Non-Renegable Selective Acknowledgments (NR-SACKs) for MPTCP

Fan Yang
Computer and Information Sciences Department
University of Delaware
Newark, Delaware, USA 19716
yangfan@udel.edu

Paul Amer
Computer and Information Sciences Department
University of Delaware
Newark, Delaware, USA 19716
amer@cis.udel.edu

***Abstract*—We introduce Non-Renegable Selective Acknowledgments (NR-SACKs) to MPTCP, and investigate their impact in situations where an MPTCP receiver never discards received out-of-order data from the MPTCP receive buffer (i.e., never renegs). NR-SACKs not only allow an MPTCP receiver to report the reception of out-of-order data, but also allow an MPTCP sender to free reported out-of-order data in the MPTCP send buffer sooner than the advance of the MPTCP level cumulative acknowledgement (DATA ACK). We implemented NR-SACKs in the Linux kernel. Experiments show that (i) the MPTCP data transfers with NR-SACKs never perform worse than those without NR-SACKs, and (ii) NR-SACKs improve throughput in MPTCP when the total congestion window (cwnd) of all subflows is greater than the MPTCP send buffer size (i.e., the send buffer size is the bottleneck).**

***Keywords*-multipath; NR-SACKs; reneging; selective acknowledgment**

I. Introduction

Multipath reliable data transfer has received a lot of recent attention as seen by extensions to TCP and SCTP to support multihoming. However, extensions to TCP [12], [13], [14] have never been implemented nor deployed [9]. SCTP's extension, called Concurrent Multipath Transmission (CMT) [2], is implemented but not widely deployed since many Internet middle-boxes by default block SCTP-PDUs.

In both TCP and SCTP, a receiver informs a sender about the reception of out-of-order data through SACKs, but the receiver is permitted to later discard the SACKed data [7]. Discarding data that has been previously SACKed is known as *reneging*. Because of the possibility of reneging, a sender needs to keep SACKed data in the send buffer until they are cumulatively acknowledged (cum-acked). Both TCP and SCTP are designed to tolerate reneging. This design has been challenged [5] since (i) reneging rarely occurs in practice, and (ii) even when it does occur, reneging alone generally does not help the operating system resume normal operation when the system is starving for memory. Non-Renegable Selective Acknowledgments (NR-SACKs) [3] allow a receiver to convey non-renegable information of received out-of-order data back to the corresponding sender. NR-SACKs allow that sender to remove NR-SACKed data from the send buffer sooner than the arrival of corresponding cum-acks. NR-SACKs have been introduced to both SCTP and SCTP with CMT, and results show that NR-SACKs not only reduce sender's memory requirements, but also improve the end-to-end throughput under certain conditions [4], [6].

The IETF has created a multipath working group to specify a standard for Multipath TCP (MPTCP). MPTCP provides an ability to simultaneously transmit data over multiple TCP paths between peers [1].

In this work, we introduce NR-SACKs to MPTCP and investigate their impact in situations where an MPTCP receiver never discards out-of-order data in the MPTCP receive buffer (i.e., never renegs). This paper is organized as follows. Section II explains data sequencing in MPTCP, and describes a problem, called GapAck-Induced send buffer blocking [10], in MPTCP data transfers when the total cwnd size of all subflows are greater than the MPTCP send buffer size. Section III introduces our proposed modified MP_CAPABLE and DSS option for MPTCP to support NR-SACKs, and briefly explains sender and receiver side NR-SACK processing details. Section IV elaborates our testbed topology and experimental parameters (e.g., delay, loss rate and send buffer size). Section V analyzes the results for MPTCP data transfers with NR-SACKs vs. without NR-SACKs. Section VI concludes our work.

II. GapAck-Induced Send Buffer Blocking in MPTCP Unordered Data Transfer

In MPTCP, each subflow is a standard TCP connection with its own sequence number space. An MPTCP level sequence number space, based on a Data Sequence Number (DSN), additionally numbers bytes at the MPTCP level. A single MPTCP send buffer and a single MPTCP receive buffer are shared among all subflows, while each subflow has its own receive buffer to hold subflow level out-of-order data (since each subflow TCP receiver must deliver subflow level data in-order to the MPTCP receive buffer).

When an application writes a stream of bytes to an MPTCP send buffer, MPTCP numbers each byte with a DSN. When bytes are then sent on a particular subflow, they are encapsulated into TCP-PDUs with MPTCP information placed in the TCP option field. When a TCP-PDU is received in-order at subflow level, the payload is delivered to the

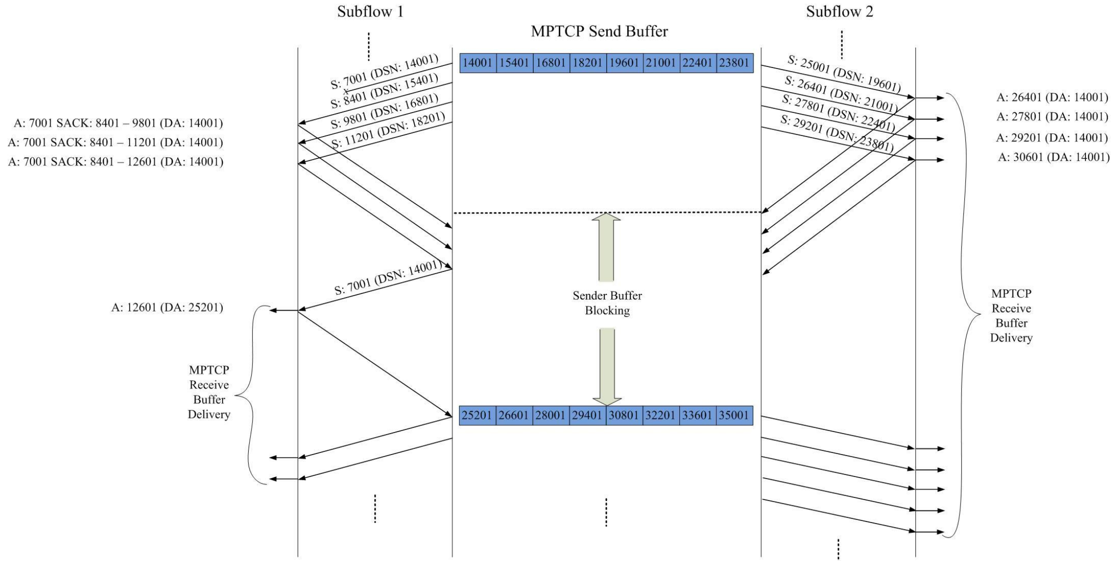

Figure 1. Timeline of an Unordered MPTCP Data Transfer

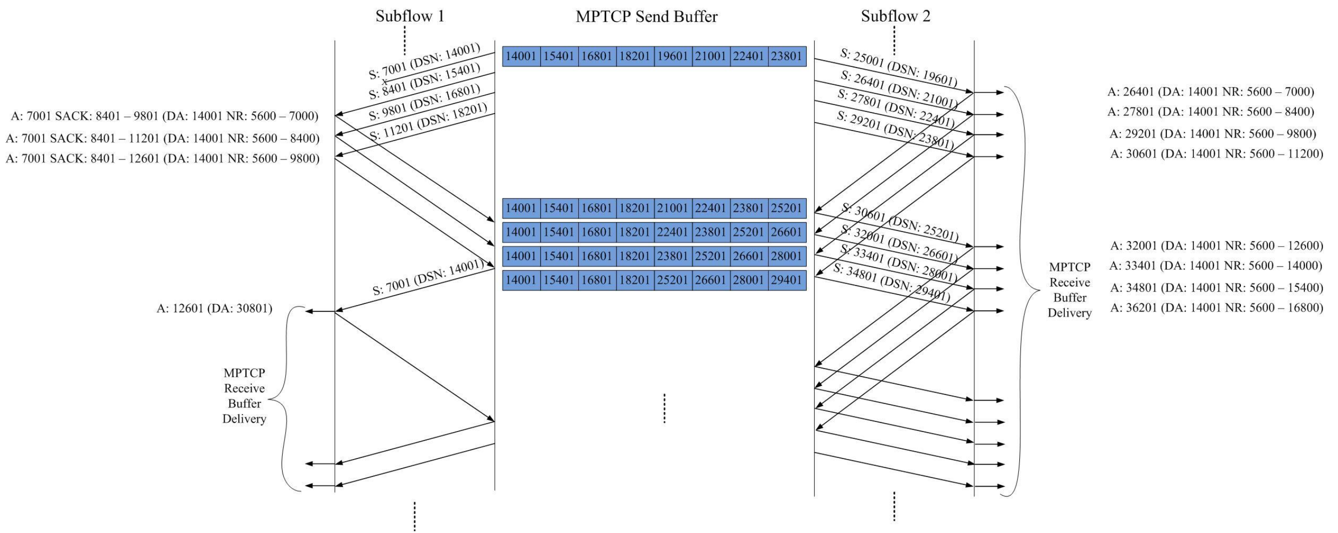

Figure 2. Timeline of an Unordered MPTCP Data Transfer with NR-SACKs

MPTCP receive buffer immediately. The MPTCP level cum-ack number, called DATA ACK, advances if the delivered data are also in-order at the MPTCP level.

The subflow receiver cum-acks those delivered data by a regular TCP cum-ack, and places the current DATA ACK in the TCP option field. An application consumes in-order data from the MPTCP receive buffer. Currently, an MPTCP sender only frees data from the MPTCP send buffer when they have been cum-acked by DATA ACK received on any subflow [1].

Data often arrive out-of-order at an MPTCP receive buffer because of loss and/or asymmetric RTTs of the subflows. In a heterogeneous network when different subflows have different characteristics (i.e., loss and RTT), the amount of out-of-order data arriving at the MPTCP receiver side can be significant.

In current MPTCP standard, an MPTCP receiver cannot report the reception of out-of-order data to an MPTCP sender. If an MPTCP receiver never renegs (as is the case in Linux kernel MPTCP implementation [8]), an MPTCP sender does not need to maintain the received out-of-order data in the send buffer.

Consider a scenario where an MPTCP receiver never renegs. In Figure 1, two subflows have been established. After some initial period of data transfer (not shown), assume both subflows have reached their congestion avoidance

phase, and they have roughly the same RTT and the same MSS of 1400 bytes. The MPTCP send buffer, denoted by the shaded rectangular box, can hold up to 11200 bytes of application data. The entire send buffer is equally divided into 8 pieces (each 1400 bytes) and each piece is denoted by its starting Data Sequence Number (DSN) inside a small rectangular box.

The timeline slice shown in Figure 1 starts at a point in the data transfer when both subflows have cwnd $= 4$. When bytes are then to be transmitted on a subflow, they are encapsulated into TCP-PDUs which are denoted by both the respective subflow's TCP sequence number (S) and the DSN of the first byte of the payload. Each ack contains not only a subflow cum-ack number (A) and SACKs (if any), but also a DATA ACK (DA). TCP-PDU S: 7001 (DSN: 14001) of subflow 1 is assumed lost.

Upon reception of the first ack (A: 26401 (DA: 14001)) on subflow 2, the MPTCP sender could in theory continue to transmit new data on subflow 2, since subflow 2 has available cwnd (i.e. $cwnd - num_{packet_in_flight} > 0$). However, the MPTCP send buffer does not have any new data. Actually, before the ack of the retransmission of TCP-PDU S: 7001 (DSN: 14001) arrives at the MPTCP sender, even though data corresponding to DSNs 19601 - 25200 have been successfully received by the MPTCP receiver, the MPTCP sender cannot free these data from the send buffer since the DATA ACK does not advance. This scenario illustrates *GapAck-Induced send buffer blocking* (hereafter called send buffer blocking). Send buffer blocking occurs when the total cwnds of all subflows are greater than the MPTCP send buffer size. Send buffer blocking prevents the MPTCP sender from fully utilizing the cwnds of subflows.

In the case where an MPTCP receiver never renegs, this simple timeline illustrates the following:

- After bytes have been received out-of-order by an MPTCP receiver, maintaining these data in the MPTCP send buffer is *unnecessary*, i.e., a waste of memory.
- When send buffer blocking occurs, the MPTCP send buffer size becomes a bottleneck of throughput.

III. Non-Renegable Selective Acknowledgments (NR-SACKs) for MPTCP

We propose using NR-SACKs to enable an MPTCP receiver to inform an MPTCP sender about the reception and "non-renegability" of out-of-order data.

A. Modified Multipath Capable (MP_CAPABLE) and Data Sequence Signal (DSS) Options supporting NR-SACKs

Because of page limit, the details of the proposed modified MP_CAPABLE and DSS options which support NR-SACKs can be found on our lab website at http://www.eecis.udel.edu/~amer/PEL/.

B. MPTCP Unordered Data Transfer with NR-SACKs

Figure 2 is analogous to Figure 1's example, this time using NR-SACKs. The MPTCP sender and receiver are assumed to have negotiated using NR-SACKs. As in Figure 1, TCP-PDU S: 7001(DSN: 14001) of subflow 1 is presumed lost. Notice that, the first three acks on subflow 1 and the first four acks on subflow 2 carry NR-SACK information. When the first ack on subflow 2 arrives, the MPTCP sender is informed that data corresponding to DSNs from $(14001 + 5600)$ to $(14001 + 7000 - 1)$ have been received and are non-renegable. The MPTCP sender immediately frees these NR-SACKed data from the MPTCP send buffer, allowing the application to write new data to the MPTCP send buffer. This new data is transmitted on subflow 2 which has available cwnd. Then the first ack on subflow 1 arrives, but its NR-SACK information is same as the first ack of subflow 2. On the reception of the second, third and fourth acks, more new data are transmitted on subflow 2.

Figure 2 illustrates the following observations on MPTCP data transfers with NR-SACKs:

- The MPTCP send buffer only contains necessary data, thus, NR-SACKs allow a more efficient MPTCP send buffer usage.
- Although subflow 1 is blocked due to the loss, new application data can still be transmitted on subflow 2. NR-SACKs alleviate send buffer blocking hence higher throughput is achieved than the scenario in Figure 1.

IV. Experimental Setup

We extended the Linux kernel MPTCP implementation [8] to support and process NR-SACKs at the data receiver and data sender, respectively. The experiment evaluates the performance of MPTCP data transfers (with two subflows) with NR-SACKs vs. without NR-SACKs under various conditions (path loss rate, delay and send buffer size). The coupled congestion control option is disabled in this evaluation since we want to focus on the impact of NR-SACKs.

A. Test-bed Topology

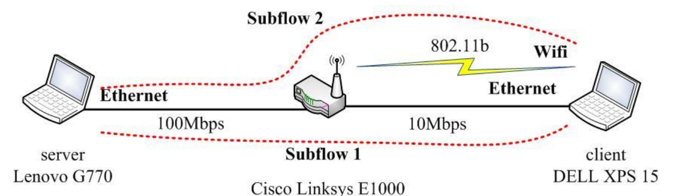

Figure 3. Test-bed Topology

The test-bed (Figure 3) is composed of a Cisco Linksys E1000 router and two laptops running Ubuntu 11.10. A server is connected to the router with a tethered 100Mbps Ethernet cable. A multihomed client is connected to the

router by both an Ethernet cable and a wireless link. To prevent the link between the server and the router being a bottleneck, the Ethernet cable connecting the client and the router has a 10Mbps capacity, and 802.11b (maximum raw data rate is 11Mbps) is used for the wireless link. An MPTCP connection with two subflows is created. Subflow 1 is a TCP connection established over the wired-wired path, and subflow 2 is a TCP connection established over the wired-wireless path. The traffic is generated by moving a 1.46GB file from server to client with MPTCP.

B. Experimental Parameters

Before discussing about the experimental parameters, consider a problem of the current scheduler of the Linux MPTCP. When bytes are ready to send, a scheduler selects which subflow to send these data. The current scheduler selects the subflow with the shortest sample RTT ($srtt$) among all subflows with available cwnd. In our test-bed, if neither loss nor delay is introduced, the $srtt$ of subflow 1 is always shorter than that of subflow 2. So subflow 1 is selected to send data whenever it has available cwnd, while subflow 2 is only selected when subflow 1 has no available cwnd. Figure 4 shows the load sharing of subflows 1 and 2 during the first 100s of data transfer. We can see subflow 2 only carries a small portion of the load. This problem has been reported in [15]. If only one subflow is used during most of the data transfer, the receiver will seldom generate NR-SACKs. To force the using of both subflows during the data transfer, loss is introduced to both subflows. When loss occurs on subflow 1, its cwnd reduces. At times, subflow 1 will have no available cwnd, and the scheduler will select subflow 2. The current scheduler is obviously suboptimal, our future work includes implementation and evaluation of different scheduling policies for MPTCP [11].

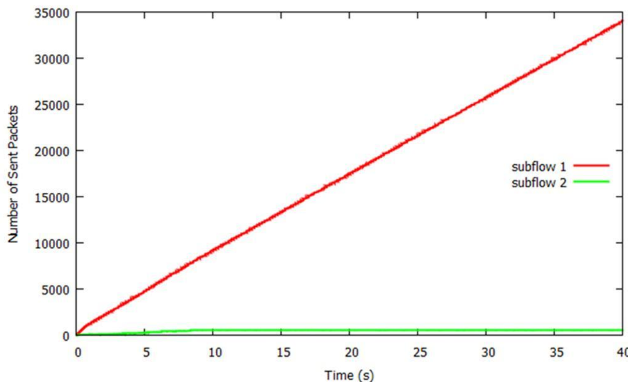

Figure 4. Load Sharing of Two Subflows (no extra loss or delay)

In our experiments, four different delays {5ms, 10ms, 50ms, 500ms} and three different loss rates {0.5%, 1%, 5%} are configured on the outgoing direction of the server's Ethernet interface by using the Linux traffic control [16].

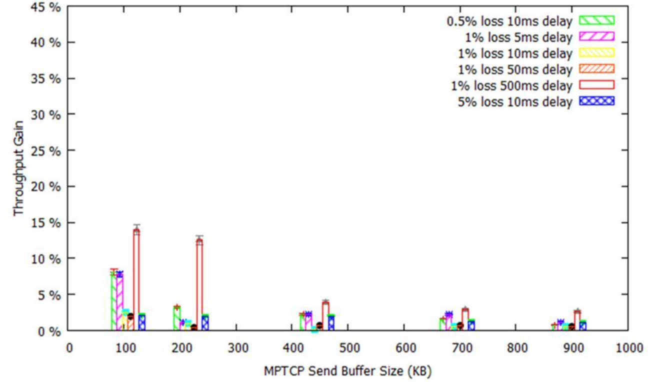

Figure 5. Throughput Gain with NR-SACKs (899KB, 700K, 449KB, 224KB, 112KB send buffer sizes)

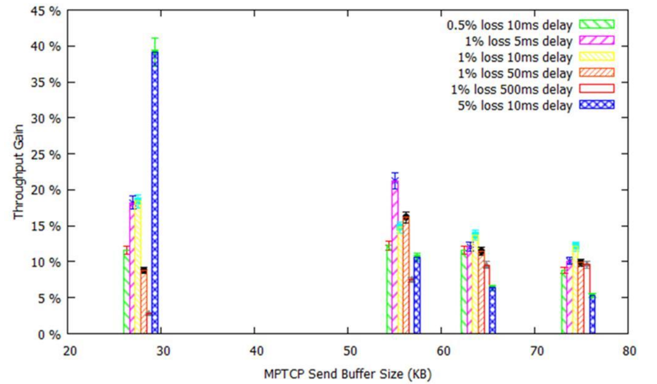

Figure 6. Throughput Gain with NR-SACKs (74KB, 64KB, 56KB, 28KB send buffer sizes)

The performance of NR-SACKs has been tested for Linux MPTCP send buffers ranging in size from 14KB to 899KB.

V. Results

To evaluate the performance of MPTCP data transfers with NR-SACKs vs. without NR-SACKs precisely, we employ the metric *throughput gain* defined in [6] as $(T_{NR-SACK} - T)/T$ where $T_{NR-SACK}$ is the throughput achieved with NR-SACKs and T is the throughput achieved without NR-SACKs for an identical set of experimental parameters (send buffer size, loss rate, bandwidth, and delay). We also use a *region of gain* [6] defined as the send buffer size interval, [a, b], where any send buffer size between a and b results in an expected throughput gain of at least 5%.

NR-SACKs require minimal additional processing time at both end hosts and only a few extra bytes on the wire. Thus, our first hypothesis was that these overheads would be negligible, and that MPTCP data transfers with NR-SACKs would always perform at least as well as those without NR-SACKs. For clarity, Figures 5 and 6 only show the throughput gain of part of the parameter combinations tested.

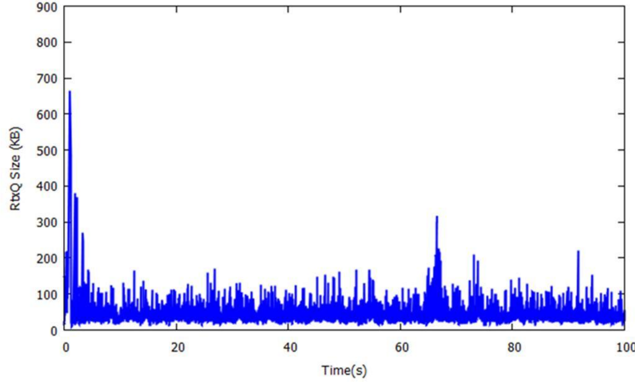

Figure 7. Retransmission Queue Evolution without NR-SACKs (899KB send buffer size, 1% loss, 10ms delay)

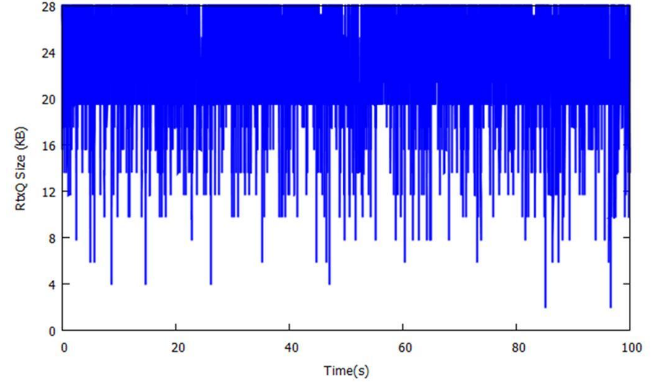

Figure 8. Retransmission Queue Evolution without NR-SACKs (28KB send buffer size, 1% loss, 10ms delay)

Our first hypothesis is confirmed by both figures.

Importantly, as the MPTCP send buffer size decreases, we observe increasing throughput gain with NR-SACKs from Figures 5 and 6. Based on the previous discussion, NR-SACKs can free received out-of-order data from the send buffer prior sooner than the arrival of the corresponding cum-ack. When send buffer blocking occurs, the total cwnds of all subflows, and hence RtxQ, grow large enough to fill the entire send buffer. NR-SACKs allow more new application data be transmitted. Therefore, our second hypothesis was that when send buffer blocking occurs, MPTCP data transfers with NR-SACKs would outperform those without.

A. Retransmission queue evolution

To confirm our second hypothesis and gain insight into the send buffer blocking, consider how the Retransmission Queue (RtxQ) size varies over time. Figures 7 and 8 show how the RtxQ size varies for send buffer sizes 899KB and 28KB, respectively. In both figures, the loss rate and delay on the outgoing direction of the server's interface are 1% and 10ms, respectively. In Figure 7, the RtxQ size never reaches 899KB, thus no send buffer blocking occurs and no significant throughput gain is expected by using NR-SACKs (as confirmed in Figure 5). In Figure 8, the RtxQ size frequently reaches 28KB each time causing send buffer blocking. When send buffer blocking occurs, significant throughput gain is expected by using NR-SACKs (as confirmed in Figure 6). These results confirm our second hypothesis.

B. Impact of Loss Rate

For a given bandwidth-delay combination, higher loss rates result in smaller total cwnds (and hence smaller RtxQ size). When send buffer blocking occurs, higher loss rates result in more serious blocking than smaller loss rates. Additionally, higher loss rates make an MPTCP receiver generate more NR-SACK information. Therefore, we hypothesized that (i) the right edge of the region of gain would be smaller for a higher loss rate than for a smaller loss rate, and (ii) in the region of gain, the maximum throughput gain of a higher loss rate would be greater than that of a smaller loss rate.

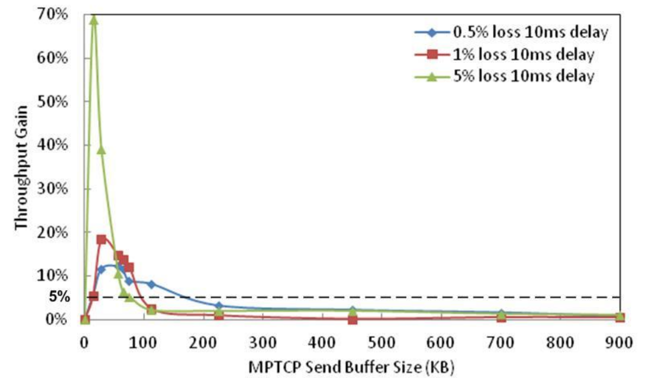

Figure 9. Throughput Gain with NR-SACKs (same delay different loss rates)

These hypotheses are confirmed by Figure 9. As the loss rate decreases, the right edge of region of gain moves to the left and the maximum throughput gain in the region of gain moves up. When the loss rate is 5%, the throughput gain can reach as high as 70% under a 14KB MPTCP send buffer.

C. Impact of Delay

For a given bandwidth, longer delays result in greater Bandwidth-Delay Product (BDP). When the BDP $<$ MPTCP send buffer size, no send buffer blocking occurs since the total cwnd size is bounded by the BDP. Send buffer blocking occurs only when BDP $\geq$ MPTCP send buffer size. Therefore, we hypothesized that the right edge of the region of gain would be bigger for a longer delay than for a shorter delay.

Our hypothesis is confirmed by Figure 10. We can see, as the delay increases, the right edge of region of gain moves

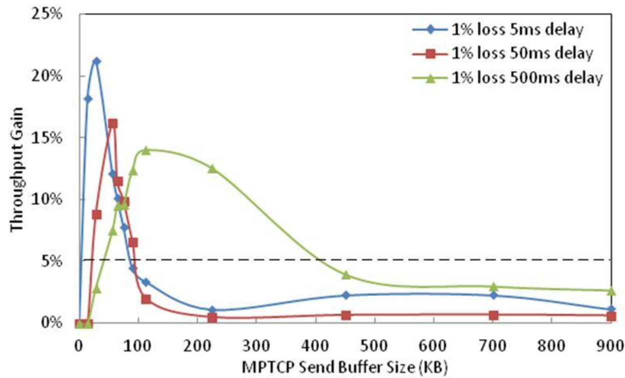

Figure 10. Throughput Gain with NR-SACKs (same loss rate different delay)

right. For all loss rates tested, we also observed that the throughput gain with NR-SACKs is greater over a link with a shorter delay (consistent with the results in [6]).

VI. CONCLUSIONS AND FUTURE WORK

In this work, we introduced NR-SACKs to MPTCP and investigated their impact in situations where an MPTCP receiver never renegs. We extended the Linux MPTCP implementation to support NR-SACKs. Based on the experiment, we concluded that (i) MPTCP data transfers with NR-SACKs never perform worse than those without NR-SACKs, and (ii) NR-SACKs can improve end-to-end throughput in MPTCP when send buffer blocking occurs. In an MPTCP connection with several high-BDP subflows, send buffer blocking can occur and seriously decrease the end-to-end throughput. NR-SACKs can alleviate the send buffer blocking and achieve higher throughput. Based on the argument that the design to tolerate reneging is wrong, we recommend that NR-SACKs SHOULD be added to the MPTCP standard.

We are currently implementing and evaluating different scheduling policies for MPTCP, since the current Linux MPTCP scheduler implementation is suboptimal and no standard for scheduling policy is published.

ACKNOWLEDGMENT

The authors would like to thank Nasif Ekiz, Jonathan Leighton and the anonymous reviewers for their valuable comments and suggestions.

REFERENCES

[1] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, *TCP Extensions for Multipath Operation with Multiple Addresses*, draft-ietf-mptcp-multiaddressed-09. IETF Internet draft, June 2012.

[2] J. Iyengar, P. Amer, and R. Stewart, *Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-end Paths*. IEEE/ACM Trans on Networking, 14(5), October 2006.

[3] P. Amer, M. Becke, T. Dreibholz, N. Ekiz, J. Iyengar, P. Natarajan, R. Stewart, and M. Tüxen, *Load Sharing for the Stream Control Transmission Protocol (SCTP)*, draft-tuexen-tsvwg-sctp-multipath-05. IETF Internet draft, September 2012.

[4] P. Natarajan, N. Ekiz, E. Yilmaz, P. Amer, J. Iyengar, and R. Stewart, *Non-Renegable Selective Acks (NR-SACKs) for SCTP*. IEEE International Conference on Network Protocols, Orlando, Florida, USA, October 2008.

[5] N. Ekiz, *Transport Layer Reneging*. PhD Dissertation, CIS Department, University of Delaware, May 2012.

[6] E. Yilmaz, N. Ekiz, P. Amer, J. Leighton, F. Baker, and R. Stewart, *Throughput Analysis of Non-Renegable Selective Acknowledgments (NR-SACKs) for SCTP*. Computer Communications, 33(16):1982–1991, October 2010.

[7] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, *TCP Selective Acknowledgment Options*. RFC 2018, October 1996.

[8] *MultiPath TCP - Linux Kernel Implementation*. http://mptcp.info.ucl.ac.be/.

[9] S. Barré, C. Paasch, and O. Bonaventure, *MPTCP: From Theory to Practice*. Networking 2011, Valencia, Spain, May 2011.

[10] H. Adhari, T. Dreibholz, M. Becke, E.P. Rathgeb and M. Tüxen, *Evaluation of Concurrent Multipath Transfer over Dissimilar Paths*. 1st International Workshop on Protocols and Applications with Multi-Homing Support, Singapore, 2011.

[11] F. Yang and P. Amer, *Scheduling Policies for MPTCP* (in preparation).

[12] H. Hsieh and R. Sivakumar, *pTCP: An End-to-end Transport Layer Protocol for Striped Connections*. IEEE International Conference on Network Protocols, Paris, France, November 2002.

[13] K. Rojviboonchai, T. Osuga, and H. Aida, *R-M/TCP: Protocol for Reliable Multipath Transport Over the Internet*. AINA 2005, Taiwan, 2005.

[14] M. Zhang, J. Lai, and A. Krishnamurthy, *A Transport Layer Approach for Improving End-to-end Performance and Robustness Using Redundant Paths*. 2004 USENIX Annual Technical Conference, Boston, MA, USA.

[15] S. Nguyen, X. Zhang, T. Nguyen and G. Pujolle, *Evaluation of Throughput Optimization and Load Sharing of Multipath TCP in Heterogeneous Networks*. WOCN 2011, New Orleans, Louisiana, 2011.

[16] *Linux Advanced Routing and Traffic Control*. http://www.lartc.org/.