A Scheduler for Multipath TCP

Fan Yang
CIS Department
University of Delaware
Newark, Delaware, USA 19716
yangfan@udel.edu

Paul Amer
CIS Department
University of Delaware
Newark, Delaware, USA 19716
amer@udel.edu

Nasif Ekiz
F5 Networks
Seattle, Washington, USA 98119
n.ekiz@f5.com

***Abstract*—We first explain problems with the default scheduler used by the Linux kernel MPTCP implementation. Then we propose the design of a new scheduler. Preliminary empirical results show that our proposed scheduler improves the throughput in MPTCP by alleviating the problems caused by the default scheduler.**

***Keywords*-multipath; scheduling; congestion control**

I. Introduction

The IETF has created a working group to specify a standard for Multipath TCP (MPTCP). MPTCP, perhaps the most significant change to TCP in the past 20 years [10], simultaneously uses multiple TCP paths between peers [1]. MPTCP not only increases robustness during time of path failure, but also potentially achieves higher end-to-end throughput.

Currently, a Linux kernel MPTCP implementation [15] is available for real world experiments. Several publications have pointed out problems with this implementation. For example, [6] argues that scheduling of traffic on the subflows is unequal when the subflows experience similar network characteristics, i.e., delay, packet loss and bandwidth. [7] points out that in heterogeneous networks, the throughput of MPTCP using multiple paths can even be worse than that of one TCP connection using only the best path. We believe these publications do not focus on the major reason for these problems. Having found a problem with this implementation during our experiments with MPTCP NR-SACKs [12], we believe these problems lie with the default scheduler.

Consider the task of a scheduler. Whenever an MPTCP sender wants to send data, the sender needs to make three decisions. First, which subflow(s) can be used to send data? This decision is made by the MPTCP congestion control mechanism which maintains a per-subflow cwnd. The subflows with available cwnd can be used to send data. Second, if several subflows have available cwnd, a scheduler selects on which to send data. Third, after selecting a subflow, the scheduler determines how much data should be sent on it. The third decision concerns the granularity of the allocation. In this paper, we assume each subflow has the same maximum segment size (MSS), and an MPTCP sender allocates one MSS at a time in step 3. Future work is needed to support other allocation granularities. Here we focus on the scheduler's second decision.

Obviously, the full benefit of MPTCP deployment is achieved by the sender which coordinates the scheduler and the congestion control mechanism. The MPTCP congestion control mechanism has been designed [3] and standardized [2], while the scheduler is still under study. [5] proposes a Self-adapted Rescheduling Reliable Multipath Transfer Protocol which can adaptively reschedule data to different paths according to the their current average bandwidth. In [4], the performances of several scheduling algorithms are compared, showing that a scheduler minimizing the packet delivery delay yields the best overall performance. Actually, both the scheduler proposed in [5] and the conclusion in [4] agree that the 'fastest' (i.e., largest bandwidth or lowest packet delivery delay) subflow is chosen to send data at any given time. This is the default scheduler used by the Linux MPTCP implementation. However, based on our research, the default scheduler can not only be inconsistent with the targets of the congestion control mechanism but can also use network resources inefficiently.

This paper is organized as follows. Section II describes the default scheduler, its problems, as well as our thinking to derive a better scheduler. Section III describes the algorithms of our proposed scheduler. Section IV elaborates our test-bed topology. Section V compares the throughput and load sharing achieved by the default and our proposed schedulers under different scenarios. Section VI concludes our work.

II. What is a Good Scheduler?

The Linux kernel MPTCP implementation scheduler is:

- when multiple subflows can be used to send data, data is transmitted on the subflow with the shortest smoothed round trip time (srtt).

This default scheduler seems reasonable. Srtt reflects the time between when a packet is sent out and when its corresponding acknowledgement comes back. The default scheduler selects the 'fastest' subflow to send next packet.

A. Problems

Since a scheduler works in cooperation with the congestion control mechanism, they need to be consistent.

One important target of the MPTCP congestion control mechanism is moving data away from congested path(s) [3]. However, if multiple subflows have available cwnd, this target cannot be achieved without a good scheduler.

Consider a scenario of an MPTCP connection with two subflows as shown in Figure 1. Subflow 1 is established on a 3G path with a large buffer (can queue up to 200 packets), resulting in possible longer delays but lower drop rates. Subflow 2 is established on a Wifi path which has a smaller buffer (can queue up to 20 packets), resulting in possible shorter delays but higher drop rates. At the shown time, assume the buffer of the Wifi path is full, while that of the 3G path is only half full. The Wifi path (subflow 2) is congested, i.e., packet drops are more likely. However, if both subflows have available cwnd, the default scheduler will choose subflow 2 to send the next packet. In this scenario, a subflow's srtt does not reflect its congestion. As demonstrated by this hypothetical scenario, a scheduler only based on srtt may be inconsistent with the target of the MPTCP congestion control mechanism.

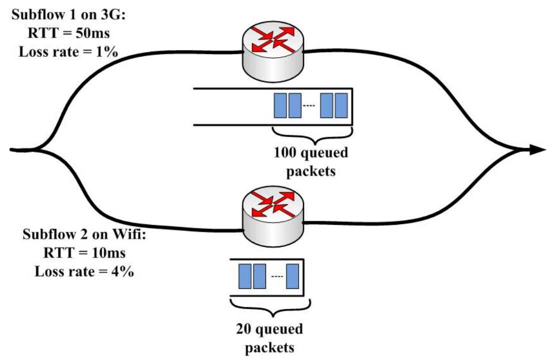

Figure 1. A scenario in which RTT and congestion mismatch

In above scenario, sending more packets on subflow 2 may cause loss due to congestion. More seriously, after a loss is detected, lost packets need to be retransmitted and subflow 2's cwnd will decrease. It takes time for subflow 2 to increase its cwnd to the value before the loss. This situation causes an inefficient use of network resources. Unlike a TCP sender, the MPTCP sender has choices. In the above scenario, the MPTCP sender could have sent the packet on subflow 1 rather than subflow 2.

B. Analysis

MPTCP employs an additive-increase multiplicative-decrease (AIMD) coupled congestion control mechanism. Each subflow continually increases its cwnd even to a point that exceeds the available path capacity (defined as the maximum number of packets in flight of this subflow) before detecting a loss. If the number of outstanding packets of a subflow has reached its available path capacity, sending more packets through the subflow will cause congestion loss.

To both be consistent with the target of the congestion control mechanism and use network resources efficiently, a good scheduler needs to select a 'fastest' subflow without causing loss because of congestion. In other words, **a good scheduler needs to select a subflow based on not only its srtt but also its congestion situation.**

A scheduler needs to estimate the congestion situation of each subflow. At any given time, a scheduler knows the number of outstanding packets on a subflow. If the available path capacity of each subflow could be accurately estimated, the scheduler could tell whether sending more packets on a subflow will make it reach and/or surpass its available path capacity. Consider a per-subflow ratio $Occupied = (Number\_of\_outstanding\_packets + 1)/Estimated\_path\_capacity$. Define two congestion thresholds, $\gamma < \delta$. For subflow i, when $Occupied_i \leq \gamma$, sending one more packet to subflow i is considered not to cause congestion on the path. When $\gamma < Occupied_i \leq \delta$, sending one more packet to subflow i is considered to cause congestion on the path. When $Occupied_i > \delta$, sending one more packet to subflow i is considered to cause loss because of congestion on the path. Now, the solution seems to be simple. For an MPTCP connection with two subflows, if sending one packet to both subflows will not cause congestion (i.e., $Occupied \leq \gamma$ for both subflows), the scheduler sends the next packet on the subflow with shorter srtt (i.e., the current default). If sending one packet to both subflows will cause loss because of congestion (i.e., $Occupied > \delta$ for both subflows), the scheduler delays sending the next packet temporarily. Otherwise, the less congested subflow is selected.

C. Techniques

Since the available capacities of network paths are changing all the time, accurately estimating the available path capacity of a subflow is challenging. Actually, the problem of estimating the available path capacity of a TCP connection is not new. The target of TCP congestion control is trying to dynamically adapt cwnd size to be roughly the available path capacity. Therefore, we can employ the techniques of TCP congestion control to estimate the available path capacity of a subflow.

Consider what parameters can be used to estimate the available path capacity of a subflow. Available parameters include per-subflow cwnd, slow start threshold, and RTT related values (sample RTTs, srtt, RTO values, etc).

An obvious question is why not just use per-subflow cwnd as the estimated available path capacity? As mentioned in the previous subsection, each MPTCP subflow employs a modified AIMD congestion control algorithm. The subflow's cwnd can temporarily exceed the available path capacity before detecting a loss. After detecting a loss, the cwnd decreases and the new slow start threshold is below the available path capacity. Even worse, if the available path

```
if (a loss has been detected since last run so that $ss\_threshold$ has been updated) then
    $Max = 2 * ss\_threshold$
    $Min = ss\_threshold$
    $Estimated\_path\_capacity = \frac{1}{2} * (Max + Min)$
else if ($Num\_of\_outstanding \geq Estimated\_path\_capacity$) and ($Num\_of\_outstanding < Max$) then
    $Min = Num\_of\_outstanding$
    $Estimated\_path\_capacity = \frac{1}{2} * (Max + Min)$
else if ($Num\_of\_outstanding \geq Max$) then
    $Max = Cwnd$
    $Min = Num\_of\_outstanding$
    $Estimated\_path\_capacity = \frac{1}{2} * (Max + Min)$
end if
```

Figure 2. Algorithm to Estimate Available Path Capacity of a Subflow

capacity is relatively large, it may take several round trips for the cwnd to reach the available path capacity after a loss. Therefore, using cwnd or slow start threshold as the estimated available path capacity can be imprecise.

What about RTT related values? In TCP congestion control techniques, one preventive rather than reactive algorithm is end-to-end delay-based congestion avoidance algorithm (DCA) [19]. DCA algorithms keep track of packet RTTs (called sample RTT). An increase in sample RTTs indicates increased queuing delay and more congestion in intermediate routers. TCP-Vegas [13] and FAST [14] employ this technique in their congestion control mechanisms. Can we use changes of sample RTTs to estimate the available path capacity? The answer remains no. As argued in [20], congestion information contained in TCP RTT samples cannot reliably predict packet loss, and thus cannot be used to accurately estimate available path capacity. The reasons are (i) the collected sample RTTs are too coarse to accurately track the bursty congestion associated with packet loss over high-speed paths, and (ii) sometimes short-term queue fluctuations which are not associated with losses make the changes of sample RTTs not reliably assess the congestion level at the router.

We need a stable method to estimate the available path capacity of a subflow. A feasible method is using multiple parameters. Reconsider the AIMD congestion control algorithm used by each subflow. After a loss (assume this loss is caused by congestion) is detected on a subflow, the subflow will decrease its cwnd. The current available path capacity must be somewhere between the cwnd size when a loss is detected and the new slow start threshold. Thus the technique of BI-TCP [9] can be used. BI-TCP uses a binary search algorithm where the cwnd grows to the mid-point between the last cwnd size where TCP has a packet loss and the last cwnd size TCP does not have a loss for one RTT period.

III. A Scheduling Policy Based on Estimated Subflow Path Capacities

Estimating available path capacity of a subflow: Initially, the estimated available path capacity of subflow i is set to a default maximum (a large constant). If a loss is detected on subflow i, the estimated available path capacity is set to the mid-point between the cwnd (i.e., max) before loss detection and the new slow start threshold (i.e., min). After the number of outstanding packets of subflow i reaches the estimated available path capacity, if subflow i does not detect further packet loss, it means that the available path capacity is under-estimated. Then a new min is set to the number of outstanding packets, and the estimated available path capacity is recalculated. After the number of outstanding packets reaches the max, if no loss has been detected, it means that the actual available path capacity has increased since the last loss. Then a new max is set to the current cwnd size, a new min is set to the current number of outstanding packets, and a new estimated available path capacity is recalculated. The full proposed algorithm to estimate available path capacity is shown in Figure 2.

Scheduling policy: When a packet is ready to be sent, the MPTCP sender side determines the available subflows. $Occupied_i$ is calculated for each subflow i. If the packet is an MPTCP level retransmission, it will not be re-sent on the subflow used for the original. Otherwise, subflows with $Occupied_i \leq \delta$ can be used to send the packet. If multiple subflows can be used and some available subflows will not be congested by sending one more packet (i.e., $Occupied_i \leq \gamma$), the subflow with the shortest srtt is selected. When all available subflows would be congested by sending one more packet (i.e., $Occupied_i > \gamma$), the subflow with the smallest $Occupied_i$ is selected. The full scheduling algorithm is shown in Figure 3.

Just as BI-TCP, our algorithm keeps the available path capacity of a subflow longer at the saturation point. Our proposed scheduler considers both the 'speed' and the congestion situation of each subflow. Thus, this proposed sched-

```
Each time the scheduler runs:
$Min\_srtt = $ 0xFFFFFFFF
$Min\_occupied = $ 0xFFFFFFFF
$Num\_uncongested\_path = 0$
$Num\_available\_path = 0$

for each subflow $i$ do
  if (next packet is a retransmission) and (it has been transmitted on subflow $i$) then
    continue
  end if
  if ($Num\_of\_outstanding_i \geq Cwnd_i$) then
    continue
  end if
  $Occupied_i = (Num\_of\_outstanding_i + 1)/Estimated\_path\_capacity_i$
  if ($Occupied_i > \delta$) then
    continue
  end if
  $Num\_available\_path$++
  if ($Occupied_i \leq \gamma$) then
    $Num\_uncongested\_path$++
  end if
end for

if ($Num\_uncongested\_path > 0$) then
  for each uncongested subflow $i$ do
    if ($Srtt_i < Min\_srtt$) then
      $Min\_srtt = Srtt_i$
      $Selected\_subflow = i$
    end if
  end for
else if ($Num\_available\_path > 0$) then
  for each available subflow $i$ do
    if ($Occupied_i < Min\_occupied$) then
      $Min\_occupied = Occupied_i$
      $Selected\_subflow = i$
    end if
  end for
else
  $Selected\_subflow = NULL$
end if
```

Figure 3. Algorithm of Proposed Scheduler

uler is consistent with the targets of the congestion control mechanism, and using network resources more efficiently.

IV. Evaluation Preliminaries

We implemented our proposed scheduler in the Linux kernel, and evaluated the performance of MPTCP data transfers with two subflows with our proposed scheduler vs. with the default scheduler. The comparison was made with and without cross traffic. Obviously, the coupled congestion control option is turned on.

Our test-bed is composed of two Cisco Linksys routers and three laptops running Ubuntu 11.10 (see Figure 4). Both laptops 1 and 2 are multihomed by using the tethered Ethernet interface and a Cisco USB Ethernet adapter. An MPTCP connection is established between laptops 1 and 2. Subflow 1 is established between the two tethered Ethernet interfaces, while subflow 2 is established between the two Cisco USB Ethernet adapters. Each Cisco USB Ethernet adapter has a small internal buffer that can queue up to 3 packets, thus the available path capacity of subflow 2 is less than that of subflow 1. A TCP connection is established between laptops 2 and 3. Our test-bed topology, as well as the speed of each link, are shown in Figure 4. The MPTCP traffic is generated by moving a 150MB file from laptop 1 to laptop 2, while the cross traffic is generated by moving a file of unbounded size from laptop 3 to laptop 2 with TCP.

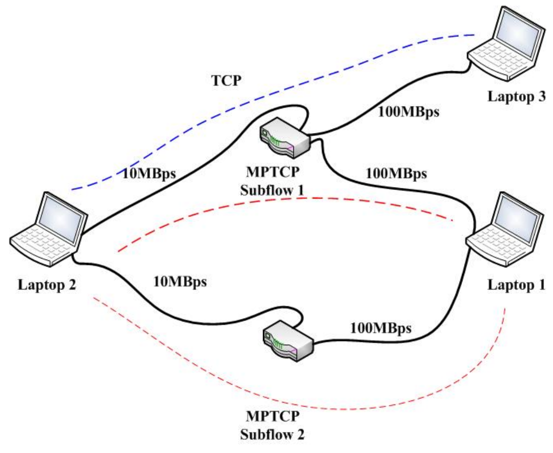

Figure 4. Test-bed Topology

V. Performance Evaluation

A. *Results without Cross Traffic*

Based on the analysis in Section II, we hypothesize our proposed scheduler uses network resources more efficiently than the default scheduler. Table I shows the results for the 150MB data transfer without cross traffic. Each entry in Table I represents the average of ten transfers with different schedulers (including the default scheduler (denoted Default) and several versions of proposed scheduler with different combinations of γ and δ). For example, a version of proposed scheduler with $\gamma = 0.5$ and $\delta = 1.0$ is denoted P-(0.5,1.0).

Since no artificial loss is introduced, all losses are caused by congestion. The default scheduler causes more retransmissions than our proposed scheduler with $\delta = 1.0$. Figures 5 and 6 show the cwnd of subflow 1 during the first five seconds of data transfer with the default scheduler and P-(0.9,1.0), respectively. As seen in Figure 5, the default scheduler sends data to subflow 1, and its cwnd increases until the number of outstanding packets exceeds the available path capacity and congestion loss occurs. The sender detects these losses and decreases its cwnd. Consequently, the default scheduler is continually over-sending packets to a subflow and creating its own losses. We call this phenomenon *over-feeding*. In Figure 6, over-feeding is avoided since P-(0.9,1.0) prevents the number of outstanding packets from exceeding the estimated available path capacity.

When $\delta = 1.0$, our proposed scheduler gains roughly 14.5% throughput over the default scheduler. When $\delta > 1.0$, the number of retransmitted packets increases and over-feeding occurs. In our proposed scheduler, the default value

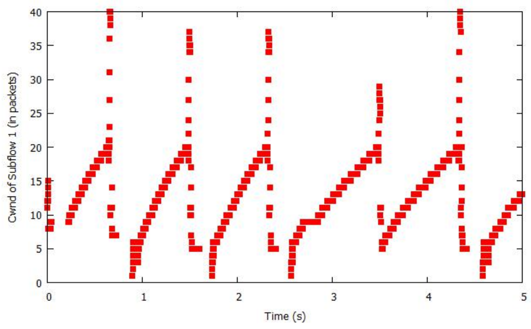

Figure 5. Cwnd of Subflow 1 with Default Scheduler

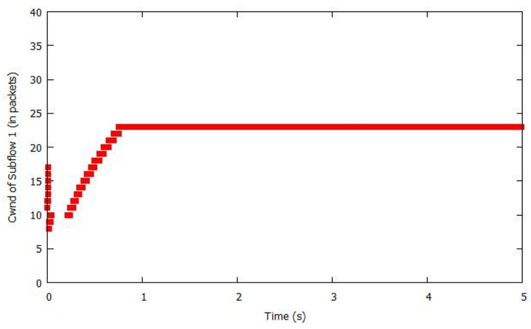

Figure 6. Cwnd of Subflow 1 with P-(0.9,1.0)

of δ is 1.0. Therefore, our first hypothesis is confirmed by the results in Table I. By alleviating over-feeding, our proposed scheduler can use network resources more efficiently.

B. *Results with Cross Traffic*

Our second hypothesis is our proposed scheduler is consistent with the target of the MPTCP congestion control mechanism. In other words, our proposed scheduler moves data away from congested links. This hypothesis is confirmed by the results in Table II, which shows the results for the 150MB data transfer with cross traffic. When $\delta = 1.0$, our proposed scheduler gains roughly 6.5% throughput over the default scheduler. More importantly, our proposed scheduler sends more data to subflow 2 than the default scheduler since subflow 2 is less congested than subflow 1.

C. *Discussion*

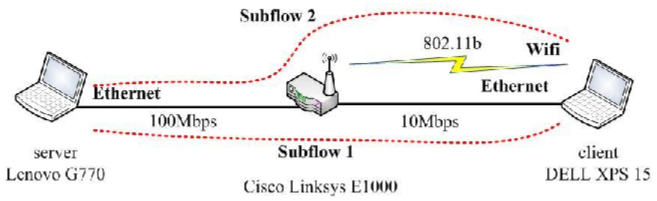

Figure 7. Test-bed Topology in our NR-SACKs paper

Obviously, the efficiency of our proposed scheduler depends on γ and δ. The over-feeding is alleviated by setting

Table I
MPTCP DATA TRANSFER WITHOUT CROSS TRAFFIC

	Default	P-(0.5,1.0)	P-(0.7,1.0)	P-(0.9,1.0)	P-(0.9,1.2)	P-(0.9,1.4)
Throughput (MBps)	1.58	1.82	1.81	1.81	1.80	1.59
Retransmissions (in packets)	1288	4	18	13	30	838
Packets sent on subflow 1	68123	68168	68358	68588	68927	68231
Packets sent on subflow 2	45513	44184	44008	43773	43249	44955

Table II
MPTCP DATA TRANSFER WITH CROSS TRAFFIC

	Default	P-(0.5,1.0)	P-(0.9,1.0)	P-(0.5,1.2)	P-(0.9,1.2)
Throughput (MBps)	0.92	0.99	0.98	0.97	0.96
Retransmissions (in packets)	1684	11	10	47	29
Packets sent on subflow 1	44459	32060	31270	32138	20110
Packets sent on subflow 2	69573	80299	81086	80257	92267

$\delta = 1.0$. What should the proper value of γ be? On one hand, γ cannot be too large (i.e., approaches δ). Reconsider the problem we reported in [12] shown in Figure 7. Obviously, the srtt of subflow 1 is always shorter than that of subflow 2. If the default scheduler is used, 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. If the available path capacity of subflow 1 $\geq$ the send buffer size, subflow 2 will only be used at the beginning of the data transfer. We call this phenomenon *biased-feeding*. If only subflow 1 is used, the maximum throughput is only 10MBps. While if both subflows are used, the maximum throughput is 21MBps. Biased-feeding seriously decreases the parallelism of data transfer. If our proposed scheduler is used, a large γ also causes biased-feeding, since our proposed scheduler is exactly the same as the default scheduler when all subflows' $Occupied_i \leq \gamma$.

On the other hand, γ cannot be too small (i.e., approaches 0). Consider an MPTCP connection with two subflows. The total available path capacity of both subflows $\leq$ the send buffer size. Also assume subflow 1 is established on a satellite link with longer delay and higher loss rate. Assume subflow 2 is established on a 3G path with shorter delay and lower loss rate. As γ approaches 0, more packets will be sent on subflow 1. Although biased-feeding is alleviated, at the MPTCP receiver side, the data arriving in-order on subflow 2 may not be delivered to the application layer since MPTCP level in-order data sent on subflow 1 has not arrived. Using both subflows can be worse than only using subflow 2. This phenomenon is similar as the problem reported in [7]. This phenomenon decreases the throughput and can cause *GapAck-Induced send buffer blocking* [11].

Currently, in our proposed scheduler, the default values of γ and δ are 0.5 and 1.0, respectively. Actually, the value of γ should be dynamically determined based on the characteristics (i.e., delay, packet loss and bandwidth) of subflows. Future work is needed to dynamically determine the value of γ.

VI. CONCLUSION AND FUTURE WORK

MPTCP may be the most significant change to TCP in the past 20 years. To achieve successful deployment of MPTCP, the key is the cooperation of the scheduler and the congestion control mechanism. While the MPTCP congestion control mechanism has been designed and standardized, it is the time to standardize the scheduler.

Our future work includes several problems. First, we need to do more extensive experimentations with our proposed scheduler. Second, we plan to integrate our proposed scheduler with the idea of NR-SACKs to analyze the throughput gain under different scenarios. Third, we plan to use our proposed scheduler to decide when to add/remove subflows. Fourth, we plan to design an algorithm to dynamically determine the value of γ in our proposed scheduler.

ACKNOWLEDGMENT

The authors would like to thank Qi Wang, 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] C. Raiciu, M. Handley, D. Wischik, *Coupled Congestion Control for Multipath Transport Protocols*, RFC 6356, October 2011.

[3] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, *Design, Implementation and Evaluation of Congestion Control for Multipath TCP*. 8th USENIX Symposium on Networked Systems Design and Implementation, Boston, Massachusetts, USA, March 2011.

[4] A. Singh, C. Goerg, A. Timm-Giel, M. Scharf and T.R. Banniza, *Performance Comparison of Scheduling Algorithms for Multipath Transfer*. .

[5] W. Zhang, Q. Wu, W. Yang and H. Li, *Reliable Multipath Transfer Scheduling Algorithm Research and Prototype Implementation*. 34th Proceedings of the Asia Pacific Advanced Network, Colombo, Sri Lanka, August 2012.

[6] A.S. Carmelita Görg, A. Timm-Giel, and M. Thomas-Ralf Banniza, *Performance Evaluation of Multipath TCP Linux Implementations*. Wrzburg Workshop on IP: Joint ITG and Euro-NF Workshop Visions of Future Generation Networks. 2011.

[7] 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.

[8] S. Barré, *Implementation and assessment of Modern Host-based Multipath Solutions*. PhD Dissertation, Universit catholique de Louvain, 2011.

[9] L. Xu, K. Harfoush, and I. Rhee, *Binary Increase Congestion Control for Fast, Long Distance Networks*. 23th AnnualJoint Conference of the IEEE Computer and Communications Societies, Hong Kong, China, March, 2004.

[10] O. Bonaventure, M. Handley and C. Raiciu, *An overview of Multipath TCP*. USENIX login;, October 2012.

[11] 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.

[12] F. Yang and P. Amer, *Non-renegable Selective Acks (NR-SACKs) for MPTCP*. The 3rd International Workshop on Protocols and Applications with Multi-Homing Support, Barcelona, Spain, March, 2013.

[13] L.S. Brakmo, and L.L. Peterson, *TCP Vegas: End to End Congestion Avoidance on a Global Internet*. IEEE Journal on Selected Areas in Communications, October 1995.

[14] C. Jin, D.X. Wei, S.H. Low, *FAST TCP: Motivation, Architecture, Algorithms, Performance*. 23rd AnnualJoint Conference of the IEEE Computer and Communications Societies, Hong Kong, China, March, 2004.

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

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

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

[18] D. Leith, and R. Shorten, *H-TCP:TCP for High-speed and Long-distance Networks*. 2nd PFLDNet Workshop, Illinois, USA, February 2004.

[19] R. Jain, *A Delay-based approach for Congestion Avoidance in Interconnected heterogeneous computer networks*. Comput. Commun. Rev. October, 1989.

[20] J. Martin, A. Nilsson, and I. Rhee. *Delay-Based Congestion Avoidance for TCP*. IEEE/ACM Transactions on Networking, June 2003.

[21] S. Vazhkudai. *Enabling the Co-allocation of Grid Data Transfers*. Proceedings of the 4th International Workshop on Grid Computing, 2003.