Multipath TCP Proxy: Unshackling Network Nodes from Today's End-to-End Connection Principle

Christos Pollalis(✉), Paris Charalampou, and Efstathios Sykas

Computer Networks Laboratory, School of Electrical and Computer Engineering, National Technical University of Athens, Zografou, Greece
pollalisbelg@yahoo.gr, {pchara,sykas}@cn.ntua.gr

Abstract. Nowadays, mobile devices are equipped with multiple radio interfaces, data centers provide redundant routing paths, and multihoming is the new tendency in existing, extensive server farms. Meanwhile, the unending growth rate of Internet traffic generation raises difficulties in meeting end user demands regarding bandwidth availability and Quality of Service standards, while TCP itself persists as a single-path transport protocol. Multipath TCP, as a set of extensions to legacy TCP, permits the simultaneous utilization of the available interfaces on a multihomed host, while preserving the standard TCP socket API. Consequently, smart terminals possess the distinct capability of leveraging path diversity in order to provide robust data transfers and enhance the overall connection performance. However, the implementation of Multipath TCP is still at a premature state. Ergo, we propose and evaluate a Multipath TCP Proxy as a mechanism towards the incremental adaptation of the extended protocol by service delivery platforms. Particularly, we examine the use of an HTTP Proxy as a protocol converter that will allow MPTCP-enabled clients to benefit from Multipath TCP even when communicating with legacy servers.

Keywords: Multipath TCP · Protocol conversion · Congestion control · Policy routing · Bandwidth throttling

1 Introduction

Computer networks are continuously evolving. When the TCP/IP stack was originally designed, hosts were equipped with a single network interface. On modern epoch terminals, however, multiple wired and/or wireless interfaces are available. Despite its age, the Transmission Control Protocol (TCP) remains the dominant transport mechanism on the Internet in an era when smart devices crave to exploit their numerous, usually underutilized, interfaces and harvest the available network resources.

A number of protocols have been proposed to provide multihoming functionalities to end user devices without the use of legacy routing protocols, such as BGP. Two main categories can be identified: (a) host based solutions – where modifications should be implemented on user handsets and devices, and (b) router based solutions – where end devices are left unchanged and new protocol implementations are fulfilled on router

L. Mamatas et al. (Eds.): WWIC 2016, LNCS 9674, pp. 55–65, 2016.
DOI: 10.1007/978-3-319-33936-8_5

networking stack. Since multihoming and simultaneous usage of network interfaces should be agnostic of the basic network connectivity, solutions that are based on the introduction of middle-boxes seem more appropriate in multivendor and multi-ISPs environments. In this paper we try to address the issues imposed from higher bandwidth requirements, especially on mobile nodes.

The first and more widely adopted approach is the SCTP (Stream Control Transmission Protocol). SCTP is a message-oriented protocol like UDP that requires SCTP associations over diverse network paths. SCTP is based on delivering chunks of information from different network interfaces, transported on different paths inside an IP network and finally multiplexed on host level. Even though SCTP is adopted from the vast majority of Operating Systems and is already used in the telecom industry as a replacement to the SS7 signaling protocol, SCTP cannot be consider as the optimal way forward for end-devices. The main drawback of SCTP implementations [1] is the high complexity and the negative effect on computing resources, which are relatively expensive on mobile nodes. Furthermore, existing experimental analysis of SCTP in heterogeneous networks demonstrates poor performance in high packet and bandwidth rates.

Another approach is the Loc/ID (Locator and Identity) separation method. In this context, the nodes have both an identity that uniquely distinguishes the end host and an associated locator that describes the network connectivity structure. This concept separates the two name spaces (identities and addresses), which are combined in traditional IP networks. In Loc/ID separation architectures the IDs are used for end to end communications, whereas the locators are assigned to different network interfaces. The main representative in this category is the LISP implementation for mobile nodes, described as Lisp-MN [2]. LISP is not yet implemented on large scale networks and requires significant changes in the infrastructure of Internet Service Providers (ISPs). Furthermore, it lacks the functionality to parallel use multiple interfaces on the same dialogue maximizing throughput and user experience.

A third approach is Multipath TCP (MPTCP) [3]. Multipath TCP enables an Internet device to efficiently use its multiple interfaces by installing a set of extensions on top of traditional TCP, which permit the establishment of additional TCP subflows under the umbrella of a single MPTCP session. Consequently, supplementary capabilities are introduced to the protocol's core functionality empowering features such as enhanced robustness against network malfunction, bandwidth aggregation, as well as dynamic data offloading. Moreover, cellular networks leveraging MP-TCP can enhance Quality of Experience (QoE) by providing robustness towards data communication technology availability and cell-to-cell service degradation. At the same time, MPTCP preserves the standard socket API and maintains backward compatibility with Legacy TCP at both the network and application levels. This fallback procedure serves also as a fail-safe mechanism, since MPTCP deployment is at a premature state and/or middle-boxes may strip down the MPTCP signaling options.

In an attempt to achieve effective "resource pooling" [4] and aggregate the bandwidth available, MPTCP implements a congestion control mechanism coupled at the MPTCP-level of the transport layer [5], consequently creating a coalescence of links effectively behaving as a shared channel of higher capacity. As a result, hosts are able to ceaselessly monitor the dynamic state of the network routes involved in data transfers and shift

Internet traffic from degraded paths to more efficient ones according to the available bandwidth per utilized link. On the other hand, in order to ensure continuous packet transmissions, unaffected by individual malfunctioning connections or link-related congestion phenomena, a unique receive window shared across all the TCP subflows is implemented at the recipient. Additionally, a buffer of adequate space is required [6], so as to assure in-order delivery of the information segments to the application layer. The buffering space should accommodate an increased amount of packets, compared to legacy TCP systems, in an effort to cope with the head-of-line blocking caused by the dissimilar transmission delays of the diverse routing paths.

Nonetheless, the deployment of MPTCP solutions is still lagging behind. Currently, a host can benefit from Multipath TCP only if its peer possesses MPTCP-capability as well. This requirement raises barriers in the deployment of MPTCP, since service delivery platforms dawdle as far as protocol installation is concerned [7]. While the multipath functionality can be introduced quite seamlessly to user equipment due to the frequent OS updates of personal computers and smart devices, such changes are more complex to conduct inside an Internet Service Provider's core infrastructure, especially on application servers handling a myriad of requests on a constant basis. The inset of an intermediate node as a protocol converter, aiming at the creation of a split TCP-MPTCP connection between the MPTCP-enabled clients and MPTCP-unaware systems, as illustrated in Fig. 1, may serve as a stepping stone towards the gradual incorporation of MPTCP, partially sidestepping initial compatibility issues and consequently user QoE degradation.

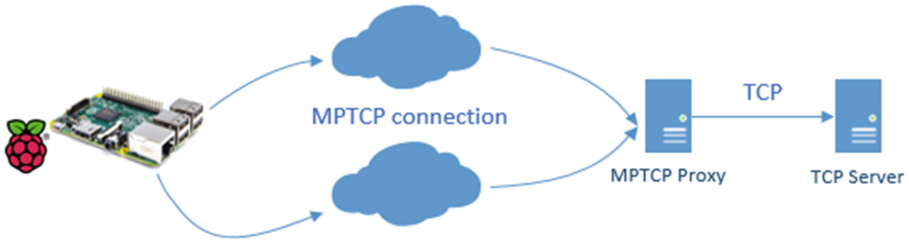

Fig. 1. Creation of the split TCP-MPTCP connection, after insertion of the MPTCP Proxy in between the communication ends.

2 Multipath TCP Proxy

Deploying proxies as a middle-box is often not considered best practice by researchers [8] regarding traditional network architectures and datacenter topologies [9]. Such deployments reduce the overall service availability, downgrade service performance by creating bandwidth bottlenecks, and increase the operational costs. Even though middle-boxes impose all these drawbacks, service providers are still keen on deploying them as a way to enforce network policies for charging or traffic steering purposes. Furthermore, installing middle-boxes is relatively common in order to benefit from new technologies without waiting to be adopted by terminals and network devices [10, 11].

Our proposal aims at the introduction of a protocol converter in the form of a Multipath TCP Proxy in order to provide multipath functionality on behalf of traditional TCP systems, bypassing compulsory protocol adaptation in the ISP core infrastructure [12]. As a result, MPTCP-capable hosts will be able to effectively take advantage of their enhanced, multipath features, while the established path redundancy will give network operators the opportunity to more efficiently distribute traffic load and mitigate bottleneck occurrences.

In addition, our MPTCP Proxy is enhanced with a bandwidth throttling [13] mechanism as a reactive means for network traffic regulation and congestion minimization. The bandwidth manipulation algorithm, which runs inside the MPTCP Proxy, permits the application of load control policies, compatible with the specifications of Multipath TCP with regard to traffic engineering and billing purposes, while aiming at the satisfaction of Quality of Service (QoS) standards. The mechanism utilizes a Scapy [14] sniffer, which is responsible for monitoring the characteristics of incoming connections by parsing the options field of the TCP header. As soon as an MPTCP connection is established consisting of at least two active TCP subflows, the MPTCP Proxy throttles the desired TCP connection based on pre-determined criteria. The eventual traffic redirection is feasible due to the insertion of packet mangling rules on top of the Netfilter Framework [15].

The overall scheme allows operators to get in the middle of the process of outgoing data scheduling by virtually sub-dividing a packet queue into multiple ones and reconfiguring them using classful queuing disciplines. Each independent queue is responsible for the management and forwarding of packets of a predetermined destination, allowing network operators to manipulate Internet traffic by assigning the most fitting packet queue to specific TCP subflows based on routing policy criteria. The concurrent utilization of diverse wired and/or radio access technologies in conjunction with the application of capacity limitation techniques creates the illusion of less suitable routing paths, ergo resulting in a force-steering of the forwarded traffic. As a result, it becomes possible to re-allocate network load and alleviate bottleneck phenomena, while harnessing the excess, available bandwidth, as well as offer MPTCP-enabled clients enhanced QoE due to the traversal of less congested areas.

3 Experimental Evaluation

We have evaluated our proposal in an experimental environment similar to the aforementioned Fig. 1, consisting of a Raspberry Pi Model B running Debian Wheezy on top of a custom, MPTCP-compatible 3.12.35 + Linux kernel, an MPTCP Proxy with kernel version 3.14.0 and Squid Proxy installed, equipped with the stable release v0.89.2 of the extended protocol, and a MPTCP-unaware HTTP server with Apache2 running on port 80. During our experimental scenarios we have disabled Squid's caching mechanism in order to measure pure forwarding performance. In particular, we examined the impact of memory space allotment with regards to the MPTCP attainable bandwidth, the responsiveness of our bandwidth throttling mechanism, as well as the packet forwarding delay inserted due to the protocol conversion process.

3.1 Baseline Scenario

In our initial experimental scenario, we depict the performance accomplished by Legacy TCP during direct communication between our Raspberry Pi and the MPTCP-unaware HTTP server. Since the MPTCP Proxy is not explicitly introduced to our client-side device, the negotiation of multipath functionality fails between the end hosts. As the MPTCP options embedded inside the TCP header cannot be interpreted by the legacy server-side system, multipath support is not advertised by the HTTP server causing the RPi to fall back to regular TCP and only establish a single – traditional – TCP connection. Figure 2 illustrates the achieved throughput for two types of TCP sessions, a wired and a wireless connection, respectively.

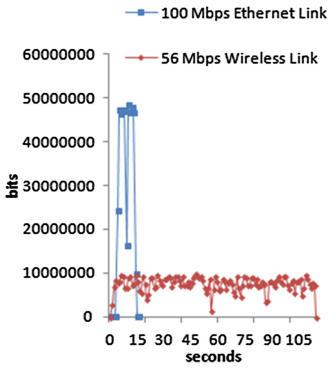

Fig. 2. Legacy TCP performance

3.2 Dynamic Load Balancing

Since in our current setup the MPTCP-enabled Raspberry Pi is still unable to simultaneously utilize both its network interfaces, we inset our MPTCP Proxy as an intermediate node in order to provide multipath functionality on behalf of the legacy TCP server. The MPTCP session terminates on the client-side of the protocol converter, while a regular TCP connection is initiated on the server-side towards the desired destination. Figure 3 depicts the aggregated throughput achieved under MPTCP's modified congestion control algorithm.

As the Ethernet interface offers a higher link capacity alongside a smaller Round Trip Time (RTT) value compared to the wireless connection, the respective subflow's congestion window tends to inflate faster, ergo allowing for the largest percentage of packets to be forwarded via the wired routing path. On the other hand side, the wireless link remains almost fully underutilized, since MPTCP perceives it as an inferior alternative for load balancing or data offloading in the absence of congestion on the wired connection. Therefore, the observations above come to confirm MPTCP's capability to strip data across the most efficient of the available paths, while offering no less capacity compared to a legacy TCP session.

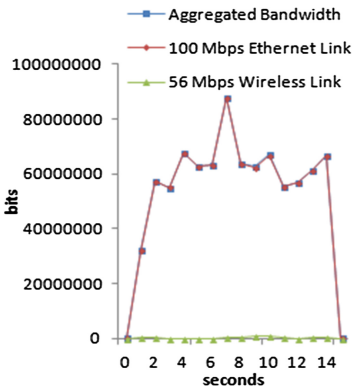

Fig. 3. Multipath TCP performance

3.3 TCP Buffer Size Impact on MPTCP Performance

TCP Memory Allocation. One of the major challenges arising as a consequence of the deployment of MPTCP solutions is the allocation of adequate buffering space in order to cope with the increased demands regarding packet reordering at the receiver [16]. The disparate propagation delays along the utilized network links create discontinuities in the re-assembled information stream, since TCP segments often reach their final destination out of sequence. In such case, the head-of-line blocking effect becomes quite noticeable, temporarily interrupting the final in-order delivery to the application layer.

Figure 4 illustrates the performance of MPTCP corresponding to two distinct receive buffer configurations. According to MPTCP specifications, the required accommodation space for TCP segments amounts to a maximum of 0.48 MB. As clearly shown in Fig. 4a, adopting a lower bound results in data loss at the receiver, leading to unavoidable packet retransmissions and an overall service degradation.

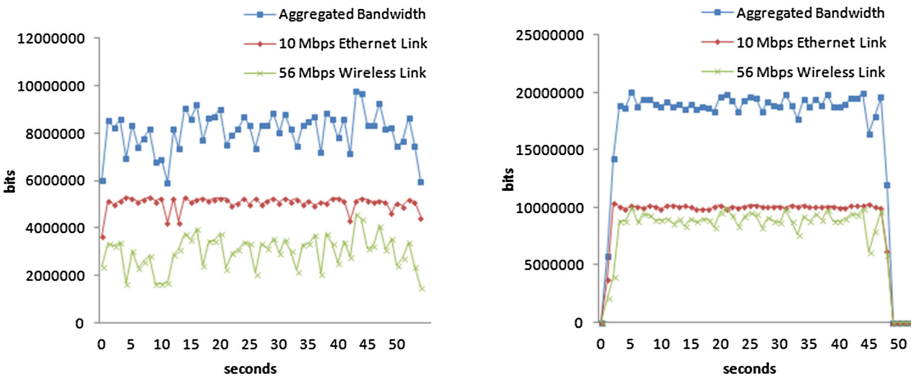

Fig. 4. MPTCP performance based on different receive buffer configurations. Max memory allocation set at (a) 0.18 MB, (b) 0.72 MB

On the other hand, Fig. 4b depicts a much more efficient data transfer. Besides the naturally stable behavior of the wired access link, the wireless interface achieves a satisfactory throughput rate, as well, especially if we additionally consider the potential randomness of the wireless transmission environment resulting in elevated propagation delays. The TCP memory allocation is adequate enough to sustain a constant delivery rate of segments to the application layer, minimizing the impact of head-of-line blocking. In general, MPTCP paths of commensurate delays tend to decrease receiver memory consumption even close to zero, since packets routed over different paths arrive timely at their destination without causing extensive gaps in the packet sequence at the recipient.

Maximum Transmitted Unit Size Adjustment. Similar considerations apply in case of Maximum Transmitted Unit (MTU) size variations. Intuitively, a large MTU size leads to higher data rates, since the number of interrupts, which need to be processed by the receive system, diminishes. On the other hand, a small MTU size compels the recipient to handle an increased amount of TCP segments provoking packet dropping, as depicted in Fig. 5, and overall destabilizing the MPTCP session compared to Fig. 4b. Default MTUs carry enough payload so as to cause no impairment during the data transmission, while gradual size reduction leads more swiftly to service degradation compared to legacy TCP systems. This more abrupt impact derives from a combination of reasons, including the increased number of packets that need to be handled and their dissimilar transfer delays. Since more packets are accommodated inside the receive buffer, the re-ordering process becomes more intense due to the different arrival periods of consequent segments. Practically, in case of MTU size variations, a proportionate TCP memory extension should be allocated in order to cope with buffering demands and avoid service quality downgrading.

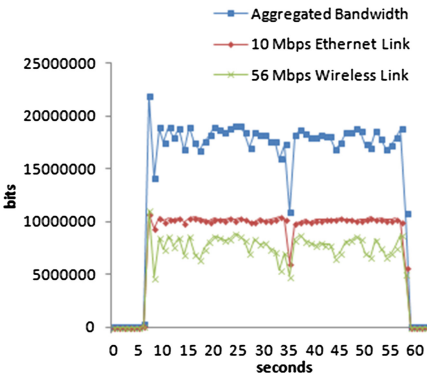

Fig. 5. MPTCP performance degradation as a result of decreasing MTU size down to 1000 bytes, while keeping buffer size at 0.72 MB

3.4 Bandwidth Throttling Mechanism

While Multipath TCP is capable of pooling the accessible network resources per interface, spreading load over a wider network, as well as achieving dynamic data offloading,

the parallel application of bandwidth throttling allows for further traffic manipulation. Administrators are capable of further managing the MPTCP sessions and force-shifting data across eclectic routing paths, while offering the multipath connection no less capacity than a single-path TCP session. This is all feasible via the prioritization of the TCP subflows by applying the desired levels of bandwidth restriction per available network interface, ergo outsmarting MPTCP into perceiving select TCP connections as less efficient paths for packet routing, even though their potential link capacity indicates otherwise. As a result, packet flows can be redirected away from congested areas, while concurrently alleviating bottlenecks.

Figure 6a illustrates the effect of the bandwidth restriction introduced on the Ethernet interface of our Raspberry Pi. The process completion delay alongside the moderate utilization of the available resources tends to cause a miniature impact on delivery time. However, even if the queuing disciplines are not configured timely in case of transfers of minimal size, a tolerable scenario for network-friendly communications in terms of congestion, the desired control policies will be already installed in order to support future multipath transmissions without the original process overhead, as depicted in Fig. 6b.

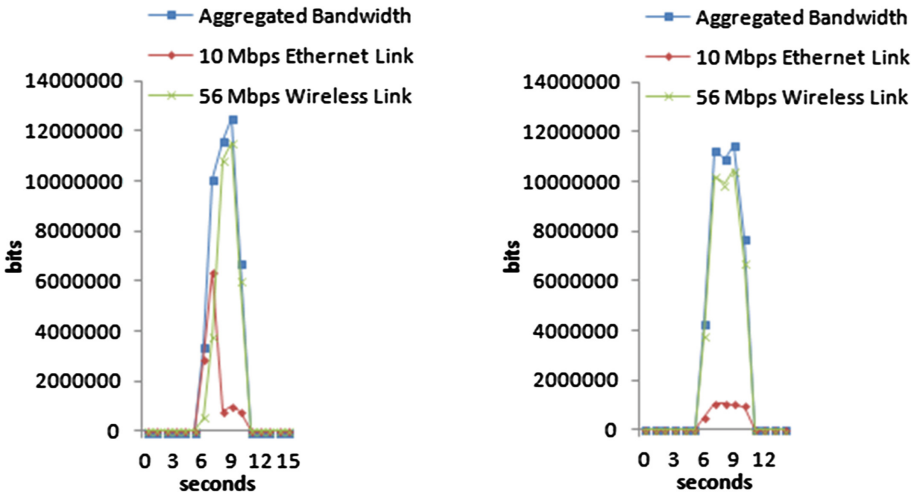

Fig. 6. Application of bandwidth throttling on the wired network interface (a) initial 5 MB file transfer, (b) 5 MB file re-transmitted

Finally, Fig. 7 exhibits the bigger picture of leveraging MPTCP with our bandwidth throttling mechanism, as well as the generic behavior of the involved TCP subflows. As soon as the control policies have been established, the MPTCP congestion control algorithm tends to offload data to the wireless connection, which manages to achieve a higher throughput rate compared to throttling-free transmissions (Fig. 4b). Moreover, the overall connection performance becomes slightly inferior to regular data transfers (Fig. 4b) due to the extreme underutilization of the Ethernet link, a differentiation that can be undone by the proper reconfiguration of the capacity limitation levels.

Nevertheless, the bottom line is that MPTCP still outperforms legacy TCP, as well as meets its design goals.

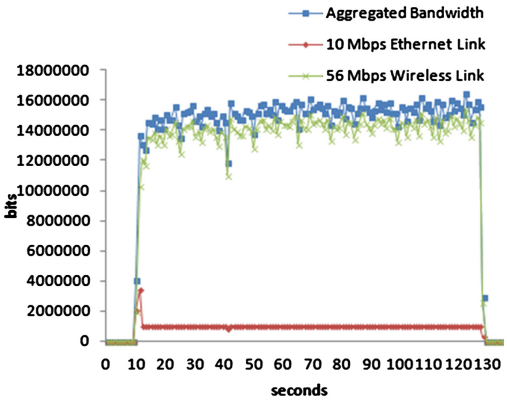

Fig. 7. Application of bandwidth throttling on the wired network interface during the transfer of a 200 MB file

3.5 Protocol Conversion Delay

Table 1 presents the chain-performance of the MPTCP Proxy regarding protocol conversion and packet forwarding. The measurements were attained via the insertion of IPtables mangling rules on top of the Netfilter Framework, which targeted corresponding packets of specific characteristics on both communication ends. As a matter of fact, our middle-box tends to synchronize one session at a time. Therefore, the MPTCP-to-TCP SYN segment conversion delay is equivalent to the time between the client-side connection establishment and the server-side 3-way-handshake initialization. Moreover, data segments are reconstructed internally, meaning they are converted and forwarded, after they have been received by the application layer. Of course, such interaction between low-level kernel and user space tends to introduce additional conversion delay, as packets are processed along the entire TCP/IP stack. Furthermore, since the system is actually hosted inside an ESXi hypervisor, the server is also running on constrained memory resources and computation power, ergo its proportionate underperformance. Undoubtedly, an application-level solution cannot ensure effective service delivery in real world networks compared to – preferably application agnostic – highly performing kernel module implementations [17]. Such solutions can provide efficient protocol conversion by minimizing unnecessary memory allocations and costly `read()` – `write()` system calls.

Table 1. Conversion time within the MPTCP Proxy between the MPTCP and TCP connections

	SYN	Data
Time (sec.)	0.0078341 ± 0.0006805	0.0153311 ± 0.0080645

4 Conclusion

The traditional TCP protocol obeys rigorously to the end-to-end connection principle, thus deterring smart terminals from efficiently exploiting their numerous network access interfaces. While TCP is upper bounded by the available capacity of the bottleneck link, Multipath TCP permits the concurrent distribution of traffic load over a wider network. As a consequence, MPTCP enhances user experience by providing improved performance alongside redundancy, as well as mitigates congestion via dynamic data offloading between the available network links.

Since MPTCP capability still remains to be implemented on the server side, the deployment of a MPTCP Proxy at the ISP infrastructure can provide multipath functionality on behalf of legacy TCP systems, as well as allow MPTCP-enabled clients to achieve effective "resource pooling", without the requirement for service providers to undergo major changes inside their core network. However, technical and economic concerns emerge regarding protocol adaptation, such as increased buffer space demands, which may lead to infrastructure upgrades and more expensive implementations; re-evaluation of existing mechanisms, such as TCP's re-ordering algorithm in order to effectively handle re-ordering events and lessen head-of-line blocking phenomena, as well as revision of today's routing policies in order to cope with MPTCP path utilization.

Thereupon, we emphasized the importance of memory allocation sufficiency and how inadequate buffer space can lead to performance degradation. In addition, we introduced a bandwidth throttling technique as a means to apply traffic engineering schemas compatible with MPTCP path utilization without downgrading service quality. Finally, we underlined the importance of application-agnostic, kernel module implementations in order to efficiently cope with real world networks' demands by minimizing superfluous system processes and overall protocol conversion overhead.

References

1. Dainotti, A., Loreto, S., Pescape, A., Ventrem, G.: SCTP performance evaluation over heterogeneous networks. Special Issue: Perform. Anal. Enhancements Wirel. Netw. **19**(8), 1207–1218 (2007). doi:10.1002/cpe.1159
2. Menth, M., Klein, D., Hartmann, M.: Improvements to LISP mobile node. In: 22nd International Teletraffic Congress (ITC22), Amsterdam, The Netherlands, September 2010
3. ICTEAM: MultiPath TCP – Linux Kernel Implementation. http://www.multipath-tcp.org/. Accessed Oct 2014
4. Wischik, D., Handley, M., Braun, M.B.: The resource pooling principle. ACM SIGCOMM **38**(5), 47–52 (2008). doi:10.1145/1452335.1452342
5. Wischik, D., Raiciu, C., Greenhalgh, A., Handley, M.: Design, implementation and evaluation of congestion control for multipath TCP. In: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, Lombard, IL, USA, pp. 99–112, April 2011
6. Raiciu, C., Paasch, C., Barre, S., Ford, A., Honda, M., Duchene, F., Bonaventure, O., Handley, M.: How hard can it be? Designing and implementing a deployable multipath TCP. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, San Jose, CA, USA, pp. 29–42, April 2012

7. Mehani, O., Holz, R., Ferlin, S., Boreli, R.: An early look at multipath TCP deployment in the wild. In: Proceedings of the 6th International Workshop on Hot Topics in Planet-Scale Measurement, Paris, France, pp. 7–12, September 2015. doi:10.1145/2798087.2798088
8. Sherry, J., Hasan, S., Scott, C., Krishnamurthy, A., Ratnasamy, S., Sekas, V.: Making middleboxes someone else's problem: network processing as a cloud service. In: Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Helsinki, Finland, pp. 13–24, August 2012. doi:10.1145/2342356.2342359
9. Potharaju, V, Jain, N.: Demystifying the dark side of the middle: a field study of middlebox failures in datacenters. In: Proceedings of the Conference on Internet Measurement Conference, Barcelona, Spain, pp. 9–22, October 2013. doi:10.1145/2504730.2504737
10. Yap, K.-K., Huang, T.-Y., Kobayashi, M., Yiakoumis, Y., McKeown, N., Katti, S., Parulkar, G.: Making use of all the networks around us: a case study in android. In: Proceedings of the ACM SIGCOMM Workshop on Cellular Networks: Operations, Challenges, and Future Design, Helsinki, Finland, pp. 19–24, August 2012. doi:10.1145/2342468.2342474
11. Fayazbakhsh, S.K., Chiang, L., Sekar, V., Yu, M., Mogul, J.C.: Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags. In: Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, Seattle, WA, USA, pp. 533–546, April 2014
12. Pollalis, C., Charalampou, P., Sykas, E.: HTTP data offloading using multipath TCP proxy. In: Proceedings of the 15th IEEE Conference on Computer and Information Technology, Liverpool, UK, pp. 777–782, October 2015. doi:10.1109/CIT/IUCC/DASC/PICOM.2015.114
13. Wikipedia: Bandwidth Throttling. https://en.wikipedia.org/wiki/Bandwidth_throttling. Accessed Mar 2015
14. SECDEV: Scapy. http://www.secdev.org/projects/scapy/. Accessed Mar 2015
15. Netfilter Core Team: netfilter: firewalling, NAT, and packet mangling for Linux. http://www.netfilter.org/. Accessed Mar 2015
16. Barré, S., Paasch, C., Bonaventure, O.: MultiPath TCP: from theory to practice. In: Domingo-Pascual, J., Manzoni, P., Palazzo, S., Pont, A., Scoglio, C. (eds.) NETWORKING 2011, Part I. LNCS, vol. 6640, pp. 444–457. Springer, Heidelberg (2011)
17. Detal, G., Paasch, C., Bonaventure, O.: Multipath in the middle(Box). In: Proceedings of the 2013 Workshop on Hot Topics in Middleboxes and Network Function Virtualization, Santa Barbara, CA, USA, pp. 1–6, December 2013. doi:10.1145/2535828.2535829