Multipath in the Middle(Box)

Gregory Detal, Christoph Paasch and Olivier Bonaventure
ICTEAM, Université catholique de Louvain
Louvain-la-Neuve – Belgium
firstname.name@uclouvain.be

ABSTRACT

Multipath TCP (MPTCP) is a major modification to TCP that enables a single transport connection to use multiple paths. Smartphones can benefit from MPTCP by using both WiFi and 3G/4G interfaces for their data-traffic, potentially improving the performance and allowing mobility through vertical handover. However, MPTCP requires a modification of the end hosts, thus suffers from the chicken-and-egg deployment problem. A global deployment of MPTCP is therefore expected to take years. To increase the incentives for clients and servers to upgrade their system, we propose MiMBox an efficient protocol converter that can translate MPTCP into TCP and vice versa to provide multipath benefits to early adopters of MPTCP.

MiMBox is application agnostic and can be used transparently or explicitly. Moreover, a close attention was paid to the implementation's design to achieve good forwarding performance. MiMBox is implemented entirely in the Linux kernel so that it is able to more easily circumvent the bottlenecks of a user-space implementation. Measurements show that we always outperform user-space solutions and that the performance is close to plain IP packet forwarding.

Categories and Subject Descriptors

C.2.5 [**Local and Wide-Area Networks**]: Internet (e.g., TCP/IP)

Keywords

Deployment; Multipath TCP

1. INTRODUCTION

The *Transmission Control Protocol* (TCP) exists since 1981, the date of the publication of RFC 793. Despite its age, TCP is still the dominant transport protocol on the Internet. More than 95% of the Internet traffic relies on TCP [1]. During the last thirty years TCP evolved significantly. In the late 80's the congestion collapse lead to the development of the congestion control scheme [2]. This required only small changes to the TCP implementation but no protocol changes. As the bandwidth requirements grew, the limited TCP window size became a clear bottleneck. This problem was solved by the TCP large window extension [3]. Selective acknowledgements were also introduced as an extension to TCP. However, measurements show that deploying a new TCP option can take up to a decade [4]. This is because once standardized, an extension needs to be adopted and implemented by operating system vendors and supported by middleboxes such as firewalls [5].

Regular TCP connections are bound to the IP addresses that were used during connection establishment. This implies that a change in the hosts' IP address (e.g. in the case of a mobile host) results in a shutdown of all established TCP connections. MPTCP [6] addresses this problem by allowing the utilization of several paths for a single connection. In practice, these paths could be a WiFi and a 3G interface on a smartphone, two 10 Gbps interfaces on a server or an IPv4 and IPv6 address on a laptop.

On today's Internet, smartphones have a motivation for using MPTCP as this would allow them to efficiently exploit their 3G and WiFi interfaces and provide mobility [7]. However, for this, MPTCP must be supported on both the smartphones and the servers. Although the designers of MPTCP took great care of avoiding interfering with various types of middleboxes [5, 6], it is still expected that the its deployment will take several years. It is also expected that clients will support MPTCP before servers. Indeed, *Apple Inc.* recently enabled MPTCP for a specific application in which they control the server-side [8].

In this paper, we propose the utilization of protocol converters, that we call *Multipath in the Middle Box* (MiMBox), to allow early adopters to benefit from MPTCP during its deployment. MiMBox supports both MPTCP and TCP and converts the MPTCP connections used by clients into regular TCP connections to allow clients to benefit from MPTCP when communicating with legacy servers. Economic studies show that such converters can play an important role in the deployment of a new protocol [9]. MiMBox can be placed in operator networks or on commodity servers in the cloud (e.g. as Network Function Virtualization [10], etc.).

This paper is organized as follows. First, we present the design of MiMBox. We then describe how implementing it inside the Linux kernel allows to achieve high performance. We then present a thorough performance evaluation and show that it outperforms existing proxies. Finally, before concluding, we discuss related work.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
HotMiddlebox'13, December 9, 2013, Santa Barbara, CA, USA.
Copyright 2013 ACM 978-1-4503-2574-5/13/12 ...$15.00.
http://dx.doi.org/10.1145/2535828.2535829.

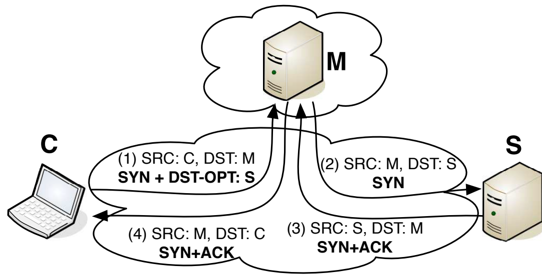

Figure 1: Explicit redirection of connection establishment through a MiMBox using the Dst_Opt TCP option.

2. DESIGN

Deploying a new transport protocol is hard and often referred as the chicken-and-egg problem. MPTCP, as every new protocol, suffers from this problem. Even by being backward compatible with regular TCP, neither servers nor clients have incentives to deploy it when the other end does not support it. To solve this adoption problem, we propose to deploy MiMBoxes that transparently convert MPTCP from MPTCP-enabled clients to regular TCP.

This section first presents how TCP flows can be redirected through MiMBoxes. Then the MPTCP-TCP conversion is outlined.

2.1 Traffic Redirection

To allow the protocol conversion, TCP segments must be sent to a MiMBox. There exists two possible redirection modes: *explicit* and *transparent*. With explicit redirection, the client sends its segments directly to a MiMBox to allow the latter to translate MPTCP to TCP. This is similar to the operation of an explicit proxy except that MiMBox is not restricted to a particular application (e.g. HTTP).

When using an explicit HTTP proxy, the client establishes a TCP connection to the proxy and includes the destination server as an HTTP header field. That way the proxy knows the original destination. Other solutions, like the SOCKS proxy, also require a specific application-level protocol to allow the client to indicate its desired destination. These solutions require that the applications support the proxy mechanism which is a major burden.

MiMBox does not modify the application layer. We propose a new TCP option, that we call Dst_Opt, to allow the client to announce the server address. The Dst_Opt provides the server's IP address to the MiMBox. Figure 1 shows how the establishment of new connections is performed via a MiMBox. When establishing a new connection the client places the Dst_Opt inside the SYN segment and the destination address for this connection is MiMBox's address. This allows the latter to forward the connection establishment to the server by rewriting the segment's IP addresses. By using its own IP address, all the reply-segments will be sent via the MiMBox. The Dst_Opt is added by the MPTCP/TCP stack and is thus transparent for the application.

In the rest of this paper we focus on the above explicit mode. Transparent redirection is also possible with MiMBox if it is on the path between the client and the server or through tunneling solutions, where all traffic is explicitly sent to a MiMBox by the border gateway of the local network (e.g., WCCP uses GRE tunnels [11], or a recent proposal by Sherry *et al.* [12]).

2.2 Protocol Conversion

The operations performed by the MiMBox to translate data segments can be viewed as a pipe, channeling segments from TCP to MPTCP and vice versa. Incoming segments on the MPTCP-side contain MPTCP options inside the TCP header. MiMBox has to handle the options' operation (e.g. new subflow establishment, etc.) and strip these options before forwarding them. MPTCP uses a separate sequence number space than the TCP sequence numbers [6]. Upon forwarding, MiMBox has to translate the MPTCP-level sequence numbers to the TCP sequence numbers on the server-side and vice versa. Further, as the TCP/IP header is modified, MiMBox has to update the TCP checksum.

As MPTCP creates multiple subflows, segments can arrive independently on each of these subflows. MiMBox therefore reorders the segments so that they form an in-order sequence of packets. Finally, MiMBox sends this sequence to the TCP side. For incoming traffic from the TCP side, MiMBox distributes the segments among the subflows. MiMBox distributes these segments using MPTCP's default scheduling algorithm [6].

3. IMPLEMENTATION

A MiMBox could be implemented as a user-space application. Existing HTTP proxies, such as Squid[1] and HAProxy[2] run in user space, which simplifies the development but may affect performance. First, these proxies are limited to specific applications and services. Each of these services runs on a specific port. Furthermore, the application needs to include a redirection mechanism to allow the *explicit mode* of MiMBox. MiMBox is application agnostic and does not require any application change.

To achieve high performance, the following goals are important:

Avoid Memory Allocation/Copy.
: The memory bus is a major limiting factor of the overall system performance. The gap between the processor and memory performance is important and still increasing [13]. To overcome the memory access bottleneck it is preferred to try to avoid memory allocations or copy when possible.

Minimize Context Switching.
: Limiting the number of context switches is important to obtain high performance. Context switching consumes CPU cycles that could have been used for other resources and it has an effect on the space locality of the data in the caches if processes are moved from one CPU to another.

To achieve both these goals MiMBox is implemented as a kernel module. This allows to avoid memory copy between user space and kernel space as well as to react as quickly as the packets arrive in the TCP/IP stack of the kernel. The module receives all incoming packets and performs two

[1] `http://www.squid-cache.org`
[2] `http://haproxy.1wt.eu`

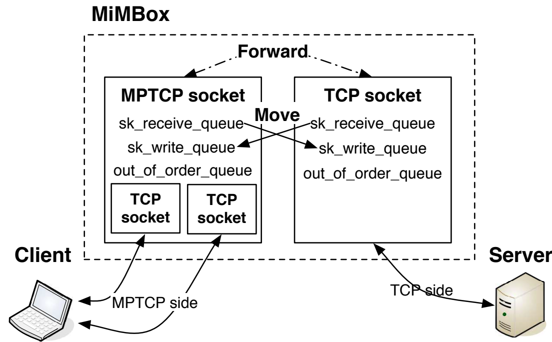

Figure 2: MiMBox maintains fully-functional sockets and forwards in-order segments from one side to the other.

operations: *(i)* handles connections establishment from the client and *(ii)* forwards packets from the client-side to the server-side and vice versa.

When receiving a SYN segment that requires a translation from TCP to MPTCP and vice versa, MiMBox creates two fully-functional sockets that are used as two separate connections. MiMBox terminates both connections meaning that the acknowledgements are not end-to-end but are handled by each socket. Figure 2 shows the behavior of MiMBox in this scenario. To understand how MiMBox can achieve high performance, one must understand how network packets are handled inside the kernel.

The Linux kernel uses special buffers, called `sk_buffs`, to handle network packets. When a segment arrives at the NIC a `sk_buff` is allocated. The buffer then traverses the TCP/IP stack and, for a TCP connection, is finally stored in the receive queue of its corresponding socket waiting for the application to copy the payload into its own buffer. `sk_buffs` are also allocated by the kernel when the application wants to send data. A user space application that would move data from one socket to another using the `read()` and `write()` system calls would cause two buffer allocations on top of two memory copies. To achieve high performance, MiMBox limits the number of allocations and avoids costly copy operations. Indeed, its operation consists of moving `sk_buffs` from the receive queue of one socket to the send queue of the other socket and vice versa. This is done by modifying pointer references as well as updating sequence numbers and the checksum. The cost of these operations is therefore minimal.

4. EVALUATION

In this section we evaluate the performance of MiMBox and compare it with user-space solutions.

We run experiments on three servers connected as described in Figure 3. They all use Intel Xeon X3440 running @ 2.53GHz and have 8GB of RAM. Each of these servers has at least one dual-port Intel 82599 10Gbit Ethernet card that supports hardware offloading.

Each host runs the v0.87 release of MPTCP[3]. On top of that, the converter runs our MiMBox implementation based on this MPTCP implementation. The client and the server run different pieces of software, they use either Weighttp[4] and Apache or Netperf [14]. The protocol converter is where the evaluation takes place. It is running either MiMBox, Squid, HAProxy, a custom application-level converter or acting as an IP router.

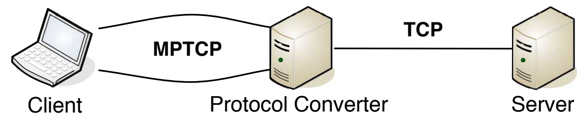

Figure 3: The setup used throughout the experiments.

4.1 Goodput

Inserting the protocol converter on the path between the client and the server may reduce the throughput, meaning the rate at which the end hosts can send/receive data over the network. To measure the impact on the application-level throughput (called goodput) we use Netperf. Netperf allows to send data from the client to the server at the highest possible speed up to the capacity provided by the network.

Our baseline to compare the performance is the setup where the converter is acting as a regular IP router. As the router does not perform any protocol conversion, we enable MPTCP on the server-side for this scenario.

We implemented two application-layer protocol converters to compare with our kernel-space implementation. These converters open a socket using the Netperf port on the client-side and listen for incoming connections. When a connection is established, the converter opens a new connection towards the server using the same port and an hardcoded address. The first application uses the `read()` and `write()` system calls to forward the data received on one socket to the other. The second application uses `splice()` [15]. `splice()` is a system call that allows to move data from one file descriptor to another without having to copy the data from kernel space to user space. Using `splice()` from one TCP socket file descriptor to another can be seen as forwarding at the TCP layer. Splicing is employed in web proxies where, after inspecting some part of a connection at the application level, it can make the rest of the connection fall back to TCP forwarding for optimal processing.

It must be noted that these application-level protocol converters are difficult to realize in practice. They must listen explicitly on a port and so are not application agnostic. Furthermore, current applications do not allow to specify the real server's destination IP address, they must modify the application data stream.

We evaluate the impact of the *Maximum Segment Size* (MSS) on the goodput. Figure 4 shows that MiMBox achieves better performance, close to simple IP forwarding. Overall, with a small MSS the maximum goodput, i.e. 10 Gbps, is not achieved with any solution. The lower the MSS, the higher the number of segments transmitted to achieve full goodput. The number of segments a device can generate/receive is a function of the number of interruptions that it can handle. The number of segments generated/received is therefore fixed. Increasing the MSS size thus increases

[3]`http://multipath-tcp.org`

[4]`http://redmine.lighttpd.net/projects/weighttp/wiki`

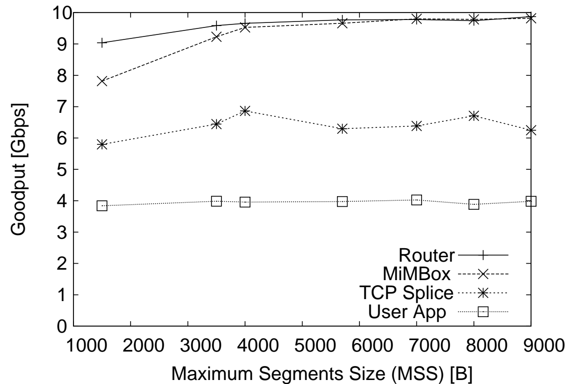

Figure 4: MiMBox always outperforms application-level solutions.

	SYN	SYN+ACK	Data
Router	5.2 ± 0.1 µs	5.2 ± 0.1 µs	5.3 ± 0.1 µs
`read()/write()`	143.4 ± 11.4 µs	N/A	29 ± 0.3 µs
MiMBox	17.5 ± 1.2 µs	41.6 ± 2 µs	16 ± 0.1 µs

Table 1: MiMBox introduces a very moderate per-packet delay – application-level solutions are much worse.

the goodput, as a larger amount of data is transmitted per segment.

The performance difference between MiMBox and its user-space counterparts can be explained by the fact that the latter ones are CPU-limited due to memory allocation and copy. `splice()`'s performance is mainly impacted by the memory allocation. When performing `splice()`, pages are moved, however `splice()` causes a new `sk_buff` allocation while this is not required in MiMBox which moves `sk_buff`s as is.

4.2 Forwarding delay

One important performance factor is the forwarding delay introduced by MiMBox. To validate our design we measure the forwarding delay which is the time the packet spends inside the converter. We use a custom application that sends bursts of 1300 bytes of data at the rate of 20 Kbps. We capture all packets entering and leaving the converter with `tcpdump` into a *ramdisk*. `tcpdump` stores, together with the packet, a timestamp of the moment the packet is entering/leaving the host. This timestamp allows us to measure the time each packet forwarded by the converter has spent inside the host. We also ensure that a route cache exists prior to measuring the delay to avoid influencing the SYN and SYN+ACK delays.

As in the previous section, the baseline is the performance achieved using the converter as a regular IP router. We profile MiMBox with a TCP and MPTCP-enabled server as well as the previously described `read()/write()` application. Table 1 shows the per segment forwarding delay introduced by each of the solutions averaged over 100 measures as well as the 95% confidence interval.

A first observation, is that the difference between the application and MiMBox is large for SYN packets. The poor performance of the `read()/write()` application is because the `accept()` call only returns when the ACK is received on the client-side of the converter. Therefore, the connection on the server-side is only established after the three-way handshake is completed on the client-side, resulting thus in a higher delay. Measuring the SYN+ACK delay is impossible since the SYN+ACK on the client-side is sent before the SYN+ACK from the server-side.

The difference in the SYN forwarding delay between MiMBox and the router is due to the additional operations performed by MiMBox. While the former needs to match to a route cache entry, the latter needs to change the IP addresses, verify the port numbers to avoid 5-tuple collisions and finally recompute the TCP checksum before forwarding the SYN to its final destination. When the server replies with the SYN+ACK, sockets must be created (see Section 3), this causes additional delay as the sockets must be created before the SYN+ACK on the client-side is generated.

After the three-way handshake, the forwarding delay for segments containing data is lower than for SYNs and SYN+ACKs as there are fewer operations to perform. The segments pass through the TCP/MPTCP stack to be forwarded. Traversing inside the two stacks introduces this additional delay.

The `read()/write()` application gives worse results when forwarding data as it consumes many CPU cycles transitioning from kernel space to user space.

4.3 Workload

In the previous sections we have evaluated the performance of a single connection through MiMBox. We now evaluate the cost of supporting network-heavy applications. We use weighttp running 40 parallel clients sending HTTP requests for varying file size. This is a standard stress test for TCP performance with short flows. We measure the number of requests per second that the client is able to perform.

As a baseline we use again the kernel-based router. We also profile existing HTTP proxies: HAProxy and Squid. These proxies terminate the MPTCP connections on the client-side and start a new TCP connection on the server-side. Their caching mechanisms are disabled as we want to measure pure forwarding performance. These proxies are application specific but still act similarly to MiMBox at the data level, they forward the HTTP requests and the response back to the client. We configured HAProxy to use `splice()`. For this the proxy performs a `read()` of the client requests so that it can lookup the destination server in the HTTP header and establish a new connection. As the reply from the server comes back, HAProxy uses `splice()` to forward the data to the client. Compared to Squid, HAProxy can use multiple threads to handle incoming requests. As our test server has 8 cores, we configured HAProxy to use either 1 or 8 threads.

Figure 5 shows (in log-scale) that MiMBox outperforms both HAProxy and Squid. For a request of a 1KB file respectively MiMBox, HAProxy multi-threaded, HAProxy mono-threaded and Squid are able to sustain around 22k, 18k, 9.5k and 2.5k requests per second. The result can be explained by the fact that MiMBox is quickly able to fully utilize the available bandwidth as well as optimize the three-way handshake. Indeed, the HTTP proxies have to wait for a HTTP GET request from the client to identify the server and establish the server-side connection. The HTTP proxy also has to parse the server address in the HTTP header. Figure 5 shows also that MiMBox sustains a similar num-

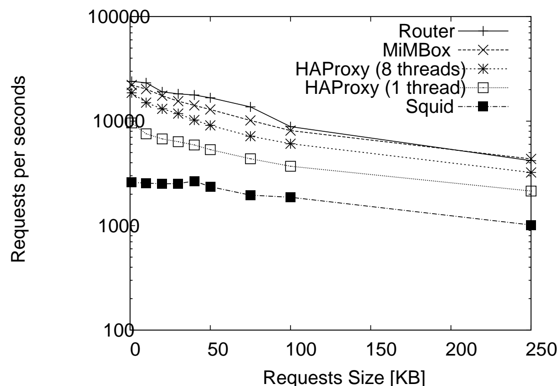

Figure 5: MiMBox supports a large number of HTTP clients – close to the performance of an IP router.

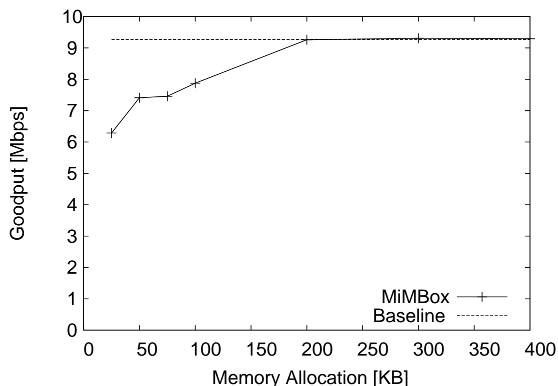

Figure 6: Client memory consumption is quite small to achieve a high bandwidth.

ber of requests per second as plain IP forwarding. For large file sizes the cost of establishing new connections is negligible. Nevertheless, in this case MiMBox still outperforms the user-space proxies and the limit comes from the end hosts. We also reduced the MSS to small sizes and we have not observed any difference in the results, MiMBox is always close to direct routing.

4.4 Buffering

By terminating connections MiMBox buffers data. Buffering consumes memory on MiMBox and thus might limit its ability to support a large number of clients. To analyze the impact of the buffer usage we performed two different evaluations where we limit the maximum memory allocated for each socket. For this, we used the system configuration parameters `tcp_rmem` and `tcp_wmem` that allow to configure the minimal, initial and maximum sizes of respectively the send and receive buffers.

We used a real-life simulated environment as presented in Figure 3. In this scenario, the client simulates a smartphone that has a WiFi (through ADSL) and a 3G connectivity and the proxy is present in the cloud. The links of the setup used in the previous evaluation where configured so that the client's ADSL-link has 8Mbps downstream and 512Kbps upstream capacity and 20 msec RTT to the proxy as well as a 3G link with 2Mbps downstream and 256Kbps upstream capacity and 80 msec to the proxy. The link between the proxy and the server is configured with a symmetric 100Mbps and 10 msec RTT with the server. The client-side is therefore the bottleneck in this scenario.

We first evaluate the memory consumption of MiMBox when the client transfers at the highest bandwidth available, i.e. 10 Mbps in download and 768 Kbps in upload. For this experiment we used Netperf to measure the bandwidth and varied the `tcp_rmem` and `tcp_wmem`. We use as baseline the mode without a proxy between the client and the server.

Figure 6 shows the download performance when the memory allocated varies. The client can achieve the highest performance with only 200KB of memory per socket (400 KB in total). A MiMBox having 8GB of memory could therefore scale up to 20,000 connections in this environment[5]. The memory usage could be improved. Indeed, in the download scenario, the socket facing the server does not require 200KB to achieve 10Mbps. Due to space limitations we do not evaluate this scenario.

Our second evaluation analyzes the impact of the buffering on interactive applications. For this we used the same environment and added a 1% loss on the client-side to cause retransmissions. We used a custom application [6] that sends at a fixed 100Kbps rate to simulate an interactive application. We then measured the application delay. We observed for a 200KB buffer that there is a 2 msec advantage for the MiMBox. More interestingly the maximum delay observed is around 150ms with MiMBox while it is 250ms without. Having a large delay can be problematic for interactive sessions. The reason that MiMBox reduces the delay in this scenario is that by terminating the connection it allows end hosts to react quickly to losses therefore causing fewer queuing inside the network. A similar tendency has also been observed with other buffer size limitations as well as when the path between the MiMBox and the server is lossy. We did not observe any significant difference when there is no loss.

5. RELATED WORK

The end-to-end principle assumes that TCP connections extend only between hosts. This is less and less true in today's Internet dominated by middleboxes [5, 12]. Starting in the 1980s, researchers have proposed protocol converters that terminate one transport connection on one side and initiate another one on the other side [16]. This idea has been applied by various authors. A first use case are wireless networks where end-to-end performance can be improved by splitting the TCP connection [17] and using improved retransmission techniques on the wireless link [18, 19]. MiMBox enables wireless hosts to simultaneously use several wireless interfaces at the same time, even when the servers do not yet support MPTCP.

Another example are the HTTP proxies. We discussed the differences between MiMBox and proxies in Section 3. Recently, a few authors have proposed some kind of MPTCP

[5]8GB is fairly small. Some Amazon EC2 instances can have up to 244 GB of memory. In this case MiMBox could support up to 600k connections.

proxies. Raiciu *et al.* [20] discuss the role that such proxies could play to support mobile hosts. However, this paper does not implement any such proxy. Ayar *et al.* [21] propose to use multiple paths inside the core network to improve performance while still using regular TCP between the end hosts and their proposed Splitter/Combiners. A prototype implementation of this solution above Linux `netfilter` is evaluated. Hampel and Klein [22] propose MPTCP proxies and anchors as well as some extensions to the MPTCP protocol to support them.

MiMBox could be implemented using solutions such as *netmap* [23] that allows a user-space application to interact directly with the NIC without going through the host's stack. While this brings high benefit for simple application such as switching, it still requires to implement a complete TCP/IP stack in user space to support MiMBox. As there do not exist such stack, it is unclear whether such implementation would achieve better performances.

6. CONCLUSION

Deploying a new TCP extension like MPTCP can be difficult despite its clear benefits for users. We propose a protocol converter called MiMBox to allow clients to already benefit from MPTCP during its deployment phase. MiMBox efficiently translates MPTCP to TCP and vice versa. By implementing MiMBox entirely in the Linux kernel, we achieve high performance on commodity x86 servers.

We compare the performance of MiMBox with existing HTTP proxies and simpler designs, e.g. using TCP Splice. Our evaluation shows that MiMBox overcomes existing proxies both when handling long TCP flows and when serving a large number of HTTP clients. From a performance viewpoint, MiMBox is close to the performance of an IP router. Furthermore, by evaluating the buffering cost of a client in a real life scenario we show that MiMBox can easily scale to a large number of clients using commodity hardware.

Acknowledgments

This work is partially funded by the European Commission funded CHANGE (INFSO-ICT-257422) project, by the Belgian Walloon Region under its FIRST Spin-Off Program (RICE project) and by the IAP-BESTCOM project.

7. REFERENCES

[1] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian, "Internet Inter-domain Traffic," in *ACM SIGCOMM*, 2010.

[2] V. Jacobson, "Congestion Avoidance and Control," in *ACM SIGCOMM*, 1988.

[3] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," RFC1323, May 1992.

[4] K. Fukuda, "An Analysis of Longitudinal TCP Passive Measurements," *Traffic Monitoring and Analysis*, 2011.

[5] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP?" in *ACM SIGCOMM IMC*, 2011.

[6] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP," in *USENIX NSDI*, 2012.

[7] C. Paasch, G. Detal, F. Duchene, C. Raiciu, and O. Bonaventure, "Exploring mobile/WiFi handover with multipath TCP," in *ACM SIGCOMM workshop CellNet*, 2012.

[8] O. Bonaventure, "Apple seems to also believe in Multipath TCP," 2013, see http://perso.uclouvain.be/olivier.bonaventure/blog/html/2013/09/18/mptcp.html.

[9] S. Sen, Y. Jin, R. Guérin, and K. Hosanagar, "Modeling the Dynamics of Network Technology Adoption and the Role of Converters," *IEEE/ACM Transactions on Networking*, vol. 18, no. 6, 2010.

[10] ETSI, "Network Functions Virtualisation – An Introduction, Benefits, Enablers, Challenges & Call for Action," Tech. Rep., Oct 2012, http://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[11] D. McLaggan, "Web Cache Communication Protocol V2, Revision 1," Working Draft, Internet-Draft draft-mclaggan-wccp-v2rev1-00, Aug. 2012.

[12] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service," in *ACM SIGCOMM*, 2012.

[13] B. Jacob, S. Ng, and D. Wang, *Memory systems: cache, DRAM, disk.* Morgan Kaufmann, 2010.

[14] R. Jones *et al.*, "Netperf: a network performance benchmark," *Information Networks Division, Hewlett-Packard Company*, 1996.

[15] D. A. Maltz and P. Bhagwat, "TCP Splicing for Application Layer Proxy Performance," *Journal of High Speed Networks*, vol. 8, no. 3, 1999.

[16] I. Groenbaek, "Conversion Between the TCP and ISO Transport Protocols as a Method of Achieving Interoperability Between Data Communications Systems," *IEEE Journal on Selected Areas in Communications*, vol. 4, no. 2, 1986.

[17] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts," in *International Conference on Distributed Computing Systems*, 1995.

[18] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links," in *ACM SIGCOMM*, 1996.

[19] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," RFC 3135, 2001.

[20] C. Raiciu, D. Niculescu, M. Bagnulo, and M. J. Handley, "Opportunistic Mobility with Multipath TCP," in *ACM workshop MobiArch*, 2011.

[21] T. Ayar, L. Budzisz, and A. Wolisz, "TCP over Multiple Paths Revisited: Towards Transparent Proxy Solutions," in *IEEE ICC*, June 2012.

[22] G. Hampel and T. Klein, "MPTCP Proxies and Anchors," Working Draft, Internet-Draft draft-hampel-mptcp-proxies-anchors-00, Feb. 2012.

[23] L. Rizzo, "Netmap: a novel framework for fast packet i/o," in *Proc. of the USENIX conference on Annual Technical Conference*, ser. USENIX ATC'12. USENIX Association, 2012, pp. 9–9.