Vol.2014-HPC-145 No.30
2014/7/29

Performance Evaluation of MPTCP over OpenFlow Network

Chawanat Nakasan[1,a)] Kohei Ichikawa[1,b)] Putchong Uthayopas[2,c)]

Abstract: This paper discusses the use of Multipath TCP (MPTCP), which is a TCP extension that allows multiple TCP flows to be associated to one application-layer logical connection, coupled with OpenFlow traffic engineering in a single stack to provide a comprehensive multipathing solution, with OpenFlow providing optimal path sets while MPTCP utilizing them. This design should be able to maximize bandwidth and network path utilization by allowing hosts to take advantage of presently-unused paths. Design of the testbed, also to be used by our research group in future projects, is also discussed in this paper. Finally, we discuss evaluation of network performance when using multiple paths, as well as concerns raised by our work. In summary, our system functioned as expected and provided feasible performance in small virtual network. This design should be scalable to benefit distributed file storage systems, data-intensive services, or any high-performance computing systems.

1. Introduction

Recently, networked systems, including distributed file storage, application or web-service servers, and computation clusters, have become more complex, having more server nodes and being scattered around the world across the wide-area network. Additionally, many networked systems are *multi-homed*, that is, is connected to multiple gateway networks leading into the Internet. Prevalent examples include mobile phones that can connect to the Internet via the cellular network using 3G and the local-area network using WiFi.

While new applications, server distributions, and data-intensive systems can be easily created, modifying networking mechanisms or protocols to accommodate them is not very easy due to the distributed nature of networking. Any changes in networking must take into account backward- and forward-compatibility. *Software-defined networking* or *SDN* can be utilized to rapidly test and implement new networking protocols and concepts with minimal cost.

It is important that networked systems are able to benefit from the existence of multiple paths by using multiple paths or routes at once to transfer data from one host to another. For network logistics reasons, we believe that the application should be able to create only one *application-layer connection* and let the transport mechanisms in the operating system handle the splitting and stripping of such *application-layer connection*, by creating and managing multiple *transport-layer connections* as deemed necessary by the operating system itself. However, such system is not in general use.

However, current networked systems still work on the logic that one *application-layer connection* is *"backed"* by only one *transport-layer connection* [1]. Therefore, if an application wishes to create multiple paths for performance or failover reasons, it must perform splitting and/or stripping of the connections manually. Moreover, if two connections are made to transfer a file which is pre-striped by the application and one connection fails, the application must resend the failed data transfer on the working connection by itself.

By creating a working multipathing solution at the transport layer or lower, multipathing is potentially available to every application without the need for the application developer to create specific complex handling solutions for multipathing scenarios like the application-level multipathing. In the previous file transfer example, a working transport-layer multipathing would use the same transport-layer buffer for the two transport-layer connections. When one connection fails, the (generalized) retransmission and sliding window mechanism would work on the remaining connection without requiring application intervention. As an additional benefit, we can meet higher demands for bandwidth used by big-data operations in the industry.

As mentioned earlier, there are many kinds of networked systems that work across the wide-area network. In our research, we plan to focus on distributed storage. Wide-area networks have higher latency and smaller bandwidth compared to local-area network. Additionally, wide-area network routing is much more complex, consisting of multiple autonomous systems working together to provide connectivity to the end hosts. Distributed storage tends to have large file transfers, a situation where bandwidth utilization is important.

[1] Nara Institute of Science and Technology, Ikoma, Nara 630–0101, Japan
[2] Kasetsart University, Chatuchak, Bangkok 10900, Thailand
[a)] chawanat.nakasan.cb5@is.naist.jp
[b)] ichikawa@is.naist.jp
[c)] pu@ku.ac.th

2. Background

2.1 SDN and OpenFlow

OpenFlow is an SDN protocol which allows network traffic control and management from the centralized OpenFlow Controller instead of distributing such control to each network element (switches, routers, etc.). By centralizing network control at the controller, the network elements can be programmed to add or remove any switching or routing rules in its *flow table*. While OpenFlow provides programming flexibility to the network and allows many concepts such as QoS or traffic engineering to be realized, it cannot modify communication pattern between the end hosts themselves. In traditional TCP/IP protocol suite, only one route or path is used per connection. This limits the maximum bandwidth of the entire connection to that of the segment with the least bandwidth, and this cannot be overcome by simply adopting OpenFlow.

2.2 Multipathing

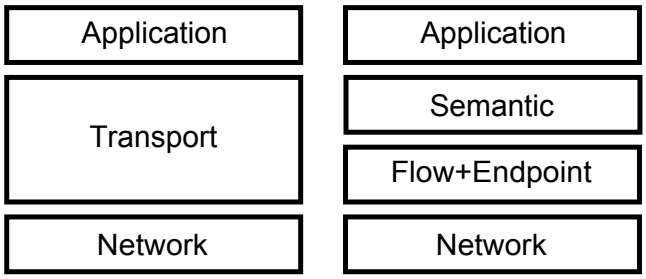
RFC 2992 [2] mentions many existing multipathing concepts and algorithms which allow multiple paths to be used in one connection, including *Equal-Cost Multipath (ECMP)* which is available in some routing protocols including Open Shortest Path First (OSPF), Intermediate System to Intermediate System (ISIS), and implicitly in Routing Information Protocol (RIP); all of these work at network layer. By using ECMP, routers will load-balance among the paths which have equal cost with respect to the destination. However, ECMP routing is performed at network layer and is problematic to TCP, because TCP is not aware of the presence of ECMP operations. Some ECMP implementations scatter packets in a round-robin fashion, making packets prone to arriving out-of-order which will prompt TCP to use retransmission mechanisms when multiple consecutive ACKs are received, decreasing network performance [1].

On the other hand, application layer multipathing is possible but error-prone [3] and hard to maintain [1]. The task of working with the multiple paths and flows will fall upon the application, which is not aware of the many mechanisms that are already available and working in the transport layer [4]. Some mechanisms that involve specific transport-layer packets (TCP *SYN* and *ACK* are common examples) may or may not work at all when implemented in applications, because applications do not see the information necessary for doing so.

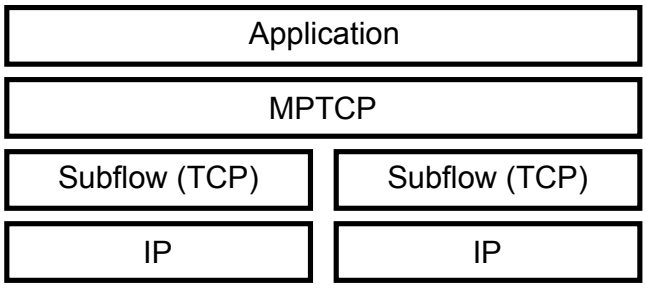
2.3 Multipathing in Transport Layer

For reasons described above, it is more desirable that multiple paths be managed from the transport layer, which is more informed about each path than applications [3] but is also aware of the high-level connections not accounted for in the network layer. While SCTP, a transport-layer protocol, is also capable of multipathing, the feature is used only for redundancy purposes and not to increase bandwidth utilization [5].

Many researchers look at transport layer as a viable position to multipath. Many works such as *concurrent TCP (cTCP)* [1], *M/TCP* [6], and *Heterogeneous Multipath Transport Protocol (HMTP)* [7] provide multipathing solutions in transport layer. Many of these are mentioned in [8]. Among these, we find MPTCP [9] to be an interesting protocol as it is implemented as TCP options and not a separate protocol, allows paths to be added and removed while the connection is still established, allows multipathing to be initiated from sender or recipient which allows it to work in a greater variety of environments, manages congestion control as a group (not per individual TCP connection), and has integral security features. It also has extensive research, including works on making a Linux kernel implementation [10].



(a) Comparison between the original TCP/IP model (left) and the decomposed-transport layer model (right)



(b) MPTCP interpretation of the decomposed layers. Implemented in this way, MPTCP can present itself to the application as a single entity that manages multiple TCP subflows.

Fig. 2: Relationship between the traditional network layer model and derivatives that lead to the core design of MPTCP

2.4 MPTCP

Multipath Transmission Control Protocol, or *MPTCP* is an extension to *TCP* at transport layer to utilize multiple paths between two network endpoints by stripping data into multiple *subflows*. Each subflow then behaves like a TCP flow, with its own congestion control, send and receive windows, and so on. Multipathing allows us to use more than one path in one logical connection, increasing bandwidth utilization, improving redundancy and stability, as well as allowing seamless handovers in certain environments. It is especially useful in multi-homed networks and systems, which are more prevalent today.

By decomposing the transport layer into two sublayers as in **Fig. 2a** [11], MPTCP can separately recognize the end-to-end and point-to-point situations better than the traditional model by having the upper half, which is the MPTCP extension, work only with managing the connection and subflows, while the lower half works like ordinary TCP, dealing with congestion and other matters in each subflow as in **Fig. 2b**.

However, even with all the advantages MPTCP could offer, there still is no guarantee that the multiple paths employed (a *"path set"*) will always be the most optimal *path set* possible, as

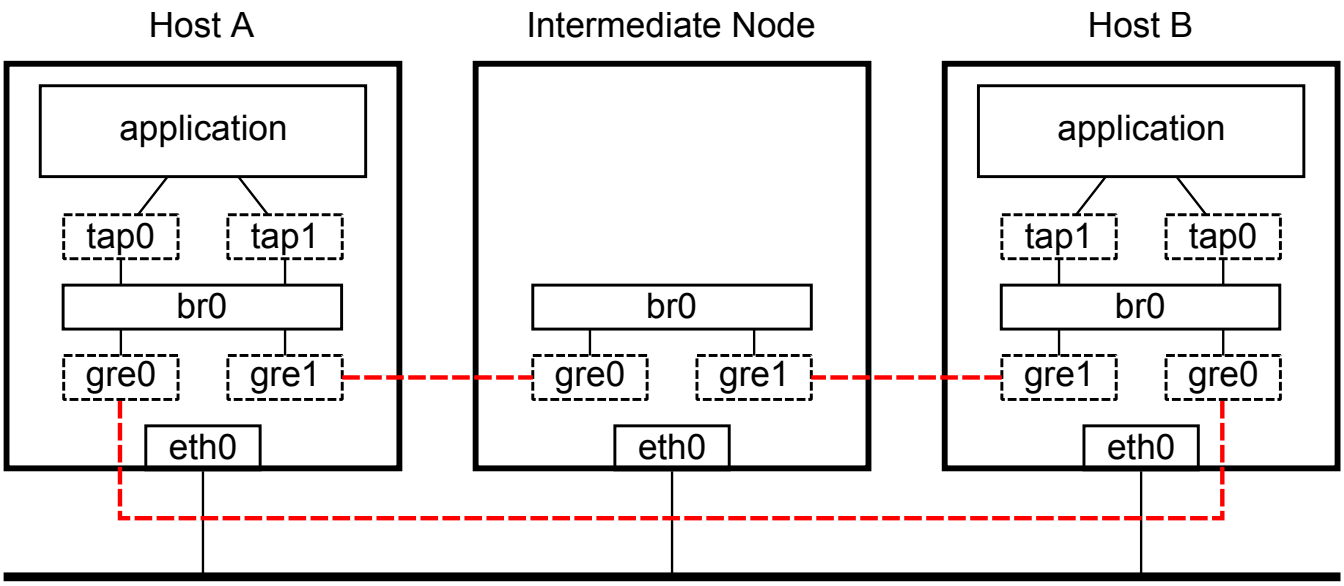


Fig. 1: Our implementation of the MPTCP/OpenFlow stack. MPTCP is used to provide multipath connectivity. Currently, OpenFlow flow entries are manually set up to behave like a simple routing element.

the paths in a *path set* may overlap or collide with each other reducing the leverage provided by MPTCP. MPTCP does not have control over path selection at network layer.

While OpenFlow and MPTCP each has its own limitations, we believe that by putting them to work in a single implementation, the traffic engineering and routing techniques in OpenFlow can provide an optimal *path set* that MPTCP can use to manage multiple subflows while presenting a single interface to the application, allowing maximum performance to be achieved with minimal additional effort from the application. While such solutions have already been proved workable in [12], but we have a further plan to optimize it specially for wide-area networking and distributed data storage environments, as well as being a generic, one-size-fits-most solution.

3. Design and Implementation

3.1 Design Goals

We aim to create a suite of MPTCP and OpenFlow working cooperatively in the same system implementation. Ideally, OpenFlow would find optimal *path sets* while MPTCP utilize them to distribute traffic across multiple paths to maximize the use of available bandwidth in the network.

We choose to approach the problem and solve it in certain conditions, in order of importance: (1) to maintain compatibility to applications, our solution should require minimal or no changes to applications; (2) multiple routes should be used between the two end hosts; (3) data transfer rate should increase proportionally to the number of paths being concurrently used; and (4) where possible, the multiple routes should not overlap or intersect with each other, at least in the overlay network level .

In this work, we will attempt to fulfill the first two conditions by using MPTCP, while hypothesizing on the third (more information in section 3.4). The final condition will be met in a later work by using a more advanced OpenFlow controller.

3.2 Design Decisions

It is essential to maintain compatibility between the multipathing solution and applications. MPTCP preserves *vertical backwards compatibility* by providing the same interface to applications as TCP, so we think it is a viable method. If an application does not already use application-layer multipathing on its own, it will benefit from MPTCP. If the application already does, it can keep using its own multipathing mechanisms in conjunction with MPTCP. We expect the performance may be degraded, but the application should still function. It is also trivial to mention that MPTCP suite is compatible with IP, because TCP packets generated by TCP are already encapsulated in IP. Additionally, MPTCP is also *laterally backwards compatible*, able to revert back to TCP if a host does not have MPTCP installed. This is because MPTCP is implemented as a set of TCP options, initiated by sending `MP_CAPABLE` option during 3-way handshake. If a similar option is not returned, the initiating host immediately knows it has to fall back.

As mentioned in the previous section, MPTCP has no control over the paths taken by each subflow, and a path set can be suboptimal. In order to use the most optimal multiple paths, multiple routes have to be created at the network level so MPTCP can utilize them. These paths are situated on different networks, and OpenFlow-based solution is used to provide the overlay network that fits this description. We currently set up flow entries manually, but we plan to create a controller that works well in multipathing context in the future to make the work scalable and suitable for general use.

3.3 The Testbed

We implemented our testbed as described in **Fig. 1** on a VMware vSphere environment. Each node in our setup is deployed onto different host machines, interconnected with a total rate of 1 Gbps, shared between all users. The GRE connections established between each pair of hosts (red lines) are manually

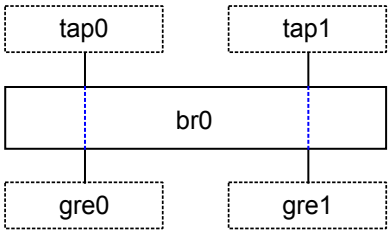


Fig. 3: We set up flow entries in the virtual bridge to forward packets between corresponding GRE and TAP devices.

limited to 100 Mbps.

We obtained MPTCP kernel (Linux 3.11) and utilities from [13]. The kernel provides MPTCP functionality while the other utilities enable us to prevent MPTCP from using certain interfaces. In our case, we disabled `eth0` to restrict MPTCP to GRE connections.

For OpenFlow connections, we used Open vSwitch to provide virtual OpenFlow switches. The GRE and TAP devices are connected by using the flow entries in the switches as shown in **Fig. 3**. The TAP devices are then used by MPTCP.

For reference, we used following versions of software components:

- CentOS 6.5
- MPTCP Kernel based on mainline version 3.11 [13]
- Open vSwitch version 1.11.0
- iperf version 2.0.4

3.4 Hypotheses on Performance

The GRE connections have data rate limited to 100 Mbps. Therefore, the total theoretical bandwidth between host A to host B should be 200 Mbps. When we test a single connection using `iperf`, we measured a bandwidth of approximately 95 Mbps. We therefore hypothesize that when using two GRE connections, the total data rate should not be greater than 190 Mbps.

3.5 Evaluation Method

We evaluate the network performance by running ordinary testing mechanisms via `iperf`, which in our opinion is a very straightforward TCP test. If MPTCP is active and working with unmodified applications, `iperf` should be able to utilize more bandwidth than without MPTCP. In addition to comparison against our hypothesis, we will create two variables that may affect total bandwidth. The first variable is the MPTCP State between on, on (but only one path is allowed), and off. The other variable is the number of `iperf` threads from 1 to 4. This will yield 12 settings in total. Each setting is then run for 10 seconds, 5 times, with 5-second pause period in between to make sure that the next test is not affected by the buffers of previous test. The result average bandwidth is then averaged for each setting. Transient bandwidth information is also obtained every 0.5 seconds.

MPTCP is integrated into the kernel and can be turned on and off at the file `/proc/sys/net/mptcp/mptcp_enabled`, while number of connections on `iperf` can be set by using `-P #`, where # is the number of parallel threads to run.

Table 1: Bandwidth measured (in Mbps) between two host nodes over two GRE links. Each test condition is run for ten seconds for five times, five seconds apart.

	MPTCP, many paths	MPTCP, one path	w/o MPTCP
1 Thread	189.13	95.17	89.14
2 Threads	190.16	95.46	95.33
3 Threads	190.84	95.23	95.62
4 Threads	191.41	95.52	95.67

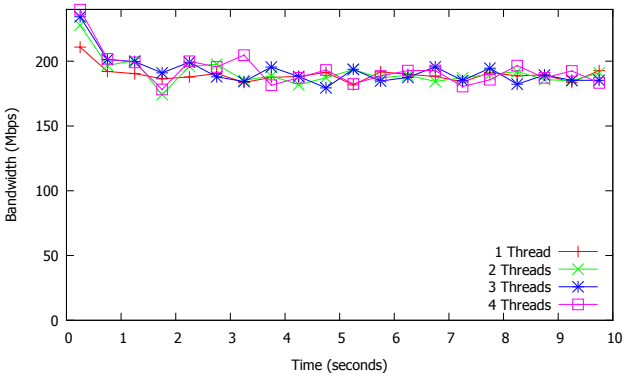


Fig. 4: Transient bandwidth measured during 1-thread and 4-thread MPTCP tests.

4. Evaluation Results

Since our testbed is implemented on a virtual environment, our system can be easily configured and modified. In this state, we decided to keep the network topology as simple as possible, by having only two nodes with one intermediate. After running the experiment, we managed to capture data rates when using different configurations as detailed in **Table 1**.

4.1 Bandwidth utilization of MPTCP

From our results in Table 1, we discovered that by using MPTCP over two paths, the maximum bandwidth between the two hosts increased approximately one-fold. This means MPTCP can fully utilize the new path assigned to the two hosts. As shown in the second column in Table 1, MPTCP can function as well as without MPTCP even if there is only one path, suggesting that performance drop, if any, is negligible when using MPTCP. Additionally, MPTCP seems to be able to adapt to changing network conditions on its own. This should allow MPTCP to adapt to any kind of network as long as we provide appropriate paths to it by the means of OS- or network-level configuration.

4.2 Performance of multiple threads

In addition to average bandwidth, we also collected transient bandwidth measured by `iperf` every 0.5 seconds, as shown in **Fig. 4**. We discovered that MPTCP can utilize bandwidth in a stable manner over both time and number of threads, no worse than ordinary TCP.

4.3 Packet analysis

In order to check if MPTCP initialized and operated properly, we have captured the packets exchanged between the two hosts

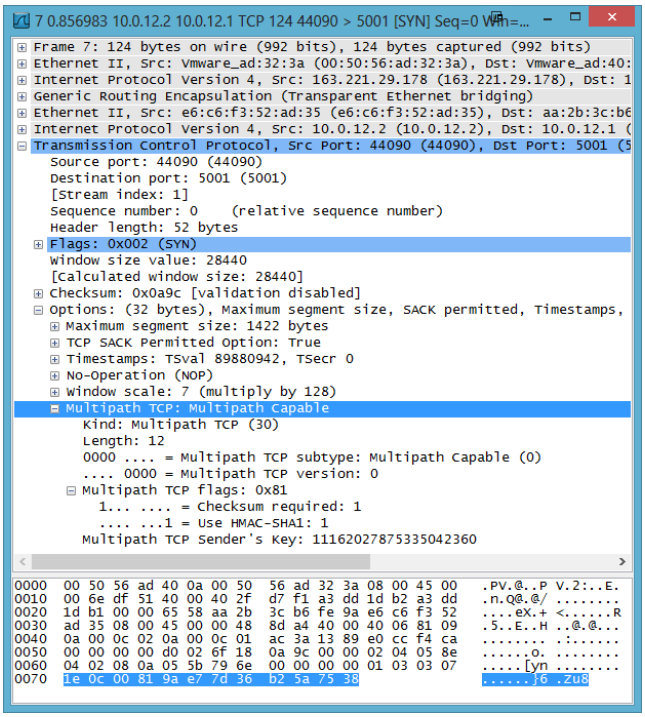


Fig. 5: Wireshark analysis of the first packet captured while MPTCP connection is being established. Here, an MP_CAPABLE option is seen. GRE encapsulation between public and private IP addresses are also visible.

using `tcpdump` and put them into Wireshark, version 1.8.6. [*1] **Fig. 5** shows the first MPTCP packet with `MP_CAPABLE` option being sent from `iperf` client to server. By inspecting multiple packets in capture files, we also found out that multiple endpoints are used during communication. When each pair of endpoints is being initialized, an `MP_JOIN` option is sent in addition to normal TCP handshake.

5. Discussion

In this work we have successfully implemented a host which utilizes both MPTCP and primitive flow entries to increase application-layer bandwidth between two end hosts. We also discovered that in this case of a virtual local network, MPTCP can work with no significant performance drop compared to ordinary TCP.

By successfully designing and implementing a working MPTCP–OpenFlow combined testbed that yields superior bandwidth utilization and point-to-point data rate, we further reinforce the existing proof that MPTCP is viable as a drop-in improvement to TCP. This brings us closer to our research goal.

This, however, is still far from our main research theme. While `iperf` can simulate large file transfer, it still works on a host-to-host basis and still does not fully reflect distributed storage or other HPC systems.

5.1 Future Works

To continue upon our research focus, we will need to create an OpenFlow controller that is capable of multipath routing, being able to find optimal path set for MPTCP. One among many possible ways is to predict the throughput and choose the most optimal path set. [14]

Concerns regarding network saturation (with a massive number of connections) should be addressable by quality-of-service schemes that can be enforced by OpenFlow. Another concern was OS-level overhead because TCP already uses memory to maintain buffers and states. By expanding to MPTCP, we multiply TCP memory overhead and add MPTCP overhead to it. Additionally, MPTCP adds a new transport-semantic sublayer, which can contribute to latency and CPU usage.

The MPTCP kernel we use also allows `ip link` option of a network device to be configured to `backup` and `handover` modes, in addition to `on` and `off` which we are already using. We believe that by experimenting and using these options, we should be able to adjust network behavior and make it suitable for real-world scenarios.

This work only tested MPTCP–OpenFlow stack in a virtual local-area network. We believe that the different conditions present in physical local-area network, with Open vSwitch or hardware OpenFlow switches, and wide-area networks will present new dimensions of challenges to our research. To gain confidence that our system can improve networked system performance while being easily implemented and deployed, we need to expand our research to test in those challenging environments.

6. Conclusion

In this paper, we discussed our rationale to support transport-layer multipathing, and evaluated performance of a system configured to use Multipath TCP with OpenFlow on the same stack. By using MPTCP kernel which works transparently, unmodified applications can use MPTCP at optimal performance by using maximum available bandwidth and could use secondary path automatically when it is routable. Even with one path, MPTCP performs no worse than ordinary TCP. Therefore, MPTCP should be a good option for networked systems engineers and operators when considering multipathing solutions, especially in distributed file storage systems, data-intensive services, or any high-performance computing systems. However, MPTCP adds a new sublayer to the transport layer and may increase overhead in terms of memory and CPU processing. We will investigate this in a future work.

Acknowledgments This work was partly supported by JSPS KAKENHI Grant Number 25730075. The first author also expresses his appreciation to the Japan Student Services Organization (JASSO) for the Honors Scholarship during FY2013.

References

[1] Dong, Y., Wang, D., Pissinou, N. and Wang, J.: Multi-Path Load Balancing in Transport Layer, *Next Generation Internet Networks, 3rd EuroNGI Conference on*, pp. 135–142 (online), DOI: 10.1109/NGI.2007.371208 (2007).
[2] Hopps, C.: Analysis of an Equal-Cost Multi-Path Algorithm, RFC 2992, RFC Editor (2000).
[3] Barré, S., Bonaventure, O., Raiciu, C. and Handley, M.: Experimenting with Multipath TCP, *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, New York, NY, USA, ACM, pp. 443–444 (online), DOI: 10.1145/1851182.1851254 (2010).
[4] Jacobson, V.: Congestion Avoidance and Control, *Symposium Pro-*

[*1] Wireshark supports MPTCP since version 1.7.1.

ceedings on Communications Architectures and Protocols, SIGCOMM '88, New York, NY, USA, ACM, pp. 314–329 (online), DOI: 10.1145/52324.52356 (1988).

[5] Ford, A., Raiciu, C., Handley, M., Barre, S. and Iyengar, J.: Architectural Guidelines for Multipath TCP Development, RFC 6182, RFC Editor (2011).

[6] Rojviboonchai, K. and Hitoshi, A.: An evaluation of multi-path transmission control protocol (M/TCP) with robust acknowledgement schemes, *IEICE transactions on communications*, Vol. 87, No. 9, pp. 2699–2707 (2004).

[7] Hwang, Y., Obele, B. O. and Lim, H.: Multipath transport protocol for heterogeneous multi-homing networks, *Proceedings of the ACM CoNEXT Student Workshop*, ACM, p. 5 (2010).

[8] Chihani, B. and Collange, D.: A Survey on Multipath Transport Protocols.

[9] Ford, A., Raiciu, C., Handley, M. and Bonaventure, O.: TCP Extensions for Multipath Operation with Multiple Addresses, RFC 6824, RFC Editor (2013).

[10] Raiciu, C., Paasch, C., Barre, S., Ford, A., Honda, M., Duchene, F., Bonaventure, O. and Handley, M.: How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP, *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, Berkeley, CA, USA, USENIX Association, pp. 29–29 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2228298.2228338⟩ (2012).

[11] Ford, B. and Iyengar, J.: Breaking up the transport logjam, *ACM HotNets, October* (2008).

[12] van der Pol, R., Boele, S., Dijkstra, F., Barczyk, A., van Malenstein, G., Chen, J. H. and Mambretti, J.: Multipathing with MPTCP and OpenFlow, *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, IEEE, pp. 1617–1624 (2012).

[13] Paasch, C., Detal, G. and Heidelberger, D.: Multipath TCP, https://github.com/multipath-tcp (2014).

[14] Tatsunori, M., Hirotake, A. and Kazuhiko, K.: MPTCP with path selection mechanizm based on predicted throughput on OpenFlow-enabled environment, *IPSJ SIG Notes*, Vol. 2013, No. 7, pp. 1–5 (online), available from ⟨http://ci.nii.ac.jp/naid/110009633037/en/⟩ (2013).