This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2010 proceedings

OmTCP: increasing performance in server farms

Iljitsch van Beijnum
IMDEA Networks
iljitsch.vanbeijnum@imdea.org

Arturo Azcorra
Universidad Carlos III de Madrid
and IMDEA Networks
azcorra@it.uc3m.es

Marcelo Bagnulo
Universidad Carlos III de Madrid
marcelo@it.uc3m.es

***Abstract*—Normal TCP/IP operation is for the routing system to select a best path that remains stable for some time, and for TCP to adjust to the properties of this path to optimize throughput. By executing TCP's congestion control algorithms on multiple paths at the same time, a multipath TCP can shift its traffic to a less congested path, thus maximizing both the throughput for the multipath TCP user and leaving more capacity available for other traffic on more congested paths. And when a path fails, this can be detected and worked around by multipath TCP much more quickly than by waiting for the routing system to repair the failure.**

This paper proposes a one-ended multipath TCP that is implemented on the sending host only, without requiring modifications on the receiving host, for the purposes of maximizing performance in transmissions from multiply connected large servers towards singly connected end-users and recovering from failures more quickly.

***Index Terms*—Multipath, TCP, congestion control, SACK.**

I. INTRODUCTION

THE current division of labor between routing protocols and transport protocols is leaving unused bandwidth and the potential for faster recovery from failures on the table. Real networks have many redundant links, but the most widely used routing protocols, such as BGP, OSPF and IS-IS, and their implementations allow packets belonging to a single TCP flow to travel through only a single path. If TCP were able to see more than just a single "best" path between two hosts, the rate adaption provided by TCP's congestion control algorithms would increase performance by using the capacity on currently unused or under-used links. According to [1], this can provide a performance increase of up to 25%. Not only would this increase the performance of multipath TCP sessions, but because these sessions use previously unused capacity, they leave more capacity to other flows on the default path, thereby providing resource pooling benefits [2] without the need for extensive manual or semi-automatic traffic engineering.

Sometimes, the challenge with protocol design is not coming up with a protocol that exhibits the desired behavior, but rather, to create a protocol that provides benefits in such a way that deployment is attractive. In the case of multipath data transfer, one approach is to create a new protocol or modify an existing protocol to make it aware of multiple paths and allow it to make use of them. This has been proposed within the context of SCTP [3], which is designed to manage multiple paths from its inception for redundancy purposes, so using multiple paths concurrently is a natural extension. Adding the multiple path capability to TCP has been suggested with ([4], [5], [6], [7], [8], among others) and without [9] the ability to use them concurrently. Creating a new protocol or requiring protocol changes poses no problems in the lab, where both the sender and the receiver are under complete control and run the same versions of the protocols. But in the real world, requiring changes in multiple systems that are not under the same administrative control is a huge barrier to deployment. When a new capability provides no immediate benefit because it is not implemented elsewhere yet, users are reluctant to spend time and money deploying those capabilities. Even when the capabilities are available automatically and for free, there may be hesitation: if the new mechanisms do not work properly, this causes problems at an unknown future time, when remote systems also deploy them.

For these reasons, we have developed a one-ended multipath TCP (OmTCP). All the modifications that allow for the use of multiple paths are contained in the TCP code of the sender, so OmTCP can be deployed without changes to applications or changes to TCP receivers. Deploying new mechanisms in TCP in just the sender is not without precedent: the well-known congestion control algorithms that have been added to TCP as of 1988 [10] are also implemented on the sending host only. These one-ended changes are possible because the receiver merely sends acknowledgments and window updates, all the decision making is done by the sender. We exploit this division of labor between the sender and the receiver by limiting our changes to the sender. An OmTCP sender simply sends standard TCP packets to the receiver over different paths, and the receiver acknowledges these packets as usual. Although

This work has been partly funded by Trilogy, a research project (ICT-216372) supported by the European Community under its Seventh Framework Programme. The views expressed here are those of the author(s) only. The European Commission is not liable for any use that may be made of the information in this document.

978-1-4244-6404-3/10/$26.00 ©2010 IEEE

today's network equipment works hard to make a each flow traverse a single path to avoid unnecessarily triggering the fast retransmit mechanism, packets flowing over multiple paths is not illegal. In fact, at one time this condition, and the resulting reordering of packets by the time they arrive at the receiver, was rather common [11].

Since acknowledgments do not explicitly indicate the path used by the sender, the sender must remember which packet was sent over which path, so it can perform congestion control over the different paths independently. However, because different paths will have different delay characteristics, packets will tend to arrive out of order. This complicates the fast retransmit algorithm and it obscures the sender's view of which packets were received by the sender. This is a problem limiting the usefulness of cTCP [8]. To work around this limitation, OmTCP requires the presence of SACK. With SACK (selective acknowledgments), TCP receivers can inform TCP senders of exactly which packets were received and which ones are still expected. The SACK capability is negotiated at the start of a TCP session. As SACK is implemented in both all major operating systems, it is generally available. We confirmed that Mac OS X 10.5.8 and Windows XP SP 3 not only negotiate the SACK capability in the three-way handshake, but also do in fact send SACK blocks when there are out-of-order or missing packets.

The one-ended multipath TCP that we propose is useful for bulk data transfers where a TCP sender has two or more attachments to the Internet at its disposal. Services that distribute large amounts of data, such as Youtube (video) or firefox.com (application updates) could be examples of such senders. Often, this type of service is replicated in multiple locations and clients are directed to the instance of the service that is determined to be closest to the client or best performing from the client's perspective. This is often done through DNS-based server selection. However, this is not always effective and has some disadvantages [12]. OmTCP comes into play after a server is selected by then finding and selecting the best of the available paths to the receiver.

In a content delivery scenario, at the receiving end, only a single attachment to the Internet will be utilized, as most users, and especially residential users, only use a single attachment to the Internet at one time, even if they have multiple (ADSL or cable as well as wireless 3G) at their disposal. So the different paths from the sender to the receiver are not disjoint: at some point between the sender and the receiver, the paths converge and merge. However, if there are differences in the non-overlapping parts of the paths, these differences can be exploited by selecting the path with the best properties. We do this by executing TCP congestion control for each path separately, so the sending rates over each path react to dynamic changes in round trip times and loss rates. In this paper, we acknowledge, but largely sidestep the issues of TCP fairness and limited receive buffers by only fully utilizing the path that provides the best performance. This leaves more of the capacity on congested paths available to flows that are incapable of moving to less congested paths.

The design of TCP, where all the decision making is performed by the sender, allows for the implementation of these mechanisms at the sending end, so that the intended benefits can be realized if an updated sender is communicating with an existing TCP receiver, allowing for easy incremental deployment in existing networks.

However, adding multipath capability to TCP brings up a number of challenges:

1. Avoiding reduced performance because of packet reordering in transit.
2. Avoiding stalls in the data flow when the receive buffer fills up after one path experiences losses.
3. How is fairness / TCP friendliness defined, and how is it achieved?
4. How is resource pooling achieved without harming performance compared to single path TCP under a wide variety of circumstances?
5. Will use of multiple paths increase the packet loss?
6. Does the use of multipath create the potential for instability?
7. Is OmTCP deployable?

II. Overview

The use case that OmTCP addresses is the one where a set of large servers has multiple connections to the Internet that they use to distribute large files to end-users. We assume that the vast majority of end-users only have a single attachment to the Internet. Although the degree of interconnectivity between routing domains is large and overprovisioning is common, it is still possible for an end-user communicating with the server farm to experience suboptimal performance; even though the server farm can distribute its traffic over its different attachments to the Internet, it does so in the blind if it does not measure congestion on the different paths towards the destination. This does not happen today because there are few mechanisms to move flows from more congested to less congested paths; manual or semi-automatic adjustments are not fine-grained enough to dynamically adjust to changing network environments.

Suboptimal performance can be the result of congestion and the resulting packet loss inside or between routing domains, or because certain paths are longer, increasing the RTT. As such, it is useful for servers to automatically discover which path provides the best performance towards a given user at a given time. We propose to do this by splitting a TCP flow into *subflows*, where each subflow maps to a

different path.

The mechanisms for selecting a path for each TCP segment are to be defined in future work. A simple such mechanism would be for each server to have multiple network interfaces where packets transmitted over each interface are routed through a different attachment to the Internet.

As OmTCP sends packets over multiple paths and needs to execute congestion control for each path, it records for each packet sent the path it was sent over. When acknowledgments arrive, those are attributed to the path that was used to transmit the data that is now acknowledged. This allows congestion control to be performed for each subflow individually. TCP's normal cumulative acknowledgments are not sufficient to make this work well, as the cumulative ACK will progress only when a consecutive new part of the sequence number space can be acknowledged. To avoid the inefficiencies depending on the cumulative ACK would produce, OmTCP requires SACK selective acknowledgments [13] so each acknowledgment can be correlated with the packet that generated it, even if packets arrive at the receiver out of order.

Most networks avoid distributing packets belonging to a single TCP flow over different links or paths, as this causes said reordering. The main issue with packets arriving at the receiver out-of-order is that this triggers the fast retransmit algorithm. When a packet is lost and the receiver receives packets that follow the hole in the packet stream, the receiver cannot acknowledge the data in the new packets with TCP's cumulative acknowledgment, so it keeps acknowledging the data that was received up to the hole. So the sender sees duplicate acknowledgments. After three of these duplicate acknowledgments, the fast retransmit algorithm assumes the packet immediately following the duplicate ACK has been lost and retransmits it without waiting for a timeout. The congestion window is reduced as it is assumed the loss was result of too high a send rate. Reordering of packets through the use of multiple paths may trigger fast retransmit, causing an unnecessary retransmission and congestion window reduction In OmTCP this situation is avoided by executing a fast retransmit when three duplicate ACKs are received for packets belonging to the same subflow. As a result, reordering between packets belonging to different subflows is ignored, only reordering within the same subflow triggers fast retransmit.

In addition to normal fast retransmit modified to work per subflow, there is an additional form of fast retransmit, which we will call "slow retransmit". When fast retransmit is incapable of retransmitting lost packets (because too many packets got lost or the lost packet was one of the last three before the sender stopped transmitting), retransmissions are normally delayed until the retransmission timeout (RTO) expires. Although in the meantime, the sender can continue to transmit using other subflows, all packets received after the missing packet(s) must be buffered by the receiver in order to be able to hand the data over to the application in the correct order. Typical values for the maximum receive buffer are 64k or a small multiple of that, while the minimum RTO is one second [14]. As a result, the receive buffer will almost certainly fill up before the RTO expires for sending rates above 1 MB/s and possibly at lower rates, making it impossible for the sender to transmit more data until the missing packets have been retransmitted successfully. To avoid this, OmTCP retransmits unacknowledged packets sent on subflows waiting for an RTO over another subflow when the receive buffer advertised by the receiver starts running low.

III. ALGORITHMS

Flow control is handled per-session, but separate instances of all congestion control variables, congestion control is independently handled by each subflow, including keeping a per-subflow RTO and user timeout.

A. SACK

The cumulative ACK does not provide the information necessary for OmTCP to correlate incoming ACKs with outgoing subflows; hence OmTCP depends on the presence of SACK. The SACK mechanism makes it possible for a receiver to indicate that three or four additional ranges of data were received in addition to what is acknowledged using a normal cumulative ACK. When packets are sent over multiple paths and arrive out of order, the information in the SACK returned by the receiver tells the sender exactly what data is being acknowledged, allowing for per subflow congestion control. If the receiving host does not indicate the SACK capability during the three-way handshake, a multipath TCP implementation should limit itself to using only a single subflow and thus disable multipath processing for the session in question.

B. Fast retransmit

To avoid the reordering issues discussed in [3], fast retransmit is executed per subflow. For each subflow, the amount of data acknowledged out-of-order is tracked. When this becomes more than 2 * *MSS* then the first unacknowledged segment is considered lost and retransmitted and the normal congestion window reduction happens for the subflow in question. However, the retransmission may be sent over any path.

C. Slow retransmit

Because missing packets create holes in the data stream, subsequent packets received, over the same or other paths, must be buffered in the receive buffer until the missing data is retransmitted. This is necessary to fulfill the requirement that data is handed over to the application in the original order. Normally, fast retransmit will tend to retransmit lost

packets before the receive buffer fills up. However, if a path completely breaks, or possibly if it is very severely congested, the receiver sees insufficient additional packets to trigger fast retransmit so retransmissions will only happen after a timeout. Unless the receive buffer is extremely large, this means all subflows stall when the receive buffer fills up. This situation persists until the RTO expires for the congested or broken path so the missing packets can be retransmitted. Should the path in question be completely broken, this will then lead to an almost immediate new stall, and the stall/RTO cycles will continue until the user timeout or R2 timer [15] for the subflow expires.

This can be solved by identifying subflows that have exhausted their congestion window and are now waiting for the RTO to expire. Unacknowledged packets that were sent over such a subflow can then be retransmitted over another subflow, and when those retransmissions reach the receiver, the holes in the data stream are filled in and the receive buffer is emptied.

These retransmissions should be done such that the missing packet arrives before it becomes necessary to stop sending data altogether because the receiver advertises an exhausted receive buffer. Such retransmissions therefore happen as the receive buffer space advertised by the receiver drops below *cwnd* for the path that will be used for the retransmission, presumably the path with the lowest *RTT*.

D. User timeouts

When the per-subflow R2 timer expires, the corresponding path is declared defective and no longer used. When the R2 timers for all subflows that are part of a session have expired, the session is torn down.

IV. Performance, Fairness and Resource Pooling

Multipath congestion control in general must achieve the following goals:

1. Performance must be no worse than in the single path case using only the fastest of the available paths.
2. No additional synchronization or instability beyond those present in existing TCP.
3. In the situation where multiple subflows use the same path, or part of the same path (sharing a bottleneck), the aggregate of these subflows must be fair to individual competing flows.
4. Multipath TCP must move traffic to less congested paths in order to provide resource pooling benefits.

Resource pooling means making a collection of resources behave like a single pooled resource. For this to happen, for each link, a fraction of its users must be able to move to another link ([2], [16]).

One way to implement multipath congestion control would be to execute normal (New)Reno congestion control on each subflow, so that each individual subflow competes with other TCPs on the same footing as a regular TCP session. If all subflows use disjoint physical paths, other TCPs are no worse off than in the situation where the multipath TCP were a regular TCP sharing their path, so this could be considered fair. The fact that the multipath TCP increases its bandwidth in direct relationship to the number of subflows used does not impact other TCP users. In this case, although multipath TCP sends at the same rate as regular TCP on a given path, resource pooling benefits are still realized to some degree because a given transmission completes faster so it uses up resources on each path for a shorter amount of time.

But if this approach is used when several logical paths share a physical path, OmTCP would take a larger share of the bandwidth compared to regular TCPs on that physical path. This would only be acceptable as TCP friendly for a very small number of subflows. The other end of the spectrum would be for OmTCP to conform to exactly the same additive increase, multiplicative decrease (AIMD) congestion window increase and decrease envelope that a regular TCP exhibits in its congestion avoidance state. We sidestep this issue by only executing normal AIMD on a single subflow. All other active subflows eventually end up with a *cwin* of 2 * *MSS* and thus only marginally contribute to the sending rate. However, we will first explore the fairness and performance issues of fully utilizing multiple subflows concurrently in slightly more detail.

A. Slow start fairness

Executing slow start independently for the different subflows largely accommodates each of the four requirements. A multipath TCP session a with subflows $a_1 \ldots a_n$ that are all in slow start has an aggregate congestion window of $n * x$ at time t, which is an *RTT* earlier than a regular TCP flow for every doubling of n. So while a regular TCP session b has a congestion window of x at time t, due to the doubling in size of the congestion window each *RTT*, session b will have a congestion window of $n * x$ at $t+RTT_b*\log_2(n)$ so the difference in transmission rate between a and b boils down to a small number of *RTT*s worth of time and should be considered negligible from a fairness perspective for bulk downloads, which is the use case considered for OmTCP at this time.

B. Congestion avoidance fairness

In congestion avoidance, the congestion window is allowed to grow by one maximum segment size per round trip. If multiple subflows each grow by one segment per *RTT*, the session as a whole (i.e., the aggregate of all n subflows belonging to the same session) will increase its sending rate with n maximum segment sizes per RTT. So such a multipath flow would grow its its sending rate faster than a regular, single path flow by a factor of n. This means that without

additional measures, multipath TCP claims a larger part of the available bandwidth and is thus more aggressive than regular TCP.

In order to avoid synchronization and stability issues, a lost packet resulting from congestion on one subflow cannot lead to a decrease in the congestion window for other subflows. That is because in that case congestion on one additional path would cause other paths to slow down, making it impossible to meet goal 1. This leaves two ways to couple the congestion states for different subflows in congestion avoidance, which is necessary to meet goals 3 and 4. The first approach for coupling the congestion states is to decrease the congestion window for a subflow based on the sum of the congestion windows for all subflows [17]. The second approach is to limit the sum of the congestion window increases to approximately one segment size per roundtrip.

An added complication is that, although adjusting to congestion based on the aggregate rate or aggregate congestion window is attractive from a theoretical standpoint, it is not possible to just add up the sizes of the *cwnd*s for the different subflows. The size of the congestion window in and of itself is rather meaningless: performance is determined by *cwnd* / *RTT*. So coupling the *cwnd* decrease requires taking into account the *RTT* to avoid favoring subflows with large congestion windows, which will tend to be those with high *RTT*s, for a given amount of bandwidth. Without this correction, traffic would move to the subflows with the highest *RTT*s. Also, the reduction based on the sum of the (corrected for *RTT*) congestion windows may be larger than the current value of the congestion window for the affected subflow, making it impossible to back off enough to achieve fairness, and complicating modeling. Finally, backing off more than regular TCP makes the difference between the minimum and maximum sending rates larger, increasing burstiness in the network.

As such, we adopt the second approach and decrease the congestion window for a subflow after a loss in the usual way and achieve fairness through limiting the growth of the congestion window. This creates a budget of one maximum segment size per *RTT* that can be spent on increasing the congestion window of any of the subflows that are in the congestion avoidance state. An obvious way to implement this is to simply extend equation 1 in [10] as follows:

$$cwnd_i \mathrel{+}= MSS_i * MSS_i / cwnd_i / n_{ca} \qquad (1)$$

With n_{ca} being the number of subflows that are in congestion avoidance. (Note that we use the maximum segment size currently in effect for the path, not the sender maximum segment size as in [10].) However, simply increasing the congestion windows for the different subflows at a rate of one maximum segment size per *RTT* divided by the number of subflows in congestion avoidance allows for suboptimal performance under certain circumstances. For example:

- if path A has a high *MSS* and path B a low one, or
- if path A has a low *RTT* and path B a high one, or
- if path A has a low loss probability and B a high one,

In each of these cases, the combined *cwnd* of a subflow over path A and a subflow over path B will grow slower than the *cwnd* of a single flow over path A. In other words, the *cwnd* growth allowance can be put to better use on some subflows than on others.

A simple approximation of the benefit of growing the *cwnd* for a subflow would be the number of bytes by which the *cwnd* grows multiplied by the number of segments that can be expected to be sent with the new *cwnd*. We use *LC* (loss correction factor) for the number of segments that can be transmitted before a loss happens and the *cwnd* must shrink. Further, because the AIMD envelope allows growing the *cwnd* by a segment size per round trip, shorter round trips mean more increase per unit of time. So the benefit gained is divided by the round trip time. Assuming that we may distribute the one segment size per *RTT* growth over the available subflows as we choose as long as the per-subflow growth is no less than 0 and no more than 1, we can formulate the following optimization problem:

$$\max \sum_{i \in n_{ca}} \frac{x_i MSS_i LC_i}{RTT_i} \qquad (2)$$

Subject to:

$$\forall i \in n_{ca} \quad 0 \le x_i \le 1 \qquad (3)$$

$$\sum_{i \in n_{ca}} x_i = fair \qquad (4)$$

Where *fair* is the factor by which multipath TCP is allowed to be more aggressive than single path (New)Reno TCP, presumably a value not too much higher than 1. x_i is the fraction of a maximum segment size that subflow *i* is allowed to grow per *RTT*. n_{ca} is the set of subflows in the congestion avoidance state. *LC* is the loss correction factor. Under the assumption that all loss is caused by external factors, a good choice for *LC* is the number of packets that can be transmitted before a loss occurs and the *cwnd* must be shrunk, in other words, $1/p$ (where p is the packet loss probability). Alternatively, under the assumption that all loss is caused by the subflow itself, *LC* should be high when the *cwnd* is close to the *ssthresh* and a much lower value as the *cwnd* approaches and exceeds *ssthresh* * 2, such as in the CUBIC window growth function [18].

Assuming *LC* is the inverse of p and *fair* = 1, the solution to the optimization problem is to grow the *cwnd* for the subflow with the most favorable $MSS_i * LC_i / RTT_i$ by one

MSS per *RTT*, and not grow the *cwnd*s of any other subflows. We can implement this by periodically recomputing MSS_i * LC_i / RTT_i for all subflows in the congestion avoidance state and selecting the subflow with the most favorable properties to be the active or primary one which sees its *cwnd* grow. Evaluating the properties of the different paths and possibly selecting a new active/primary path could happen at an interval in the order of 250 ms, but care must be taken to randomize this interval to avoid synchronization between different OmTCP flows sharing the same paths.

The optimization problem and its solution suggest that, based on the assumption of conforming to the (New)Reno AIMD *cwnd* growth envelope and maintaining TCP fairness *fair* = 1, it is impossible to both have optimal performance and fully use multiple paths concurrently. The only way to achieve performance equal to the single path TCP case is to grow the *cwnd* over the path with the best properties equally aggressively as single path TCP (one *MSS* per *RTT*), leaving no *cwnd* growth budget for other subflows. Actually using multiple paths concurrently with *cwnd*s larger than 2 * *MSS* either implies reduced performance compared to the single path case under some circumstances, or higher aggressiveness than single path TCP (*fair* > 1).

C. *ssthresh*

For subflows that do not grow their *cwnd*, it is likely that the subflow will still experience losses when other TCPs on the path grow their *cwnd*s (or for other reasons), which make the subflow's *cwnd* decrease until the *cwnd* reaches the minimum of two maximum segment sizes. The *ssthresh* is adjusted in the same way, so it also ends up being 2 * *MSS*. This means that if after some time of not growing its *cwnd*, a subflow becomes the primary subflow and starts competing head-to-head with other TCP flows, the *cwnd* for the subflow will have to grow from the minimum 2 * *MSS* to the appropriate size in congestion avoidance rather than slow start. This would be unnecessarily conservative. As such, we propose the following *ssthresh* handing when a subflow does not grow its *cwnd*. In this case, the *ssthresh* gets to grow by an *MSS* per *RTT* and when there is a loss, the *ssthresh* is reduced by half. As a result, the *ssthresh* will have (close to) the value the *cwnd* would have had if the *cwnd* had grown normally so when the subflow starts growing its *cwnd* again, it can do so in slow start initially, rather than go to congestion avoidance immediately.

IV. Conclusion and Future Work

We have explored the issues that come up with making TCP multipath-capable and presented the algorithms required for a one-ended multipath TCP that addresses these issues. The intersection between local performance, fairness and resource pooling is a topic of ongoing study for the concurrent multipath case, but we believe our results show that using a one-ended multipath TCP to dynamically select a primary path with the best properties is an achievable goal.

Our next steps will be to experimentally evaluate our analytical findings, and then address the more general case of a multipath TCP that makes full use of multiple concurrent paths. There are also protocol details to solve, such as how to use ECN with multipath TCP, and interactions to study, such as what happens when some subflows are in slow start while others are in congestion avoidance. Last but not least, mechanisms to interact with multipath routing would be an interesting extension of our approach.

Acknowledgment

Members of the Trilogy project, especially Costin Raiciu, Damon Wischik, Bob Briscoe, Lars Eggert, Mark Handley and Sébastien Barré have contributed valuable insights.

References

[1] A. Akella, S. Seshan, and A. "Shaikh. Multihoming Performance Benefits: An Experimental Evaluation of Practical Enterprise Strategies", proc. of the USENIX 2004 Annual Technical Conference, Boston, MA, 2004.

[2] D. Wischik, M. Handley and M. Bagnulo Braun, "The resource pooling principle", Computer Communication Review 38, 2008.

[3] J. Iyengar, P. Amer and R. Stewart, "Concurrent multipath transfer using sctp multihoming over independent end-to-end paths", IEEE/ACM Trans. Netw. 14, 5 (2006), 951–964, 2006.

[4] H. Hsieh and R. Sivakumar, "pTCP: An End-to-End Transport Layer Protocol for Striped Connections", proc. 10th IEEE International Conference on Network Protocols, 2002.

[5] V. Sharma, S. Kalyanaraman, K. Kar, K. Ramakrishnan and V. Subramanian, "MPLOT: A transport protocol exploiting multipath diversity using erasure codes", proc. IEEE INFOCOM, pp. 121–125, April 2008.

[6] M. Honda, E. Balandina, P. Sarolahti and L. Eggert, "Designing a Resource Pooling Transport Protocol", proc. IEEE GIS, 2009.

[7] K. Rojviboonchai and H. Aida, "An evaluation of multi-path transmission control protocol (M/TCP) with robust acknowledgement schemes", IEICE Trans. Communications, 2004.

[8] Y. Dong, D. Wang, N. Pissinou and J. Wang, "Multi-path load balancing in transport layer", proc. 3rd EuroNGI Conference on Next Generation Internet Networks, 2007.

[9] C. Huitema, "Multi-homed TCP", Internet-Draft, IETF, 1995.

[10] M. Allman, V. Paxson and W. Stevens, "TCP Congestion Control", RFC 2581, 1999.

[11] J.C.R. Bennett, C. Partridge, N. Shectman, "Packet Reordering is Not Pathological Network Behavior", IEEE/ACM Transactions on Networking, 1999.

[12] A Shaikh, R. Tewari and M. Agrawal, "On the Effectiveness of DNS-based Server Selection", proc. IEEE INFOCOM, 2001.

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

[14] V. Paxson and M. Allman, "Computing TCP's Retransmission Timer", RFC 2988, 2000.

[15] R. Braden, "Requirements for Internet Hosts - Communication Layers", RFC 1122, 1989.

[16] P. Key, L. Massoulié and D. Towsley, "Path selection and multi- path congestion control", proc. IEEE Infocom, 2007.

[17] F. Kelly and T. Voice, "Stability of end-to-end algorithms for joint routing and rate control", Computer Communication Review 35:2, 2005.

[18] S. Ha, I. Rhee and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant", ACM SIGOPS Operating System Review, Volume 42, Issue 5, July 2008, Pages 64-74, 2008.