Evaluation of Concurrent Multipath Transfer over Dissimilar Paths

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

Michael Tüxen
Münster University of Applied Sciences
Department of Electrical Engineering and Computer Science
Bismarckstraße 11, 48565 Steinfurt, Germany
tuexen@fh-muenster.de

***Abstract*—The steadily growing deployment of resilience-critical Internet services is leading to an increasing number of Multi-Homed network sites. Asymmetric Digital Subscriber Lines (ADSL) are an inexpensive way to add a secondary Internet access connection. With the development of Multi-Path Transport Layer protocols – like Multipath TCP (MPTCP) and the Stream Control Transmission Protocol (SCTP) furnished by a Concurrent Multipath Transfer (CMT-SCTP) extension – there is also a strong interest in utilising all access connections *simultaneously* to improve the data throughput of the applications. However, combining network paths over ADSL with paths over other access technologies like fibre optic links implies highly dissimilar paths with significantly different bandwidths, delays and queuing behaviours. Efficient Multi-Path transport over such dissimilar paths is a challenging task for the new Transport Layer protocols under development.**

In this paper, we show the difficulties of Multi-Path transport in a real-world dissimilar path setup which consists of a high-speed fibre optic link and an ADSL connection. After that, we present an optimised buffer handling technique which solves the transport efficiency issues in this setup. Our optimisation is first analysed by simulations. Finally, we also show the usefulness of our approach by experimental evaluation in a real Multi-Homed Internet setup.[1][2][3][4]

Keywords: Concurrent Multipath Transfer, Dissimilar Paths, Buffer Handling, Performance Analysis, Experimental Validation

I. Introduction

Using more than one interface to networks like the Internet is a common feature of currently available communication devices such as laptops or smart phones. Using an IP-based protocol stack allows different network carriers and technologies to be part of the communication. The existence of multiple IP addresses is denoted as *Multi-Homing*. An example could be the simultaneous use of different access media – like the widespread Asymmetric Digital Subscriber Line (ADSL) and Universal Mobile Telecommunications System (UMTS) – within a single communication device. However, current standards focus on the use of a *single* network interface and only take advantage of additional interfaces for providing mobility or for increasing availability. Therefore, it seems to be a natural evolution to aggregate bandwidths in order to generate throughput benefits. The load balancing of data – which is denoted as *Multi-Path Transfer* – may be realised on different layers of the OSI model, such as the Application, the Transport or the Network Layer. However, the Transport Layer stands out by being the only layer which is able to realise a path-transparent congestion control [1], [2]. Furthermore, a Transport Layer-based approach does not require to modify the applications (which may be numerous) or change the Network Layer protocol (i.e. IPv4/IPv6).

Multi-Homing support on more than one endpoint is an absolute prerequisite to provide Multi-Path Transfer on the Transport Layer. However, the most common Transport Layer protocol in the Internet – the Transmission Control Protocol (TCP) – does not support any Multi-Homing features. The Multipath TCP (MPTCP) [3] approach defines a modification of TCP to support more than one IP address per endpoint within the same connection. For an endpoint, each peer address describes a unidirectional path to the remote instance. Multi-Path Transfer is applied on these paths.

Another Transport Layer protocol, which provides Multi-Homing support out of the box, is the Stream Control Transmission Protocol (SCTP) [4]. Similar to MPTCP, paths are described by IP addresses of the remote endpoint. However, standard SCTP just uses one selected path per transport direction to actually transmit user data – unless a failure occurs. CMT-SCTP [5] is an extension of SCTP to provide Multi-Path Transfer. A further extension – denoted as CMT/RP-SCTP [6] – applies the approach of Resource Pooling (RP) [1] – to realise TCP-friendly Multi-Path congestion control.

Independently of the used Transport Layer protocol, the Multi-Path Transfer of data over dissimilar paths – i.e. paths having different characteristics like bandwidths, delays and queuing behaviours – is challenging and still an open issue. In this paper, we present an approach to cope with these

[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 Irene Rüngeler for her friendly support regarding INET and Randall Stewart for initial discussions about Receiver Buffer Splitting.

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

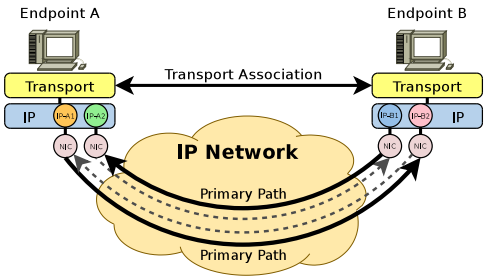

Figure 1. Multi-Path Transport between Multi-Homed Endpoints

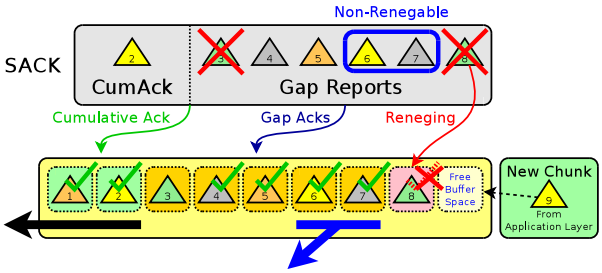

Figure 2. Retransmission Queue and Selective Acknowledgements

challenges by optimising the buffer handling of the communication instances. Our approach will first be evaluated by simulations; after that we also prove its usefulness by experimental validation in a real Multi-Homed network setup.

II. Reliable Transport between Multi-Homed Systems

In this paper, we use SCTP as the Transport Layer protocol to evaluate our approach. However, our solutions could be transferred to other protocols – like MPTCP – as well.

A. Basics

SCTP [4], as illustrated in Figure 1, selects a so-called *Primary Path* in each transport direction from the set of all paths to a peer endpoint. The Primary Path is used for the actual user data transmission; all other paths remain as backup and are only used in case of failures on the Primary Path. If necessary, the Primary Path can be switched.

The SCTP protocol provides a connection-oriented, unicast and TCP-like congestion-controlled transport of user messages within multiple independent streams. An SCTP connection including all of its streams is denoted as *Association*. The user messages can be bundled into a single SCTP packet by using one or multiple chunks, denoted as DATA chunks. Each DATA chunk is identified by a unique Transmission Sequence Number (TSN). The TSNs are used to assure reliability as well as to allow possible chunk reordering on the receiver side. While TCP always transfers data in an ordered way, SCTP also has the possibility to pass data to the application in an unordered way. SCTP provides reliable transport. That is, DATA chunks are acknowledged by the receiver and gaps in the TSN sequence (i.e. missing DATA chunks) are reported to the sender by the so-called Selective Acknowledgement (SACK) chunks.

In order to cope with changing network conditions, SCTP uses congestion control mechanisms which are similar to TCP. For each path, a congestion window applying AIMD (Additive Increase, Multiplicative Decrease) behaviour is used to avoid network congestion. In addition to it, and in order to avoid receiver overload, the flow control mechanism is used by utilising a receiver window to protect the sink. The receiver window is calculated by the sender on base of the Advertised Receiver Window (A_RWND). The A_RWND is reported as part of a SACK chunk. The receiver window itself is the sum of A_RWND and the amount of data, which is still in flight and/or not yet acknowledged by the receiver. This amount of data is commonly denoted as *outstanding data* and is counted in bytes.

B. Selective Acknowledgements

A functionality example of the SCTP acknowledgement mechanism is illustrated in Figure 2. In the ideal case, DATA chunks are received in their original sequence and without any gaps. In this case, two data chunks (here: TSN #1 and TSN #2) are transmitted, successfully received and acknowledged by a Cumulative Acknowledgement (CumAck). This CumAck uses the last TSN without any gaps (here: TSN #2) and is also the trigger for the sender to release the memory occupied by the TSNs up to #2 in the send buffer. This memory is occupied by the corresponding user messages to make a retransmission possible.

DATA chunks with a TSN above the TSN of a CumAck are noticed in a gap report as part of a SACK chunk, in order to optimise the retransmission behaviour (GapAck; here: TSN #4 to TSN #5). In contrast to TSNs acknowledged by a CumAck, the DATA chunks in the gap report are renegable. That is, the receiver may revoke their acknowledgement at any time and request their retransmission. This revocation of an acknowledgement is denoted as *reneging* (here: TSN #8).

That is, in case of standard SCTP, a GapAck'ed chunk has to stay in the send buffer until it is finally acknowledged by a CumAck. The Non-Renegable Selective Acknowledgements (NR-SACKs) protocol extension [7] adds the possibility of marking selected GapAcks as non-renegable. In Figure 2, the DATA chunks with TSN #6 and TSN #7 are marked this way. The sender does not have to hold non-renegably acknowledged chunks in the send buffer any more, giving it the possibility to free buffer space earlier and reuse it for new DATA chunks.

C. Multi-Path Transfer

Multi-Path Transfer support for SCTP is provided by the Concurrent Multipath Transfer extension (CMT-SCTP) [5], [8]. Since SCTP already supports Multi-Homing out of the box, this extension is small in comparison to the effort of adapting plain TCP to MPTCP. CMT-SCTP just adds some further mechanisms to the retransmission, congestion control and acknowledgement handling. CMT-SCTP works well on homogeneous paths, i.e. paths with similar characteristics like bandwidth, delay and error rate. However, applying CMT-SCTP over dissimilar paths leads to some challenges [9], [10].

III. Buffer Blocking on Dissimilar Paths

Problems of Multi-Path Transfer over dissimilar paths are blocking issues at the send and receive buffers. We categorise these issues as described in the following subsections.

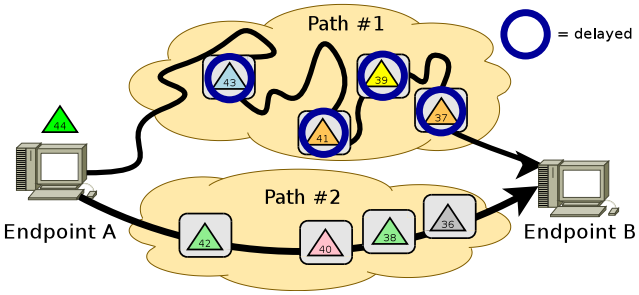

Figure 3. Receiver Buffer Blocking

A. Window-Induced Receiver Buffer Blocking

An example for the blocking issue at the receive buffer – which we denote as *Window-Induced Receiver Buffer Blocking* – is illustrated in Figure 3: endpoint A is trying to send all of its DATA chunks using unordered delivery to endpoint B. These chunks are numbered from TSN #36 to TSN #44; a round robin scheduling [9] between the two paths is applied. Path #1 has a large delay and large queuing capacity (e.g. a long queue on an ADSL modem). The delay and queuing capacity of path #2 are much smaller.

TSN #36 is the first chunk reaching Endpoint B; it is directly delivered to the application layer. TSN #37 to TSN #41 are still outstanding. The sender calculates the *receiver window* for the association based on A_RWND and the number of already-outstanding bytes. The larger the number of outstanding bytes, the smaller the actual possibility to transmit new DATA chunks into the network. Once the long-delay path #1 has a high number of bytes outstanding, A_RWND limits the possibility to send more data on path #2.

In order to fully utilise its capacity, a path needs to steadily have at least a number of bytes given by its bandwidth-RTT product outstanding. If it cannot have this amount of data in flight, its throughput will suffer. A problem will always occur if the queue capacity of path #1 is larger than the receive buffer capacity of endpoint B. In this example, the receive buffer of endpoint B has the capacity to store up to six full-sized packets. Therefore, the calculated receiver window variable for endpoint B maintained in the sender instance of endpoint A will be decreased by the queued data on the slow path #1. When more and more data is buffered on path #1, endpoint A decreases the calculated receiver window of endpoint B down to zero.

In this example, this is the reason for the sender to interrupt its transmission until a SACK for TSN #38 arrives. This triggers a new receiver window calculation and restarts sending[5] with TSN #42 on path #2. This transmission decreases the receiver window to zero again – and the sender has to wait until TSN #40 is acknowledged. Then, the sender can again continue transmitting. The same happens to TSN #43 and the sender has to wait before it is able to send it. The large buffer capacity of path #1 works as a queue ahead of the receive buffer. In case of just sending over one path, this path characteristic has no effect: in this case, it is just a neutral element for the flow control.

[5]We assume that the data size of TSN #42 is larger than the Silly Window Syndrome Avoidance threshold.

B. Reordering-Induced Receiver Buffer Blocking

Using ordered delivery, the receiver side also has to take care of the message order and store out-of-order DATA chunks in the receive buffer for later reordering. This can lead to another kind of receive buffer blocking, which we denote as *Reordering-Induced Receiver Buffer Blocking*. Details can be found in our paper [10]. Since the focus of this paper is on unordered delivery, it is not further explained here.

C. GapAck-Induced Sender Buffer Blocking

At the send buffer, *GapAck-Induced Sender Buffer Blocking* is caused by TSNs having been GapAck'ed (i.e. renegably acknowledged) but not yet CumAck'ed. NR-SACKs [7] help to solve this problem, as shown by us in [10].

D. Transmission-Induced Sender Buffer Blocking

Transmission-Induced Sender Buffer Blocking denotes the unbalanced distribution of buffer space among the paths: when the sender transmits too many DATA chunks on a subset of the paths, too few buffer space may remain to utilise other paths. This situation may happen when one of the paths has to reduce its congestion window due to a loss while the other path may steadily increase its congestion window. We will demonstrate this problem in Section VI.

IV. Optimisations for Multi-Path Transfer over Dissimilar Paths

The key idea for countermeasures against the buffer blocking issues described in Section III is to prevent that problems or special characteristics on one path lead to insufficient buffer resources on well-working paths. Our solutions – Receiver and Sender Buffer Splitting – have been introduced in [10]. They split the corresponding buffer resources into fixed per-path sections, i.e. all paths get an equal buffer share. The transfer of this approach to a real system shows (to be explained in Subsection VI-B) that Buffer Splitting is a useful solution and works in combination with NR-SACKs as expected. But on a system without NR-SACK support, some optimisation is possible: a dynamic use of the buffer space would give a faster path the possibility to send more data by granting it to occupy more buffer space.

The goal is to keep a balance of the amount of outstanding bytes on all paths. This increases the throughput of the whole system. For this purpose, our approach in this paper is not based on the occupied buffer space any more. Instead, we just focus on the amount of outstanding data, because this value better represents the dynamic allocation of resources in the system. Like [10], our enhanced approach splits the send buffer of size B^{Sender} into n (i.e. number of paths) sections.

Let Outstanding_i be the number of outstanding bytes and MTU_i be the MTU on path P_i. Then, a new chunk on path P_i may be sent if the congestion window of path P_i allows its transmission *and* its buffer share allows another MTU-sized packet:

$$\text{Outstanding}_i + \text{MTU}_i \leq \frac{B^{\text{Sender}}}{n}$$

We denote this approach against Sender Buffer Blocking issues as *Sender Buffer Splitting based on Outstanding Bytes*.

Similar to the send buffer handling, the sender(!) also takes care of the receive buffer. It may send a new chunk on path P_i

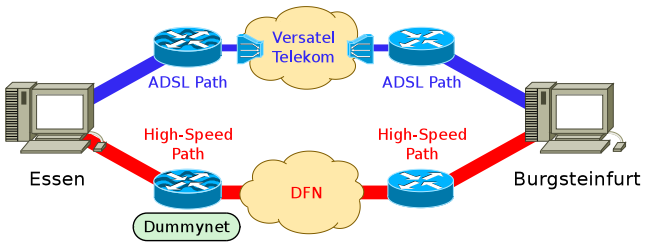

Figure 4. The Testbed and Simulation Setup

if the congestion window of path P_i allows its transmission *and* its assumed receive buffer share allows another MTU-sized packet:

$$\text{Outstanding}_i + MTU_i \leq \frac{\text{A\_RWND} + \sum_{j=1}^{n} \text{Outstanding}_j}{n}$$

We denote this approach against Receiver Buffer Blocking issues as *Receiver Buffer Splitting based on Outstanding Bytes*. Note, that this approach – being located at the sender side – also handles Window-Based Receiver Blocking issues, occurring for unordered transmission.

To clearly distinguish our new blocking countermeasures from the Buffer Splitting approaches in [10], the following sections will denote the mechanisms from [10] – which are based on buffered instead of outstanding bytes – as *Sender Buffer Splitting based on Buffered Bytes* and *Receiver Buffer Splitting based on Buffered Bytes*.

V. System Setup

For our performance evaluation of CMT-SCTP, we have used the OMNeT++-based INET framework with our CMT-SCTP simulation model [11]. The SimProcTC [12] toolchain has been used for parametrization and result processing. Our experimental validation in the real Internet has used the testbed setup depicted in Figure 4. Sender and receiver have run FreeBSD stable release 8.1 with the upcoming kernel 8.2 CMT-SCTP version (still under development). The sender located in Essen/Germany has transmitted its data over two paths to the receiver in Burgsteinfurt/Germany. NetPerfMeter [13] has been used for CMT-SCTP transmission and statistics recording. Path A has utilised a high-speed fibre optic connection over the German Research Network (Deutsches Forschungsnetz – DFN), with a typical RTT of 4 ms. The actual outgoing bandwidth can be varied using Dummynet [14] on the first router. Path B has used 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 has been 56 ms. Both paths have been known to be disjoint. This testbed setup has also been modelled in the simulation environment.

The following configuration parameters have been used for simulation and testbed setup, unless otherwise specified:

- The sender has been saturated (i.e. it has tried to transmit as much data as possible); the message size has been 1,444 bytes at an MTU of 1,492 bytes on the ADSL links (i.e. MTU-sized packets [4]). All messages have used unordered delivery.
- The send buffer has been set to 300,000 bytes, the receive buffer has been set to 100,000 bytes.
- In the simulation, FIFO queues of 100 packets (ADSL path) and 10 packets (high-speed path) have been configured on the first router.
- The runtime of each throughput test has been 60 s. Tests have been repeated multiple times (24 times for simulation, 8 times for measurement) in order to ensure a sufficient statistical accuracy. The results plots show the average values and their corresponding 95% confidence intervals.

VI. Evaluation

A. Simulation Results

For the evaluation of the two Buffer Splitting variants, we have performed simulations varying the bandwidth of the high-speed path β from 0.1 Mbit/s to 10 Mbit/s. The resulting CMT-SCTP payload throughputs are presented in Figure 5: Buffer Splitting based on Buffered Bytes is shown in the left-hand plot, Buffer Splitting based on Outstanding Bytes in the right-hand plot. Each plot displays the four cases: Buffer-Splitting on both sides (i.e. Sender and Receiver Buffer Splitting) turned on (i.e. Π=bothSides; curves 1 and 2 drawn in red colour on a colour plot) or off (i.e. Π=none; curves 3 and 4 drawn in blue colour on a colour plot) as well as NR-SACKs turned on (i.e. ρ=true; curves 1 and 3 drawn as solid lines) or off (i.e. ρ=false; curves 2 and 4 drawn as dotted lines).

Clearly, without Buffer Splitting, the sender is unable to utilise the high-speed path by having a sufficient number of outstanding bytes. The problem here is Window-Induced Receiver Buffer Blocking (see Subsection III-A), caused by the ADSL path: due to the long transmission queue of the ADSL modem, too many bytes are outstanding on the slow ADSL path – leaving no more room for increasing the number of outstanding bytes on the high-speed path (to be explained in detail below). The total number of outstanding bytes is limited by the advertised receiver window (100,000 bytes) – which is clearly much smaller than the send buffer (300,000 bytes). This size difference is also the reason why NR-SACK has no effect here: NR-SACK can help to reduce the send buffer space requirements (see Section II-B), but the send buffer is not fully occupied here.

Turning on Buffer Splitting significantly improves the payload throughput: in combination with NR-SACKs, both variants of Buffer Splitting (i.e. based on buffered bytes as well as based on outstanding bytes; see Section IV) reach the expected throughput of nearly 10.4 Mbit/s. Without NR-SACK, the throughput is significantly lower in more dissimilar path scenarios (here: $\beta \geq 4$ Mbit/s). Obviously, GapAck- and Transmission-Induced Sender Buffer Blocking (see Subsection III-C and Subsection III-D) occurs and prevents the high-speed path from utilising its bandwidth. As introduced in Section IV, this problem is clearly stronger when basing the Buffer Splitting on buffered instead of outstanding bytes (compare curve 2 in the right-hand plot to the corresponding one in the left-hand plot of Figure 5). That is, our improved variant of Buffer Splitting (based on outstanding bytes) achieves a significant performance improvement when NR-SACK cannot be applied, e.g. if the receiver side does not support this protocol extension [15].

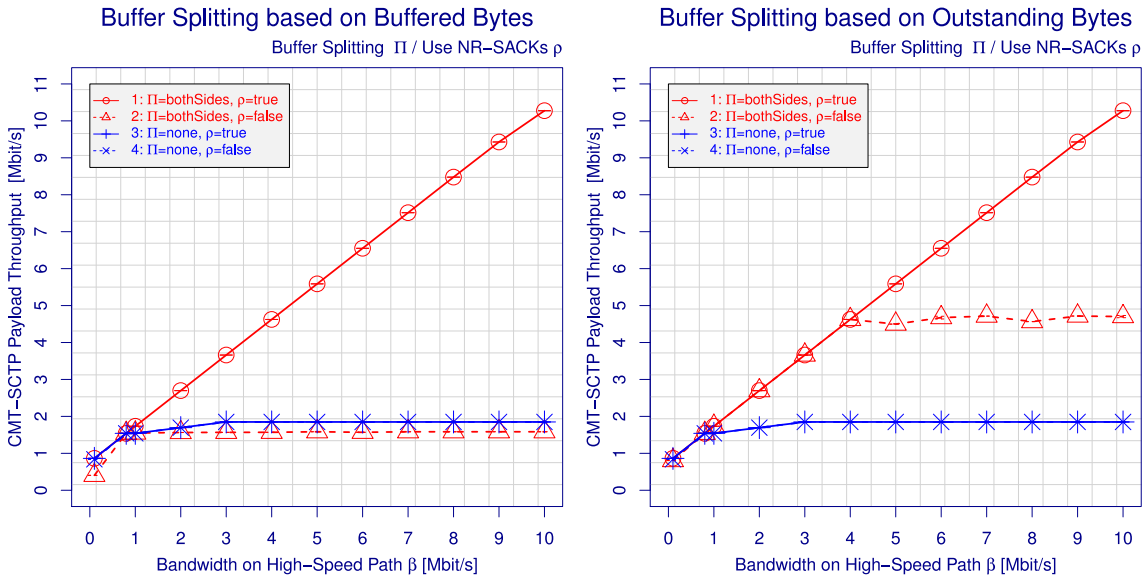

Figure 5. Simulation Results for Bandwidth Variation on the High-Speed Path

In order to further explain the effects causing the throughput results shown above, Figure 6 presents an extract from time t_0=1 s to t_1=2.5 s of the corresponding congestion window (solid lines) and slow start threshold (dotted lines) behaviour of ADSL path (blue colour) and high-speed path (red colour) for each of the four cases (i.e. Sender and Receiver Buffer Splitting based on Outstanding Bytes on/off; NR-SACKs on/off).

Plot 1 shows the results for Buffer Splitting turned off. Obviously, the congestion window of the ADSL path steadily increases to almost the size of the advertised receiver window (i.e. 100,000 bytes): as long as there is room in the advertised receiver window, the saturated sender tries to increase the congestion window of a path on reception of a new acknowledgement (see also [11] for details). DSL modems typically have a very long queue (here: 100 packets) – which is useful to avoid losses – but also leads to a linearly increased message delay on growing queue occupation. In our case, the rising number of outstanding bytes on the ADSL path fills this queue – which causes more delay but not an improvement of the throughput. Even worse, since the high-speed path is bandwidth-limited and using only a short queue (here: 10 packets), the congestion window of the ADSL path may take more buffer space when the high-speed path experiences a packet loss (which is caused regularly as part of the normal AIMD behaviour). In the end, the ADSL path – experiencing no loss due to its long queue – almost occupies all space of the send buffer. This implies no possibility for the congestion window of the high-speed path to grow large enough (i.e. at least up to the bandwidth-RTT product of the high-speed path) to fully utilise its capacity.

Turning on NR-SACKs (plot 2) has the effect that the congestion window growth becomes almost perfectly linear: without the need to wait for a cumulative acknowledgement, it can increase the number of outstanding bytes immediately on reception of an acknowledgement (i.e. Sender Buffer Blocking – see Section [10] is avoided). Once the congestion window of the ADSL path is large enough (here: $t \geq$1.8 s, it causes Window-Induced Receiver Buffer Blocking and the typical AIMD behaviour on the high-speed path (i.e. growing until a loss, then restarting from slow start threshold) is made impossible.

With Buffer Splitting turned on, but without NR-SACKs (shown in plot 3), it is clearly observable that the congestion window of the ADSL path can now only take a send buffer space of at most half of the advertised receiver window (i.e. 50,000 bytes) – which would in fact leave enough room for increasing the number of outstanding bytes on the high-speed path. However, while the throughput of this path is actually increased (see curve 2 on the right-hand plot of Figure 5), the congestion window curve of the high-speed path does not show the typical AIMD behaviour. Now, the problem is GapAck-Induced Sender Buffer Blocking: there is need to wait for TSNs on the long-delay (in particular due to the filling DSL modem queue) ADSL path to be acknowledged in order to cumulatively acknowledge a sequence of chunks. This is needed to actually gain space in the send buffer to transmit new chunks.

Also turning on NR-SACK (plot 4) solves this Sender Buffer Blocking problem: the feature of non-renegably acknowledging chunks allows the sender to remove selectively – but not yet cumulatively – acknowledged chunks from the send buffer and gain the space required to put more bytes in flight. This leads to fully utilising both paths and achieving the expected payload throughput (see curve 1 in Figure 5).

In summary, our simulations have shown that Buffer Splitting combined with NR-SACKs is necessary to fully utilise both paths of the high-speed path/ADSL path setup. If NR-SACK cannot be applied – e.g. if it is unsupported by the receiver side – Buffer Splitting must be based on outstanding bytes instead of buffered bytes to still achieve a performance improvement. In the following, we show the experimental validation of these results in our testbed setup – in order to transfer our conclusions from simulation to reality.

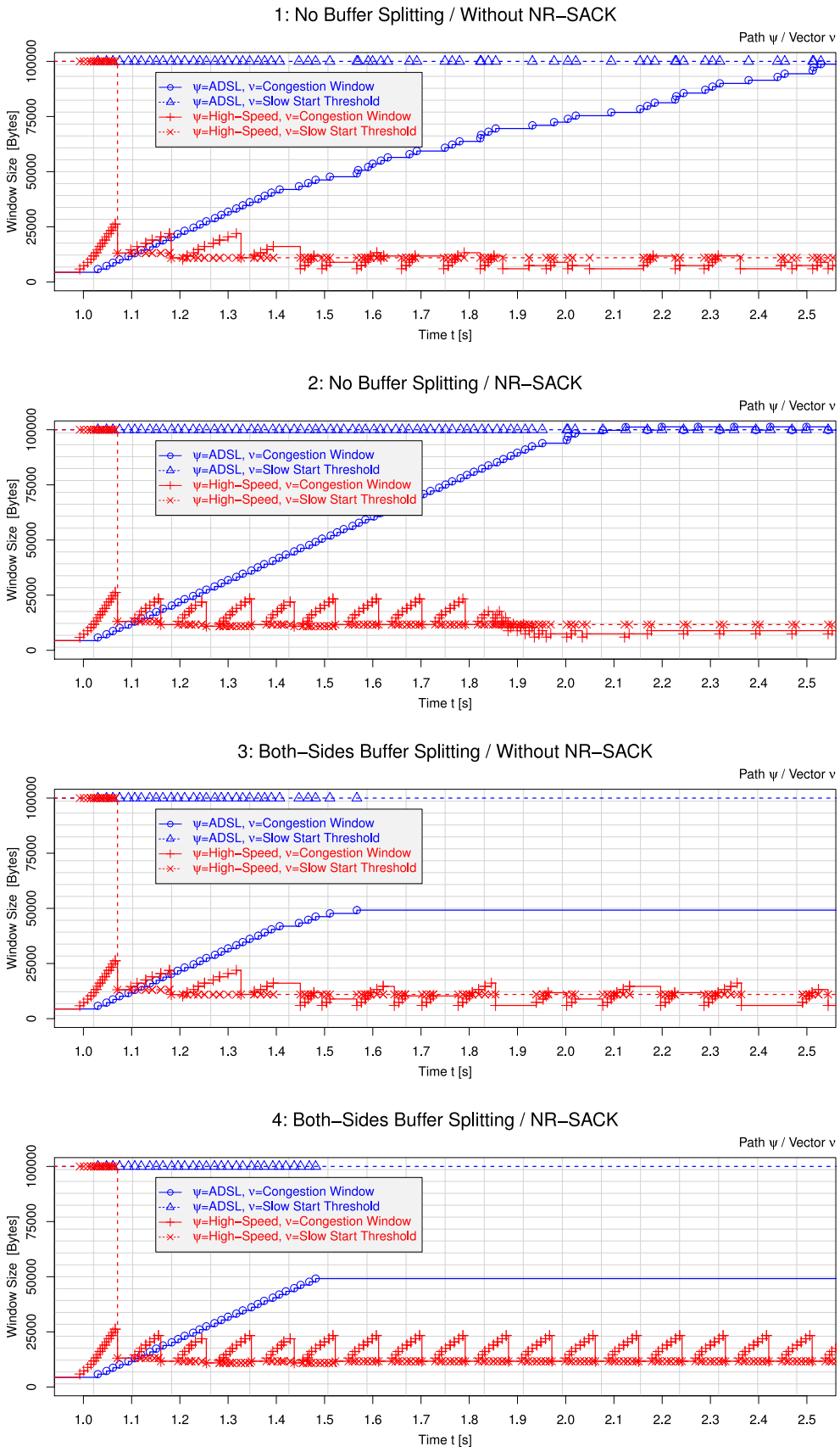

Figure 6. Congestion Window and Slow Start Behaviour Example

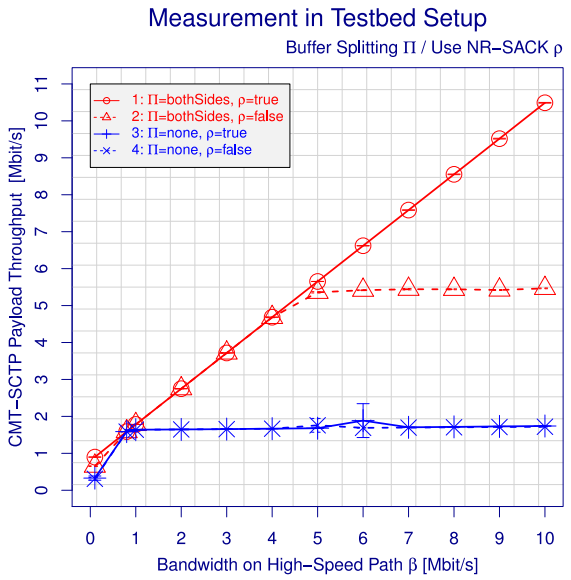

Figure 7. Experimental Validation in the Testbed Setup

B. Experimental Validation Results

For our experimental validation, we have varied the bandwidth of the high-speed link from 0.1 Mbit/s to 10 Mbit/s (i.e. as in the simulation scenario). Figure 7 shows the achieved CMT-SCTP payload throughput in the four configuration cases: Buffer-Splitting based on Outstanding bytes on both sides turned on (i.e. Π=bothSides; curves 1 and 2 drawn in red colour on a colour plot) or off (i.e. Π=none; curves 3 and 4 drawn in blue colour on a colour plot) as well as NR-SACKs turned on (i.e. ρ=true; curves 1 and 3 drawn as solid lines) or off (i.e. ρ=false; curves 2 and 4 drawn as dotted lines).

When comparing the measurement to the simulation results depicted on the right-hand plot of Figure 5, the reality matches the expectations from the simulation results quite well: the payload throughput linearly increases with the growing high-speed path bandwidth β if combining Buffer Splitting with NR-SACKs (curve 1). Just applying Buffer Splitting alone (i.e. without NR-SACKs; curve 2) still achieves a similar performance improvement, up to a certain dissimilarity of the paths (here: $\beta \geq 4$ Mbit/s). Without Buffer Splitting, only a quite constant payload throughput of less than 2 Mbit/s is achieved – regardless of turning NR-SACKs on (curve 3) or off (curve 4).

In summary, our measurements have shown that our extension Buffer Splitting based on Outstanding Bytes in combination with NR-SACKs is necessary to achieve the expected payload throughput in the high-speed path/ADSL path setup also under real Internet conditions. As expected, a significant throughput improvement is still achieved even without NR-SACKs. Our optimised Buffer Splitting extension will be included in the upcoming FreeBSD release 8.2.

VII. Conclusions

Using an ADSL connection as secondary Internet access link is an inexpensive option for providing Multi-Homing to an Internet site. However, using a multi-path Transport Layer protocol in such a setup – consisting of a high-speed primary access link and the secondary ADSL connection – imposes a significant challenge on the transport task. In this paper, we have presented an optimised approach to improve the send and receive buffer handling by preventing one path to dominate the buffer occupation. In simulations as well as in a real multi-homed Internet setup, we have furthermore proven the usefulness of our approach.

As part of future work, we are going to optimise the multi-path transport for messages using ordered delivery. Furthermore, we are also going to contribute our results into the ongoing IETF standardization process of SCTP [8], [16] and Multipath TCP.

References

[1] 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.

[2] F. Kelly and T. Voice, "Stability of End-to-End Algorithms for Joint Routing and Rate Control," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 2, pp. 5–12, Apr. 2005, ISSN 0146-4833.

[3] A. Ford, C. Raiciu, S. Barré, and J. Iyengar, "Architectural Guidelines for Multipath TCP Development," IETF, Network Working Group, Internet-Draft Version 02, Oct. 2010, draft-ietf-mptcp-architecture-02, work in progress.

[4] R. Stewart, "Stream Control Transmission Protocol," IETF, Standards Track RFC 4960, Sept. 2007.

[5] 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.

[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] P. Natarajan, N. Ekiz, E. Yilmaz, P. D. Amer, and J. 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.

[8] M. Becke, T. Dreibholz, J. Iyengar, P. Natarajan, and M. Tüxen, "Load Sharing for the Stream Control Transmission Protocol (SCTP)," IETF, Network Working Group, Internet-Draft Version 00, July 2010, draft-tuexen-tsvwg-sctp-multipath-00, work in progress.

[9] T. Dreibholz, R. Seggelmann, M. Tüxen, and E. P. Rathgeb, "Transmission Scheduling Optimizations for Concurrent Multipath Transfer," in *Proceedings of the 8th International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT)*, Lancaster, Pennsylvania/U.S.A., Nov. 2010.

[10] 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.

[11] 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++*, Málaga/Spain, Mar. 2010, ISBN 978-963-9799-87-5.

[12] 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.

[13] T. Dreibholz, *NetPerfMeter Homepage*, 2010.

[14] T. Dreibholz, M. Becke, 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.

[15] P. Natarajan, P. Amer, E. Yilmaz, R. Stewart, and J. Iyengar, "Non-Renegable Selective Acknowledgements (NR-SACKs) for SCTP," IETF, Network Working Group, Internet-Draft Version 06, Aug. 2010, draft-natarajan-tsvwg-sctp-nrsack-06, work in progress.

[16] T. Dreibholz and M. Becke, "SCTP Socket API Extensions for Concurrent Multipath Transfer," IETF, Network Working Group, Internet-Draft Version 00, Nov. 2010, draft-dreibholz-tsvwg-sctpsocket-multipath-00, work in progress.