Evaluation of A New Multipath Congestion Control Scheme using the NetPerfMeter Tool-Chain

Thomas Dreibholz, Martin Becke, Hakim Adhari, Erwin P. Rathgeb
University of Duisburg-Essen, Institute for Experimental Mathematics
Ellernstraße 29, 45326 Essen, Germany
{dreibh,martin.becke,hakim.adhari,rathgeb}@iem.uni-due.de

***Abstract*—Multi-homed Internet sites become more and more widespread, due to the rising dispersal of inexpensive Internet access technologies combined with the growing deployment of resilience-critical applications. Concurrent Multipath Transfer (CMT) denotes the Transport Layer approach to utilise multiple network paths *simultaneously*, in order to improve application payload throughput. Currently, CMT is a quite hot topic in the IETF – in form of the Multipath TCP (MPTCP) and CMT-SCTP protocol extensions for TCP and SCTP. However, an important issue is still not fully solved: multipath congestion control.**

In order to support the IETF activities, we have set up a distributed Internet testbed for CMT evaluation. An important tool – which we have developed for multi-protocol Transport Layer performance analysis – is the Open Source NETPERFMETER tool-chain. It supports the parametrisation and processing of measurement runs as well as results collection, post-processing and plotting. However, its key feature is to support multiple Transport Layer protocols, which makes a quantitative comparison of different protocols – including state-of-the-art features like CMT – possible. In this paper, we first introduce NETPERFMETER and then show a proof-of-concept performance evaluation of CMT congestion controls which are currently discussed in the IETF standardisation process of CMT-SCTP.[1][2][3][4]

Keywords: NetPerfMeter Tool-Chain, Concurrent Multipath Transfer, Congestion Control, Dissimilar Paths, Performance Analysis

I. INTRODUCTION

The increasing deployment rate of resilience-critical Internet services leads to a steady growth in the number of multi-homed Internet sites. Technologies like ADSL (Asymmetric Digital Subscriber Line) make adding access redundancy possible at a competitive cost level. The availability of a secondary (or even more) access path to different Internet Service Providers (ISP) for redundancy reasons also leads to the demand for utilising all paths *simultaneously* – in order to increase application payload throughput. The idea of this so-called Concurrent Multipath Transfer (CMT) is to apply load sharing among all available paths. It is currently a hot topic in IETF standardisation, in form of the Multipath TCP (MPTCP) [1] extension for TCP as well as the CMT-SCTP [2], [3] extension for the Stream Control Transmission Protocol (SCTP) [4], [5]. However, Congestion Control (CC) for CMT is still an open topic: the possible existence of shared bottlenecks [6] may lead to an unfair bandwidth distribution. [7] provides a set of new CC strategies, which are simulatively evaluated. However, a proof of concept in a real network is missing.

In order to evaluate these new CCs, we have set up a testbed and developed Open Source tools – denoted as *NetPerfMeter* tool-chain – to perform performance evaluations and analyse the results. These tools and testbed setup experiences are also useful for other projects evaluating and comparing Transport Layer protocols performance – particularly with respect to distributed environments like G-LAB [8]. We finally apply *NetPerfMeter* to evaluate the CC schemes of [7] – on the example of CMT-SCTP – in a real-world, heterogeneous Internet setup. However, the results are protocol-independent and may be adapted to other ones – particularly to MPTCP – as well.

II. THE SCTP PROTOCOL

SCTP [4], [5] is a general-purpose, connection-oriented, unicast transport protocol which provides the reliable transport of user messages and a multi-homing concept out of the box. An SCTP connection between two peers is denoted as *association*. In an association, each peer is able to use multiple IP addresses. This is called *multi-homing*. Each IP address of the peer endpoint defines a unidirectional *path*. In each direction, one of the paths is chosen as *primary path*. This designated path is actually utilised for the data transmission; the other paths are only used for retransmissions. Upon failure of the primary path, it may be switched to one of the backup paths.

The user data is segmented into units of so-called DATA chunks. Each DATA chunk is identified by a unique Transmission Sequence Number (TSN). The receiver acknowledges the successful reception of DATA chunks by using a Selective Acknowledgement (SACK) chunk. Such a SACK chunk contains a cumulative acknowledgement (CumAck) which acknowledges all TSNs until a given TSN. It may furthermore acknowledge chunks above the CumAck in form of so-called GapAcks. This allows the sender instance to selectively retransmit only the particular chunks which are still missing in the reception sequence. A SACK chunk is sent back over the peer path of the last received DATA chunk packet. While CumAcks are obviously non-renegable (i.e. the receiver

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

[2]Parts of this work have been funded within the scope of the G-Lab project by the German Federal Ministry of Education and Research (Bundesministerium für Bildung und Forschung – BMBF).

[3]The authors would like to thank Robin Seggelmann and Michael Tüxen for their friendly support.

[4]The authors would like to thank the anonymous reviewers for their helpful comments.

may not "unacknowledge" a CumAck), it may revoke GapAcks. This could e.g. happen when the receiver queue gets too full to store earlier chunks which are necessary for the next CumAck. In order to improve efficiency – and to avoid retaining the non-outstanding GapAck'ed chunks in the send buffer – the Non-Renegable SACK extension (NR-SACK) [9] can be used to signalise non-renegable GapAcks.

Two different mechanisms are applied to handle retransmissions [5]: Fast Retransmissions (Fast RTX) are used to retransmit a DATA chunk *once*, after being GapAck'ed for 3 times by default [5]. Fast RTX occurs frequently, due to sporadic packet loss caused by concurrency. Timer-Based Retransmissions (Timer-Based RTX) are triggered by a timeout for any further retransmission. They should occur rarely and are usually a sign of severe congestion.

SCTP applies window-based congestion control [10] – similar to TCP – on each path, i.e. the amount of outstanding data (i.e. in flight through the network and/or not yet acknowledged by the receiver) is limited by the *congestion window* c_P of path P. AIMD (Additive Increase, Multiplicative Decrease) behaviour is used to adapt c_P to changing network conditions: on α newly acknowledged bytes on path P in a fully-utilised congestion window, c_P is adapted as follows [5]:

$$c_P = c_P + \begin{cases} \min\{\alpha, \mathrm{MTU}_P\} & (c_P \leq s_P) \\ \mathrm{MTU}_P & (c_P > s_P \land p_P \geq c_P) \end{cases} \tag{1}$$

Here, s denotes the so-called *slow-start threshold* s. For $c_P \leq s_P$, the increase phase is denoted as *slow start* and leads to an exponential growth of c_P. In slow start, SCTP applies Appropriate Byte Counting [11], i.e. c_P is only advanced by the minimum of the acknowledged bytes α and MTU_P. The other case, i.e. $c_P > s_P$, is denoted as *congestion avoidance* phase. Here, c_P is only increased by one MTU when a full window c_P has been acknowledged. The variable p_P ("partially acknowledged") counts the acknowledged bytes; when it reaches the threshold c_P, the window is increased and p_P reset. In case of a retransmission (i.e. Fast or Timer-Based) on path P, s_P and c_P are adapted as follows [5]:

$$s_P = \max\left\{c_P - \frac{1}{2} * c_P, 4 * \mathrm{MTU}_P\right\} \tag{2}$$

$$c_P = \begin{cases} s_P & \text{(Fast RTX)} \\ \mathrm{MTU}_P & \text{(Timer-Based RTX)} \end{cases} \tag{3}$$

That is, on a Fast RTX, the sender remains in congestion avoidance (and is therefore able to quickly recover from the frequently occurring sporadic packet loss); on Timer-Based RTX – which is a sign for severe congestion – c_P slow-starts again from one MTU_P.

The CMT-SCTP extension [2], [3] adds CMT support to SCTP. It is used to distribute data across multiple paths in order to improve the payload throughput. CMT-SCTP is quite straightforward; it just adds a few mechanisms to improve retransmission, congestion window update and acknowledgements handling (for examples, see [12]). To efficiently handle dissimilar paths – i.e. paths having different QoS characteristics like bandwidths, bit error rates and delays – so-called *Buffer Splitting* techniques [13], [14] in combination with NR-SACKs [9] are necessary to solve blocking issues.

III. Multipath Congestion Control

Plain CMT-SCTP CC handles each path independently, i.e. each path shows the same AIMD behaviour as a normal TCP flow. In result, having n paths sharing a single bottleneck, a CMT-SCTP flow gets n times the bandwidth of a concurrent TCP flow – which is quite unfair [6], [7].

Key idea for fair multipath CC is Resource Pooling (RP), which denotes "making a collection of resources behave like a single pooled resource" [15]. That is, the set of all paths should behave like a single high-capacity path [6], fulfilling the following goals [16]: (1) In comparison to a single-homed flow via the best path, a CMT flow should get at least the same (or – if possible – better) bandwidth. (2) On a shared bottleneck path, a CMT flow should not take more bandwidth than a single-homed flow. (3) Congestion should be balanced among all paths of a CMT flow.

CMT/RPv1 CC [6] scales the congestion window increment (see Equation 1) by the so-called *slow-start threshold ratio* $\hat{s}_P = \frac{s_P}{\sum_i s_i}$, and tries to halve the total congestion window ($\sum_i c_i$) on decrement (see Equation 1 and 3). While CMT/RPv1 CC works well for similar paths [6], the simulations in [7] show performance issues for dissimilar paths.

These issues are solved by CMT/RPv2 CC [7], which scales the congestion window increment c_P on path P (see Equation 1) by the so-called *increase factor* $\hat{i}_P = \frac{\frac{c_P}{\mathrm{RTT}_P}}{\sum_i \frac{c_i}{\mathrm{RTT}_i}}$, which denotes the bandwidth ratio of P on the total bandwidth of the association. On decrement, it is tried to halve the *bandwidth* of the flow, not the total congestion window.

The MPTCP CC [16] – defined for MPTCP – is also based on RP. The main difference to CMT/RPv1 and CMT/RPv2 is that, while CMT/RP tries to halve the *total* congestion window/total bandwidth on a retransmission on path P, MPTCP CC behaves exactly like standard TCP or SCTP by only halving the *path* congestion window c_P. To ensure fairness, MPTCP uses the idea of controlling engineering: growth and decrease of c_P have to be brought into equilibrium by adapting the congestion window growth by a per-flow *aggressiveness factor* $\hat{a}$ (see [7], [16] for derivation and details). That is, $\hat{a}$ is used to scale the congestion window increment c_P on path P (see Equation 1).

While Plain-CMT-SCTP, CMT/RPv1, CMT/RPv2 for SCTP, as well as MPTCP CC for MPTCP have been examined by simulations [7], a comparison of all four approaches in a real network had been missing.

IV. The NetPerfMeter Tool-Chain

In order to evaluate the CCs mentioned, we had first considered to apply one of several existing tools. Each tool has its own advantages and disadvantages. However, none of the existing tools has fulfilled our requirements, and we have finally designed our own multi-platform and Open Source tool-chain: NetPerfMeter.

In fact, the first difficulty we were confronted with was that most of the known tools are designed to work

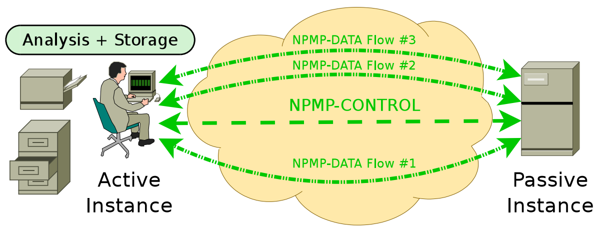

Figure 1. The Concept of a NetPerfMeter Measurement

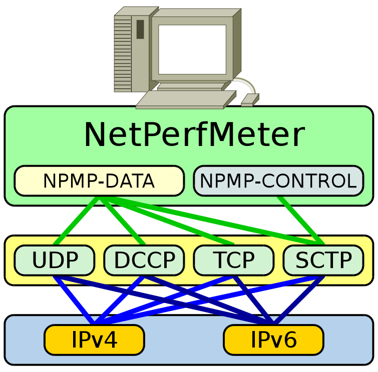

Figure 2. The NetPerfMeter Protocol Stack

mainly with TCP and UDP. Tools such as TTCP or NTTCP do not support SCTP. Modified versions of IPERF are available and are able to support this protocol. However, only basic functionalities are possible, due to the lack of parametrisation possibilities. On the other hand, TSCTP is a powerful SCTP tool, which makes it possible to address most of the important SCTP-specific parameters. However, this tool is only designed for SCTP and does not support other Transport Layer protocols.

In our self-developed tool, it is possible to use UDP, DCCP, TCP as well as SCTP (including multi-homing and the parametrisation of extensions like CMT-SCTP [2], [13], [17]). It allows to configure multiple transport connections, denoted as flows, between two systems using varying parameters. Furthermore, each flow may use its own Transport Layer protocol. In fact, independent traffic configuration is possible for each stream of an SCTP association, as well as bidirectional data transmission and on-off traffic (i.e. setting time stamps when to start, stop and restart a flow or sub-flow) are possible. One example configuration is to start with one SCTP flow, to start a second SCTP flow after 10 s and a TCP flow after 30 s. This example shows the advantage of our tool in comparison with other existing tools, since none of them is able to manage different flows with different protocols out of the box, and to start the flows at different times without having to write additional scripts.

Another criterion related with network testing tools is whether and how the results of an experiment are recorded. Here, basic functionalities are supported by other tools. IPERF, for example, generates text log files. With NETPERF, it is possible re-direct the results shown on the screen into a text file. In comparison to it, NETPERFMETER makes it possible to directly write the results as scalar and in particular as vector files, which simplifies the creation of plots showing the results.

To ease the configuration and measurement process, the parametrisation of a run is only necessary at *one side* of the communication, as illustrated in Figure 1: the remote NETPERFMETER node (denoted as *passive instance*), accepts incoming NETPERFMETER connections from the local node (denoted as *active instance*). The passive instance is fully controlled by the active instance, by using a control protocol denoted as NETPERFMETER Control Protocol (NPMP-CONTROL). That is, the passive instance may run as daemon; no manual interaction – e.g. restart before a new measurement run – is required. This feature is highly practical for a setup distributed over multiple Internet sites and allows for parameter studies consisting of many measurement runs. The payload data between active and passive instances is transported by using the NETPERFMETER Data Protocol (NPMP-DATA). Figure 2 illustrates the protocol stack to make the protocol interaction clearer.

A new measurement run is initiated by first establishing an NPMP-CONTROL association (using SCTP for transport) to the passive instance. Then, the configured NPMP-DATA connections are established by using their configured Transport Layer protocols. The passive instance is informed about the identification and parameters of each new flow by using NPMP-CONTROL Add Flow messages; on startup of the NPMP-DATA flow, an NPMP-DATA Identify message allows the mapping of a new incoming connection to a configured flow by the passive NETPERFMETER instance. It acknowledges each newly set up flow by an NPMP-CONTROL Acknowledge message. After sequentially setting up all flows, the scenario is ready to start the measurement run.

The actual measurement run is initiated by using an NPMP-CONTROL Start Measurement message, which is acknowledged by an NPMP-CONTROL Acknowledge message. Then, both instances start running the configured scenario by transmitting NPMP-DATA Data messages over their configured flows. During the measurement run, incoming and outgoing flow bandwidths may be recorded as vectors (i.e. time series) at both instances, since NPMP-DATA Data traffic may be bidirectional. Furthermore, the CPU utilisations – separately for each CPU and CPU core – are also tracked. This allows to identify performance bottlenecks, which is particularly useful when debugging and comparing transport protocol implementation performances. Furthermore, if clocks are appropriately synchronised, the one-way delay of messages can be recorded. To use this feature, the clocks of both instances need to be appropriately synchronised by using the Network Time Protocol (NTP) [18].

The end of a measurement run is initiated by the active NETPERFMETER instance by using an NPMP-CONTROL Stop Measurement message. It is also acknowledged by an NPMP-CONTROL Acknowledge message. At the end of the measurement, average bandwidth and one-way delay of each flow are recorded as scalars (i.e. single values). They may provide an overview of the long-term performance.

After stopping the measurement, the flows are sequentially removed, triggered by NPMP-CONTROL Remove Flow messages. On flow removal, the passive instance archives vector and scalar results of the corresponding flow and reports them to the active instance by using NPMP-CONTROL Results messages. The active instance, as well, archives its local vector and scalar

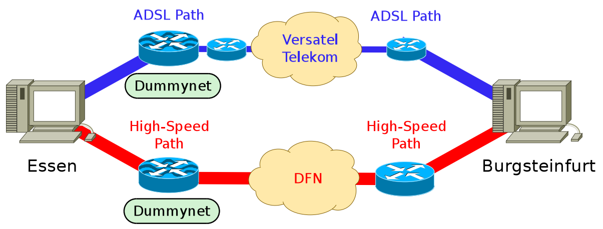

Figure 3. The Testbed and Simulation Setup

statistics and stores them – together with the results received from its peer – locally. All data is compressed by using BZIP2 [19], which may save a significant amount of bandwidth[5] and disk space.

By using shell scripts, it is possible to apply NETPERFMETER for parameter studies, i.e. to create a set of runs for each input parameter combination. For example, a script could iterate over a number of flows n from 1 to 5 and a number of frame sizes f_S from 100 bytes to 1000 bytes and perform 5 measurement runs for each combination. Then, the NETPERFMETER tool `createsummary` combines the scalar results from each run into a table, containing columns for n and f_S as well as the flow number and its resulting average payload throughput. For disk space saving, the resulting tables are again compressed by using BZIP2 [19]. These tables may be read by the statistics and plotting language GNU R [20], which is used by our tool `plotter.R` to post-process (i.e. filter parameter ranges, compute 95% confidence intervals, etc.) and finally plot the results. `plotter.R` is shared with our OMNET++ simulation processing tool-chain SIMPROCTC; a detailed description is provided in [21].

NETPERFMETER has been designed with reusability in mind. Particularly, it has been developed to support research in the G-LAB [8] plattform. Therefore, it has been released as Open Source under GPLv3 license. Currently, Linux, FreeBSD and MacOS X are supported; it is freely available at [22]. Also, it has been contributed to Debian and Ubuntu Linux (allowing to install it directly from the distributions' standard package repositories) as well as to FreeBSD (allowing to install it from the FreeBSD ports collection). MacOS X and Solaris are also supported.

Furthermore, in order to support teaching and debugging purposes, dissectors for NPMP-CONTROL and NPMP-DATA have been contributed to the Wireshark [23] network analysis tool. Also, the IANA has assigned SCTP Payload Protocol Identifiers and a DCCP Service Code for the NETPERFMETER protocols.

V. OUR CMT-SCTP TESTBED

Our experimental testbed setup is shown in Figure 3. Sender and receiver are running FreeBSD 8.2 (already including CMT/RPv1 CC) with CMT/RPv2 and MPTCP CCs added. NETPERFMETER runs on sender (active instance) and receiver (passive instance). The sender located in Essen/Germany is connected to the receiver in Burgsteinfurt/Germany through two different paths which are known to be disjoint: the first path uses a high-speed fibre optic connection over the German Research Network (Deutsches Forschungsnetz – DFN), with a typical RTT of 4 ms. The second path uses an ADSL connection in Essen (Versatel; 800 Kbit/s upstream) as well as an ADSL connection in Burgsteinfurt (Telekom; 16 Mbit/s downstream); the typical RTT is 56 ms.

In order to perform experiments with different network characteristics (here: bandwidth), the FreeBSD packet filtering mechanism DUMMYNET [24] is used to apply traffic limitations on the routers connecting the host in Essen to the ISPs. In case of the DFN network, this has been realised without any problem by using DUMMYNET on the first router. In contradiction, performing the same procedure for the DSL path has been more complicated. The router delivered by the ADSL ISP has to be used for establishing the connection. However, its configuration had been fixed by the ISP – with no possibility to reconfigure it[6]. Therefore, we have put another FreeBSD router – also running DUMMYNET – between the sender host and ADSL router. In order to subnet the IP network provided by the ADSL ISP, we have furthermore applied Proxy ARP on the FreeBSD router (since the locked-down ADSL router also did not provide a way to configure subnets).

One more restriction caused by the ADSL network has been the lack of support for IPv6. Native IPv6 connectivity is still very unusual for end-users in Germany. Most of the ISPs – including our ADSL ISP – still do not support it, yet. With IPv6 connectivity only via the DFN network, we therefore had to restrict the measurements in this paper to IPv4 only.

Furthermore we have also started to investigate an integration of our experiments into global platforms such as G-LAB [8]. These platforms consist of a group of computers, which are available as a large testbed for computer networking and distributed systems research. A research project has access to a so-called *slice*, which consists of virtual machine access to a subset of the nodes. Virtual links may be combined with real-existing physical links (like an ADSL connection) to set up complex topologies and perform experiments in these setups. This will bring significant benefits in terms of scalability, extensibility and reliability.

VI. SYSTEM SETUP

For our performance evaluation of the CMT-SCTP CCs introduced in Section III, we have used the following configuration parameters, unless otherwise specified: The senders have been saturated (i.e. sending as much as possible); the message size has been 1,444 bytes at an MTU of 1,492 bytes (i.e. MTU-sized packets [5] on the ADSL path applying PPPoE). All messages have used unordered delivery. CMT-SCTP has been applied for the first flow; the second one – denoted as reference flow – has not applied CMT. The send buffer has been set to 1,000,000 bytes, the receive buffer has been set to 500,000 bytes; Buffer splitting as defined in [13] (provided by FreeBSD kernel SCTP) has been applied. The duration of each throughput measurement has been 5 min. Tests have been repeated 5 times in order to ensure a sufficient statistical accuracy. The results plots show the average values and their corresponding 95% confidence intervals.

[5]Of course, the passive node compresses the data *before* transmission.

[6]We have been restrained from a password recovery procedure, since this would have violated the ADSL contract with the ISP.

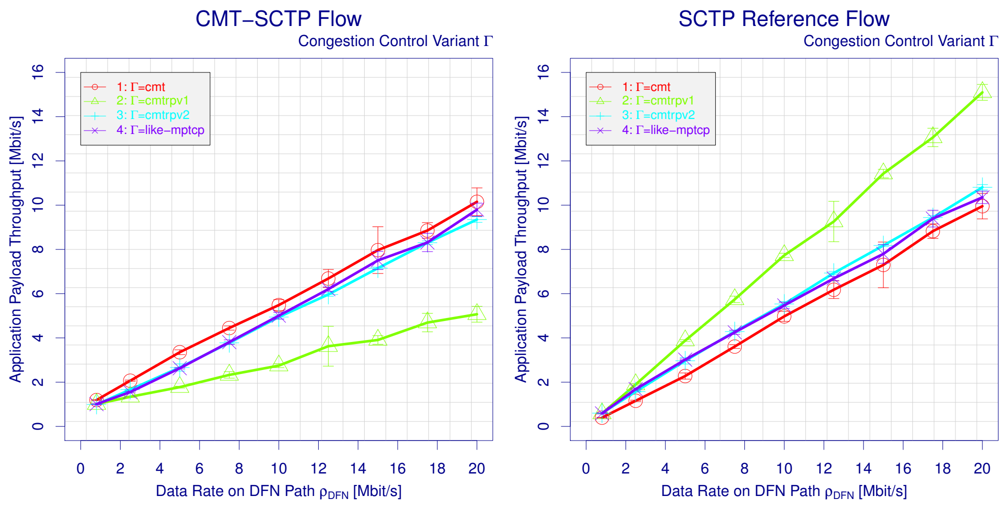

Figure 4. Application Payload Throughput for CMT-SCTP versus SCTP Reference Flow

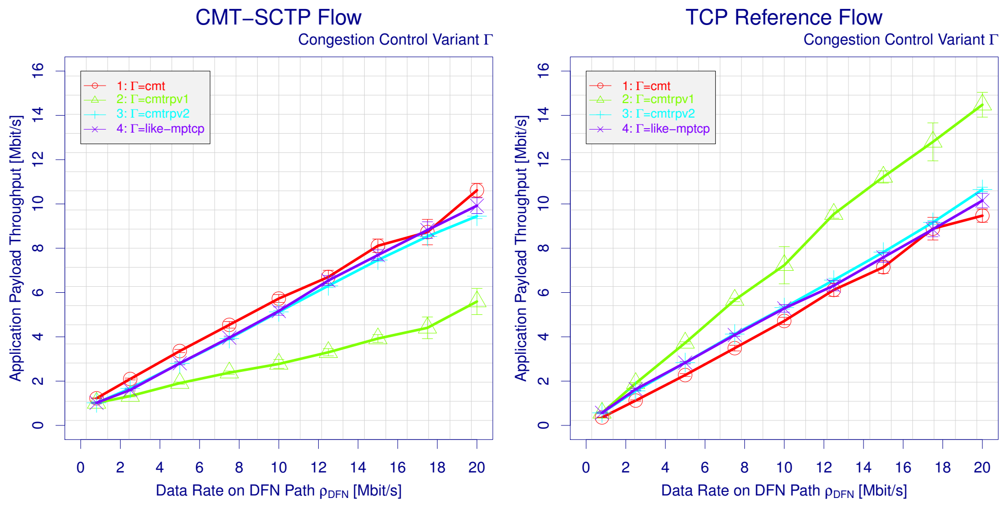

Figure 5. Application Payload Throughput for CMT-SCTP versus TCP Reference Flow

VII. Evaluation

A. *CMT-SCTP versus SCTP*

In our first measurement, we have examined the performance of CMT-SCTP versus standard SCTP. The bandwidth of the DFN path ρ_{DFN} has been varied from 800 Kbit/s to 20 Mbit/s, while the bandwidth of the ADSL path has been fixed at 800 Kbit/s (ADSL uplink speed). This scenario is challenging, due to the dissimilarity of the paths (different RTTs, different bandwidths) and in particular due to so-called Buffer Bloat [25], which is caused here by the typically long ADSL modem queue; see also [13] for more details on this subject. The resulting application payload throughputs are displayed in Figure 4 for the CMT-SCTP flow (left-hand plot) and the standard SCTP reference flow (right-hand plot) for the four CCs (i.e. Γ=cmt for plain CMT-SCTP CC; Γ=cmtrpv1 for CMT/RPv1 and Γ=cmtrpv2 for CMT/RPv2 respectively, as well as Γ=like-mptcp for MPTCP CC.

Clearly, plain CMT-SCTP CC performs as expected: at ρ_{DFN}=20 Mbit/s, the CMT-SCTP flow achieves a payload throughput of about 10.5 Mbit/s (i.e. it gets half of the ρ_{DFN}=20 Mbit/s shared with the reference flow, as well as the 800 Kbit/s of the ADSL link exclusively). The reference flow achieves the corresponding throughput (i.e. about 9.6 Mbit/s) on its single path.

The performance of CMT/RPv1 is obviously bad, due to the very different slow-start thresholds on both paths. Due to the Buffer Bloat caused by the ADSL modem queue, the congestion window c_{ADSL} grows very large in comparison to c_{DFN}, leading to corresponding slow-start thresholds s_{ADSL} and s_{DFN} – and in result to a slowly-growing congestion window c_{DFN} on the fast path (due to small slow-start threshold ratio $\hat{s}_{\text{DFN}}$). Even worse, due to the large s_{ADSL}, c_{DFN} has to restart from one MTU each time a retransmission occurs on the DFN path. This happens frequently – due to concurrency with the reference flow. In result, the achieved application payload throughput is less than 5 Mbit/s at ρ_{DFN}=20 Mbit/s. This clearly violates RP goal #1 of Section III.

CMT/RPv2 as well as MPTCP-like CC behave as expected: for the CMT-SCTP flow, the achieved throughput is slightly less than the throughput for plain CMT-SCTP CC. This is an intended effect of RP, which tries to shift congestion to less-congested paths (goal #3 of Section III). Since the CMT-SCTP flow may exclusively use the 800 Kbit/s of the ADSL path, it is less aggressive on the DFN path. In result, both flows achieve a quite similar payload throughput (e.g. about 10 Mbit/s at ρ_{DFN}=20 Mbit/s. That is, both CCs fulfil the goals set for RP in an CMT-SCTP-versus-SCTP scenario.

B. *CMT-SCTP versus TCP*

In order to show the performance of CMT-SCTP on concurrency against a TCP flow, we have repeated the measurement described in Subsection VII-A with the SCTP reference flow replaced by a TCP one. The corresponding application throughput results are shown in Figure 5 for the CMT-SCTP flow (left-hand plot) and the TCP reference flow (right-hand plot).

Obviously, the results are very similar to the previous CMT-SCTP versus SCTP results. The CMT-SCTP flow throughput for CMT/RPv2 as well as MPTCP-like CC are slightly lower than for plain CMT-SCTP CC – due to a shifting of congestion to fulfil RP goal #3 – and CMT/RPv1 is unsuitable in the dissimilar path setup. The particularly important result of this measurement is the fact that the CMT transport does not introduce unfairness when concurring with TCP flows. This point is highly crucial, since a major prerequisite for standardising CMT protocol extensions in the IETF is to not cause unfairness to the widely deployed TCP protocol.

VIII. Conclusions

In order to examine the multipath congestion controls which are currently under discussion in the IETF within the scope of standardisation activities for the CMT protocol extensions CMT-SCTP and MPTCP, we have set up a distributed CMT testbed based on a real-world Internet setup. To actually run measurements, we have developed our Open Source NetPerfMeter tool-chain, which helps researchers to perform tests in the testbed – as well as in upcoming large-scale, multi-homing-capable, virtualised network testbeds like the G-Lab. Our tool-chain is not only useful for our congestion control evaluation. It may also be used for other network performance research purposes as well.

By using the NetPerfMeter tool-chain in our Internet testbed setup, we have performed a proof-of-concept performance evaluation of the four relevant CMT congestion controls: plain CMT-SCTP, CMT/RPv1, CMT/RPv2, as well as MPTCP-like. We have shown that both, CMT/RPv2 as well as MPTCP-like congestion control, fulfil the set goals and are suitable for handling CMT traffic in a heterogeneous Internet setup.

As part of future work, we are going to perform parameter studies using a wider range of parameters in more complex setups (e.g. based on G-Lab). Furthermore, we are also going to simulatively evaluate the congestion control approaches. We will use these results to formalise the fairness criteria of multipath transport, in order to provide a more fine-granular classification. Finally, we are also going to contribute our results into the ongoing IETF standardisation process of CMT-SCTP and MPTCP.

References

[1] A. Ford, C. Raiciu, M. Handley, S. Barré, and J. R. Iyengar, "Architectural Guidelines for Multipath TCP Development," IETF, Informational RFC 6182, Mar. 2011, ISSN 2070-1721.

[2] J. R. Iyengar, P. D. Amer, and R. Stewart, "Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths," *IEEE/ACM Transactions on Networking*, vol. 14, no. 5, pp. 951–964, Oct. 2006, ISSN 1063-6692.

[3] M. Becke, T. Dreibholz, J. R. Iyengar, P. Natarajan, and M. Tüxen, "Load Sharing for the Stream Control Transmission Protocol (SCTP)," IETF, Network Working Group, Internet Draft Version 02, July 2011, draft-tuexen-tsvwg-sctp-multipath-02.txt, work in progress.

[4] T. Dreibholz, I. Rüngeler, R. Seggelmann, M. Tüxen, E. P. Rathgeb, and R. R. Stewart, "Stream Control Transmission Protocol: Past, Current, and Future Standardization Activities," *IEEE Communications Magazine*, vol. 49, no. 4, pp. 82–88, Apr. 2011, ISSN 0163-6804.

[5] R. R. Stewart, "Stream Control Transmission Protocol," IETF, Standards Track RFC 4960, Sept. 2007, ISSN 2070-1721.

[6] T. Dreibholz, M. Becke, J. Pulinthanath, and E. P. Rathgeb, "Applying TCP-Friendly Congestion Control to Concurrent Multipath Transfer," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, Perth/Australia, Apr. 2010, pp. 312–319, ISSN 1550-445X.

[7] T. Dreibholz, M. Becke, H. Adhari, and E. P. Rathgeb, "On the Impact of Congestion Control for Concurrent Multipath Transfer on the Transport Layer," in *Proceedings of the 11th IEEE International Conference on Telecommunications (ConTEL)*, Graz/Austria, June 2011, pp. 397–404, ISBN 978-953-184-152-8.

[8] M. Kleis, J. Müller, A. Siddiqui, and M. Becke, "Evaluating a Future Internet Cross-Layer Composition Prototype," in *Proceedings of the 7th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom)*, Shanghai/China, Apr. 2011.

[9] P. Natarajan, N. Ekiz, E. Yilmaz, P. D. Amer, and J. R. Iyengar, "Non-Renegable Selective Acknowledgments (NR-SACKs) for SCTP," in *Proceedings of the 16th IEEE International Conference on Network Protocols (ICNP)*, Orlando, Florida/U.S.A., Oct. 2008, pp. 187–196, ISBN 978-1-4244-2506-8.

[10] M. Welzl, *Network Congestion Control: Managing Internet Traffic*. Chichester, West Sussex/United Kingdom: John Wiley & Sons, 2005, ISBN 978-0-470-02528-4.

[11] M. Allman, "TCP Congestion Control with Appropriate Byte Counting (ABC)," IETF, RFC 3465, Feb. 2003, ISSN 2070-1721.

[12] T. Dreibholz, M. Becke, J. Pulinthanath, and E. P. Rathgeb, "Implementation and Evaluation of Concurrent Multipath Transfer for SCTP in the INET Framework," in *Proceedings of the 3rd ACM/ICST International Workshop on OMNeT++*, Torremolinos, Málaga/Spain, Mar. 2010, ISBN 978-963-9799-87-5.

[13] H. Adhari, T. Dreibholz, M. Becke, E. P. Rathgeb, and M. Tüxen, "Evaluation of Concurrent Multipath Transfer over Dissimilar Paths," in *Proceedings of the 1st International Workshop on Protocols and Applications with Multi-Homing Support (PAMS)*, Singapore, Mar. 2011, pp. 708–714, ISBN 978-0-7695-4338-3.

[14] T. Dreibholz, M. Becke, E. P. Rathgeb, and M. Tüxen, "On the Use of Concurrent Multipath Transfer over Asymmetric Paths," in *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, Miami, Florida/U.S.A., Dec. 2010, ISBN 978-1-4244-5637-6.

[15] D. Wischik, M. Handley, and M. B. Braun, "The Resource Pooling Principle," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 5, pp. 47–52, Oct. 2008, ISSN 0146-4833.

[16] C. Raiciu, M. Handley, and D. Wischik, "Practical Congestion Control for Multipath Transport Protocols," University College London, London/United Kingdom, Tech. Rep., 2009.

[17] T. Dreibholz and M. Becke, "SCTP Socket API Extensions for Concurrent Multipath Transfer," IETF, Network Working Group, Internet Draft Version 01, May 2011, draft-dreibholz-tsvwg-sctpsocket-multipath-01.txt, work in progress.

[18] D. L. Mills, J. Martin, J. Burbank, and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms," IETF, Standards Track RFC 5905, June 2010, ISSN 2070-1721.

[19] J. Seward, *bzip2 – A Program and Library for Data Compression*, Dec. 2007.

[20] R Development Core Team, *R: A Language and Environment for Statistical Computing*, Vienna/Austria, July 2011.

[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 International Workshop on OMNeT++*, Rome/Italy, Mar. 2009, pp. 1–8, ISBN 978-963-9799-45-5.

[22] T. Dreibholz, "NetPerfMeter Homepage," 2011, http://www.iem.uni-due.de/ dreibh/netperfmeter/.

[23] Wireshark, "Wireshark: The World's Most Popular Network Protocol Analyzer," 2011.

[24] M. Becke, T. Dreibholz, E. P. Rathgeb, and J. Formann, "Link Emulation on the Data Link Layer in a Linux-based Future Internet Testbed Environment," in *Proceedings of the 10th International Conference on Networks (ICN)*, St. Maarten/Netherlands Antilles, Jan. 2011, pp. 92–98, ISBN 978-1-61208-002-4.

[25] J. Gettys, "Bufferbloat – Dark Buffers in the Internet," Jan. 2011.