Implementation and Evaluation of Concurrent Multipath Transfer for SCTP in the INET Framework* †

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

ABSTRACT

The steadily growing importance of resilience-critical Internet applications leads to a rising number of multi-homed sites and systems. But since the protocols of the classical Internet – particularly TCP – assume a single access path only, the number of programs supporting multiple network paths is still small. The Stream Control Transport Protocol (SCTP), which is an advanced general-purpose transport protocol and the possible successor of TCP, brings the support of multi-homing into the applications.

For technical reasons, SCTP uses one network path for data transmission and utilizes the other paths for backup only. An extension to support the load balancing of user data onto multiple paths in order to increase the payload throughput is Concurrent Multipath Transfer for SCTP, denoted as CMT-SCTP. In this paper, we describe our CMT-SCTP extension for the SCTP model provided by the INET framework. By using proof-of-concept simulations, we furthermore demonstrate its usability and configuration parameters.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: SCTP, Concurrent Multipath Transfer; D.4.8 [**Operating Systems**]: Performance—*Simulation*; I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis

General Terms

Algorithms, Design, Experimentation

Keywords

INET Framework, SCTP, Concurrent Multipath Transfer, Model, Evaluation

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

†The authors would like to thank Irene Rüngeler, Michael Tüxen and Brad Penoff for their friendly support.

OMNeT++ 2010 March 15–19, 2010, Torremolinos, Málaga, Spain.

1. INTRODUCTION

When the Internet was designed a long time ago, its endpoints have been equipped with a single network interface only. This worked well for the application of that time – despite of the limited resilience on network failures. Nowadays, however, there is a steadily growing demand for a higher robustness – which leads to more and more devices being offered with several network interfaces. This allows for so-called *multi-homing*, which denotes the simultaneous connection to several – possibly independent – networks. But the dominant protocols – particularly TCP – support a single path access only. However, new protocols like the Stream Control Transport Protocol (SCTP) [1] comply with the requirements of multi-homing.

Instead of using a single network path for user data transport and keeping the others for backup only, it is desirable to utilize *all* paths to improve the payload transmission throughput. This approach is denoted as Concurrent Multipath Transfer (CMT). CMT for SCTP, which we denote as CMT-SCTP, has been proposed by [2] and it is already implemented in the FreeBSD kernel. However, in order to examine open topics on the usage of CMT-SCTP for MPI (Message Passing Interface) [3] or issues on its fairness against TCP have made a simulation model necessary. In this paper, we therefore describe our CMT-SCTP extension for the SCTP simulation model [4] of the INET framework in OMNeT++.

This paper is structured as follows: First, we shortly introduce the SCTP protocol in section 2. Next, we present CMT-SCTP in section 3. In section 4, we introduce our implementation of CMT-SCTP into the INET framework. Using our simulation setup described in section 5, we finally demonstrate the results from our proof-of-concept simulation in section 6. A conclusion is provided by section 7.

2. THE SCTP PROTOCOL

SCTP, which is defined in [1], is a connection-oriented, message-oriented[1], reliable, unicast transport protocol. An SCTP connection is denoted as *association*. As part of a single association, both communication endpoints can bind multiple addresses. This feature is denoted as *multi-homing* and illustrated in figure 1. The binding allows for the transmission of message-oriented data over different network interfaces as a way to enhance end-to-end robustness. The multi-homing feature in SCTP supports both, IPv4 and IPv6 addresses.

2.1 Data Transmission and Acknowledgement

[1]SCTP preserves the message framing – in contrast to TCP, which is stream-oriented and just handles bytes.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8673
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8673

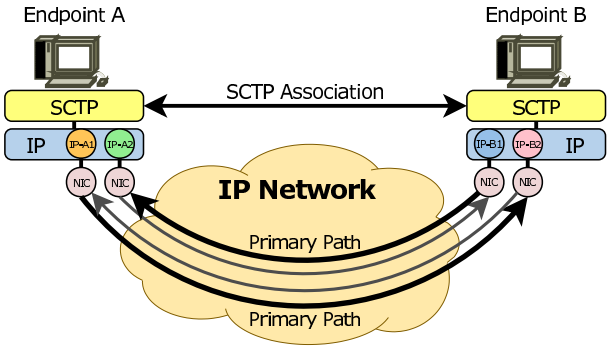

Figure 1: SCTP Multi-Homing and Primary Paths

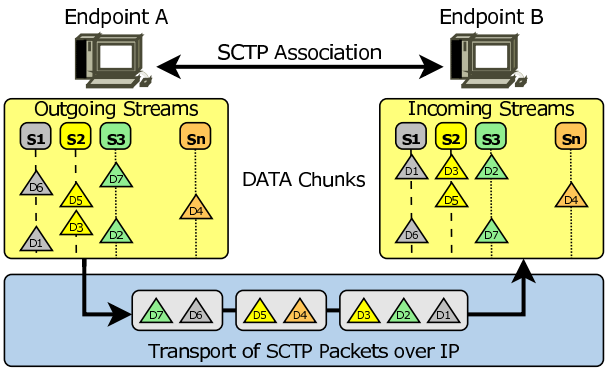

Figure 2: SCTP Multi-Streaming

SCTP messages contain so-called *chunks*. A set of control chunks is defined for the association maintenance; so-called DATA chunks contain segmented user data messages. Every DATA chunk includes a so-called Transmission Sequence Number (TSN). The TSN is used by the receiving endpoint to detect lost, reordered or duplicated DATA chunks [1]. Furthermore, SCTP shares some functionalities with TCP: the congestion control and the *selective acknowledgement* (SACK) scheme. A SACK chunk is used to acknowledge chunks (identified by their TSNs) to the sender. It contains the following information:

Cumulative Acknowledgement (CumAck) is a cumulative acknowledgement for all TSNs up to the given number.

Gap Acknowledgements (GapAcks) lists ranges of the TSNs which have been received *after* the CumAck. That is, there are gaps in the TSN sequence (e.g. due to packet loss or reordering).

Missing chunks – identified by their TSNs – are handled in two different ways:

- Once a DATA chunk is gap-reported as missing for 3 times (default setting [1]), it is retransmitted immediately on the same path (Fast Retransmission [1, subsection 7.2.4]).
- Further retransmissions (possibly on alternative paths) are triggered by a timer (Timer-Based Retransmission).

For the transmission of SACKs, the mechanism of so-called Delayed Acknowledgement – as defined in [5] – is used. By default, a SACK chunk is sent for every second packet (not chunk!) received within 200ms of the arrival of any unacknowledged DATA chunk (see also [1, subsection 6.2]). This mechanism reduces the acknowledgement overhead. But when receiving a DATA chunk with an out-of-order TSN, a SACK chunk is sent immediately.

2.2 Congestion Control

To handle congestion, SCTP uses a separate congestion window for each path. The adaption of a congestion window is basically the same as in TCP, i.e. it leads to a fair partitioning of the bandwidth on congested links. But a possible parallel transmission of SCTP DATA chunks on multiple paths (i.e. load balancing) can excite an unfairness to existing TCP connections if multiple paths share the same congested bottleneck link. For example, if two SCTP paths of one association share the bottleneck with one TCP connection, the SCTP association would get $\frac{2}{3}$ of the bandwidth, whereas the TCP connection has to be content with the remaining $\frac{1}{3}$ of the bandwidth. For this reason, the current SCTP standard – as defined in [1] – uses only one path in each direction to transmit DATA chunks at the same time. This selected path is denoted as *primary path* (see also figure 1). The alternative paths are only used for retransmissions. SCTP may change the primary path e.g. in case of path errors. The usage of the single primary path solves the fairness issue. Using so-called Resource Pooling (RP) [6], the fairness issue can also be handled when using multiple paths for data transmission. [7] introduces CMT/RP-SCTP, a CMT-SCTP variant being extended by RP.

2.3 Additional Features

SCTP provides in-sequence delivery of user data messages within several independent streams – denoted as *multi-streaming* (see figure 2 for an illustration). All user data messages of the different streams are multiplexed over the same SCTP association. The number of streams in each direction is negotiated during association setup. An SCTP user application specifies the corresponding stream for each message. By default, SCTP makes sure that all user messages belonging to the same stream are delivered in the same sequence as they were sent (ordered delivery). Ordered delivery can be turned off on a per-message basis when not necessary. The sequence of the messages among the different streams might change. This reduces the "head-of-line blocking" problem in case of message loss and retransmission. Multiple user messages can be bundled into an SCTP packet to reduce the number of small messages.

In addition to standard SCTP [1], several protocol extensions have been defined:

Partial Reliability (PR-SCTP) defined in [8] provides an optional per-message unreliable transport. A sender can signal its peer to move the CumAck point forward, in order to skip TSNs. As service to the upper layer, an SCTP implementation can provide configurable policies (e.g. time or number of retransmissions) to limit the number of transmission trials for each message. This feature is useful for real-time communications, where too-late retransmissions become useless.

Dynamic Address Reconfiguration is defined in [9] and allows for the dynamical reconfiguration of IP adresses during the lifetime of an association. This feature can be used to support mobility [10].

Packet Drop defined in [11] allows for nodes announcing packet losses not determined by congestion. Thus, the

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8673
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8673

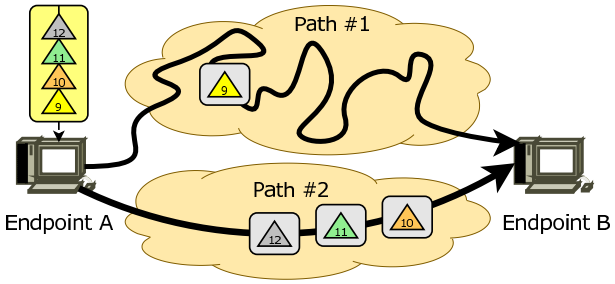

Figure 3: Unnecessary Fast Retransmissions

congestion window need not to be reduced on lossy links (e.g. when using satellite communications).

In the long-term future, SCTP may replace TCP [12]. Especially, the advanced features of SCTP are beneficial for applications like data transport in grid scenarios [3] and Reliable Server Pooling (RSerPool) [13, 14].

3. CMT-SCTP

Modifying standard SCTP to just send DATA chunks over multiple paths instead of a selected primary one would already realize CMT – but lead to a very poor throughput performance. In order to perform CMT efficiently, several additional optimizations are necessary [2]. These optimizations are described in the following subsections.

3.1 Split Fast Retransmission

When sending over multiple paths, DATA chunks may overtake each other due to different path latencies. An example is illustrated in figure 3: sending TSN #9 over path #1 and TSN #10 to TSN #12 over path #2, the sink may first see the TSN #10 to TSN #12 while TSN #9 is still on its travel through the network. Consequently, the sink will notice 3 "out of order" TSNs and acknowledge each one with a CumAck for TSN #8 (i.e. the last "in order" TSN) and corresponding Gap Reports for the already-received TSNs #10 to #12. Then, the sender sees three CumAcks for the same TSN and therefore triggers a Fast Retransmission – which requires a retransmission of TSN #9 and leads to a reduction of the congestion window. When finally the original TSN #9 arrives at the sink, the following retransmitted chunk can be ignored (i.e. it has wasted network bandwidth) and the congestion window has to grow again (i.e. time will be wasted).

The solution to this problem – denoted as Split Fast Retransmission (SFR) [2] – is reasonably simple: the SACK handling has to take care of the individual paths. For each transmitted chunk, the path on which it has been sent is remembered. On reception of a SACK, the path P of a chunk with TSN t_{Missing} reported as missing is checked. TSN t_{Missing} is only assumed as missing when t_{Missing} is smaller than the TSN of the highest successfully acknowledged chunk $t^{P}_{\text{HighestAckedOnPath}}$ on path P. Since both, the chunks with TSNs t_{Missing} and $t^{P}_{\text{HighestAckedOnPath}}$ have been transmitted on path P, and the later TSN $t^{P}_{\text{HighestAckedOnPath}}$ has been received successfully, TSN t_{Missing} is probably lost. Otherwise, there is nothing to do.

In the example above this means that TSN #9 would be Fast Retransmitted if a new TSN #13 on path #1 would have been acknowledged while TSN #9 still remains missing.

3.2 Congestion Window Updates

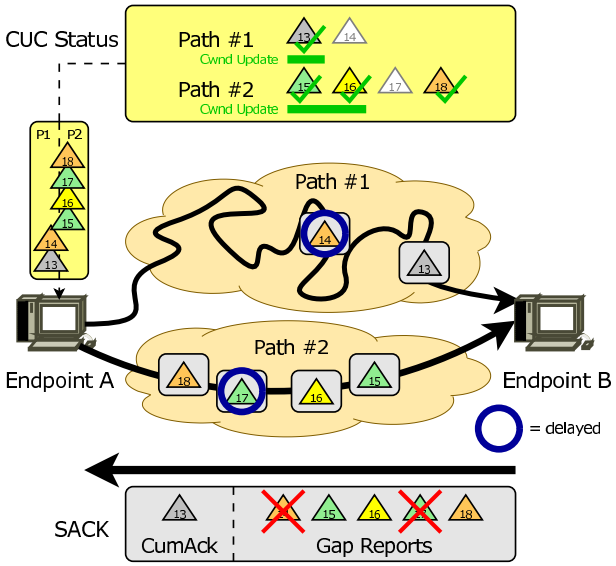

Figure 4: Congestion Window Update Challenge

When a new CumAck has been received, the congestion window of SCTP [1] – as well as of TCP [15] – may grow. On the reception of SACKs with the same CumAck'ed TSN, the congestion window is left unchanged. An example is illustrated in figure 4, where endpoint A sends the chunks with TSN #13 and TSN #14 over path #1 to endpoint B. The chunks with TSNs #15 to #18 use path #2. Due to congestion and reordering, the chunks with TSNs #14 and #17 do not arrive before a SACK chunk is sent back to endpoint A. This SACK contains a CumAck for TSN #13 (i.e. all chunks up to TSN #13 have been received successfully) and Gap Acknowledgements for the TSNs #15, #16 and #18 (i.e. the TSNs #14 and #17 are still missing).

If endpoint A used the congestion window update strategy of standard SCTP, it would see an increase of the CumAck to #13. That is, the SACK has acknowledged one TSN on path #1 and the sender may grow the congestion window of path #1. However, the congestion window on path #2 would not increase, since TSN #14 – which has been sent on the *other* path – is still unacknowledged. Also, the congestion window handling of later SACKs will only take care of *newly* acknowledged TSNs. That is, when the missing TSNs are eventually acknowledged later, the window must not grow. This restriction is necessary to avoid sudden increases of the congestion window leading to bursts of data.

In order to improve the efficiency of CMT-SCTP, the Congestion Window Update for CMT (CUC) strategy [2] takes the existence of different paths into account. On reception of a SACK, it looks for the earliest outstanding TSN on each path. For the example in figure 4, this is TSN #14 on path #1 and TSN #17 on path #2. Therefore:

1. On path #1, the SACK has newly acknowledged the TSN #13 – i.e. one new TSN has been acknowledged and the congestion window on path #1 may grow accordingly.

2. On path #2, there are no outstanding TSNs smaller than TSN #15; the TSNs #15 and #16 have been acknowledged newly. Therefore, the congestion window on path #2 may also grow accordingly.

Since the CUC strategy maintains a virtual CumAck for each path, it is also denoted as Pseudo CumAck. A further

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8673
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8673

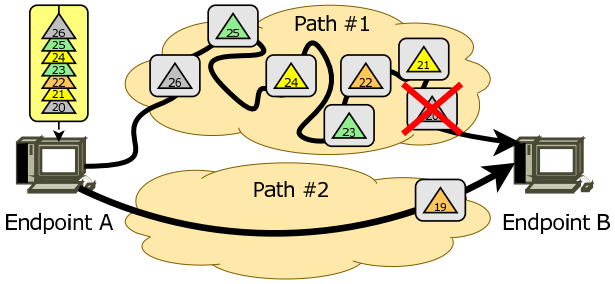

Figure 5: Delayed Acknowledgement Challenge

improvement denoted as CUC version 2 (CUCv2) [2] also takes care of TSNs being retransmitted. When there is a retransmission of a TSN on another path than the original, CUC cannot reliably keep track of the Pseudo CumAck of either path. Therefore, CUCv2 makes a distinction between TSNs which have been transmitted only once and TSNs having been retransmitted. A second Pseudo CumAck – denoted as RTX Pseudo CumAck – is maintained; whenever one of these two Pseudo CumAcks is increased, the congestion window may grow.

3.3 Delayed Acknowledgement

When DATA chunks arrive in sequence, SCTP does not immediately send a SACK chunk for acknowledgement. Instead, it realizes the Delayed Ack algorithm defined by [5] (similar to TCP) in order to reduce overhead traffic. But for reordered chunks, a SACK is transmitted immediately. By default, the reception of 3 SACKs for the same TSN triggers a Fast Retransmission to retransmit a lost chunk (a loss on reordering is likely in the non-CMT case). For CMT, however, reordering is frequent and usually does *not* imply a loss. But each received out-of-order DATA chunk would require the immediate transmission of a SACK chunk. Unnecessary Fast Retransmissions are avoided by SFR already (see subsection 3.1), but the increased SACK traffic overhead remains.

In order to cope with the CMT inefficiency, the Delayed Acknowledgement for CMT (DAC) strategy [2] simply delays all SACK transmissions. However, this would also delay the recovery of a real packet loss which is triggered by the SACKs. An example is provided in figure 5:

1. TSN #20 is lost. No SACK chunk will be transmitted.
2. The TSNs #21 and #22 are received and acknowledged by a SACK chunk. Endpoint A recognises the missing TSN #20 for the first time.
3. The TSNs #23 and #24 are received and another SACK chunk is sent. Endpoint A recognises the gap for the second time.
4. The TSNs #24 and #25 are received and SACK'ed. Endpoint A sees the gap for the third time. This is the default threshold [1] for a Fast Retransmit of the lost chunk.

That is, the loss is detected after six more chunks have been sent. Using standard SCTP behaviour, it would have been detected after only three chunks (the third SACK would have been sent after receiving the out-of-order TSN #23).

DAC solves the problem by two steps [2]: First, the receiver has to put the number of TSNs received since sending the previous SACK chunk into each SACK chunk. Then, the SACK handling procedure of SFR (see subsection 3.1) for a missing TSN t_{Missing} can be modified. An incoming SACK is handled as follows:

- If all newly acknowledged TSNs have been transmitted over the same path:
  - If there are newly acknowledged TSNs t_L and t_H so that $t_L < t_{\text{Missing}} < t_H$, the missing count is incremented by one.
  - Else if *all* newly acknowledged TSNs t_N satisfy the condition $t_{\text{Missing}} < t_N$, the missing count is incremented by the number of TSNs reported in the SACK chunk.
- Otherwise (i.e. there are newly acknowledged TSNs on different paths), the missing count is incremented by one (like for standard SCTP [1]).

For the example shown in figure 5, applying DAC means that TSN #21 is GapAck'ed with a CumAck for TSN #19. Since this SACK chunk contains newly acknowledged TSNs for both paths, the missing counter for TSN #20 will be increased to 1. After receiving the second SACK chunk – containing the newly GapAck'ed TSNs #22 and #23 and a count of 2 TSNs since the last SACK – the missing counter for TSN #20 grows by 2 to 3. By default, this is the threshold for a Fast Retransmission. That is, this retransmission is triggered after 3 TSNs – which is as quickly as in a non-CMT scenario.

4. OUR CMT-SCTP SIMULATION MODEL

We have realized the CMT-SCTP functionalities described in section 3 in an experimental version of the SCTP module [4] in INET. In the following subsections, we will provide an overview of the implementation.

4.1 Module Parameters

The existing SCTP module has been extended by CMT-SCTP, i.e. all SCTP-based INET simulation models could make use of CMT. In order to configure the CMT-SCTP functionality, the parameters summarized in table 1 have been added. By default, CMT is turned off (i.e. allowCMT set to "false"). Unless explicitly changing allowCMT to "true", the new SCTP module behaves exactly like the old, non-CMT model.

The parameters cmtUseSFR and cmtUseDAC enable or disable the SFR (see subsection 3.1) and the DAC (see subsection 3.3) optimizations. The useful default is to turn on both, of course. cmtCUCVariant selects the congestion window update algorithm (see subsection 3.3): "normal" as for standard SCTP [1], "pseudoCumAck" (CUC) or "pseudoCumAckV2" (CUCv2 – the useful default).

The last parameter – cmtSendAllVariant – controls the path selection for outgoing DATA chunks. If there is no path explicitly specified for a chunk, the setting "smallestLastTransmission" chooses the least recently used path (this is the default). This results in a round-robin selection. A setting of "largestSpace" uses the path whose congestion window can currently accept the largest number of bytes, "largestSSThreshold" selects the path having the highest Slow Start Threshold and "largestSpaceAndSSThreshold" is a combination of both settings.

4.2 TCPDump and ExtInterface Interaction

The TCPDump module [16] of the INET framework provides to write the packet trace of an IP node into a dump file. Such dump files can be processed by packet analyzer tools like WIRESHARK [17]. The ExtInterface module [16]

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8673
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8673

Parameter	Functionality	Default Setting
allowCMT	Enable/Disable CMT	false
cmtUseSFR	Enable/Disable SFR	true
cmtUseDAC	Enable/Disable DAC	true
cmtCUCVariant	Congestion Window Update Strategy	pseudoCumAckV2
cmtSendAllVariant	Path Selection Policy	smallestLastTransmission

Table 1: CMT Parameters for the SCTP Module

allows for connecting simulated IP nodes to real network interfaces. That is, simulated applications on IP nodes can communicate with real applications on real IP endpoints in the Internet. The SCTP module [4] as well as our CMT-SCTP improvement are also compatible with the TCPDump and ExtInterface modules. In particular, we have intensively used the SCTP packet trace analysis functionalities of WIRESHARK during development and debugging.

4.3 Multi-Homed Auto-Routing Module

Setting up a test network with manual IP address and routing configuration is a rather time-consuming and error-prone task. Therefore, the INET framework already provides two auto-routing modules (see [18]): FlatNetworkConfigurator and NetworkConfigurator. These modules configure IP addresses to the network interfaces and apply Dijkstra's Shortest Path algorithm to set routing tables. However, the existing modules are not aware of multi-homing. Depending on the configuration, a path #1 which is intentionally longer will not be used in favor of a shorter path #2 (see figure 3 for an example).

To overcome the described challenge, we have added a new auto-routing module called MultihomedFlatNetworkConfigurator. For each link, the new channel parameter netID can be specified. The netID provides the identification for an independent network. A network interface connected to a link belongs to the corresponding network. For the routing table computation, the Dijkstra algorithm is applied on the network corresponding to each ID separately. That is, the paths within one network will not interfere with paths in another network.

But sometimes, links shared between different networks are intended. This behaviour is supported by the special setting of netID=0. When the routing is computed for a certain network ID n, it will contain all interfaces and links having set netID=n or netID=0.

An example network is presented in figure 6, its corresponding NED file is shown in listing 1. This scenario is used in an evaluation of CMT-SCTP for MPI (Message Passing Interface) applications (see also [3]). Line 5 imports the new MultihomedFlatNetworkConfigurator module; it is instantiated in lines 34 to 37. Similar to the auto-routing modules already available in INET, its object does not require any connections. The test network consists of a configurable number of hosts numHosts (see lines 26 to 29), connected to a configurable number of independent networks numNetworks. Each network has its own router (see lines 30 to 32), which constitutes the centre of a star topology. The channel for the links (see lines 18 to 24) defines the attribute netID, which is used in line 40 to define the independent networks.

4.4 NetPerfMeter Test Application

In order to evaluate CMT-SCTP, we had first used the SCTPClient and SCTPServer modules provided by INET. However, these modules lack of statistics features and were therefore not really useful for performance analyses. Also, the protocol support of these programs is restricted to SCTP.

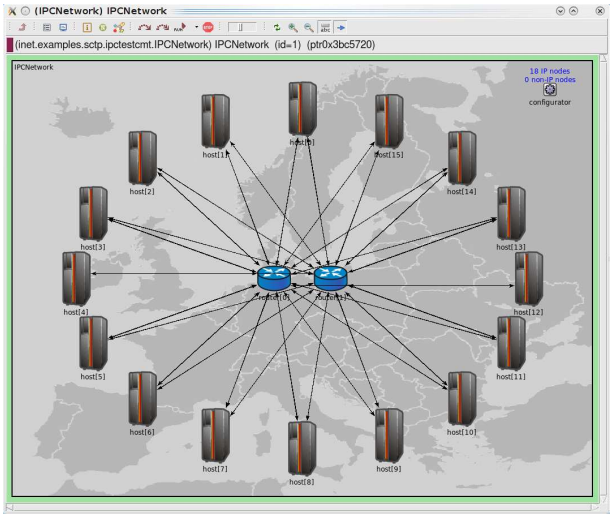

Figure 6: A Dual-Homed Test Network

To compare results to UDP or TCP, other applications (like TCPBasicClientApp and TCPSinkApp, which are part of the INET TCP application modules) – with different and incompatible parametrization – would have been necessary. We have therefore developed our own performance metering application module NetPerfMeter.

Similar to the real Linux/FreeBSD-based performance metering program NETPERFMETER [19], the NetPerfMeter application module provides the unidirectional and bidirectional transmission of saturated and non-saturated flows as well as statistics recording. Its parameters are summarized in table 2. At a given time (connectTime), a NetPerfMeter client (activeMode=true) establishs a connection using the given protocol (protocol, i.e. "SCTP", "TCP" or "UDP") to a NetPerfMeter server (activeMode=false) at a given IP address (remoteAddress) and port (remotePort). After a given time (startTime), the transmission starts; its statistics may be reset after a certain time span (resetTime) to cut off startup behaviour. NetPerfMeter writes its flow statistics in form of scalars at the end of the measurement (stopTime) before connection teardown.

The outgoing data is transmitted as frames in a given interval (frameRate) with a given size (frameSize). Both parameters are volatile. Frames are segmented into messages with a given maximum size (maxMsgSize). The special setting of frameSize=0 turns the flow off; frameRate=0Hz configures a saturated sender. A saturated sender means that it tries to send as much data as possible. The message queue is therefore filled with up to queueSize messages. Currently, a saturated sender is only possible for SCTP, since the TCP and UDP modules of INET have no support for finite transmission queues yet. A work-around for TCP is to ensure a sufficient fill level by using a high frame size. For SCTP,

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8673
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8673

Listing 1 The NED File for the Dual-Homed Test Network

```
package inet.examples.sctp.ipctestcmt;

import inet.nodes.inet.DumpRouter;
import inet.nodes.inet.StandardHost;
import inet.networklayer.autorouting.MultihomedFlatNetworkConfigurator;
import ned.DatarateChannel;

network IPCNetwork
{
    parameters:
        @display("bgi=maps/europe,s;bgb=1024,768");

        int numHosts;
        int numNetworks;
        ...

    types:
        channel path extends DatarateChannel
        {
            // For new MultihomedFlatNetworkConfigurator: path either belongs to
            // all networks (netID=0) or a specific network (netID=n, n>0).
            int netID = default(0);
            datarate  = 1Gbps;
        }
    submodules:
        host[numHosts]: StandardHost {
            parameters:
                @display("p=112,84,ring,400,300;i=device/mainframe_vl");
        }
        router[numNetworks]: DumpRouter {
            parameters:
                @display("p=462,384,r,100;i=abstract/router_l");
        }
        configurator: MultihomedFlatNetworkConfigurator {
            parameters:
                @display("p=950,50");
        }
    connections:
        for j=0..numNetworks-1, for i= 0..numHosts-1 {
            host[i].pppg++ <--> path { netID=j+1; } <--> router[j].pppg++;
        }
}
```

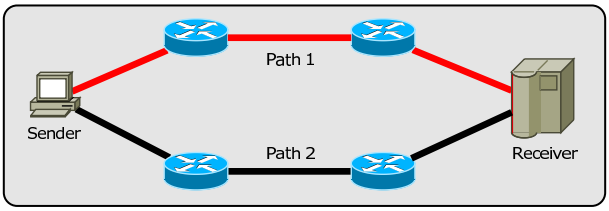

Figure 7: The Proof-of-Concept System Setup

a given fraction of the messages may be sent in unordered (unordered) and PR-SCTP unreliable modes (unreliable). For each message, its ordering and reliability mode is selected randomly according to the configured fractions.

5. OUR SIMULATION SETUP

In order to evaluate our CMT-SCTP module, we have set up a simple test network as depicted in figure 7. A NetPerfMeter sender has transmitted data over a dual-homed network to a NetPerfMeter receiver.

The following configuration parameters have been used, unless otherwise specified:

- The sender has been saturated (i.e. it tried to transmit as much data as possible), the message size has been 1,452 bytes at an MTU of 1,500 bytes (i.e. the DATA chunk packets have fully utilized the MTU [1]). All messages have used the "unordered" transport mode. The advertised receiver window has been large enough to accept all data generated by the sender.
- After association establishment and transmission start, the actual throughput measurement has been started after 19s. The duration of the throughput measurement has been 30s.
- The bandwidth of the PPP links between the routers of each path has been configurable, their delay has been 1ms (realistic for an Ethernet setup). All other links and the routing have been delay-free. The bottleneck network interfaces have used RED queues [20] ($w_q = 0.002$, $\min_{\text{th}} = 20$, $\max_{\text{th}} = 80$, $\max_p = 0.02$).

The SimProcTC [21] tool-chain has been utilized for the parametrization and results processing. The results plots in this paper show the average values of 32 runs and their 95% confidence intervals.

For comparison, we have also set up a similar scenario in reality, consisting of PCs running FreeBSD 8.0 and being interconnected by Gigabit Ethernet links. The FreeBSD 8.0

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8673
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8673

Parameter	Functionality	Default Setting
connectTime	Absolute Time of Connection Establishment	0s
startTime	Relative Time of Transmission Start	1s
resetTime	Relative Time of Measurement Start	5s
stopTime	Relative Time of Measurement Stop	30s
activeMode	Client Mode (true) or Server Mode (false)	true
protocol	Transport Protocol (“SCTP”, “TCP” or “UDP”)	“SCTP”
localAddress	Local IP Address (“” for “Any” Address)	“”
localPort	Local Port	9000
remoteAddress	Remote IP Address (Client Mode only)	“”
remotePort	Remote Port (Client Mode only)	9000
primaryPath	Primary Path (SCTP only)	“”
frameRate	Frame Rate (0Hz = saturated sender)	10Hz
frameSize	Frame Size (0B = flow turned off)	1452B
maxMsgSize	Maximum Message Size (SCTP and UDP only)	1452B
queueSize	Queue Size (currently SCTP only)	1000
unordered	Fraction of Unordered Messages (SCTP only)	0.0
unreliable	Fraction of Unreliable Messages (PR-SCTP only)	0.0

Table 2: Parameters for the NetPerfMeter Module

kernel provides an experimental CMT-SCTP implementation, which has been be activated by `sysctl` settings. The performance metering application NetPerfMeter [19] has been applied to measure the SCTP payload throughput. Bandwidth limitation has been introduced in the routers by applying Dummynet [22], which is the network test infrastructure of FreeBSD.

6. PROOF OF CONCEPT EVALUATION

To show the effects of CMT transport, we have simultaneously varied the path bandwidth ρ of each path in our simulation setup as well as in the real FreeBSD scenario. The payload throughput results for the simulation are shown on the left-hand side of figure 8. When CMT is turned off (i.e. CMT=false; denoted by a red line on a colour plot), the payload throughput linearly scales up to about 95 Mbit/s for a link bandwidth of ρ=100 Mbit/s. This is the expected behaviour. Turning on CMT (i.e. CMT=true; denoted by blue lines on a colour plot), a bandwidth of around 190 Mbit/s is achieved when SFR is turned on (i.e. SFR=true; denoted by solid lines) and CUCv2 (i.e. χ=pseudoCumAckV2) is used. As expected, turning SFR off (i.e. SFR=false; denoted by dotted lines) results in a vast decrease of the throughput – a significant amount of bandwidth is wasted by unnecessary Fast Retransmissions (see subsection 3.1). Also, it can be seen that without an appropriate congestion window growth behaviour for CMT (i.e. χ=normal), reordering – and in consequence a non-growing congestion window (see subsection 3.2) – leads to a reduced throughput.

For comparison, the results of a FreeBSD 8.0 measurement are shown on the right-hand side of figure 8. Since the FreeBSD kernel is intended for application usage and not for research purposes, it does not contain options to turn specific CMT features on or off. The plot therefore shows the results for activated SFR and CUCv2 only. For a link bandwidth of ρ=100 Mbit/s, CMT-SCTP (i.e. CMT=true; denoted by a blue line on a colour plot), a payload throughput of 188 Mbit/s is achieved vs. about 94 Mbit/s for CMT turned off (i.e. CMT=false; denoted by a red line on a colour plot). That is, the results obtained from the measurement look similar to the simulation results. Note again that there is no exact 1:1 mapping possible – due to the different system setups. Nevertheless, is can be observed that our simulation results are as expected and correspond to a real system setup quite well.

7. CONCLUSIONS

With multi-homed sites becoming more and more common, there is a growing interest in utilizing all available network interfaces of a system to improve its data transfer throughput. The CMT-SCTP extension of SCTP provides this feature. In order to investigate the performance of CMT-SCTP in detail, we have realized it as part of the INET SCTP module. Also, we have developed an auto-routing module for the easy configuration of multi-homed networks as well as the performance metering application module NetPerfMeter. In a proof-of-concept simulation, we have shown that our CMT-SCTP model works as expected. Furthermore, its performance corresponds to a real FreeBSD-based system setup.

As part of our ongoing work on CMT-SCTP, we are currently examining its performance in more detail. Particularly, we are analysing and improving the fairness of its congestion control. Another important topic is its performance in heterogeneous network scenarios, where the Quality of Service properties of the paths differ. The goal of our ongoing work is to provide comprehensive configuration guidelines for application developers and users of CMT-SCTP. Finally, we also intend to bring the results of our research into application by contributing to the ongoing IETF standardization process for SCTP.

8. REFERENCES

[1] R. Stewart. Stream Control Transmission Protocol. Standards Track RFC 4960, IETF, September 2007.

[2] Janardhan R. Iyengar, Paul D. Amer, and Randall Stewart. Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths. *IEEE/ACM Transactions on Networking*, 14(5):951–964, 2006.

[3] Brad Penoff, Mike Tsai, Janardhan Iyengar, and Alan Wagner. Using CMT in SCTP-based MPI to Exploit Multiple Interfaces in Cluster Nodes. In *Proceedings of the EuroPVM/MPI*, Paris/France, September 2007.

[4] I. Rüngeler, M. Tüxen, and E. P. Rathgeb. Integration of SCTP in the OMNeT++ Simulation Environment. In *Proceedings of the 1st OMNeT++ Workshop*, Marseille/France, March 2008. ISBN 978-963-9799-20-2.

[5] M. Allman, V. Paxson, and W. Stevens. TCP

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8673
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8673

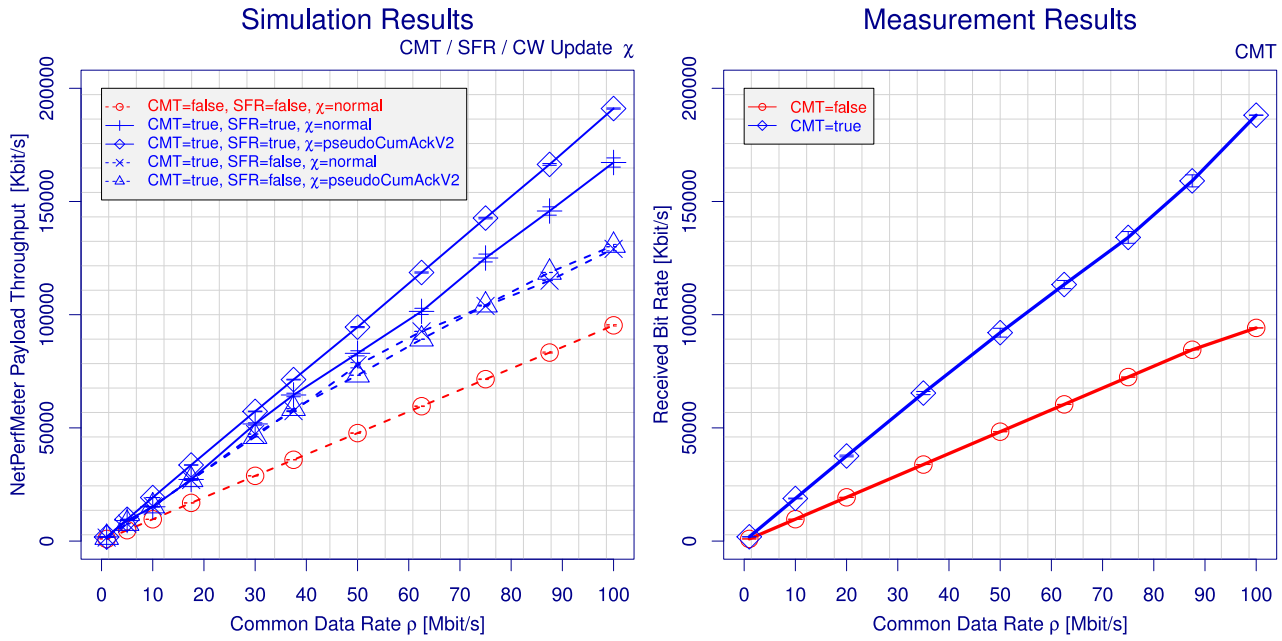

Figure 8: SCTP Application Payload Throughput Results

Congestion Control. Technical Report 2581, IETF, April 1999.

[6] Damon Wischik, Mark Handley, and Marcelo Bagnulo Braun. The Resource Pooling Principle. *ACM SIGCOMM Computer Communication Review*, 38(5):47–52, 2008.

[7] T. Dreibholz, M. Becke, J. Pulinthanath, and E. P. Rathgeb. Applying TCP-Friendly Congestion Control to Concurrent Multipath Transfer. In *Proceedings of the IEEE 24rd International Conference on Advanced Information Networking and Applications (AINA)*, Perth/Australia, April 2010.

[8] R. Stewart, M. Ramalho, Q. Xie, M. Tüxen, and P. Conrad. Stream Control Transmission Protocol (SCTP) Partial Reliability Extension. Standards Track RFC 3758, IETF, May 2004.

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

[10] T. Dreibholz, A. Jungmaier, and M. Tüxen. A new Scheme for IP-based Internet Mobility. In *Proceedings of the 28th IEEE Local Computer Networks Conference (LCN)*, pages 99–108, Königswinter/Germany, November 2003. ISBN 0-7695-2037-5.

[11] R. Stewart, P. Lei, and M. Tüxen. Stream Control Transmission Protocol (SCTP) Packet Drop Reporting. Internet-Draft Version 09, IETF, Individual Submission, December 2009. draft-stewart-sctp-pktdrprep-09.txt, work in progress.

[12] T. Dreibholz and E. P. Rathgeb. Towards the Future Internet – An Overview of Challenges and Solutions in Research and Standardization. In *Proceedings of the 2nd GI/ITG KuVS Workshop on the Future Internet*, Karlsruhe/Germany, November 2008.

[13] P. Lei, L. Ong, M. Tüxen, and T. Dreibholz. An Overview of Reliable Server Pooling Protocols. Informational RFC 5351, IETF, September 2008.

[14] T. Dreibholz and E. P. Rathgeb. Overview and Evaluation of the Server Redundancy and Session Failover Mechanisms in the Reliable Server Pooling Framework. *International Journal on Advances in Internet Technology (IJAIT)*, 2(1):1–14, June 2009.

[15] 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.

[16] 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, March 2008. ISBN 978-963-9799-20-2.

[17] Wireshark. Wireshark: The World's Most Popular Network Protocol Analyzer, 2009. http://www.wireshark.org.

[18] A. Varga. *INET Framework for OMNeT++ 4.0*, 2009. http://inet.omnetpp.org/.

[19] T. Dreibholz. *NetPerfMeter Homepage*, 2009. http://www.iem.uni-due.de/~dreibh/netperfmeter/.

[20] S. Floyd and V. Jacobsen. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[21] 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, March 2009. ISBN 978-963-9799-45-5.

[22] M. Carbone and L. Rizzo. Dummynet Revisited. Technical report, Dipartimento di Ingegneria dell'Informazione, Università di Pisa, Pisa/Italy, May 2009.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8673
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8673