acmqueue

Multipath TCP

Decoupled from IP, TCP is at last able to support multihomed hosts

Christoph Paasch and Olivier Bonaventure, UCL

The Internet relies heavily on two protocols. In the network layer, IP (Internet Protocol) provides an unreliable datagram service and ensures that any host can exchange packets with any other host. Since its creation in the 1970s, IP has seen the addition of several features, including multicast, IPsec (IP security), and QoS (quality of service). The latest revision, IPv6 (IP version 6), supports 16-byte addresses.

The second major protocol is TCP (Transmission Control Protocol), which operates in the transport layer and provides a reliable bytestream service on top of IP. TCP has evolved continuously since the first experiments in research networks.

Still, one of the early design decisions of TCP continues to frustrate many users. TCP and IP are separate protocols, but the separation between the network and transport protocols is not complete. To differentiate the individual data streams among incoming packets, a receiving end host demultiplexes the packets based on the so-called 5-tuple, which includes the IP addresses, port numbers, and protocol identifiers. This implies that a TCP connection is bound to the IP addresses used on the client and the server at connection-establishment time. Despite the growing importance of mobile nodes such as smartphones and tablets, TCP connections cannot move from one IP address to another. When a laptop switches from Ethernet to Wi-Fi it obtains another IP address. All existing TCP connections must be torn down and new connections restarted.

Various researchers have proposed solutions to this problem over the past several years. The first approach was to solve the problem in the network layer. Examples of such solutions include Mobile IP, HIP (Host Identity Protocol), and Shim6 (Site Multihoming by IPv6 Intermediation). A significant drawback of these network-layer solutions is that they hide all changes to the addresses and thus the paths from TCP's congestion-control scheme. This is inefficient since the congestion-control scheme sends data over paths that may change without notice.

Another possible solution is to expose multiple addresses to the transport layer. This is the approach chosen for SCTP (Stream Control Transmission Protocol).[17] SCTP is an alternative transport protocol capable of supporting several IP addresses per connection. The first versions of SCTP used multiple addresses in failover scenarios, but recent extensions have enabled it to support the simultaneous use of several paths.[9]

Unfortunately, except in niche applications such as signaling in telephony networks, SCTP has not been widely deployed. One reason is that many firewalls and NAT (Network Address Translation) boxes are unable to process SCTP packets and thus simply discard them. Another reason is that SCTP exposes a different API from the socket API to the applications. These two factors lead to the classic chicken-and-egg problem. Network manufacturers do not support SCTP in their firewalls because no application is using this protocol; and application developers do not use SCTP because firewalls discard SCTP packets. There have been attempts to break this vicious circle by encapsulating SCTP

on top of UDP (User Datagram Protocol) and exposing a socket interface to the application, but widespread usage of SCTP is still elusive.

MPTCP (Multipath TCP) is designed with these problems in mind. More specifically, the design goals for MPTCP are:[15]

- It should be capable of using multiple network paths for a single connection.
- It must be able to use the available network paths at least as well as regular TCP, but without starving TCP.
- It must be as usable as regular TCP for existing applications.
- Enabling MPTCP must not prevent connectivity on a path where regular TCP works.

The following section describes the main architectural principles that underlie MPTCP. In an ideal network, these simple principles should have been sufficient. Unfortunately, this is not the case in today's Internet, given the prevalence of middleboxes, as explained later.

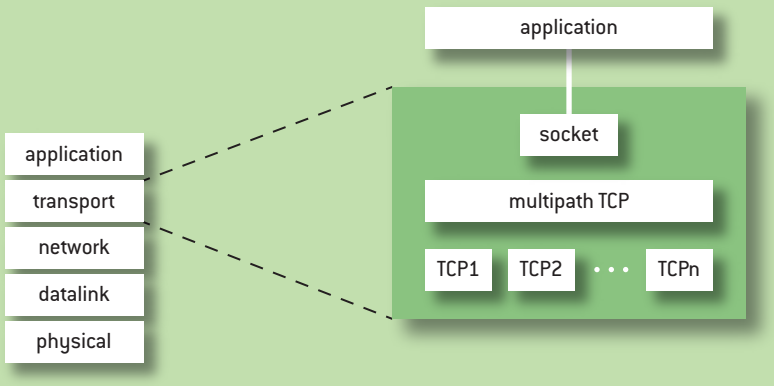
THE ARCHITECTURAL PRINCIPLES

Applications interact through the regular socket API, and MPTCP manages the underlying TCP connections (called subflows[6]) that are used to carry the actual data. From an architectural viewpoint, MPTCP acts as a shim layer between the socket interface and one or more TCP subflows, as shown in figure 1. MPTCP requires additional signaling between the end hosts. It achieves this by using TCP options to achieve the following goals:

- Establish a new MPTCP connection.
- Add subflows to an MPTCP connection.
- Transmit data on the MPTCP connection.

An MPTCP connection is established by using the three-way handshake with TCP options to negotiate its usage. The `MP_CAPABLE` option in the SYN segment indicates that the client supports MPTCP. This option also contains a random key used for security purposes. If the server supports MPTCP, then it replies with a SYN+ACK segment that also contains the `MP_CAPABLE` option. This option contains a random key chosen by the server. The third ACK of the three-way handshake also

FIGURE 1 **Multipath TCP in the Stack**



includes the MP_CAPABLE option to confirm the utilization of MPTCP and the keys to enable stateless servers.

The three-way handshake shown in figure 2 creates the first TCP subflow over one interface. To use another interface, MPTCP uses a three-way handshake to establish one subflow over this interface. Adding a subflow to an existing MPTCP connection requires the corresponding MPTCP connection to be uniquely identified on each end host. With regular TCP, a TCP connection is always identified by using the tuple <SourceIP, DestIP, SourcePort, DestPort>.

Unfortunately, because NAT is present, the addresses and port numbers that are used on the client may not be the same as those exposed to the server. Although on each host the 4-tuple is a unique local identification of each TCP connection, this identification is not globally unique. MPTCP needs to be able to link each subflow to an existing MPTCP connection. For this, MPTCP assigns a locally unique token to each connection. When a new subflow is added to an existing MPTCP connection, the MP_JOIN option of the SYN segment contains the token of the associated MPTCP connection. This is illustrated in figure 3.

The astute reader may have noticed that the MP_CAPABLE option does not contain a token. Still, the token is required to enable the establishment of subflows. To reduce the length of the MP_CAPABLE option and avoid using all the limited TCP options space (40 bytes) in the SYN segment, MPTCP derives the token as the result of a truncated hash of the key. The second function of the MP_JOIN option is to authenticate the addition of the subflow. For this, the client and the server exchange random nonces, and each host computes an HMAC (hash-based message authentication code) over the random nonce chosen by the other host and the keys exchanged during the initial handshake.

Now that the subflows have been established, MPTCP can use them to exchange data. Each host can send data over any of the established subflows. Furthermore, data transmitted over one subflow can be retransmitted on another to recover from losses. This is achieved by using two levels of sequence numbers.[6] The regular TCP sequence number ensures that data is received in order over each subflow and allows losses to be detected. MPTCP uses the data sequence number to reorder the data received over different subflows before passing it to the application.

FIGURE 2 **Connection Establishment**

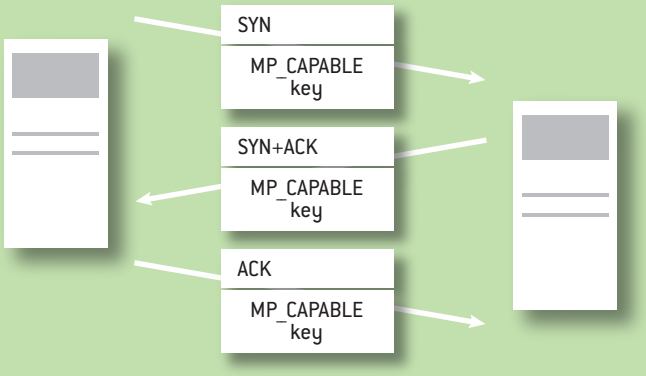


FIGURE 3

Establishment of an Additional Subflow

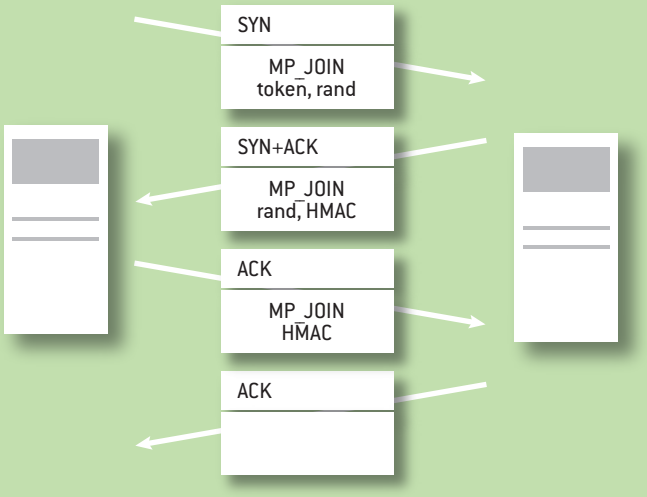
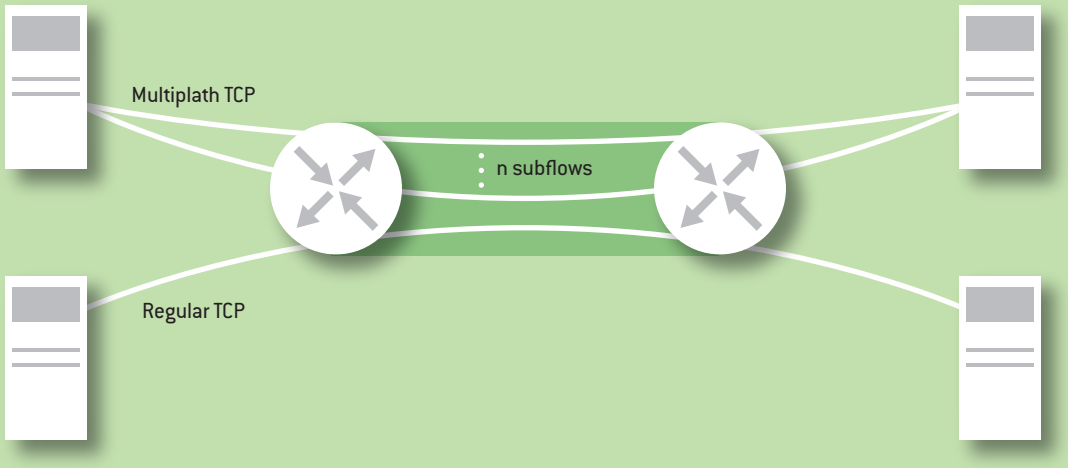


FIGURE 4

Coupled Congestion Control



From a congestion-control viewpoint, using several subflows for one connection leads to an interesting problem. With regular TCP, congestion occurs on one path between the sender and the receiver. MPTCP uses several paths, and two paths will typically experience different levels of congestion. A naive solution to the congestion problem in MPTCP would be to use the standard TCP congestion-control scheme on each subflow. This can be easily implemented but leads to unfairness with regular TCP. In the network depicted in figure 4, two clients share the same bottleneck link. If the MPTCP-enabled client uses two subflows, then it will obtain two-thirds of the shared bottleneck.

This is unfair because if this client used regular TCP, it would obtain only half of the shared bottleneck.

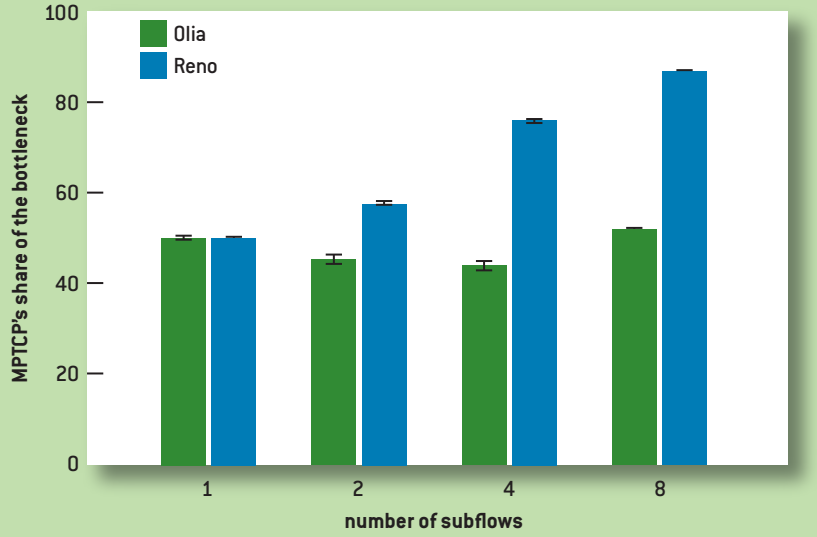
Figure 5 shows the effect of using up to eight subflows across a shared bottleneck with the standard TCP congestion-control scheme (Reno). In this case, MPTCP would use up to 85 percent of the bottleneck's capacity, effectively starving regular TCP. Specific MPTCP congestion-control schemes have been designed to solve this problem.[10,18] Briefly, they measure congestion on each subflow and try to move traffic away from those with the most congestion. To do so, they adjust the TCP congestion window on each subflow based on the level of congestion. Furthermore, the congestion windows on the different subflows are coupled to ensure that their aggregate does not grow faster than a single TCP connection. Figure 5 shows that the OLIA congestion-control scheme[10] preserves fairness with regular TCP across the shared bottleneck.

THE DEVIL IS IN THE MIDDLEBOXES

Today's Internet is completely different from the networks for which the original TCP was designed. These networks obeyed the end-to-end principle, which implies that the network is composed of switches and routers that forward packets but never change their contents. Now the Internet contains various middleboxes in addition to these routers and switches. A recent survey revealed that enterprise networks often contain more middleboxes than regular routers.[16] These middleboxes include not only the classic firewalls and NATs, but also devices such as transparent proxies, virtual private network gateways, load balancers, deep-packet inspectors, intrusion-detection systems, and WAN accelerators. Many of these devices process and sometimes modify the TCP header and even the payload of passing TCP segments. Dealing with those middleboxes has been one of the most difficult challenges in designing the MPTCP protocol, as illustrated by the examples that follow.[6,7,15]

FIGURE 5

Coupled Congestion Control Is Fair to Regular TCP Across Shared Bottlenecks



Upon receiving data on a TCP subflow, the receiver must know the data-sequence number to reorder the data stream before passing it to the application. A simple approach would be to include the data-sequence number in a TCP option inside each segment. Unfortunately, this clean solution does not work. There are deployed middleboxes that split segments on the Internet. In particular, all modern NICs (network interface cards) act as segment-splitting middleboxes when performing TSO (TCP segmentation offloading). These NICs split a single segment into smaller pieces. In the case of TSO, CPU cycles are offloaded from the operating system to the NIC, as the operating system handles fewer, larger segments that are split down to MTU (maximum transmission unit)-sized segments by the NIC. The TCP options, including the MPTCP data-sequence number, of the large segment are copied in each smaller segment. As a result, the receiver will collect several segments with the same data-sequence numbers, and be unable to reconstruct the data stream correctly. The MPTCP designers solved this by placing a mapping in the data-sequence option, which defines the beginning (with respect to the subflow sequence number) and the end of the data-sequence number (indicating the length of the mapping). MPTCP can thus correctly work across segment-splitting middleboxes and can be used with NICs that use TCP segmentation offloading to improve performance.

Using the first MPTCP implementation in the Linux kernel to perform measurements revealed another type of middlebox. Since the implementation worked well in the lab, it was installed on remote servers. The first experiment was disappointing. An MPTCP connection was established but could not transfer any data. The same kernel worked perfectly in the lab, but no one could understand why longer delays would prevent data transfer. The culprit turned out to be a local firewall that was changing the sequence numbers of all TCP segments. This feature was added to firewalls several years ago to prevent security issues with hosts that do not use random initial sequence numbers. In some sense, the firewall was fixing a security problem in older TCP stacks, but in trying to solve this problem, it created another problem. The mapping from subflow-sequence number to data-sequence number was wrong as the firewall modified the former. Since then, the mapping in the data-sequence option uses relative subflow-sequence numbers compared with the initial sequence number, instead of using absolute sequence numbers.[6]

The data-sequence option thus accurately maps each byte from the subflow-sequence space to the data-sequence space, allowing the receiver to reconstruct the data stream. Some middleboxes may still disturb this process: the application-level gateways that modify the payload of segments. The canonical example is active FTP. FTP uses several TCP connections that are signaled by exchanging ASCII-encoded IP addresses as command parameters on the control connection. To support active FTP, NAT (Network Address Translation) boxes have to modify the private IP address that is being sent in ASCII by the client host. This implies a modification of not only the content of the payload, but also, sometimes, its length, since the public IP address of the NAT device may have a different length in ASCII representation. Such a change in the payload length will make the mapping from subflow-sequence space to data-sequence space incorrect. MPTCP can even handle such middleboxes thanks to a checksum that protects the payload of each mapping. If an application-level gateway modifies the payload, then the checksum will be corrupted; MPTCP will be able to detect the payload change and perform a seamless fallback to regular TCP to preserve the connectivity between the hosts.

Several researchers have analyzed the impact of these middleboxes in more detail. One study used

active measurements over more than 100 Internet paths to detect the prevalence of various forms of middleboxes.[8] The measurements revealed that protocol designers can no longer assume that the Internet is transparent when designing a TCP extension; and no TCP extension can assume that a single field of the TCP header will reach the destination without any modification. TCP options are not safer. Some of these options can be modified on their way to the destination. Furthermore, some middleboxes remove or copy TCP options. Despite these difficulties, MPTCP is able to cope with most middleboxes.[7] When faced with middlebox interference, MPTCP always tries to preserve connectivity. In some situations, this is achieved by falling back to regular TCP.[6,7]

USE CASES

Two use cases that have already been analyzed in the literature will serve to illustrate possible applications of MPTCP. Other use cases will probably appear in the future.

MPTCP ON SMARTPHONES

Smartphones are equipped with Wi-Fi and 3G/4G interfaces, but they typically use only one interface at a time. Still, users expect their TCP connections to survive when their smartphone switches from one wireless network to another. With regular TCP, switching networks implies changing the local IP address and leads to a termination of all established TCP connections.[4] With MPTCP, the situation could change because it enables seamless handovers from Wi-Fi to 3G/4G and vice versa.[13]

Different types of handovers are possible:

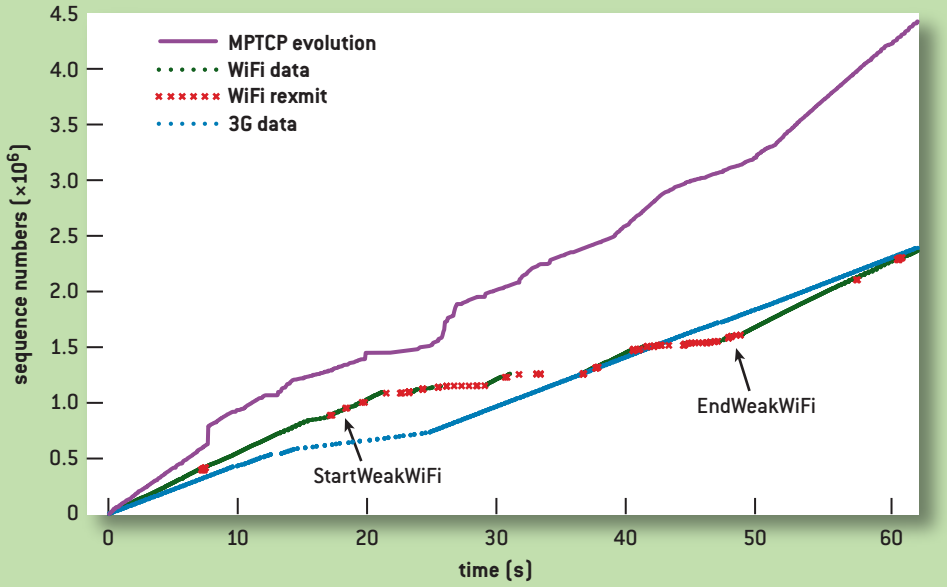
- First, the *make-before-break* handover can be used if the smartphone can predict that one interface will disappear soon (e.g., because of a decreasing radio signal). In this case, a new subflow will be initiated on the second interface and the data will switch to this interface.
- With the *break-before-make* handover, MPTCP can react to a failure on one interface by enabling another interface and starting a subflow on it. Once the subflow has been created, the data that was lost as a result of the failure of the first interface can be retransmitted on the new subflow, and the connection continues without interruption.
- The third handover mode uses two or more interfaces simultaneously. With regular TCP, this would be a waste of energy. With MPTCP, data can be transmitted over both interfaces to speed up the data transfer. From an energy viewpoint, enabling two radio interfaces is more costly than enabling a single one; however, the phone's display often consumes much more energy than the radio interfaces.[2] When the user looks at the screen (e.g., while waiting for a Web page), increasing the download speed by combining two interfaces could reduce the display usage and thus the energy consumption.

As an illustration, let's analyze the operation of MPTCP in real Wi-Fi and 3G networks. The scenario is simple but representative of many situations. A user starts a download inside a building over Wi-Fi and then moves outside. The Wi-Fi connectivity is lost as the user moves, but thanks to MPTCP the data transfer is not affected. Figure 6 shows a typical MPTCP trace collected by using both 3G and Wi-Fi. The x-axis shows the time since the start of the data transfer, while the y-axis displays the evolution of the sequence numbers of the outgoing packets.

For this transfer, MPTCP was configured to use both Wi-Fi and 3G simultaneously to maximize performance. The blue curve shows the evolution of the TCP sequence numbers on the TCP subflow over the 3G interface. The 3G subflow starts by transmitting at a regular rate. Between seconds 14

FIGURE 6

3G/WiFi Handover with Multipath TCP



and 24, the 3G interface provides a lower throughput. This can be explained by higher congestion or a weaker radio signal since the smartphone moves inside the building during this period. After this short period, the 3G subflow continues to transmit at a constant rate. Note that TCP does not detect any loss over the 3G interface. The black curve shows the evolution of the TCP sequence number on the Wi-Fi subflow. The red crosses represent retransmitted packets.

A comparison of the 3G and Wi-Fi curves reveals that Wi-Fi is usually faster than 3G, but packets are more frequently lost over the Wi-Fi interface. While the smartphone moves, there are short periods of time during which the Wi-Fi interface behaves like a black hole. The interface seems to be active, but no packet is transmitted correctly. These periods can last several seconds and severely impact the user experience when Wi-Fi is used alone. With MPTCP, the packets that are lost over the Wi-Fi interface are automatically retransmitted over the 3G subflow, and the MPTCP connection continues without interruption.

The upper gray curve shows the evolution of this MPTCP connection based on the returned acknowledgments. During the first seven seconds, MPTCP aggregates the Wi-Fi and 3G interfaces perfectly. The MPTCP throughput is the sum of the throughput of the two interfaces. Then a few packets are lost over the Wi-Fi interface. The first vertical bar on the gray curve corresponds to the reception of the first acknowledgment after the recovery of these losses. When the Wi-Fi interface is weak, the MPTCP throughput varies with the quality of the interface, but the data transfer continues. This ability to exploit the available interfaces quickly is a key benefit of MPTCP from a performance viewpoint.

Improving the data-transfer performance is not the only use case for MPTCP on smartphones.[13] It is also possible to use only the 3G interface as a backup instead of enabling both the Wi-Fi and

the 3G interfaces. When the Wi-Fi interface is active, all data is sent over it. If it becomes inactive, MPTCP automatically switches to the 3G interface to continue the data transfer and stops using it as soon as the Wi-Fi interface comes back.

Another possible use case is the 802.11 community network. Many cities are now covered by a large number of Wi-Fi access points accessible to many users.[3] The density of these access points makes it possible for slow-moving users to rely mainly on Wi-Fi for Internet access and on MPTCP to maintain the connections when moving from one access point to another.

The smartphone use case has motivated Apple to implement MPTCP in iOS 7. As of this writing, MPTCP is not yet available through the socket API on iOS 7 and is used only to support the Siri voice-recognition application. It is also possible to use the MPTCP Linux kernel on some recent rooted Android smartphones (see http://multipath-tcp.org/).

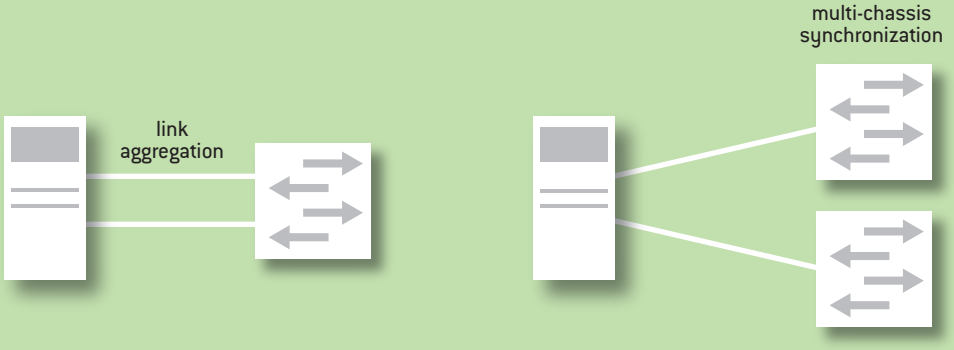
MPTCP IN THE DATA CENTER

Another important use case for MPTCP lies in data centers.[14] Today, most servers are equipped with several high-speed interfaces, which can be combined to achieve either higher performance or better resilience to failures. When two or more interfaces are grouped together to improve performance, they are usually attached to the same switch, as illustrated on the left-hand side of figure 7. To enable TCP to use these two interfaces efficiently, they usually appear as a single logical interface having one MAC address and one IP address. The bonding driver on the server distributes the packets over the combined interfaces; several load-balancing algorithms exist.[4] A round-robin system allows efficient spreading of the packets over the different interfaces. If the links have different delays, however, it causes reordering, which hurts TCP performance.

Most deployments opt for hash-based load-balancing techniques. Some fields of the Ethernet and IP and/or TCP headers are hashed to select the outgoing interface. Thanks to this hashing, the packets that belong to the same TCP connection are sent over the same interface to prevent reordering, and packets belonging to different connections are spread over the available interfaces. This solution works well if the traffic is composed of a large number of short TCP connections. Since

FIGURE 7

Link Bonding for Performance (Left) and Availability (Right)



all the packets belonging to one TCP connection are sent over the same interface, however, a single TCP connection cannot achieve a higher throughput than the link it uses. This is a major limitation, given that long TCP connections are frequent in data centers.[1]

In some deployments, two or more interfaces are combined to improve availability. In this case, each interface is attached to a different switch, as illustrated on the right-hand side of figure 7. Most deployments opt for using the same MAC and IP addresses on the two interfaces. One interface is used to carry all packets, and the other serves as a backup. When this mode is chosen, only one interface at a time is used to carry packets.

With MPTCP, it is possible both to improve performance and to achieve high availability. A typical MPTCP-aware server design would use two interfaces attached to different switches, as illustrated in figure 8. Each interface has its own MAC and IP addresses. Once an MPTCP connection has been initiated over one interface, MPTCP will announce the other available IP addresses to the remote host by using the `ADD_ADDR` option,[6] and subflows will be established over these interfaces.

Once these subflows have been established, the MPTCP congestion-control scheme will dynamically spread the load over the available interfaces. If one interface or any intermediate switch fails, MPTCP automatically changes to the remaining paths.

Note that in contrast with existing load-balancing solutions, the interfaces used by MPTCP do not need to have the same bandwidth. This allows MPTCP to increase the throughput even for a single data stream by distributing the load over all interfaces. Indeed, the Linux kernel implementation of MPTCP described earlier[11] allowed memory-to-memory transfers between two high-end servers, each equipped with six 10-Gbps interfaces. MPTCP was able to use the capacity of the six interfaces efficiently and reached 53 Gbps for a single connection.[12]

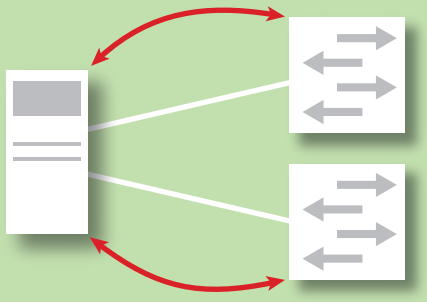
On the server side, most experiments with MPTCP have been performed with the Linux MPTCP kernel (available from http://multipath-tcp.org). A FreeBSD implementation is being developed, and one commercial load balancer supports MPTCP. [5]

CONCLUSION

MPTCP is a major extension to TCP. By decoupling TCP from IP, TCP is at last able to support multihomed hosts. With the growing importance of wireless networks, multihoming is becoming

FIGURE 8

Link Bonding with Multipath TCP Needs One IP Address Per Interface



the norm instead of the exception. Smartphones and data centers are the first use cases where MPTCP can provide benefits.

ACKNOWLEDGMENTS

The development of MPTCP has been a collective work involving many researchers, including Sébastien Barré, Gregory Detal, Fabien Duchene, Phil Eardley, Alan Ford, Mark Handley, Benjamin Hesmans, and Costin Raiciu. This work has been supported by the European Commission under FP7 (the seventh Framework Programme for Research) projects Trilogy, CHANGE, and Trilogy 2; a gift from Google; and the Bestcom IAP.

REFERENCES

1. Benson, T., Akella, A., Maltz, D. A. 2010. Network traffic characteristics of data centers in the wild. In Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC).
2. Carroll, A., Heiser, G. 2010. An analysis of power consumption in a smartphone. In Proceedings of the Usenix Annual Technical Conference (USENIXATC).
3. Castignani, G., Blanc, A., Lampropulos, A., Montavont, N. 2012. Urban 802.11 community networks for mobile users: current deployments and prospectives. *Mobile Networks and Applications* 17(6): 796-807.
4. Davis, T., et al. 2011. Linux Ethernet bonding driver how-to; https://www. kernel.org/doc/Documentation/networking/bonding.txt.
5. Eardley, P. 2013. Survey of MPTCP implementations. Internet draft, work in progress; http://tools.ietf.org/html/draft-eardley-mptcp-implementations-survey-02.
6. Ford, A., Raiciu, C., Handley, M., Bonaventure, O. 2013. TCP extensions for multipath operation with multiple addresses. RFC6824; http://www.rfc-editor.org/rfc/rfc6824.txt.
7. Hesmans, B., Duchene, F., Paasch, C., Detal, G., Bonaventure, O. 2013. Are TCP extensions middlebox-proof? In Proceedings of the 2013 Workshop on Hot Topics in Middleboxes and Network Function Virtualization (Hot-Middlebox).
8. Honda, M., Nishida, Y., Raiciu, C., Greenhalgh, A., Handley, M., Tokuda, H. 2011. Is it still possible to extend TCP? In Proceedings of the ACM SIGCOMM Internet Measurement Conference (IMC).
9. Iyengar, J. R., Amer, P. D., Stewart, R. 2006. Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths. *IEEE/ACM Transactions on Networking* 14(5): 951-964.
10. Khalili, R., Gast, N., Popovic, M., Upadhyay, U., Le Boudec, J.-Y. 2012. MPTCP is not Pareto-optimal: performance issues and a possible solution. In Proceedings of the 8th ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT).
11. Paasch, C., Barré, S., Detal, G., Duchene, F. 2014. Linux kernel implementation of Multipath TCP; http://multipath-tcp.org.
12. Paasch, C., Detal, G., Barré, S., Duchene, F., Bonaventure, O. 2013. The fastest TCP connection with MultiPath TCP; http://multipath-tcp.org/pmwiki.php?n=Main.50Gbps.
13. Paasch, C., Detal, G., Duchene, F., Raiciu, C., Bonaventure, O. 2012. Exploring mobile/Wi-Fi handover with Multipath TCP. In Proceedings of the ACM SIGCOMM workshop on Cellular Networks: Operations, Challenges, and Future Design (CellNet).
14. Raiciu, C., Barré, S., Pluntke, C., Greenhalgh, A., Wischik, D., Handley, M. 2011. Improving

datacenter performance and robustness with Multipath TCP. ACM SIGCOMM *Computer Communication Review* 41(4): 266-277.
15. Raiciu, C., Paasch, C., Barré, S., Ford, A., Honda, M., Duchene, F., Bonaventure, O., Handley, M. 2012. How hard can it be? Designing and implementing a deployable Multipath TCP. In Proceedings of the 9th Symposium on Networked Systems Design and Implementation (NSDI).
16. Sherry, J., Hasan, S., Scott, C. Krishnamurthy, A., Ratnasamy, S., Sekar, V. 2012. Making middleboxes someone else's problem: network processing as a cloud service. ACM SIGCOMM *Computer Communication Review* 42(4): 13-24.
17. Stewart, R. Xie, Q. 2001. *Stream Control Transmission Protocol (SCTP): A Reference Guide*. Addison-Wesley.
18. Wischik, D., Raiciu, C. Greenhalgh, A., Handley, M. 2011. Design, implementation and evaluation of congestion control for Multipath TCP. In Proceedings of the 8th Usenix Symposium on Networked Systems Design and Implementation (NSDI).

LOVE IT, HATE IT? LET US KNOW

feedback@queue.acm.org

CHRISTOPH PAASCH is a Ph.D student at Université catholique de Louvain (Louvain-la-Neuve, Belgium), where he leads the development of Multipath TCP in the Linux kernel.

OLIVIER BONAVENTURE is a professor at Université catholique de Louvain (Louvain-la-Neuve, Belgium), where he leads the networking group. His research focuses on Internet protocols. He wrote an open source networking textbook and is education director of ACM SIGCOMM.