Multipath TCP: Great Appeal for Telecommunications Providers

Mamta Kumari[1] Mr. Vipin Arora[2]
[1]M. Tech Student [2]Guide & Lecturer
[2]Department of Computer Science
[1,2]BITS, Bhiwani (hr)

Abstract— Networks have become multipath: mobile devices have multiple radio interfaces, datacenters have redundant paths and multihoming is the norm for big server farms. Mean-while, TCP is still only single-path.Is it possible to extend TCP to enable it to support multiple paths for current applications on today's Inter-net? The answer is positive. We carefully review the constraints—partly due to various types of middleboxes— that influenced the design of Multipath TCP and show how we handled them to achieve its deployability goals. We report our experience in implementing Multipath TCP in the Linux kernel and we evaluate its performance. Our measurements focus on the algorithms needed to ef-ficiently use paths with different characteristics, notably send and receive buffer tuning and segment reordering. We also compare the performance of our implementa-tion with regular TCP on web servers. Finally, we dis-cuss the lessons learned from designing MPTCP.

Key words: MPTCP, TCP, ACK

I. INTRODUCTION

In today's Internet, servers are often multi-homed to more than one Internet provider, datacenters provide mul-tiple parallel paths between compute nodes, and mobile hosts have multiple radios. Traditionally, it was the role of routing to take advantage of path diversity, but this has limits to responsiveness and scaling. To really gain both robustness and performance advantages, we need transport protocols engineered to utilize multiple paths. Multipath TCP is an attempt to extend the TCP pro-tocol to perform this role. At the time of writing, it is in in the final stage of standardization in the IETF.

Multipath TCP stripes data from a single TCP con-nection across multiple subflows, each of which may take a different path through the network. A linked con-gestion control mechanism controls how much data is sent on each subflow, with the goal of explicitly mov-ing traffic off the more congested paths onto the less congested ones. This paper is not about congestion con-trol, but rather it is about the design of the Multipath TCP protocol itself. In principle, extending TCP to use multiple paths is not difficult, and there are a number of obvious ways in which it could be done. Indeed it was first proposed by Christian Huitema. In practice though, the existence of middleboxes greatly constrains the design choices. The challenge is to make Multipath TCP not only robust to path failures, but also robust to failures in the presence of middleboxes that at-tempt to optimize single-path TCP flows. No previous extension to the core Internet protocols has needed to consider this issue to nearly the same extent.

In the first half of this paper we examine the design options for multipath TCP, with the aim of understand-ing both the end-to-end problem and the end-to-middle-to-end constraints. We use the results of a large Internet study to validate these design choices.

Designing MPTCP turned out to be more difficult than expected. For instance, a key question concerns how MPTCP metadata should be encoded — embed it in the TCP payload, or use the more traditional TCP to find and utilize unused network capacity in redun-dant topologies. Rather, the main contribution of this paper is the exploration of the design space for MPTCP confined by the many constraints imposed by TCP's original design, today's networks which embed TCP knowledge, and the need to perform well within the limitations imposed by the operating system.

II. HOW IT WORK?

A TCP session is normally set up by way of a three-way handshake. The initiating host sends a packet with the SYN flag set, the receiving host, if it is amenable to the connection, responds with a packet containing both the SYN and ACK flags. The final ACK packet sent by the initiator puts the connection into the "established" state; after that, data can be transferred in either direction.

An MPTCP session starts in the same way, with one change: the initiator adds the new MP_CAPABLE option to the SYN packet. If the receiving host supports MPTCP, it will add that option to its SYN-ACK reply; the two hosts will also include cryptographic keys in these packets for later use. The final ACK (which must also carry the MP_CAPABLE option) establishes a multipath session, albeit a session using a single path just like traditional TCP.
When MPTCP is in use, both sides recognize a distinction between the session itself and any specific "subflow" used by that session. So, at any point, either party to the session can initiate another TCP connection to the other side, with the proviso that the address and/or port at one end or the other of the connection must differ. So, if a smartphone has initiated an MPTCP connection to a server using its WiFi interface:

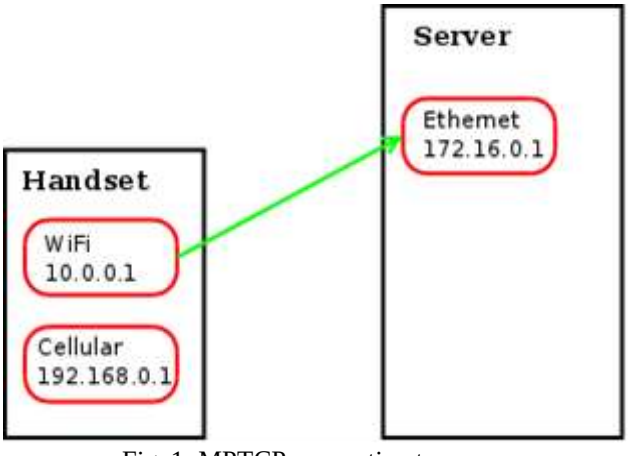

Fig. 1: MPTCP connection to server

It can add another subflow at any time by connecting to the same server by way of its cellular interface:

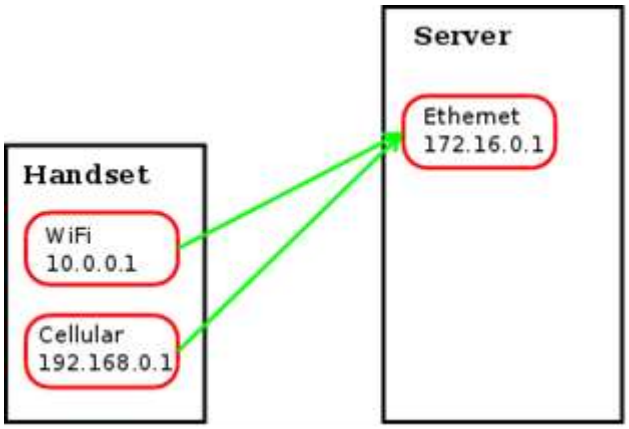

Fig. 2: Server by way of its cellular interface:

III. GOALS

As many researchers have lamented, changing the be-havior of the core Internet protocols is very difficult [7]. An idea may have great merit, but without a clear de-ployment path whereby the cost/benefit tradeoff for early adopters is positive, widespread adoption is unlikely.

We wish to move from a single-path Internet to one where the robustness, performance and load-balancing benefits of multipath transport are available to all appli-cations, the majority of which use TCP for transport. To support such unmodified applications we must work be-low the sockets API, providing the same service as TCP: byte-oriented, reliable and in-order delivery. In theory we could use different protocols to implement this func-tionality as long as fallback to TCP is possible when one end does not support multipath. In practice there is no widely deployed signaling mechanism to select between transport protocols, so we have to use options in TCP's SYN exchange to negotiate new functionality.

The goal is for an unmodified application to start (what it believes to be) a TCP connection with the regular API. When both endpoints support MPTCP and multiple paths are available, MPTCP can set up additional subflows and stripe the connection's data across these subflows, sending most data on the least congested paths.

The potential benefits are clear, but there may be costs too. If negotiating MPTCP can cause connections to fail when regular TCP would have succeeded, then deploy-ment is unlikely. The second goal, then, is for MPTCP to work in all scenarios where TCP currently works. If a subflow fails for any reason, the connection must be able to continue as long as another subflow has connectivity.

Third, MPTCP must be able to utilize the network at least as well as regular TCP, but without starving TCP. The congestion control scheme described in [23] meets this requirement, but congestion control is not the only factor that can limit throughput.

Finally MPTCP must be implementable in operating systems without using excessive memory or processing. As we will see, this requires careful consideration of both fast-path processing and overload scenarios.

IV. DESIGN

The five main mechanisms in TCP are:

- Connection setup handshake and state machine.
- Reliable transmission & acknowledgment of data.
- Congestion control.
- Flow control.
- Connection teardown handshake and state machine.

The simplest possible way to implement Multipath TCP would be to take segments coming out of the reg-ular stack and "stripe" them across the available paths somehow 1. For this to work well, the sender would need to know which paths perform well and which don't: it would need to measure per path RTTs to quickly and ac-curately detect losses. To achieve these goals, the sender must remember which segments it sent on each path and use TCP Selective Acknowledgements to learn which segments arrive. Using this information, the sender could drive retransmissions independently on each path and maintain congestion control state.

A. Connection Setup

Both the connections can be used together for one single connection

1) MPTCP (Multi path TCP) will increase the bandwidth

It will increase the bandwidth because two connection links with two separate paths are used in a single connection. Due to congestion if one path is only providing a small percentage of its bandwidth, the other path can also be utilized. Hence the total bandwidth for a MPTCP connection will be the combined bandwidth used by both the paths/links

2) MPTCP (Multi Path TCP) provides Better Redundancy

Multi path TCP provides a better connection redundancy, because your connection will not be affected even if one link goes down. An example use case is suppose you are watching a video online and you are over your WiFi connection. Even if you walk out of your WiFi connection range, the video streaming should not be affected because it should automatically stop sending data through WiFi connection and should now only use cellular network. The end user must never feel that there was a connection loss of some kind.

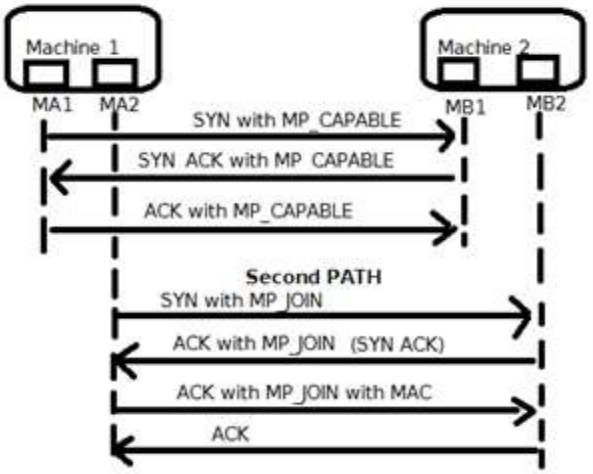

Fig. 3: Connection between two machines.

Technical details of a proposed technology are very easy to describe theoretically. But a proper implementation of that in the real world is quite difficult. The main difficulty in launching any protocol is that, it should not break an already existing one, and should be capable of working at ease with the already deployed infrastructure.

B. Adding Subflows

Once two endpoints have negotiated MPTCP, they can open additional subflows. In an ideal world there would be no need to send new SYN packets before sending data on a new subflow - all that would be needed is a way to identify the connection that packets belong to. The strawman design simply sent TCP segments along dif-ferent paths, and the endpoints used the 5-tuple to iden-tify the proper connection. In practice though, we see that NATs and Firewalls rarely pass data packets that were not preceded by a SYN.

Adding a subflow raises two problems. First, the new subflow needs to be associated with an existing MPTCP flow. The classical five-tuple cannot be used as a con-nection identifier, as it does not survive NATs. Second, MPTCP must be robust to an attacker that attempts to add his own subflow to an existing MPTCP connection.

When the first MPTCP subflow is established, the client and the server insert 64-bit random keys in the MP CAPABLE option. These will be used to verify the authenticity of new subflows.

To open a new subflow, MPTCP performs a new SYN exchange using the additional addresses or ports it wishes to use. Another TCP option, MP JOIN is added to the SYN and SYN/ACKs. This option carries a MAC of the keys from the original subflow; this prevents blind spoofing of MP JOIN packets from an adversary who wishes to hijack an existing connection. MP JOIN also contains a connection identifier derived as a hash of the recipient's key this is used to match the new subflow to an existing connection.

C. Reliable Multipath Delivery

In a world without middleboxes, MPTCP could sim-ply stripe data across the multiple subflows, with the se-quence numbers in the TCP headers indicating the se-quence number of the data in the connection in the nor-mal TCP way. Our measurements show that this is in-feasible in today's Internet: connection identifier derived as a hash of the recipient's key this is used to match the new subflow to an existing connection.

D. Reliable Multipath Delivery

In a world without middleboxes, MPTCP could sim-ply stripe data across the multiple subflows, with the se-quence numbers in the TCP headers indicating the se-quence number of the data in the connection in the nor-mal TCP way. Our measurements show that this is in-feasible in today's Internet:

- We observed that 10% of access networks rewrite TCP initial sequence numbers (18% on port 80). Some of this re-writing is by proxies that remove new op-tions; a new subflow will fail on these paths. But many that rewrite do pass new options - these appear to be firewalls that attempt to increase TCP initial se-quence number randomization. As a result, MPTCP cannot assume the sequence number space on a new subflow is the same as that on the original subflow.
- Striping sequence numbers across two paths leaves gaps in the sequence space seen on any single path. We found that 5% of paths (11% on port 80) do not pass on data after a hole - most of these seem to be proxies that block new options on SYNs and so don't present a problem as MPTCP is never enabled on these paths. But a few do not appear to be proxies, and so would stall MPTCP. Perhaps worse, 26% of paths (33% on port 80) do not correctly pass on an ACK for data the middlebox has not observed - ei-ther the ACK is dropped or it is "corrected".

Given the nature of today's Internet, it appears extremely unwise to stripe a single TCP sequence space across more than one path. The only viable solution is to use a separate contiguous sequence space for each MPTCP subflow. For this to work, we must also send informa-tion mapping bytes from each subflow into the overall data sequence space, as sent by the application. We shall return to the question of how to encode such mappings after first discussing flow control and acknowledgments, as the three are intimately related.

E. Connection and Subflow Teardown

TCP has two ways to indicate connection shutdown: FIN for normal shutdown and RST for errors such as when one end no longer has state. With MPTCP, we need to distinguish subflow teardown from connection teardown.

With RST, the choice is clear: it must only terminate the subflow, or an error on a single subflow would cause the whole connection to fail.

Normal shutdown is slightly more subtle. TCP FINs occupy sequence space; the FIN/FIN-ACK/ACK hand-shake and the cumulative nature of TCP's acknowledg-ments ensure that not only all data has been received, but also both endpoints know the connection is closed and know who needs to hold TIMEWAIT state.

How then should a FIN on an MPTCP subflow be in-terpreted? Does it mean that the sending host has no more data to send, or only that no more data will be sent on this subflow? Another way to phrase this is to ask whether a FIN on a subflow occupies data sequence space, or just subflow sequence space?

Consider first what would happen if a FIN occupied data sequence space. This could be achieved by extend-ing the length of the DSM mapping in a packet to cover the FIN. Mapping the FIN into the data sequence space in this way tells the receiver what the data sequence number of the last byte of the connection is, and hence whether any more data is expected from other subflows.

V. MPTCP PERFORMANCE

The two main motivations to deploy MPTCP today are wireless networks where MPTCP could enable hosts to use both WiFi and 3G networks and datacen-ters where MPTCP allows servers to better exploit the load-balanced paths . We experimentally evaluate the performance of our MPTCP implementation in these two environments.

A. MPTCP over WiFi and 3G

In the previous section, we used emulated networks to improve the algorithms used in our MPTCP implementation. Here, we use MPTCP over a 3G network offered by a commercial provider in Belgium; TCP's maximum throughput on this network is 2Mbps. Our MPTCP implementation correctly works over this 3G network de-spite its installed middleboxes. We also used a WiFi access point that was capped at 2 Mbps. This capping was implemented on the access point and would rep-resent the bandwidth offered on a shared public WiFi network such as BT's FON. Figure 9 shows the aver-age goodput achieved by TCP and MPTCP in function of the receive/send buffer sizes. Regular TCP achieves roughly the same goodput with both 3G and WiFi except when the buffer size is small where the larger round-trip-time penalizes the performance over 3G. MPTCP gets most of the available bandwidth when the buffer reaches 200KB, and it never underperforms TCP.

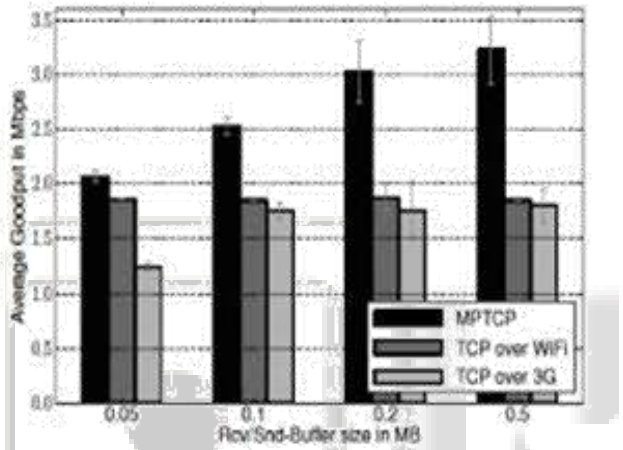

Fig. 4: MPTCP over WiFi and 3G

Our measurements show that MPTCP is able to utilize both the 3G and WiFi networks when the buffer is large enough. With a 500 KBytes buffer, MPTCP achieves almost the double of the goodput of regular TCP. With a 100 KBytes buffer, reaches a goodput that is 25% larger than regular TCP over WiFi or 3G.

B. Connection Setup Latency

During MPTCP connection setup the client and server generate a random key and verify that its hash is unique among all established connections .
shows a PDF of the delay between receiving a SYN and sending the SYN/ACK, measured at the server. For regular TCP, 91% of the 20,000 connection setup attempts are processed in 6µs.

Each connection is closed before the next attempt is made. Setting up the first subflow of an MPTCP connection takes the server between 10 and 11µs if it has no es-tablished connections. The extra latency is mainly be-cause MPTCP must hash the received key, generate the server key and verify that its hash is unique. how the latency increases when the server already has 100 and 1000 established MPTCP connections.

This additional latency, although small compared to a LAN RTT, could be significantly reduced by maintaining a pool of precomputed keys.

VI. RELATED WORK

There has been a good deal of work on building mul-tipath transport protocol.Most of this work aims to leave applications unchanged and focuses on the protocol mechanisms needed to im-plement multipath transmission. Key goals are robust-ness to long term path failures and to short term varia-tions in conditions on the paths.

Both MTCP and M/TCP use a single se-quence number together with a scoreboard at the sender that allows maintaining congestion state and perform-ing retransmissions per path. MTCP uses a single return path for ACKs which decreases its robustness; also it has been designed to run over an overlay network (RON), reducing its deployability and efficiency.

pTCp is one of the most complete proposals to date. The SYN exchange signals the addresses that will be used in the multipath connection, and this set is fixed - pTCP does not support mobility. Congestion control and retransmissions are performed per subflow. At con-nection level there are global send and receive buffers; a data sequence number and acknowledgment helps deal with reordered data at the connection level. This is in-serted in a pTCP header that follows the TCP header.

RMTP is a rate-based protocol targeted for mo-bile hosts that uses packet-pairs to estimate available bandwidth on each path and supports both reliable an unreliable delivery. RMTP does not offer the same ser-vice as TCP, and requires app changes.

The Stream Control Transmission Protocol (SCTP) has been designed with multihoming in mind to sup-port telephony signaling applications. SCTP's multi-homing support was initially only capable of recovering from failures. However, several authors have extended it to support load-sharing. SCTP-CMT uses a single sequence number across all paths and keeps a scoreboard and other information to have accurate per-path congestion windows and to drive retransmissions.

Our protocol design has drawn on all this literature, and has been further guided by our experimental study of middleboxes. In light of this study, none of the exist-ing approaches are deployable as most use single sub-flow sequence numbers which will be dropped. pTCP does not use subflow sequence numbers but it is un-clear how its additional headers should be encoded. Fur-ther, pTCP will not cope with resegmenting or content-changing middleboxes.

On the OS side, none of the previous works on TCP have addressed the practical problems of getting mul-tipath transport working in reality. Most use simula-tion analysis, and do not consider receive-buffer issues. SCTP-CMT has been implemented in the FreeBSD ker-nel, but its performance has not been evaluated in detail.

A technique that was previously proposed to reduce the size of the receive-buffer is to use sender-side schedul-ing to get the packets "in- order" at the receiver Unfortunately, this solution is brittle: any packet losses or just variations in RTT will disrupt the ordering, causing the receiver to buffer just as much data. Further, the sender still has to buffer as much data as before.

VII. LESSONS LEARNED

In today's Internet, the three-way-handshake involves not only the two communicating hosts, but also all the middleboxes on the path. Verifying the presence of a particular TCP option in a SYN+ACK is not sufficient to ensure that a TCP extension can be safely used. , some middleboxes pass TCP options that they don't understand. This is safe for TCP options that are purely informative but causes problems with other options such as those that redefine the semantics of TCP header fields. For exam-ple, the large window extension in RFC1323 changes the semantics of the window field of the TCP header and extends it beyond 16 bits. Nearly 20 years after the publication of RFC1323, there are still stateful firewalls that do not understand this option in SYNs but block data packets that are sent in the RFC1323 extended window. A TCP extension that changes the semantics of parts of the packet header must include mechanisms to cope with middleboxes that do not understand the new semantics.

In an end-to-end Internet, all the information carried inside TCP packets is immutable. Today this is no longer true: the entire TCP header and the payload must be considered as mutable fields. If a TCP extension needs to rely on a particular field, it must check its value in a way that cannot be circumvented by middleboxes that do not understand this extension. The DSM checksum is an example of a solution to deal with these problems. Most importantly, deployable TCP extensions must necessarily include techniques that enable them to fall-back to regular TCP when something wrong happens. If a middlebox interferes badly with a TCP extension, the problem must be detected and the extension auto-matically disabled to preserve the data transfer. A TCP extension will only be deployed if it guarantees that it will transfer data correctly (and hopefully better) in all the cases where a regular TCP is able to transfer data.

VIII. CONCLUSIONS

TCP was designed when the Internet strictly obeyed the end-to-end principle and each host had a single IP address. Single-homing is disappearing and a growing fraction of hosts have multiple interfaces/addresses. In this paper we evaluated whether TCP can be extended to efficiently support such hosts.

We explored whether it was possible to design Mul-tipath TCP in a way that is still deployable in today's Internet. The answer is positive, but any major change to TCP must take into account the various types of mid-dleboxes that have proliferated. In fact, they influenced most of the design choices in Multipath TCP besides the congestion control. Our experiments show that MPTCP safely operates through all the middleboxes we've iden-tified in our previous study [9].

From an implementation viewpoint, we proposed new algorithms to solve practical but important problems such as sharing a limited receive buffer between multiple flows on a smartphone, or optimizing the MPTCP receive code. Experiments show that our techniques are effective, mak-ing MPTCP ready for adoption.

This work highlights once again the fact that hidden middleboxes increase the complexity of the Internet, mak-ing evolution difficult. We should revisit the Internet architecture to recognize explicitly their role. The big challenge, however, is to build a solution that is deploy-able in today's Internet.

REFERENCES

[1] S. Barr´e. Implementation and Assessment of Modern Host-based Multipath Solutions. PhD thesis, Universit´ catholique de Louvain, November 2011.
[2] M. Becke et al. Load sharing for the stream control transmission protocol (sctp). Internet draft, draft-tuexen-tsvwg-sctp-multipath-02.txt, work in progress, July 2011.
[3] A. Bittau, M. Hamburg, M. Handley, D. Mazi`eres, and D. Boneh. The case for ubiquitous transport-level encryption. In USENIX Security'10, pages 26–26, Berkeley, CA, USA, 2010. USENIX Association.
[4] Y. Dong, D. Wang, N. Pissinou, and J. Wang. Multi-path load balancing in transport layer. In EuroNGI Conference, 2007.
[5] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP extensions for multipath operation with multiple addresses, Jan 2012. IETF draft (work in progress).
[6] FreeBSD Project. tcp – internet transmission control protocol. FreeBSD Kernel Interfaces Manual.
[7] M. Handley. Why the internet only just works. BT Technology Journal, 24:119–129, 2006.
[8] M. Handley, V. Paxson, and C. Kreibich. Network intrusion detection: evasion, traffic normalization, and end-to-end protocol semantics. In Proc. USENIX Security Symposium, pages 9–9, 2001.
[9] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda. Is it still possible to extend TCP? In IMC 2011, 11th Internet Measurement Conference, Nov. 2011.
[10] H.-Y. Hsieh and R. Sivakumar. A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts. In Proc. MobiCom '02, pages 83–94, New York, NY, USA, 2002. ACM.
[11] C. Huitema. Multi-homed TCP. Internet draft, IETF, 1995.
[12] J. Iyengar, P. Amer, and R. Stewart. Performance implications of a bounded receive buffer in concurrent multipath transfer. Computer Communications, 30(4), February 2007.
[13] J. R. Iyengar, P. D. Amer, and R. Stewart. Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths. IEEE/ACM Trans. Netw., 14(5):951–964, 2006.
[14] P. Key, L. Massouli´e, and D. Towsley. Path selection and multipath congestion control. In Proc. IEEE Infocom, May 2007.
[15] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. ACM Trans. Comput. Syst., 18:263–297, August 2000.
[16] L. Magalhaes and R. Kravets. Transport level mechanisms for bandwidth aggregation on mobile hosts. ICNP, page 0165, 2001.
[17] F. Mirani, M. Kherraz, and N. Boukhatem. Forward prediction scheduling: Implementation and performance evaluation. In ICT 2011, pages 321 –326, may 2011.

[18] C. Pluntke, L. Eggert, and N. Kiukkonen. Saving mobile device energy with Multipath TCP. In MobiArch, 2011.
[19] C. Raiciu, S. Barr´e, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving datacenter performance and robustness with Multipath TCP. In Proc ACM Sigcomm, 2011.
[20] C. Raiciu, D. Niculescu, M. Bagnulo, and M. Handley. Opportunistic mobility with Multipath TCP. In MobiArch, 2011.
[21] K. Rojviboonchai and H. Aida. An evaluation of multi-path transmission control protocol (M/TCP) with robust acknowledgement schemes. IEICE Trans. Communications, 2004.
[22] P. Srisuresh and M. Holdrege. IP Network Address Translator (NAT) terminology and considerations. RFC 2663, August 1999.
[23] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. Design, implementation and evaluation of congestion control for multipath tcp. In NSDI'11, Berkeley, CA, USA, 2011. USENIX Association.
[24] M. Zhang, J. Lai, A. Krishnamurthy, L. Peterson, and R. Wang. A transport layer approach for improving end-to-end performance and robustness using redundant paths. In Proc USENIX '04, 2004.

