | no |
| **Enhancements** | **Value** |
| F-RTO | subset |
| D-SACK | subset |
| Timestamps (Eifel) | subset |
| Eifel + RED | subset |
| Eifel + RED + ECN | subset |
| Eifel + CBI | subset |
| Eifel + CBI + RED + ECN | subset |

Table 6: TCP options and algorithms

## 4.6 Metrics

Below we introduce the different metrics used in our tests. After the measurements we conduct our analysis based on these metrics. The main metric for TCP is elapsed time. Another important metric is the number of packet drops since it affects the elapsed time of the TCP transfer a lot.

In every workload there are two TCP connections and in some test cases one additional UDP flow. The two TCP connections are divided into two sets. In cases where the two connections are started at the same, the first set contains the faster TCP connection and the second set contains the slower. In the workload case where there is a five second difference between the starts of the TCP connections the first connection is always counted to the first set and the second connection to the second set. The metrics are then calculated based on these two sets.

**Elapsed Time** is our main TCP performance metric. The elapsed time is the time between sending the first TCP (*SYN*) packet of the connection and receiving the last ACK for the (*FIN*) packet of the same connection. These are measured at the TCP sender side. We will report elapsed time in two different aspects:

- **connection time** (ctime) is the elapsed time calculated separately for the first connection set and the second connection set. There are listed in the results the 25% and 75% percentiles and the median value.
- **overall time** (otime) is the elapsed time calculated over all connections in both connection sets. In the results only the median value is listed.

**Throughput** is directly calculated by dividing the amount of workload data by the elapsed time. UDP throughput is calculated by dividing the amount of received data, including the UDP headers (8 bytes per packet), with the time between the first and the last received packet. For UDP the minimum value, 50% percentile, and the maximum value of the throughput are listed. For TCP this metric is listed only in the tables in the appendices.

**Fairness** is studied using the difference of elapsed times between the two TCP connection sets. It is not automatically calculated and listed, but determined by comparing the times.

**Stability** is the difference of 25% and 75% percentiles within a connection set. The closer they are, the better the stability. It is not automatically calculated, only discussed.

**Number of retransmissions** is the amount of TCP segments retransmitted as they are considered lost by the sender. These retransmissions often affect the elapsed time significantly since they increase the amount of data that is sent across the link. Some of the data is really lost and the retransmissions can not be avoided but in some cases the sender retransmits the data needlessly.

**Number of drops** is the number of packets lost in the network. In our environment there are three reasons for a packet to be dropped; a link error has occurred and the packet is lost, the router buffer overflows and the packet must be dropped, or a RED capable router drops the packet as a sign of increasing congestion. Median values of the packet drops for each of the three drop types are listed in the drop column. ECN marks as well as actual drops with ECN are both counted under the RED drop column. For UDP flow listed values are the minimum value, 50% percentile and the maximum value for each of the drop types.

**Number of RTOs** can not be counted easily by any automatic method and therefore they are counted manually in some selected cases. An RTO affects the performance of the connection a lot since typically there is a relatively long idle period before it and the sender must start with slow start and small congestion window after an RTO.

**Router queue length** is measured to analyze the congestion status in a router. This metric is not listed in tables but shown in graphs in the cases where it affects the behavior of a data flow. A queuing delay increases as the queue length increases and therefore it affects the round trip time.

**Jitter** is listed for the UDP packets. These values show the spacing between two received UDP packets in milliseconds. If a packet is lost and the two received packets are not therefore consecutive, the spacing value is ignored. We provide 50%, 75% and 90% percentiles and the maximum values. There are two maximum values, one is the lowest maximum value among the replications and the second is the highest of the maximum values.

# 5 Results

In this section we analyze the results. We start by analyzing enhancements over optimal link and then over lossy links, from low ARQ to high ARQ. With all link types we analyze one workload type at a time with all enhancements. This way we can easily compare different enhancements to each other with a certain workload and certain link characteristics.

There is a short summary in the beginning of each workload section which gives overall view of the results. The summary includes one or more tables, which list the measured results. Most of the metrics introduced in section 4.6 are listed in the tables but some are left to appendices. For the two TCP connections starting simultaneously there are listed 25%, 50% and 75% percentiles of the elapsed times for the slower and for the faster connection sets. Elapsed time of the slower connection in the case where the two TCP connections start simultaneously is important as it tells the total time. The elapsed time is marked as *ctime*. When the two TCP connections do not start simultaneously the same percentiles are listed for the first and for the second connection set even in the case the second connection would be faster. The amount of dropped packets are listed separately for queue overflow, RED, and error related drops. The values are medians and are marked `q / r / e`, respectively. The overall time is listed as the last column and marked as *otime*. For UDP flows there are two tables in summary sections. One has the information concerning the three different types of packet drops and the second table has the information about throughput and spacing of the received UDP packets. There is a graph in the summary section, which shows the minimum, 50% percentile and the maximum values of the elapsed times of the two TCP connection sets. The graphs make it easy to compare the effect of the different enhancements to elapsed time.

The following markings are used for the different workloads:

**2tcp0udp**: Two TCP connections starting at the same time.

**2tcp0udp5s**: 5 second delay between the starts of the TCP connections.

**2tcp1udp**: Two TCP connections starting at the same time and an additional UDP flow.

## 5.1 Optimal Link

This section introduces the analyzed results of the optimal link. There are three workloads and a total of seven TCP enhancements or combinations in addition to the baseline TCP for the *2tcp0udp* workload. *2tcp1udp* and *2tcp0udp5s* workloads were not measured with different RTO variants.

### 5.1.1 Two TCP Connections Starting at the Same Time

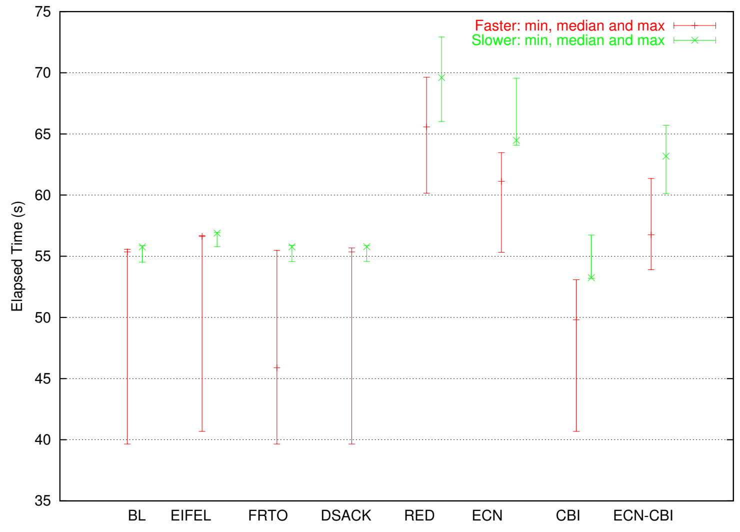

Figure 7: *Optimal link: 2tcp0udp*: Elapsed times of different enhancements

Summary of the test results with different options is listed in Table 7 and elapsed times are shown in Figure 7. Since there were no errors, our baseline TCP worked well although in some cases the fairness was very poor because of the differences in slow start overshoot recovery. Stability was much better with RED or CBI compared to baseline TCP. The RTO-variants did not affect the situation significantly as there were no RTOs. RED increased the elapsed times significantly, but kept the stability better. ECN worked similarly to pure RED, but decreased the elapsed times a bit. With CBI the fairness was good, because the slow start overshoot was avoided in most cases, and the number of congestion related packet losses was low. Use of the ECN with the CBI did not introduce any significant improvements since the CBI alone gave very good results and the additional ECN marks increased the elapsed times.

| TCP variant | ctime (s) Faster 25% | 50% | 75% | ctime (s) Slower 25% | 50% | 75% | Drops (50%) Faster q / r / e | Drops (50%) Slower q / r / e | otime (s) 50% |
|---|---|---|---|---|---|---|---|---|---|
| BL[1] | 46.2 | 55.4 | 55.4 | 54.7 | 55.8 | 55.8 | 14 / 0 / 0 | 15 / 0 / 0 | 55.4 |
| EIFEL | 40.8 | 56.6 | 56.6 | 56.8 | 56.9 | 56.9 | 14 / 0 / 0 | 15 / 0 / 0 | 56.7 |
| FRTO | 39.7 | 45.9 | 55.3 | 54.6 | 55.8 | 55.8 | 14 / 0 / 0 | 12 / 0 / 0 | 55.0 |
| DSACK | 39.7 | 55.4 | 55.4 | 55.7 | 55.8 | 55.8 | 14 / 0 / 0 | 15 / 0 / 0 | 55.4 |
| RED | 64.6 | 65.6 | 65.9 | 68.2 | 69.6 | 70.8 | 14 / 7 / 0 | 12 / 10 / 0 | 67.1 |
| ECN | 57.8 | 61.1 | 61.6 | 64.4 | 64.5 | 65.5 | 13 / 6 / 0 | 15 / 6 / 0 | 63.8 |
| CBI | 49.8 | 49.8 | 53.1 | 53.2 | 53.2 | 53.3 | 0 / 0 / 0 | 1 / 0 / 0 | 53.1 |
| CBI-ECN | 56.3 | 56.8 | 59.1 | 60.6 | 63.2 | 63.5 | 0 / 6 / 0 | 0 / 9 / 0 | 60.4 |

Table 7: *Optimal link: 2tcp0udp*: Summary of the results. 20 replications ([1]10 replications).

## Baseline TCP

Without any link errors there was only little variance between the basic tests, which was expected. All the basic tests divided into two different behavior categories, except one. The differency between these two categories was the slow start overshoot recovery.

In the first category both connections behaved very similarly and recovered like the normal efficient SACK recovery but in the second category one of the TCP connections recovered very slowly which was the main problem in this test. It managed to retransmit only one packet per RTT (see Figure 8) because roughly half of the packets in the TCP window were lost. The first ACK that acknowledged new data after the fast retransmit decreased the number of packets in flight and the `cwnd` was only one packet greater than the number of packets in flight. Therefore the sender could retransmit only one packet. The next ACK acknowledged this newly retransmitted packet and again only one packet could be retransmitted. After the recovery point the connection continued congestion avoidance but the other connection was in the meanwhile getting almost the all available bandwidth and it was able to complete the transfer much faster.

Fairness was good in the first category as the connections proceeded similarly. In the second category the fairness was much worse; faster connection was ready in most replications in 46 seconds, but for the slower it took roughly 55 seconds. Most

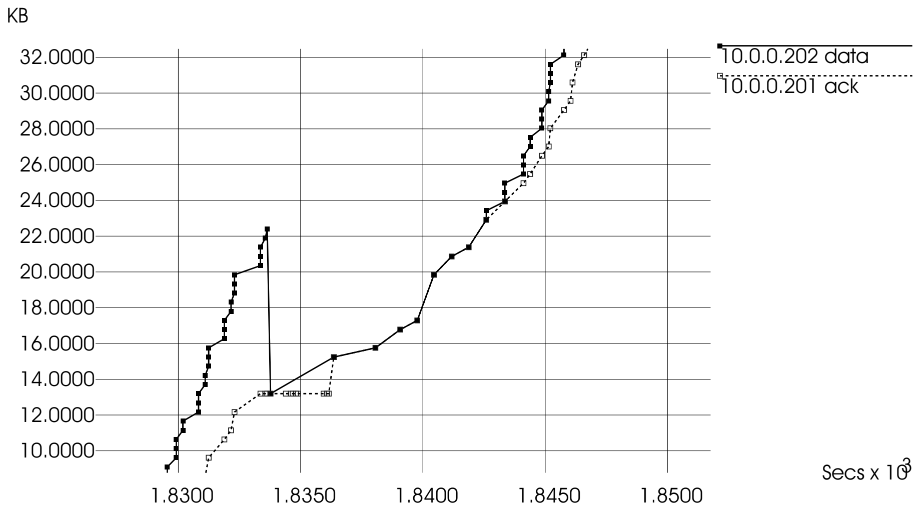

Figure 8: Inefficient slow start overshoot recovery

of the replications falled into the first category.

## Eifel, F-RTO and D-SACK

Because the connections did not experience any RTOs there were no significant differences between baseline TCP, Eifel, F-RTO, and D-SACK. Only small difference was the additional overhead caused by timestamps in the Eifel test. The lower elapsed time median of the faster F-RTO connection was because the latter behavior discussed with the baseline TCP was slightly more common. The latter behavior was more common because there happened to be slightly lesser amount of packet losses.

## RED

Using a RED router the medians of elapsed times were about 10 seconds greater than without it. This was because the RED router dropped packets as a mark of increasing congestion as the RED algorithms average queue size increased. There were median of 8 RED based drops for the faster connection and 10 for the slower connection in addition to the packet drops due to queue overflow. The effect of the drops can be severe because these drops lead easily to an RTO in the slow start overshoot recovery as the difference between packets in flight and `cwnd` is small and the sender can therefore retransmit only a few packets per RTT. The reason for the high RED drop probability during slow start overshoot recovery can be seen in Figure 9. Because the average queue size of a RED router did not decrease immediately, it was still high when the sender was recovering from the slow start overshoot.

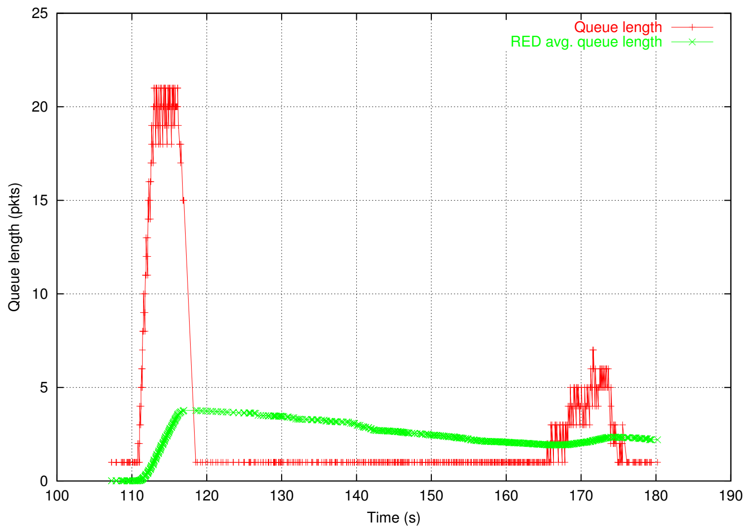

Figure 9: An actual queue length and an average queue length of a RED router

Although the packet drops due to RED slowed down the connections significantly, the connections were more stable than without RED. RED improved the fairness compared to second category discussed with the baseline TCP, the median of faster connection set with RED was only 6% faster than the slower. Queuing delays were lower with RED than without it because, as seen in Figure 9, the input queue size stayed low after the slow start overshoot. Lower queuing delays gave the recovering connection lower RTT and therefore it was able to increase the sending rate faster. Without RED the recovering connection could experience congestion drops because the faster connection had filled up the queue, but with the RED algorithm the faster connection had greater probability to be marked and therefore it had to slow down more likely than the slower connection. Therefore there was a lower possibility for a lock-out situation, a situation where the faster connection locks out the slower one by filling up the route queue and getting all available bandwidth.

### ECN

Using the ECN option with the RED algorithm introduced roughly 5% improvement in median of elapsed time compared to pure RED. The improvement was expected because there are fewer packet drops with ECN since the packets are just marked, not dropped.

The ECN RFC ([RFB01] section 6.1.5) states that ECN ECT flag must not be set on retransmitted packets. Therefore when a RED algorithm marks a packet that is a retransmission the router must drop the packet. It can not mark the packet with the CE flag, even if the ECN is supported among the sender, the receiver, and the router. Because of this the ECN had the same negative effect on the slow start overshoot recovery as with a RED router without ECN option; an RTO was easily experienced, because some retransmitted packets were dropped.

### CBI

The effect of the saved `ssthresh` value was very significant; the slow start phase was very short, only few round trip times. Because the slow recovery from slow start overshoot was avoided the use of CBI option gave better throughput than the baseline TCP. Since the CBI prevented slow start overshoot after the first replication the queue size remained low and both connections proceeded at the same rate.

Usually there was a single congestion drop on both connections at the start of the last quarter of the transfer as shown in Figure 10. Both recovered efficiently and continued the transmission at roughly the same rate. A few times the congestion drop occurred only for one of the connections. This introduced lower fairness as the recovering connection had to slow done while the other connection got more bandwidth. If both of the connections experienced a congestion related packet drop at the same time, the fairness was very good. Even if not, the fairness was still good since the one congestion related packet drop occurred quite at the end of the transfer and therefore the connections did not have time to get separated much.

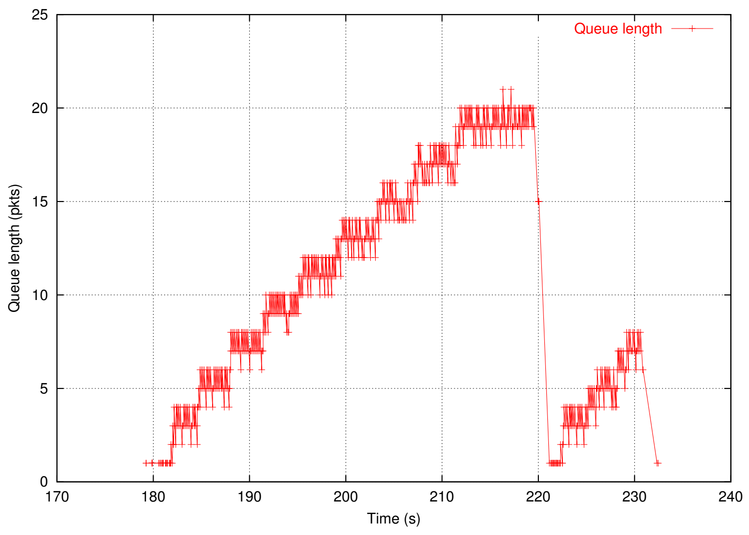

Figure 10: Queue length of a router when a sender uses CBI option. The replication started at about time 180 seconds.

### CBI and ECN

ECN and CBI combination with this type of workload and these link layer characteristics yielded a median of overall elapsed time that was 7 seconds longer than with pure CBI. Fairness was not as good as with pure CBI.

The reason for the poor behavior of the ECN option is the significant slowdown caused by the ECN marks. There was a median of seven marks for the faster connection and median of 10 marks for the slower. If the connections had been longer, the combination of ECN and CBI could have managed better compared to pure CBI because there possibly would have been more congestion related packet drops with the pure CBI and the combination could have kept the queue length low so that almost all congestion related drops could have been avoided.

### 5.1.2 Two TCP Connections, 5 Seconds Between Starts

Summary of the results is shown in Figure 11 and the test results are listed in Table 8. In this test the Eifel acted as the baseline TCP and other enhancements are compared to it.

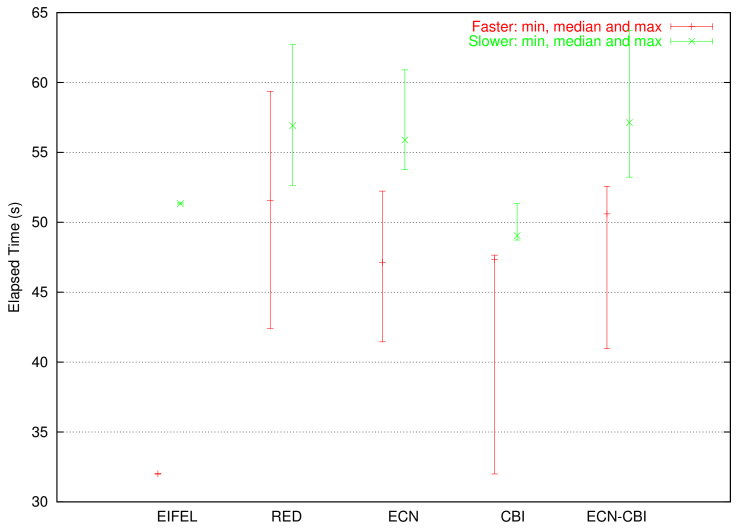

Figure 11: *Optimal link: 2tcp0udp5s*: Elapsed times of different enhancements

| TCP | ctime (s) | | | ctime (s) | | | Drops (50%) | Drops (50%) | otime (s) |
|---|---|---|---|---|---|---|---|---|---|
| | First | | | Second | | | First | Second | |
| variant | 25% | 50% | 75% | 25% | 50% | 75% | q / r / e | q / r / e | 50% |
| EIFEL | 32.0 | 32.0 | 32.0 | 51.3 | 51.3 | 51.4 | 18 / 0 / 0 | 2 / 0 / 0 | 41.7 |
| RED | 44.5 | 51.6 | 55.0 | 56.3 | 56.9 | 57.3 | 17 / 7 / 0 | 2 / 4 / 0 | 55.6 |
| ECN | 46.0 | 47.1 | 51.3 | 55.8 | 55.9 | 56.1 | 18 / 7 / 0 | 2 / 4 / 0 | 53.0 |
| CBI | 43.1 | 47.3 | 47.6 | 48.7 | 49.0 | 49.1 | 0 / 0 / 0 | 0 / 0 / 0 | 48.2 |
| CBI-ECN | 49.0 | 50.6 | 50.7 | 54.0 | 57.1 | 59.6 | 0 / 5 / 0 | 0 / 4 / 0 | 52.9 |

Table 8: *Optimal link: 2tcp0udp5s*: Summary of the results

Main problem was the lock-out of the second connection. With this link bandwidth and router queue size, and a 5 second delay between the starts of the first and second connections, the second connection usually suffered a lot. This is because the first connection was usually quite at the end of the slow start phase and the queue was starting to get full. This led to high queuing delays and higher RTT and therefore slower increase of the sending rate of the second connection. Queue overflow drops were also likely to occur to the later connection even if it had only a few packets in the queue. Because of the high queuing delays and the packet losses, the second connection was efficiently locked out by the first connection. Because of the lock-out the second connection was often able to transfer only a small fraction of its data before the first connection had finished.

After the first connection finished, the other connection had the whole bandwidth to use and in the beginning, when the second connection had not yet started, the first connection was in slow start and did not yet use the whole bandwidth. Without this timing issue, the fairness would be slightly worse when simply comparing the elapsed times.

The ECN and RED enhancements did not help in the beginning of the test but about the time the first connection had recovered from the slow start overshoot the RED algorithm started to work better and the latter connection got more bandwidth. The CBI option gave a significant improvement with this workload since it prevented the slow start overshoot of the first connection and the later connection did therefore experience only a very few congestion related packet drops and low queuing delays.

**Eifel**

The first connection experienced slow start overshoot and recovered from it efficiently. The second connection suffered a lot at the beginning and was not able to start efficiently transferring data until the first connection was finished. At this point the second connection had transferred only about 30 kb. There was no variance in the results as there were no error related packet losses or any other phenomena that could have brought randomness to the results and therefore every replication behaved very similarly. The difference between the minimum and the maximum elapsed time was less than 0.1 seconds in both connection sets.

The fairness was very poor because the first connection locked out the second very efficiently. The stability was also excellent because of the lock-out. The first connection was about 60% faster than the second in every replication.

**RED**

With RED the later connection still suffered like it did without RED since in the beginning the average queue length of the RED algorithm was low and it did not affect the slow start overshoot. Therefore the later connection usually experienced some congestion drops in a very early state and also sometimes the RED algorithm dropped a retransmitted packet at about the time the first connection was recovered from the slow start overshoot. Because of these packet drops and high queuing delays the start of the later connection was very slow but after the first connection recovered from the slow start overshoot the RED algorithm started to work better and also the latter connection got more bandwidth.

The main reason for the lower throughput compared to Eifel were additional RED drops, which there were a median of 8 for the first and 4 for the second connection set.

Overall elapsed time was much worse with RED than without it but the median of latter connection set was only 5 seconds slower while the median of first connection set was almost 20 seconds slower and therefore fairness was quite good compared to Eifel. The stability was worse than with the Eifel, although it was still very good with the latter connection.

### ECN

Overall elapsed time was few seconds better than with RED. The stability was slightly better with ECN. Compared to pure RED, the median of first connection set was three seconds faster and the latter was only one second faster and therefore the fairness was a bit worse than with pure RED. There were fewer marks for the first connection and more marks for the latter connection compared to pure RED, which could explain the worse fairness. Still the fairness was much better than with the Eifel.

Again, the slow start phase of the latter connection was very inefficient due to congestion related packet drops, RED drops of retransmitted packets, and high queuing delays.

### CBI

In the first replication the TCP sender had no knowledge of the previous connections to that specific host and therefore the connections behaved like without the CBI option. In this case the first connection was very fast (32 seconds) and the events were similar to Eifel.

During the rest of the replications there was no slow start overshoot. The first connection transferred about 37 kb in seven seconds and moved from slow start to congestion avoidance. Therefore the queue length stayed low at the beginning of the basic test and the second connection got started more efficiently.

Fairness was extremely good, the first connection was only slightly faster than the second, but because the second connection had the whole bandwidth to use at the end of the test and the first connection in the beginning of the test the elapsed times can not be compared directly.

#### CBI and ECN

Fairness was very good with pure CBI and therefore ECN markings were too rough to improve it anymore. The later connection was proceeding slightly slower than the first connection. When ECN started to mark packets in the second half of the transfer, both connections slowed a bit and the throughput started to variate according to ECN marks. Situation was the same as with the previous workload where the two TCP connections started simultaneously. There was a median of five ECN marks for the first connection set and seven for the second connection set which resulted to slower throughput than with pure CBI.

### 5.1.3 Two TCP Connections, One UDP Flow

Summary of the results with different options concerning TCP is listed in Table 9. Summary of UDP packet drops is in Table 10 and summary of UDP throughputs and jitter is in Table 11.

Although the slow start overshoot was the main reason for poor fairness it was not as severe with this workload as it was with the first workload since the UDP flow also filled the input queue of the router and neither of the TCP connections could increase `cwnd` to a high value before the slow start overshoot occurred. In this test the Eifel test acted as the baseline TCP. Although with Eifel the faster connection the fastest, it had the worst fairness and each enhancement improved it. With the exception of the faster connection of the Eifel test, the stability was relatively good, especially with the slower connections. With this workload the combination of ECN and CBI worked well as can be seen in Figure 12.

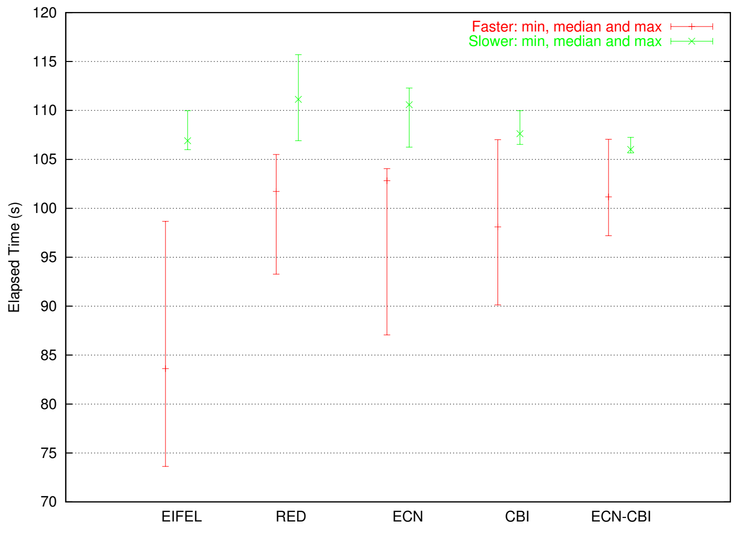

Figure 12: *Optimal link: 2tcp1udp*: Elapsed times of different enhancements

| TCP variant | ctime (s) Faster 25% | 50% | 75% | ctime (s) Slower 25% | 50% | 75% | Drops (50%) Faster q / r / e | Drops (50%) Slower q / r / e | otime (s) 50% |
|---|---|---|---|---|---|---|---|---|---|
| EIFEL | 74.2 | 83.6 | 90.5 | 106.4 | 106.9 | 107.4 | 10 / 0 / 0 | 14 / 0 / 0 | 102.3 |
| RED | 95.8 | 101.7 | 104.3 | 109.9 | 111.1 | 112.6 | 8 / 11 / 0 | 8 / 14 / 0 | 106.2 |
| ECN | 101.9 | 102.8 | 103.6 | 109.8 | 110.6 | 110.7 | 8 / 12 / 0 | 9 / 16 / 0 | 105.2 |
| CBI | 98.1 | 98.1 | 101.9 | 107.5 | 107.6 | 108.0 | 4 / 0 / 0 | 5 / 0 / 0 | 106.8 |
| CBI-ECN | 99.5 | 101.2 | 103.1 | 105.8 | 106.0 | 106.1 | 0 / 18 / 0 | 0 / 16 / 0 | 105.8 |

Table 9: *Optimal link: 2tcp1udp*: Summary of the results

| TCP variant | Queue Drops (pkts) Min/50%/Max | Error Drops (pkts) Min/50%/Max | RED Drops (pkts) Min/50%/Max |
|---|---|---|---|
| EIFEL | 12 / 17 / 25 | 0 / 0 / 0 | - / - / - |
| RED | 3 / 6 / 10 | 0 / 0 / 0 | 28 / 41 / 47 |
| ECN | 4 / 9 / 9 | 0 / 0 / 0 | 29 / 39 / 70 |
| CBI | 5 / 11 / 20 | 0 / 0 / 0 | - / - / - |
| CBI-ECN | 0 / 1 / 6 | 0 / 0 / 0 | 38 / 47 /101 |

Table 10: *Optimal link: 2tcp1udp*: Drop statistics of UDP flows

**Eifel**

The slow start overshoot was not as severe as it was without the UDP flow because the router queue was more utilized and a single TCP connection did not have as

| TCP variant | Throughput (b/s) Min/50%/Max | Spacing (ms) 50%/75%/90%/max |
|---|---|---|
| EIFEL | 3993 / 4013 / 4031 | 127/130/211/427-427 |
| RED | 3918 / 3927 / 3976 | 127/138/211/427-427 |
| ECN | 3840 / 3933 / 3970 | 127/138/211/427-460 |
| CBI | 4016 / 4034 / 4051 | 127/138/211/427-655 |
| CBI-ECN | 3765 / 3933 / 3959 | 127/138/211/427-655 |

Table 11: *Optimal link: 2tcp1udp*: Throughput and spacing statistics of UDP flows

many packets in it as it would have otherwise had. During the normal congestion avoidance after the slow start overshoot the connections experienced several congestion related packet drops which can be seen in Figure 13.

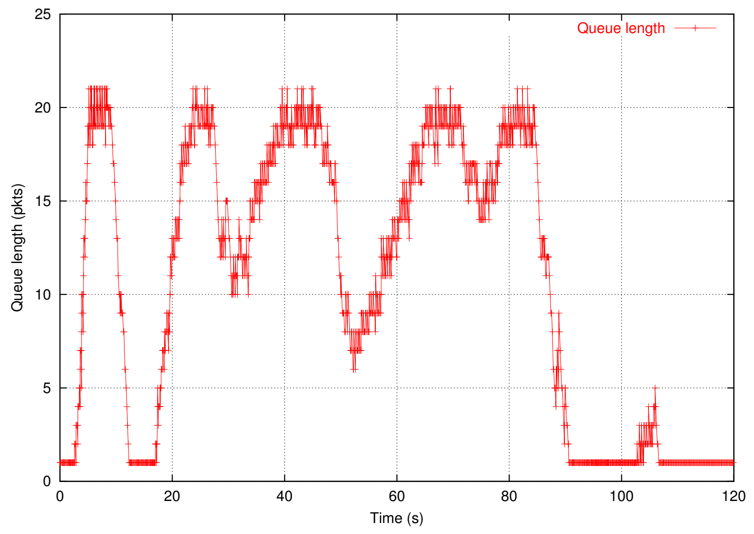

Figure 13: A queue of a router when there is a UDP flow in addition to two baseline TCP connections

Both TCP senders had transferred about 26 kb when the first sign of slow start overshoot was received by both senders. In many cases one of the connections recovered inefficiently from the slow start overshoot managing to send only one retransmission per RTT and suffering heavily because of it. Because of the inefficient overshoot recovery of one the TCP connections running concurrently the fairness was not good. Median of the elapsed time of the faster connection set was almost 30% faster than the slower connection set.

UDP flow lost almost a median of 20 packets because of router buffer overflows. Jitter of the UDP flow was good up to 75% percentile; the spacing of the UDP packets was roughly 130 ms. 90% percentile was 211 ms, so 10% of the UDP packets had waited about one additional transmission delay. Maximum spacing was 427 ms, which means that some UDP packets waited for about four additional transmission delays. These additional delays are because of queueing delays experienced in the router queue.

### RED

With the RED algorithm the median of the overall elapsed time was about seven seconds greater compared to Eifel. The RED algorithm succeeded to improve fairness compared to baseline TCP, the faster connection was only 9% faster than the slower connection.

The number of TCP buffer overflows was about one third lower than without RED, but on the other hand there was a median of 11 RED drops for the faster connection and 16 for the slower. This was the main reason for slower TCP throughput. The lower number of buffer overflows can be seen on Figure 14; the RED algorithm kept the queue nicely in proper limits. The queue size was 20 packets and it overflowed only in the slow start overshoot. Therefore there was always free buffer space and the slower connection was able to recover better and increase the transfer rate faster because of the low queuing delays.

UDP flow experienced smaller amount of congestion drops compared to the Eifel, but a much larger number of RED drops and therefore the throughput of the UDP flow was lower than without the RED router. Although the congestion was not as severe as it was without RED, the jitter was the same as with the Eifel.

### ECN

The use of ECN introduced a slight improvement compared to pure RED: median of the overall elapsed time was about one second smaller. The faster connection was almost 8% faster than the slower which is about the same as it was with pure RED. Utilization of the queue was very similar to a pure RED router.

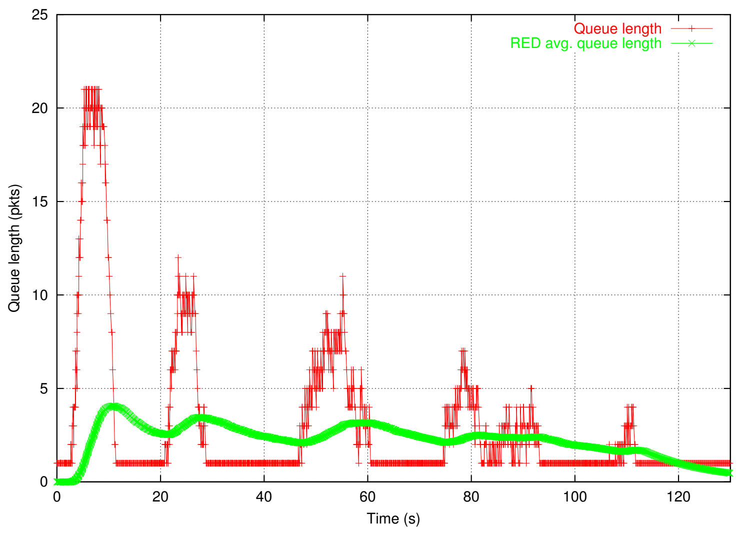

Figure 14: Queue of a RED router

In many cases RTO expired in the slow start overshoot recovery because a retransmitted packet got dropped by the RED router. There were many RTOs also because the fast recovery algorithm could not recover from the overshoot as there were not enough duplicate ACKs to decrease the number of packets outstanding in the network below the `cwnd` and thus allowing the sender to start retransmitting packets. In some cases the `cwnd` allowed a retransmission of a single packet which resulted a recovery of one packet per RTT.

The ECN capable router treated UDP packets like a pure RED router, since UDP does not support ECN. One UDP flow experienced 70 RED marks in the worst case, while the same number using a pure RED router was only 47. The rest of the UDP metrics were roughly the same between the pure RED router and the ECN capable RED router.

### CBI

Because of the CBI option, both TCP connections started equally, but because of competing UDP traffic there were more overflows and hence, more congestion related packet drops compared to same workload without UDP flow. These packet losses decreased fairness as one of the connections experienced packet losses and slowed down, while the other got more bandwidth. The faster connection was still only 10% faster than the slower but the difference would probably be larger with longer connections because of greater number of packet losses.

Only a median of 11 UDP packets were dropped. The throughput of the UDP flow was good, because there were fewer congestions related packet drops than without CBI.

#### CBI and ECN

The combination of CBI and ECN enhancements worked very well. Median of the overall elapsed time was 3 seconds worse than the with Eifel but the faster connection was only less 5% faster than the slower. CBI prevented the slow start overshoot and ECN kept the queue in efficiently in specified limits avoiding buffer overflows in most cases.

The UDP flow experienced very few congestion drops as the CBI and ECN options kept the queue utilization low but on the other hand, the UDP flow experienced a lot of RED marks which lowered the throughput significantly.

## 5.2 Lossy Link with Low ARQ Persistency

In this section we introduce the analyzed results with a lossy link. The link layer tried once to retransmit the lost packet which introduced a 700 ms delay and possible drop after the delay if the packet was lost again. The workload with two TCP connections starting simultaneously was run with every enhancement and the rest of the workloads were analyzed only with RED, ECN, and CBI enhancements and with the combination of CBI and ECN.

### 5.2.1 Two TCP Connections Starting at the Same Time

The graphical summary comparing the different enhancements is shown in Figure 15 and the summary of the results is listed in Table 12.

In this test the performance of baseline TCP was relatively good compared to the other enhancements, except RED and ECN, which decreased the elapsed times because of additional RED drops and ECN marks. The different RTO variants behaved roughly similarly as there were no spurious retransmissions. The fairness was not especially good with any of the analyzed enhancements. The RED algorithm was not efficient with this link since there were many error related packet losses.

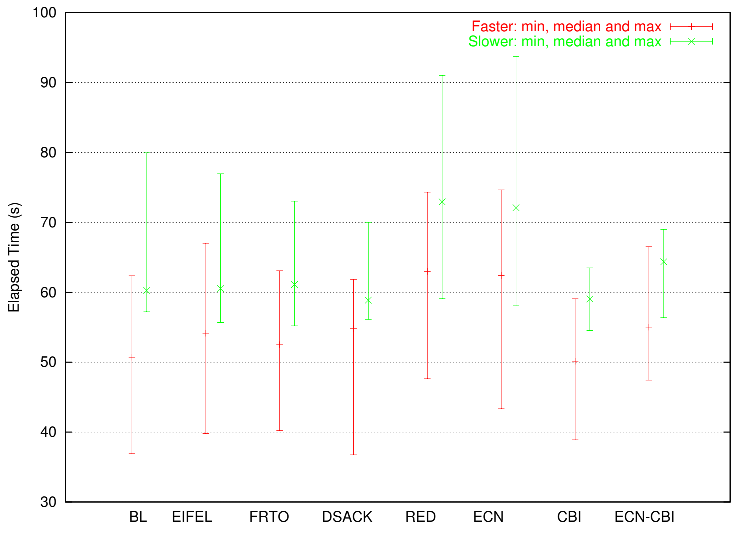

Figure 15: *e1 2tcp0udp*: Elapsed times of different enhancements

| TCP variant | ctime (s) Faster 25% | 50% | 75% | ctime (s) Slower 25% | 50% | 75% | Drops (50%) Faster q / r / e | Drops (50%) Slower q / r / e | otime (s) 50% |
|---|---|---|---|---|---|---|---|---|---|
| BL | 43.7 | 50.7 | 57.4 | 58.8 | 60.2 | 64.0 | 1 / 0 / 2 | 0 / 0 / 3 | 58.7 |
| EIFEL | 49.1 | 54.1 | 58.4 | 58.0 | 60.5 | 64.5 | 3 / 0 / 3 | 0 / 0 / 2 | 58.4 |
| FRTO | 47.5 | 52.5 | 56.6 | 58.1 | 61.1 | 64.5 | 11 / 0 / 1 | 12 / 0 / 2 | 57.5 |
| DSACK | 51.5 | 54.8 | 57.1 | 57.0 | 58.9 | 62.3 | 13 / 0 / 0 | 11 / 0 / 1 | 57.1 |
| RED | 59.7 | 63.0 | 63.7 | 67.2 | 73.0 | 75.0 | 8 / 3 / 1 | 1 / 3 / 2 | 66.6 |
| ECN | 56.9 | 62.4 | 67.7 | 68.2 | 72.1 | 75.0 | 10 / 2 / 0 | 12 / 4 / 4 | 68.2 |
| CBI | 46.4 | 50.1 | 52.5 | 56.4 | 59.0 | 60.6 | 0 / 0 / 0 | 0 / 0 / 3 | 55.6 |
| CBI-ECN | 51.3 | 55.0 | 58.0 | 60.5 | 64.3 | 66.6 | 0 / 0 / 1 | 0 / 0 / 3 | 59.9 |

Table 12: *Low ARQ link: 2tcp0udp*: Summary of the results

## Baseline TCP

Although the link layer hide some errors by retransmitting dropped packets there still were a median of four error related packet drops for the faster and five for the slower connection. These error related packet losses did not affect the performance a lot compared to baseline TCP with same workload on the optimal link test. Single packet losses were recovered efficiently using the recovery algorithms.

There were a total number of 16 RTOs, of which only a very few were spurious. Most of the RTOs occurred during the slow start overshoot recovery. These were recovered quickly because even if they introduced a few second additional delay the non-SACKed packets were retransmitted after the RTO in slow start and therefore sending two new retransmission per one ACK. In some cases in fast recovery the `cwnd` allowed packets to be sent and there were packets that could be sent and in these case Linux TCP marked the packet that triggered RTO only as lost and did not start the RTO algorithm. In very rare cases it took almost ten seconds before the RTO timer expired. The ACK for the packet that triggered the fast retransmit was received after the delay caused by the bad state. The ACK advanced the window and therefore the sender reseted the RTO timer but the `cwnd` still did not allow new packets to be sent and therefore the sender had to wait for the RTO to expire.

Errors and delays lowered the fairness between the connections since often the connections experienced a bad state differently and while the other connection recovered from a more severe consequences the other connection got most of the bandwidth. One significant factor was the different recovery from slow start overshoot. If one connection recovered faster then the slower connection had even more trouble recovering because of the lock-out. In some cases a bad state occurred while the connection was in slow start just before overshooting. Because of an error related packet drop the connection avoided the overshoot and changed to congestion avoidance.

### Eifel, F-RTO and D-SACK

There were many RTOs in the slow start overshoot recovery but spurious RTOs were very rare and therefore there were no larger differences between the baseline, Eifel, F-RTO, and D-SACK. F-RTO behaved mostly like the baseline TCP since most of the RTOs occurred during the fast recovery and there were still incoming dupacks.

### RED

The RED algorithm was very inefficient in this test case. There was a median of about four error related packet drops in both test cases, this and the baseline TCP. RED introduced additional four RED drops. Because there was already about the same amount of random packet drops the RED drops did not manage to level the differences of the connections; the difference of elapsed times of the faster and the slower connections grew from 16% to 19% compared to baseline. The overall elapsed time also grew almost eight seconds because of the additional drops.

### ECN

Situation with ECN was roughly the same as with pure RED. The algorithm worked inefficiently because of the error related drops, the ECN marks was no efficient enough to control the flows. Fairness was slightly worse compared to pure RED.

There were a few severe RTOs that decreased the performance significantly. In few cases the RED algorithm, bad state or queue overflow dropped a retransmission of the same packet twice in the slow start overshoot recovery. The result was two timeouts of which the latter was about eight seconds. Often one of the dropped retransmission was dropped by RED algorithm since the ECN capable flag can not be used in retransmissions.

### CBI

With the CBI option the slow start overshoot was avoided which was also the case with the optimal link. The medians of elapsed times of the both connections were good. One significant factor for the good throughput was the absence of the slow start overshoot. Another reason for the good throughput of the faster connection was the lower number of the error related packet drops compared to baseline, although the link parameters were the same.

Normally both connections proceeded equally until the first bad state. At that point one of the two connections often lost more packets than the other or one connection experienced only a delay and therefore the recovery took a different amount of time for connections. Therefore the fairness got worse and the faster connection was 18% faster than the slower connection.

#### CBI and ECN

The use of ECN option in addition to CBI did not introduce any additional gain compared to pure CBI, ECN marks only slowed down the connections. Because of the erroneous link the size of the average RED queue was low and the router marked only a median of one packet for the faster connection and a median of two packets for the slower connection.

### 5.2.2 Two TCP Connections, 5 Seconds Between Starts

A graphical comparison can be seen on Figure 16. Summary of the results with each enhancement is listed in Table 13. The Eifel acted as the baseline TCP in this test.

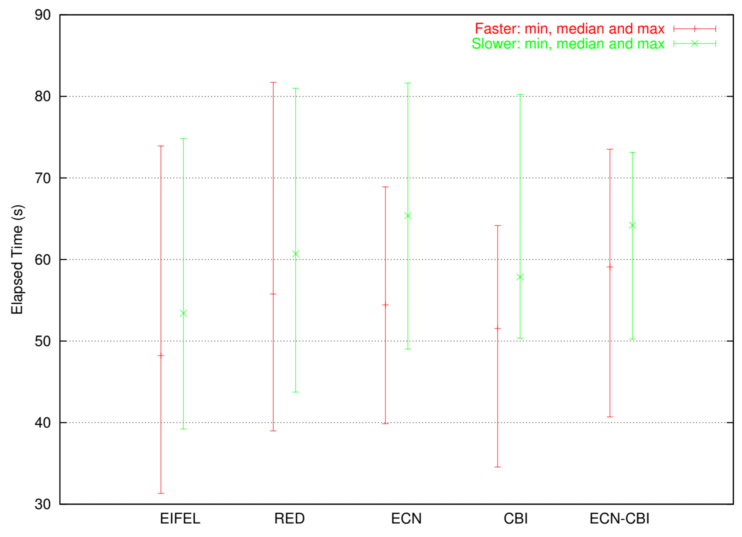

Figure 16: *Low ARQ link: 2tcp0udp5s*: Elapsed times of different enhancements

The Eifel had good elapsed times as did the CBI. The RED algorithm gave the best fairness but on the other hand the elapsed times were a bit higher compared to Eifel or CBI. The ECN enhancement introduced slightly better stability compared to RED but the fairness was worse.

| TCP variant | ctime (s) First 25% | ctime (s) First 50% | ctime (s) First 75% | ctime (s) Second 25% | ctime (s) Second 50% | ctime (s) Second 75% | Drops (50%) First q / r / e | Drops (50%) Second q / r / e | otime (s) 50% |
|---|---|---|---|---|---|---|---|---|---|
| EIFEL | 34.3 | 48.2 | 56.6 | 51.8 | 53.4 | 58.0 | 3 / 0 / 1 | 2 / 0 / 1 | 53.3 |
| RED | 49.4 | 55.8 | 64.6 | 55.9 | 60.7 | 65.4 | 3 / 3 / 2 | 2 / 1 / 2 | 59.6 |
| ECN | 45.7 | 54.4 | 57.0 | 57.3 | 65.4 | 67.3 | 3 / 3 / 1 | 2 / 1 / 2 | 57.3 |
| CBI | 45.2 | 51.5 | 57.4 | 52.0 | 57.9 | 61.1 | 0 / 0 / 1 | 0 / 0 / 2 | 53.4 |
| CBI-ECN | 47.6 | 59.1 | 63.2 | 55.0 | 64.2 | 67.8 | 0 / 0 / 3 | 0 / 0 / 3 | 60.8 |

Table 13: *Low ARQ link: 2tcp0udp5s*: Summary of the results

### Eifel

There were a median of two error related packet drops per connection. In the worst case, the connection experienced 16 error drops. Because of these packet drops the median of the elapsed time was lower than with the optimal link. There were total number of four RTOs over all 20 replications, none of which were spurious. Two of them were caused by a drop of retransmitted packet.

In few basic tests the bad state happened just before the slow start overshoot of the first connection. Therefore the first connection slowed down and avoided the overshoot. In these cases the second connection experienced a mild slow start overshoot from which it recovered efficiently.

The stability of the first connection was weak; the elapsed time varied from 34.3 seconds to 56.6 seconds. For the latter connection the stability was better. In many cases the latter connection had transferred only about one third of its data at the time the first connection had already finished but there were also a couple of cases in which the second connection was faster than the first connection.

### RED

The RED algorithm improved slighty the fairness compared to baseline TCP. When comparing the medians of the elapsed times, the first connection was only 9% shorter than the second when the difference was 11% with the Eifel. The RED algorithm dropped a median of five packets from the first connection and only a median of 2 packets from the second connection. Because of these additional RED drops the median of elapsed times over all connections grew over six seconds compared to Eifel.

### ECN

In this test case the router marked roughly the same amount of packets as it dropped with RED. The first connection experienced only 14 congestion related packet losses and with RED there was 17 packet drops and therefore the elapsed times with ECN were slightly smaller. The second connection experienced slightly more RED marks and error related packet drops and therefore the fairness was worse than with pure RED.

### CBI

Most of the connections did not experience a single congestion related drop. The CBI option prevented slow start overshoot after the first replication and the error drops kept the transfer rate at such a low rate that the router buffers did not overflow. Because of the earlier phase change to congestion avoidance and the better fairness the throughput of the first connection was lower compared to Eifel. The second connection was also slower compared to Eifel but the median of overall elapsed time was roughly the same.

### CBI and ECN

Again the combination of ECN and CBI enhancements did not introduce any gain compared to pure CBI, ECN marks only slowed down the connections. The elapsed times of the both connections grew remarkably although there were only a median of zero ECN marks for the first connection and a median of two marks for the latter connection. The number of error related drops were greater than with pure CBI. Fairness was about the same with pure CBI and with the combination; the faster connection was 9% faster than the slower.

### 5.2.3 Two TCP Connections, One UDP Flow

Summary of the results with each enhancement is listed in three tables. Table 14 shows the summary of numbers concerning TCP, Table 15 and Table 16 show the summary of UDP results. The Eifel acted as the baseline TCP.

As can be seen in the graphical summary in Figure 17 the best median elapsed time for the slower connection was with the Eifel. With the CBI option the elapsed times were good, but the fairness was not. The ECN option improved elapsed times compared to RED and fairness and stability were good. The elapsed times were low and the fairness was excellent. The behavior of the UDP flow was mainly dictated by RED drops and the link layer retransmission. The RED algorithm introduced a lower number of congestion drops but a much higher number of RED drops and the one link layer retransmission introduced an additional delay of about one RTT to some packets, because they had to wait for a link retransmission.

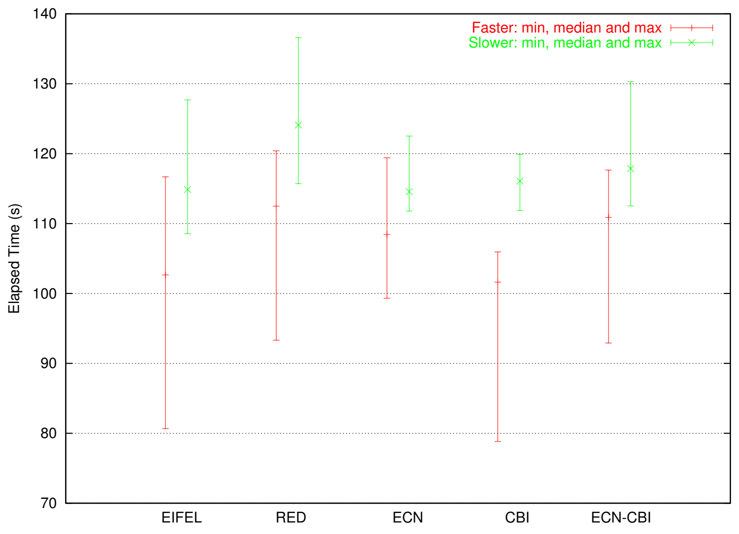

Figure 17: *Low ARQ link: 2tcp1udp*: Elapsed times of different enhancements

#### Eifel

A mild overshoot phenomenon occurred also elsewhere besides in the slow start phase. A few packets were dropped near to each other because of a buffer overflow. Many of the packet losses were recovered only by retransmitting one packet per

| TCP variant | ctime (s) Faster 25% | 50% | 75% | ctime (s) Slower 25% | 50% | 75% | Drops (50%) Faster q / r / e | Drops (50%) Slower q / r / e | otime (s) 50% |
|---|---|---|---|---|---|---|---|---|---|
| EIFEL | 92.5 | 102.7 | 106.4 | 112.0 | 114.9 | 117.5 | 8 / 0 / 2 | 6 / 0 / 2 | 111.7 |
| RED | 108.4 | 112.5 | 114.9 | 120.2 | 124.1 | 126.3 | 8 / 6 / 2 | 5 / 9 / 4 | 118.2 |
| ECN | 103.0 | 108.4 | 111.8 | 113.9 | 114.6 | 114.9 | 6 / 8 / 2 | 2 / 9 / 4 | 112.9 |
| CBI | 91.2 | 101.6 | 104.8 | 112.5 | 116.1 | 116.8 | 2 / 0 / 0 | 3 / 0 / 3 | 108.9 |
| CBI-ECN | 104.6 | 110.9 | 112.7 | 114.8 | 117.9 | 121.4 | 0 / 10 / 2 | 0 / 13 / 2 | 113.8 |

Table 14: *Low ARQ link: 2tcp1udp*: Summary of the results

| TCP variant | Queue Drops (pkts) Min/50%/Max | Error Drops (pkts) Min/50%/Max | RED Drops (pkts) Min/50%/Max |
|---|---|---|---|
| EIFEL | 5 / 19 / 30 | 8 / 14 / 23 | - / - / - |
| RED | 2 / 7 / 13 | 8 / 15 / 29 | 15 / 29 / 44 |
| ECN | 3 / 10 / 13 | 6 / 13 / 25 | 25 / 33 / 50 |
| CBI | 2 / 13 / 27 | 8 / 14 / 21 | - / - / - |
| CBI-ECN | 0 / 1 / 12 | 6 / 14 / 27 | 16 / 43 / 98 |

Table 15: *Low ARQ link: 2tcp1udp*: Drop statistics of UDP flows

| TCP variant | Throughput (b/s) Min/50%/Max | Spacing (ms) 50%/75%/90%/max |
|---|---|---|
| EIFEL | 3950 / 3961 / 4002 | 127/128/211/911-1133 |
| RED | 3863 / 3915 / 3964 | 127/139/211/916-1127 |
| ECN | 3846 / 3930 / 3933 | 127/128/211/911-1055 |
| CBI | 3940 / 3976 / 4025 | 127/128/211/839-1055 |
| CBI-ECN | 3817 / 3906 / 3976 | 127/139/211/911-1199 |

Table 16: *Low ARQ link: 2tcp1udp*: Throughput and spacing statistics of UDP flows

RTT because there were not enough dupacks for a more efficient recovery. There were total number of 18 RTOs, none of which were spurious.

The faster connection was about 12% faster than the slower connection when comparing the medians of the elapsed times which is roughly the same as the difference with the Eifel without the UDP flow.

UDP flows experienced about the same amount of congestion related packet drops as with the optimal link but there was a median of 14 error related packet drops, which lowered the throughput. Maximum jitter grew about one RTT compared to optimal link because of the one link layer retransmission. However 90% percentile of the jitter was still the same.

### RED

The RED algorithm kept the queue length below 10 packets most of the time. Without RED the queue overflowed several times in every replication. Therefore there were a median of three congestion related packet drops less in the faster connection and a median of four drops less in the slower connection than in the Eifel test. But because of the RED algorithm there were about a median of 10 RED drops for both connections and that lead to lower elapsed times. Therefore the median of the overall elapsed time grew about six seconds compared to Eifel. On the other hand RED managed to improve the fairness slightly.

The UDP flow experienced the same effect as the TCP connections. The number of congestion drops was much lower compared to Eifel but on the other hand the number of RED drops was significantly greater than the difference between the amounts of congestion related packet drops. Therefore the throughput decreased a bit. RED did not affect the jitter of the UDP flow, even though it kept the queue less utilized.

### ECN

With the ECN option the TCP connections were about as fast as with the Eifel, when comparing the overall elapsed time. There were more RED marks with ECN than pure RED and a slightly lesser amount of error related packet drops. ECN also achieved good fairness in this test; the faster connection was less than 6% faster than the slower connection. There were again few RTOs because the ECN option can not be used in retransmissions.

UDP flow experienced roughly the same amount of packet drops than with pure RED and the jitter was also roughly the same.

### CBI

With the CBI option the faster connection was very fast, over two seconds faster than with the Eifel. This is because with the CBI option the slow start overshoot was avoided. Another reason is that there was only a median of one error related packet drop for the faster connection and five for the slower. Fairness was poor, the faster connection was about 14% faster than the slower connection. Both connections proceeded equally in the beginning, then a bad state occurred and often one of the connections experienced a packet loss while the other connection experienced only a delay. After this event the connection that recovered first remained faster until the end of the test.

The UDP flow experienced a slightly lesser amount of congestion drops than the Eifel since there was no TCP slow start overshoot and therefore throughput was better with CBI than with Eifel. The CBI option did not affect the jitter of the UDP flow.

### CBI and ECN

There were median of zero congestion related drops mainly due to absence of slow start overshoot and the ECN option which kept the size of queue low. The overall elapsed time was high but the fairness was the best with this workload; the faster connection was less than 6% faster than the slower connection.

The UDP flow experienced a very low number of congestion related packet drops: a median of one. The UDP flow did, however, experience a very high amount of RED drops. The median was 43 RED drops and the maximum value for a single flow was 98 RED drops. Therefore the throughput was low compared to Eifel or pure CBI tests. Neither the ECN option nor the CBI option affected the jitter of the UDP flow.

## 5.3 Lossy Link with Medium ARQ Persistency

In this section we discuss the results run over medium ARQ persistency link. The link layer tries to retransmit a lost packet at most three times which is enough to hide all error related packet losses from upper protocol layers.

### Two TCP Connections Starting at the Same Time

The summary of the results is listed in Table 17. The CBI option introduced the best overall elapsed time. The next best results were given by the baseline TCP and the different RTO variants, but they all had poor fairness which, on the other hand, was very good with the CBI as can be seen in Figure 18. Using the ECN in addition to CBI gave good results also, but mostly because of the CBI option. The combination did, however, increase the stability compared to pure CBI. Pure RED slowed the connections down a lot and the fairness was not as good as it was with the CBI. RED with ECN introduced a few second improvement over pure RED.

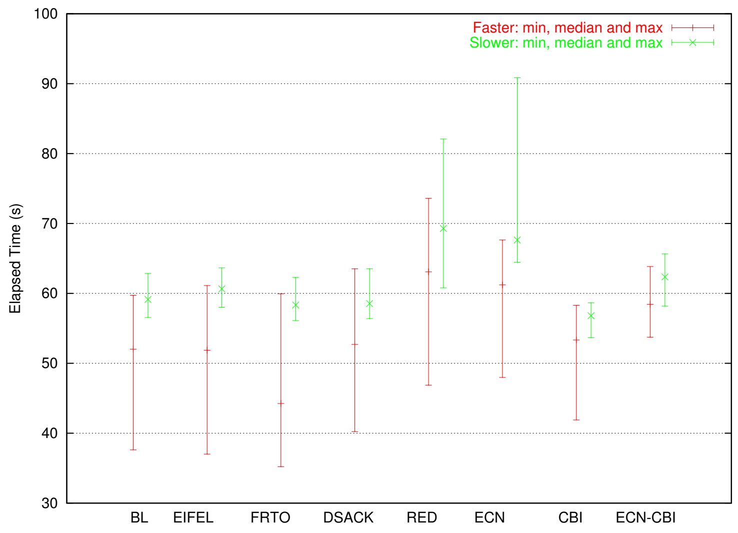

Figure 18: *Medium ARQ link: 2tcp0udp*: Elapsed times of different enhancements

### Baseline TCP

The stability for the faster connection was low and the elapsed times varied from 37.8 seconds to 59.7 seconds. For the slower connection the stability was much better as the range was from 56.5 to 62.9 seconds. Because the connections did not experience error related packet drops, the situation was similar to the baseline TCP with the optimal link.

| TCP variant | ctime (s) Faster 25% | 50% | 75% | ctime (s) Slower 25% | 50% | 75% | Drops (50%) Faster q / r / e | Drops (50%) Slower q / r / e | otime (s) 50% |
|---|---|---|---|---|---|---|---|---|---|
| BL | 42.1 | 52.0 | 54.2 | 58.2 | 59.1 | 59.6 | 12 / 0 / 0 | 13 / 0 / 0 | 57.4 |
| EIFEL | 43.8 | 51.9 | 55.9 | 59.1 | 60.6 | 61.5 | 11 / 0 / 0 | 13 / 0 / 0 | 58.3 |
| FRTO | 39.8 | 44.2 | 52.6 | 57.4 | 58.3 | 59.7 | 10 / 0 / 0 | 13 / 0 / 0 | 57.0 |
| DSACK | 47.5 | 52.7 | 55.5 | 57.5 | 58.5 | 59.4 | 11 / 0 / 0 | 12 / 0 / 0 | 57.6 |
| RED | 57.4 | 63.1 | 65.6 | 65.5 | 69.3 | 71.0 | 11 / 5 / 0 | 10 / 5 / 0 | 65.8 |
| ECN | 56.4 | 61.2 | 64.4 | 66.7 | 67.6 | 69.4 | 12 / 4 / 0 | 12 / 5 / 0 | 66.2 |
| CBI | 52.5 | 53.3 | 55.1 | 56.3 | 56.8 | 57.7 | 0 / 0 / 0 | 1 / 0 / 0 | 55.9 |
| CBI-ECN | 56.1 | 58.4 | 60.5 | 59.6 | 62.4 | 64.3 | 0 / 4 / 0 | 0 / 5 / 0 | 60.4 |

Table 17: *Medium ARQ link: 2tcp0udp*: Summary of the results

### Eifel, F-RTO and D-SACK

There were hardly any spurious RTOs and therefore the different RTO variants behaved similarly. The medians of overall elapsed times of other RTO variants were slightly better compared to Eifel because of the overhead caused by the timestamps option. F-RTO had a better median of the elapsed time of faster connection.

### RED

With a RED router there was smaller amount of congestion related packet drops as the RED algorithm kept the queue from overflowing but there was a median of six RED drops for a connection and therefore the elapsed times were much greater compared to baseline TCP. The RED algorithm improved the fairness; with the baseline TCP the median of the elapsed time of the faster connection set was 14% smaller than the same of the slower set and with RED the difference was only about 10%. RED decreased the stability especially for the slower connection.

There were RTOs because the RED algorithm dropped some retransmissions. In some cases there were two RTOs in the slow start overshoot recovery and also three RTOs were experienced in the overshoot recovery although this case was very rare. There were also cases where the RED router dropped the same packet twice, first the original transmission and then the retransmission of it.

### ECN

The use of ECN option in addition to a RED router introduced a few second improvement to medians of elapsed times. The fairness was about the same as with pure RED. The main reason for unfair connections was the different slow start overshoot recoveries of the connections. In many cases the slower connection recovered only by one retransmission per one RTT. Another reason for poor recovery were RTOs caused by RED drops on retransmitted segments since the ECN is not used in retransmissions.

### CBI

The CBI option introduced again very good results. The overall elapsed time was two seconds better than in the baseline and still the faster connection was only 6% faster than the slower connection. The stability was also good with CBI, especially for the slower connection. In most of the replications the connections proceeded equally from the beginning to the end, all bad states were experienced and recovered very similarly.

### CBI and ECN

Elapsed time got a bit longer compared to pure CBI because there was a median of five ECN marks for the faster connection and a median of six ECN marks for the slower connection. The fairness was about the same as it was with pure CBI. Median of the overall elapsed time was three seconds higher than with the baseline TCP and five seconds higher than with pure CBI but because of ECN the stability was good for both connections.

## 5.4 Lossy Link with High ARQ Persistency

In this section we discuss results of the high ARQ persistency link. The bad state was long but because the link layer tried to retransmit a lost packet up to six times, no error related packet losses were experienced by the upper protocol layers. Because of the long bad states there were often several spurious RTOs experienced by the

TCP connections. With this link only the workload with two TCP connections starting simultaneously was measured.

### Two TCP Connections Starting at the Same Time

The summary of the results is listed in Table 18. In this test case there were roughly from zero to five spurious RTOs experienced by a connection. The Eifel and F-RTO algorithms recovered efficiently from spurious RTOs.

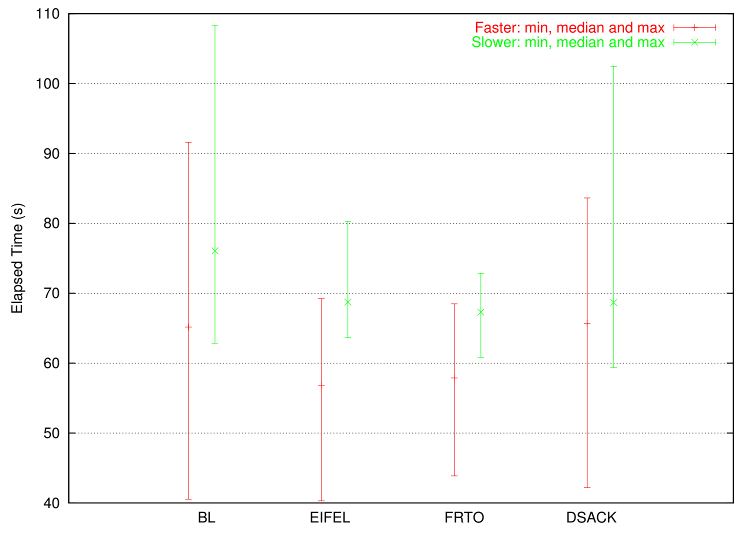

Figure 19: *High ARQ link: 2tcp0udp*: Elapsed times of different enhancements

| TCP | ctime (s) | | | ctime (s) | | | Drops (50%) | Drops (50%) | otime (s) |
|---|---|---|---|---|---|---|---|---|---|
| | Faster | | | Slower | | | Faster | Slower | |
| variant | 25% | 50% | 75% | 25% | 50% | 75% | q / r / e | q / r / e | 50% |
| BL | 57.2 | 65.2 | 72.1 | 69.5 | 76.1 | 82.5 | 9 / 0 / 0 | 8 / 0 / 0 | 71.5 |
| EIFEL | 49.8 | 56.8 | 64.1 | 65.8 | 68.7 | 71.8 | 10 / 0 / 0 | 13 / 0 / 0 | 65.6 |
| FRTO | 49.3 | 57.9 | 62.5 | 64.1 | 67.3 | 68.7 | 11 / 0 / 0 | 1 / 0 / 0 | 63.7 |
| DSACK | 53.9 | 65.7 | 71.8 | 67.0 | 68.7 | 75.0 | 12 / 0 / 0 | 12 / 0 / 0 | 67.3 |

Table 18: *High ARQ link: 2tcp0udp*: Summary of the results

### Baseline TCP

With the baseline TCP the sender retransmitted every packet in current TCP window after a spurious RTO even if they had all been received by the receiver. Because the receiver had all the packets when the retransmitted packets arrived it generated a dupack for every duplicate packet which, in turn, lead to unnecessary fast retransmit. When an RTO expired spuriously in a fast recovery, after slow start overshoot for example, no unnecessary fast retransmit was triggered since unnecessary retransmissions were avoided because of the SACK information.

The sender did often receive the ACKs for the original packets in bursts, because the link layer had buffered the packets while retransmitting some earlier lost packet. The sender then sent a new packet for every incoming ACK and therefore also the data packets were often sent in bursts. Linux TCP will send only three packets in one burst and wait 5 ms before sending again. Nevertheless this delay is very small compared to transmission delay of 72 ms for a packet of 576 bytes using a 64 kbit/s link and therefore in some cases this burstiness lead to queue overflow by several packets.

Because of the many RTOs the congestion window remained small and the connections were most of the time slowly increasing their speed and therefore there were a relatively small number of packet losses caused by an queue overflow. The stability was poor with both connections.

### Eifel

The fairness was very poor; median of the elapsed time of the faster connection set was 21% faster than the median of the slower set. This was mainly because often one of the connections recovered very poorly from the slow start overshoot. Because of the long bad states there were many spurious RTOs but they were recovered efficiently using the Eifel undo mechanism. There were no unnecessary retransmissions besides the packet that triggered the RTO.

#### F-RTO

The F-RTO variant behaved similarly to Eifel; the spurious RTOs were recovered efficiently. F-RTO improved the fairness only slightly but increased the stability of both connections significantly. Median of the overall elapsed time was two seconds better with F-RTO than with the Eifel and seven seconds better compared to baseline TCP.

#### D-SACK

The elapsed times were similar to the baseline TCP because the unnecessary retransmissions after a spurious RTO were also made with D-SACK and therefore the fast retransmit was also triggered. Although the sender undid the modifications to `ssthresh` and `cwnd` after the spurious RTO was recovered, the `ssthresh` and `cwnd` were decreased again because of the fast retransmit.

The fairness with D-SACK was the best among the enhancements with this workload and link combination. The faster connection was less than 4% faster than the slower connection. The stability was as low as it was with the baseline TCP.

## 5.5 Summary

In this section we summarize the results analyzed above. For TCP connections median, minimum and maximum elapsed time over all replications in each test case are drawn in the summary Figure 20. On the graph the enhancements are from left to right in the same order as they were discussed earlier in this section. In the graph marking `2T` means the workload with two TCP connections starting simultaneously, `2T5s` means the workload with two TCP connections starting with a five second difference and `2T1U` means the workload with two TCP connections starting simultaneously with one UDP flow.

With the baseline TCP and the Eifel the fairness among the two competing TCP connections was usually poor. If the connections started simultaneously on an optimal link and both recovered efficiently from the slow start overshoot, they proceeded at roughly the same speed until the end. However, in many cases one of

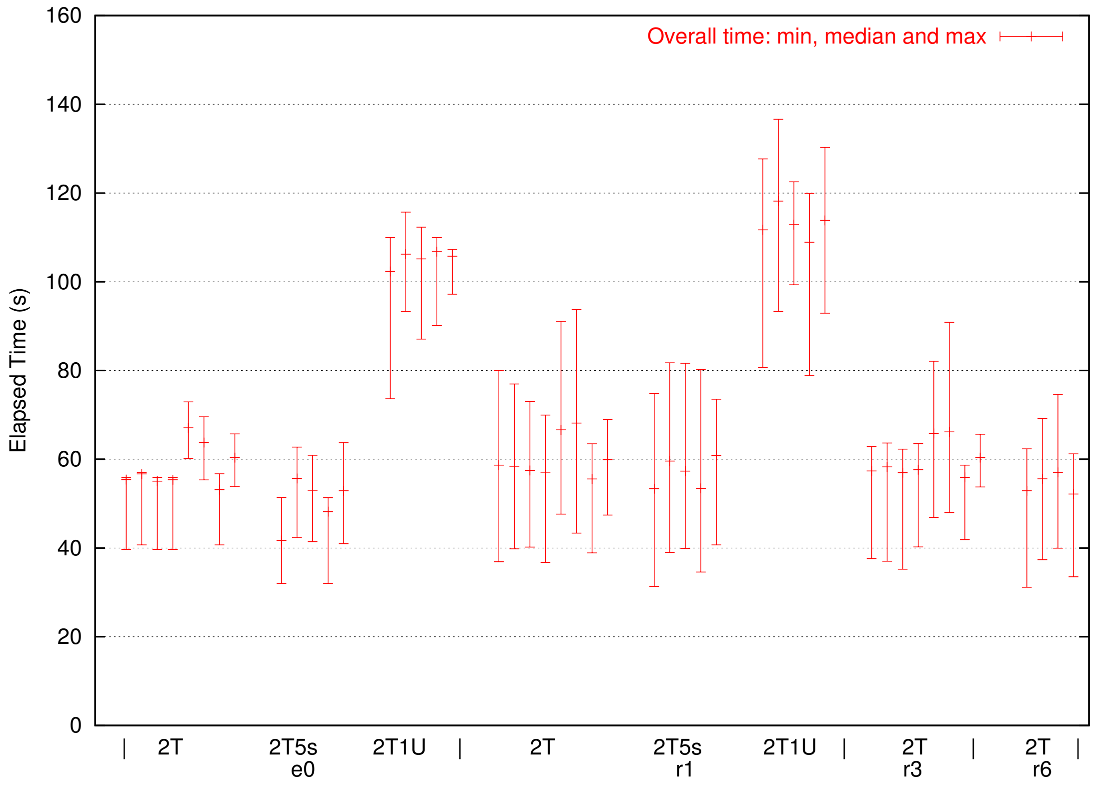

Figure 20: Overall elapsed times of all workloads, link environments and enhancements

the connections did not manage to recover efficiently from the slow start overshoot and transferred data very slowly for a relatively long period of time. Meanwhile the other connection that recovered efficiently from the overshoot was increasing its transfer rate. As it utilized more and more of the router queue, the queuing delays got longer and the recovery of the slower connection got even slower. Even after the recovery the slower connection still suffered from high queueing delays and managed to start transferring data at normal rate only after the faster connection was finished.

In most of the test cases where the latter connection started five seconds later than the first connection, the first connection efficiently locked the latter connection out. At the starting time of the latter connection, the first connection was usually at the end of the slow start overshoot and the latter connection experienced very high queuing delays and often congestion related packet drops as the faster connection had filled the queue. Even when the first connection had recovered from the overshoot the latter connection did not have much bandwidth. As a result it had usually transferred only a small fraction of its data by the time the first connection had finished. The situation was better with lossy link as the first connection had to slow down when it experienced a packet loss and often no lock-out situation occurred. The use of the RED algorithm also helped the situation after the first connection was recovered from the slow start overshoot. Before that the average queue was not high enough for the RED router to start marking packets. Because of the CBI option no slow start overshoot occurred after the first connection and the lock-out situation was efficiently avoided.

The CBI enhancement gave very good results in most link environments. It improved fairness among the connections and the overall elapsed time was often better with CBI than with the baseline TCP or with the Eifel. The better fairness was gained as the use of the CBI option avoided slow start overshoot and the queue size remained low in the beginning of the connection and therefore there were no congestion related packet losses and the queueing delays were low. Although the CBI enhancement gave good results with our workloads, it might not be the case generally. The connections behaved well in the beginning because of the saved `ssthresh` value, but the stability got worse toward the end of the test because of the few congestion related packet losses. With longer transfers the effect of the avoided slow start overshoot could be less significant. A greater number of concurrent data flows could also change the situation more against the good results of the CBI enhancement as there would be more congestion related packet losses.

In general the RED active queue management algorithm worked well. It usually lowered the throughput as it dropped a number of packets. Still the number of RED drops could probably not have been any smaller as the effectiveness of the RED algorithm would have diminished. The use of the ECN option in addition to the RED algorithm usually introduced a slight improvement because the retransmissions caused by the RED drops were avoided. However, the RED algorithm was not efficient with the lossy link as there were many error related packet losses and effect of RED drops was not significant enough to control the data flows. The RED algorithm could have worked better with more aggressive RED parameters, but then the throughput would probably have been even lower.

The different RTO variants, baseline TCP, Eifel, F-RTO and D-SACK, gave very similar results with the optimal link and the low and median ARQ persistency compared to baseline TCP as there were roughly any spurious RTOs. With the high ARQ persistent link and long bad states the difference was significant. The baseline TCP had no undo mechanism nor a way to distinguish between ACKs of the original packets and retransmitted packets and therefore the packets were retransmitted unnecessarily after a spurious RTO. This often led to unnecessary fast retransmit and therefore the sender slowed down the transfer rate twice. The situation was almost the same with D-SACK even though it has an undo mechanism. The packets were retransmitted unnecessarily and the fast retransmit was triggered also unnecessarily and the sender proceeded slowly in the congestion avoidance phase. Because of unnecessary retransmissions and fast retransmit these two variants gave poor results. The Eifel and The F-RTO algorithms on the other hand worked efficiently. The effect of a spurious RTO was relatively small; the RTO was detected as spurious, and the TCP variants continued sending new data. Although the F-RTO variant did not undo the modifications to `ssthresh` and `cwnd`, its results were roughly the same as with the Eifel.

With the presence of a RED router the UDP flow experienced a large number of RED drops and therefore the throughput of the UDP flow was lower than without the RED router even if the flow experienced a low number of congestion related drops. The jitter of the UDP flow was mainly dictated by the link layer retransmissions as they delayed packets for one additional RTT. Without link layer retransmissions the UDP flow experienced some additional delays caused by queuing delays as there sometimes were few packets in the queue before the UDP packet.

# 6 Conlusion

Our objective was to analyze the impact of concurrent TCP connections and streaming traffic to each other in a wireless environment. A TCP connection is affected by other traffic because it employs a congestion control and will slow down sending rate when congestion in the network increases. The streaming traffic in our measurements was a constant bit rate UDP flow which injected packets to the network at a steady rate. The UDP sender did not get any feedback from the network nor had implemented any congestion control mechanism therefore ignoring possible congestion. We measured workloads where two TCP connections started at the same and at different times. Third workload had an UDP flow in addition to two TCP connections. In this workload all data flows started at the same time.

The workloads were run over software emulated link and last-hop router using real end hosts. With the Seawind emulator we were able to control the characteristics of the link and we analyzed the workloads with an optimal link, with a link with errors, and with a link with long delays.

The main problem in our environment was the unfair use of the bandwidth between the two TCP connections and the long delays caused by link layer retransmissions. The slow start overshoot recovery was the main reason for the unfair use of the bandwidth and packet losses also decreased the fairness between the TCP connections. Spurious RTOs caused by long delays were efficiently recovered with F-RTO and Eifel algorithms. When UDP traffic was used, the UDP flow consumed half of the bandwidth, increasing congestion in the last-hop router. The most important metric for a UDP flow was jitter because interactive streaming traffics need a steady data flow.

We measured and analyzed the behavior of the baseline TCP, CBI enhancement, and the RED active queue management algorithm as well as different RTO recovery algorithms, Eifel, F-RTO and D-SACK. In addition to RED, ECN was also analyzed as well as a combination of ECN and CBI enhancements. The SACK enhancement was included already in the baseline TCP.

The CBI enhancement seemed to improve the results in every link environment because slow recovery from slow start overshoot was avoided. The RED algorithm generally improved the fairness among the TCP connections and ECN introduced a slight improvement over RED. Eifel and F-RTO detected efficiently spurious RTOs unlike the baseline TCP and D-SACK. The jitter of the UDP was slightly increased by queueing delays in congested network. Link layer retransmissions increased the jitter significantly as the last-hop router buffered packets while retransmitting a lost packet that should be delivered before the other ones.

In future we intend to analyze more complicated workloads; more concurrent TCP connections and UDP flows and also uplink traffic. The behavior of traffic flows would be interesting to analyze in presence of a DiffServ capable router.

## References

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

[AFP02] M. Allman, S. Floyd, and C. Partridge, Increasing TCP's Initial Window. *Request for Comments*, 3390, October 2002.

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

[BA03] E. Blanton and M. Allman, Using TCP DSACKs and SCTP Duplicate TSNs to Detect Spurious Retransmissions. Available at: `http://www.ietf.org/internet-drafts/` `draft-ietf-tsvwg-dsack-use-01.txt`, last checked 03.10.2003.

[BAF+03] E. Blanton, M. Allman, K. Fall, and L. Wang A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP. *Request for Comments*, 3517, April 2003

[BBC98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, An Architecture for Differentiated Service. *Request for Comments*, 2475, December 1998.

[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. *Request for Comments*, 2309, April 1998.

[BPS+96] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. *Proceedings of ACM SIGCOMM '96*, Standford, Ca, August 1996.

[Bra89] R. Braden, Requirements for Internet Hosts – Communication Layers. *Request for Comments*, 1122, October 1989

[BW98] G. Brasche and B. Walk, Concepts, Services and Protocols of the New GSM Phase 2+ General Packet Radio Service. *IEEE Communication magazine*, pages 94-104, August 1997.

[BZB+98] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. *Request for Comments*, 2205, September 1997.

[FH99] S. Floyd and T. Henderson, The NewReno Modification to TCP's Fast Recovery Algorithm. *Request for Comments*, 2582, April 1999.

[FJ93] S. Floyd and V. Jacobson, Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 397-413, August 1993.

[FMM00] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, An Extension to the Selective Acknowledgement (SACK) Option for TCP. *Request for Comments*, 2883, July 2000.

[Flo] S. Floyd, RED: Discussions of Setting Parameters. Available at: `http://http://www.icir.org/floyd/REDparameters.txt`, last checked 18.09.2003.

[FW02] G. Fairhurst and L. Wood, Advice to link designers on link Automatic Repeat reQuest (ARQ). *Request for Comments*, 3366, August 2002.

[H263] Video coding for low bit rate communication. *ITU-T Recommendations* H.263, March 1996.

[IIP] IIP Mixture Project, Patches to Linux kernels. Available at: `http://www.cs.helsinki.fi/research/iwtcp/kernel-patch/`, last checked 01.10.2003.

[IML+02] H. Inamura, G. Montenegro, R. Ludwig, A. Gurtov, and F. Khafizov, TCP over Second (2.5G) and Third (3G) Generation Wireless Networks. *Request for Comments*, 3481, February 2003.

[IP81] Internet Protocol. *Request for Comments*, 791, September 1981.

[Jac88] V. Jacobson, Congestion Avoidance and Control. *Computer Communication Review*, vol. 18, no. 4, pp. 314-329, August 1988.

[Jac90] V. Jacobson, Compressing TCP/IP Headers for Low-Speed Serial Links. *Request for Comments*, 1144, February 1990.

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

[KRL+97] M. Kojo, K. Raatikainen, M. Liljeberg, J. Kiiskinen, and T. Alanko, An Efficient Transport Service for Slow Wireless Telephone Links. *IEEE Journal on Selected Areas in Communications*, Vol. 15, No. 7, September 1997.

[LK00] R. Ludwig and R. H. Katz, The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions. *ACM Computer Communications Review*, 30(1), January 2000.

[Man] J. Manner, Jugi's Traffic Generator. Available at: `http://www.cs.helsinki.fi/u/jmanner/software/jtg/`, last checked 18.09.2003.

[MDK+00] G. Montenegro, S. Dawkins, M. Kojo, V. Magret, and N. Vaidya, Long Thin Networks. *Request for Comments*, 2757, January 2000.

[MMF+96] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, TCP Selective Acknowledgment Options. *Request for Comments*, 2018, October 1996.

[MM92] M. Mouly and M. Pautet, The GSM Systems for Mobile Communications. *Europe Media Duplication*, S.A., 1992.

[MPEG4] MPEG-4 Overview. *INTERNATIONAL ORGANISATION FOR STANDARDISATION*, ISO/IEC JTC1/SC29/WG11 N4668, March 2002.

[MSM+99] M. Mathis, J. Semke, J. Mahdavi and K. Lahey, The Rate-Halving Algorithm for TCP Congestion Control, June 1999. Available at: `http://www.psc.edu/networking/ftp/papers/draft-ratehalving.txt`, last checked 18.09.2003.

[Ost] S. Ostermann, Tcptrace, Available at: `http://www.tcptrace.org`, last checked 18.09.2003.

[PA00] V. Paxson and M. Allman, Computing TCP's Retransmission Timer. *Request for Comments*, 2988, Noveber 2000.

[PS98] S. Parker and C. Schmechel, Some Testing Tools for TCP Implementors *Request for Comments*, 2398, August 1998

[Pos80] J. Postel, User Datagram Protocol. *Request for Comments*, 768, August 1980.

[Pos81] J. Postel, Transmission Control Protocol. *Request for Comments*, 793, September 1981.

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

[SCF+96] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, RTP: A Transport Protocol for Real-Time Applications. *Request for Comments*, 1889, January 1996.

[SKR02] P. Sarolahti, M. Kojo, and K. Raatikainen, F-RTO: A New Recovery Algorithm for TCP Retransmission Timeouts. *Series of Publications C, No. C-2002-07*, University of Helsinki, Department of Computer Science, February 2002.

[SK02] P. Sarolahti and A. Kuznetsov, Congestion Control in Linux TCP. *In Proceedings of Usenix 2002/Freenix Track*, pp. 49–62. Monterey, CA, USA, June 2002.

[SRL98] H. Schulzrinne, A. Rao, and R. Lanphier, Real Time Streaming Protocol (RTSP). *Request for Comments*, 2326, April 1998.

[Tou97] J. Touch, TCP Control Block Interdependence. *Request for Comments*, 2140, April 1997.

[Wro98] J. Wroclawski, The Use of RSVP with IETF Integrated Services *Request for Comments*, 2210, September 1997.

# A Configuration Parameters

In this appendix we list the different parameters of Seawind, workload generator and the kernel that were used in our analysis.

## A.1 Seawind Parameters

In this section we list the Seawind parameters, that were used in the analysis. Table 19 holds the common parameters for all test cases. With the optimal case there were no errors nor state changes. Table 20 has the information conserning the two states with low ARQ persistency link. Tables 21 has the same information for medium ARQ link and Table 22 for the high ARQ link. Between the low and medium ARQ links the only difference is the lenght of `delay_drop_threshold` parameter. With high ARQ link the `state_duration` of the bad state and the `error_probability` is different.

| Parameter | Value | |
|---|---|---|
| edge | BOTH | |
| traffic_type | TUN_IP | |
| snaplen | 100b | |
| packet_logging_level | END_TO_END | |
| filter_level | BINARY | |
| rewind_user_distribution | 1 | |
| logging_level | NORMAL | |
| Parameter | DL Value | UL Value |
| link_receive_buffer_size | 9600b | 9600b |
| link_layer_ack_emulation | TRUE | TRUE |
| queue_drop_policy | Tail | Tail |
| reordering | FALSE | FALSE |
| link_send_buffer_size | 9600b | 9600b |
| propagation_delay | 300 | 300 |
| queue_max_length | 20 | 1 |
| queue_overflow_handling | DROP | DROP |
| rate_base | 64000bit/s | 64000bits/s |
| reassembly | FALSE | FALSE |
| user_max_rate | 1 | 1 |
| max_packet_size | 2048b | 2048b |

Table 19: Link parameters for all tests

| Good state | | |
|---|---|---|
| Parameter | Value | |
| state_duration | exponential, 15.0s, 1.0s, 20.0s | |
| | | |
| Bad state | | |
| Parameter | Value | |
| state_duration | uniform, 0.2s, 1.5s | |
| | | |
| Parameter | DL Value | UL Value |
| error_delay_function | static, 700ms | static, 700ms |
| error_rate_type | UNIT | UNIT |
| error_handling | DELAY_ITERATE | DELAY_ITERATE |
| error_probability | static, 0.63% | static, 0.63% |
| delay_drop_threshold | 701ms | 701ms |

Table 20: Low ARQ persistency link parameters

| Good state | | |
|---|---|---|
| Parameter | Value | |
| state_duration | exponential, 15.0s, 1.0s, 20.0 | |
| | | |
| Bad state | | |
| Parameter | Value | |
| state_duration | uniform, 0.2s, 1.5s | |
| | | |
| Parameter | DL Value | UL Value |
| error_delay_function | static, 700ms | static, 700ms |
| error_rate_type | UNIT | UNIT |
| error_handling | DELAY_ITERATE | DELAY_ITERATE |
| error_probability | static, 0.63% | static, 0.63% |
| delay_drop_threshold | 2101ms | 2101ms |

Table 21: Medium ARQ persistency link parameters

| Good state | | |
|---|---|---|
| Parameter | Value | |
| state_duration | exponential, 15.0s, 1.0s, 20.0s | |
| | | |
| Bad state | | |
| Parameter | Value | |
| state_duration | uniform, 0.4s, 4.0s | |
| | | |
| Parameter | DL Value | UL Value |
| error_delay_function | static, 700ms | static, 700ms |
| error_rate_type | UNIT | UNIT |
| error_handling | DELAY_ITERATE | DELAY_ITERATE |
| error_probability | static, 0.95% | static, 0.95% |
| delay_drop_threshold | 4201ms | 4201ms |

Table 22: High ARQ persistency link parameters

## A.2 Kernel Configuration

Table 23 contains kernel parameters provided by our patch and Table 24 contains the rest of the kernel parameter sused in our measurements. `0/1` means that the parameter was in some tests enabled and in some tests disabled. **`iip_rto_behaviour`** parameter value 1 means normal Linux RTO behavior and 3 enables F-RTO.

| Parameter | Value |
|---|---:|
| iip_cbi | 0/1 |
| iip_rto_behaviour | 1/3 |
| iip_iw | 2 |
| iip_delack_mode | 1 |
| iip_limitedxmit | 1 |
| iip_ratehalving | 0 |
| iip_srwnd_max | 16384 |
| iip_srwnd_min | 1024 |
| iip_srwnd_size | 16384 |
| iip_srwnd_addr | 0 |

Table 23: Kernel parameters

| Parameter | Value | Parameter | Value |
|---|---|---|---|
| tcp_dsack | 0/1 | tcp_ecn | 0/1 |
| tcp_timestamps | 0/1 | tcp_sack | 1 |
| tcp_fack | 0 | tcp_window_scaling | 0 |
| tcp_tw_reuse | 0 | tcp_adv_win_scale | 2 |
| tcp_app_win | 31 | tcp_rmem | 4096 87380 174760 |
| tcp_wmem | 4096 32768 131072 | tcp_mem | 23552 24064 24576 |
| tcp_reordering | 3 | tcp_orphan_retries | 0 |
| tcp_max_syn_backlog | 256 | 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 | 7200 | tcp_max_tw_buckets | 16384 |
| tcp_max_orphans | 8192 | tcp_synack_retries | 5 |
| tcp_syn_retries | 5 | tcp_retrans_collapse | 1 |
| icmp_ratemask | 6168 | icmp_ratelimit | 100 |
| icmp_ignore_bogus_error_responses | 0 | icmp_echo_ignore_broadcasts | 0 |
| icmp_echo_ignore_all | 0 | inet_peer_gc_maxtime | 120 |
| inet_peer_gc_mintime | 10 | inet_peer_maxttl | 600 |
| inet_peer_minttl | 120 | inet_peer_threshold | 65664 |
| ipfrag_low_thresh | 196608 | ipfrag_high_thresh | 262144 |
| ipfrag_time | 30 | ip_local_port_range | 1024 4999 |
| ip_dynaddr | 0 | ip_nonlocal_bind | 0 |
| ip_no_pmtu_disc | 0 | ip_autoconfig | 0 |
| ip_default_ttl | 64 | ip_forward | 0 |

Table 24: Kernel parameters

## A.3 Workload Generator Parameters

With JTG workload generator the parameters listed in Table 25 were used. With timestamps option the length of packets was 524 bytes and the number of packets was 360 and without timestamps the values was 536 and 352, respectively. The value `swd` means that Seawind assigned the port numbers. The `-A` option was used with UDP workloads. The middle section of the table describes the UDP only parameters and the last section describes the receiver only parameters.

| Parameter | Description | Value |
|---|---|---|
| -l## | length of packets written to network | 524/536 |
| -n## | number of bufs written to network | 360/352 |
| -p## | port number to send to/listen at | swd |
| -w | force the sender to use busy waiting | |
| -W | force the sender to use select (-w -W: use both based on HZ) | |
| -P | take into account processing latencies (send pckts more accurately) | |
| -Axxx | define another [addr:]port for communicating with the receiver | swd |
| -e## | delay in seconds before ending test (sender & receiver) | 10 |
| -Q | "be quiet" otherwise, but print the log of received packets on stdout | |
| -u | use UDP instead of TCP | subset |
| -d## | transmission time in seconds | 180 |
| -b## | Bandwidth for CBR in bits/s | 32000 |
| -r | receive | |
| -q | "be quiet!" | |

Table 25: JTG parameters

# B Statistics

There are listed in the tabels some metrics of the results. Those not discussed in section 4.6 are explained here as well as the other markings in the tables.

**TCP SET 1, CLI→SRV** is the first TCP connection set from the client (cli) to the server (srv) or from the server to the client (SRV→CLI). For the UDP there is only one set. The first set contains always the faster connection or the first connection in the workload case where the two TCP connection started at different time. The second TCP set contains the other connections.

**rexmt data pkts** is the number of retransmitted packets.

**duplicate acks** is the number of dupacks the sender has received and therefore the sent but dropped dupacks are not counted.

**triple dupacks** is the number of times the sender has received three consecutive dupacks and therefore the sent but dropped dupacks are not counted.

**pkts dropped** is the number of packets dropped by Seawind. These drops are listed separately by marking (q) for a queue drop, (r) for a RED drop, and (e) for an error drop. In the cases where the ECN option is used a RED drop means an ECN mark and also the actual RED drops with ECN are counted. If the "pkts dropped" metric is ommitted, then there were no packets dropped.

**sacks sent** is the number of SACK blocks received by the sender and therefore the sent but dropped SACK blocks are not counted.

## B.1 Optimal Link

### B.1.1 Two TCP Connections Starts at the Same Time

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 39.65 | 46.21 | 55.35 | 55.36 | 55.57 |
| throughput | 3395.00 | 3401.00 | 3408.00 | 4082.00 | 4758.00 |
| rexmt data pkts | 12.00 | 14.00 | 15.00 | 15.00 | 15.00 |
| duplicate acks | 22.00 | 27.00 | 41.00 | 41.00 | 43.00 |
| triple dupacks | 1.00 | 2.00 | 2.00 | 2.00 | 3.00 |
| pkts dropped (q) | 12.00 | 14.00 | 15.00 | 15.00 | 15.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 33.00 | 39.00 | 54.00 | 54.00 | 55.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 54.52 | 54.68 | 55.76 | 55.78 | 55.86 |
| throughput | 3377.00 | 3382.00 | 3383.00 | 3393.00 | 3460.00 |
| rexmt data pkts | 12.00 | 15.00 | 15.00 | 15.00 | 17.00 |
| duplicate acks | 28.00 | 36.00 | 42.00 | 42.00 | 44.00 |
| triple dupacks | 1.00 | 2.00 | 2.00 | 2.00 | 3.00 |
| pkts dropped (q) | 12.00 | 15.00 | 15.00 | 15.00 | 17.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 28.00 | 51.00 | 54.00 | 55.00 | 58.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 26: 2 TCP connections started simultaneously, no competing traffic. Baseline, no errors, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 40.68 | 40.75 | 56.62 | 56.63 | 56.68 |
| throughput | 3328.00 | 3331.00 | 3332.00 | 4630.00 | 4638.00 |
| rexmt data pkts | 14.00 | 14.00 | 15.00 | 15.00 | 15.00 |
| duplicate acks | 24.00 | 42.00 | 42.00 | 46.00 | 46.00 |
| triple dupacks | 1.00 | 2.00 | 2.00 | 2.00 | 2.00 |
| pkts dropped (q) | 14.00 | 14.00 | 15.00 | 15.00 | 15.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 35.00 | 54.00 | 54.00 | 58.00 | 58.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 55.79 | 56.76 | 56.89 | 56.91 | 56.95 |
| throughput | 3312.00 | 3315.00 | 3316.00 | 3323.00 | 3381.00 |
| rexmt data pkts | 12.00 | 15.00 | 15.00 | 17.00 | 17.00 |
| duplicate acks | 29.00 | 37.00 | 42.00 | 42.00 | 42.00 |
| triple dupacks | 1.00 | 2.00 | 2.00 | 2.00 | 2.00 |
| pkts dropped (q) | 12.00 | 15.00 | 15.00 | 17.00 | 17.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 28.00 | 51.00 | 54.00 | 54.00 | 54.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 27: 2 TCP connections started simultaneously, no competing traffic. Eifel, no errors, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 39.65 | 39.72 | 45.72 | 55.34 | 55.48 |
| throughput | 3400.00 | 3409.00 | 4097.00 | 4748.00 | 4757.00 |
| rexmt data pkts | 13.00 | 14.00 | 14.00 | 15.00 | 15.00 |
| duplicate acks | 22.00 | 24.00 | 41.00 | 43.00 | 45.00 |
| triple dupacks | 1.00 | 1.00 | 2.00 | 2.00 | 2.00 |
| pkts dropped (q) | 13.00 | 14.00 | 14.00 | 15.00 | 15.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 35.00 | 35.00 | 54.00 | 55.00 | 56.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 54.56 | 54.64 | 55.76 | 55.78 | 55.89 |
| throughput | 3375.00 | 3381.00 | 3383.00 | 3446.00 | 3458.00 |
| rexmt data pkts | 12.00 | 12.00 | 15.00 | 17.00 | 17.00 |
| duplicate acks | 28.00 | 29.00 | 36.00 | 42.00 | 44.00 |
| triple dupacks | 1.00 | 1.00 | 2.00 | 2.00 | 3.00 |
| pkts dropped (q) | 12.00 | 12.00 | 15.00 | 17.00 | 17.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 28.00 | 28.00 | 51.00 | 54.00 | 58.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 28: 2 TCP connections started simultaneously, no competing traffic. F-RTO, no errors, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 39.65 | 39.74 | 55.34 | 55.36 | 55.68 |
| throughput | 3388.00 | 3407.00 | 3408.00 | 4147.00 | 4758.00 |
| rexmt data pkts | 13.00 | 14.00 | 15.00 | 15.00 | 15.00 |
| duplicate acks | 22.00 | 41.00 | 41.00 | 41.00 | 45.00 |
| triple dupacks | 1.00 | 2.00 | 2.00 | 2.00 | 2.00 |
| pkts dropped (q) | 13.00 | 14.00 | 15.00 | 15.00 | 15.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 33.00 | 54.00 | 54.00 | 55.00 | 56.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 54.57 | 55.71 | 55.78 | 55.80 | 55.87 |
| throughput | 3376.00 | 3381.00 | 3382.00 | 3386.00 | 3457.00 |
| rexmt data pkts | 12.00 | 15.00 | 15.00 | 15.00 | 17.00 |
| duplicate acks | 28.00 | 36.00 | 42.00 | 42.00 | 42.00 |
| triple dupacks | 1.00 | 2.00 | 2.00 | 2.00 | 2.00 |
| pkts dropped (q) | 12.00 | 15.00 | 15.00 | 15.00 | 17.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 28.00 | 51.00 | 54.00 | 54.00 | 55.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 29: 2 TCP connections started simultaneously, no competing traffic. D-SACK, no errors, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 60.16 | 64.59 | 65.57 | 65.92 | 69.64 |
| throughput | 2709.00 | 2862.00 | 2877.00 | 2921.00 | 3136.00 |
| rexmt data pkts | 20.00 | 21.00 | 21.00 | 24.00 | 24.00 |
| duplicate acks | 51.00 | 52.00 | 57.00 | 63.00 | 63.00 |
| triple dupacks | 5.00 | 6.00 | 6.00 | 7.00 | 7.00 |
| pkts dropped (q) | 13.00 | 14.00 | 14.00 | 14.00 | 14.00 |
| pkts dropped (r) | 6.00 | 7.00 | 8.00 | 10.00 | 10.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 64.00 | 65.00 | 66.00 | 73.00 | 77.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 66.01 | 68.16 | 69.60 | 70.80 | 72.92 |
| throughput | 2587.00 | 2665.00 | 2710.00 | 2768.00 | 2858.00 |
| rexmt data pkts | 20.00 | 22.00 | 22.00 | 24.00 | 24.00 |
| duplicate acks | 47.00 | 60.00 | 63.00 | 67.00 | 74.00 |
| triple dupacks | 6.00 | 6.00 | 7.00 | 8.00 | 9.00 |
| pkts dropped (q) | 11.00 | 12.00 | 13.00 | 14.00 | 15.00 |
| pkts dropped (r) | 7.00 | 10.00 | 11.00 | 11.00 | 13.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 58.00 | 74.00 | 76.00 | 84.00 | 86.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 30: 2 TCP connections started simultaneously, no competing traffic. Eifel + RED, no errors, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 55.33 | 57.83 | 61.13 | 61.64 | 63.47 |
| throughput | 2972.00 | 3060.00 | 3086.00 | 3262.00 | 3410.00 |
| rexmt data pkts | 13.00 | 14.00 | 15.00 | 15.00 | 15.00 |
| duplicate acks | 23.00 | 30.00 | 31.00 | 33.00 | 34.00 |
| triple dupacks | 1.00 | 2.00 | 2.00 | 2.00 | 2.00 |
| pkts dropped (q) | 13.00 | 13.00 | 13.00 | 13.00 | 14.00 |
| pkts dropped (r) | 6.00 | 6.00 | 8.00 | 8.00 | 9.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 31.00 | 34.00 | 36.00 | 36.00 | 38.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 64.08 | 64.38 | 64.47 | 65.53 | 69.56 |
| throughput | 2712.00 | 2879.00 | 2926.00 | 2930.00 | 2944.00 |
| rexmt data pkts | 14.00 | 15.00 | 16.00 | 17.00 | 17.00 |
| duplicate acks | 20.00 | 20.00 | 22.00 | 46.00 | 63.00 |
| triple dupacks | 1.00 | 1.00 | 1.00 | 2.00 | 2.00 |
| pkts dropped (q) | 14.00 | 15.00 | 15.00 | 15.00 | 15.00 |
| pkts dropped (r) | 4.00 | 6.00 | 6.00 | 10.00 | 11.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 33.00 | 33.00 | 34.00 | 55.00 | 72.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 31: 2 TCP connections started simultaneously, no competing traffic. Eifel + ECN, no errors, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 40.68 | 49.80 | 49.80 | 53.06 | 53.08 |
| throughput | 3554.00 | 3555.00 | 3788.00 | 3788.00 | 4637.00 |
| rexmt data pkts | 0.00 | 0.00 | 0.00 | 1.00 | 14.00 |
| duplicate acks | 1.00 | 1.00 | 1.00 | 21.00 | 46.00 |
| triple dupacks | 0.00 | 0.00 | 0.00 | 1.00 | 2.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 1.00 | 14.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 0.00 | 0.00 | 0.00 | 20.00 | 58.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 53.20 | 53.20 | 53.25 | 53.26 | 56.73 |
| throughput | 3325.00 | 3542.00 | 3543.00 | 3546.00 | 3546.00 |
| rexmt data pkts | 1.00 | 1.00 | 2.00 | 2.00 | 17.00 |
| duplicate acks | 20.00 | 20.00 | 20.00 | 21.00 | 37.00 |
| triple dupacks | 1.00 | 1.00 | 1.00 | 1.00 | 2.00 |
| pkts dropped (q) | 1.00 | 1.00 | 2.00 | 2.00 | 17.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 20.00 | 21.00 | 21.00 | 21.00 | 51.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 32: 2 TCP connections started simultaneously, no competing traffic. Eifel + CBI, no errors, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 53.90 | 56.26 | 56.75 | 59.08 | 61.36 |
| throughput | 3075.00 | 3193.00 | 3324.00 | 3353.00 | 3500.00 |
| rexmt data pkts | 0.00 | 0.00 | 0.00 | 0.00 | 15.00 |
| duplicate acks | 1.00 | 1.00 | 1.00 | 1.00 | 28.00 |
| triple dupacks | 0.00 | 0.00 | 0.00 | 0.00 | 2.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 14.00 |
| pkts dropped (r) | 4.00 | 6.00 | 7.00 | 7.00 | 9.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 0.00 | 0.00 | 0.00 | 0.00 | 31.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 60.12 | 60.64 | 63.18 | 63.53 | 65.71 |
| throughput | 2871.00 | 2969.00 | 2986.00 | 3111.00 | 3138.00 |
| rexmt data pkts | 0.00 | 0.00 | 0.00 | 0.00 | 16.00 |
| duplicate acks | 1.00 | 1.00 | 1.00 | 1.00 | 48.00 |
| triple dupacks | 0.00 | 0.00 | 0.00 | 0.00 | 2.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 14.00 |
| pkts dropped (r) | 8.00 | 9.00 | 10.00 | 12.00 | 14.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 0.00 | 0.00 | 0.00 | 0.00 | 48.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 33: 2 TCP connections started simultaneously, no competing traffic. Eifel + CBI + ECN, no errors, queue 20 pkt.

### B.1.2 Two TCP connections, 5 Seconds Between Starts

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 31.99 | 32.00 | 32.01 | 32.02 | 32.03 |
| throughput | 5889.00 | 5891.00 | 5893.00 | 5896.00 | 5896.00 |
| rexmt data pkts | 18.00 | 18.00 | 18.00 | 18.00 | 18.00 |
| duplicate acks | 44.00 | 44.00 | 44.00 | 44.00 | 44.00 |
| triple dupacks | 2.00 | 2.00 | 2.00 | 2.00 | 2.00 |
| pkts dropped (q) | 18.00 | 18.00 | 18.00 | 18.00 | 18.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 60.00 | 60.00 | 60.00 | 60.00 | 60.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 51.31 | 51.32 | 51.34 | 51.35 | 51.39 |
| throughput | 3671.00 | 3674.00 | 3674.00 | 3676.00 | 3677.00 |
| rexmt data pkts | 2.00 | 2.00 | 2.00 | 2.00 | 2.00 |
| duplicate acks | 6.00 | 6.00 | 6.00 | 6.00 | 6.00 |
| triple dupacks | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| pkts dropped (q) | 2.00 | 2.00 | 2.00 | 2.00 | 2.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 6.00 | 6.00 | 6.00 | 6.00 | 6.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 34: 2 TCP connections, 5 second difference at start times, no competing traffic. Eifel, no errors, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 42.40 | 44.46 | 51.56 | 55.02 | 59.37 |
| throughput | 3177.00 | 3429.00 | 3659.00 | 4243.00 | 4449.00 |
| rexmt data pkts | 22.00 | 24.00 | 25.00 | 25.00 | 26.00 |
| duplicate acks | 61.00 | 74.00 | 82.00 | 87.00 | 91.00 |
| triple dupacks | 4.00 | 6.00 | 6.00 | 7.00 | 7.00 |
| pkts dropped (q) | 17.00 | 17.00 | 17.00 | 18.00 | 18.00 |
| pkts dropped (r) | 5.00 | 7.00 | 8.00 | 8.00 | 9.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 79.00 | 83.00 | 96.00 | 101.00 | 107.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 52.64 | 56.28 | 56.91 | 57.33 | 62.72 |
| throughput | 3008.00 | 3290.00 | 3315.00 | 3352.00 | 3584.00 |
| rexmt data pkts | 4.00 | 6.00 | 6.00 | 6.00 | 7.00 |
| duplicate acks | 16.00 | 25.00 | 29.00 | 31.00 | 38.00 |
| triple dupacks | 3.00 | 3.00 | 5.00 | 5.00 | 5.00 |
| pkts dropped (q) | 2.00 | 2.00 | 2.00 | 2.00 | 2.00 |
| pkts dropped (r) | 2.00 | 4.00 | 4.00 | 4.00 | 5.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 17.00 | 25.00 | 30.00 | 32.00 | 39.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 35: 2 TCP connections, 5 second difference at start times, no competing traffic. Eifel + RED, no errors, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 41.45 | 45.99 | 47.14 | 51.31 | 52.23 |
| throughput | 3611.00 | 3677.00 | 4002.00 | 4102.00 | 4551.00 |
| rexmt data pkts | 19.00 | 19.00 | 19.00 | 19.00 | 21.00 |
| duplicate acks | 52.00 | 57.00 | 59.00 | 59.00 | 60.00 |
| triple dupacks | 2.00 | 2.00 | 2.00 | 2.00 | 2.00 |
| pkts dropped (q) | 18.00 | 18.00 | 18.00 | 18.00 | 18.00 |
| pkts dropped (r) | 6.00 | 7.00 | 7.00 | 7.00 | 9.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 60.00 | 61.00 | 61.00 | 61.00 | 61.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 53.76 | 55.79 | 55.89 | 56.14 | 60.90 |
| throughput | 3098.00 | 3360.00 | 3376.00 | 3381.00 | 3509.00 |
| rexmt data pkts | 2.00 | 2.00 | 2.00 | 2.00 | 3.00 |
| duplicate acks | 6.00 | 6.00 | 6.00 | 6.00 | 6.00 |
| triple dupacks | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| pkts dropped (q) | 2.00 | 2.00 | 2.00 | 2.00 | 2.00 |
| pkts dropped (r) | 3.00 | 4.00 | 6.00 | 7.00 | 7.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 6.00 | 6.00 | 6.00 | 6.00 | 6.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 36: 2 TCP connections, 5 second difference at start times, no competing traffic. Eifel + ECN, no errors, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 32.00 | 43.08 | 47.33 | 47.61 | 47.65 |
| throughput | 3959.00 | 3962.00 | 3986.00 | 4379.00 | 5894.00 |
| rexmt data pkts | 0.00 | 0.00 | 1.00 | 2.00 | 18.00 |
| duplicate acks | 1.00 | 1.00 | 22.00 | 28.00 | 44.00 |
| triple dupacks | 0.00 | 0.00 | 1.00 | 2.00 | 2.00 |
| pkts dropped (q) | 0.00 | 0.00 | 1.00 | 2.00 | 18.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 0.00 | 0.00 | 21.00 | 28.00 | 60.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 48.73 | 48.74 | 49.04 | 49.09 | 51.34 |
| throughput | 3674.00 | 3843.00 | 3846.00 | 3871.00 | 3871.00 |
| rexmt data pkts | 0.00 | 0.00 | 0.00 | 1.00 | 2.00 |
| duplicate acks | 1.00 | 1.00 | 1.00 | 19.00 | 19.00 |
| triple dupacks | 0.00 | 0.00 | 0.00 | 1.00 | 1.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 1.00 | 2.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 0.00 | 0.00 | 0.00 | 18.00 | 18.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 37: 2 TCP connections, 5 second difference at start times, no competing traffic. Eifel + CBI, no errors, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 40.97 | 49.00 | 50.60 | 50.73 | 52.57 |
| throughput | 3588.00 | 3718.00 | 3728.00 | 3850.00 | 4604.00 |
| rexmt data pkts | 0.00 | 0.00 | 0.00 | 0.00 | 19.00 |
| duplicate acks | 1.00 | 1.00 | 1.00 | 1.00 | 58.00 |
| triple dupacks | 0.00 | 0.00 | 0.00 | 0.00 | 2.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 18.00 |
| pkts dropped (r) | 4.00 | 5.00 | 5.00 | 6.00 | 7.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 0.00 | 0.00 | 0.00 | 0.00 | 61.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 53.23 | 53.96 | 57.13 | 59.62 | 63.72 |
| throughput | 2961.00 | 3164.00 | 3302.00 | 3496.00 | 3544.00 |
| rexmt data pkts | 0.00 | 0.00 | 0.00 | 0.00 | 2.00 |
| duplicate acks | 1.00 | 1.00 | 1.00 | 1.00 | 6.00 |
| triple dupacks | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 2.00 |
| pkts dropped (r) | 3.00 | 4.00 | 7.00 | 8.00 | 9.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 0.00 | 0.00 | 0.00 | 0.00 | 6.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 38: 2 TCP connections, 5 second difference at start times, no competing traffic. Eifel + CBI + ECN, no errors, queue 20 pkt.

### B.1.3 Two TCP Connections, One UDP Flow

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 73.62 | 74.24 | 82.50 | 90.52 | 98.67 |
| throughput | 1912.00 | 2084.00 | 2226.00 | 2541.00 | 2562.00 |
| rexmt data pkts | 10.00 | 10.00 | 12.00 | 13.00 | 14.00 |
| duplicate acks | 34.00 | 42.00 | 47.00 | 55.00 | 60.00 |
| triple dupacks | 3.00 | 3.00 | 3.00 | 4.00 | 5.00 |
| pkts dropped (q) | 10.00 | 10.00 | 12.00 | 13.00 | 14.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 40.00 | 49.00 | 55.00 | 64.00 | 67.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 105.99 | 106.40 | 106.61 | 107.38 | 109.97 |
| throughput | 1715.00 | 1757.00 | 1759.00 | 1773.00 | 1780.00 |
| rexmt data pkts | 13.00 | 14.00 | 14.00 | 15.00 | 17.00 |
| duplicate acks | 32.00 | 35.00 | 37.00 | 42.00 | 58.00 |
| triple dupacks | 3.00 | 4.00 | 4.00 | 4.00 | 5.00 |
| pkts dropped (q) | 13.00 | 14.00 | 14.00 | 15.00 | 17.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 43.00 | 44.00 | 46.00 | 51.00 | 67.00 |
| UDP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 179.97 | 179.97 | 179.97 | 179.97 | 179.97 |
| throughput | 3993.00 | 4011.00 | 4013.00 | 4019.00 | 4031.00 |
| pkts dropped (q) | 12.00 | 16.00 | 17.00 | 19.00 | 25.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 39: 2 TCP connections started simultaneously, 1 competing 32 kbit/s UDP flow. Eifel, no errors, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 93.27 | 95.85 | 101.72 | 104.26 | 105.50 |
| throughput | 1788.00 | 1809.00 | 1855.00 | 1968.00 | 2023.00 |
| rexmt data pkts | 17.00 | 18.00 | 21.00 | 21.00 | 29.00 |
| duplicate acks | 63.00 | 65.00 | 69.00 | 76.00 | 81.00 |
| triple dupacks | 8.00 | 8.00 | 9.00 | 9.00 | 10.00 |
| pkts dropped (q) | 6.00 | 8.00 | 9.00 | 10.00 | 11.00 |
| pkts dropped (r) | 10.00 | 11.00 | 11.00 | 12.00 | 21.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 73.00 | 75.00 | 84.00 | 88.00 | 89.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 106.91 | 109.90 | 111.13 | 112.60 | 115.70 |
| throughput | 1630.00 | 1675.00 | 1697.00 | 1716.00 | 1764.00 |
| rexmt data pkts | 18.00 | 22.00 | 25.00 | 25.00 | 27.00 |
| duplicate acks | 53.00 | 55.00 | 67.00 | 73.00 | 91.00 |
| triple dupacks | 8.00 | 8.00 | 9.00 | 12.00 | 12.00 |
| pkts dropped (q) | 8.00 | 8.00 | 9.00 | 10.00 | 10.00 |
| pkts dropped (r) | 10.00 | 14.00 | 16.00 | 16.00 | 18.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 63.00 | 68.00 | 75.00 | 84.00 | 100.00 |
| UDP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 179.97 | 179.97 | 179.97 | 179.97 | 179.97 |
| throughput | 3918.00 | 3924.00 | 3927.00 | 3973.00 | 3976.00 |
| pkts dropped (q) | 3.00 | 3.00 | 6.00 | 10.00 | 10.00 |
| pkts dropped (r) | 28.00 | 32.00 | 41.00 | 44.00 | 47.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 40: 2 TCP connections started simultaneously, 1 competing 32 kbit/s UDP flow. Eifel + RED, no errors, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 87.06 | 101.93 | 102.82 | 103.63 | 104.05 |
| throughput | 1813.00 | 1820.00 | 1835.00 | 1851.00 | 2167.00 |
| rexmt data pkts | 8.00 | 9.00 | 9.00 | 10.00 | 10.00 |
| duplicate acks | 12.00 | 14.00 | 15.00 | 17.00 | 30.00 |
| triple dupacks | 1.00 | 1.00 | 1.00 | 1.00 | 2.00 |
| pkts dropped (q) | 7.00 | 8.00 | 9.00 | 10.00 | 10.00 |
| pkts dropped (r) | 10.00 | 12.00 | 15.00 | 16.00 | 33.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 20.00 | 21.00 | 23.00 | 23.00 | 31.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 106.25 | 109.77 | 110.58 | 110.65 | 112.29 |
| throughput | 1680.00 | 1705.00 | 1706.00 | 1719.00 | 1775.00 |
| rexmt data pkts | 10.00 | 11.00 | 11.00 | 11.00 | 12.00 |
| duplicate acks | 11.00 | 12.00 | 23.00 | 26.00 | 31.00 |
| triple dupacks | 1.00 | 1.00 | 2.00 | 2.00 | 2.00 |
| pkts dropped (q) | 9.00 | 9.00 | 10.00 | 10.00 | 11.00 |
| pkts dropped (r) | 12.00 | 16.00 | 16.00 | 17.00 | 26.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 20.00 | 20.00 | 24.00 | 29.00 | 37.00 |
| UDP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 179.97 | 179.97 | 179.97 | 179.97 | 179.97 |
| throughput | 3840.00 | 3924.00 | 3933.00 | 3953.00 | 3970.00 |
| pkts dropped (q) | 4.00 | 7.00 | 9.00 | 9.00 | 9.00 |
| pkts dropped (r) | 29.00 | 31.00 | 39.00 | 51.00 | 70.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 41: 2 TCP connections started simultaneously, 1 competing 32 kbit/s UDP flow. Eifel + ECN, no errors, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 90.13 | 98.09 | 98.10 | 101.86 | 107.01 |
| throughput | 1763.00 | 1852.00 | 1923.00 | 1923.00 | 2093.00 |
| rexmt data pkts | 3.00 | 4.00 | 4.00 | 4.00 | 13.00 |
| duplicate acks | 30.00 | 34.00 | 36.00 | 36.00 | 42.00 |
| triple dupacks | 3.00 | 3.00 | 3.00 | 3.00 | 4.00 |
| pkts dropped (q) | 3.00 | 4.00 | 4.00 | 4.00 | 13.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 30.00 | 34.00 | 36.00 | 36.00 | 45.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 106.53 | 107.54 | 107.61 | 108.05 | 109.97 |
| throughput | 1715.00 | 1746.00 | 1753.00 | 1754.00 | 1771.00 |
| rexmt data pkts | 5.00 | 5.00 | 5.00 | 7.00 | 15.00 |
| duplicate acks | 33.00 | 33.00 | 33.00 | 42.00 | 44.00 |
| triple dupacks | 3.00 | 3.00 | 3.00 | 4.00 | 4.00 |
| pkts dropped (q) | 5.00 | 5.00 | 5.00 | 7.00 | 15.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 34.00 | 34.00 | 34.00 | 44.00 | 50.00 |
| UDP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 179.97 | 179.97 | 179.97 | 179.97 | 179.97 |
| throughput | 4016.00 | 4028.00 | 4034.00 | 4034.00 | 4051.00 |
| pkts dropped (q) | 5.00 | 11.00 | 11.00 | 13.00 | 20.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 42: 2 TCP connections started simultaneously, 1 competing 32 kbit/s UDP flow. Eifel + CBI, no errors, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 97.20 | 99.51 | 101.17 | 103.08 | 107.06 |
| throughput | 1762.00 | 1830.00 | 1865.00 | 1896.00 | 1941.00 |
| rexmt data pkts | 0.00 | 0.00 | 0.00 | 0.00 | 10.00 |
| duplicate acks | 1.00 | 1.00 | 1.00 | 1.00 | 23.00 |
| triple dupacks | 0.00 | 0.00 | 0.00 | 0.00 | 2.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 9.00 |
| pkts dropped (r) | 11.00 | 18.00 | 23.00 | 32.00 | 51.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 0.00 | 0.00 | 0.00 | 0.00 | 25.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 105.66 | 105.84 | 106.01 | 106.06 | 107.25 |
| throughput | 1759.00 | 1779.00 | 1779.00 | 1782.00 | 1785.00 |
| rexmt data pkts | 0.00 | 0.00 | 0.00 | 5.00 | 12.00 |
| duplicate acks | 1.00 | 1.00 | 1.00 | 10.00 | 11.00 |
| triple dupacks | 0.00 | 0.00 | 0.00 | 1.00 | 1.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 3.00 | 11.00 |
| pkts dropped (r) | 14.00 | 16.00 | 19.00 | 32.00 | 38.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 0.00 | 0.00 | 0.00 | 11.00 | 20.00 |
| UDP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 179.97 | 179.97 | 179.97 | 179.97 | 179.97 |
| throughput | 3765.00 | 3820.00 | 3933.00 | 3956.00 | 3959.00 |
| pkts dropped (q) | 0.00 | 0.00 | 1.00 | 3.00 | 6.00 |
| pkts dropped (r) | 38.00 | 40.00 | 47.00 | 84.00 | 101.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 43: 2 TCP connections started simultaneously, 1 competing 32 kbit/s UDP flow. Eifel + CBI + ECN, no errors, queue 20 pkt.

## B.2 Low ARQ persistency

### B.2.1 Two TCP Connections Starts at the Same Time

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 36.90 | 43.74 | 50.67 | 57.36 | 62.36 |
| throughput | 3025.00 | 3272.00 | 3719.00 | 3942.00 | 5112.00 |
| rexmt data pkts | 6.00 | 8.00 | 15.00 | 17.00 | 22.00 |
| duplicate acks | 18.00 | 28.00 | 40.00 | 52.00 | 75.00 |
| triple dupacks | 1.00 | 2.00 | 3.00 | 3.00 | 4.00 |
| pkts dropped (q) | 0.00 | 1.00 | 13.00 | 14.00 | 19.00 |
| pkts dropped (e) | 0.00 | 2.00 | 4.00 | 5.00 | 12.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 27.00 | 34.00 | 50.00 | 66.00 | 90.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 1.00 | 2.00 | 3.00 | 6.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 57.20 | 58.77 | 59.98 | 64.01 | 79.95 |
| throughput | 2359.00 | 2945.00 | 3117.00 | 3203.00 | 3298.00 |
| rexmt data pkts | 4.00 | 7.00 | 16.00 | 19.00 | 21.00 |
| duplicate acks | 21.00 | 31.00 | 49.00 | 51.00 | 59.00 |
| triple dupacks | 2.00 | 3.00 | 3.00 | 4.00 | 5.00 |
| pkts dropped (q) | 0.00 | 0.00 | 11.00 | 14.00 | 17.00 |
| pkts dropped (e) | 1.00 | 3.00 | 5.00 | 6.00 | 9.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 27.00 | 41.00 | 50.00 | 66.00 | 76.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 1.00 | 1.00 | 2.00 | 4.00 |

Table 44: 2 TCP connections started simultaneously, no competing traffic. Baseline, low ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 39.80 | 49.14 | 54.01 | 58.39 | 67.01 |
| throughput | 2815.00 | 3226.00 | 3475.00 | 3776.00 | 4740.00 |
| rexmt data pkts | 3.00 | 8.00 | 14.00 | 17.00 | 22.00 |
| duplicate acks | 22.00 | 32.00 | 50.00 | 56.00 | 66.00 |
| triple dupacks | 1.00 | 2.00 | 3.00 | 3.00 | 5.00 |
| pkts dropped (q) | 0.00 | 3.00 | 11.00 | 14.00 | 17.00 |
| pkts dropped (e) | 0.00 | 3.00 | 4.00 | 6.00 | 12.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 29.00 | 36.00 | 51.00 | 72.00 | 79.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 1.00 | 2.00 | 4.00 | 5.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 55.68 | 58.04 | 60.15 | 64.47 | 76.95 |
| throughput | 2451.00 | 2897.00 | 3099.00 | 3222.00 | 3388.00 |
| rexmt data pkts | 3.00 | 7.00 | 15.00 | 17.00 | 25.00 |
| duplicate acks | 10.00 | 35.00 | 45.00 | 46.00 | 74.00 |
| triple dupacks | 1.00 | 3.00 | 3.00 | 3.00 | 5.00 |
| pkts dropped (q) | 0.00 | 0.00 | 9.00 | 14.00 | 18.00 |
| pkts dropped (e) | 0.00 | 2.00 | 5.00 | 6.00 | 10.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 10.00 | 37.00 | 50.00 | 58.00 | 84.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 1.00 | 2.00 | 3.00 | 7.00 |

Table 45: 2 TCP connections started simultaneously, no competing traffic. Eifel, low ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 40.21 | 47.46 | 52.47 | 56.59 | 63.08 |
| throughput | 2990.00 | 3333.00 | 3592.00 | 3964.00 | 4692.00 |
| rexmt data pkts | 5.00 | 11.00 | 14.00 | 19.00 | 22.00 |
| duplicate acks | 21.00 | 30.00 | 41.00 | 47.00 | 63.00 |
| triple dupacks | 1.00 | 2.00 | 2.00 | 3.00 | 5.00 |
| pkts dropped (q) | 0.00 | 11.00 | 13.00 | 13.00 | 15.00 |
| pkts dropped (e) | 0.00 | 1.00 | 3.00 | 5.00 | 9.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 26.00 | 36.00 | 51.00 | 61.00 | 76.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 1.00 | 1.00 | 2.00 | 4.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 55.20 | 58.12 | 60.37 | 64.55 | 73.04 |
| throughput | 2583.00 | 2862.00 | 3051.00 | 3239.00 | 3418.00 |
| rexmt data pkts | 4.00 | 15.00 | 18.00 | 19.00 | 23.00 |
| duplicate acks | 15.00 | 37.00 | 45.00 | 52.00 | 60.00 |
| triple dupacks | 2.00 | 2.00 | 3.00 | 3.00 | 6.00 |
| pkts dropped (q) | 0.00 | 12.00 | 12.00 | 14.00 | 16.00 |
| pkts dropped (e) | 1.00 | 2.00 | 4.00 | 6.00 | 10.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 20.00 | 48.00 | 56.00 | 62.00 | 76.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 1.00 | 3.00 | 3.00 |

Table 46: 2 TCP connections started simultaneously, no competing traffic. F-RTO, low ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 47.63 | 59.71 | 62.95 | 63.69 | 74.33 |
| throughput | 2538.00 | 2944.00 | 2993.00 | 3140.00 | 3961.00 |
| rexmt data pkts | 6.00 | 17.00 | 19.00 | 20.00 | 30.00 |
| duplicate acks | 50.00 | 56.00 | 59.00 | 68.00 | 106.00 |
| triple dupacks | 3.00 | 4.00 | 5.00 | 7.00 | 7.00 |
| pkts dropped (q) | 0.00 | 8.00 | 12.00 | 13.00 | 16.00 |
| pkts dropped (r) | 2.00 | 3.00 | 4.00 | 7.00 | 8.00 |
| pkts dropped (e) | 0.00 | 1.00 | 4.00 | 6.00 | 11.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 52.00 | 62.00 | 72.00 | 80.00 | 115.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 1.00 | 2.00 | 3.00 | 6.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 59.08 | 67.20 | 72.37 | 75.02 | 91.02 |
| throughput | 2073.00 | 2470.00 | 2565.00 | 2783.00 | 3193.00 |
| rexmt data pkts | 5.00 | 14.00 | 21.00 | 24.00 | 32.00 |
| duplicate acks | 32.00 | 45.00 | 55.00 | 66.00 | 76.00 |
| triple dupacks | 3.00 | 4.00 | 5.00 | 7.00 | 8.00 |
| pkts dropped (q) | 0.00 | 1.00 | 12.00 | 13.00 | 15.00 |
| pkts dropped (r) | 1.00 | 3.00 | 4.00 | 7.00 | 11.00 |
| pkts dropped (e) | 0.00 | 2.00 | 5.00 | 8.00 | 11.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 33.00 | 56.00 | 64.00 | 74.00 | 83.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 2.00 | 5.00 | 9.00 |

Table 47: 2 TCP connections started simultaneously, no competing traffic. Eifel + RED, low ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 43.34 | 56.94 | 61.75 | 67.74 | 74.64 |
| throughput | 2527.00 | 2769.00 | 2993.00 | 3283.00 | 4353.00 |
| rexmt data pkts | 3.00 | 10.00 | 15.00 | 18.00 | 21.00 |
| duplicate acks | 16.00 | 24.00 | 36.00 | 43.00 | 72.00 |
| triple dupacks | 1.00 | 2.00 | 3.00 | 3.00 | 5.00 |
| pkts dropped (q) | 0.00 | 10.00 | 12.00 | 14.00 | 15.00 |
| pkts dropped (r) | 1.00 | 2.00 | 4.00 | 5.00 | 11.00 |
| pkts dropped (e) | 0.00 | 0.00 | 2.00 | 4.00 | 7.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 15.00 | 34.00 | 43.00 | 49.00 | 78.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 1.00 | 3.00 | 7.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 58.07 | 68.17 | 71.94 | 75.05 | 93.74 |
| throughput | 2012.00 | 2350.00 | 2611.00 | 2684.00 | 3248.00 |
| rexmt data pkts | 5.00 | 15.00 | 19.00 | 23.00 | 26.00 |
| duplicate acks | 20.00 | 34.00 | 46.00 | 57.00 | 61.00 |
| triple dupacks | 1.00 | 2.00 | 3.00 | 5.00 | 7.00 |
| pkts dropped (q) | 0.00 | 12.00 | 14.00 | 15.00 | 18.00 |
| pkts dropped (r) | 2.00 | 4.00 | 5.00 | 7.00 | 24.00 |
| pkts dropped (e) | 0.00 | 4.00 | 5.00 | 6.00 | 11.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 26.00 | 44.00 | 59.00 | 65.00 | 73.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 1.00 | 1.00 | 3.00 | 4.00 |

Table 48: 2 TCP connections started simultaneously, no competing traffic. Eifel + ECN, low ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 38.89 | 46.41 | 50.03 | 52.48 | 59.07 |
| throughput | 3193.00 | 3579.00 | 3753.00 | 4017.00 | 4851.00 |
| rexmt data pkts | 0.00 | 0.00 | 1.00 | 4.00 | 11.00 |
| duplicate acks | 1.00 | 1.00 | 16.00 | 31.00 | 39.00 |
| triple dupacks | 0.00 | 0.00 | 1.00 | 3.00 | 3.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 11.00 |
| pkts dropped (e) | 0.00 | 0.00 | 1.00 | 4.00 | 6.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 0.00 | 0.00 | 16.00 | 31.00 | 39.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 2.00 | 2.00 | 4.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 54.54 | 56.41 | 59.03 | 60.56 | 63.49 |
| throughput | 2971.00 | 3063.00 | 3195.00 | 3295.00 | 3458.00 |
| rexmt data pkts | 1.00 | 3.00 | 5.00 | 7.00 | 19.00 |
| duplicate acks | 13.00 | 17.00 | 28.00 | 33.00 | 44.00 |
| triple dupacks | 1.00 | 2.00 | 2.00 | 3.00 | 4.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 14.00 |
| pkts dropped (e) | 1.00 | 3.00 | 5.00 | 6.00 | 10.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 14.00 | 18.00 | 30.00 | 40.00 | 48.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 1.00 | 2.00 | 3.00 | 6.00 |

Table 49: 2 TCP connections started simultaneously, no competing traffic. Eifel + CBI, low ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 47.43 | 51.32 | 54.77 | 58.02 | 66.52 |
| throughput | 2836.00 | 3180.00 | 3414.00 | 3658.00 | 3977.00 |
| rexmt data pkts | 0.00 | 1.00 | 2.00 | 5.00 | 22.00 |
| duplicate acks | 1.00 | 9.00 | 17.00 | 24.00 | 53.00 |
| triple dupacks | 0.00 | 1.00 | 1.00 | 2.00 | 3.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 20.00 |
| pkts dropped (r) | 0.00 | 0.00 | 1.00 | 3.00 | 53.00 |
| pkts dropped (e) | 0.00 | 1.00 | 1.00 | 4.00 | 6.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 0.00 | 8.00 | 18.00 | 24.00 | 61.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 2.00 | 2.00 | 5.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 56.37 | 60.50 | 64.30 | 66.57 | 68.97 |
| throughput | 2735.00 | 2819.00 | 2929.00 | 3018.00 | 3347.00 |
| rexmt data pkts | 2.00 | 3.00 | 4.00 | 5.00 | 9.00 |
| duplicate acks | 4.00 | 17.00 | 21.00 | 29.00 | 36.00 |
| triple dupacks | 1.00 | 2.00 | 2.00 | 3.00 | 4.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 2.00 | 5.00 | 51.00 |
| pkts dropped (e) | 1.00 | 3.00 | 4.00 | 5.00 | 9.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 4.00 | 18.00 | 21.00 | 29.00 | 41.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 1.00 | 2.00 | 5.00 |

Table 50: 2 TCP connections started simultaneously, no competing traffic. Eifel + CBI + ECN, low ARQ, queue 20 pkt.

### B.2.2 Two TCP connections, 5 Seconds Between Starts

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 31.32 | 34.29 | 45.25 | 56.59 | 73.92 |
| throughput | 2552.00 | 3233.00 | 3684.00 | 5466.00 | 6022.00 |
| rexmt data pkts | 3.00 | 9.00 | 18.00 | 21.00 | 24.00 |
| duplicate acks | 23.00 | 42.00 | 50.00 | 70.00 | 88.00 |
| triple dupacks | 1.00 | 2.00 | 3.00 | 4.00 | 5.00 |
| pkts dropped (q) | 1.00 | 3.00 | 16.00 | 18.00 | 22.00 |
| pkts dropped (e) | 0.00 | 1.00 | 2.00 | 6.00 | 16.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 23.00 | 52.00 | 62.00 | 84.00 | 95.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 2.00 | 3.00 | 5.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 39.21 | 51.84 | 53.40 | 57.98 | 74.84 |
| throughput | 2520.00 | 3188.00 | 3531.00 | 3624.00 | 4811.00 |
| rexmt data pkts | 1.00 | 3.00 | 7.00 | 13.00 | 20.00 |
| duplicate acks | 4.00 | 13.00 | 24.00 | 42.00 | 76.00 |
| triple dupacks | 1.00 | 2.00 | 2.00 | 3.00 | 6.00 |
| pkts dropped (q) | 1.00 | 2.00 | 2.00 | 9.00 | 18.00 |
| pkts dropped (e) | 0.00 | 1.00 | 2.00 | 5.00 | 8.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 5.00 | 12.00 | 28.00 | 52.00 | 89.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 2.00 | 2.00 | 5.00 |

Table 51: 2 TCP connections, 5 second difference at start times, no competing traffic. Eifel, low ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 38.98 | 49.43 | 55.11 | 64.59 | 81.73 |
| throughput | 2308.00 | 2832.00 | 3343.00 | 3620.00 | 4840.00 |
| rexmt data pkts | 9.00 | 13.00 | 23.00 | 26.00 | 29.00 |
| duplicate acks | 29.00 | 49.00 | 69.00 | 85.00 | 100.00 |
| triple dupacks | 3.00 | 3.00 | 5.00 | 6.00 | 9.00 |
| pkts dropped (q) | 0.00 | 3.00 | 17.00 | 17.00 | 20.00 |
| pkts dropped (r) | 3.00 | 3.00 | 5.00 | 6.00 | 9.00 |
| pkts dropped (e) | 0.00 | 2.00 | 3.00 | 5.00 | 10.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 35.00 | 68.00 | 81.00 | 94.00 | 117.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 1.00 | 2.00 | 3.00 | 5.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 43.75 | 55.93 | 60.62 | 65.44 | 81.01 |
| throughput | 2329.00 | 2855.00 | 3104.00 | 3328.00 | 4312.00 |
| rexmt data pkts | 4.00 | 6.00 | 9.00 | 17.00 | 27.00 |
| duplicate acks | 14.00 | 23.00 | 33.00 | 52.00 | 84.00 |
| triple dupacks | 2.00 | 3.00 | 4.00 | 5.00 | 7.00 |
| pkts dropped (q) | 0.00 | 2.00 | 2.00 | 9.00 | 18.00 |
| pkts dropped (r) | 0.00 | 1.00 | 2.00 | 5.00 | 7.00 |
| pkts dropped (e) | 0.00 | 2.00 | 4.00 | 6.00 | 7.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 15.00 | 26.00 | 37.00 | 58.00 | 107.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 1.00 | 2.00 | 5.00 |

Table 52: 2 TCP connections, 5 second difference at start times, no competing traffic. Eifel + RED, low ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 39.86 | 45.70 | 54.34 | 57.02 | 68.90 |
| throughput | 2738.00 | 3212.00 | 3460.00 | 4014.00 | 4733.00 |
| rexmt data pkts | 4.00 | 7.00 | 14.00 | 21.00 | 30.00 |
| duplicate acks | 33.00 | 39.00 | 48.00 | 58.00 | 80.00 |
| triple dupacks | 1.00 | 2.00 | 3.00 | 3.00 | 6.00 |
| pkts dropped (q) | 0.00 | 3.00 | 13.00 | 18.00 | 22.00 |
| pkts dropped (r) | 0.00 | 3.00 | 5.00 | 6.00 | 9.00 |
| pkts dropped (e) | 0.00 | 1.00 | 2.00 | 6.00 | 11.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 36.00 | 43.00 | 58.00 | 68.00 | 88.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 1.00 | 3.00 | 5.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 49.02 | 57.28 | 64.57 | 67.33 | 81.64 |
| throughput | 2311.00 | 2777.00 | 2851.00 | 3288.00 | 3848.00 |
| rexmt data pkts | 2.00 | 4.00 | 7.00 | 16.00 | 25.00 |
| duplicate acks | 10.00 | 17.00 | 25.00 | 46.00 | 75.00 |
| triple dupacks | 1.00 | 2.00 | 3.00 | 4.00 | 7.00 |
| pkts dropped (q) | 0.00 | 2.00 | 2.00 | 12.00 | 14.00 |
| pkts dropped (r) | 0.00 | 1.00 | 3.00 | 5.00 | 9.00 |
| pkts dropped (e) | 0.00 | 2.00 | 5.00 | 7.00 | 15.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 9.00 | 16.00 | 28.00 | 59.00 | 83.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 1.00 | 3.00 | 8.00 |

Table 53: 2 TCP connections, 5 second difference at start times, no competing traffic. Eifel + ECN, low ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 34.56 | 45.24 | 51.01 | 57.35 | 64.17 |
| throughput | 2940.00 | 3286.00 | 3622.00 | 4055.00 | 5458.00 |
| rexmt data pkts | 1.00 | 3.00 | 3.00 | 6.00 | 19.00 |
| duplicate acks | 12.00 | 17.00 | 20.00 | 35.00 | 52.00 |
| triple dupacks | 1.00 | 1.00 | 2.00 | 3.00 | 5.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 18.00 |
| pkts dropped (e) | 1.00 | 1.00 | 3.00 | 5.00 | 8.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 11.00 | 19.00 | 20.00 | 36.00 | 57.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 1.00 | 1.00 | 3.00 | 6.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 50.35 | 52.00 | 57.00 | 61.12 | 80.24 |
| throughput | 2351.00 | 3073.00 | 3214.00 | 3600.00 | 3747.00 |
| rexmt data pkts | 0.00 | 2.00 | 4.00 | 7.00 | 11.00 |
| duplicate acks | 1.00 | 13.00 | 27.00 | 36.00 | 47.00 |
| triple dupacks | 0.00 | 1.00 | 2.00 | 4.00 | 7.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 2.00 |
| pkts dropped (e) | 0.00 | 2.00 | 4.00 | 7.00 | 11.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 0.00 | 13.00 | 28.00 | 40.00 | 51.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 1.00 | 2.00 | 2.00 | 4.00 |

Table 54: 2 TCP connections, 5 second difference at start times, no competing traffic. Eifel + CBI, low ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 40.69 | 47.60 | 58.07 | 63.16 | 73.53 |
| throughput | 2565.00 | 2878.00 | 3140.00 | 3804.00 | 4636.00 |
| rexmt data pkts | 1.00 | 2.00 | 4.00 | 6.00 | 22.00 |
| duplicate acks | 8.00 | 19.00 | 25.00 | 30.00 | 60.00 |
| triple dupacks | 1.00 | 1.00 | 2.00 | 3.00 | 5.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 18.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 2.00 | 8.00 |
| pkts dropped (e) | 1.00 | 3.00 | 4.00 | 7.00 | 12.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 8.00 | 19.00 | 25.00 | 32.00 | 78.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 1.00 | 2.00 | 2.00 | 7.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 50.25 | 55.02 | 64.05 | 67.83 | 73.14 |
| throughput | 2579.00 | 2727.00 | 2931.00 | 3361.00 | 3754.00 |
| rexmt data pkts | 0.00 | 3.00 | 5.00 | 6.00 | 9.00 |
| duplicate acks | 1.00 | 15.00 | 32.00 | 35.00 | 47.00 |
| triple dupacks | 0.00 | 2.00 | 3.00 | 4.00 | 6.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 2.00 |
| pkts dropped (r) | 0.00 | 0.00 | 2.00 | 4.00 | 7.00 |
| pkts dropped (e) | 0.00 | 3.00 | 6.00 | 6.00 | 12.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 0.00 | 15.00 | 31.00 | 38.00 | 47.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 1.00 | 2.00 | 4.00 | 7.00 |

Table 55: 2 TCP connections, 5 second difference at start times, no competing traffic. Eifel + CBI + ECN, low ARQ, queue 20 pkt.

### B.2.3 Two TCP Connections, One UDP Flow

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 80.66 | 92.49 | 102.43 | 106.36 | 116.68 |
| throughput | 1617.00 | 1765.00 | 1833.00 | 1967.00 | 2339.00 |
| rexmt data pkts | 4.00 | 12.00 | 15.00 | 19.00 | 24.00 |
| duplicate acks | 31.00 | 42.00 | 53.00 | 60.00 | 73.00 |
| triple dupacks | 3.00 | 4.00 | 5.00 | 5.00 | 8.00 |
| pkts dropped (q) | 2.00 | 8.00 | 12.00 | 15.00 | 22.00 |
| pkts dropped (e) | 0.00 | 2.00 | 3.00 | 4.00 | 9.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 31.00 | 48.00 | 63.00 | 70.00 | 86.00 |
| pkts dropped (e) | 0.00 | 0.00 | 2.00 | 2.00 | 4.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 108.55 | 112.04 | 114.29 | 117.49 | 127.69 |
| throughput | 1477.00 | 1594.00 | 1634.00 | 1672.00 | 1738.00 |
| rexmt data pkts | 7.00 | 11.00 | 14.00 | 19.00 | 27.00 |
| duplicate acks | 25.00 | 37.00 | 47.00 | 56.00 | 85.00 |
| triple dupacks | 3.00 | 4.00 | 5.00 | 6.00 | 8.00 |
| pkts dropped (q) | 1.00 | 6.00 | 11.00 | 14.00 | 20.00 |
| pkts dropped (e) | 0.00 | 2.00 | 4.00 | 6.00 | 12.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 27.00 | 46.00 | 55.00 | 65.00 | 106.00 |
| pkts dropped (e) | 0.00 | 0.00 | 2.00 | 3.00 | 5.00 |
| UDP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 179.97 | 179.97 | 179.97 | 179.97 | 180.67 |
| throughput | 3950.00 | 3953.00 | 3961.00 | 3982.00 | 4002.00 |
| pkts dropped (q) | 5.00 | 12.00 | 19.00 | 25.00 | 30.00 |
| pkts dropped (e) | 8.00 | 13.00 | 14.00 | 18.00 | 23.00 |

Table 56: 2 TCP connections started simultaneously, 1 competing 32 kbit/s UDP flow. Eifel, low ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 93.32 | 108.39 | 112.24 | 114.94 | 120.40 |
| throughput | 1567.00 | 1620.00 | 1673.00 | 1728.00 | 2022.00 |
| rexmt data pkts | 14.00 | 20.00 | 22.00 | 24.00 | 29.00 |
| duplicate acks | 52.00 | 62.00 | 69.00 | 72.00 | 96.00 |
| triple dupacks | 6.00 | 8.00 | 9.00 | 10.00 | 13.00 |
| pkts dropped (q) | 0.00 | 8.00 | 9.00 | 9.00 | 13.00 |
| pkts dropped (r) | 4.00 | 6.00 | 8.00 | 11.00 | 17.00 |
| pkts dropped (e) | 1.00 | 2.00 | 5.00 | 6.00 | 12.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 64.00 | 76.00 | 80.00 | 84.00 | 104.00 |
| pkts dropped (e) | 0.00 | 1.00 | 2.00 | 3.00 | 6.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 115.71 | 120.18 | 123.82 | 126.27 | 136.60 |
| throughput | 1381.00 | 1490.00 | 1517.00 | 1568.00 | 1630.00 |
| rexmt data pkts | 16.00 | 22.00 | 24.00 | 26.00 | 30.00 |
| duplicate acks | 51.00 | 59.00 | 70.00 | 78.00 | 84.00 |
| triple dupacks | 7.00 | 9.00 | 10.00 | 11.00 | 12.00 |
| pkts dropped (q) | 0.00 | 5.00 | 7.00 | 9.00 | 11.00 |
| pkts dropped (r) | 5.00 | 9.00 | 11.00 | 13.00 | 17.00 |
| pkts dropped (e) | 2.00 | 4.00 | 5.00 | 6.00 | 14.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 61.00 | 71.00 | 80.00 | 86.00 | 96.00 |
| pkts dropped (e) | 0.00 | 2.00 | 2.00 | 3.00 | 5.00 |
| UDP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 179.27 | 179.97 | 179.97 | 179.97 | 180.67 |
| throughput | 3863.00 | 3894.00 | 3915.00 | 3924.00 | 3964.00 |
| pkts dropped (q) | 2.00 | 5.00 | 7.00 | 10.00 | 13.00 |
| pkts dropped (r) | 15.00 | 24.00 | 29.00 | 33.00 | 44.00 |
| pkts dropped (e) | 8.00 | 12.00 | 15.00 | 19.00 | 29.00 |

Table 57: 2 TCP connections started simultaneously, 1 competing 32 kbit/s UDP flow. Eifel + RED, low ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 99.33 | 103.03 | 108.42 | 111.83 | 119.41 |
| throughput | 1580.00 | 1687.00 | 1740.00 | 1831.00 | 1899.00 |
| rexmt data pkts | 9.00 | 12.00 | 12.00 | 14.00 | 14.00 |
| duplicate acks | 25.00 | 25.00 | 38.00 | 41.00 | 50.00 |
| triple dupacks | 2.00 | 3.00 | 3.00 | 3.00 | 6.00 |
| pkts dropped (q) | 3.00 | 6.00 | 9.00 | 9.00 | 12.00 |
| pkts dropped (r) | 7.00 | 8.00 | 11.00 | 12.00 | 17.00 |
| pkts dropped (e) | 1.00 | 2.00 | 3.00 | 3.00 | 9.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 33.00 | 33.00 | 43.00 | 51.00 | 59.00 |
| pkts dropped (e) | 1.00 | 1.00 | 2.00 | 5.00 | 6.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 111.78 | 113.92 | 114.57 | 114.94 | 122.53 |
| throughput | 1540.00 | 1641.00 | 1646.00 | 1656.00 | 1688.00 |
| rexmt data pkts | 5.00 | 8.00 | 10.00 | 11.00 | 11.00 |
| duplicate acks | 14.00 | 20.00 | 24.00 | 36.00 | 50.00 |
| triple dupacks | 1.00 | 2.00 | 3.00 | 4.00 | 8.00 |
| pkts dropped (q) | 2.00 | 2.00 | 6.00 | 6.00 | 8.00 |
| pkts dropped (r) | 9.00 | 9.00 | 16.00 | 16.00 | 18.00 |
| pkts dropped (e) | 0.00 | 4.00 | 4.00 | 4.00 | 7.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 20.00 | 25.00 | 27.00 | 37.00 | 53.00 |
| pkts dropped (e) | 0.00 | 0.00 | 2.00 | 3.00 | 4.00 |
| UDP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 179.97 | 179.97 | 179.97 | 179.97 | 179.97 |
| throughput | 3846.00 | 3875.00 | 3930.00 | 3933.00 | 3933.00 |
| pkts dropped (q) | 3.00 | 6.00 | 10.00 | 11.00 | 13.00 |
| pkts dropped (r) | 25.00 | 31.00 | 33.00 | 45.00 | 50.00 |
| pkts dropped (e) | 6.00 | 10.00 | 13.00 | 21.00 | 25.00 |

Table 58: 2 TCP connections started simultaneously, 1 competing 32 kbit/s UDP flow. Eifel + ECN, low ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 78.83 | 91.15 | 100.00 | 104.77 | 105.95 |
| throughput | 1781.00 | 1801.00 | 1827.00 | 2069.00 | 2393.00 |
| rexmt data pkts | 3.00 | 5.00 | 6.00 | 8.00 | 16.00 |
| duplicate acks | 26.00 | 35.00 | 37.00 | 39.00 | 57.00 |
| triple dupacks | 2.00 | 3.00 | 3.00 | 4.00 | 4.00 |
| pkts dropped (q) | 2.00 | 2.00 | 3.00 | 5.00 | 15.00 |
| pkts dropped (e) | 0.00 | 0.00 | 1.00 | 4.00 | 6.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 27.00 | 34.00 | 40.00 | 43.00 | 70.00 |
| pkts dropped (e) | 0.00 | 0.00 | 1.00 | 3.00 | 4.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 111.86 | 112.51 | 115.85 | 116.83 | 119.91 |
| throughput | 1573.00 | 1615.00 | 1622.00 | 1677.00 | 1686.00 |
| rexmt data pkts | 6.00 | 8.00 | 10.00 | 12.00 | 14.00 |
| duplicate acks | 20.00 | 32.00 | 43.00 | 50.00 | 53.00 |
| triple dupacks | 3.00 | 3.00 | 5.00 | 6.00 | 7.00 |
| pkts dropped (q) | 2.00 | 3.00 | 4.00 | 5.00 | 9.00 |
| pkts dropped (e) | 0.00 | 3.00 | 5.00 | 7.00 | 16.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 26.00 | 36.00 | 46.00 | 53.00 | 59.00 |
| pkts dropped (e) | 0.00 | 0.00 | 1.00 | 3.00 | 5.00 |
| UDP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 179.97 | 179.97 | 179.97 | 179.97 | 180.67 |
| throughput | 3940.00 | 3967.00 | 3976.00 | 3996.00 | 4025.00 |
| pkts dropped (q) | 2.00 | 13.00 | 13.00 | 19.00 | 27.00 |
| pkts dropped (e) | 8.00 | 11.00 | 14.00 | 18.00 | 21.00 |

Table 59: 2 TCP connections started simultaneously, 1 competing 32 kbit/s UDP flow. Eifel + CBI, low ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 92.92 | 104.58 | 110.68 | 112.66 | 117.66 |
| throughput | 1603.00 | 1666.00 | 1698.00 | 1797.00 | 2030.00 |
| rexmt data pkts | 2.00 | 3.00 | 4.00 | 5.00 | 10.00 |
| duplicate acks | 6.00 | 11.00 | 15.00 | 24.00 | 35.00 |
| triple dupacks | 1.00 | 2.00 | 2.00 | 3.00 | 5.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 8.00 |
| pkts dropped (r) | 7.00 | 10.00 | 12.00 | 20.00 | 45.00 |
| pkts dropped (e) | 0.00 | 2.00 | 3.00 | 5.00 | 10.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 6.00 | 10.00 | 16.00 | 25.00 | 35.00 |
| pkts dropped (e) | 0.00 | 0.00 | 2.00 | 3.00 | 5.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 112.53 | 114.79 | 117.60 | 121.36 | 130.28 |
| throughput | 1448.00 | 1548.00 | 1597.00 | 1640.00 | 1676.00 |
| rexmt data pkts | 1.00 | 3.00 | 5.00 | 9.00 | 21.00 |
| duplicate acks | 4.00 | 12.00 | 18.00 | 26.00 | 57.00 |
| triple dupacks | 1.00 | 2.00 | 3.00 | 4.00 | 8.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 14.00 |
| pkts dropped (r) | 7.00 | 13.00 | 17.00 | 19.00 | 43.00 |
| pkts dropped (e) | 1.00 | 2.00 | 4.00 | 8.00 | 12.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 3.00 | 11.00 | 22.00 | 26.00 | 70.00 |
| pkts dropped (e) | 0.00 | 1.00 | 2.00 | 4.00 | 6.00 |
| UDP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 179.27 | 179.97 | 179.97 | 179.97 | 180.67 |
| throughput | 3817.00 | 3863.00 | 3906.00 | 3941.00 | 3976.00 |
| pkts dropped (q) | 0.00 | 0.00 | 1.00 | 1.00 | 12.00 |
| pkts dropped (r) | 16.00 | 26.00 | 43.00 | 48.00 | 98.00 |
| pkts dropped (e) | 6.00 | 12.00 | 14.00 | 19.00 | 27.00 |

Table 60: 2 TCP connections started simultaneously, 1 competing 32 kbit/s UDP flow. Eifel + CBI + ECN, low ARQ, queue 20 pkt.

## B.3 Medium ARQ persistency

### Two TCP Connections Start at the Same Time

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 37.61 | 42.09 | 51.28 | 54.24 | 59.73 |
| throughput | 3158.00 | 3441.00 | 3576.00 | 4473.00 | 5015.00 |
| rexmt data pkts | 6.00 | 12.00 | 13.00 | 14.00 | 15.00 |
| duplicate acks | 20.00 | 21.00 | 23.00 | 43.00 | 48.00 |
| triple dupacks | 1.00 | 1.00 | 1.00 | 2.00 | 3.00 |
| pkts dropped (q) | 6.00 | 12.00 | 13.00 | 14.00 | 15.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 29.00 | 33.00 | 34.00 | 52.00 | 57.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 56.54 | 58.17 | 59.13 | 59.60 | 62.85 |
| throughput | 3001.00 | 3152.00 | 3189.00 | 3239.00 | 3336.00 |
| rexmt data pkts | 12.00 | 13.00 | 15.00 | 16.00 | 18.00 |
| duplicate acks | 16.00 | 19.00 | 35.00 | 40.00 | 44.00 |
| triple dupacks | 1.00 | 1.00 | 2.00 | 2.00 | 2.00 |
| pkts dropped (q) | 12.00 | 13.00 | 15.00 | 16.00 | 18.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 29.00 | 33.00 | 42.00 | 52.00 | 57.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 61: 2 TCP connections started simultaneously, no competing traffic. Baseline, medium ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 37.01 | 43.83 | 50.78 | 55.93 | 61.12 |
| throughput | 3086.00 | 3360.00 | 3563.00 | 4233.00 | 5098.00 |
| rexmt data pkts | 7.00 | 11.00 | 14.00 | 15.00 | 23.00 |
| duplicate acks | 21.00 | 23.00 | 34.00 | 45.00 | 50.00 |
| triple dupacks | 1.00 | 1.00 | 2.00 | 2.00 | 3.00 |
| pkts dropped (q) | 6.00 | 11.00 | 14.00 | 14.00 | 16.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 29.00 | 34.00 | 44.00 | 56.00 | 58.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 58.02 | 59.07 | 60.51 | 61.45 | 63.65 |
| throughput | 2964.00 | 3069.00 | 3105.00 | 3163.00 | 3251.00 |
| rexmt data pkts | 11.00 | 13.00 | 15.00 | 16.00 | 17.00 |
| duplicate acks | 17.00 | 37.00 | 39.00 | 41.00 | 46.00 |
| triple dupacks | 1.00 | 2.00 | 2.00 | 2.00 | 3.00 |
| pkts dropped (q) | 11.00 | 13.00 | 15.00 | 16.00 | 17.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 26.00 | 44.00 | 51.00 | 54.00 | 57.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 62: 2 TCP connections started simultaneously, no competing traffic. Eifel, medium ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 35.21 | 39.81 | 44.22 | 52.63 | 59.94 |
| throughput | 3147.00 | 3552.00 | 4261.00 | 4701.00 | 5358.00 |
| rexmt data pkts | 7.00 | 10.00 | 11.00 | 13.00 | 15.00 |
| duplicate acks | 20.00 | 23.00 | 25.00 | 35.00 | 48.00 |
| triple dupacks | 1.00 | 1.00 | 1.00 | 2.00 | 3.00 |
| pkts dropped (q) | 7.00 | 10.00 | 11.00 | 13.00 | 15.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 30.00 | 34.00 | 35.00 | 43.00 | 58.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 56.11 | 57.44 | 58.32 | 59.70 | 62.29 |
| throughput | 3028.00 | 3126.00 | 3232.00 | 3273.00 | 3362.00 |
| rexmt data pkts | 9.00 | 13.00 | 15.00 | 16.00 | 20.00 |
| duplicate acks | 17.00 | 19.00 | 20.00 | 36.00 | 46.00 |
| triple dupacks | 1.00 | 1.00 | 1.00 | 2.00 | 2.00 |
| pkts dropped (q) | 9.00 | 13.00 | 15.00 | 16.00 | 20.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 25.00 | 32.00 | 34.00 | 46.00 | 59.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 63: 2 TCP connections started simultaneously, no competing traffic. F-RTO, medium ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 40.23 | 47.45 | 52.68 | 55.52 | 63.51 |
| throughput | 2970.00 | 3239.00 | 3577.00 | 3933.00 | 4689.00 |
| rexmt data pkts | 10.00 | 11.00 | 12.00 | 14.00 | 15.00 |
| duplicate acks | 20.00 | 22.00 | 32.00 | 43.00 | 48.00 |
| triple dupacks | 1.00 | 1.00 | 2.00 | 2.00 | 3.00 |
| pkts dropped (q) | 10.00 | 11.00 | 12.00 | 14.00 | 15.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 31.00 | 33.00 | 40.00 | 54.00 | 58.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 56.40 | 57.55 | 58.50 | 59.43 | 63.52 |
| throughput | 2970.00 | 3174.00 | 3220.00 | 3267.00 | 3345.00 |
| rexmt data pkts | 11.00 | 12.00 | 13.00 | 15.00 | 17.00 |
| duplicate acks | 18.00 | 28.00 | 40.00 | 41.00 | 44.00 |
| triple dupacks | 1.00 | 1.00 | 2.00 | 2.00 | 3.00 |
| pkts dropped (q) | 11.00 | 12.00 | 13.00 | 15.00 | 17.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 28.00 | 33.00 | 51.00 | 54.00 | 56.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 64: 2 TCP connections started simultaneously, no competing traffic. D-SACK, medium ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 46.87 | 57.42 | 62.36 | 65.56 | 73.60 |
| throughput | 2563.00 | 2857.00 | 2957.00 | 3195.00 | 4024.00 |
| rexmt data pkts | 14.00 | 16.00 | 18.00 | 19.00 | 21.00 |
| duplicate acks | 28.00 | 49.00 | 60.00 | 67.00 | 76.00 |
| triple dupacks | 2.00 | 3.00 | 4.00 | 5.00 | 6.00 |
| pkts dropped (q) | 9.00 | 11.00 | 12.00 | 13.00 | 14.00 |
| pkts dropped (r) | 2.00 | 5.00 | 6.00 | 7.00 | 10.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 40.00 | 56.00 | 67.00 | 76.00 | 86.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 60.78 | 65.48 | 69.01 | 71.04 | 82.09 |
| throughput | 2298.00 | 2636.00 | 2711.00 | 2782.00 | 3104.00 |
| rexmt data pkts | 13.00 | 17.00 | 19.00 | 21.00 | 24.00 |
| duplicate acks | 29.00 | 45.00 | 59.00 | 66.00 | 95.00 |
| triple dupacks | 2.00 | 4.00 | 5.00 | 6.00 | 8.00 |
| pkts dropped (q) | 8.00 | 10.00 | 12.00 | 13.00 | 14.00 |
| pkts dropped (r) | 3.00 | 5.00 | 6.00 | 9.00 | 13.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 41.00 | 51.00 | 67.00 | 73.00 | 99.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 65: 2 TCP connections started simultaneously, no competing traffic. Eifel + RED, medium ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 47.98 | 56.43 | 61.02 | 64.40 | 67.64 |
| throughput | 2789.00 | 2912.00 | 3071.00 | 3226.00 | 3932.00 |
| rexmt data pkts | 10.00 | 12.00 | 15.00 | 16.00 | 17.00 |
| duplicate acks | 16.00 | 22.00 | 27.00 | 32.00 | 52.00 |
| triple dupacks | 1.00 | 1.00 | 2.00 | 2.00 | 2.00 |
| pkts dropped (q) | 9.00 | 12.00 | 13.00 | 14.00 | 17.00 |
| pkts dropped (r) | 3.00 | 4.00 | 6.00 | 8.00 | 16.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 26.00 | 29.00 | 34.00 | 38.00 | 64.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 64.46 | 66.71 | 67.49 | 69.39 | 90.85 |
| throughput | 2077.00 | 2664.00 | 2783.00 | 2814.00 | 2927.00 |
| rexmt data pkts | 10.00 | 12.00 | 15.00 | 15.00 | 22.00 |
| duplicate acks | 19.00 | 23.00 | 28.00 | 30.00 | 49.00 |
| triple dupacks | 1.00 | 1.00 | 1.00 | 2.00 | 3.00 |
| pkts dropped (q) | 9.00 | 12.00 | 13.00 | 14.00 | 17.00 |
| pkts dropped (r) | 4.00 | 5.00 | 6.00 | 8.00 | 23.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 28.00 | 30.00 | 33.00 | 35.00 | 62.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 66: 2 TCP connections started simultaneously, no competing traffic. Eifel + ECN, medium ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 41.89 | 52.46 | 53.31 | 55.06 | 58.31 |
| throughput | 3235.00 | 3402.00 | 3534.00 | 3575.00 | 4503.00 |
| rexmt data pkts | 0.00 | 0.00 | 0.00 | 1.00 | 9.00 |
| duplicate acks | 1.00 | 1.00 | 1.00 | 21.00 | 26.00 |
| triple dupacks | 0.00 | 0.00 | 0.00 | 1.00 | 1.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 1.00 | 9.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 0.00 | 0.00 | 0.00 | 20.00 | 33.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 53.67 | 56.27 | 56.71 | 57.67 | 58.65 |
| throughput | 3216.00 | 3270.00 | 3314.00 | 3345.00 | 3515.00 |
| rexmt data pkts | 1.00 | 1.00 | 2.00 | 2.00 | 17.00 |
| duplicate acks | 17.00 | 20.00 | 20.00 | 21.00 | 27.00 |
| triple dupacks | 1.00 | 1.00 | 1.00 | 1.00 | 2.00 |
| pkts dropped (q) | 1.00 | 1.00 | 2.00 | 2.00 | 17.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 19.00 | 20.00 | 20.00 | 21.00 | 32.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 67: 2 TCP connections started simultaneously, no competing traffic. Eifel + CBI, medium ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 53.74 | 56.13 | 58.12 | 60.52 | 63.85 |
| throughput | 2954.00 | 3112.00 | 3210.00 | 3352.00 | 3510.00 |
| rexmt data pkts | 0.00 | 0.00 | 0.00 | 1.00 | 9.00 |
| duplicate acks | 1.00 | 1.00 | 1.00 | 2.00 | 35.00 |
| triple dupacks | 0.00 | 0.00 | 0.00 | 0.00 | 2.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 8.00 |
| pkts dropped (r) | 2.00 | 4.00 | 5.00 | 6.00 | 9.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 0.00 | 0.00 | 0.00 | 0.00 | 36.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 58.18 | 59.64 | 61.88 | 64.34 | 65.64 |
| throughput | 2874.00 | 2926.00 | 3002.00 | 3158.00 | 3242.00 |
| rexmt data pkts | 0.00 | 0.00 | 0.00 | 1.00 | 13.00 |
| duplicate acks | 1.00 | 1.00 | 1.00 | 2.00 | 19.00 |
| triple dupacks | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 13.00 |
| pkts dropped (r) | 4.00 | 5.00 | 6.00 | 7.00 | 13.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 0.00 | 0.00 | 0.00 | 0.00 | 30.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (r) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 68: 2 TCP connections started simultaneously, no competing traffic. Eifel + CBI + ECN, medium ARQ, queue 20 pkt.

## B.4 High ARQ persistency

### Two TCP Connections Start at the Same Time

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 40.54 | 57.16 | 65.09 | 72.08 | 91.61 |
| throughput | 2059.00 | 2567.00 | 2891.00 | 3290.00 | 4653.00 |
| rexmt data pkts | 8.00 | 12.00 | 14.00 | 23.00 | 56.00 |
| duplicate acks | 14.00 | 22.00 | 23.00 | 34.00 | 71.00 |
| triple dupacks | 1.00 | 1.00 | 1.00 | 3.00 | 5.00 |
| pkts dropped (q) | 0.00 | 9.00 | 12.00 | 14.00 | 20.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 0.00 | 11.00 | 32.00 | 35.00 | 86.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 62.85 | 69.48 | 76.02 | 82.53 | 108.35 |
| throughput | 1741.00 | 2280.00 | 2478.00 | 2662.00 | 3001.00 |
| rexmt data pkts | 12.00 | 17.00 | 33.00 | 46.00 | 55.00 |
| duplicate acks | 13.00 | 32.00 | 38.00 | 44.00 | 67.00 |
| triple dupacks | 1.00 | 2.00 | 3.00 | 4.00 | 6.00 |
| pkts dropped (q) | 0.00 | 8.00 | 14.00 | 15.00 | 24.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 0.00 | 11.00 | 33.00 | 36.00 | 57.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 69: 2 TCP connections started simultaneously, no competing traffic. Baseline, high ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 40.30 | 49.80 | 56.77 | 64.08 | 69.22 |
| throughput | 2725.00 | 2883.00 | 3315.00 | 3667.00 | 4681.00 |
| rexmt data pkts | 8.00 | 10.00 | 13.00 | 14.00 | 22.00 |
| duplicate acks | 22.00 | 24.00 | 25.00 | 42.00 | 48.00 |
| triple dupacks | 1.00 | 1.00 | 1.00 | 2.00 | 3.00 |
| pkts dropped (q) | 7.00 | 10.00 | 13.00 | 14.00 | 15.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 28.00 | 33.00 | 35.00 | 54.00 | 59.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 63.63 | 65.75 | 68.58 | 71.77 | 80.30 |
| throughput | 2349.00 | 2583.00 | 2739.00 | 2844.00 | 2965.00 |
| rexmt data pkts | 12.00 | 15.00 | 16.00 | 17.00 | 18.00 |
| duplicate acks | 17.00 | 20.00 | 38.00 | 42.00 | 46.00 |
| triple dupacks | 1.00 | 1.00 | 2.00 | 2.00 | 3.00 |
| pkts dropped (q) | 12.00 | 13.00 | 15.00 | 16.00 | 17.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 24.00 | 33.00 | 49.00 | 54.00 | 58.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 70: 2 TCP connections started simultaneously, no competing traffic. Eifel, high ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 43.88 | 49.32 | 57.16 | 62.46 | 68.48 |
| throughput | 2754.00 | 2979.00 | 3218.00 | 3767.00 | 4299.00 |
| rexmt data pkts | 2.00 | 11.00 | 13.00 | 14.00 | 17.00 |
| duplicate acks | 1.00 | 21.00 | 22.00 | 33.00 | 61.00 |
| triple dupacks | 0.00 | 1.00 | 1.00 | 2.00 | 3.00 |
| pkts dropped (q) | 0.00 | 11.00 | 13.00 | 14.00 | 17.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 0.00 | 31.00 | 34.00 | 48.00 | 76.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 60.83 | 64.13 | 66.47 | 68.72 | 72.84 |
| throughput | 2590.00 | 2719.00 | 2769.00 | 2930.00 | 3101.00 |
| rexmt data pkts | 1.00 | 3.00 | 14.00 | 16.00 | 17.00 |
| duplicate acks | 0.00 | 9.00 | 24.00 | 39.00 | 44.00 |
| triple dupacks | 0.00 | 1.00 | 1.00 | 2.00 | 3.00 |
| pkts dropped (q) | 0.00 | 1.00 | 14.00 | 15.00 | 17.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 0.00 | 8.00 | 34.00 | 51.00 | 58.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 71: 2 TCP connections started simultaneously, no competing traffic. F-RTO, high ARQ, queue 20 pkt.

| TCP SET 1, CLI→SRV | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|
| elapsed time | 42.21 | 53.90 | 65.61 | 71.85 | 83.63 |
| throughput | 2256.00 | 2604.00 | 2866.00 | 3499.00 | 4469.00 |
| rexmt data pkts | 10.00 | 13.00 | 15.00 | 43.00 | 62.00 |
| duplicate acks | 21.00 | 25.00 | 45.00 | 55.00 | 59.00 |
| triple dupacks | 1.00 | 2.00 | 2.00 | 4.00 | 7.00 |
| pkts dropped (q) | 0.00 | 12.00 | 13.00 | 15.00 | 18.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 1, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 32.00 | 34.00 | 57.00 | 63.00 | 82.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, CLI→SRV | min | 25% | 50% | 75% | max |
| elapsed time | 59.37 | 67.01 | 67.85 | 75.03 | 102.46 |
| throughput | 1841.00 | 2355.00 | 2714.00 | 2810.00 | 3177.00 |
| rexmt data pkts | 8.00 | 15.00 | 28.00 | 42.00 | 73.00 |
| duplicate acks | 14.00 | 30.00 | 41.00 | 48.00 | 70.00 |
| triple dupacks | 1.00 | 2.00 | 2.00 | 4.00 | 8.00 |
| pkts dropped (q) | 0.00 | 12.00 | 15.00 | 16.00 | 20.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TCP SET 2, SRV→CLI | min | 25% | 50% | 75% | max |
| sacks sent | 15.00 | 39.00 | 51.00 | 65.00 | 95.00 |
| pkts dropped (q) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| pkts dropped (e) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 72: 2 TCP connections started simultaneously, no competing traffic. D-SACK, high ARQ, queue 20 pkt.