Integration of SCTP in the OMNeT++ Simulation Environment

Irene Rüngeler
Münster University of Applied Sciences
Fachbereich Elektrotechnik und Informatik
Stegerwaldstrasse 39
D-48565 Steinfurt, Germany
i.ruengeler@fh-muenster.de

Michael Tüxen
Münster University of Applied Sciences
Fachbereich Elektrotechnik und Informatik
Stegerwaldstrasse 39
D-48565 Steinfurt, Germany
tuexen@fh-muenster.de

Erwin P. Rathgeb
University of Duisburg-Essen
Institute for Experimental Mathematics
Ellernstrasse 29
D-45326 Essen, Germany
erwin.rathgeb@iem.uni-due.de

ABSTRACT

The INET framework for the widely used OMNeT++ simulation environment supports discrete event simulation for IP-based networks. This paper describes an implementation of the Stream Control Transmission Protocol (SCTP) within this framework. SCTP is a new transport protocol originally designed for signaling transport in telephony networks. The simulation model and parameters of the SCTP implementation in INET are discussed in detail. The INET framework has also been extended to support external interfaces. These interfaces allow to set up hybrid scenarios where simulated nodes communicate with real external IP-based nodes. This new feature was used to test the simulation against multiple SCTP implementations at the last SCTP Interoperability tests. Other methods that were used to verify the SCTP simulation are finally discussed.

Categories and Subject Descriptors

I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis; I.6.6 [**Simulation and Modeling**]: Simulation Output Analysis

General Terms

Protocol simulation, Transport protocol

Keywords

SCTP, Omnet++, INET

1. INTRODUCTION

Our groups have been actively involved in the standardization of the new IETF Transport protocol SCTP (Stream Control Transmission Protocol [7]) from the start and have developed the SCTPLIB [3], an open source SCTP implementation, together with an industry partner. To be able to evaluate this feature-rich and complex protocol and to develop new features and extensions, a fully featured, standards conformant simulation model of SCTP was required.

In INET, models for the standard protocols of the TCP/IP protocol family are already provided. Since standard conformance was one of the major goals for the SCTP model, it was highly desirable to be able to use the real traffic traces, the test suite implementations for testing real SCTP implementations and of course the readily available real SCTP implementations for debugging, testing and validating the SCTP model. As a consequence, we extended the INET framework with an RFC conformant SCTP simulation with suitable user applications, a dump module to provide traffic traces and an external interface for testing and evaluating the simulation with real-time implementations.

This paper will introduce the concept, realization and evaluation of the SCTP simulation and is structured as follows: In Section 2 the enhancements to the INET framework in general are described. The SCTP protocol and its implementation in the INET framework are briefly introduced in Section 3. Section 4 describes the implementation explaning the realised architecture, additional modules and configurable parameters. A brief overview of the testing methods in given in Section 5 and Section 6 offers several examples featuring the variety of possible scenarios. Finally conclusions and a short outlook on future work is given in Section 7.

2. ENHANCEMENTS TO INET

As OMNeT++ is a very versatile tool there are various ready-made simulation models provided for download. One of those is the INET framework [1].

The INET framework consists of a variety of protocol implementations, among them are TCP, UDP, ICMP, IP, PPP, Ethernet and some routing protocols. In addition protocol independent modules like *RoutingTables*, *Routers*, *Switches* and *Hubs* are available. They are all simple modules and can be combined to form compound modules and networks.

One of those compound modules for instance is the *StandardHost* which consists of a complete IP stack with PPP or Ethernet interfaces, a network layer, a Ping application, TCP or UDP as transport layer and corresponding appli-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
OMNET++ 2008, March 03, Marseille, France
Copyright © 2008 ICST 978-963-9799-20-2
DOI 10.4108/ICST.SIMUTOOLS2008.3062

cations. We have complemented this host with the transport protocol SCTP, a suitable application, external interfaces ([12]) and a dump module (see Figure 1). We will specify these modules in the following sections and show miscellaneous examples.

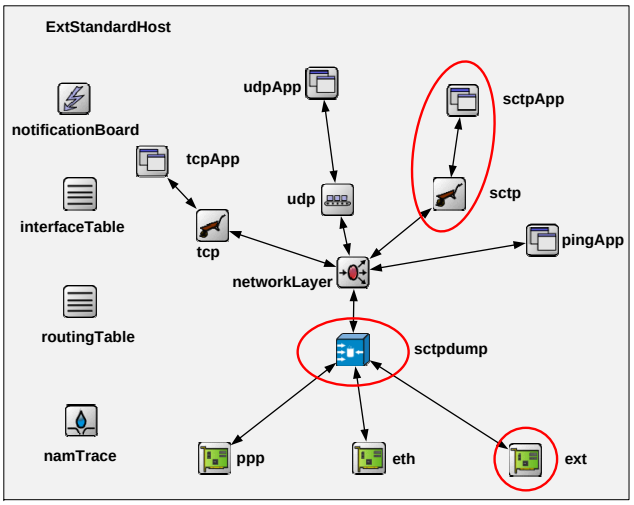

Figure 1: Compound Module ExtStandardHost

Another important feature of INET is the ability to use real network addresses and do the routing according to rules derived from routing tables. Although a *FlatNetworkConfigurator* can be used to automatically distribute addresses among the hosts of a network, we set up routing tables to be able to guide the traffic on distinguished paths through the network.

3. THE STREAM CONTROL TRANSMISSION PROTOCOL (SCTP)

3.1 The Design of SCTP

The Stream Control Transmission Protocol (SCTP), specified in [7], was developed as the generic transport protocol for signaling transport (SIGTRAN) in IP-based telephone signaling networks. Therefore, one of the most important design goals was network fault tolerance. During the last six years a lot of improvements of the base protocol [8] together with several extensions have been standardized. SCTP has gained quite some acceptance in the meantime and is part of the Linux 2.6 kernels, the Solaris 10 operating system and will be incorporated in FreeBSD 7. It is deployed in signaling networks of telephone network operators and used in IP-based signaling for Universal Mobile Telecommunication System (UMTS) networks. It should be noted that SCTP is the first transport protocol being specified by the Internet Engineering Task Force (IETF) and deployed in commercial networks after the introduction of the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP) in the 1980s. A good overview of SCTP is given in [5].

3.2 Features of SCTP

SCTP as specified in [7] is a connection oriented protocol providing reliable transport of user messages. An SCTP connection is called ”association”. Each SCTP endpoint can use multiple IP addresses, which can be configured by different network interfaces, to provide network fault tolerance. Currently this multihoming support is used for redundancy only, but ongoing research analyzes the possibility to use it also for load sharing. All addresses used by the endpoints are negotiated during association setup.

The base protocol provides in-sequence delivery of user messages within several independent streams. The number of streams is negotiated during association setup and can be different in both directions. When an SCTP user application sends a message, it specifies the stream the message belongs to. In case of ordered streams SCTP makes sure that all user messages belonging to the same stream are delivered to the application in the same sequence as they were sent. However, the sequence of the messages among the different streams might change. This reduces the head of line blocking problem in case of message loss.

Multiple user messages can be bundled into an SCTP packet to reduce the number of small messages. User messages which are too large are fragmented by the sending SCTP endpoint and reassembled by the receiving SCTP endpoint.

SCTP and TCP have the flow and congestion control mechanisms in common, which have been adapted from the byte stream oriented protocol TCP to the message oriented protocol SCTP. However, compared to TCP there are several additional protocol parameters, which need to be adjusted appropriately to use SCTP in the telephony signaling environment .

Several SCTP extensions have been developed and have already been standardized or are in the process of standardization:

- The partial reliability extension PR-SCTP has been specified in [6]. It allows the sender to control the level of reliability in different ways. This provides a service to the user which is not available when using UDP or TCP.

- The IP-addresses of the SCTP endpoint can only be negotiated during the setup of an association, which is a severe restriction for long term SCTP associations. Therefore, an extension called ADD-IP has been specified in [11]. It allows SCTP endpoints to change the set of IP addresses being used during the lifetime of an SCTP-association. A security extension required to avoid association hijacking in this case called SCTP-AUTH has been specified in [13].

- Several groups using SCTP have requested to be able to switch back individual SCTP streams to the state they had directly after association setup. An extension called STREAM-RST has been developed to provide this functionality and is specified in [9].

- Links with high bit error rates lead to spurious retransmissions. As a consequence the congestion window will be reduced and less packets sent. In order to announce packet drops resulting from bit errors the extension PKTDROP has been introduced in [10].

The base protocol together with these extensions provides a very powerful general purpose transport layer.

4. IMPLEMENTATION OF SCTP IN INET

4.1 Realized Features

The INET simulation model has been extended by a fully featured SCTP implementation using the infrastructure of this framework. It supports IPv4 and IPv6 as network layers and multihomed hosts. The base protocol as specified in [7] is realized except for fragmentation. In [8] very important modifications concerning congestion and flow control like the calculation of the slowstart threshold, the handling of the congestion window, zero window probing or the sending of gap reports were introduced and also included in the simulation.

One of the main design goals was to be able to freely configure all relevant protocol parameters in order to analyze their influence on the protocol performance.

It is planned to publish this SCTP implementation as Open Source and to include it in a future version of the INET framework.

4.2 The Simulation Architecture

SCTP is a complex protocol combining features from TCP and UDP plus realizing new concepts like streams. Figure 2 shows a schematic overview of the different parts of SCTP. The major blocks that have to be distinguished specify the behavior of the data sender, the reaction of the data receiver and the control messages including the handshakes to setup and take down an association. Furthermore, Congestion Control and Flow Control have a significant impact on the behavior of the transport endpoints.

In the following sections we will give a short description of the main parts of the implementation.

4.2.1 Messages

The means of communication in OMNeT++ are messages. The main class provided is *cMessage* that defines the necessary attributes and methods to send and receive messages.

The programmer can define subclasses by adding message fields. We needed a lot of different classes for the primitives as commands between the application and the transport layer and the messages containing the SCTP packets. The primitives are directly subclassed from *cMessage*, whereas the SCTP packets are more complex and can consist of several subclasses of *cMessage*. The main classes are *SCTPMessage*, *SCTPChunk* and *SCTPParameter*.

SCTPMessage contains the SCTP message header and a number of chunks. Each chunk type is subclassed from *SCTPChunk* and has to be handled individually as it has a format of its own. Some chunk types just consist of the header like the COOKIE-ACK chunk, some contain several parameters which can be mandatory or optional. This is the case for the INIT or INIT-ACK chunk. Each parameter has to be subclassed from *SCTPParameter*, consisting of a header with the type and length information and a body containing the value. When a single message has to be included in another one, encapsulation is used. But when several messages have to be inserted, like the chunks in an *SCTPMessage*, a dynamic array is used.

4.2.2 Association Setup and Takedown

A four-way-handshake starts an SCTP association. It consists of the control messages INIT, INIT-ACK, COOKIE-ECHO and COOKIE-ACK and is initiated by sending the primitive SCTP-ASSOCIATE from the upper layer to the transport layer. The receiving side must have sent an SCTP-OPEN-PASSIVE before, so that a listening socket has been created. The handshake is normally started by a client wishing to set up a connection with a server, but a peer-to-peer communication is also possible with both peers opening listening sockets and starting the setup combining their control data to one association. Both alternatives are realized in the simulation and will be referred to in Section 4.3.1.

As OMNeT++ is a discrete event simulation environment it uses state machines and provides methods to react to occuring events. In Figure 3 the different possible states of the simulation are shown with the corresponding events and the necessary actions to transit from one state to another. The upper half of the figure presents the setup, the lower half the takedown. Data messages are only accepted in the states ESTABLISHED and SHUTDOWN-PENDING.

Timers play a very important role in SCTP. Besides the ones listed in the figure a lot more are needed for instance to trigger the retransmission of data chunks, to send selective acknowledgements (SACKs) or HEARTBEATs. In OMNeT++ timers are realized by sending so-called self-messages. They get a certain arrival time and are inserted in the list of scheduled events. Thus they are handled like other messages and can even carry information.

4.2.3 Data Sender

After the upper layer received the information that an association had been established it can start sending data (see Figure 2). SCTP provides the use of several streams for incoming and outgoing connections the number of which is negotiated in the setup process. Each stream can carry data messages, which can be either unordered or ordered. As the use of the streams is application dependent, the upper layer has to provide the number of streams and also the information which stream each data message belongs to and an indication whether it should be delivered ordered or unordered.

Arriving at the transport layer the data messages are sorted into the appropriate send stream queues. The overall send queue size can be unlimited or limited with configurable size. Whenever the limited queue is emptied to half its size, a notification is sent to the upper layer to order more data.

The sequence in which the stream queues are emptied is controlled by the stream scheduler. The scheduling strategy to be used is not specified by the RFC 4960 ([7]). At present only the Round Robin queueing strategy is realized.

The amount of data that may be sent is influenced by many factors. It has to be calculated from the number of outstanding bytes, the congestion window, the advertized receiver window and of course the amount of data provided in the send streams. This calculation will be discussed in Section 4.2.5. If the user has configured the sender to consider the Nagle algorithm to reduce the number of small packages (see [4] for details), every packet is bundled with data chunks up to the (configurable) Nagle point. Before inserting them in the packet, a Transmission Sequence Number (TSN) has to be set for the chunk to have a unique means of identification. The data messages that are sent to the peer are stored in the backup queue, until they can be finally removed. A second queue, the transmission queue, is provided for the temporary storage of messages that have to be retransmitted. The information on which path retrans-

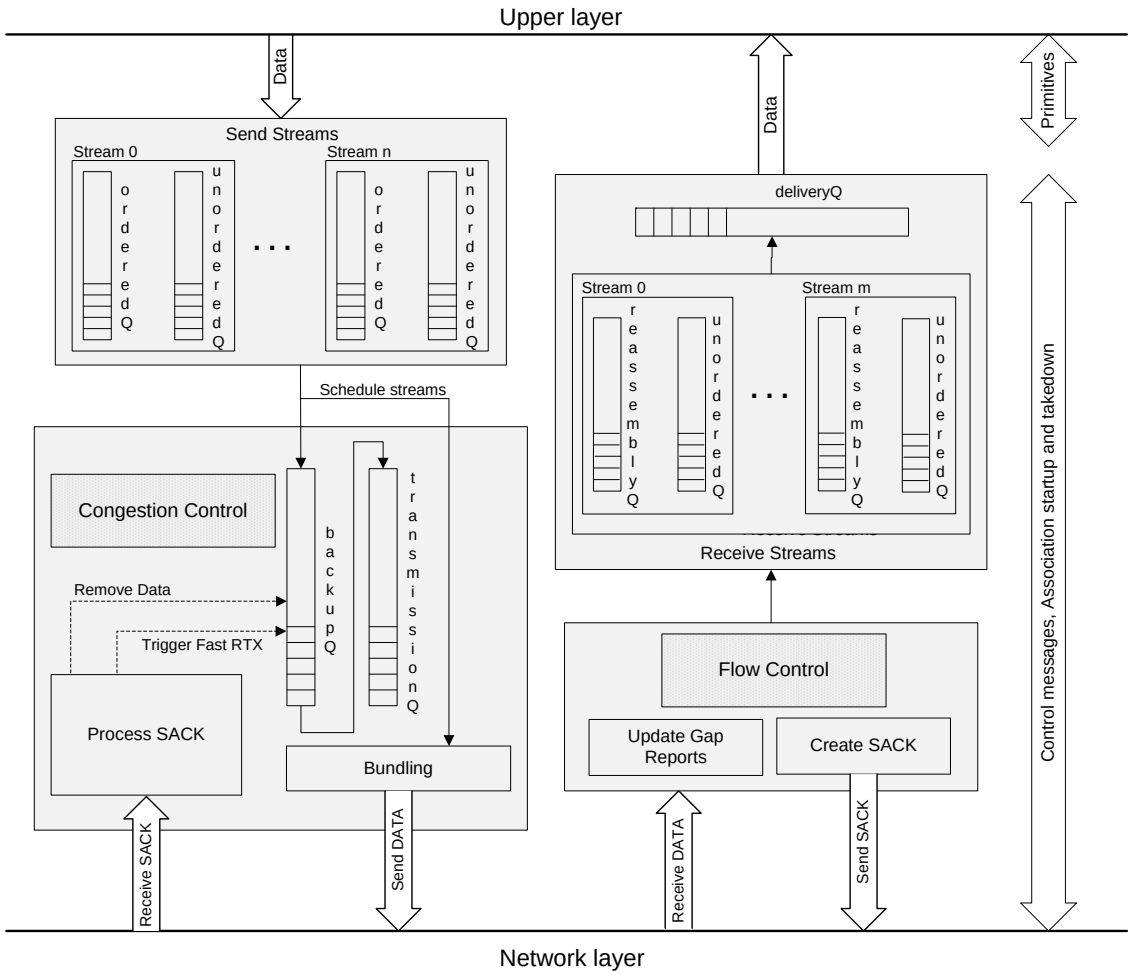

Figure 2: The Simulation Architecture of the SCTP module

missions occured, their number, whether the data has been acknowledged or counts as outstanding, and many more attributes characterize a data message and have to be stored with the data. When putting a packet together the messages scheduled for retransmission have to be considered first and only the remaining space can be filled with new data.

An arriving SACK chunk influences the transmission of data as it announces the Cumulative TSN Ack. All TSNs up to this number can be finally removed from the backup queue. Present gap reports lead to an increase in the gap report count of the missing TSNs, which results in a transfer of the chunk from the backup queue to the transmission queue for fast retransmission, if the user defined gap report limit has been exceeded.

4.2.4 Data Receiver

The data receiver is featured on the right hand side of Figure 2. The reception of the data messages is influenced by the Flow Control that will be discussed in Section 4.2.6. As the TSNs have to be in sequence, the missing ones are announced in the gap reports that are sent back to the sender for information. The acknowledged data messages are stored in the receive streams. Again each stream consists of two queues, one for unordered and one for ordered data. The order is provided by the Stream Sequence Numbers (SSN) that are maintained for each stream. If data with the appropriate SSN is found, it is stored in the delivery queue, and an SCTP-DATA-ARRIVED-NOTIFICATION is sent to the upper layer, that in turn asks to deliver the data.

The SACK summarizes the results of the TSN analysis and sends the actual Cumulative TSN Ack and the information about the gap reports and possible duplicate TSNs back to the data sender. In addition, the size of the updated Advertised Receiver Window (arwnd) is given (see Section 4.2.6).

4.2.5 Congestion Control

The Congestion Control that SCTP uses is in most parts derived from TCP. Yet some important differences are due to special SCTP features.

As SCTP allows a host to be multihomed, the congestion control mechanism has to be applied to each path separately. This means that a path has its own congestion window (cwnd), slow-start threshold (ssthresh), counter of outstanding bytes and retransmission timeout calculation.

As mentioned before, the Congestion Control influences the amount of data to be sent separately for each path in that not more than the difference between cwnd and the

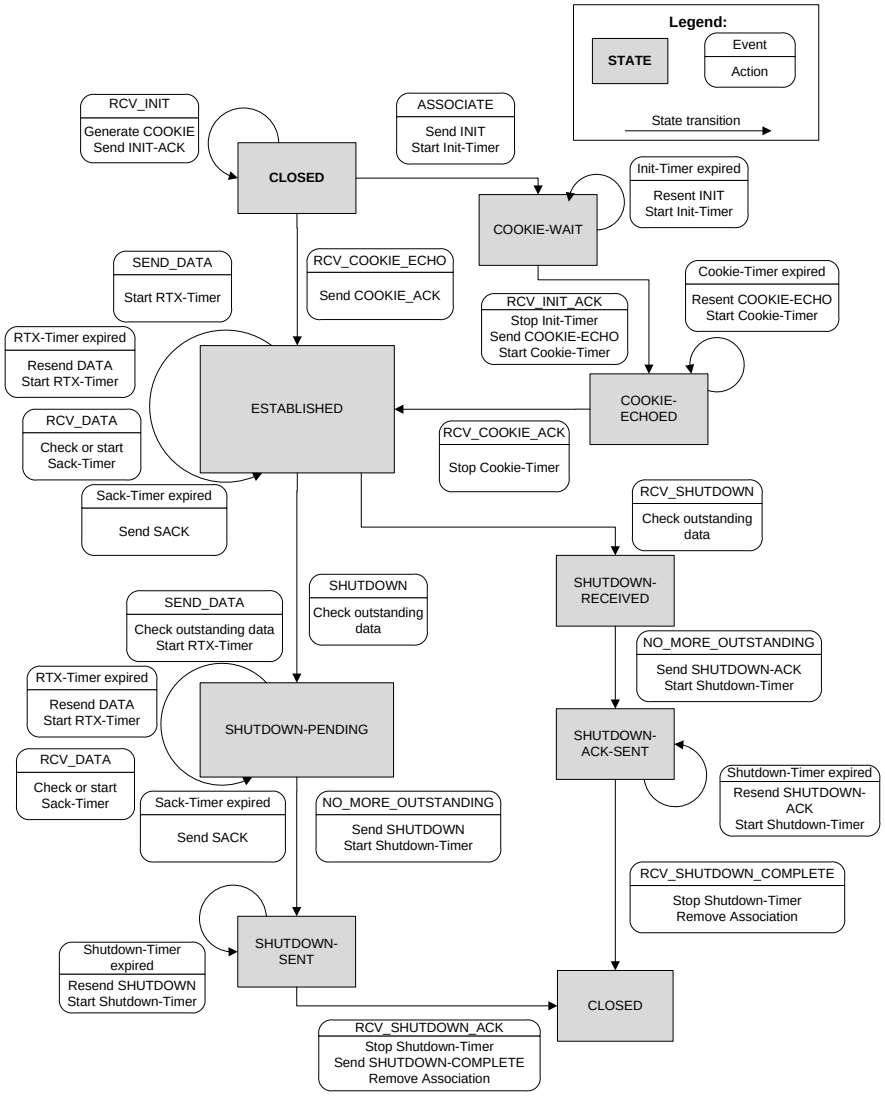

Figure 3: Simulation State Machine

number of outstanding bytes may be transmitted, if permitted by the receiver's arwnd.

While TCP is stream-oriented, SCTP is message-based and thus the overhead of many chunks bundled in one packet can lead to a discrepancy between the transmitted bytes and the sent user data. Therefore we have given the user the possibility to switch between the counting of the outstanding bytes including or excluding the header.

4.2.6 Flow Control

The Flow Control as adopted from TCP shall protect a receiver from a fast sender. Therefore, the receiver announces the amount of empty space in the receive buffer by sending the arwnd attribute in the SACK chunks. The simulation follows this approach and reduces the arwnd with every arriving data chunk and increases it when data is delivered to the upper layer. If the window is reduced to zero, the data sender may only send one chunk to probe the window. If a suitable TSN arrives, i.e. one that fills a gap or advances the Cumulative TSN Ack, it has to replace the highest TSN accepted so far, leaving the former unacknowledged again. Announcing this change in the SACK chunk by adjusting the gap reports leads to a change in the attributes of the affected TSNs in the backup queue. Therefore, even TSNs that have been accepted and acknowledged have to be kept in the queue in case they have to be marked as unacknowledged again.

4.3 Additional Modules

4.3.1 SCTP Applications

The SCTP module (Figure 1) has interfaces to the network layer and the application layer. The interoperability between the transport and its upper layer is realized by sending notifications and primitives (Figure 2) as specified in [7]. In the simulation both a callback and a socket API are realized to provide the upper layer with calls to *bind*, *listen*, *connect*, *send* or *receive*. Thus the application layer takes the initiative to start an association. SCTP answers by either sending notifications, indicating for instance that the

ESTABLISHED state (Figure 3) has been entered, the peer has closed the connection or data are waiting in the receive queue to be picked up.

As examples for upper layer implementations the simulation provides three different applications which work as traffic generators and/or collectors. One is a client with a callback API, one a server with a socket API and the third is a peer that combines both client and server functionality.

The client as a sender can be either configured to send a predefined number of data chunks of a certain length or to start at a certain time and stop at a set time independent from the number of packets. If the packets shall not be sent as fast as possible, a sending interval can be defined. When sending a very large number of messages, the client uses limited sendqueues, as described in Subsection 4.2.3. The client can also work as a receiver being able to discard or echo the messages.

The server can send or receive data. As it is implemented as a combined server, incoming packets can be discarded or echoed. But the server can also generate data, just like the client, representing a peer-to-peer network. The server keeps a record of all the sent or received amounts of data for each association thus providing statistical data for further use.

To support multihoming, a function *sctp_bindx()* is realized. The user can either set the IP adresses that should be bound explicitly or just leave the default value (empty string), if all available addresses should be used. The bound addresses are included in the INIT or INIT-ACK chunk.

When the peer is initialized, it starts by calling *bind* and *listen*, thus being configured as a server. When a certain start time is set, the peer sends a request to SCTP to associate. The peer application is useful, when testing the so-called initialization collisions i.e. when both parties try to set up an association at the same time.

4.3.2 The Dump Module

Although the GUI of OMNeT++ helps to observe the flow of data, we wanted to get a better overview of the packets that were sent to and from the hosts. A dump module was placed between the link layer and the network layer, instead of between the network and the transport layer, to be able to distinguish between the interfaces the message passed through (Figure 1).

```
[2.000] 10.1.1.1.30544 > 10.1.4.1.6666: numberOfChunks=1
INIT
        1: INIT[InitiateTag=267064354; a_rwnd=65535; OS=4; IS=10;
                InitialTSN=1000; Addresses=10.1.1.1,10.2.1.1]

[2.021] 10.1.4.1.6666 > 10.1.1.1.30544: numberOfChunks=1
INIT_ACK
        1: INIT_ACK[InitiateTag=502656; a_rwnd=65535; OS=4; IS=10;
                InitialTSN=2000; CookieLength=0; Addresses=10.1.4.1]

[2.021] 10.1.1.1.30544 > 10.1.4.1.6666: numberOfChunks=1
COOKIE_ECHO
        1: COOKIE_ECHO[CookieLength=0]

[2.042] 10.1.4.1.6666 > 10.1.1.1.30544: numberOfChunks=1
HEARTBEAT
        1: HEARTBEAT[InfoLength=12; time=2.03156]

[2.042] 10.1.4.1.6666 > 10.2.1.1.30544: numberOfChunks=1
HEARTBEAT
        1: HEARTBEAT[InfoLength=12; time=2.03156]
```

Figure 4: Output of the dump module

The incoming packets are examined by decapsulating the messages and analyzing their contents. Afterwards they are transferred unchanged to the next layer. Figure 4 shows part of the four-way handshake and two HEARTBEAT chunks to different destinations on the server side of the communication. The different IP addresses of the client, to which the HEARTBEATs are sent, are underlined.

4.3.3 The External Interface

During the testing and debugging of the SCTP simulation module it seemed to be very attractive to be able to use existing tools like the Wireshark packet analyzer for analyzing the message transfers and to test the interoperability with existing real implementations. Existing implementations of SCTP test suites, like the ETSI conformance test suites, could also be used to test the simulation model.

Therefore, we integrated an interface module in the INET simulation model which allows communication between the simulated nodes and real nodes connected to the host running the simulation via an IP-based network. This capability proved to be very helpful during analysis and testing of the SCTP simulation model.

As the formats of packets used in the simulation differ from the ones sent on the wire there have to be two methods to convert the simulated messages to the real packets and vice versa. We called them *Serializer* and *Parser*.

The *Serializer*, reading the data from the simulation and writing them on the wire, also provides an additional value for the dump modul. As the dump output can only be traced in fast mode, a mechanism is needed to store the simulation traces. Therefore, the packets are converted by the *Serializer* with additional headers to the pcap format and stored in files. Thus they can be analyzed by the Wireshark packet analyzer. A user can turn this feature on by providing a name for the tracefile. For more details refer to [12].

4.4 Configurable parameters

In order to be able to test the network under different conditions there are specific parameters that can be easily configured by editing a textfile. They are divided into those concerning the transport layer and those specifying the applications.

4.4.1 SCTP parameters

The parameters concerning the transport layer can be configured for each host individually. In [7] a number of parameters with their default values are listed, that can be set by the user. Among them are *assocMaxRetrans*, *pathMaxRetrans* and *maxInitRetrans* to set the maximum number of unsuccessful retransmissions before a path is set inactive or the peer unreachable, or the parameters to influence the calculation of the retransmission timeout (*rtoMin, rtoMax, rtoInitial, rtoAlpha* and *rtoBeta*).

Most implementations provide additional SCTP kernel parameters. We made all major attributes configurable. The most important of them are the advertised receiver window (arwnd), the parameters to handle the Nagle algorithm (*nagleEnabled* and *naglePoint*), and those to influence the sending of SACKs (*sackFrequency* and *sackPeriod*).

4.4.2 SCTPApp parameters

In addition to those parameters that are needed for every connection like destination *address*, *primaryPath*, number of *outboundStreams* or *message length*, some parameters are provided that allow to change the sending behavior (*de-*

layFirstRead, thinkTime and *echoDelay*). To allow more predictable and longer testing times the parameters *queueSize, startTime* and *stopTime* can be configured. As a host can start several applications, so that for instance a server can have associations with numerous clients, the application parameters can be configured independently for every application.

5. TESTING THE SIMULATION

As a simulation is self-contained, the question arises how it can be tested, to evaluate its correctness.

The obvious way is to configure scenarios, test them and see whether the results are plausible.

A more systematic approach is the use of testcases. As we implemented the external interface, it was possible to connect to an external computer. An SCTP testtool and the corresponding ETSI conformance tests are provided in [2]. The testtool ran on the external computer and used the simulation as an SUT (System under test). After having passed the tests we were sure that the most important features of the protocol were implemented correctly.

But still the RFCs leave the possibility to interpret some specifications in different ways. Therefore, interoperability events bring developers of various implementations together to test their products against each other. In 2006 we attended the 8th SCTP InterOp in Vancouver, Canada, and in 2007 the 9th SCTP InterOp in Kyoto, Japan. Each time we had the chance to enhance our simulation and make it more robust.

6. EXAMPLES

Of the many possible scenarios we picked three that either show characteristics of SCTP or the advantages of a simulation over a real implementation.

6.1 Sharing the bandwidth between clients

In the internet a link is often shared by numerous connections. Sharing normally implies that the link bandwidth is divided in a fair way. As a consequence, the individual throughput is reduced, when a new connection is established.

We set up a network of five clients (C1 - C5) that could only reach their server over a 5 MBit/sec bottleneck link. We configured their start and stop times so that every 60 seconds an event occured, meaning that either a new client joined or one stopped transmitting. Figure 5 shows the throughput of the first client, that sends data starting at 1 s and stopping at 360 s. The start and stop times of the other clients are marked in the graph. The diagram confirms that the fair bandwidth sharing algorithm works as expected.

6.2 Performance test with external Interface

As mentioned before we can connect our simulation to the outside world. To accomplish this we have to set a route on the external computer to send all packets destined to the simulation to the host where the simulation runs. The packets passing through the network interface are filtered and those destined for the simulation are grabbed by the external interface of the external router.

In Figure 6 the devices with the grey background are part of the simulation. The *extRouter* belongs to both worlds as it has additional external interfaces. Thus it is possible that

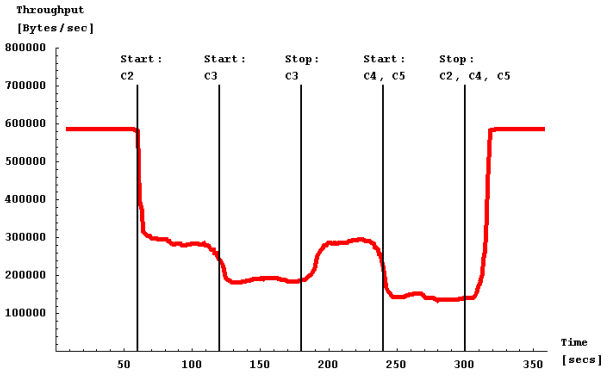

Figure 5: Five clients sharing a limited link

internal and external data pass through this router. The link between *cli1* and *Router1* is limited to 10 MBit/sec. First we sent 200,000 data chunks of increasing sizes between 10 and 1400 Bytes on the red path (solid) from *cli1* to the *external server* and measured the throughput. In Figure 7 the red graph with the diamond-shaped symbol shows the result. As a comparison the maximal theoretical throughput is represented by the orange graph with the square-shaped symbol. The figure shows that the simulation is able to fully utilize a link of 10 Mbit/sec.

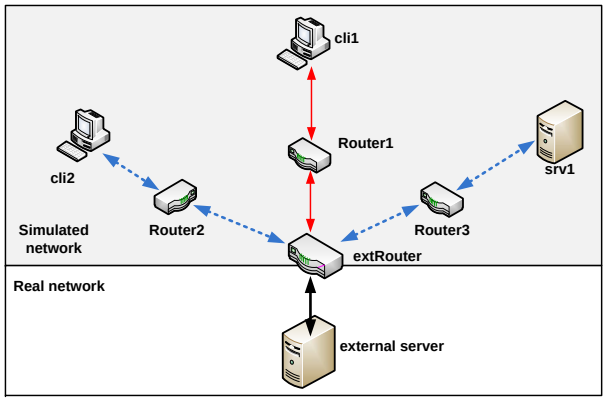

Figure 6: *Cli1* sends data via *extRouter* to a real PC while internal traffic from *cli2* to *srv1* is passing through *extRouter*

A second series of measurements was performed to find out whether additional traffic passing through the router would have an impact on the throughput, meaning that the processing of the events from the external router could not keep up with the packets arriving. Therefore, *cli2* sending data on the blue path (dashed) to *srv1* was to start earlier than *cli1* and had to run longer than the external association. The throughput is shown by the blue graph with the triangle-shaped symbol. It is obvious that in this scenario the internal traffic has no significant influence on the external traffic.

6.3 Small RTO-Min versus large Rto-Min

The main variable in the algorithm for the retransmission timeout (RTO) is the round-trip time (RTT) of messages, that have not been retransmitted, or of HEARTBEATs.

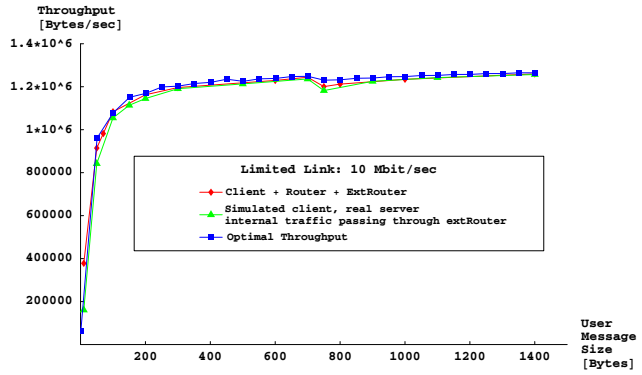

Figure 7: Throughput of an external connection when competing with an internal one

RTO is limited by the two parameters RTO-Min and RTO-Max which are by default 1 sec and 60 secs respectively. As a normal RTT is much smaller than RTO-Min it does not really have an influence on RTO.

In this example we wanted to see whether a smaller RTO-Min which has to lead to a smaller RTO and therefore to earlier retransmissions would have an influence on the throughput compared to a connection using the default values.

We set up a network with two dualhomed clients that were connected via two routers with a dualhomed server. The link between the two routers was limited to 1 MBit/sec. *Client 1* was started first with an RTO-Min of 1 sec. After 20 seconds *Client 2* began sending data. It only had an RTO-Min of 10 ms.

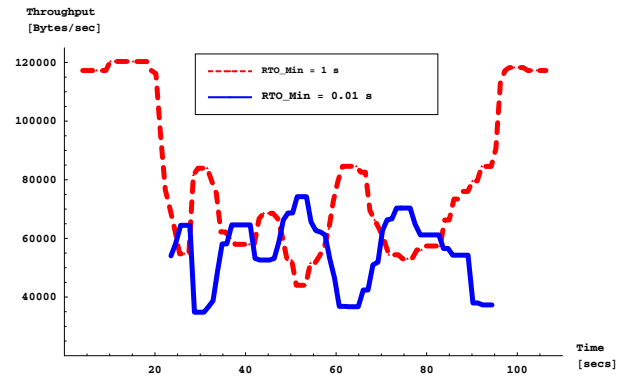

Figure 8: Throughput of two associations with different values for RTO-Min

Figure 8 shows that the throughput of *Client 2* (solid blue line) is in most parts even lower than that of *Client 1* (dashed red line). Looking at the results in more detail revealed the explanation. According to our plan the small RTO-Min resulted in more frequent timer-based retransmissions. But every time the retransmission timer expired the congestion window was resetted to the path-MTU. As the congestion window is one of the limiting factors for the amount of data being allowed to be sent, only a few data chunks could be transmitted and the throughput decreased.

7. CONCLUSION AND OUTLOOK

This paper discussed an SCTP simulation model for the INET framework in detail. This information can be used to extend or simply use the model to analyze the performance of the SCTP protocol or its extensions. The usability of an external interface has been proven and some of the methods used to verify the simulation model have been discussed.

The SCTP model and the external interface will be integrated into a future version of the INET framework.

The simulation model will be extended to be suitable for analyzing SCTP and its interworking with classical telephony signalling networks.

8. ACKNOWLEDGEMENTS

We would like to thank Andras Varga for supporting us during the development of the external interface, Christian Dankbar for developing a prototype of it, and the anonymous reviewers for their comments.

9. REFERENCES

[1] INET Framework Documentation. *Retrieved from: http://www.omnetpp.org/staticpages/index.php?page=20041019113420757.*

[2] SCTP Testtool. *Retrieved from: http://sctp.fh-muenster.de/sctp-testtool.html.*

[3] A. Jungmaier, M. Tüxen, T. Dreibholz, et al. SCTPLIB–an SCTP implementation, 2005.

[4] J. Nagle. Congestion Control in IP/TCP Internetworks. *RFC 896*, January 1984.

[5] L. Ong and J. Yoakum. An Introduction to Stream Control Transmission Protocol. *RFC 3286*, May 2002.

[6] M. Stewart, R. Ramalho, Q. Xie, M. Tüxen, and P. Conrad. Stream control transmission protocol (SCTP) Partial Reliability Extension. *RFC 3758*, May 2004.

[7] R. Stewart. Stream Control Transmission Protocol. *RFC 4960*, September 2007.

[8] R. Stewart, I. Arias-Rodriguez, K. Poon, A. Caro, and M. Tüxen. Stream control transmission protocol (SCTP) specification errata and issues. *RFC 4460*, April 2006.

[9] R. Stewart, P. Lei, and M. Tüxen. Stream Control Transmission Protocol (SCTP) Stream Reset, Internet-Draft draft-stewart-sctpstrrst-04 (work in progress). Technical report, IETF, January 2007.

[10] R. Stewart, P. Lei, and M. Tüxen. Stream Control Transmission Protocol (SCTP) Packet Drop Reporting. Draftstewart-sctp. pktdrprep-06. txt, 2007.

[11] R. Stewart, Q. Xie, M. Tüxen, S. Maruyama, and M. Kozuka. Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration. *RFC 5061*, September 2007.

[12] M. Tüxen, I. Rüngeler, and E. Rathgeb. Interface connecting the INET simulation framework with the real world. *International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SimuTools)*, March 2008.

[13] M. Tüxen, R. Stewart, P. Lei, and E. Rescorla. Authenticated Chunks for the Stream Control Transmission Protocol (SCTP). *RFC 4895*, August 2007.