On the TCP Minimum Retransmission Timeout in a High-speed Cellular Network

Mats Folke, Sara Landström and Ulf Bodin

Division of Computer Science and Networking
Department of Computer Science and Electrical Engineering
Luleå University of Technology
e-mail: {mats.folke,sara.landstrom,ulf.bodin}@ltu.se

***Abstract*: HS-DSCH is a high-speed shared radio channel extension for WCDMA which is used for cellular mobile telephony. The algorithm for distributing the channel resources together with the characteristics of the radio medium result in delay variations. The TCP minimum retransmission timeout interval has effectively alleviated delay variations in its range from deteriorating TCP performance. But recently, this bound has been shortened in modern widely spread TCP implementations. The aim of our study is to find out how a shorter minimum retransmission timeout affects TCP performance over HS-DSCH.**

We have implemented a model of HS-DSCH in the network simulator ns-2. Our simulations cover a wide range of different minimum retransmission timeout settings and loads, two types of schedulers (Round-Robin and Signal-to-Interference-Ratio (SIR) scheduling) and two versions of TCP (SACK and NewReno).

Our results show that the number of spurious timeouts increase with the load. The SIR scheduler causes fewer spurious timeouts than the RR scheduler for shorter minimum retransmission timeout settings, however, for longer settings the RR scheduler causes fewer spurious timeouts compared to the SIR scheduler. The minimum retransmission timeout has consequences for goodput fairness, but it does not affect the total system throughput. Both TCP versions produced similar results.

1. Introduction

The High-Speed Downlink Shared Channel (HS-DSCH) in Wideband CDMA (WCDMA) release 6 has theoretical peak bitrates for data services of 14 Mbps [15]. Moreover, delays considerably shorter than for other shared data channel technologies in previous releases of WCDMA are possible.

HS-DSCH is primarily shared in the time domain, where users are assigned time slots according to a scheduling algorithm that runs independently at each base station. The short Transfer Time Interval (TTI) of 2 ms enables fast link adaptation, fast scheduling and fast Hybrid Automatic Repeat reQuest (HARQ). The channel is designed for bursty Internet traffic, typical of web traffic.

TCP (Transmission Control Protocol) [17] ensures reliable transfer of HTTP traffic. Avoiding delay spikes is important to TCP. In particular, delay spikes may cause spurious timeouts, resulting in unnecessary retransmissions and multiplicative decreases in congestion window sizes as described in [11] and [9]. There are several mechanisms in HS-DSCH that can cause considerable delay variations appearing as delay spikes to TCP.

In HS-DSCH, the data rate depends on the Signal to Interference Ratio (SIR) of the receiving user. Consequently, fluctuations in the interference levels lead to delay variations. SIR is affected by path-loss, fading and interference from other transmissions. Schedulers aiming at optimising system throughput give precedence to the channel to users with high SIR. With a Round Robin (RR) scheduler the delay of an individual IP packet is determined both by the number of active users and by the SIR of the receiving user.

Using ns-2 [13] we evaluate the performance of TCP SACK [6, 12, 4] and TCP NewReno [7] for the RR and SIR scheduler respectively. Modern implementations of TCP have a lower minimum bound on the retransmission timer (minRTO) than the customary 1 second [18]. In [5] the authors discuss whether a minRTO of 1 second is too long for connections with RTTs much shorter than 1 second. The lower bound shields against spurious timeouts in its range and we therefore find it interesting to study different settings of the minRTO. In this paper we evaluate the sensitivity of TCP regarding the setting of this minimum bound and its impact on the number of spurious timeouts, goodput fairness and system throughput.

This paper is organised as follows. Section 2 consists of TCP fundamentals, such as describing the different algorithms used for congestion control. Our method is described in Section 3 followed by the results in Section 4. Section 5 contains a discussion of the results and the paper ends with conclusions in Section 6.

2. TCP fundamentals

In TCP the send rate is gradually increased and drastically decreased according to its congestion control and avoidance mechanisms, thus providing the link layer with an irregular flow of data. In the following discussion we assume that the congestion windows and not the receiving windows limit the TCP sources' sending windows and that HS-DSCH is the bottleneck.

Typically, a TCP source in slow start, begins by sending two to four segments [1] and then waits for the receiver to acknowledge them before releasing more data. The send rate is increased exponentially as long as the acknowledgements keep arriving in time. This results in TCP sources alternating between releasing bursts of data and being idle until they have opened their congestion windows enough to always have data buffered for HS-DSCH. For short transfers, TCP may never reach such a window size. When the first packet is lost, TCP leaves slow start and enters congestion avoidance, where the send rate is increased linearly.

When a new segment creates a gap in the receive

buffer (i.e. its segment number is not consecutive with respect to previous segments'), the receiver generates a duplicate acknowledgement indicating where the beginning of the first gap is. If three duplicate acknowledgements (ACKs) are consecutively received, the TCP source assumes that the bytes pointed at have been lost due to buffer overflow somewhere along the data path. The missing bytes are retransmitted and the congestion window is reduced to half its current size. This retransmission is called a *fast retransmit* [2].

For a fast retransmit to take place, at least three segments sent after the first lost segment must arrive at their destination and trigger duplicate acknowledgements. If segments at the end of a transfer are lost or multiple packet losses from a window occur, there might not be enough segments left to trigger a fast retransmit. The send window may also be too small to begin with. In such cases the TCP source must rely on its timeout mechanism for recovery. If the oldest segment is not acknowledged within a time frame, called the retransmit timeout (RTO), the TCP source starts over from one segment and re-enters the slow start phase. It then retransmits the presumably lost segment.

The sender continuously samples the round trip time (RTT) and adjusts the RTO, for more details on the TCP retransmit timer see [16]. The RTO is based on the mean RTT and a factor accounting for the fluctuations in the RTT. Traditionally, there has been a lower bound of 1 second on the RTO due to poor clock granularity. We will refer to this bound as the minRTO. The clock granularity has however improved and therefore some modern implementations have chosen to significantly reduce the lower bound. For instance Linux version 2.4 uses a minRTO of 200 ms [18] and the FreeBSD implementation does not have a minRTO. Instead a value corresponding to two times the clock granularity is always added to the RTT estimate [5].

The reduction or complete removal of the minRTO might have an impact on TCP performance over wireless links where the lower bound has shielded against delay spikes in the range of the lower bound. This fact and some of the problems associated with the current retransmit timer are discussed in [5]. The importance of the setting of minRTO on TCP performance was also emphasised in [3]. Their results show that the minRTO setting had larger impact on performance, than the timer granularity, how often RTT estimates were made and the settings of the low-pass filter used for RTT estimation.

The mentioned delay spikes can occur if the available forwarding capacity rapidly decreases and may cause the retransmit timer to expire prematurely. This event is denoted a *spurious timeout* and is explained in depth in [11].

With *Selective Acknowledgements* (SACKs) [6, 12], the receiver can inform the sender about all non-contiguous blocks of data that have been received. This way the sender knows which segments to retransmit. Without the SACK option the sender does not know exactly which packets are lost.

TCP NewReno [7] is the TCP variant recommended if one of the two communicating TCP end points in a session does not support the use of SACK. The NewReno algorithm is active during *Fast recovery*, i.e., from the receipt of three duplicate ACKs to a timeout or until all data sent have been acknowledged. In short, the NewReno algorithm considers each duplicate acknowledgement to be an indication of a segment leaving the network and therefore the sender is allowed to send a new segment on each duplicate acknowledgement. This variation of the TCP congestion recovery behaviour is more likely to keep the ack clock going during loss events than TCP Reno's, thereby avoiding a timeout. The difference pf NewReno compared to SACK-based loss recovery [4] is that the NewReno sender does not know where the gaps in the receive sequence are.

3. Method

The impact of different settings of the TCP retransmit timeout lower bound (minRTO) has been evaluated through simulations. In this section we introduce the simulation environment, thereafter the chosen evaluation metrics are presented.

3.1. Simulation Environment

A model of HSDPA has been implemented in the Network Simulator version 2.27 (ns-2) [13]. The radio model includes lognormal shadow fading with a standard deviation of 8 dB and exponential path loss with a propagation constant of 3.5. Self interference is assumed to be 10 percent and the interference from simultaneous transmissions within a user's own cell is approximated to 40 percent. Code multiplexing for up to three users in the same time slot for a given cell is supported. The interference from transmissions in other cells than a user's own cell is dampened through distance. The available coding and modulation combinations are accounted for in Table 1. Block errors are uniformly distributed. When SIR is less than –3.5 dB approximately every second block is received in error, for better SIR conditions the block error rate is 10 percent. No fast HARQ is implemented; instead, damaged radio-blocks are immediately retransmitted.

Coding (rate)	Modulation (type)	SIR (dB)	Bitrate (Mbps)	Size (bytes)
0.25	QPSK	–3.5	1.44	360
0.50	QPSK	0.0	2.88	720
0.38	16QAM	3.5	4.32	1080
0.63	16QAM	7.5	7.20	1800

Table 1: Link adaptation parameters. Size refers to the radio-block sizes.

When starting a simulation the mobile terminals are randomly distributed according to a uniform distribution for the x-axis and the y-axis on a cell plan consisting of seven cells. All cells have omnidirectional antennas and a radius of 500 m. The traffic sources are at equal distance from the base stations and the mobile users are associated with the closest base station. During a transfer a mobile node moves with a speed drawn from a low-speed mobility model [14]. All directions are equally likely to be taken when beginning a new transfer. Wrap-around

is supported both for the moving users and interference calculations.

A session consists of a user (mobile terminal) downloading a file followed by a waiting time drawn from an exponential distribution with a mean of 1.5 seconds. The waiting time is initiated as soon as the last data byte has reached the receiver. We reset the TCP endpoints, thus no teardown is performed, but connection establishment takes place for every flow. The file sizes are drawn from a Pareto distribution with a mean of 25000 bytes and the shape parameter set to 1.1 [14]. The mobile node is also moved to a new position each time it starts a new transfer. A simplified model of the topology can be found in Figure 1.

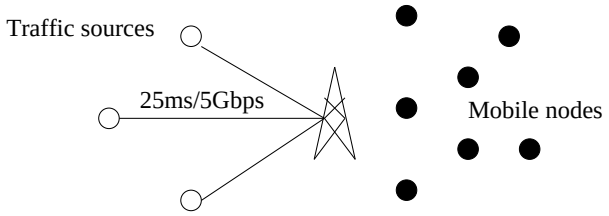


Figure 1: Simplified topology illustrating the connection between the traffic sources and the mobile nodes.

We vary the load by setting the total number of users for the simulation. Initial studies suggested that 20–50 users generate what can be regarded as low load, 50–100 users gave moderate load and above 100 users the load is high. We initially used a range of up to 500 users, but we settled on 150 users as a maximum. Above this point about 50% of the transfers had a goodput of less than 10 kbit/s, regardless of scheduler and minRTO. We consider 10 kbit/s to be too low for the web traffic that HS-DSCH is primarily designed for.

Each scenario is run 10 times with different initial values for the positions and velocities of the users as well as the starting times of the file transfers and their sizes.

3.2. Evaluation metrics

If the RTO is set too low the risk for premature timeouts is high. With timeout intervals larger than necessary the sender might be idle for periods waiting for the timer to expire. When studying the effect of different minRTO settings, we count the number of spurious timeouts on a per-flow basis. Although extended idle periods negatively influences the transfer rates of individual flows, they do not result in unnecessary retransmissions. We therefore consider spurious timeouts more destructive for system performance and focus on them for this study.

System throughput is defined as the total amount of transport layer data transferred during a given time period. This data includes transport layer retransmissions. System goodput is the system throughput with any replicated transport layer packets removed. For a mobile user, the goodput is the size of its transfer divided by the transmission time.

Users that have been struck by a spurious timeout perform transport layer retransmissions and reduce their send rate when the timeout occurs. We therefore expect spurious timeouts to exert an influence on the goodput observed by individual users and thus also on the goodput fairness. We use the fairness metric given by Equation 1, where x_i is the goodput experienced during a particular flow i to measure fairness. This metric was suggested in [10].

$$f(x_1, x_2, \cdots, x_n) = \frac{(\sum_{i=1}^{n} x_i)}{n \sum_{i=1}^{n} x_i} \quad (1)$$

The best setting of the minRTO, is the setting which maximises system goodput while maintaining fairness between the users. The results presented in this study are the average values with 90% confidence intervals for the ten simulation runs. The normality assumption that must be fulfilled to be able to compute confidence intervals has been verified.

4. Results

In Figure 2 we clearly see that a longer minRTO results in a smaller share of the flows suffering from spurious timeouts. We have also looked at the total number of spurious timeouts which agree with the results depicted. By comparing Figures 2.1 and 2.2 we find that the SIR scheduler causes fewer spurious timeouts for a shorter minRTO, however the RR scheduler is better (i.e. fewer spurious timeouts) for a longer minRTO. When looking at Figure 2.2 we see that most delay spikes do not last for more than 0.5 seconds, because for longer minRTOs the share of flows suffering from spurious timeouts is virtually zero.

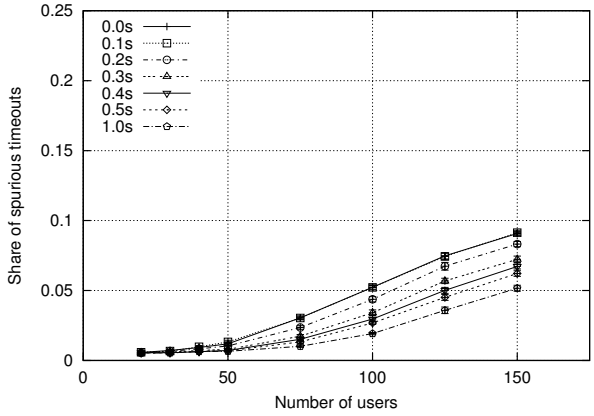
Different values of the minRTO do not result in any significant differences in goodput fairness, except when using an RR scheduler at high loads. For this case a longer minRTO is better than a short one, as shown in Figure 3. We believe that this decrease in fairness is the result of the increase in spurious timeouts. Comparing the two schedulers, we see that the RR scheduler produces slightly higher fairness than the SIR scheduler for moderate load. Regardless of scheduler and minRTO, the fairness steadily decreases as the load increases above 75 users.

Figure 4 presents the throughput for the whole system. For SIR scheduling, the different values of the minRTO do not result in any differences in throughput. We note a small difference in throughput when using an RR scheduler at high loads.

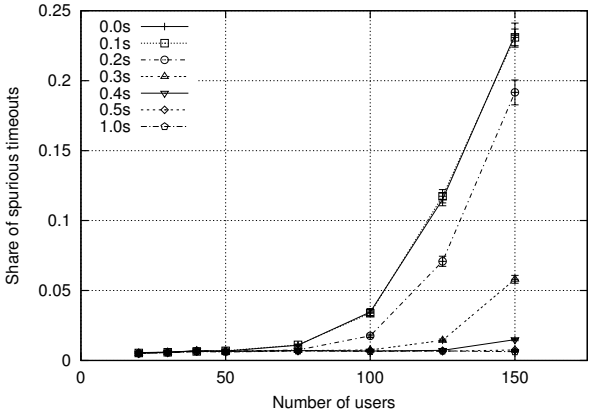
5. Discussion

The results presented raise a key question: Why is RR scheduling more sensitive to changes in the minRTO when compared to SIR scheduling? Spurious timeouts occur when the delay suddenly increases, such that a packet will be delayed causing the RTO timer to go off. In our system, increased packet delays are the result of intensified competition at the MAC layer.

With an RR scheduler the competition is intensified for all users whenever a new user arrives to a cell, since they all compete on equal terms. However, given the

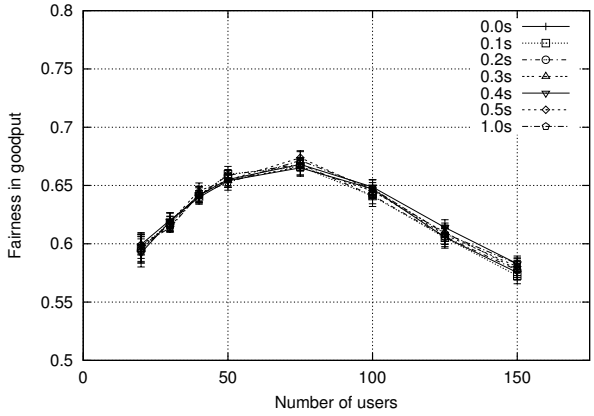


2.1: SIR scheduling

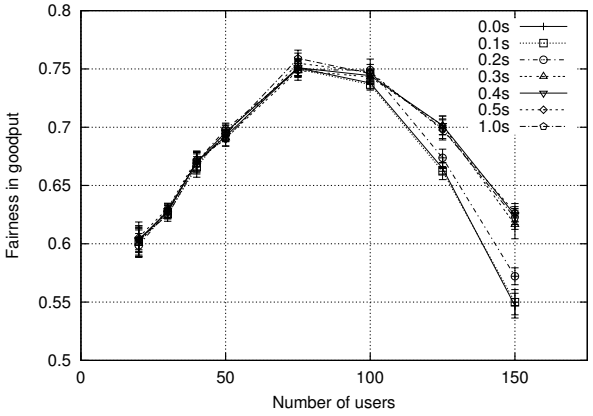


2.2: RR scheduling

Figure 2: The share of all flows experiencing at least one spurious timeout using TCP SACK and a specified scheduler for different values of minRTO. The confidence level is 90%. TCP NewReno gave similar results.

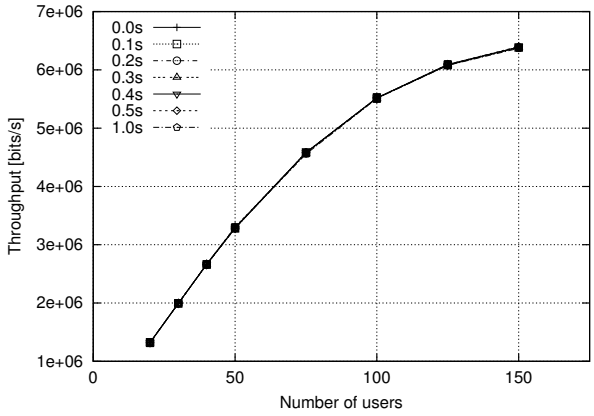


3.1: SIR scheduling

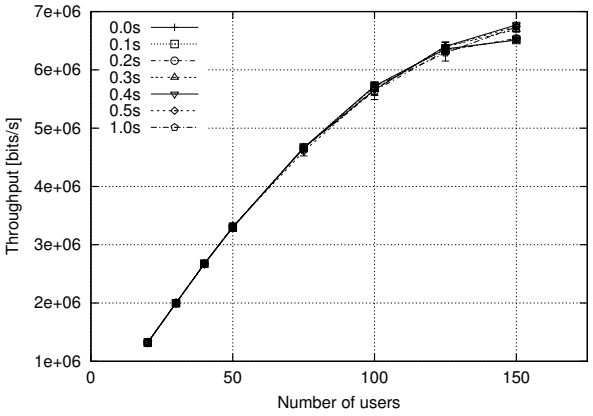


3.2: RR scheduling

Figure 3: The fairness in goodput among different flows using TCP SACK and a specified scheduler for different values of minRTO. The confidence level is 90%. TCP NewReno gave similar results.



4.1: SIR scheduling



4.2: RR scheduling

Figure 4: The total throughput in the system using TCP SACK, averaged over ten runs for different values of minRTO. The confidence level is 90%. TCP NewReno gave similar results.

slow start behaviour of TCP, the traffic of one new user is not enough to create a delay spike. There must be several new users arriving within a short period of time in order for any rapid increase in competition to occur. Because of the selected application model with its majority of small files the number of users needed are increased even further. With SIR scheduling the arriving users only compete with the users having worse SIR than themselves. This means that if a number of users arrive at a cell, the likelihood of all of them contributing to the competition observed by a particular user is less for SIR scheduling, than with RR scheduling.

The size of the files which the new users chose to download also affects the outcome. The files must be big enough to capture a large amount of time slots. In our application file size distribution, small files are common and large files rare. This makes it less probable that spurious timeouts would occur compared to a scenario where a majority of the file sizes are large.

The effect of our file size distribution is different for the two schedulers. If a high-SIR user arrives to a cell which uses SIR scheduling and begins to transfer a small file, other users are not affected, since the new transfer will use few time slots. On the other hand, if RR scheduling is applied, a new user would probably consume more time slots since the RR scheduler does not try to optimise system throughput.

From the results it is likely that the delay spikes when using the RR scheduler have a duration of less than 1 second. However, no such conclusion can be drawn when we use the SIR scheduler. It could be that delay spikes during SIR scheduling are longer than 1 second, thus a minRTO of 1 second will not be able to capture them. It could also be that because the SIR scheduler optimises system throughput it is able to process more data faster, thus actively reduce the load. If we would increase the load beyond the current point we speculate that the minRTO might have an effect for SIR scheduling.

We also made some preliminary studies of the average round trip times of packets in the system. It revealed that the average round trip time for the same number of users, is shorter for SIR scheduling than for RR scheduling, indicating that the SIR scheduler is more efficient. This is more evident during high loads. Furthermore, we have compared cumulative distributions of the RTTs for the two schedulers. In general the RTT for SIR scheduling is shorter, but there are several occurrences of really long RTTs compared to RR scheduling.

We have only studied TCP traffic since TCP probably will be the protocol used by most applications. However, there are ideas [19] suggesting that HS-DSCH could be used for Voice over IP (VoIP) which makes it interesting to study the performance of TCP when competing with traffic using a protocol which is not congestion-adaptive, i.e., UDP. A UDP traffic source in general may very well start sending at a high rate compared to the start-up behaviour of TCP. This means that a single, or a few high-rate UDP flows can cause sudden service interruptions manifesting themselves as delay spikes. These will be interpreted as packet losses by the TCP flows transferring data in the same cell.

6. Conclusions

We have, through simulations of the high-speed radio channel HS-DSCH, studied the effect of varying the lower bound of the TCP retransmission timer (minRTO) on system throughput, goodput fairness and spurious timeouts. Three parameters have been varied; scheduler type, length of the minRTO and load. Load is described as the number of users in the system.

We see that there are differences in the number of spurious timeouts when using the two schedulers for the minimum retransmission bounds studied and for our application model. These differences do not seem to have any major effect on fairness, goodput or throughput, nor do the two TCP versions. This suggests that the minRTO could be removed. For those users who suffer from spurious timeouts, solutions such as the Eifel algorithm [11, 8] are in the process of standardisation. Further studies may investigate access control and the effects of applications not adapting to congestion.

REFERENCES

[1] M. Allman, S. Floyd, and C. Partridge. “Increasing TCP’s Initial Window”. RFC Standards Track 3390, IETF, Oct. 2002.

[2] M. Allman, V. Paxson, and W. Stevens. “TCP Congestion Control”. RFC Standards Track 2581, IETF, Apr. 1999.

[3] Mark Allman and Vern Paxson. “On Estimating End-to-End Network Path Properties”. In *Proceedings of the ACM SIGCOMM Conference*, pages 263–274, Sep. 1999.

[4] E. Blanton, M. Allman, K. Fall, and L. Wang. “A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP”. RFC Standards Track 3517, IETF, Apr. 2003.

[5] Hannes Ekström and Reiner Ludwig. “The Peak-Hopper: A New End-to-End Retransmission Timer for Reliable Unicast Transport”. In *Proceedings of the IEEE INFOCOM Conference*, volume 4, pages 2502–2513, Mar. 2004.

[6] K. Fall and S. Floyd. “Simulation-based Comparisons of Tahoe, Reno and SACK TCP”. *Computer Communications Review*, 26(1):5–21, Jul. 1996.

[7] S. Floyd, T. Henderson, and A. Gurtov. “The NewReno Modification to TCP’s Fast Recovery Algorithm”. RFC Standards track 3782, IETF, Apr. 2004.

[8] A. Gurtov and R. Ludwig. “Responding to spurious timeouts in TCP”. In *Proceedings of the IEEE INFOCOM Conference*, volume 3, pages 2312–2322, Mar. 2003.

[9] H. Inamura, G. Montenegro, R. Ludwig, A. Gurtov, and F. Khafizov. “TCP over Second (2.5G) and Third (3G) Generation Wireless Networks”. RFC Best Current Practice 3481, IETF, Feb. 2003.

[10] Raj Jain. *"The Art of Computer System Performance Analysis"*. John Wiley & Sons Inc., 1991.

[11] Reiner Ludwig and Randy H. Katz. "The Eifel algorithm: making TCP robust against spurious retransmissions". *ACM SIGCOMM Computer Communication Review*, 30(1):30–33, 2000.

[12] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. "TCP Selective Acknowledgement Options". RFC Standards Track 2018, IETF, Oct. 1996.

[13] S. McCanne and S. Floyd. "The Network Simulator – ns-2". http://www.isi.edu/nsnam/ns.

[14] Motorola. "Evaluation Methods for High Speed Downlink Packet Accesss (HSDPA)". Technical report, 3GPP, Jul. 2000.

[15] Technical Specification Group Radio Access Network. "Medium Access Control (MAC) protocol specification (Release 6)". Technical Report TS 25.321, 3rd Generation Partnership Project (3GPP), Dec. 2004.

[16] V. Paxson and M. Allman. "Computing TCP's Retransmission Timer". RFC Standards Track 2988, IETF, Nov. 2000.

[17] J. Postel. "Transmission Control Protocol". RFC Standards Track 793, IETF, Sep. 1981.

[18] P. Sarolahti and A. Kuznetsov. "Congestion control in Linux TCP". In *Proceedings of the USENIX Annual Technical Conference*, pages 49–62, Jun. 2002.

[19] Haito Zheng, Gee Rittenhouse, and Michael Recchione. "The Performance of Voice over IP over 3G Downlink Shared Packet Channels under Different Delay Budgets". In *Proceedings of IEEE Vehicular Technology Conference*, volume 4, pages 2501–2505, Oct. 2003.