The Virtualized MPTCP Proxy Performance in Cellular Network

Sunyoung Chung, Seonghoon Moon, Songkuk Kim
Yonsei Institute of Convergence Technology
School of Integrated Technology
Yonsei University
Incheon, Korea
sunychung@yonsei.ac.kr, seonghoon.m@yonsei.ac.kr, songuk@yonsei.ac.kr

***Abstract*—For massive traffic handling in the cellular network, network function virtualization (NFV) is considered to be the most cost-efficient solution in the 5G networks. Since NFV decouples the network function from the underlying hardware, the purpose-built machines can be replaced by the commodity hardware. However, NFV might suffer from the very fact that it is a solely software-based solution. The objective of this paper is to find out the NFV performance issue in cellular network. Also, we want to investigate whether NFV is comparable with MPTCP connections. Since not many servers are MPTCP-enabled, a SOCKS proxy is usually deployed in between to enable MPTCP connections. We regarded a virtualized proxy as an NFV instance and set up two types of virtualized SOCKS proxies, one as KVM and the other as docker. We also tried to find out if there is a performance difference between hypervisor-based and container-based virtualization in our setting. As the results show, the docker proxy performs better than the KVM proxy. In terms of resource consumption, for example, the docker utilized 31.9% of host CPU, whereas the KVM consumed 36.9% when both of them handling 2,000 concurrent requests. The throughput comparison of different TCP connections reflects the characteristics of MPTCP flow that performs best in a long and large flow. The latency between the server and the proxy determined the throughput of MPTCP with a virtualized proxy. If the latency between the server and the proxy gets larger (RTT 100ms), the MPTCP proxy throughput of all three different flow got worse than the single TCP connections, whether it is a short flow (1KB) or a long flow (164MB). However, if the latency is in the middle range (RTT 50ms), the MPTCP proxy throughput of a short (1KB) and medium (900KB) flow works poorly, but a long flow (164MB) still works better than the single TCP connections.**

Keywords*—*Multi-path TCP, Multi-path TCP proxy, network function virtualization, cellular network, KVM, docker, network performance

I. Introduction

With the prevalent use of mobile devices, such as smart phones, tablets, and smart watches, the data traffic in the cellular network has been increased dramatically over the past few years. As stated in the Cisco Global Mobile Data Traffic Forecast, global mobile data traffic grew 63 percent in 2016 and it has reached 7.2 exabytes per month at the end of 2016 [1]. The traffic growth will, even more, be accelerated each year since IoT (Internet-of-Things) technologies will be introduced in the next-generation mobile networks. When almost everything is connected and diverse entities are communicating with each other using various communication technologies, we need to find out proper methodologies to cope with this massive data traffic.

The straightforward solution for handling the increasing traffic is to build more infrastructures that carry more traffic. However, the hardware-based solutions have limited scalability and operational costs. Thus, scalable software-based solutions, like network function virtualization (NFV) are emerging as an alternative to the hardware-based solutions. NFV enables decoupling the required network functions from the specific hardware devices. Rather than the dedicated machines working solely for the limited purpose, NFV enables software-implemented boxes, usually in the form of virtual machines or containers, handle the traffic on demand [2].

However, there are several technical issues when deploying virtualized network function (VNF) as a substitute for a typical network appliance [3]. Most frequent one is the performance issue. Since the capacity of VNF might be less than that of a dedicated machine, VNF would not perform as well as the purpose-built machines do. Therefore, a proper algorithm for building a clustered VNFs and balancing the workload between them is required for consistent network performance. Another issue is dynamic deployment. Unlike the dedicated hardware, VNF should be initiated in the right location and at the right time with flexible management.

Other than the network function virtualization, we can also consider for enhancing the performance of the existing data transfer protocol to carry more data fast and reliable. In this light, MPTCP (Multipath TCP) has been gaining the attention recently to relieve the network traffic in the cellular network. Multipath TCP (MPTCP) is an extension of TCP, utilizing multiple paths at the same time by a single connection to maximize the data transfer performance. The legacy TCP only utilize a single path connection between the two endpoints even when it is possible to employ multiple paths between them. Whereas MPTCP enables multiple paths by setting up independent TCP subflows. The MPTCP layer is responsible for handling the subflows' TCP connections. The MPTCP layer issues sequence number for each subflow and orders the connection-level data sequence accordingly. Hence, for better MPTCP performance, the creation of available subflow connection and the subflows' packet scheduling should be comparable with the existing network applications [4] [5].

In this paper, we tried to figure out NFV performance in the cellular network. As mentioned in the beginning, NFV has

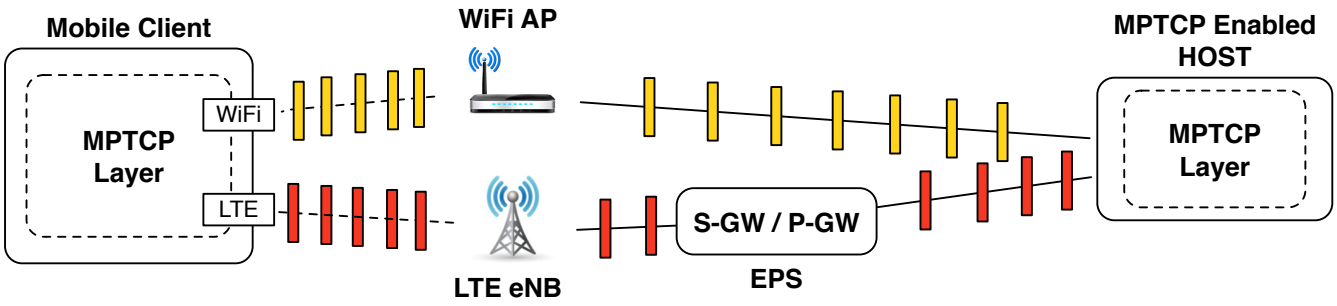

Fig. 1. Experiment Setting : MPTCP-enabled hosts

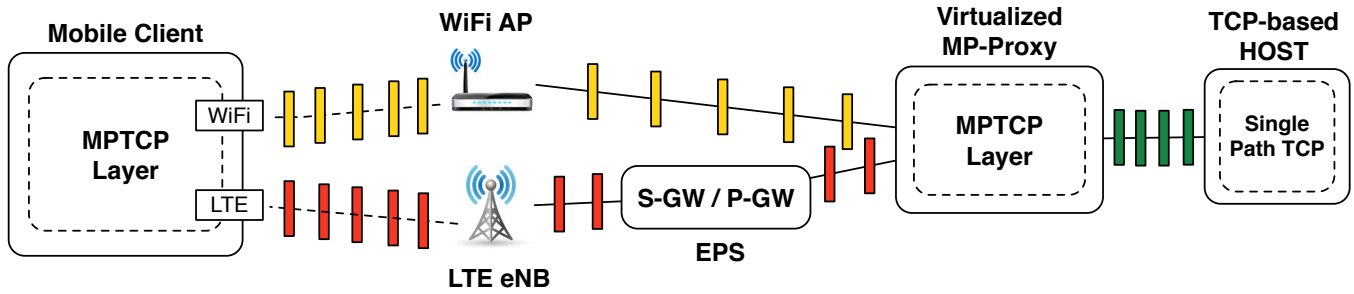

Fig. 2. Experiment Setting : MPTCP-proxy

several technical challenges and one of them is performance degradation. We also want to find out if there is any unusual behavior when NFV is working with MPTCP connection. Since not many application servers are MPTCP-enabled, usually a SOCKS proxy is deployed in between. We considered the SOCKS proxy as NFV instance and set it up as a virtualized proxy. We deployed two types of virtualized proxies, one as KVM and the other as docker, so we can compare the performance between the hypervisor-based and the container-based virtualization.

II. Background and Related works

Virtualization is a mature technology and software -defined networking (SDN) [6] and network function virtualization (NFV) will be the key players in the fifth generation (5G) mobile networks. The main concept in NFV is to decouple the network functions from the dedicated hardware working only for the given purpose. The decoupling makes commodity hardware available for the same network functions. Therefore, the capital investment costs and the operating costs for the same network functions would be reduced effectively [7].

There are roughly two types of virtualization; hypervisor-based and container-based technology. Hypervisor-based virtualization runs its own operating system (OS), isolated from the host system. Linux's Kernel-based Virtual Machine (KVM), Xen, VirtualBox, and VMware belong to this category. The hardware level virtualization makes it possible to run different OS from the host OS. Container-based virtualization runs on the host OS. It is an isolated process at OS level, thus, much lighter than the hardware virtualization. Researches have been conducted to compare the performance of different kinds of virtualization techniques. [8] compared KVM, LXC, Docker, and OSv using a variety of benchmarking applications. They measured the CPU, memory, disk I/O, and network I/O performances. KVM showed limited performance due to its hardware-based virtualization. [9] also compared KVM and Docker using real application, MySQL, to understand the overhead only caused by virtual machines and containers.

The original implementation of MPTCP is in the data center networking [10]. Since MPTCP splits a single TCP connection into a number of paths, it can send a huge amount of flow more reliably. Recently, MPTCP is regaining the attention in the cellular network. Since smart phones are already equipped with the LTE and WiFi modems, utilizing both of them with MPTCP will guarantee the increased bandwidth and reliability for mobile users. In [11], they selected eight network-heavy smartphone applications and analyzed the bandwidth usage of the WiFi and LTE on MPTCP-enabled Android phones. They showed that if the connection lasts for a short period of time, most of the data is sent over the initial interface only. The result is due to the fact that MPTCP needs time to initiate a connection. Therefore, applications based on the long connections enable MPTCP to utilize both of the available interfaces and guarantee good performances.

The most recent research for MPTCP in the cellular network is [12]. They collected user data from the MPTCP-enabled smart phones and analyzed MPTCP performance and its cross-layer interactions. Since MPTCP was originally introduced as a data center networking protocol, they claim that MPTCP would not benefit from the multipath throughput

in the mobile ecosystem, if implemented naively. Based on the measurements and analysis, they suggest a software architecture for mobile multipath communication.

To improve the overall MPTCP performance, various MPTCP congestion control algorithms have been studied and suggested [13] [14]. The MPTCP congestion control algorithm that focuses on the fairness between the single TCP and the MPTCP flow is LIA (Linked Increases Algorithm) [15]. However, as mentioned in [16] even when TCP connections are replaced by MPTCP connections, the updated MPTCP connections gain no benefits but to reduce the other TCP connections. OLIA (Opportunistic Linked Increase Algorithm) was introduced to solve this issue. BALIA (Balanced Linked Adaptation) is another MPTCP congestion algorithm which exploits a fluid model and tries to achieve a balance between TCP-friendliness, responsiveness and window oscillation [17]. LIA, OLIA, and BALIA are all standardized MPTCP congestion control algorithms by IETF (Internet Engineering Task Force).

III. Experiment Setup

As mentioned in the introduction, we try to figure out the characteristics of MPTCP network traffic in the cellular network, especially when a SOCKS proxy is relaying the MPTCP and legacy TCP traffics. We set up 3 desktops as a mobile client, an MPTCP-enabled host, and a virtualized SOCKS proxy. All three of them are running the latest MPTCP Linux kernel 4.1.38 in Ubuntu 14.04 (64-bit). The bandwidth in our LAN setting is around 100Mbps, thus the bandwidth would not limit the overall experiments. Since we wanted to measure the virtualized proxy overhead only, none of the configuration parameters in these machines are modified for better performance. The MPTCP scheduler was set as the default one which makes the packet scheduling decision based on the subflow's RTT. And the congestion control algorithm was CUBIC, which is the default Linux congestion control algorithm.

For the virtualized proxy comparison, the two different types of SOCKS proxy was set up, one as a KVM and the other one as a docker. Both proxies are also running Ubuntu 14.04 (64-bit) and they are configured to use the same resources from the host, 4-core 3.10GHz CPU, and 12GB memory. A desktop mobile client is an inevitable choice since we need to repeat the network load tests from the client side. The mobile client utilizes two wireless interfaces, LTE and WiFi, tethering via a USB cable.

For repeated wireless network measurements, software-based LTE network, the Amari LTE 100 was used. The Amari LTE 100 is a 100% software package including eNB, EPC, eMBMS gateway and IMS test server. It is widely used LTE emulation for research and commercial LTE service testbeds for its easy configuration and 3GPP compliant. A dedicated WiFi AP was used to prevent the interference between the users sharing the same WiFi channel. Due to its dedicated wireless setup, we could repeat the measurements as many as we want with the least interference and without cost issues. To test the socks proxy load, we used Apache JMeter. It is a load testing tool for various applications, mainly used for the web application tests.

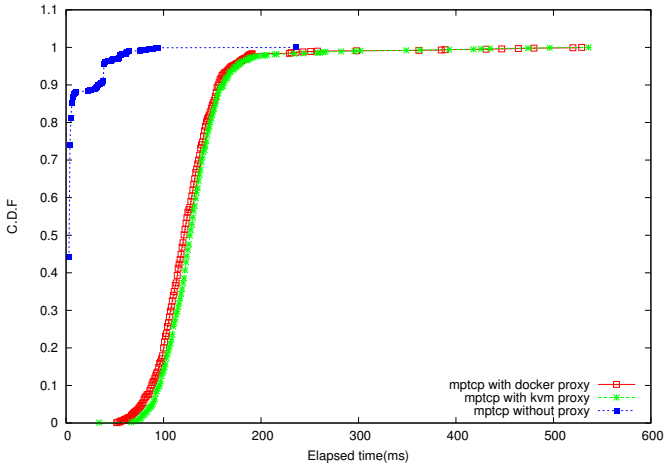

Fig. 3. A short flow comparison between MPTCP-enabled connection and Docker, KVM proxy : The 1KB request was sent sequentially and repeated for 1,000 times.

IV. Experiment Evaluation

Figure 3 shows the SOCKS proxy overhead in the cellular network. We measured the elapsed time between the short HTTP request (1KB) and the server reply. The test kept repeating for 1,000 times sequentially. The figure shows the elapsed time as CDF (cumulative distribution function). If the client can directly connect to the MPTCP-enabled server, the average elapsed time takes 8.8ms. However, if the application server is not able to utilize MPTCP protocol, the client requests should be routed to the SOCKS proxy and then relayed to the application server. Due to this proxy routing, the CDF of two virtualized proxies shifted to the right, meaning that the increased response time for using MPTCP proxy. The average elapsed time for the KVM and the docker proxy was 131.4ms and 125.5ms, respectively.

Next, we tested how well the virtualized proxies handle concurrent requests. We tested from 1,000 concurrent requests and seized the measurement at 4,000. More than 4,000 requests generated a request send error, thus the number of concurrent requests was not valid. Host CPU consumption (%) was also measured and the numerical results are listed in the table below.

	1,000	2,000	4,000
Docker	25.0%	31.9%	37.0%
KVM	26.0%	36.9%	48.9%

The table shows that the KVM proxy utilizes more CPU, and from the following figures (figure 4, 5, 6), we can find out that it is due to a longer processing time. Whereas the docker proxy consumes less CPU than the KVM, and this is also confirmed by the same figures (figure 4, 5, 6), thus docker can handle the same request faster than KVM. Figure 4, 5, 6 each shows CDF of elapsed time for concurrent requests where the elapsed time represents the time between the request and the last response.

The final test is for summing up the relation between a user request volume and the various TCP connection performances. Three different volume of TCP requests (1KB,

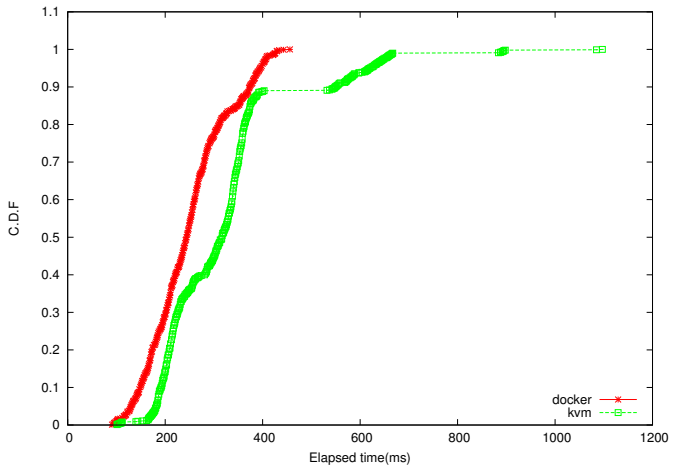

Fig. 4. CDF of elapsed time for 1,000 concurrent requests (Docker vs. KVM)

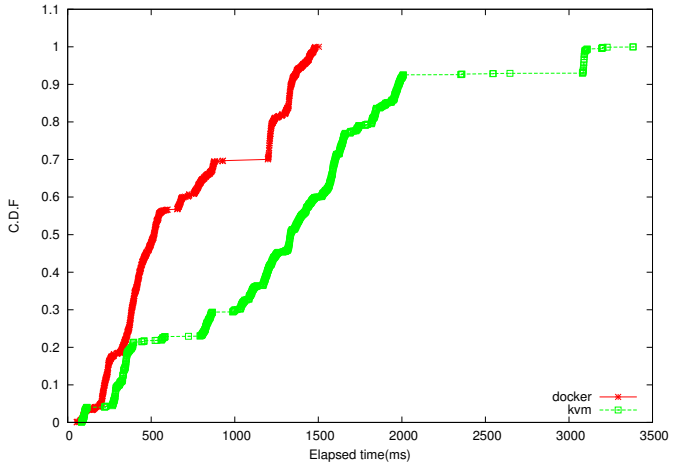

Fig. 5. CDF of elapsed time for 2,000 concurrent requests (Docker vs. KVM)

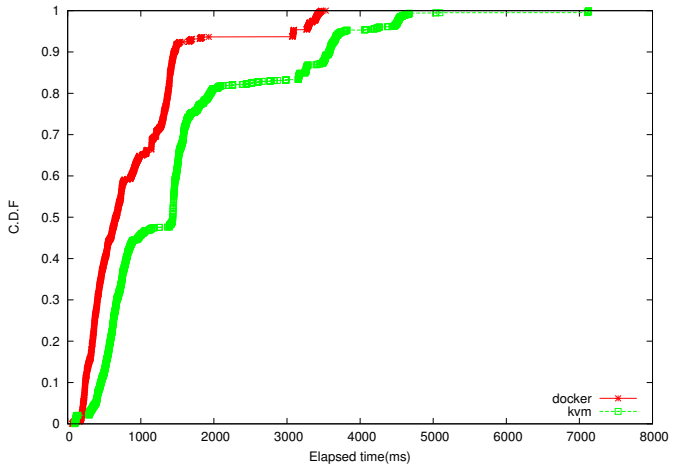

Fig. 6. CDF of elapsed time for 4,000 concurrent requests (Docker vs. KVM)

900KB, 164MB) were considered and their average throughputs are listed in the table. All requests were sent sequentially after the previous one got its response. The elapsed time denotes the time between the request and the last response.

In figure 7, the left group, marked as 'Default', shows the throughput of a short TCP request (1KB). The '900KB' in the middle is a test for the medium-sized (900KB) TCP request. And the 'HLS' on the right measured HTTP Live Streaming requests for a 164MB video segment and it represents long and a large volume of TCP flow.

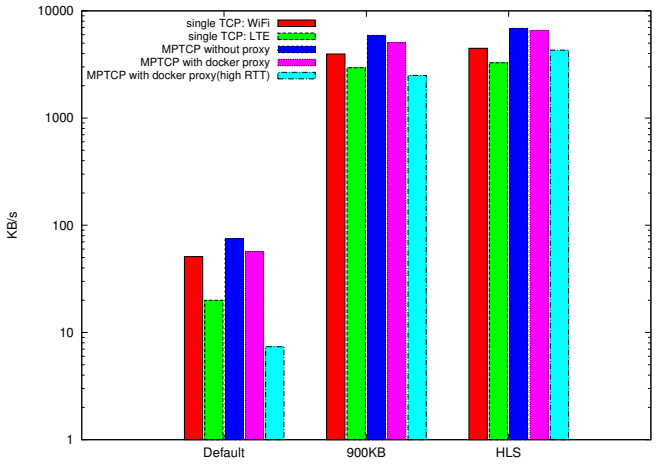

Fig. 7. Throughput comparison between various TCP connections : 'Default' represents a short TCP flow (1KB), '900KB' represents a medium TCP flow (900KB), and 'HLS' denotes a long TCP flow (164MB)

	WiFi	LTE	MPTCP	Proxy	Proxy(+100ms)
Default	51 KB/s	20 KB/s	75 KB/s	57 KB/s	7.36 KB/s
900KB	3957 KB/s	2948 KB/s	5905 KB/s	5048 KB/s	2496 KB/s
HLS	4472 KB/s	3277 KB/s	6875 KB/s	6563 KB/s	4299 KB/s

One of the things that we should check in this test is the MPTCP connection with the docker proxy throughput. There is a bit of penalty due to the proxy routing as expected. Since the experiment was set up in the LAN (Local Area Network), the latency between the proxy and the application server was way to small compared to the real cellular network settings (0.05ms). We try to mediate this short latency by imposing a delay between the proxy and the server using Linux TC command. The result is the sky-blue colored marks in each test group. The numerical values are also listed in the last column in the table.

As we apply a long latency (RTT 100ms) between the proxy and the application server, the MPTCP proxy throughput was degraded in all 3 tests and it performed even worse than the single TCP connections. This implies that when we set up a virtualized SOCKS proxy in the real cellular network, we should consider the latency between the two end points. Since NFV's performance degradation comes from its various configurable network parameters, the latency would be one of the factors that determines the performance in NFV deployment.

When we applied a medium latency (RTT 50ms) between the proxy and the application server, it only deteriorated short (Default) and medium-sized (900KB) TCP flows. The throughput, in this case, was 13 KB/s, 2820 KB/s, respectively. HLS throughput was 5091 KB/s, worse than the MPTCP-enabled connection, but sill performs better than the single TCP connections. This also confirms our expectation that

MPTCP performs well in long and large volume of TCP connections.

In short, MPTCP can achieve a throughput enhancement in cellular network, even when the connections are routing to and from the SOCKS proxy. However, the network administrators deploying the virtualized SOCKS proxy to enable MPTCP connections should consider the latency between the proxy server and the application server, for the optimal performance. If the latency between the two entities are not in the proper boundaries, it does not benefit from the multipath connections and even performs worse than the single path connections. This is especially critical when the sudden surge of user requests make the existing virtualized proxy unable to handle the traffic alone and other virtualized proxies need to be launched on demand. Properly launching a new virtualized proxy in the best path and selecting the already functioning proxies in the network based on the dynamic user request pattern would be an interesting and important job in our future work.

V. Conclusion

The data traffic in cellular network is growing massively each year and even more so in the next 5G mobile network. Many kinds of researches have been conducted to relieve the traffic volume in the mobile network. Among other things, network function virtualization (NFV) is the key technology that enables a flexible traffic handling. NFV decouples the network function from the purpose-built hardware. Separating the network function from the specific hardware makes commodity machines handle network functions, hence, lowers the capital expenditures (CAPEX) and the operating expenditure(OPEX). To improve the data transfer efficiency, MPTCP has been reintroduced in the cellular network. Unlike the legacy TCP which uses only one TCP connection at a time, MPTCP exploits multiple TCP connections all at the same time. Thus, the overall throughput is improved. Due to the lack of MPTCP-enabled servers in the network, SOCKS proxy is usually deployed between the client and the server to make the MPTCP connection available.

In this paper, we considered a virtualized proxy as an instance of NFV and analyzed its behavior in emulated cellular network. We also wanted to analyze the behavior of a virtualized proxy in MPTCP connections. Two of the most representative virtualization techniques, KVM as a hypervisor-based and docker as a container-based virtualization, were compared. As the results show, container-based virtualization can handle the same traffic with less resource consumption. The main finding in this experiment is that when NFV is deployed in the network, configurable network parameters should be considered thoroughly to get the exact performance value. In our experiment, the key parameter was the latency between the server and the proxy, and this value significantly determined the network performance.

Acknowledgment

This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ICT Consilience Creative Program (IITP-R0346-16-1008) supervised by the IITP(Institute for Information & communications Technology Promotion).

This work was supported by ICT R&D program of MSIP/IITP. [R7124-16-0004, Development of Intelligent Interaction Technology Based on Context Awareness and Human Intention Understanding]

References

[1] D. Webster, "Cisco visual networking index (vni)," *Global Forecast Update*, p. 6, 2017.

[2] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "Nfv: state of the art, challenges, and implementation in next generation mobile networks (vepc)," *IEEE Network*, vol. 28, no. 6, pp. 18–26, 2014.

[3] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[4] O. Bonaventure, M. Handley, C. Raiciu *et al.*, "An overview of multipath tcp," *USENIX login*, vol. 37, no. 5, 2012.

[5] C. Paasch and O. Bonaventure, "Multipath tcp," *Communications of the ACM*, vol. 57, no. 4, pp. 51–57, 2014.

[6] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[7] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[8] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. lightweight virtualization: a performance comparison," in *Cloud Engineering (IC2E), 2015 IEEE International Conference on*. IEEE, 2015, pp. 386–393.

[9] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 171–172.

[10] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath tcp," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 266–277.

[11] Q. De Coninck, M. Baerts, B. Hesmans, and O. Bonaventure, "Observing real smartphone applications over multipath tcp," *IEEE Communications Magazine*, vol. 54, no. 3, pp. 88–93, 2016.

[12] A. Nikravesh, Y. Guo, F. Qian, Z. M. Mao, and S. Sen, "An in-depth understanding of multipath tcp on mobile devices: measurement and system design," in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*. ACM, 2016, pp. 189–201.

[13] S. Barré, C. Paasch, and O. Bonaventure, "Multipath tcp: from theory to practice," *NETWORKING 2011*, pp. 444–457, 2011.

[14] C. Xu, J. Zhao, and G.-M. Muntean, "Congestion control design for multipath transport protocols: a survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2948–2969, 2016.

[15] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath tcp." in *NSDI*, vol. 11, 2011, pp. 8–8.

[16] R. Khalili, N. G. Gast, M. Popovic, U. Upadhyay, and J.-Y. Le Boudec, "Non-pareto optimality of mptcp: Performance issues and a possible solution," Tech. Rep., 2012.

[17] Q. Peng, A. Walid, J. Hwang, and S. H. Low, "Multipath tcp: Analysis, design, and implementation," *IEEE/ACM Transactions on Networking*, vol. 24, no. 1, pp. 596–609, 2016.