TCP-Aix
Making TCP Robust to Reordering and Delay Variations

Hannes Ekström, Sara Landström
Reiner Ludwig, Lars-Åke Larzon
TCP-Aix: Making TCP Robust to Reordering and Delay Variations

Sara Landström†
Lars-Ake Larzon†‡
Luleå University of Technology
†Uppsala University
‡Sweden

Hannes Ekström*
Reiner Ludwig*
Ericsson Research
*Sweden, *Germany

Abstract

In this paper, we describe and evaluate TCP-Aix, a sender-side TCP algorithm designed to make TCP performance robust to packet reordering and delay. TCP-Aix consists of three novel components: (i) decoupling of loss recovery actions from congestion control actions, (ii) candid growth of the congestion window upon reception of dupacks, allowing for improved performance in reordering environments, and (iii) an algorithm for determining an appropriate dupthresh value.

The results from our simulation study show that, by combining these three components, the performance of TCP-Aix is largely impervious even to the highest packet reordering rates investigated in this paper of 8%, where the duration of the reordering is in the order 2-3 end-to-end RTTs. TCP-Aix outperforms a simulated state-of-the-art TCP sender as well as a TCP-NCR sender [10], the performance of which both display a much stronger dependence on packet reordering. Improved performance in the presence of delay spikes is also shown.

Strengthened by simulation results, we further argue that the improved performance of TCP-Aix does not come at the expense of other, competing, TCP senders. Rather, the TCP-Aix senders utilise bandwidth that would otherwise have been left unutilised. We therefore believe that a gradual deployment of TCP-Aix would be feasible.

1 Introduction

In this paper we present a set of modifications to the sender-side of TCP, which improves its performance in reordering environments and in the case of delay spikes. In particular, these extensions enhance the operation of TCP with respect to long reordering lengths, high reordering rates and a large bandwidth×delay product. We refer to this modified TCP as TCP-Aix.

The sensitivity of TCP performance to packet reordering is well documented. TCP assumes that a certain number of consecutive duplicate acknowledgements, dupacks, or a timeout, are congestion indications. However, reordering and
delay variations may also trigger these signals which leads to an undesired reduction of the TCP sending rate. To avoid this, most Internet components are required to deliver packets in order.

The in-order delivery demand constitutes an obstacle for deployment of promising techniques which may cause reordering, such as parallelism in packet forwarding [16], differentiated services [12] and multi-path routing. It may be possible to add functionality for re-establishing the incoming packet order before releasing the data packets to enable these technologies. This has been done for many link layer retransmission schemes. The cost is however increased complexity and larger delay variations due to the operation of the re-sequencing buffer. By making TCP agnostic to reordering, the requirement for in-order delivery on network components could be relaxed, thereby reducing overall system complexity and making deployment feasible of the previously mentioned techniques.

Earlier proposals aimed at improving the robustness of TCP to reordering, like Eifel-, DSACK- and F-RTO-based algorithms, have in common that they restore the congestion control state after detecting that the sending rate was reduced by mistake. Instead both NCR and Aix delay the congestion control decision. However, while NCR triggers loss recovery and congestion control at the same point in time, Aix separates these two decision points. It initiates loss recovery before congestion control. While waiting for more information to base the congestion control decision on after the retransmission, packet conservation is used to keep the acknowledgement clock running while not stressing the network unfairly. The separation of the loss recovery and congestion control decisions gives TCP-Aix sufficient time to differentiate between congestion and reordering events where a segment is delayed less than one RTT.

To be able to identify reordering events longer than an RTT, TCP-Aix uses the packet conservation principle, conservative timer management and a higher duplicate acknowledgement threshold, dupthresh than default. We introduce a novel algorithm, called the winthresh algorithm, for calculating the highest possible dupthresh setting with regards to the available buffer space at both ends of the connection.

However, when the bandwidth×delay product is large and reordering is often encountered it is not sufficient to be able to distinguish between congestion and reordering events to get good performance. TCP-Aix uses a mechanism which we call candid growth to achieve high throughput in scenarios where dupacks are frequent.

Through simulations we have verified the operation of TCP-Aix and compared its performance to TCP-NCR and a modern standards-compliant TCP including Limited Transmit [3], SACK-based loss recovery [15] and Congestion Window Validation [23]. Our results show that TCP-Aix can handle reordering lengths in the order of two to three RTTs. It also manages to utilise capacity better than TCP-NCR for high reordering rates when the bandwidth×delay product is large and the reordering length is less than one RTT.

The remainder of this paper is organised as follows; Section 2 introduces previous work in this area. It is followed by a presentation of TCP-Aix 3 and the winthresh algorithm in Section 4. Thereafter the simulation environment is described in Section 5 and the results from the evaluation are given in Sections 6 and 7. We discuss our design in Section 8 and finally in Section 9 we present our conclusions.
2 Related work

Although the sensibility of TCP to reordering has inferred a limitation on the design space of the Internet, observations of reordering in today’s network due to the transient behaviour of routers [29], [30] and in certain high-speed switch architectures [8] have been reported. In [22], it is demonstrated that there is a high prevalence of packet reordering relative to packet loss. They conclude that new application and protocol designs should be robust to both packet reordering and packet loss. A reordering tolerant TCP could thus also enhance performance in the present Internet.

Previous approaches to improve the robustness of TCP to reordering include using a higher dupthresh setting and revalidating the congestion control decision when more information is received. We first describe the methods proposed for determining whether a retransmission was necessary, before discussing the different approaches to setting dupthresh.

Through the Duplicate SACK (DSACK) option the receiver can inform the sender whenever duplicate segments have arrived, which makes it possible for the sender to determine when it has needlessly retransmitted a segment [14]. However, it takes at least an RTT from the segment being retransmitted until the sender becomes aware of the unnecessary retransmission. Another proposal is Forward RTO Recovery (F-RTO) which does not require any TCP options to operate. It can however only detect spurious timeouts [32].

TCP-Aix builds on TCP-Eifel [27] since it requires a reliable event that can trigger the congestion control decision. Furthermore, the Eifel algorithm can determine whether the retransmission was unnecessary when it gets the acknowledgement of the segment indicated as lost. The Eifel algorithm tags each segment with information such that the original transmission and subsequent retransmissions of the same segment can be distinguished. This extra information may either be a time stamp or a dedicated bit in the TCP header. A congestion control decision can then be revalidated and the congestion control action revoked if the retransmission was unnecessary, upon receiving the first acknowledgement that advances the cumulative acknowledgement point after a congestion event.

To avoid spurious fast retransmits and incorrect congestion control actions, dupthresh should be set higher than the reordering length. Currently, dupthresh is usually set to three [5]. In [13] and [33] algorithms for adapting dupthresh in accordance to the current network characteristics were proposed. These algorithms gather information about the characteristics of a certain connection and may therefore provide only limited improvement for short flows and during the initial phase of a transfer, unless information can be shared between connections. If the network characteristics changes frequently, this approach may not lead to effective dupthresh values.

Instead of attempting to determine the degree of reordering in the network, TCP-DCR [11] sets dupthresh as high as possible with regards to the retransmission timer of TCP. This means that reordering events shorter than approximately one RTT can be correctly identified. TCP-DCR is being standardised by the IETF under the name TCP-NCR [10].

The algorithms in [13], [33] and [11] extend Limited transmit [3] in order to increase the probability of reaching the higher dupthresh. Extending Limited Transmit comes at the cost of additional buffer space being needed at the re-
TCP−Aix initiates LR
RFC2581 initiates LR and reduces cwnd
Loss indication −>
one RTT −> TCP−NCR initiates LR and reduces cwnd
newACK −> TCP−Aix reduces cwnd

(a) TCP-Aix with the standard dupthresh algorithm.

TCP−Aix initiates LR
(numDupacks = dupthresh) −>
TCP−Aix sets dupthresh
Depends on dupthresh One RTT
TCP−Aix reduces cwnd

(b) TCP-Aix with increased dupthresh.

Figure 1: A comparison of when the different TCP flavors perform Loss Recovery, LR, and reduces cwnd in response to congestion.

cceiver, which increases the risk of the TCP sender being forced to refrain from sending due to buffer space limitations, e.g., window stalling. Window stalling makes the TCP flows burstier [4]. When the acknowledgement that has been holding the window up arrives a large burst of segments can often be released.

3 Description of TCP-Aix

Upon receiving a congestion indication, i.e., a timeout or dupthresh dupacks, a standards-compliant TCP initiates loss recovery and reduces its congestion window, cwnd. In TCP-Aix a congestion indication immediately triggers loss recovery, but the decision whether to reduce cwnd, is postponed until a more reliable decision can be made. The additional information that TCP-Aix waits for is the acknowledgement of the segment which has been indicated as lost. It triggers the congestion control decision. We will refer to this acknowledgement as the newACK. If the newACK was sent in response to the retransmission, the TCP-Aix sender reduces its cwnd otherwise it enters slow start for a limited time period. Figure 1(a) illustrates when loss recovery and congestion control is performed for a standards-compliant TCP, TCP-NCR and TCP-Aix, if the original segment is lost.

TCP-Aix and TCP-NCR both reduce their cwnds at approximately the same time, since the newACK arrives one RTT after the retransmission and TCP-NCR sets its dupthresh to correspond to approximately one RTT. Both proposals thus can distinguish reordering events where the segment is reordered by less than one RTT. By increasing dupthresh and applying among other things con-
servative timer management we have made it possible for TCP-Aix to also correctly class reordering events where a segment is delayed by more than one RTT, see Figure 1(b). The time interval during which the reordered segment has to arrive to avoid a performance penalty, consists of the one RTT that it takes for the acknowledgement to arrive after the retransmission and the time until retransmission which depends on the `dupthresh` setting.

In Figure 3 the details of TCP-Aix are laid out. During loss free periods and after a packet loss has been corroborated, TCP-Aix follows the same set of rules as TCP with SACK-based loss recovery [5], [15]. The operation of TCP-Aix has been split into two parts; one which deals with reordering and another for delay spikes. There is a corroboration state for each type of event. When a retransmission is performed after a loss free period, TCP-Aix pass into the corresponding `Corroboration` state where the sender remains until a `newACK` is received. A retransmission is as usual triggered by either a timeout or `dupthresh` `dupacks`.

At present the `Adapt dupthresh` and `Adapt RTO` actions shown when returning to `Non Loss Recovery` via a corroboration state are placeholders for algorithms which may attempt to find improved `dupthresh` or `RTO` settings.
Timeout
retransmit
cwnd = LW
recover = HighACK
Erase SACK-Scoreboard

Loss Recovery Phase
(see RFC3517)

DUPACK
Packet Conservation
Grow cwnd_aix_

ACK for Retransmit
set recover_
ssthresh = cwnd / 2
cwnd = ssthresh + 1
set pipe_

DUPACK Packet Conservation
grow cwnd_aix_

Non Loss Recovery Phase
(see Standards - Tracks RFCs)

Number DUPACKs = dupthresh
Grow cwnd_aix_

ACK for Original and ACK > high_retr_
cwnd = min(cwnd + 1, cwnd_aix)
ssthresh = max(cwnd_aix, ssthresh)
SND.NXT = SND.MAX
Adapt dupthresh
Don't Erase SACK-Scoreboard

ACK for Retransmit
ssthresh = cwnd / 2
cwnd = LW + 1
SND.NXT = SND.UNA
Erase SACK-Scoreboard

ACK for Original

cwnd = min(cwnd + 1, cwnd_aix)
ssthresh = max(cwnd_aix, ssthresh)
SND.NXT = SND.MAX
Adapt RTO
Don't Erase SACK-Scoreboard

NewACK

Timeout
retransmit
cwnd = LW
recover = HighACK
Erase SACK-Scoreboard

Reordering Corroboration Phase

ACK for Retransmit
new ACK > high_retr_

cwnd = min(cwnd_aix + 1, cwnd_aix)
ssthresh = max(cwnd_aix, ssthresh)
SND.NXT = SND.MAX
Adapt dupthresh
Don't Erase SACK-Scoreboard

Timeout
retransmit
cwnd = LW

NewACK > recover_

Timeout
retransmit
cwnd = LW

DUPACK Packet Conservation
grow cwnd_aix_

DUPACK Phase

Figure 2: TCP-Aix state chart.
3.1 Fast Recovery-based Error Recovery

In this section we will describe the parts of the TCP-Aix sender algorithm which relates to reordering. It covers the left side of Figure 3. In order to maintain performance in a reordering environment the behaviour when receiving dupacks should be similar to the behaviour in the Non Loss Recovery Phase, that is when the acknowledgements arrive as expected.

Starting in Non Loss Recovery, the TCP Aix sender enters the Dupack phase upon receiving the first dupack and initialises cwnd_aix. cwnd_aix is an estimate of the size that cwnd would have had, if the sender had remained in the Non Loss Recovery Phase. When a dupack arrives cwnd_aix is increased as cwnd would have been increased upon the acknowledgement of a new segment. The management of cwnd_aix is called candid growth.

Each dupack that arrives below dupthresh also clocks out a new segment as in Limited Transmit [3]. The TCP-Aix sender thus obeys the rule of packet conservation, refraining from increasing its sending rate during this period. In cases of severe congestion the number of dupacks will be fewer, thereby the sending rate will decrease when necessary. The retransmission timer is also restarted on each dupack. This conservative timer management increases the number of options for setting dupthresh.

When the number of dupacks received reaches dupthresh a transition to Reordering Corroboration occurs and a fast retransmit is performed. The purpose of this phase is to corroborate that the fast retransmit was sent in response to a packet having been lost. The retransmission timer is not restarted after the retransmission has been made, since if the retransmission fails a timeout is required.

In case multiple segments have been indicated as lost in a window of data, more than one segment may be retransmitted in Reordering Corroboration. To detect if any of the original transmissions have been lost, the highest retransmitted packet is kept track of in the variable high_retr. Reordering Corroboration ends if a newACK was sent in response to a retransmission, or when all original transmissions for which a retransmission has been made have been accounted for.

If the loss is corroborated, i.e., an acknowledgement for the retransmit is received, Loss Recovery is entered. In the transition the recovery point is set and the congestion control state is adjusted in response to congestion as in [5], except for cwnd which is set to (ssthresh + 1) instead of to ssthresh. This is a compensation for the delayed entrance to Loss Recovery caused by TCP-Aix.

TCP-Aix takes a conservative position regarding timeouts that occur when the sender is in Loss Recovery. Instead of attempting to verify whether the timeout was genuine or not, an immediate reduction of cwnd is performed and Loss Recovery is terminated. The variable recover that was introduced to prevent multiple fast retransmits during one cwnd, see [19] for details, is used to force the sender out of Loss Recovery in this situation. The SACK option protects against congestion actions being taken for dupacks triggered by unnecessary retransmissions, since the SACK information makes it possible to disregard dupacks which carry no new information.
3.2 Timeout-based Error Recovery

A timeout that occurs either in the Dupack phase or in Reordering Corroboration will cause a transition to the Timeout Corroboration state, since a genuine timeout is regarded to be a stronger congestion signal than a set of dupacks. If the TCP-Aix sender is in Non Loss Recovery when experiencing a timeout, $cwnd_{aix}$ will be initialised when transitioning to Timeout Corroboration.

While in Timeout Corroboration, packet conservation is followed until it has been corroborated that the timeout was caused by a packet being lost. Each subsequent timeout will trigger a new retransmission. If the timeout was genuine $cwnd$ is set to $(LW + 1)$, where $LW$ is the TCP loss window. This is the size $cwnd$ has after a successful retransmission.

3.3 Candid growth and $cwnd$ compensation

In order to make TCP performance more robust to reordering and delay variations, appropriate congestion window management needs to be considered carefully.

Both after a spurious fast retransmit and a spurious timeout TCP-Aix performs $cwnd$ compensation. It utilises $cwnd_{aix}$, which is an estimate of the size that $cwnd$ would have had, if the sender had remained in the Non Loss Recovery Phase. As such, $cwnd_{aix}$ is the target value of $cwnd$ following a spurious timeout or fast retransmit. Setting $cwnd$ directly to $cwnd_{aix}$ at the end of a corroboration phase may cause a large burst of segments to be sent out. To avoid this $cwnd$ is set to $\min(cwnd + 1, cwnd_{aix})$ and then the sender slow starts until $cwnd$ has grown to $cwnd_{aix}$. This is achieved by setting $ssthresh$ equal to $cwnd_{aix}$. Also when exiting the Dupack phase this compensation is made.

The reason for entering slow start after a reordering event is that only non-dupacks grow $cwnd$. Even at relatively low reordering rates few non-dupacks will be received when the bandwidth×delay product is large. To be able to utilise the bandwidth in the presence of reordering it is therefore necessary to take the dupacks into account in order to improve throughput.

If instead the newACK was sent in response to a retransmission, indicating true congestion in the network, a sending rate reduction according to [5] is carried out and either SACK-based [15] or timeout initiated loss recovery is entered.

4 Winthresh algorithm

In this section we describe a novel algorithm for calculating dupthresh called the winthresh algorithm. TCP-Aix can however be combined with any other scheme for setting dupthresh.

Traditional timer management, where the retransmission timer is not restarted upon reception of a dupack, limits the range of effective dupthresh values. This is because setting a high dupthresh value may result in a timeout. By restarting the timer on each dupack received during the Dupack phase as in TCP-Aix and using packet conservation, the range of usable dupthresh settings can be expanded. This added freedom is exploited in the winthresh design.
If the sender waits long before performing a retransmission, a new potential limitation is the available buffer space at each end of the connection. If the segment is truly lost, the left edge of the window will not advance until it is successfully retransmitted meanwhile new segments are being sent continuously. The higher `dupthresh` is, the larger is thus the demand on buffering and the higher the risk of window stalling.

The `winthresh` algorithm minimises the amount of spurious retransmits that a sender inserts into the network by waiting as long as possible before retransmitting while at the same time avoiding window stalling. Theoretically, if one packet is lost, the highest `dupthresh` which avoids window stalling can be computed as:

\[
\text{\textit{winthresh}} = (\text{\textit{usable\_win}} + 2) - (\text{\textit{flight\_size}} - 1) \nonumber \\
= \text{\textit{usable\_win}} - \text{\textit{flight\_size}} + 3. \tag{1}
\]

The `usable\_win` is the number of segments that a TCP sender could send at any given point (if permitted by its `cwnd`) until becoming receiver limited and `flight\_size` is the amount of currently sent but not acknowledged data as defined in [5]. If the send buffer is smaller than the receive buffer, the available send buffer space should be used to calculate the `usable\_win` in Equation 1 instead.

From the arrival of the first `dupack` until the `newACK` arrives, (`usable\_win` + 2) `dupacks` will reach the sender. There may be as many segments in the network as allowed by the `usable\_win` size. One acknowledgement will trigger the fast retransmit and will thus not advance the right window edge and one acknowledgement is the response to the fast retransmit, which will release the left window edge. From the time that the fast retransmit is sent until the associated acknowledgement arrives, (`flight\_size` - 1) acknowledgements will reach the TCP-Aix sender. Each of these `dupacks` should be able to release a new data segment, thus there has to be enough space left in the receiver buffer to allow for the new segments to be received. The lost packet is the reason for the reduction of the `flight\_size`. Flight size is reduced by the number of segments lost, therefore a higher `winthresh` is possible for more severe congestion events.

Note that this analysis is specific to TCP-Aix, since TCP-Aix allows data to be transmitted upon each `dupack` that arrives. It would however be straightforward to make a corresponding analysis for any existing or future TCP flavor.

### 4.1 Limiting winthresh

There are situations when the receiver advertised window, `rwnd`, may not be appropriately set in relation to the network path, or `cwnd` may be small in comparison to `rwnd`. In these cases it may take several RTTs to reach the calculated winthresh and to trigger a fast retransmit in case of a genuine loss event, since the winthresh algorithm strives to wait as long as possible before retransmitting. This is illustrated in Figure 3, where it takes 6 RTTs before the retransmission is triggered. Initially the flow is in Non Loss Recovery, but after 1.4 s a segments is lost. The TCP-Aix sender thus enters the Dupack phase, when the `dupacks` start to arrive at 1.5 s. It stays in this phase until the retransmission is performed at the time 2.7 s. The acknowledgement of the retransmission that acknowledges almost all the segments in flight arrives 0.1 s later. The available buffer space at the receiver is shown in the lower figure,
which confirms that all space is being used while the sending rate is kept until the arrival of this newACK.

While in the Dupack phase $cwnd$ is static. In dynamic and high bandwidth scenarios it is however important to be able to grow $cwnd$ quickly in order to utilise the available capacity fully. This may not be possible with prolonged dupack phases, which will result from a high dupthresh setting. To improve control over the duration of the Dupack phase we introduce an upper bound for the calculated winthresh. By applying an upper bound we also get a more predictable behaviour and better performance, since the size of $cwnd$ at the time when we enter the Dupack phase becomes less significant.

Since TCP congestion control operates on an RTT scale, expressing the bound in RTTs is in line with the current semantics. Each RTT, approximately a send window, $swnd$, of segments will be sent since each dupack clocks out a new segment. The bound can thus be expressed as a multiple of $swnd$, where $R$ is a configurable parameter corresponding to the maximum number of RTTs to wait before performing a fast retransmit. In the case of severe packet loss, packet conservation will increase the chances of reaching dupthresh, but it may take longer than $R \times RTTs$. The complete wintresh algorithm is summarised in Equation 2, where $std_{dup}$ is the standard dupthresh value of 3 [5].

$$dupthresh = \min\{R \times swnd, \max\{wintresh, std_{dup}\}\}$$ (2)

Assuming that the advertised window is not a limitation, the speed with which TCP-Aix reacts to congestion can be adjusted by adapting the upper
bound, $R$, placed on the winthresh algorithm. The issues that must be considered when configuring $R$ are explored next.

In Figure 4(a) an illustration of the history of $cwnd$ with TCP-Aix in a single user scenario with a fixed capacity is shown. In case of a real loss, the delay of the congestion response is the time from $std_{dup}$ $dupacks$ have been received until the newACK arrives. By adjusting $R$, the time until the retransmission is made can be controlled.

During a long $Dupack$ $Phase$, the bottleneck buffer will be kept full and it is likely that there will be more than one segment loss if there are slight variations in the capacity or the distance between segments. The longer the $Dupack$ $Phase$, the higher the likelihood of these variations occurring is. Therefore, TCP-Aix does not increase its sending rate in this state.

Most congestion events in the Internet are considered to be transient if there is a high degree of statistical multiplexing. In this case losses do not occur when $cwnd$ reaches a certain size, but at certain points in time. If TCP-Aix stays long in the $Dupack$ $phase$ the transition to $Loss$ $Recovery$ will be postponed, therefore there will be less time without losses during which $cwnd$ can grow after exiting $Loss$ $Recovery$. This is illustrated in Figure 4(b), where the standard TCP users grabs bandwidth faster than TCP-Aix after a real loss.

In order to evaluate the effect of different upper bounds on the winthresh algorithm, we have used a square-wave oscillating bandwidth scenario from [7]. It represents an extremely dynamic environment without reordering. In the model, the bandwidth oscillates between 5 and 15 Mb/s with the frequencies given on the x-axis of Figure 5. There are five flows of each type sharing a common bottleneck. The scale of the y-axis is the share of the bandwidth utilised by a flow in relation to its fair share. The advertised window is set to 10000 segments, which means that the upper bound on the winthresh algorithm will determine when fast retransmits are made.

The results are shown in Figure 5. Each dot corresponds to a flow during a simulation, and each column to a simulation setting. In this scenario the end-to-end RTT is approximately 50 ms. TCP-Aix is slower to adapt to changes in the bandwidth when the upper bound is high. When $dupthresh$ is limited by ten times $swnd$, there is a significant decrease in the throughput compared to standard TCP. In order to maintain the ability of standard TCP to quickly adapt to dynamic conditions these results thus indicate that an upper bound of
one to two swnds is reasonable. TCP-Aix has an additional margin of one RTT to the time contributed by the dupthresh setting.

In Figure 5(d) the drop rate for the different time frequencies are shown. At high frequencies, the bandwidth changes from almost one RTT to another. During the relatively short periods of lower bandwidth the buffer space is not exceeded. As the duration of the periods with 5 and 15 Mb/s becomes longer, the sender has time to adapt to the higher bandwidth and loses more segments when the lower bandwidth period is no longer short. After the peak in the drop rate, we start to see the effect of switching the bandwidth less often, which contributes to the decreasing drop rate.

TCP-Aix has similar drop rate characteristics as standard TCP also when its upper bound is ten swnds. The peak in the drop rate is lower, since TCP-Aix is passive for several RTTs before starting to increase its sending rate after a reordering/congestion event. When the bandwidth is decreased, the overshoot is thus smaller. We conclude that TCP-Aix does not influence the drop rate negatively.

4.2 Towards the end of a transfer

An additional situation to consider is the appropriate behaviour when there is no more new data to be sent upon the receipt of a dupack and the number of dupacks received so far is less than dupthresh. In this case TCP-Aix enters the Corroboration state in its current paradigm (if not already in this state) and retransmits the missing segments in the reported receive sequence, one after the other, without waiting for dupthresh dupacks. This part of the winthresh algorithm only influences performance towards the end of a flow and thus has the largest impact when most transfers are relatively short. For readability, this part of the TCP-Aix algorithm is not included in the state diagram presented in Figure 3.

In comparison to the Early Retransmit proposal [2], the winthresh algorithm is more aggressive in determining when to enter loss recovery, since it only considers whether there is new data available or not. Early Retransmit, which aims to improve the behaviour of standard TCP when approaching the end of the flow, adapts dupthresh to the actual number of segments in flight. The amount of outstanding data should be less than 4*SMSS bytes before lowering dupthresh.

5 Evaluation methodology

This initial study of TCP-Aix has been conducted in a simulation environment, where factors can be easily varied and controlled. We are foremost interested in environmental factors, but also of varying the configuration of the TCP stack. Simulations allow us to quickly explore a large number of scenarios.

For a new TCP flavor to be widely accepted it has to offer a performance gain over existing solutions, while at the same time sharing the network resources acceptably with many kinds of TCP traffic. It also has to perform well over a wide range of network conditions. The simulation results that we present relate to these aspects where reordering is created by a persistent wireless link layer without in-order delivery.
Figure 5: Standard TCP and TCP-Aix sharing a bottleneck when the bandwidth oscillates between 5 and 15 Mb/s.
5.1 Simulation Environment

We have implemented TCP-Aix and TCP-NCR in the Network Simulator version 2.27, ns-2.27 [28]. The code for TCP-DCR, which builds on tcp-sack1, was retrieved from [9]. The existing TCP agents, tcp-sack1 and tcp-dcr, were updated to follow [5], [3], [15], [23]. Appendix A gives a short summary of the changes.

There was no prior implementation of TCP-NCR, therefore we have compared our results against those received for TCP-DCR. We chose to implement Aggressive Limited Transmit for TCP-NCR. Our code is available at [25]. All simulations have been run with the old versions of the tcp-sack1 and tcp-dcr code, as well as with our new versions. The differences are small in most cases.

In Figure 6 the simulation topology used in Section 7 is shown. The characteristics of the link closest to the receiver is varied. The error model is also inserted on this link and it models the functionality of the Radio Link Control, RLC, protocol as well as determines which packets that are corrupted. The error model is a modified version of the model from [9]. We have added an option which corrupts first time transmission attempts and retransmissions with equal probability. It is also possible to configure a constant number of retransmissions to be performed. The cost of each retransmission is an additional delay corresponding to the RTT of the link. This simplified model does not take bandwidth in demand other than for the successful transmission. The distribution of the errors in the simulations was uniform and the random number generators for the error model has been seeded differently in each repetition of the same scenario.

5.2 Parameter settings

In ns-2.27, Limited transmit [3] is enabled by default. The minimum RTO in version 2.27 is 1 s, but we used 200 ms as is the default value in later versions of ns. Earlier simulation studies with modified dupthresh algorithm, thus had a much higher minimum RTO, decreasing the risk of timeouts [11], [33], [13].

Furthermore, timerfix, timestamps and ts_reset_RTO were all enabled. Throughout the simulations the packet size was 1000 bytes. To support congestion window validation [23], we enabled control_increase and introduced a condition which only allows cwnd to be increased if the entire cwnd is currently used. The old code only checked if the current swnd was being used.

Unless otherwise stated, the bottleneck queue size is set to 2.5 times the bandwidth-delay product, where the RTT is assumed to be 50 ms. When RED is used, the minimum and maximum thresholds are set to 0.25 times and 1.25 times
the bandwidth×delay product. Previous studies have shown that unfairness is
more pronounced with drop-tail [20], [6], therefore we will present results for
both queue management strategies but with a focus on drop-tail.

6 Impact on current
network traffic

In this section we evaluate the steady state properties of TCP-Aix and present
additional results on its dynamic behaviour. We consider the throughput,
smoothness and fairness in comparison to standard TCP. The impact on queuing
delays and drop statistics is also investigated.

6.1 Steady state behaviour

The simulation scenarios for the steady state evaluation are similar to those
used in [20] to evaluate TFRC performance. The topology is shown in Figure 7.
Each sender and receiver has a separate link through which they connect to
a link common to all connections. The shared link is the bottleneck of the
system. The RTT is around 50 ms with small differences between connections.
The reverse traffic consists of four TCP flows, whose advertised windows are
limited to at most 20 segments.

6.1.1 Long-term fairness

The first scenario has been chosen to assess the long-term fairness TCP-Aix
shows towards standard TCP when they are sharing a bottleneck link in a wired
network without reordering. Since the advertised window was set to 10000, the
upper limit of the winthresh algorithm controlled the value of dupthresh for
TCP-Aix.

There is a constant number of long-lived flows in each simulation of which
half the flows are TCP-Aix and the other half standard TCP. The bandwidth
and the number of flows are varied from simulation to simulation, as well as the
queueing strategy resulting in a perceived drop rate ranging from low to very
high 8(d). We measured the throughput of the long-lived connections during the
last 60 s of a 75 s long simulation. Each column of dots in Figure 8 corresponds
to a simulation and each dot represents a flow within that simulation.

In Figure 8 the normalised throughput of the TCP-Aix and standard TCP
flows are shown for a bandwidth of 15 Mb/s with RED queueing and proba-
ble settings of R. We also repeated the scenario for R set to 10, to show the
behaviour in an extreme case. As the the upper limit is increased TCP-Aix becomes less aggressive than standard TCP, but the loss rate does not seem to be adversely affected. The results for 8 Mb/s are similar.

6.1.2 Fairness and smoothness over different time scales

The throughput comparison shown in Figure 8 is based on a relatively long time interval, i.e., 60 seconds. The equivalence ratio described in [20] can be used to verify that TCP-Aix is fair towards standard TCP and towards flows of the same type over shorter time scales. Perfect fairness gives an equivalence ratio of 1 and the worst case result would be 0.

In short, each simulation is partitioned into time intervals of length $\delta$ and the sending rate in every time interval computed. The equivalence ratio in a time interval for user A and B is then defined as the minimum of the two ratios; $\frac{\text{sendrate}_A}{\text{sendrate}_A}$ and $\frac{\text{sendrate}_B}{\text{sendrate}_B}$. By taking the minimum of these two ratios, a value between 0 and 1 is received for each interval. The average of these values over all the time intervals yields the equivalence ratio, which is an estimate of how the bandwidth has been distributed on the time scale $\delta$.

The equivalence ratio of TCP-Aix is almost identical to that of standard TCP over all time scales for an upper bound of one or two send windows. In Figure 6.1.2 the equivalence ratio for $R=10$ is shown, which indicates a slight deterioration for longer time scales. The larger the upper bound on dupthresh is, the more sensitive is TCP-Aix to the congestion window size at the time when a segment loss occurs and the longer it waits to increase its sending rate after a congestion event.

Similarly, the coefficient of variation of the sending rate for a particular flow can be computed by considering the sending rate it has achieved in each time interval a sample. The coefficient of variation is then the standard deviation of these samples divided by the mean. This metric can be used to measure the variability in the sending rate, a property commonly referred to as the smoothness of a flow. A low coefficient of variation means that the flow is sending data at a steady rate.

TCP-Aix exhibits the same characteristics as standard TCP for dupthresh bounds in the order of one to two swnds as expected, since packet loss is due to congestion and not reordering. From Figure 6.1.2, we can also conclude that high dupthresh settings lead to changes in the smoothness, where R=10 is regarded to be a high bound.

6.1.3 Effects on queue dynamics

Slowly-responsive congestion control algorithms reduce their sending rate slower than standard TCP. TCP-Aix can be considered to be slowly responsive since it delays the congestion control decision. Whether this behaviour increases the drop rate or negatively influences the link utilisation or delays at the bottleneck router has been investigated. We considered one scenario with only a few number of flows sharing the bottleneck link and one with a higher degree of statistical multiplexing. The metrics studied include link utilisation and drop rate over the entire simulation, average and variations in queueing delay over time, but also over the simulation as a whole.
Figure 8: Equal shares of TCP-Aix and standard TCP flows competing. The queueing strategy is RED and the upper `dupthresh` limit is 1, 2 and 10 `swnds`.
Figure 9: Fairness measured by the equivalence ratio and smoothness measured by the coefficient of variation. Bottleneck bandwidth 15Mb/s shared by 16 TCP-Aix and 16 standard TCP flows.

The simulation time was 200 s and flows were starting up during the initial 20 s. In addition short-lived standard TCP flows take up approximately 20% percent of the bottleneck capacity. We observed the link utilization and drop-rates reported in Table 6.1.3 when 40 flows were competing. Both link utilization and drop rates are similar for standard TCP and TCP-Aix with low bounds. The differences when there are four flows sharing a 1.5Mb/s link are slightly larger. The other investigated metrics also strengthen the conclusion that TCP-Aix does not have a negative impact on queueing.

6.2 Dynamic behaviour

Figure 5 shows the throughput achieved when the bandwidth is oscillating. We are also concerned with the loss rates when the bandwidth rapidly decreases. As in [7] we investigate the transient behaviours. In Figure 10 the drop rate for a scenario where twenty flows of the same type are competing over a single bottleneck with a CBR source is shown. At time 150 s the CBR source stops and restarts again at 180 s. When active it generates data corresponding to half the bottleneck bandwidth. We used both RED and drop-tail queue management. The peak in the drop rate is more pronounced with drop-tail and we therefore show the results for this queue management technique. The highest peak for TCP-Aix is observed for an upper bound of one, since TCP-Aix is more aggressive with a lower bound.

A similar scenario, where the bandwidth reduction is due to a flash crowd of
Figure 10: Effects on the drop rate and the link utilisation, when first rapidly increasing the available bandwidth and then decreasing it when a CBR source is turned off and on. Only the drop rate for TCP-Aix with R set to one is shown.
<table>
<thead>
<tr>
<th>Flavor/Queue</th>
<th>Drop rate</th>
<th>Utilisation</th>
<th>Average</th>
<th>Stdev</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP/RED</td>
<td>0.0474</td>
<td>0.996</td>
<td>0.0321</td>
<td>0.00857</td>
</tr>
<tr>
<td>Aix-1/RED</td>
<td>0.0475</td>
<td>0.994</td>
<td>0.0323</td>
<td>0.00967</td>
</tr>
<tr>
<td>Aix-2/RED</td>
<td>0.0453</td>
<td>0.996</td>
<td>0.0308</td>
<td>0.0102</td>
</tr>
<tr>
<td>Aix-10/RED</td>
<td>0.0395</td>
<td>0.997</td>
<td>0.0269</td>
<td>0.0127</td>
</tr>
<tr>
<td>TCP/DT</td>
<td>0.0331</td>
<td>0.999</td>
<td>0.114</td>
<td>0.0116</td>
</tr>
<tr>
<td>Aix-1/DT</td>
<td>0.0334</td>
<td>0.999</td>
<td>0.114</td>
<td>0.0128</td>
</tr>
<tr>
<td>Aix-2/DT</td>
<td>0.0333</td>
<td>0.999</td>
<td>0.114</td>
<td>0.0126</td>
</tr>
<tr>
<td>Aix-10/DT</td>
<td>0.0269</td>
<td>0.999</td>
<td>0.117</td>
<td>0.00795</td>
</tr>
</tbody>
</table>

Table 1: Statistics for the bottleneck link with 40 long-lived flows and a bottleneck of 15Mb/s over a 200 s period. The average and stdev estimate the delay experienced.

1000 TCP connections starting, all producing 10 segments each, has also been simulated, see Figure 11. The starting point of the short TCP connections were evenly spaced over a 5 s interval. We studied the aggregate throughput of the flash crowd and the long-lived flows respectively and found that the differences compared to the corresponding throughputs with standard TCP were minor.

### 6.3 Co-existence in a reordering environment

For completeness, we present simulation results for a scenario when standards-compliant TCP and TCP-Aix senders are competing over a link which reorders segments. When there are both reordering and congestion, TCP-Aix will observe another loss rate than standards-compliant TCP since it is able to separate between reordering and segment loss. Therefore we have looked for any throughput degradation for TCP when TCP-Aix is gradually introduced in a reordering environment. In cases where the TCP-Aix sender gets a higher throughput, it is important that this increase in performance does not come at the expense of other types of senders.

First the average throughput of 50 standards-compliant TCP flows sharing a link is measured, thereafter we replace an increasing share of these standards-compliant TCP flows with TCP-Aix and estimate the average throughput of each type of flow. The probability of a transmission being corrupted is 1% and the bottleneck bandwidth is 5, 25 or 100 Mbps. At both 5 Mbps and 25 Mbps the bottleneck is fully utilized, but the window size which the TCP version is required to operate at is much smaller with 5 Mbps which makes loss recovery more intricate. In the final case only a share of the bandwidth is required to serve the standards-compliant TCP traffic. Each flow has its own overdimensioned access link. The RTT is approximately 100 ms for each flow, where 80 ms is the RTT of the bottleneck. This simulation strategy has previously been used in [18].

In Figure 12 the average throughput as a function of the share of TCP-Aix flows is shown. The confidence intervals are based on the throughput of the flows of the same type in one simulation. From the figure, we see that in all simulated scenarios, the throughput of the TCP senders is unaffected by the proportion of competing TCP-Aix senders. Instead, TCP-Aix is merely using otherwise unused network capacity in order to improve its performance. For
Figure 11: Effects on the drop rate and the link utilization, when first rapidly increasing the available bandwidth and then decreasing it. The competing traffic consist of 1000 short-lived TCP flows.
Figure 12: Fairness in an environment with both reordering and congestion.
instance a 20% share of TCP-Aix flows increases the utilization to 95% from 51% which was the utilization without TCP-Aix. The utilization is 98-99% for all mixes of traffic when the bottleneck is 5 or 25 Mbps.

We conclude that the better performance of TCP-Aix, does not come at the expense of standards-compliant TCP. When there is available capacity, which would not be utilised otherwise, TCP-Aix grabs this capacity. As a consequence the link utilisation is increased. These results show that an incremental deployment of TCP-Aix would be possible.

7 Performance gains

We have already shown that introducing TCP-Aix may improve link utilisation in the presence of reordering, when there is a bulk of standard TCP flows. In this section we evaluate the robustness of TCP-Aix and compare its performance against that of both standard TCP and TCP-NCR. We consider reordering caused by a persistent link layer retransmission scheme. The reordering rate, bandwidth and delay over the reordering link have been varied. We have also studied the performance of TCP-Aix and other TCP flavors combined with Eifel [26] in the presence of delay spikes.

The results presented in this section are for an upper bound on the winthresh algorithm of two CL/winths and with an unlimited advertised window. The results are based on ten repetitions in all cases, except for the performance study with delay spikes in Section 7.2, for which 30 repetitions were performed. The confidence intervals are for 95% and the assumption of the samples belonging to a normal distribution was verified. Each simulation run lasted 200 s.

The topology is shown in Figure 6. The characteristics of the link which connects to the receiver, are varied. In most scenarios we observe one long lived flow. Each scenario has however been repeated with background traffic consisting of acknowledgements for the reverse traffic and a mix of short (producing ten segments) and medium sized (producing fifty segments) standard TCP flows corresponding to approximately 10% of the capacity in total. The background traffic cause packet losses and variations in the RTT, which are not self-inflicted. The results presented in Section 6.3 on reordering and congestion explored the performance in a higher multiplexing scenario for TCP-Aix and standard TCP.
Table 2: The factors used to evaluate sensitivity to reordering.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reordering length</td>
<td>1 or 3 retransmissions á 40ms</td>
</tr>
<tr>
<td>Reordering rate</td>
<td>0.5, 3 or 8%</td>
</tr>
<tr>
<td>Pipe capacity</td>
<td>60ms×1Mbps or 60ms×100Mbps</td>
</tr>
</tbody>
</table>

7.1 Sensitivity to reordering

We have varied three factors; reordering length, reordering rate and pipe capacity. Their settings are described in Table 2. The RTT of the entire path is 60 ms, whereas the reordering lengths are 40 and 120 ms respectively. The reordering rate is the share of segments that are reordered.

From Figure 13(d) we can draw the conclusion that TCP-Aix can handle a reordering length which is twice the path RTT, even when the pipe capacity is large. TCP-NCR deals well with reordering events caused by one retransmission, Figure 13(a). This is however not sufficient to provide high throughput when the reordering events are frequent and the pipe capacity larger, which is shown in Figure 13(c). In this case the candid growth and cwnd compensation mechanisms included in TCP-Aix give it better throughput performance compared to TCP-NCR. Even at moderate reordering rates and a small pipe, standards-compliant TCP suffers a great throughput loss as illustrated in Figure 13(a).

We conclude that TCP-Aix has the desired robustness to high reordering rates and long reordering lengths. The importance of dealing with the increased amount of dupacks and the decreased number of non-dupacks is also clearly visible from the results for the large pipe capacity.

7.2 Delay spikes and varying reordering rates

TCP-Aix includes mechanisms similar to those of TCP-Eifel for handling spurious timeouts. When there is no reordering there is essentially no difference between how delay spikes are handled, except that TCP-Aix does not restore the congestion control state since it did not reduce the sending rate in the first place. When there is a reordering event during the corroboration of a timeout the dupacks will trigger new data transmissions using TCP-Aix, whereas no data will be sent by TCP-Eifel.

The intention in this case is not to model a realistic delay distribution, it is rather to verify the operation of the protocol under severe conditions. The interval between delay spikes and the duration of the spikes are normally distributed with a mean of 10 s and 3 s respectively. The standard deviation is one fourth of the average. The throughput has been computed only over the time when the link was actually in operation, i.e., we have reduced the simulation time by the aggregated duration of the delay spikes when computing the throughput. The reordering rate in this case is the probability of a transmission being corrupted.

The higher the bandwidth, the larger is the performance penalty associated with the spurious timeout. The bandwidth in this simulation scenario was 5 Mbps over the bottleneck and the RTT over the last, reordering, hop was 40 ms. The results are shown in Figure 14. Although the Eifel algorithm helps to detect and restore the congestion control state after a spurious timeout, the
Throughput (Mbps) vs. Reordering Rate (%)

(a) Bottleneck bandwidth 1 Mbps, one retransmission. The graphs for TCP-Aix and TCP-NCR overlaps.

(b) Bottleneck bandwidth 1 Mbps three retransmissions.

(c) Bottleneck bandwidth 100 Mbps, one retransmission.

(d) Bottleneck bandwidth 100 Mbps, three retransmissions. The graphs for TCP-NCR and TCP overlaps.

Figure 13: Sensitivity to reordering.
Figure 14: Performance in the presence of delay spikes when the reordering rate is varied.

performance for moderate and high error rates is dominated by the ability of the TCP flavor to detect reordering events.

We ran complementary simulations at 5 Mbps without introducing delay variations. For standards-compliant TCP the results with and without delay variations are in parity, indicating that the limiting factor is the response to reordering. TCP-NCR-Eifel with delay variations performs slightly worse than TCP-NCR under stable delay conditions for a reordering rate of 8%. TCP-Aix exhibits good robustness over the entire range of investigated reordering rates and performs better than TCP-NCR-Eifel in the presence of frequent reordering. The performance of standards-compliant TCP is poor.

7.3 Sensitivity to parameters

Most of the results shown in this paper are for an unlimited receiver buffer. If $\text{dupthresh}$ is higher than the standard $\text{dupthresh}$ of three, then the buffer requirement during loss recovery is also larger, i.e., approximately one $\text{snd}$ per RTT that loss recovery is delayed. In addition to the results we have presented, we have also investigated advertised window sizes corresponding to two times the bandwidth $\times$ delay product of the network path and even smaller.

With limited buffer space at the end-points, TCP-Aix will tolerate a shorter reordering length because the fast retransmit will be performed earlier with the winthresh algorithm. In this case a reordering event may be falsely classed as a congestion event, if the retransmit reaches the receiver before the original transmission.

With the delayed acknowledgements algorithm [17] only every second segment is acknowledged as long as segments arrive in order. TCP increases its sending rate based on the number of acknowledgements received rather than on the number of segments acknowledged. In particular this affects the rapidness of the slow start increase. In order to keep the aggressiveness of the slow start mode when the delayed acknowledgements algorithm is used Appropriate Byte Counting (ABC) [1] can be applied.

The Internet file size distribution is often described as consisting of a majority of small transfers, but the distribution also has a long tail. For the short
flows the start up behaviour is essential. Assuming there is no reordering, the behaviour of standard TCP and TCP-Aix is identical until the first segment is lost. If the data stream ends while in the *Dupack phase* of TCP-Aix a retransmission will be triggered, in order to avoid a timeout. Standard TCP normally retransmits after three *dupacks* and may thus already have retransmitted the segment, in this case TCP-Aix might add to the transmission time by at most half an RTT in the worst case. If it was a reordering event, TCP-Aix will give a higher throughput than standard TCP as previously shown.

8 Discussion

TCP-Aix waits for the acknowledgement of the segment it suspects is lost before attempting to class the event as either reordering or congestion. This information is relatively easy to get, it does not require long-term observations of the network and the necessary support can be placed in the end-points. TCP-Aix is based on pre-existing TCP options and the Eifel detection algorithm. By taking advantage of the already existing support structure, the time until deployment can be shortened.

Only the TCP senders need to be modified. The TCP receivers’ participation is limited to echoing the timestamps carried as options in the segments according to the rules defined in [24]. However, to get the most out of TCP-Aix, a larger receiver (and sender) buffer than for standard TCP is required. The receiver buffer is largely empty except during loss recovery. For each RTT that we extend Limited Transmit, i.e., the upper bound on *dupthresh*, an additional window worth of data must be buffered at the receiver.

When designing a TCP flavor, it is usually necessary to find a suitable point from a performance perspective with regards to more than one potential scenario, since several events exhibit identical signs. In Figure 15 an approximation of the throughput cost of TCP-Aix when there is reordering is shown. In the ideal case the TCP sender would know that it is a reordering event and would continue as usual, this is illustrated by the thick continuous line. When the first *dupack* arrives, TCP-Aix enters the *Dupack phase* and deviates from the ideal behaviour, which is shown by the dashed line. Until the corroboration of the reordering event, *cwnd* is maintained through packet conservation. If the *newACK* corroborates that it was a reordering event, TCP-Aix slow starts up to the *cwnd*, it would have had at the end of the corroboration phase if it had followed the ideal behaviour. Being conservative during the corroboration period thus results in an approximate throughput cost compared to the ideal performance represented by the horizontally striped area.

If instead the reordering event had been mistaken for a congestion event, the approximate throughput cost would have been both the striped areas. By allowing *cwnd* to evolve during the corroboration period, the throughput cost could have been reduced. However, if it had been a real congestion event this behaviour would have increased the risk of more packets being lost. The cost of aggravating congestion is wasted network resources and possibly a more severe sending rate reduction.
9 Conclusions

In this paper we have presented TCP-Aix; a sender-side algorithm which decouples the loss recovery and congestion control actions of standard TCP. Through this separation TCP-Aix provides robustness to reordering lengths of one RTT and delay variations. To handle reordering lengths beyond one RTT, a higher dupthresh setting than the standard value is used in combination with TCP-Aix.

In addition to TCP-Aix, we have introduced an algorithm called the winthresh algorithm for computing dupthresh. It minimises the amount of spurious retransmits that a sender inserts into the network by waiting as long as possible before retransmitting a segment while at the same time avoiding window stalling.

The performance of TCP-Aix has been evaluated and compared to that of both TCP-NCR and a standards-compliant TCP sender. We found that TCP-Aix is able to maintain almost constant performance even in scenarios displaying high reordering lengths and frequencies. In such scenarios it clearly outperforms both TCP-NCR and standard TCP. This performance gain is achieved by the aforementioned splitting of congestion control and error recovery, as well as the introduction of the winthresh algorithm. However, performance gains are also seen in scenarios displaying only moderate reordering lengths of less than one RTT. This can be explained by the candid growth and cwnd compensation mechanisms of TCP-Aix, which allow it to better handle conditions with a large portion of dupacks.

Finally, we have also shown that the performance gain of TCP-Aix in the described scenarios does not come at the expense of other, competing non-TCP-Aix flows. Rather, the TCP-Aix sender utilise bandwidth which would otherwise have been left unutilised. We therefore argue that an incremental deployment of TCP-Aix would be possible.

10 Acknowledgments

Sara Landström wants to acknowledge the feedback received from Ulf Bodin on the presentation of this report.
References


A Simulator changes

The simulations in this paper have been carried out in ns-2.27. The description of the changes we have made thus applies to this version. The TCP agent which most closely matched \cite{5}, \cite{15}, \cite{3} and \cite{23} was the tcp-sack1 agent. We therefore chose it as the starting point for a standards-compliant agent. As previously stated, the changes are relatively small and mostly affect corner cases.

There are two scoreboard implementations in ns-2. The tcp-sack1 agent used ScoreBoardRQ, but we changed it to use the ScoreBoard implementation instead since it was easier to understand. The modified tcp-sack1 agent has been named tcp-sackrfc.

A.1 Scoreboard management

The old code did not make use of SACK information carried on non-dupacks when not in fast recovery. The scoreboard was cleared when an acknowledgement that advanced the cumulative acknowledgement point was received in non-loss recovery, when exiting fast recovery and after a timeout. The only time the scoreboard should be cleared is directly following a timeout. The SACK information on any acknowledgement that arrives directly after a timeout is assumed to be valid. Also when exiting fast recovery the scoreboard should be maintained since it may contain information about segments above the recovery point.

A.2 Pipe calculations

The method \texttt{SetPipe()} described in \cite{15} has been implemented and replaces the old pipe computations. This also means that the scoreboard implementation had to be changed such that it identifies the segments which are to be considered lost according to the definition of \texttt{IsLost()} in \cite{15}. The old pipe calculations assumed that for each acknowledgement received during loss recovery, a segment had left the network. This is usually correct unless acknowledgements are lost or segments arrive in order at the receiver before having left fast recovery when the delayed acknowledgement algorithm is enabled. In the case of delayed acknowledgements, two segment at the time may be acknowledged if they arrive in-order.

A.3 Identifying a segment for retransmission

Previously the first hole in the acknowledged sequence of segments which had not yet been retransmitted was retransmitted. We have implemented \texttt{NextSeg()} from \cite{15}, it is however called \texttt{GetNextRetran()} in the implementation, to identify the segment to be retransmitted next.
A.4 Partial acknowledgements

Sending data on partial acknowledgements is to be considered experimental together with SACK and has not been used in this study. In TCP NewReno [21], it is stated that the next unacknowledged segment is to be retransmitted upon the receipt of a partial acknowledgement. In TCP SACK [15] segments are usually retransmitted when a certain number of segments that have been sent after it has been sacked.

Combining both the SACK and NewReno behaviour can cause a segment to be retransmitted more than once since it may be both the next unacknowledged segment and the segment identified for retransmission by SACK. The solution is to extend the NewReno rule to retransmit the next unacknowledged segment, which has not yet been retransmitted. We note that the current implementation does not fulfill this requirement, since it may be of use to others.

A.5 Timer management

The retransmission timer can be restarted on each acknowledgement that arrives which advances the cumulative acknowledgement point as specified in [31] it may also be restarted each time a retransmission is sent. We have not changed the timer management but it may be useful to know that is similar to the [31] behaviour.

A.6 Congestion window validation

By enabling the control_increase parameter in the tcp agent, a check of whether the send window is currently being used is executed before increasing cwnd further. However, if the user is currently limited by the available buffer space, the send window may be fully utilized although cwnd is not. Therefore we extended the check to also verify that the sending rate is in parity with cwnd.

A.7 Additions for TCP-NCR and TCP-Aix

The scoreboard implementation was also supplied with functionality for dynamically changing the desired number of dupacks before determining a segment lost or to be retransmitted, i.e., the dupthresh value.