Enhancement of the TCP Module in the OMNeT++/INET Framework

Thomas Reschka, Thomas Dreibholz, Jobin Pulinthanath, Martin Becke, Erwin Rathgeb
University of Duisburg-Essen, Institute for Experimental Mathematics
Ellernstrasse 29, 45326 Essen, Germany
t.reschka@gmx.de, {dreibh, jp, martin.becke, rathgeb}@iem.uni-due.de

Abstract

The INET framework for the simulation tool OMNeT++ *provides a TCP module, which can be used for evaluating various Internet applications. However, the implementation of this TCP module has not been state of the art. Some important features of modern TCP implementations – particularly Selective Acknowledgements (SACK) and a complete Flow Control – have been missing. In this paper, we first introduce basic TCP mechanisms. After that, we introduce the extensions we have made to the TCP module of INET. Finally, we show some results of our performance evaluation.*[1][2]

Keywords: OMNeT++, INET Framework, TCP, Selective Acknowledgement, Flow Control

1. Introduction

The Internet is the largest computer network in the world. Most of the Internet traffic is handled by the Internet Protocol (IP) and the Transmission Control Protocol (TCP). The evolution of TCP is still ongoing and there exist a variety of different TCP algorithm variants. A suitable way to test communication protocols in a deterministic manner is by using a simulation environment. The Open Source tool OMNeT++ [16] is an appropriate candidate, which is popular for academic projects. The INET framework [17] for OMNeT++ includes IP-based simulations models e.g. IP, UDP, TCP, SCTP etc.. The focus of this paper is on the TCP module of INET. TCP options were not supported and important parts of the Flow Control were also missing. Our main goal has been the implementation and evaluation of the TCP Selective Acknowledgement (SACK) option. The SACK option is part of common TCP implementations and is – according to [6] – a "recommend enhancement" for TCP. The TCP module had also used an infinite receive buffer and the data receiver had never changed its offered receiver window size. Since Flow Control is an essential TCP mechanism and may influence the performance of the SACK algorithm, the completion of TCP's Flow Control implementation has been our secondary task.

2. The TCP Protocol

TCP, which has originally been specified in RFC 793 [11], is a reliable, connection-oriented transport protocol. Messages handled by TCP are called segments. All segments contain a TCP header and possibly user data. The header may be extended by options, causing an enlarged header and a reduced segment length. To activate options (e.g. SACK), both endpoints need to permit the requested option(s) during connection setup. In the literature the TCP Reno implementation is considered to be the standard TCP implementation. Therefore new TCP variants are often compared to Reno, e.g. in [7].

2.1. Selective Acknowledgement

When multiple segments are dropped, the original cumulative acknowledgement scheme of TCP may result in reduced connection throughput. [9] specifies the SACK option to solve this problem. An example which is depicted in figure 1 illustrates the SACK generation by the data receiver.

We assume that the data receiver has already received five segments of 1000 bytes each and has sent an acknowledgement (ACK) with the cumulative acknowledgement field set to 5000. The data sender sends

1 Parts of this work have been funded by the German Research Foundation (Deutsche Forschungsgemeinschaft).

2 The authors would like to thank Andras Varga and Rudolf Hornig for their review of the modified source code and the integration of our enhancements into the INET framework.

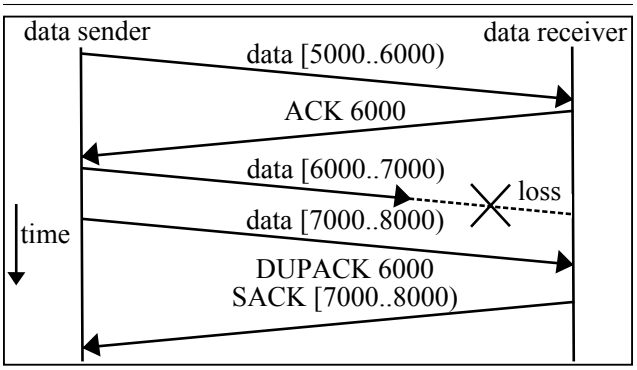

Figure 1. SACK Generation Example

a new data segment containing the sequence numbers [5000..6000). After receiving the ACK for this segment the sender transmits two new data segments. The first segment is lost while the second segment reaches the receiver. Non-contiguous blocks of data that have been received and queued by the data receiver may be selectively acknowledged by a SACK option. The receiver uses SACK blocks (sequence number boundaries of the received, non-contiguous blocks of data) for selective acknowledgements. Upon reception of the out-of-order segment [7000..8000), the cumulative acknowledgement field remains at 6000. The receiver sends a duplicate acknowledgement (DUPACK) and informs the sender about the successfully received segment [7000..8000) by using a selective acknowledgement (SACK) option. On arrival of the SACK information, the data sender should mark any affected segments in its retransmission queue by setting a SACKed flag. The data sender should skip SACKed segments during any later retransmission. A SACK option that specifies n SACK blocks has a length of $8 * n + 2$ bytes. Options in the TCP header are limited to 40 bytes, which results in a maximum of 4 SACK blocks when using no other options. The SACK-based loss recovery algorithm, which is specified in [3], is a conservative replacement for the loss recovery algorithm from [2]. It is using stored SACK information to calculate an estimate of outstanding data segments in the network. This SACK-based estimate is used during loss recovery to limit the sending rate. [3] explicitly allows to combine the SACK-based loss recovery algorithm with the Limited Transmit algorithm, which is defined in [1]. The Limited Transmit algorithm allows a sender to send a new segment, on arrival of the first or second duplicate acknowledgement, if certain requirements are satisfied. Limited Transmit may be used with or without SACK.

2.2. Flow Control

The Flow Control algorithm is used by a data receiver to inform the data sender about the current amount of additional data he is willing to accept. Depending on its receive buffer status, the data receiver is advertising its maximum number of bytes to receive in the Window field of the TCP header (see [11]). The offered Window is a receiver-side limitation for the data sender. If the Window size is very small (below the Maximum Segment Size), this could reduce the efficiency of the transmission. This problem is denoted as the "Silly Window Syndrome"; it is caused by the data receiver when it is advancing the right Window edge whenever any new buffer space is available and by the data sender when it is using any incremental Window, no matter how small, to send more data (see [4]). In order to avoid the Silly Window Syndrome, a receiver should not advertise a Window size below the Maximum Segment Size – except zero. If a Zero Window is advertised, no more data segments should be sent until the receiver opens the Window for new data. When a data sender is receiving a Zero Window and no Retransmission Timer is running, it should start its Persist Timer. On Persist Timer expiry, the data sender should send a 1 byte data segment (termed Window Probe) to ask the data receiver for a Window Update.

3. The Simulation Model

3.1. OMNeT++ and the INET-Framework

OMNeT++ [16] is a discrete event simulation environment with a modular, component-based architecture. The INET framework [17] extends OMNeT++ with protocol implementations of e.g. IP, UDP, TCP, SCTP and several application models. According to the documentation of the INET framework [15], the TCP implementation supports the following specifications:

- RFC 793 [11] (fundamental TCP specification)
- RFC 1122 [4] (updates RFC 793 and gives formal rules for implementation)
- RFC 2001 [12] (congestion control algorithms)

The TCP model can be configured by several module parameters. One of these parameters is the tcpAlgorithmClass. It is used to choose between the available TCP algorithms, e.g. TCP Reno. The current TCP implementation is not state of the art. The TCP header was predefined to a fixed length of 20 bytes and options were not supported. We found

further limitations of the TCP model; the following Flow Control related parts were missing:

- A finite receive buffer size was not modelled.
- The data receiver was always offering the maximum receiver Window size.
- The data receiver was not able to send a Zero Window.
- The Persist Timer was missing.
- A function to send a Window Probe was implemented but has never been invoked.

3.2. The INET TCP Module

The following subsections will give a short overview of the main enhancements we have realized in the TCP module of the INET framework.

3.2.1. Header Extension We have changed parts of the TCP segment format, based on RFC 793 [11] to allow the use of TCP options. When options are used in a segment, the header and byte length are increased appropriately, respecting the maximum allowed size of 40 bytes. The following TCP options have been implemented:

- End of Option List (indicates the end of the option list)
- No Operation (used as padding byte to align option fields on 32-bit boundaries)
- Maximum Segment Size (used during connection setup to communicate the Maximum Segment Size)
- SACK Permitted (used during connection setup to activate SACK)
- SACK (used to convey SACK information)

The TCP model contains a module parameter termed mss (Maximum Segment Size). With the extended header, the value selected for mss can be negotiated by the Maximum Segment Size option, as defined in [11].

3.2.2. SACK Option We have added the new module parameter sackSupport to enabled or disable SACK support. Only if both nodes send a SACK Permitted option during connection setup, SACK will be enabled for the current connection. If an out-of-order data segment arrives when SACK is enabled, the receiver adds a SACK option (implying corresponding SACK blocks) into its answer. A new data structure called TCPSACKRexmitQueue is used to represent the retransmission queue for each TCP connection. When a data segment is sent and SACK is enabled, the appropriate region (a segment represented by sequence number boundaries) is stored in the retransmission queue. All regions own a SACKed flag which is set when receiving appropriate SACK information from the receiver. SACKed regions will be skipped on later retransmissions. The basic SACK implementation – based on the RFCs [9] and [8] – is located in the main TCP algorithm. This means that all existing TCP algorithm classes may be used with SACK. We have integrated the SACK-based loss recovery algorithm – specified in the RFC [3] – into TCP Reno, because TCP Reno is the only TCP algorithm class that contains the loss recovery algorithm from [2]. The Limited Transmit algorithm, specified in [1] and explicitly allowed by [3], has also been implemented. It may be enabled or disabled by using the new module parameter limitedTransmitEnabled.

3.2.3. Flow Control Enhancements The first step to improve the Flow Control mechanism has been to add a finite receive buffer. The maximum size of this buffer is initiated by the already existing module parameter advertisedWindow. Before sending any segment, the current receiver Window is calculated. The function to update the receiver Window is based on [13, page 878–879]. When the Window size is below the Maximum Segment Size, a Zero Window is sent to avoid the Silly Window Syndrome. If the data sender is receiving a Zero Window, he stops data transmission and waits for the reception of a Window Update. To avoid possible deadlock situations, we have added the missing Persist Timer (see subsection 2.2). On Persist Timer expiry, a Window Probe is sent to ask the data receiver for a Window Update.

3.3. TCPDump Module

The TCPDump [14] module allows for capturing the messages which are sent and received by nodes. Captured logs can be shortened and viewed with packet analysing tools like WIRESHARK [18]. TCPDump has been extended to convert the TCP segment format of INET (including the newly implemented TCP header options) from/to binary (network byte order) TCP segments. This allows for comfortable TCP signalling analyses with external tools.

3.4. ExtInferface Module

The ExtInterface [14] module makes it possible to connect the INET simulation environment with real IP-based network nodes. According to [14], root privileges

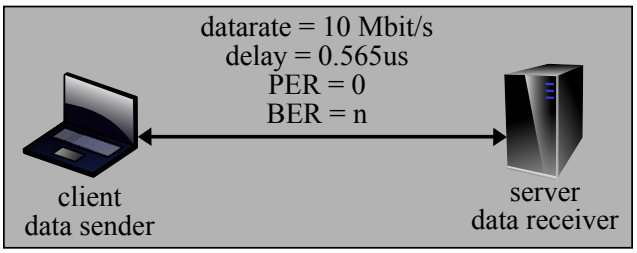

Figure 2. BER Test Network

are required to use the external interface. The external interface supports IP, ICMP, UDP and SCTP messages. We have extended it to also support TCP communications. This opens the way to connect the enhanced TCP module with a real, external TCP implementation.

4. Results

We will present the results of two example setups in the following subsections in order to evaluate our SACK implementation.

4.1. Performance Evaluation

The test network [17] shown in figure 2 consists of a client (data sender) and a server (data receiver). This setup is used to compare the TCP performance (connection throughput) with SACK disabled and enabled. All network interfaces use a Maximum Transfer Unit of 1500 bytes, which is the most common value in IP-based networks. The headers (for TCP, IP and PPP) consume 48 bytes, resulting in 1452 bytes as Maximum Segment Size. The client is configured to open a TCP connection to send 100 MB of data to the server. A ThruputMeter [15] module on the server side is used to measure the average connection throughput. The link between client and server is limited to a data rate of 10 Mbit/s.

We have varied the Bit Error Rate (BER) and performed the simulation with TCP Reno, once with SACK disabled and once with SACK enabled. If the BER is above zero, a pseudo random number generator is used to compute for each packet whether the bit error flag needs to be set. A packet with set bit error flag is dropped by the receiving network interface. For the statistical accuracy of the results, every simulation configuration has been processed ten times with varied seeds. The measured average connection throughputs have been collected to calculate the mean value for any configuration. According to our computations

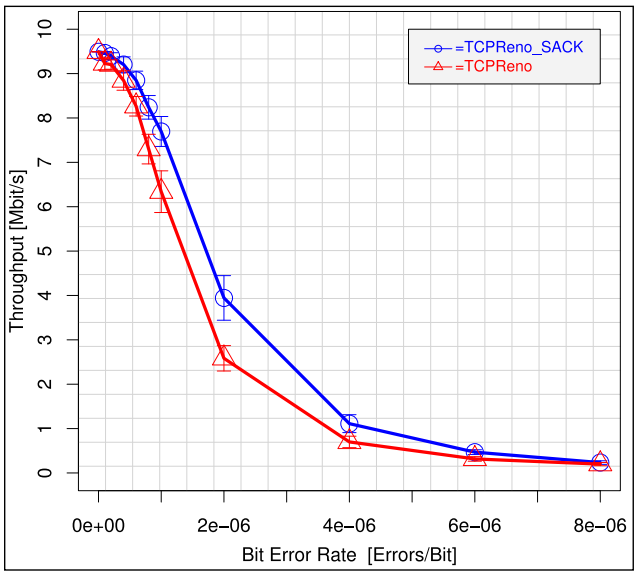

Figure 3. Throughput Results

the mean values are within a confidence interval of at least 95%. SimProcTC [5] has been used to plot the results, which are depicted in figure 3.

When the BER is set to zero, the throughput with and without SACK is at 9.49 Mbit/s. With a BER of $1 * 10^{-7}$, a small difference can be observed. The SACK gain is here at 2.57%. Up to a BER of $6 * 10^{-7}$, the SACK gain is rising continuously – however, is remaining below 10%. At a BER of $2 * 10^{-6}$, the channel utilization is distinctly reduced. The throughput is reduced to 2.58 Mbit/s without SACK and to 3.94 Mbit/s with SACK. The SACK gain has reached 52.68%. The following two result pairs show a SACK gain above 50% – but with and without SACK, the throughput is quite close to zero. The presented simulation results illustrate the performance gain that can be achieved by using the SACK option.

4.2. Interoperability

The test network [17] shown in figure 4 consists of four network nodes. This setup has been used to test the interoperability with real TCP implementations. The router has been equipped with an external interface [14] to connect the simulation environment to an external server.

The client opens a TCP connection with the external server and sends data. At the external server, a TCP server is running to accept the incoming data. The external server is simultaneously connecting to the

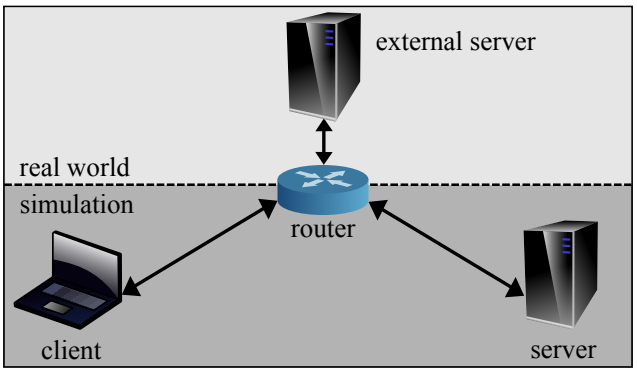

Figure 4. ExtServer Test Network

(internal) server and is sending TCP data from the real network into the simulated network. We have used an IPERF [10] script to realize this functionality. Both TCP flows have been captured at the external server. After analysing the logs we have come to the following results:

- The implemented options have been used during connection setup.
- The data transmission has been successful, because both receivers have acknowledged data arrivals.

The presented test network has shown the interoperability of the enhanced TCP module with a real TCP implementation and has demonstrated our TCP extension of the external interface.

5. Conclusion and Outlook

In this paper, we have described our enhancements of the TCP module, which is part of the INET framework for OMNeT++. Some important features of modern TCP implementations have been missing: the possibility to negotiate options, a Flow Control using a finite receiver buffer, the Persist Timer and Window Probing as well as Selective Acknowledgements. These features have been added by us. In our performance evaluation, we have demonstrated their usefulness and interoperability with real implementations using the ExtInterface. Our enhanced TCP module has already been integrated into the INET framework and is now accessible at [17].

References

[1] M. Allman, H. Balakrishnan, and S. Floyd. Enhancing TCP's Loss Recovery Using Limited Transmit. Standards Track RFC 3042, IETF, Jan. 2001.

[2] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. Technical Report 2581, IETF, Apr. 1999.

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

[4] R. Braden. Requirements for Internet Hosts – Communication Layers. Standards Track RFC 1122, IETF, Oct. 1989.

[5] T. Dreibholz, X. Zhou, and E. P. Rathgeb. SimProcTC – The Design and Realization of a Powerful Tool-Chain for OMNeT++ Simulations. In *Proceedings of the 2nd ACM/ICST OMNeT++ Workshop*, Rome/Italy, Mar. 2009. ISBN 978-963-9799-45-5.

[6] M. Duke, R. Braden, W. Eddy, and E. Blanton. A Roadmap for Transmission Control Protocol (TCP) Specification Documents. Informational RFC 4614, IETF, Sept. 2006.

[7] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno and SACK TCP. *SIGCOMM Comput. Commun. Rev.*, 26(3):5–21, 1996.

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

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

[10] NLANR/DAST and N. Richasse. Iperf: Tool for Measuring Maximum TCP and UDP Bandwidth Performance, 2009. http://iperf.sourceforge.net.

[11] J. Postel. Transmission Control Protocol. Standards Track RFC 793, IETF, Sept. 1981.

[12] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. Standards Track RFC 2001, IETF, Jan. 1997.

[13] W. R. Stevens and G. R. Wright. *TCP/IP Illustrated, Volume 2: The Implementation.* Addison-Wesley Professional, 1995. ISBN 978-0201633542.

[14] M. Tüxen, I. Rüngeler, and E. P. Rathgeb. Interface Connecting the INET Simulation Framework with the Real World. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools)*, pages 1–6, Marseille/France, Mar. 2008. ISBN 978-963-9799-20-2.

[15] A. Varga. INET Framework for OMNeT++/OMNEST snapshot 2009-03-12, 2009. http://inet.omnetpp.org/doc/INET/neddoc/index.html.

[16] A. Varga. OMNeT++ Discrete Event Simulation System, 2009.

[17] A. Varga. INET Framework for the OMNeT++ Discrete Event Simulator, 2010. http://github.com/inet-framework/inet.

[18] Wireshark. Wireshark: The World's Most Popular Network Protocol Analyzer, 2010.