

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 15, 2014

M. Bagnulo
UC3M
C. Paasch
UCLouvain
F. Gont
SI6 Networks / UTN-FRH
O. Bonaventure
UCLouvain
C. Raiciu
UPB
July 14, 2013

Analysis of MPTCP residual threats and possible fixes
draft-bagnulo-mptcp-attacks-00

Abstract

This document performs an analysis of the residual threats for MPTCP and explores possible solutions to them.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 15, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. ADD_ADDR attack	4
2.1. Possible security enhancements to prevent this attack . .	9
3. DoS attack on MP_JOIN	10
3.1. Possible security enhancements to prevent this attack . .	11
4. SYN flooding amplification	11
4.1. Possible security enhancements to prevent this attack . .	12
5. Eavesdropper in the initial handshake	12
5.1. Possible security enhancements to prevent this attack . .	13
6. Security considerations	13
7. IANA Considerations	13
8. Acknowledgments	13
9. References	13
9.1. Normative References	13
9.2. Informative References	14
Authors' Addresses	15

1. Introduction

This document provides a complement to the threat analysis for Multipath TCP (MPTCP) [RFC6824] documented in RFC 6181 [RFC6181]. RFC 6181 provided a threat analysis for the general solution space of extending TCP to operate with multiple IP addresses per connection. Its main goal was to leverage previous experience acquired during the design of other multi-address protocols, notably SHIM6 [RFC5533], SCTP [RFC4960] and MIPv6 [RFC3775] during the design of MPTCP. Thus, RFC 6181 was produced before the actual MPTCP specification (RFC6824) was completed, and documented a set of recommendations that were considered during the production of such specification.

This document complements RFC 6181 with a vulnerability analysis of the specific mechanisms specified in RFC 6824. The motivation for this analysis is to identify possible security issues with MPTCP as currently specified and propose security enhancements to address the identified security issues.

The goal of the security mechanisms defined in RFC 6824 were to make MPTCP no worse than currently available single-path TCP. We believe that this goal is still valid, so we will perform our analysis on the same grounds.

Types of attackers: for all attacks considered in this documents, we identify the type of attacker. We can classify the attackers based on their location as follows:

- o Off-path attacker. This is an attacker that does not need to be located in any of the paths of the MPTCP session at any point in time during the lifetime of the MPTCP session. This means that the Off-path attacker cannot eavesdrop any of the packets of the MPTCP session.
- o Partial time On-path attacker. This is an attacker that needs to be in at least one of the paths during part but not during the entire lifetime of the MPTCP session. The attacker can be in the forward and/or backward directions, for the initial subflow and/or other subflows. The specific needs of the attacker will be made explicit in the attack description.
- o On-path attacker. This attacker needs to be on at least one of the paths during the whole duration of the MPTCP session. The attacker can be in the forward and/or backward directions, for the initial subflow and/or other subflows. The specific needs of the attacker will be made explicit in the attack description.

We can also classify the attackers based on their actions as follows:

- o Eavesdropper. The attacker is able to capture some of the packets of the MPTCP session to perform the attack, but it is not capable of changing, discarding or delaying any packet of the MPTCP session. The attacker can be in the forward and/or backward directions, for the initial subflow and/or other subflows. The specific needs of the attacker will be made explicit in the attack description.
- o Active attacker. The attacker is able to change, discard or delay some of the packets of the MPTCP session. The attacker can be in the forward and/or backward directions, for the initial subflow and/or other subflows. The specific needs of the attacker will be made explicit in the attack description.

In this document, we consider the following possible combinations of attackers:

- o an On-path eavesdropper
- o an On-path active attacker
- o an Off-path active attacker

- o a Partial-time On-path eavesdropper
- o a Partial-time On-path active attacker

In the rest of the document we describe different attacks that are possible against the MPTCP protocol specified in RFC6824 and we propose possible security enhancements to address them.

2. ADD_ADDR attack

Summary of the attack:

Type of attack: MPTCP session hijack enabling Man-in-the-Middle.

Type of attacker: Off-path, active attacker.

Threat: Medium

Description:

In this attack, the attacker uses the ADD_ADDR option defined in RFC6824 to hijack an ongoing MPTCP session and enables himself to perform a Man-in-the-Middle attack on the MPTCP session.

Consider the following scenario. Host A with address IPA has one MPTCP session with Host B with address IPB. The MPTCP subflow between IPA and IPB is using port PA on host A and port PB on host B. The tokens for the MPTCP session are TA and TB for Host A and Host B respectively. Host C is the attacker. It owns address IPC. The attack is executed as follows:

1. Host C sends a forged packet with source address IPA, destination address IPB, source port PA and destination port PB. The packet has the ACK flag set. The TCP sequence number for the segment is *i* and the ACK sequence number is *j*. We will assume all these are valid, we discuss what the attacker needs to figure these ones later on. The packet contains the ADD_ADDR option. The ADD_ADDR option announces IPC as an alternative address for the connection. It also contains an eight bit address identifier which does not bring any strong security benefit.
2. Host B receives the ADD_ADDR message and it replies by sending a TCP SYN packet. (Note: the MPTCP specification states that the host receiving the ADD_ADDR option may initiate a new subflow. If the host is configured so that it does not initiate a new subflow the attack will not succeed. For example, on the Linux implementation, the server does not create subflows. Only the client does so.) The source address for the packet is IPB, the

destination address for the packet is IPC, the source port is PB' and the destination port is PA' (It is not required that PA=PA' nor that PB=PB'). The sequence number for this packet is the new initial sequence number for this subflow. The ACK sequence number is not relevant as the ACK flag is not set. The packet carries an MP_JOIN option and it carries the token TA. It also carries a random nonce generated by Host B called RB.

3. Host C receives the SYN+MP_JOIN packet from Host B, and it alters it in the following way. It changes the source address to IPC and the destination address to IPA. It sends the modified packet to Host A, impersonating Host B.
4. Host A receives the SYN+MP_JOIN message and it replies with a SYN/ACK+MP_JOIN message. The packet has source address IPA and destination address IPC, as well as all the other needed parameters. In particular, Host A computes a valid HMAC and places it in the MP_JOIN option.
5. Host C receives the SYN/ACK+MP_JOIN message and it changes the source address to IPC and the destination address to IPB. It sends the modified packet to IPB impersonating Host A.
6. Host B receives the SYN/ACK+MP_JOIN message. Host B verifies the HMAC of the MP_JOIN option and confirms its validity. It replies with an ACK+MP_JOIN packet. The packet has source address IPB and destination address IPC, as well as all the other needed parameters. The returned MP_JOIN option contains a valid HMAC computed by Host B.
7. Host C receives the ACK+MP_JOIN message from B and it alters it in the following way. It changes the source address to IPC and the destination address to IPA. It sends the modified packet to Host A impersonating Host B.
8. Host A receives the ACK+MP_JOIN message and creates the new subflow.

At this point the attacker has managed to place itself as a MitM for one subflow for the existing MPTCP session. It should be noted that there still exists the subflow between address IPA and IPB that does not flow through the attacker, so the attacker has not completely intercepted all the packets in the communication (yet). If the attacker wishes to completely intercept the MPTCP session it can do the following additional step.

9. Host C sends two TCP RST messages. One TCP RST packet is sent to Host B, with source address IPA and destination address IPB and source and destination ports PA and PB, respectively. The other TCP RST message is sent to Host A, with source address IPB and destination address IPA and source and destination ports PB and PA, respectively. Both RST messages must contain a valid sequence number. Note that figuring the sequence numbers to be used here for subflow A is the same difficulty as being able to send the initial ADD_ADDR option with valid Sequence number and ACK value. If there are more subflows, then the attacker needs to find the Sequence Number and ACK for each subflow.

At this point the attacker has managed to fully hijack the MPTCP session.

Information required by the attacker to perform the described attack:

In order to perform this attack the attacker needs to guess or know the following pieces of information: (The attacker need this information for one of the subflows belonging to the MPTCP session.)

- o the four-tuple {Client-side IP Address, Client-side Port, Server-side Address, Servcer-side Port} that identifies the target TCP connection
- o a valid sequence number for the subflow
- o a valid ACK sequence number for the subflow
- o a valid address identifier for IPC

TCP connections are uniquely identified by the four-tuple {Source Address, Source Port, Destination Address, Destination Port}. Thus, in order to attack a TCP connection, an attacker needs to know or be able to guess each of the values in that four-tuple. Assuming the two peers of the target TCP connection are known, the Source Address and the Destination Address can be assumed to be known.

We note that in order to be able to successfully perform this attack, the attacker needs to be able to send packets with a forged source address. This means that the attacker cannot be located in a network where techniques like ingress filtering [RFC2827] or source address validation [I-D.ietf-savi-framework] are deployed. However, ingress filtering is not as widely implemented as one would expect, and hence cannot be relied upon as a mitigation for this kind of attack.

Assuming the attacker knows the application protocol for which the TCP connection is being employed, the server-side port can also be assumed to be known. Finally, the client-side port will generally not be known, and will need to be guessed by the attacker. The chances of an attacker guessing the client-side port will depend on the ephemeral port range employed by the client, and whether the client implements port randomization [RFC6056].

Assuming TCP sequence number randomization is in place (see e.g. [RFC6528]), an attacker would have to blindly guess a valid TCP sequence number. That is,

$$\text{RCV.NXT} = < \text{SEG.SEQ} < \text{RCV.NXT} + \text{RCV.WND} \text{ or } \text{RCV.NXT} = < \text{SEG.SEQ} + \text{SEG.LEN} - 1 < \text{RCV.NXT} + \text{RCV.WND}$$

As a result, the chances of an attacker to succeed will depend on the TCP receive window size at the target TCP peer.

We note that automatic TCP buffer tuning mechanisms have been become common for popular TCP implementations, and hence very large TCP window sizes of values up to 2 MB could end up being employed by such TCP implementations.

According to [RFC0793], the Acknowledgement Number is considered valid as long as it does not acknowledge the receipt of data that has not yet been sent. That is, the following expression must be true:

$$\text{SEG.ACK} \leq \text{SND.NXT}$$

However, for implementations that support [RFC5961], the following (stricter) validation check is enforced:

$$\text{SND.UNA} - \text{SND.MAX.WND} \leq \text{SEG.ACK} \leq \text{SND.NXT}$$

Finally, in order for the address identifier to be valid, the only requirement is that it needs to be different than the ones already being used by Host A in that MPTCP session, so a random identifier is likely to work.

Given that a large number of factors affect the chances of an attacker of successfully performing the aforementioned off-path attacks, we provide two general expressions for the expected number of packets the attacker needs to send to succeed in the attack: one for MTCP implementations that support [RFC5961], and another for MPTCP implementations that do not.

Implementations that do not support RFC 5961

$$\text{Packets} = (2^{32}/(\text{RCV_WND})) * 2 * \text{EPH_PORT_SIZE}/2 * 1/\text{MSS}$$

Where the new :

Packets:

Maximum number of packets required to successfully perform an off-path (blind) attack.

RCV_WND:

TCP receive window size (RCV.WND) at the target node.

EPH_PORT_SIZE:

Number of ports comprising the ephemeral port range at the "client" system.

MSS:

Maximum Segment Size, assuming the attacker will send full segments to maximize the chances to get a hit.

Notes:

The value "2³²" represents the size of the TCP sequence number space.

The value "2" accounts for 2 different ACK numbers (separated by 2³¹) that should be employed to make sure the ACK number is valid.

The following table contains some sample results for the number of required packets, based on different values of RCV_WND and EPH_PORT_SIZE for a MSS of 1500 bytes.

Ports \ Win	16 KB	128 KB	256 KB	2048 KB
4000	699050	87381	43690	5461
10000	1747626	218453	109226	13653
50000	8738133	1092266	546133	68266

Table 1: Max. Number of Packets for Successful Attack

Implementations that do not support RFC 5961

$$\text{Packets} = (2^{32}/(\text{RCV_WND})) * (2^{32}/(\text{SND_MAX_WND})) * \text{EPH_PORT_SIZE}/2 * 1/\text{MSS}$$

Where:

Packets:

Maximum number of packets required to successfully perform an off-path (blind) attack.

RCV_WND:

TCP receive window size (RCV.WND) at the target MPTCP endpoint.

SND_MAX_WND:

Maximum TCP send window size ever employed by the target MPTCP end-point (SND.MAX.WND).

EPH_PORT_SIZE:

Number of ports comprising the ephemeral port range at the "client" system.

Notes:

The value "2³²" represents the size of the TCP sequence number space.

The parameter "SND_MAX_WND" is specified in [RFC5961].

The following table contains some sample results for the number of required packets, based on different values of RCV_WND, SND_MAX_WND, and EPH_PORT_SIZE. For these implementations, only a limited number of sample results are provided, just as an indication of how [RFC5961] increases the difficulty of performing these attacks.

Ports \ Win	16 KB	128 KB	256 KB	2048 KB
4000	91625968967	1431655765	357913941	559240

Table 2: Max. Number of Packets for Successful Attack

Note:

In the aforementioned table, all values are computed with RCV_WND equal to SND_MAX_WND.

2.1. Possible security enhancements to prevent this attack

1. To include the token of the connection in the ADD_ADDR option. This would make it harder for the attacker to launch the attack, since he needs to either eavesdrop the token (so this can no longer be a blind attack) or to guess it, but a random 32 bit number is not so easy to guess. However, this would imply that any eavesdropper that is able to see the token, would be able to launch this attack. This solution then increases the vulnerability window against eavesdroppers from the initial 3-way

handshake for the MPTCP session to any exchange of the ADD_ADDR messages.

2. To include the HMAC of the address contained in the ADD_ADDR option concatenated with the key of the receiver of the ADD_ADDR message. This makes it much more secure, since it requires the attacker to have both keys (either by eavesdropping it in the first exchange or by guessing it). Because this solution relies on the key used in the MPTCP session, the protection of this solution would increase if new key generation methods are defined for MPTCP (e.g. using SSL keys as has been proposed).
 3. To include the destination address of the ADD_ADDR msg in the HMAC. This would certainly make the attack harder (the attacker would need to know the key). It wouldn't allow hosts behind NATs to be reached by an address in the ADD_ADDR option, even with static NAT bindings (like a web server at home). Probably it would make sense to combine it option 2) (i.e. to have the HMAC of the address in the ADD_ADDR option and the destination address of the packet).
 4. To include the destination address of the SYN packet in the HMAC of the MP_JOIN message. This has the same problems than option 3) in the presence of NATs.
3. DoS attack on MP_JOIN

Summary of the attack:

Type of attack: MPTCP Denial-of-Service attack, preventing the hosts from creating new subflows.

Type of attacker: Off-path, active attacker

Threat: Low (? - as it is hard to guess the 32-bit token and still then the attacker only prevents the creation of new subflows)

Description:

As currently specified, the initial SYN+MP_JOIN message of the 3-way handshake for additional subflows creates state in the host receiving the message. This, because the SYN+MP_JOIN contains the 32-bit token that allows the receiver to identify the MPTCP-session and the 32-bit random nonce, used in the HMAC calculation. As this information is not resent in the third ACK of the 3-way handshake, a host must create state upon reception of a SYN+MP_JOIN.

Assume that there exists an MPTCP-session between host A and host B, with token Ta and Tb. An attacker, sending a SYN+MP_JOIN to host B, with the valid token Tb, will trigger the creation of state on host B. The number of these half-open connections a host can store per MPTCP-session is limited by a certain number, and it is implementation-dependent. The attacker can simply exhaust this limit by sending multiple SYN+MP_JOINS with different 5-tuples. The (possibly forged) source address of the attack packets will typically correspond to an address that is not in use, or else the SYN/ACK sent by Host B would elicit a RST from the impersonated node, thus removing the corresponding state at Host B. Further discussion of traditional SYN-flood attacks and common mitigations can be found in [RFC4987]

This effectively prevents the host A from sending any more SYN+MP_JOINS to host B, as the number of acceptable half-open connections per MPTCP-session on host B has been exhausted.

The attacker needs to know the token Tb in order to perform the described attack. This can be achieved if it is a partial on-time eavesdropper, observing the 3-way handshake of the establishment of an additional subflow between host A and host B. If the attacker is never on-path, it has to guess the 32-bit token.

Christoph: can you provide text about the birthday paradox and busy servers?

3.1. Possible security enhancements to prevent this attack

The third packet of the 3-way handshake could be extended to contain also the 32-bit token and the random nonce that has been sent in the SYN+MP_JOIN. Further, host B will have to generate its own random nonce in a reproducible fashion (e.g., a Hash of the 5-tuple + initial sequence-number + local secret). This will allow host B to reply to a SYN+MP_JOIN without having to create state. Upon the reception of the third ACK, host B can then verify the correctness of the HMAC and create the state.

4. SYN flooding amplification

Summary of the attack:

Type of attack: The attacker can use the SYN+MP_JOIN messages to amplify the SYN flooding attack.

Type of attacker: Off-path, active attacker

Threat: Medium

Description:

SYN flooding attacks [RFC4987] use SYN messages to exhaust the server's resources and prevent new TCP connections. A common mitigation is the use of SYN cookies [RFC4987] that allow the stateless processing of the initial SYN message.

With MPTCP, the initial SYN can be processed in a stateless fashion using the aforementioned SYN cookies. However, as we described in the previous section, as currently specified, the SYN+MP_JOIN messages are not processed in a stateless manner. This opens a new attack vector. The attacker can now open a MPTCP session by sending a regular SYN and creating the associated state but then send as many SYN+MP_JOIN messages as supported by the server with different source address source port combinations, consuming server's resources without having to create state in the attacker. This is an amplification attack, where the cost on the attacker side is only the cost of the state associated with the initial SYN while the cost on the server side is the state for the initial SYN plus all the state associated to all the following SYN+MP_JOIN.

4.1. Possible security enhancements to prevent this attack

1. The solution described for the previous DoS attack on MP_JOIN would also prevent this attack.
2. Limiting the number of half open subflows to a low number (like 3) would also limit the impact of this attack.

5. Eavesdropper in the initial handshake

Summary of the attack

Type of attack: An eavesdropper present in the initial handshake where the keys are exchanged can hijack the MPTCP session at any time in the future.

Type of attacker: a Partial-time On-path eavesdropper

Threat: Low

Description:

In this case, the attacker is present along the path when the initial 3-way handshake takes place, and therefore is able to learn the keys used in the MPTCP session. This allows the attacker to move away from the MPTCP session path and still be able to hijack the MPTCP session in the future. This vulnerability was readily identified at

the moment of the design of the MPTCP security solution and the threat was considered acceptable.

5.1. Possible security enhancements to prevent this attack

There are many techniques that can be used to prevent this attack and each of them represents different tradeoffs. At this point, we limit ourselves to enumerate them and provide useful pointers.

1. Use of hash-chains. The use of hash chains for MPTCP has been explored in [hash-chains]
2. Use of SSL keys for MPTCP security as described in [I-D.paasch-mptcp-ssl]
3. Use of Cryptographically-Generated Addresses (CGAs) for MPTCP security. CGAs [RFC3972] have been used in the past to secure multi addressed protocols like SHIM6 [RFC5533].
4. Use of TCPCrypt [I-D.bittau-tcp-crypt]
5. Use DNSSEC. DNSSEC has been proposed to secure the Mobile IP protocol [dnssec]

6. Security considerations

This whole document is about security considerations for MPTCP.

7. IANA Considerations

There are no IANA considerations in this memo.

8. Acknowledgments

We would like to thank Mark Handley for his comments on the attacks and countermeasures discussed in this document.

9. References

9.1. Normative References

- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
- [RFC3972] Aura, T., "Cryptographically Generated Addresses (CGA)", RFC 3972, March 2005.

- [RFC5961] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", RFC 5961, August 2010.
- [RFC6056] Larsen, M. and F. Gont, "Recommendations for Transport-Protocol Port Randomization", BCP 156, RFC 6056, January 2011.
- [RFC6528] Gont, F. and S. Bellovin, "Defending against Sequence Number Attacks", RFC 6528, February 2012.

9.2. Informative References

- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, January 2013.
- [RFC6181] Bagnulo, M., "Threat Analysis for TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6181, March 2011.
- [RFC5533] Nordmark, E. and M. Bagnulo, "Shim6: Level 3 Multihoming Shim Protocol for IPv6", RFC 5533, June 2009.
- [RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.
- [RFC3775] Johnson, D., Perkins, C., and J. Arkko, "Mobility Support in IPv6", RFC 3775, June 2004.
- [RFC2827] Ferguson, P. and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", BCP 38, RFC 2827, May 2000.
- [I-D.ietf-savi-framework]
Wu, J., Bi, J., Bagnulo, M., Baker, F., and C. Vogt,
"Source Address Validation Improvement Framework", draft-ietf-savi-framework-06 (work in progress), January 2012.
- [RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, August 2007.
- [I-D.paasch-mptcp-ssl]
Paasch, C. and O. Bonaventure, "Securing the MultiPath TCP handshake with external keys", draft-paasch-mptcp-ssl-00 (work in progress), October 2012.
- [I-D.bittau-tcp-crypt]

Bittau, A., Boneh, D., Hamburg, M., Handley, M., Mazieres, D., and Q. Slack, "Cryptographic protection of TCP Streams (tcpcrypt)", draft-bittau-tcp-crypt-03 (work in progress), September 2012.

[hash-chains]

Diez, J., Bagnulo, M., Valera, F., and I. Vidal, "Security for multipath TCP: a constructive approach", International Journal of Internet Protocol Technology 6, 2011.

[dnssec]

Kukec, A., Bagnulo, M., Ayaz, S., Bauer, C., and W. Eddy, "OAM-DNSSEC: Route Optimization for Aeronautical Mobility using DNSSEC", 4th ACM International Workshop on Mobility in the Evolving Internet Architecture MobiArch 2009, 2009.

Authors' Addresses

Marcelo Bagnulo
Universidad Carlos III de Madrid
Av. Universidad 30
Leganes, Madrid 28911
SPAIN

Phone: 34 91 6249500
Email: marcelo@it.uc3m.es
URI: <http://www.it.uc3m.es>

Christoph Paasch
UCLouvain
Place Sainte Barbe, 2
Louvain-la-Neuve, 1348
Belgium

Email: christoph.paasch@uclouvain.be

Fernando Gont
SI6 Networks / UTN-FRH
Evaristo Carriego 2644
Haedo, Provincia de Buenos Aires 1706
Argentina

Phone: +54 11 4650 8472
Email: fgont@si6networks.com
URI: <http://www.si6networks.com>

Olivier Bonaventure
UCLouvain
Place Sainte Barbe, 2
Louvain-la-Neuve, 1348
Belgium

Email: olivier.bonaventure@uclouvain.be

Costin Raiciu
Universitatea Politehnica Bucuresti
Splaiul Independentei 313a
Bucuresti
Romania

Email: costin.raiciu@cs.pub.ro

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 13, 2014

P. Eardley
BT
July 12, 2013

Survey of MPTCP Implementations
draft-eardley-mptcp-implementations-survey-02

Abstract

This document presents results from the survey to gather information from people who have implemented MPTCP, in particular to help progress the protocol from Experimental to Standards track.

The document currently includes answers from four teams: a Linux implementation from UCLouvain, a FreeBSD implementation from Swinburne, an anonymous implementation in a commercial OS, and a NetScaler Firmware implementation from Citrix Systems, Inc. Thank-you!

In summary, we have four independent implementations of all the MPTCP signalling messages, with the exception of address management, and some interoperability testing has been done by the other three implementations with the 'reference' Linux implementation. So it appears that the RFC is (at least largely) clear and correct. On address management, we have only one implementation of ADD_ADDR with two teams choosing not to implement it. We have one implementation of the working group's coupled congestion control (RFC6356) and none of the MPTCP-aware API (RFC6897).

The main suggested improvements are around

- o how MPTCP falls back (if the signalling is interrupted by a middlebox): (1) corner cases that are not handled properly, (2) at the IETF, the MPTCP community should work with middlebox vendors, either to reduce or eliminate the need for fallback or to understand the middlebox interactions better.
- o security: both better MPTCP security (perhaps building on SSL) and a lighter weight mechanism, preferably both in one mechanism.

It is hoped that the next version can include information from any other implementations. If you are an implementer and want to contribute your answers, please see the -01 version of this document for a blank survey ready to be filled in.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 13, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	4
2. Survey - summary of replies	4
3. Interesting aspects of replies	6
3.1. Question 1: Your details	6
3.2. Question 2: Preliminary information about your implementation	7
3.3. Question 3: Support for MPTCP's Signalling Functionality	7
3.4. Question 4: Fallback from MPTCP	7
3.5. Question 5: Heuristics	8
3.6. Question 6: Security	9
3.7. Question 7: IANA	9
3.8. Question 8: Congestion control and subflow policy	9
3.9. Question 9: API	10
3.10. Question 10: Deployments, use cases and operational experiences	10
3.11. Question 11: Improvements to RFC6824	11
4. IANA Considerations	11
5. Security Considerations	11
6. Acknowledgements	11
7. Full survey response for Implementation 1	11
8. Full survey response for Implementation 2	19
9. Full survey response for Implementation 3	23
10. Full survey response for Implementation 4	31
11. Normative References	38
Author's Address	38

1. Introduction

The document reports the results from a survey to gather information from people who have implemented MPTCP. The goal is to help progress the protocol from Experimental to Standards track.

Four responses have been received. Thank-you! They are independent implementations:

- o the Linux implementation from UCLouvain,
- o the FreeBSD implementation from Swinburne
- o an anonymous implementation in a commercial OS
- o a NetScaler Firmware implementation from Citrix Systems, Inc.

The Table below presents a highly-compressed summary, with each row corresponding to one question or sub-question of the survey. The following section highlights some interesting aspects of the replies in less compressed form. The full survey responses are in Appendix A, B, C and D.

It is hoped that the next version of this document can include information about a further (independent) implementation:

- o Georg Hampel's user-space implementation (publicly available but not longer maintained)
- o any other implementations.

2. Survey - summary of replies

The Table below presents a highly-compressed summary, with each row corresponding to one question or sub-question of the survey. A column is left blank for any future responses.

-----+-----					
Institution	1 UCLouvain	2 Swinburne	3 Anon	4 Citrix	
Question 2 asks about some preliminary topics, including whether the implementation is publicly available and interoperability with the Linux implementation (#1).					
OS	UCLouvain Linux	Swinburne FreeBSD-10	Anon Commercial	Citrix NetScaler	
v4 & v6	Both	IPv4	Both	Both	

public	Yes	Yes	No	Yes (pay)	
independent	Yes	Yes	Yes	Yes	
interop	Yes(!)	Mostly	Mostly	Yes	
Question 3: Support for MPTCP's signalling functionality MPTCP's signalling messages are: MP_CAPABLE, MP_JOIN, Data transfer (DSS), ADD_ADDR, REMOVE_ADDR, MP_FASTCLOSE. There are sub-questions for MP_JOIN and DSS.					
	UCLouvain	Swinburne	Anon	Citrix	
MP_CAPABLE	Yes	Yes	Yes	Yes	
MP_JOIN	Yes	Yes	Yes	Yes	
initiated by	first end	either end	first end	first end	
#subflows	32	8	no limit	6	
DSS	Yes	Yes	Yes	Yes	
DATA_ACK	4 bytes	4 or 8 byte	4 or 8 byte	4 or 8 byte	
Data seq num	4 bytes	4 or 8 byte	4 or 8 byte	4 or 8 byte	
DATA_FIN	Yes	Yes	Yes	Yes	
Checksum	Yes	No	Yes	Yes	
ADD_ADDR	Yes	No	No (never)	No (never?)	
REMOVE_ADDR	Yes	No	Partly	Yes	
FAST_CLOSE	Yes	No	Yes	Yes	
Question 4 asks about fallback from MPTCP: if a middlebox mangles MPTCP's signalling by removing MP_CAPABLE, MP_JOIN, DSS or DATA_ACK; if data is protected with Checksum in DSS option; if fallback to TCP uses an infinite mapping; and if any corner cases have been found.					
	UCLouvain	Swinburne	Anon	Citrix	
MP_CAPABLE	Yes	Yes	Yes	Yes	
MP_JOIN	Yes	Yes	Yes	Yes	
DSS	Yes	No	Yes	Yes	
DATA_ACK	Yes	No	No		
Checksum	Yes	No	Yes	Yes	
infinite map	Yes	Yes	Yes	Yes	
corner cases	No		Yes	Yes	
Question 5 asks about heuristics: aspects that are not required for protocol correctness but impact the performance. Questions are about sized the receiver and sender buffers, re-transmission policy, if additional subflows use the same port number as for the first subflow					
	UCLouvain	Swinburne	Anon	Citrix	
Recv buffer	auto-tune	TCP_MAXWIN	no tuning	tuned	
Sendr buffer	auto-tune	cwnd	no tuning	as TCP	
Re-transmits	2nd subflow	2nd subflow	2nd subflow	1st subflow	
Port usage	same ports	same ports	diff local		
Question 6 asks about what security mechanisms are implemented: the one defined in RFC6824 and any others.					
	UCLouvain	Swinburne	Anon	Citrix	

HMAC-SHA1	Yes	Yes	Yes	Yes	
other	Yes	No	No	No	
Question 7 asks whether the implementation follows the IANA-related definitions (for TCP Option Kind and sub-registries).					
	UCLouvain	Swinburne	Anon	Citrix	
RFC6824	Yes	Yes	Yes	Yes	
Question 8 asks about congestion control and related issues: how traffic is shared across multiple subflows; support for 'handover'; and support of RFC6356 (or other) coupled congestion control.					
	UCLouvain	Swinburne	Anon	Citrix	
sharing	shared, RTT	shared	active/back	active/back	
handover	Yes		Yes	Yes	
coupled cc	Yes	No	No	No	
other ccc	Yes, OLIA	No	No	No	
MP-PRIO & B	Yes	No	Yes	Yes	
Question 9 is about the API: how legacy applications interact with the MPTCP stack, and if implemented the RFC6897 API for MPTCP-aware applications.					
	UCLouvain	Swinburne	Anon	Citrix	
legacy apps	default	sysctl	private API	configured	
MPTCP API	No	No	No	No	
advanced API	No	No	No	No	
Question 10 gathers some limited information about operational experiences and deployments.					
	UCLouvain	Swinburne	Anon	Citrix	
Scenario	several	several	mobile	proxy	
environment	internet	controlled	internet	internet	
ends / proxy	end hosts	end hosts	end hosts	proxy	

3. Interesting aspects of replies

This section tries to highlight some interesting comments made in the surveys. The Appendices can be consulted for further details.

3.1. Question 1: Your details

Implementation 1 has been implemented by Sebastien Barre, Christoph Paasch and a large team, mainly at UCLouvain. Implementation 2 has been implemented by Lawrence Stewart and Nigel Williams at Swinburne University of Technology. Both these implementations are publicly available. Implementation 3 comes from an anonymous team with a

commercial OS. Implementation 4 comes from Citrix Systems, Inc.

3.2. Question 2: Preliminary information about your implementation

Three of the four implementations are publicly available, two for free (under GPLv2 and BSD licences) and one for a fee (NetScaler Firmware). Implementation 3 (commercial OS) is planned for use in a mobile environment, with MPTCP is used in active/backup mode.

All implementations support IPv4 and three of four support IPv6.

All implementations are being actively worked on, in order to improve performance and stability and conformance with the RFC.

3.3. Question 3: Support for MPTCP's Signalling Functionality

Three of the four implementations have implemented all the MPTCP signalling, with the interesting exception of address management, whilst Implementation 2 plans to add support for all those signalling capabilities it does not yet support.

On address management, two implementations have decided not to implement ADD_ADDR. (ADD_ADDR allows an MPTCP host to signal a new address explicitly to the other host to allow it to initiate a new subflow - as an alternative to using MP_JOIN to directly start a new subflow). Implementation 3 decided not to support sending ADD_ADDR or processing ADD_ADDR as it is considered a security risk. Implementation 4 decided not to support ADD_ADDR because it didn't think it would be useful as most clients are behind NATing devices. However, both implemented REMOVE_ADDR (in Implementation 3 the client can send a REMOVE_ADDR but ignores incoming REMOVE_ADDR).

In Implementations 1, 3 and 4 only the initiator of the original subflow can start a new subflow (a reason mentioned is that NATs make it hard for the server to reach the client).

All implementations support 4 bytes "Data ACK" and "Data sequence number" fields, and will interoperate with an implementation sending 8 bytes. Implementation 1 uses only 4 bytes fields; if an implementation sends an 8 byte data sequence number it replies with a 4 byte data ack.

3.4. Question 4: Fallback from MPTCP

Question 4 asks about action when there is a problem with MPTCP, for example due to a middlebox mangling MPTCP's signalling. The connection needs to fall back: if the problem is on the first subflow then MPTCP falls back to TCP, whilst if the problem is on an

additional subflow then that subflow is closed with a TCP RST, as discussed in [Section 3.6 RFC6824].

Implementations 3 and 4 made several comments about fallback.

Implementation 3 suggests that both sender and receiver behaviours could be outlined with more detail, in particular when DSS checksum is not in use and the MPTCP options are stripped. Implementation 3 falls back to TCP when there's one sub flow, but not when there are multiple sub flows (MPTCP is used in active/backup mode, and it is assumed that the sub flow transferring data is most likely to be more usable than any other established sub flow, hence the sub flow on which fallback occurred is kept alive and other sub flows are closed).

Implementation 4 found a corner case where it is not clear what to do: if a pure ack or data packet without DSS is received in middle of transaction (which can happen if the routing changes and the new path drops MPTCP options). Also, Implementation 4 suggests that clarifying whether the infinite map exchange is unidirectional or bidirectional.

Implementation 1 has developed a publicly available test suite that tests MPTCP's traversal of middleboxes.

3.5. Question 5: Heuristics

Question 5 gathers information about heuristics: aspects that are not required for protocol correctness but impact the performance. We would like to document best practice so that future implementers can learn from the experience of pioneers.

There are several differences between the implementations.

For receiver buffer, Implementation 1 uses a slightly modified version of Linux's auto-tuning algorithm; Implementation 2 determines the receiver buffer by using "TCP_MAXWIN << tp->rcv_scale" (this is a temporary measure); Implementation 3 uses MPTCP in active/backup mode, so the receive buffer sizes at the MPTCP and subflow level is the same (automatic buffer tuning is turned off); Implementation 4 varies the receiver buffer size based on the services and application type.

For the sender buffer, Implementation 1 uses Linux auto-tuning, Implementation 2 scales based on occupancy, whilst Implementation 3 turns off automatic buffer tuning, and Implementation 4 uses MPTCP-level (sub)flow control that is (almost) the same as regular TCP flow control.

Implementations 1, 2 and 3 re-transmit unacknowledged data on a different subflow (and not the same subflow), whilst Implementation 4 re-transmits on original subflow for 3 RTOs and then uses another subflow.

For port usage, Implementations 1 and 2 uses the same ports for the additional subflows, whilst Implementation 3 uses the same destination port but a different local port, so that on the wire it looks like two connections to the same remote destination.

Implementation 4 suggests that the RFC should more clearly /extensively define failure cases and how to handle unexpected signals.

3.6. Question 6: Security

Question 6 asks about security related matters.

All Implementations have implemented the hash-based, HMAC-SHA1 security mechanism defined in [RFC6824]. Implementation 3 suggests that a more secure mechanism could be tied with SSL. Implementation 4 suggests that a more secure and lightweight mechanism is needed, as keys are exchanged (in the MP_CAPABLE option) in plain text and the key generation mechanism is highly computational intensive. Implementation 1 has implemented two additional mechanisms in a separate Linux branch - one lightweight and the other SSL-based.

3.7. Question 7: IANA

All Implementations have followed the IANA-related definitions [Section 8 RFC6824] for: TCP Option Kind number (30); the sub-registry for "MPTCP Option Subtypes"; and the sub-registry for "MPTCP Handshake Algorithms".

3.8. Question 8: Congestion control and subflow policy

Question 8 asks how is shared across multiple subflows.

Implementation 1 has added support for coupled congestion control (both that defined in [RFC6356] and in OLIA, draft-khalili-mptcp-congestion-control. The other implementations do not include coupled congestion control. Whilst Implementation 2 plans to add it (currently it uses a simple algorithm spreads traffic across the subflows), Implementations 3 and 4 do not plan to add coupled congestion control - they use one subflow at a time, with others as a backup. Implementation 3 believes it is not currently useful to share load across all network interfaces on a mobile node, as the interfaces have different characteristics for cost, bring-up

and power usage. They have both found the B bit (in MP-JOIN) and MP-PRIO option very useful for this active /backup operation.

Implementation 2 is also interested in experimenting with congestion control across paths with different path-cost metrics.

3.9. Question 9: API

Question 9 gathers information about the API. None have implemented the [RFC6897] "basic MPTCP API" for MPTCP-aware applications. For three implementations MPTCP is used for all applications (set by configuration), whilst Implementation 3 uses a private API that allows MPTCP to be used on a per application basis.

3.10. Question 10: Deployments, use cases and operational experiences

Question 10 takes the opportunity of this survey to gather some limited information about operational experiences and deployments.

The Implementations mention different use cases.

Implementation 2 is interested in using MPTCP for several use cases: vehicle to infrastructure (V2I) connectivity (to provide a persistent connection using 3G and roadside wifi); multi-homed "home-user" environments; high throughput data transfers. Implementation 3 is interested in the mobile scenario, with MPTCP providing an active /backup mode so achieving session continuity across changing network environments. Implementation 4 is interested in MPTCP giving reliability and fault tolerance via a proxy. Implementation 1 already uses MPTCP on www.multipath-tcp.org and for internal ssh servers at UCLouvain.

Implementation 4 uses a proxy (MPTCP connections from a client are terminated and the TCP connection established on the other side), whilst the other Implementations are on end hosts. Implementation 2 is so far within controlled testbeds, whilst Implementation 3 is on the Internet.

Implementation 2 is currently an alpha-quality build, so limited testing so far.

Implementation 3 suggests working at the IETF with firewall vendors, to get them to change their defaults to allow MPTCP signals. This would also reduce the "over-engineering" needed to handle fallback cases. Implementation 1 suggests retrieving logs from middleboxes, as the best approach to understanding the interactions of MPTCP signalling with middleboxes.

Implementation 3 discusses a scenario that should be handled better. A backup subflow may never sent data. If the initial subflow fails, data is retransmitted on the backup subflow, but that path has a middlebox stripping options. Then it may not be possible to recover the MPTCP session.

3.11. Question 11: Improvements to RFC6824

Question 11 asks if there are any areas where RFC6824 could be improved. The main topics have been mentioned earlier:

- o fallback: the need for more clarity in the fallback cases is mentioned by Implementations 3 and 4.
- o security: the need for both a more secure and a more lightweight mechanism is mentioned.

Implementation 3 also suggests several potential improvements, which are outside the scope of RFC6824: support for sub flow level automatic buffer scaling, varying QoS support, and varying window scaling support on each sub flow; also, additional work on option signalling will be brought up in future discussions.

4. IANA Considerations

This document makes no request of IANA.

5. Security Considerations

This survey does not impact the security of MPTCP, except to the extent that it uncovers security issues that can be tackled in a future version of the protocol.

6. Acknowledgements

Many thanks to the people who replied to the survey: Christoph Paasch, Nigel Williams, anon, and Krishna Khanal. Very many thanks to all of the teams who actually did the implementation and testing and are continuing to improve them.

7. Full survey response for Implementation 1

Question 1: Your details Question 1 gathers some information about the team that has implemented MPTCP.

1. Your institution: UCLouvain, IP Networking Lab
(<http://inl.info.ucl.ac.be>)

2. Name(s) of people in your implementation and test teams: Initial design from Sebastien Barre. Since then, numerous code-contributors (ordered by number of commits): Christoph Paasch (UCLouvain) Gregory Detal (UCLouvain) Jakko Korkeaniemi (Aalto University) Mihai P. Andrei (Intel) Fabien Duchene (UCLouvain) Andreas Seelinger (RWTH Aachen) Stefan Sicleru (Intel) Lavkesh Lahngir Catalin Nicutar (PUB Bucharest) Andrei Maruseac (Intel) Andreas Ripke (NEC) Vlad Dogaru (Intel) Octavian Purdila (Intel) Niels Laukens (VRT Belgium) John Ronan (TSSG) Brandon Heller (Stanford University) Conformance Testing: Yvan Coene (UCLouvain)

3. Do you want your answers to Question 1.1 and 1.2 above to be anonymised? No.

3.2. Question 2: Preliminary information about your implementation
Question 2 gathers some preliminary information.

1. What OS is your implementation for? (or is it application layer?)
Linux Kernel.

2. Do you support IPv4 or IPv6 addresses or both? We support both.

3. Is it publicly available (or will it be?) (for free or a fee?)
Publicly available (GPLv2) at www.multipath-tcp.org

4. Overall, what are you implementation and testing plans? (details can be given against individual items later) We plan to continue to align our implementation with the IETF specifications and improve its performance and stability.

5. Is it an independent implementation? Or does it build on another MPTCP implementation -which one? Independent implementation.

6. Have you already done some interop tests, for example with UCLouvain's "reference" Linux implementation? /

7. Would you be prepared to take part in an interop event, for example adjacent to IETF-87 in Berlin? Yes. We are also ready to help in organising such an event if needed.

3.3. Question 3: Support for MPTCP's Signalling Functionality
Question 3 asks about support for the various signalling messages that the MPTCP protocol defines. *** For each message, please give a little information about the status of your implementation: for example, you may have implemented it and fully tested it; the

implementation may be in progress; you have not yet implemented it but plan to soon (timescale?); you may have no intention to implement it (why?); etc.

1. Connection initiation (MP_CAPABLE) [Section 3.1 RFC6824]

a. What is the status of your implementation? Fully support the MP_CAPABLE exchange.

b. Any other comments or information? We generate the random key as a hash of the 5-tuple, sequence number and a local secret. This significantly improves the performance, instead of using a pseudo-random number generator. The performance benefit has been shown during IETF85
<http://tools.ietf.org/agenda/85/slides/slides-85-mptcp-2.pdf>

2. Starting a new subflow (MP_JOIN) [Section 3.2 RFC6824]

a. What is the status of your implementation? Fully support the MP_JOIN exchange.

b. Can either end of the connection start a new subflow (or only the initiator of the original subflow)? Currently, only the initiator of the original subflow starts a new subflow. Given the widespread deployment of NATs, it is often difficult for the server to reach the client. This is the main reason why the server currently does not start new subflows in our implementation. But, the initiator would accept a SYN+MP_JOIN if sent by another implementation.

c. What is the maximum number of subflows your implementation can support? Currently 32.

d. Any other comments or information?

3. Data transfer (DSS) [Section 3.3 RFC6824]

a. What is the status of your implementation? Fully working implementation of data transfer.

b. The "Data ACK" field can be 4 or 8 octets. Which one(s) have you implemented? We use 4 bytes for the DATA-ACK field.

c. The "Data sequence number" field can be 4 or 8 octets. Which one(s) have you implemented? We use 4 bytes for the data sequence number.

d. Does your implementation support the "DATA_FIN" operation to close an MPTCP connection? Yes.

e. Does your implementation support the "Checksum" field (which is negotiated in the MP_CAPABLE handshake)? Yes. This is configurable via a sysctl.

f. Any other comments or information? We support interoperability with implementations that do send 64-bit data sequence numbers and data acks. However, even if the peer sends 64-bit data sequence numbers, we will only reply with a 32-bit data-ack. We do not have heuristics to trigger the sending of DATA_ACKs. We simply send the DATA_ACK in each packet.

4. Address management (ADD_ADDR and REMOVE_ADDR) [Section 3.4 RFC6824]

a. What is the status of your implementation? We support ADD_ADDR/REMOVE_ADDR messages.

b. Can your implementation do ADD_ADDRESS for addresses that appear *after* the connection has been established? Yes, as shown in: "Exploring Mobile/WiFi Handover with Multipath TCP", C. Paasch et. al, ACM SIGCOMM workshop on Cellular Networks (Cellnet'12), 2012.

c. Any other comments or information? We do not send out TCP keepalive-messages upon the reception of a REMOVE_ADDR-message.

5. Fast close (MP_FASTCLOSE) [Section 3.5 RFC6824]

a. What is the status of your implementation? We support the MP_FASTCLOSE implementation.

b. Any other comments or information?

3.4. Question 4: Fallback from MPTCP Question 4 asks about action when there is a problem with MPTCP, for example due to a middlebox mangling MPTCP's signalling. The connection needs to fall back: if the problem is on the first subflow then MPTCP falls back to TCP, whilst if the problem is on an additional subflow then that subflow is closed with a TCP RST, as discussed in [Section 3.6 RFC6824].

1. If the MP_CAPABLE option is removed by a middlebox, does your implementation fall back to TCP? Yes.

2. If the MP_JOIN option does not get through on the SYNs, does your implementation close the additional subflow? Yes.

3. If the DSS option does not get through on the first data segment(s), does your implementation fall back? (either falling back to MPTCP (if the issue is on the first subflow) or closing the

additional subflow (if the issue is on an additional subflow)) Yes. On the initial subflow we do a seamless fallback, additional subflows will be closed by a RST.

4. Similarly, if the "DATA ACK" field does not correctly acknowledge the first data segment(s), does your implementation fall back? Yes. Same as above.

5. Does your implementation protect data with the "Checksum" field in the DSS option [Section 3.3 RFC6824]? If the checksum fails (because the subflow has been affected by a middlebox), does your implementation immediately close the affected subflow (with a TCP RST) with the MP_FAIL Option? If the checksum fails and there is a single subflow, does your implementation handle this as a special case, as described in [Section 3.6 RFC6824]? Yes, we support the DSS-checksum. If the checksum is wrong and there exist other subflows, we close the current subflow with an RST. If there is no other subflow, we send an ACK + MP_FAIL and do a fallback to infinite mapping. This fallback has successfully been tested with different type of NAT middleboxes, while using FTP.

6. Does your implementation fall back to TCP by using an "infinite mapping" [Section 3.3.1 RFC6824] (so that the subflow-level data is mapped to the connection-level data for the remainder of the connection)? Yes.

7. Did you find any corner cases where MPTCP's fallback didn't happen properly? No. We have developped a test-suite to test the middlebox-traversal of MPTCP, available at <http://multipath-tcp.org/pmwiki.php/Users/AboutMeasures>

8. Any other comments or information about fallback?

3.5. Question 5: Heuristics Question 5 gathers information about heuristics: aspects that are not required for protocol correctness but impact the performance. We would like to document best practice so that future implementers can learn from the experience of pioneers. The references contain some initial comments about each topic.

1. Receiver considerations [S3.3.4, RFC6824]: What receiver buffer have you used? Does this depend on the retransmission strategy? What advice should we give about the receiver? Linux includes an autuning algorithm for the TCP receiver buffer. This algorithm has been slightly modified for Multipath TCP. The receive-buffer does not depend on the retransmission strategy.

2. Sender considerations [S3.3.5, RFC6824]: How do you determine how

much data a sender is allowed to send and how big the sender buffer is? What advice should we give about the sender? The send-buffer is autotuned similarly as the receive-buffer (see above). We send as much data as possible, filling the congestion windows of each subflow. The sender deploys the "Opportunistic Retransmission" and "Penalization" algorithms from the paper: "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP", C. Raiciu et. al, NSDI 2012.

3. Reliability and retransmissions [S3.3.6, RFC6824]: What is your retransmission policy? (when do you retransmit on the original subflow vs on another subflow or subflows?) When do you decide that a subflow is underperforming and should be reset, and what do you then do? What advice should we give about this issue? Upon an RTO on subflow A, we reinject all the unacknowledged data of subflow A on another subflows. We do not currently have a mechanism to detect that a subflow is underperforming.

4. Port usage [S3.3.8.1, RFC6824]: Does your implementation use the same port number for additional subflows as for the first subflow? Have you used the ability to define a specific port in the Add Address option? What advice should we give about this issue? We always use the same port number as for the first subflow. Except, if the ADD_ADDRESS option that has been received contained a specific port. We do not have a means to configure the specific port in the ADD_ADDRESS option, but we support reception of the port.

5. Delayed subflow start [S3.3.8.2, RFC6824]: What factors does your implementation consider when deciding about opening additional subflows? What advice should we give about this issue? As soon as we are sure that the initial subflow is fully MPTCP-capable (reception of a DATA_ACK), we create a full mesh among all IP-addresses between the two hosts. We do not explicitly delay the creation of new subflows.

6. Failure handling [S3.3.8.3, RFC6824]: Whilst the protocol defines how to handle some unexpected signals, the behaviour after other unexpected signals is not defined. What advice should we give about this issue? We did not implement the caching mentioned in Section 3.8.3.

7. Use of TCP options: As discussed in [Appendix A, RFC6824], the TCP option space is limited, but a brief study found there was enough room to fit all the MPTCP options. However there are constraints on which MPTCP option(s) can be included in packets with other TCP options - do the suggestions in Appendix A need amending or expanding? We do not implement specific heuristics to reduce the TCP option-space usage. If timestamp is enabled we will only be able to

send two SACK-blocks, because the DATA_ACK consumes the remaining bytes.

8. What other heuristics should we give advice about? Any other comments or information?

3.6. Question 6: Security Question 6 asks about Security related matters [Section 5 RFC6824].

1. Does your implementation use the hash-based, HMAC-SHA1 security mechanism defined in [RFC6824]? Yes.

2. Does your implementation support any other handshake algorithms? We have in a separate branch, an implementation of draft-paasch-mptcp-lowoverhead and draft-paasch-mptcp-ssl.

3. It has been suggested that a Standards-track MPTCP needs a more secure mechanism. Do you have any views about how to achieve this? We believe that the solution described in draft-paasch-mptcp-ssl would be a good starting point since it leverages the security of the upper layer.

4. Any other comments or information?

3.7. Question 7: IANA Question 7 asks about IANA related matters.

1. Does your implementation follow the IANA-related definitions? [Section 8 RFC6824] defines: TCP Option Kind number (30); the sub-registry for "MPTCP Option Subtypes"; and the sub-registry for "MPTCP Handshake Algorithms" Yes.

2. Any other comments or information?

3.8. Question 8: Congestion control and subflow policy Question 8 asks about how you share traffic across multiple subflows.

1. How does your implementation share traffic over the available paths? For example: as a spare path on standby ('all-or-nothing'), as an 'overflow', etc? Does it have the ability to send /receive traffic across multiple subflows simultaneously? The implementation is able to send and receive traffic on all subflows simultaneously. Our scheduler first tries to send traffic on the subflow with the lowest RTT. As this subflow's congestion window is full, we pick the subflow with the next lower RTT.

2. Does your implementation support "handover" from one subflow to another when losing an interface? Yes, as described in: "Exploring Mobile/WiFi Handover with Multipath TCP", C. Paasch et. al, ACM

SIGCOMM workshop on Cellular Networks (Cellnet'12), 2012.

3. Does your implementation support the coupled congestion control defined in [RFC6356]? Yes.

4. Does your implementation support some other coupled congestion control (ie that balances traffic on multiple paths according to feedback)? We also support the OLIA congestion control (draft-khalili-mptcp-congestion-control-00).

5. The MP_JOIN (Starting a new subflow) Option includes the "B" bit, which allows the sender to indicate whether it wishes the new subflow to be used immediately or as a backup if other path(s) fail. The MP_PRIO Option is a request to change the "B" bit - either on the subflow on which it is sent, or (by setting the optional Address ID field) on other subflows. Does your implementation support the "B" bit and MP_PRIO mechanisms? Do you think they're useful, or have another suggestion? Yes, we support the "B"-bit of the MP_JOIN and the MP_PRIO option. It is configurable on a per-interface basis. Experiences with the "B"-bit can be found in our paper: "Exploring Mobile/WiFi Handover with Multipath TCP", C. Paasch et. al, ACM SIGCOMM workshop on Cellular Networks (Cellnet'12), 2012.

6. Any other comments or information or suggestions about the advice we should give about congestion control [S3.3.7 RFC6824] and subflow policy [S3.3.8 RFC6824]?

3.9. Question 9: API Question 9 gathers information about your API. [RFC6897] considers the MPTCP Application Interface.

1. With your implementation, can legacy applications use (the existing sockets API to use) MPTCP? How does the implementation decide whether to use MPTCP? Should the advice in [Section 4, RFC6897] be modified or expanded? Yes, a standard TCP socket API can be used. By default MPTCP is enabled on all connections.

2. The "basic MPTCP API" enables MPTCP-aware applications to interact with the MPTCP stack via five new socket options. For each one, have you implemented it? has it been useful? None of them are part of the current stable release MPTCP v0.86.
<http://multipath-tcp.org/pmwiki.php?n=Main.Release86> a. TCP_MULTIPATH_ENABLE? b. TCP_MULTIPATH_ADD? c. TCP_MULTIPATH_REMOVE? d. TCP_MULTIPATH_SUBFLOWS? e. TCP_MULTIPATH_CONNNID?

3. Have you implemented any aspects of an "advanced MPTCP API"? ([Appendix A, RFC6897] hints at what it might include.) No.

4. Any other comments or information?

3.10. Question 10: Deployments, use cases and operational experiences Question 10 takes the opportunity of this survey to gather some limited information about operational experiences and deployments. Any very brief information would be appreciated, for example: 1. What deployment scenarios are you most interested in? 2. Is your deployment on "the Internet" or in a controlled environment? 3. Is your deployment on end hosts or with a MPTCP-enabled proxy (at one or both ends)? 4. What do you see as the most important benefits of MPTCP in your scenario(s)? 5. How extensively have you deployed and experimented with MPTCP so far?

Our implementation is open-source and has been discussed for various types of tests/deployments based on the messages received on the mptcp-dev mailing list. We currently use Multipath TCP on www.multipath-tcp.org and also on internal ssh servers at UCLouvain. 6. MPTCP's design seeks to maximise the chances that the signalling works through middleboxes. Did you find cases where middleboxes blocked MPTCP signalling? We have implemented a test suite based on a slightly modified version of the Multipath TCP implementation that allows to check the interoperability between Multipath TCP and middleboxes. We have used it over Internet paths and identified some potential problems. However, the best approach to test these interactions would be to control the middlebox and analyse its logs during the Multipath TCP test. The test suite can be retrieved from <http://multipath-tcp.org/pmwiki.php/Users/AboutMeasures>

7. MPTCP's design seeks to ensure that, if there is a problem with MPTCP signalling, then the connection either falls back to TCP or removes the problematic subflow. Did you find any corner cases where this didn't happen properly? See above.

8. Have you encountered any issues or drawbacks with MPTCP?

9. Any other comments or information?

3.11. Question 11: Improvements to RFC6824

1. Are there any areas where [RFC6824] could be improved, either in technical content or clarity? 2. Any other issues you want to raise?

8. Full survey response for Implementation 2

Question 1: Your details

1.1 Swinburne University of Technology, Hawthorn, Victoria, Australia

1.2 Lawrence Stewart, Nigel Williams

1.3 No

Question 2: Preliminary information about your implementation

2.1 FreeBSD-10

2.2 Currently IPv4 only (IPv6 support will eventually be added)

2.3 Publicly available (<http://caia.swin.edu.au/urp/newtcp/mptcp/>).
The code is released under the BSD license. 2.3

2.5 Independent

2.6 Yes, some limited testing to establish interoperability.

2.7 Yes, with some additional work this should be possible (if not then IETF-88). Q

Question 3: Support for MPTCP's Signaling Functionality

3.1 a) MP_CAPABLE Implemented

b) Do not currently honour checksum flag (to be implemented)

3.2 a) MP_JOIN Implemented

b) Either end can initiate a MP_JOIN

c) 8 (controlled via sysctl)

d) Currently do not include HMAC verification during handshake, but this will be enabled in the next patch (several weeks from time of submission)

3.3 a) DSS Implemented

b) 4 (default) and 8

c) 4 (default) and 8

d) Yes, however the connection tear-down exchange is not fully implemented - the connection shuts down but the DFIN may not be

correctly acknowledged.

e) No. This will be supported eventually (time-frame unknown)

3.4 a) ADD_ADDR implemented, REMOVE_ADDR not implemented (to be done, timeframe unknown)

b) No. Functionality to be added

3.5 MP_FASTCLOSE not implemented. Plan to implement eventually

Question 4: Fallback from MPTCP

4.1 Yes

4.2 The subflow PCBs remain allocated, however the subflow is not used to send data.

4.3 No, tbd

4.4 No, tbd

4.5 No, checksumming not implemented

4.6 Yes

4.8 Fallback hasn't really been put through any structured tests yet

Question 5: Heuristics

5.1 We use "TCP_MAXWIN << tp->rcv_scale". This is temporary and we will use a call into the "multipath" control layer to determine this value in future releases (we need to investigate a suitable way of calculating this).

5.2 cwnd determines the amount of data to send (given that rcv window is always very large). Sendbuffer is scaled based on occupancy.

5.3 We currently don't have Data-level retransmits enabled. However our policy is to retransmit on the next subflow that requests data to send that is suitable. There is no intelligence in the packet scheduler currently,

5.4 The same port numbers are re-used for additional subflows.

Question 6: Security

6.1 Yes

6.2 No

Question 7: IANA

7.1 Yes

Question 8: Congestion Control and subflow policy

8.1 A simple algorithm is used to divide the send buffer between subflows, so that traffic is spread across the subflows.

8.3 No. (to be added)

8.4 No

8.5 No

Question 9: API

9.1 Legacy applications are able to use MPTCP. MPTCP is set globally via a sysctl variable.

9.2 No

9.3 No

Question 10: API

10.1 Some current project work is based on MPTCPs use in vehicle to infrastructure (V2I) connectivity (to provide a persistent connection using 3G and roadside wifi). Other interests are in multi-homed "home-user" environments, high throughput data transfers.... We are also interested in experimenting with congestion control across paths with different path-cost metrics.

10.2 So far only within controlled testbeds

10.3 End hosts

10.4 Depending on the scenario, connection persistence, throughput...

10.5 Still an alpha-quality build, so limited testing so far.

9. Full survey response for Implementation 3

Survey 3.1. Question 1: Your details Question 1 gathers some information about the team that has implemented MPTCP.

1. Your institution: anonymized.

2. Name(s) of people in your implementation and test teams: There were several folks involved in the implementation and testing.

3. Do you want your answers to Question 1.1 and 1.2 above to be anonymised? Yes.

3.2. Question 2: Preliminary information about your implementation Question 2 gathers some preliminary information.

1. What OS is your implementation for? (or is it application layer?) anonymized (commercial OS)

2. Do you support IPv4 or IPv6 addresses or both? Both.

3. Is it publicly available (or will it be?) (for free or a fee?) No.

4. Overall, what are you implementation and testing plans? (details can be given against individual items later) We plan to use it in a mobile environment.

5. Is it an independent implementation? Or does it build on another MPTCP implementation -which one? It is an independent implementation.

6. Have you already done some interop tests, for example with UCLouvain's "reference" Linux implementation? Most MPTCP option formats were tested with the reference Linux implementation.

7. Would you be prepared to take part in an interop event, for example adjacent to IETF-87 in Berlin? Unsure at this point.

3.3. Question 3: Support for MPTCP's Signalling Functionality Question 3 asks about support for the various signalling messages that the MPTCP protocol defines. *** For each message, please give a little information about the status of your implementation: for example, you may have implemented it and fully tested it; the implementation may be in progress; you have not yet implemented it

but plan to soon (timescale?); you may you have no intention to implement it (why?); etc.

1. Connection initiation (MP_CAPABLE) [Section 3.1 RFC6824] a. What is the status of your implementation? Fully implemented and tested against the reference Linux implementation.

b. Any other comments or information?

2. Starting a new subflow (MP_JOIN) [Section 3.2 RFC6824] a. What is the status of your implementation? Fully implemented and tested against the reference Linux implementation.

b. Can either end of the connection start a new subflow (or only the initiator of the original subflow)? Only the initiator of the original sub flow can start other sub flows.

c. What is the maximum number of subflows your implementation can support? There is no hard limit.

d. Any other comments or information?

3. Data transfer (DSS) [Section 3.3 RFC6824] a. What is the status of your implementation? Fully implemented and tested.

b. The "Data ACK" field can be 4 or 8 octets. Which one(s) have you implemented? Both have been implemented but the use of the 4-byte field is the default. When an 8 byte DSS is received, an 8 byte Data ACK is sent in response.

c. The "Data sequence number" field can be 4 or 8 octets. Which one(s) have you implemented? Both have been implemented but the use of the 4-byte field is the default. When a wraparound of the lower 32-bit part of the DSS is detected, the full 8 byte DSS is sent.

d. Does your implementation support the "DATA_FIN" operation to close an MPTCP connection? Yes. There are cases however where the sub flows are closed (TCP FIN'd) but the DATA_FIN is not sent - in this case the MPTCP connection must be closed through a garbage collector after some idle time.

e. Does your implementation support the "Checksum" field (which is negotiated in the MP_CAPABLE handshake)? Yes.

f. Any other comments or information?

4. Address management (ADD_ADDR and REMOVE_ADDR) a. What is the status of your implementation? It does not support sending ADD_ADDR

or processing ADD_ADDR as it is considered a security risk. Also, we only have a client side implementation at the moment which always initiates the sub flows. The remote end does not send ADD_ADDR in our configuration. The client can send REMOVE_ADDR however when one of the established sub flow's source address goes away. The client ignores incoming REMOVE_ADDR options also.

b. Can your implementation do ADD_ADDRESS for addresses that appear *after* the connection has been established? No. c. Any other comments or information?

5. Fast close (MP_FASTCLOSE) [Section 3.5 RFC6824] a. What is the status of your implementation? It is supported. Though Retransmission of Fast close is not supported yet.

b. Any other comments or information?

3.4. Question 4: Fallback from MPTCP Question 4 asks about action when there is a problem with MPTCP, for example due to a middlebox mangling MPTCP's signalling. The connection needs to fall back: if the problem is on the first subflow then MPTCP falls back to TCP, whilst if the problem is on an additional subflow then that subflow is closed with a TCP RST, as discussed in [Section 3.6 RFC6824].

1. If the MP_CAPABLE option is removed by a middlebox, does your implementation fall back to TCP? Yes.

2. If the MP_JOIN option does not get through on the SYN's, does your implementation close the additional subflow? Yes.

3. If the DSS option does not get through on the first data segment(s), does your implementation fall back? (either falling back to MPTCP (if the issue is on the first subflow) or closing the additional subflow (if the issue is on an additional subflow)) Yes it falls back to TCP when there's one sub flow. When there are multiple sub flows, since MPTCP is used in active/backup mode, it is assumed that the sub flow transferring data is most likely to be more usable than any other established sub flow. So the sub flow on which fallback occurred is kept alive and other sub flows are closed. Fallback though is not guaranteed to occur safely when there are more than one sub flows because the infinite mapping option may be stripped like other DSS options and the MP_FAIL option if used in scenarios other than for reporting checksum failure can also be stripped.

4. Similarly, if the "DATA ACK" field does not correctly acknowledge the first data segment(s), does your implementation fall back? No. Current implementation just ignores the unexpected data ack.

5. Does your implementation protect data with the "Checksum" field in the DSS option [Section 3.3 RFC6824]? If the checksum fails (because the subflow has been affected by a middlebox), does your implementation immediately close the affected subflow (with a TCP RST) with the MP_FAIL Option? If the checksum fails and there is a single subflow, does your implementation handle this as a special case, as described in [Section 3.6 RFC6824]? Yes.

6. Does your implementation fall back to TCP by using an "infinite mapping" [Section 3.3.1 RFC6824] (so that the subflow-level data is mapped to the connection-level data for the remainder of the connection)? Yes.

7. Did you find any corner cases where MPTCP's fallback didn't happen properly? If the very first sub flow does not send any data and is disconnected right away, then the current implementation allows a join to occur with the addition of another sub flow which then becomes a fully mp capable sub flow. Thus we allow break before make by letting additional sub flows to be joined if the very first one disconnected even without sending any data. This is a very corner case but an instance where we do not follow the rules of fallback (allow second sub flow even when first sub flow did not send/receive data/data acks).

8. Any other comments or information about fallback? Fallback after connection establishment and after a few data packets were transferred with MPTCP options is complicated. The spec does not clearly cover the cases of options being stripped by middle boxes. It goes into good detail about what to do when the DSS checksum fails, but not when DSS checksum is not in use and the MPTCP options are stripped. Both sender/receiver behaviors could be outlined with more detail.

3.5. Question 5: Heuristics Question 5 gathers information about heuristics: aspects that are not required for protocol correctness but impact the performance. We would like to document best practice so that future implementers can learn from the experience of pioneers. The references contain some initial comments about each topic.

1. Receiver considerations [S3.3.4, RFC6824]: What receiver buffer have you used? Does this depend on the retransmission strategy? What advice should we give about the receiver? We are just using MPTCP in active/backup mode. This mode is simpler wrt receive buffer utilization. The receive buffer sizes at the MPTCP and sub flow level is the same. Automatic buffer tuning is turned off when MPTCP is in use.

2. Sender considerations [S3.3.5, RFC6824]: How do you determine how much data a sender is allowed to send and how big the sender buffer is? What advice should we give about the sender? Automatic buffer tuning is turned off when MPTCP is in use.

3. Reliability and retransmissions [S3.3.6, RFC6824]: What is your retransmission policy? (when do you retransmit on the original subflow vs on another subflow or subflows?) When do you decide that a subflow is underperforming and should be reset, and what do you then do? What advice should we give about this issue? Retransmissions at MPTCP level do not occur on the same sub flow except when MP_FAIL option is received. A sub flow is said to be underperforming when its network connectivity goes away.

4. Port usage [S3.3.8.1, RFC6824]: Does your implementation use the same port number for additional subflows as for the first subflow? Have you used the ability to define a specific port in the Add Address option? What advice should we give about this issue? The destination port is the same. The local port changes for additional sub flows so on the wire it is like two tcp connections to the same remote destination. We have not used Add Address option at all.

5. Delayed subflow start [S3.3.8.2, RFC6824]: What factors does your implementation consider when deciding about opening additional subflows? What advice should we give about this issue? The client implementation is aware of network interfaces coming up or going down and establishes new sub flows or removes existing sub flows accordingly.

6. Failure handling [S3.3.8.3, RFC6824]: Whilst the protocol defines how to handle some unexpected signals, the behaviour after other unexpected signals is not defined. What advice should we give about this issue? Fallback, post establishment is probably a case that needs to be more clearly defined.

7. Use of TCP options: As discussed in [Appendix A, RFC6824], the TCP option space is limited, but a brief study found there was enough room to fit all the MPTCP options. However there are constraints on which MPTCP option(s) can be included in packets with other TCP options - do the suggestions in Appendix A need amending or expanding? Looks good already.

8. What other heuristics should we give advice about? Any other comments or information?

3.6. Question 6: Security Question 6 asks about Security related matters [Section 5 RFC6824].

1. Does your implementation use the hash-based, HMACSHA1 security mechanism defined in [RFC6824]? Yes.
2. Does your implementation support any other handshake algorithms? No.
3. It has been suggested that a Standards-track MPTCP needs a more secure mechanism. Do you have any views about how to achieve this? No. But the mechanism could be tied with SSL because SSL is used wherever security is deemed important.
4. Any other comments or information?

3.7. Question 7: IANA Question 7 asks about IANA related matters.

1. Does your implementation follow the IANA-related definitions? [Section 8 RFC6824] defines: TCP Option Kind number (30); the sub-registry for "MPTCP Option Subtypes"; and the Page 12 of 17 Survey 6/22/13, 5:55 PM sub-registry for "MPTCP Handshake Algorithms" Yes.
2. Any other comments or information? No.

3.8. Question 8: Congestion control and subflow policy Question 8 asks about how you share traffic across multiple subflows.

1. How does your implementation share traffic over the available paths? For example: as a spare path on standby ('all-or-nothing'), as an 'overflow', etc? Does it have the ability to send /receive traffic across multiple subflows simultaneously? It uses active/backup where one sub flow is preferred or has higher priority over other sub flows. When the preferred sub flow fails or begins to experience retransmission timeouts, the other sub flows are used.
2. Does your implementation support "handover" from one subflow to another when losing an interface? Yes.
3. Does your implementation support the coupled congestion control defined in [RFC6356]? No.
4. Does your implementation support some other coupled congestion control (ie that balances traffic on multiple paths according to feedback)? No.
5. The MP_JOIN (Starting a new subflow) Option includes the "B" bit, which allows the sender to indicate whether it wishes the new subflow to be used immediately or as a backup if other path(s) fail. The MP_PRIO Option is a request to change the "B" bit - either on the subflow on which it is sent, or (by setting the optional Address ID

field) on other subflows. Does your implementation support the "B" bit and MP_PRIO mechanisms? Do you think they're useful, or have another suggestion? Yes the implementation uses the B bit and the MP_PRIO option. They are very useful for the active/backup mode of operation.

6. Any other comments or information or suggestions about the advice we should give about congestion control [S3.3.7 RFC6824] and subflow policy [S3.3.8 RFC6824]?

3.9. Question 9: API Question 9 gathers information about your API. [RFC6897] considers the MPTCP Application Interface.

1. With your implementation, can legacy applications use (the existing sockets API to use) MPTCP? How does the implementation decide whether to use MPTCP? Should the advice in [Section 4, RFC6897] be modified or expanded? The implementation does not support MPTCP with existing sockets API. MPTCP is exposed through a private SPI today. If MPTCP becomes prolific over the next few years, MPTCP use shall be expanded.

2. The "basic MPTCP API" enables MPTCP-aware applications to interact with the MPTCP stack via five new socket options. For each one, have you implemented it? has it been useful? a. TCP_MULTIPATH_ENABLE? b. TCP_MULTIPATH_ADD? c. TCP_MULTIPATH_REMOVE? d. TCP_MULTIPATH_SUBFLOWS? e. TCP_MULTIPATH_CONNID? This mode of API is not used. Proprietary methods are used for achieving these basic operations.

3. Have you implemented any aspects of an "advanced MPTCP API"? ([Appendix A, RFC6897] hints at what it might include.) No.

4. Any other comments or information?

3.10. Question 10: Deployments, use cases and operational experiences Question 10 takes the opportunity of this survey to gather some limited information about operational experiences and deployments. Any very brief information would be appreciated, for example:

1. What deployment scenarios are you most interested in? MPTCP in mobile environments is very powerful when used in the active/backup mode. Since the network interfaces available on mobile devices have different cost characteristics as well as different bring up and power usage characteristics, it is not useful to share load across all available network interfaces - at least not currently. Providing session continuity across changing network environments is the key deployment scenario.

2. Is your deployment on "the Internet" or in a controlled environment? The deployment is on the Internet.
 3. Is your deployment on end hosts or with a MPTCPenabled proxy (at one or both ends)? The deployment supports MPTCP on both ends.
 4. What do you see as the most important benefits of MPTCP in your scenario(s)? Described in point 1 of this section.
 5. How extensively have you deployed and experimented with MPTCP so far? Deployment is still in early stages. We have been experimenting with MPTCP for about a year.
 6. MPTCP's design seeks to maximise the chances that the signalling works through middleboxes. Did you find cases where middleboxes blocked MPTCP signalling? Corporate firewalls block MPTCP signaling by default. IETF is one venue where Cisco, and other firewall vendors can be asked to change their defaults to allow MPTCP signals.
 7. MPTCP's design seeks to ensure that, if there is a problem with MPTCP signalling, then the connection either falls back to TCP or removes the problematic subflow. Did you find any corner cases where this didn't happen properly? This has been covered a bit in the Fallback section. When using two sub flows in active/backup mode, there is a possibility that a backup sub flow that never sent data starts being used for retransmitting data that is not going through on the active path. While it is preferable to keep the initial sub flow that successfully sent MPTCP options and drop the backup path, the initial sub flow may be the failing one, and we may want to move to the backup path. But the backup path can be retransmitting data that did not get sent successfully on the active path and if there is a middle box in the backup sub flow's path stripping options, then we have a case where the MPTCP session may not be recoverable as it may not be evident from what point in the MPTCP sequence space, data was being sent. The spec does talk of retaining the initial sub flow and closing the failed flow. So perhaps doing the reverse is not recommended, however, it would certainly be advantageous to support MPTCP better in such a failing environment. Also, in parallel working with firewall vendors to allow MPTCP options always to not have to over-engineer these cases.
 8. Have you encountered any issues or drawbacks with MPTCP?
 9. Any other comments or information?
- 3.11. Question 11: Improvements to RFC6824 1. Are there any areas where [RFC6824] could be improved, either in technical content or clarity? Discussed in the fallback section. Other areas around

MPTCP performance such as support for sub flow level automatic buffer scaling, varying QoS support, varying window scaling support on each sub flow may be worth discussing further, although they are outside the scope of the current spec.

2. Any other issues you want to raise? Some additional work on option signaling that we will bring up in future discussions.

10. Full survey response for Implementation 4

1. Your institution: Citrix Systems, Inc.

2. Name(s) of people in your implementation and test teams: NA

3. Do you want your answers to Question 1.1 and 1.2 above to be anonymised? No

3.2. Question 2: Preliminary information about your implementation
Question 2 gathers some preliminary information.

1. What OS is your implementation for? (or is it application layer?)
NetScaler Firmware

2. Do you support IPv4 or IPv6 addresses or both? Both

3. Is it publicly available (or will it be?) (for free or a fee?)
It is available for purchase

4. Overall, what are you implementation and testing plans? (details can be given against individual items later)

5. Is it an independent implementation? Or does it build on another MPTCP implementation -which one? It is an independent implementation

6. Have you already done some interop tests, for example with UCLouvain's "reference" Linux implementation? Yes, our implementation is extensively tested with Linux reference implementation

7. Would you be prepared to take part in an interop event, for example adjacent to IETF-87 in Berlin?

3.3. Question 3: Support for MPTCP's Signalling Functionality
Question 3 asks about support for the various signalling messages that the MPTCP protocol defines. *** For each message, please give a little information about the status of your implementation: for example, you may have implemented it and fully tested it; the

implementation may be in progress; you have not yet implemented it but plan to soon (timescale?); you may have no intention to implement it (why?); etc.

1. Connection initiation (MP_CAPABLE) [Section 3.1 RFC6824] a. What is the status of your implementation? Fully implemented and tested

b. Any other comments or information? One security concern here is that the keys are exchanged in plain text which is prone to attacks and also the key generation mechanism is highly computational intensive

2. Starting a new subflow (MP_JOIN) [Section 3.2 RFC6824] a. What is the status of your implementation? Fully implemented and tested

b. Can either end of the connection start a new subflow (or only the initiator of the original subflow)? Only the initiator of the original subflow can initiate additional subflows.

c. What is the maximum number of subflows your implementation can support? we support maximum 6 subflows.

d. Any other comments or information?

3. Data transfer (DSS) [Section 3.3 RFC6824] a. What is the status of your implementation? Fully implemented and tested

b. The "Data ACK" field can be 4 or 8 octets. Which one(s) have you implemented? Our implementation supports both 4 or 8 Octets Data Ack in both the directions

c. The "Data sequence number" field can be 4 or 8 octets. Which one(s) have you implemented? Our implementation supports both 4 or 8 Octets DSN in both the directions

d. Does your implementation support the "DATA_FIN" operation to close an MPTCP connection? YES

e. Does your implementation support the "Checksum" field (which is negotiated in the MP_CAPABLE handshake)? YES

f. Any other comments or information?

4. Address management (ADD_ADDR and REMOVE_ADDR) [Section 3.4 RFC6824]

a. What is the status of your implementation? REMOVE_ADDR is implemented and tested

b. Can your implementation do ADD_ADDRESS for addresses that appear *after* the connection has been established? NO

c. Any other comments or information? ADD_ADDRESS may not be much useful in the real environment situation given that most of the clients are behind the NATing devices.

5. Fast close (MP_FASTCLOSE) [Section 3.5 RFC6824] a. What is the status of your implementation? Implemented and tested b. Any other comments or information?

3.4. Question 4: Fallback from MPTCP Question 4 asks about action when there is a problem with MPTCP, for example due to a middlebox mangling MPTCP's signalling. The connection needs to fall back: if the problem is on the first subflow then MPTCP falls back to TCP, whilst if the problem is on an additional subflow then that subflow is closed with a TCP RST, as discussed in [Section 3.6 RFC6824].

1. If the MP_CAPABLE option is removed by a middlebox, does your implementation fall back to TCP? YES

2. If the MP_JOIN option does not get through on the SYNs, does your implementation close the additional subflow? YES

3. If the DSS option does not get through on the first data segment(s), does your implementation fall back? (either falling back to MPTCP (if the issue is on the first subflow) or closing the additional subflow (if the issue is on an additional subflow)) YES

4. Similarly, if the "DATA ACK" field does not correctly acknowledge the first data segment(s), does your implementation fall back? If the sender receives pure ack for its first DSS packet then it fallbacks to regular TCP.

5. Does your implementation protect data with the "Checksum" field in the DSS option [Section 3.3 RFC6824]? If the checksum fails (because the subflow has been affected by a middlebox), does your implementation immediately close the affected subflow (with a TCP RST) with the MP_FAIL Option? If the checksum fails and there is a single subflow, does your implementation handle this as a special case, as described in [Section 3.6 RFC6824]? Yes, our implementation supports DSS checksum and will close the subflow with RST if the checksum validation fails and there are more than one subflows and sends MP_FAIL if there is a single subflow expecting infinite map from the peer.

6. Does your implementation fall back to TCP by using an "infinite mapping" [Section 3.3.1 RFC6824] (so that the subflow-level data is

mapped to the connection-level data for the remainder of the connection)? YES.

7. Did you find any corner cases where MPTCP's fallback didn't happen properly? We have found few cases where the draft is not clear about the recommended action and fallback strategy, like: 1. what is the expected behavior when pure ack or data packet without dss is received in middle of transaction? How the hosts should fallback in this case? This can happen if the routing changes and the new path drops mptcp options. In this case MP_FAIL/infinite map exchange may not be possible and so could not decide whether both parties are in sync to fallback to tcp. 2. whether infinite map is unidirectional or bidirectional? If one host is sending infinite map to peer, does the peer also needs to send infinite map to the host? Exchanging infinite map and falling back to TCP from both ends is easy from implementation point of view. 8. Any other comments or information about fallback?

3.5. Question 5: Heuristics Question 5 gathers information about heuristics: aspects that are not required for protocol correctness but impact the performance. We would like to document best practice so that future implementers can learn from the experience of pioneers. The references contain some initial comments about each topic.

1. Receiver considerations [S3.3.4, RFC6824]: What receiver buffer have you used? Does this depend on the retransmission strategy? What advice should we give about the receiver? Our implementation uses varying buffer size based on the services and application type.

2. Sender considerations [S3.3.5, RFC6824]: How do you determine how much data a sender is allowed to send and how big the sender buffer is? What advice should we give about the sender? The send side flow control is handled at mptcp level and is independent to subflows. The mptcp level flow control is (almost) same as the regular TCP flow control.

3. Reliability and retransmissions [S3.3.6, RFC6824]: What is your retransmission policy? (when do you retransmit on the original subflow vs on another subflow or subflows?) When do you decide that a subflow is underperforming and should be reset, and what do you then do? What advice should we give about this issue? The retransmission is done by the subflows as long as the subflow is alive and is not removed by the REM_ADDR/RST/... . If 3 RTO happens on the subflow doing retransmission and multiple subflows are available then the mptcp starts retransmission from additional subflow. The original subflow continues retransmission for 7RTO and will be closed after that with RST.

4. Port usage [S3.3.8.1, RFC6824]: Does your implementation use the same port number for additional subflows as for the first subflow? Have you used the ability to define a specific port in the Add Address option? What advice should we give about this issue? Our current implementation doesnot support ADD_ADDR and subflow initiation.

5. Delayed subflow start [S3.3.8.2, RFC6824]: What factors does your implementation consider when deciding about opening additional subflows? What advice should we give about this issue? NA

6. Failure handling [S3.3.8.3, RFC6824]: Whilst the protocol defines how to handle some unexpected signals, the behaviour after other unexpected signals is not defined. What advice should we give about this issue? RFC should clearly define failure case handling otherwise it creates interoperability problems among various implementations. Our strategy in most of the unexpected failuire case is to send MP_FAIL RST with expected DSN if there are multiple subflows and MP_FAIL if there is a single subflow expecting infinite map from the peer.

7. Use of TCP options: As discussed in [Appendix A, RFC6824], the TCP option space is limited, but a brief study found there was enough room to fit all the MPTCP options. However there are constraints on which MPTCP option(s) can be included in packets with other TCP options - do the suggestions in Appendix A need amending or expanding? Looks fine now. Atleast timestamp can be included with every dss packet (28bytes for dss and 12bytes for Timestamp), but if there are any other options which needs to be included in data packets then the implementation has to choose which one to include among them.

8. What other heuristics should we give advice about? Any other comments or information?

3.6. Question 6: Security Question 6 asks about Security related matters [Section 5 RFC6824].

1. Does your implementation use the hash-based, HMAC-SHA1 security mechanism defined in [RFC6824]? YES.

2. Does your implementation support any other handshake algorithms? NO.

3. It has been suggested that a Standards-track MPTCP needs a more secure mechanism. Do you have any views about how to achieve this? Yes we also feel more secure and light weight mechanism is required.

4. Any other comments or information?

3.7. Question 7: IANA Question 7 asks about IANA related matters.

1. Does your implementation follow the IANA-related definitions? [Section 8 RFC6824] defines: TCP Option Kind number (30); the sub-registry for "MPTCP Option Subtypes"; and the sub-registry for "MPTCP Handshake Algorithms" YES. 2. Any other comments or information?

3.8. Question 8: Congestion control and subflow policy Question 8 asks about how you share traffic across multiple subflows.

1. How does your implementation share traffic over the available paths? For example: as a spare path on standby ('all-or- nothing'), as an 'overflow', etc? Does it have the ability to send /receive traffic across multiple subflows simultaneously? We give preference to the path that client is currently using to send data/ack and also has policy based on primary/backup setup. We accept data from multiple subflows simultaneously but don't send it simultaneously out.

2. Does your implementation support "handover" from one subflow to another when losing an interface? YES.

3. Does your implementation support the coupled congestion control defined in [RFC6356]? NO.

4. Does your implementation support some other coupled congestion control (ie that balances traffic on multiple paths according to feedback)? NO.

5. The MP_JOIN (Starting a new subflow) Option includes the "B" bit, which allows the sender to indicate whether it wishes the new subflow to be used immediately or as a backup if other path(s) fail. The MP_PRIO Option is a request to change the "B" bit - either on the subflow on which it is sent, or (by setting the optional Address ID field) on other subflows. Does your implementation support the "B" bit and MP_PRIO mechanisms? Do you think they're useful, or have another suggestion? YES, our implementation supports both 'B' flag and MP_PRIO options, they are much useful to change the priority of the subflows and to decide which subflow to use for data transfer.

6. Any other comments or information or suggestions about the advice we should give about congestion control [S3.3.7 RFC6824] and subflow policy [S3.3.8 RFC6824]?

3.9. Question 9: API Question 9 gathers information about your API. [RFC6897] considers the MPTCP Application Interface.

1. With your implementation, can legacy applications use (the existing sockets API to use) MPTCP? How does the implementation decide whether to use MPTCP? Should the advice in [Section 4, RFC6897] be modified or expanded? NA.
2. The "basic MPTCP API" enables MPTCP-aware applications to interact with the MPTCP stack via five new socket options. For each one, have you implemented it? has it been useful? a. TCP_MULTIPATH_ENABLE? b. TCP_MULTIPATH_ADD? c. TCP_MULTIPATH_REMOVE? d. TCP_MULTIPATH_SUBFLOWS? e. TCP_MULTIPATH_CONNID? NA.
3. Have you implemented any aspects of an "advanced MPTCP API"? ([Appendix A, RFC6897] hints at what it might include.) NA. 4. Any other comments or information?

3.10. Question 10: Deployments, use cases and operational experiences Question 10 takes the opportunity of this survey to gather some limited information about operational experiences and deployments. Any very brief information would be appreciated, for example:

1. What deployment scenarios are you most interested in? MPTCP Proxy deployment where the mptcp connections from the clients are terminated and the tcp connection is established on the other side.
2. Is your deployment on "the Internet" or in a controlled environment? Targeted for the Internet deployment.
3. Is your deployment on end hosts or with a MPTCP-enabled proxy (at one or both ends)? Proxy.
4. What do you see as the most important benefits of MPTCP in your scenario(s)? Reliability and fault tolerance.
5. How extensively have you deployed and experimented with MPTCP so far?
6. MPTCP's design seeks to maximise the chances that the signalling works through middleboxes. Did you find cases where middleboxes blocked MPTCP signalling? Yes some firewalls seem dropping MPTCP options.
7. MPTCP's design seeks to ensure that, if there is a problem with MPTCP signalling, then the connection either falls back to TCP or removes the problematic subflow. Did you find any corner cases where this didn't happen properly? Few cases listed above.

8. Have you encountered any issues or drawbacks with MPTCP? 9. Any other comments or information?

3.11. Question 11: Improvements to RFC6824

1. Are there any areas where [RFC6824] could be improved, either in technical content or clarity? More clarity required in fallback cases.

2. Any other issues you want to raise?

11. Normative References

[RFC6356] Raiciu, C., Handley, M., and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols", RFC 6356, October 2011.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, January 2013.

Author's Address

Philip Eardley
BT

MPTCP
Internet-Draft
Intended status: Standards Track
Expires: April 10, 2014

D. Wing
R. Ravindranath
T. Reddy
Cisco
A. Ford
Unaffiliated
R. Penno
Cisco
October 07, 2013

Multipath TCP (MPTCP) Path Selection using PCP
draft-wing-mptcp-pcp-00

Abstract

MultiPath TCP (MPTCP) allows a host to use multiple interfaces to transfer data. Without knowledge of the characteristics of each network path, the MPTCP stack has to send data to learn those characteristics. This document communicates network characteristics using Port Control Protocol (PCP) to allow the MPTCP stack influence its functions.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 10, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Notational Conventions	3
3. MPTCP stack using PCP	4
4. Multiple Interfaces	6
4.1. Interface Availability	6
4.1.1. consolidate subflows	7
4.1.2. migrating an existing subflow	7
5. Switch-over	7
6. Using MP_PRIO mechanism of MPTCP along with PCP	7
7. PCP Instance ID usage in MPTCP flows	8
8. IANA Considerations	8
9. Security Considerations	8
10. References	8
10.1. Normative References	8
10.2. Informative References	9
Authors' Addresses	9

1. Introduction

Multipath Transmission Control Protocol (MPTCP) [RFC6182] pools multiple TCP paths within a transport connection, and is transparent to the application. Multipath TCP is primarily concerned with utilizing multiple paths end-to-end, where one or both of the end hosts are multihomed. It may also have applications where multiple paths exist within the network and can be manipulated by an end host. An MPTCP connection begins similarly to a regular TCP connection and if extra paths are available, additional TCP subflows are created on these paths, and are combined with the existing session, which continues to appear as a single connection to the applications at both ends. MPTCP provides greater throughput by using multiple paths, and also resilience against path failure. The latter property additionally provides mobility functionality.

MPTCP identifies multiple paths by the presence of multiple addresses at hosts. The discovery and setup of additional subflows will be achieved through a path management method. Section 3.3.8 of [RFC6824] discusses MPTCP policies to share traffic over the available paths. MPTCP may use all paths (for maximum throughput) or a subset of paths (for network resiliency). The path selection is mostly based on local policy, OS behavior, and the MP_PRIO option.

The MPTCP API document [RFC6897] discusses the requirements for MPTCP-aware applications to select multiple paths that can provide the required flow characteristics; for example, 5Mbps of upstream/downstream bandwidth, low loss, low delay, etc. Appendix A.3 of [RFC6897] lists two requirements (REQ-8, REQ-9) for an advanced MPTCP API which would enable the application to select paths based on the link characteristics like bandwidth, latency, etc.

This draft defines the on-the-wire protocol for such an advanced MPTCP API. It uses PCP flow extensions [I-D.wing-pcp-flowdata] to select the best path when multiple paths are available. This would be particularly relevant for applications that are highly interactive but require specific link characteristics such as certain minimum upstream or downstream bandwidth, delay, loss, or jitter characteristics. In such a situation, the MPTCP stack can use PCP to find a interface that provides the necessary characteristics. The network could even acquire the required characteristics (e.g., by assigning bandwidth to the user). The MPTCP stack may start one or more additional subflows that are not immediately used, but are available as "hot standby" for resilience and recovery purposes. PCP can be used to find those additional paths that meet the flow characteristics to handle future failover.

Readers are assumed to be familiar with MPTCP and PCP [RFC6887].

2. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This document uses the terminology defined in MPTCP Architecture [RFC6182], Multipath TCP [RFC6824] and Port Control Protocol [RFC6887].

3. MPTCP stack using PCP

This section describes the algorithm a MPTCP stack can use with PCP extensions. The application would signal the flow characteristics to the MPTCP stack. For example, the MPTCP stack would receive an abstract request from the application to provide a low-latency, low-jitter, n-Mbps of upstream bandwidth and m-Mbps of downstream bandwidth service. The MPTCP stack would send PCP flow extension requests to the default router on each interface, receive PCP flow extension responses indicating the network characteristics, and tune the MPTCP stack accordingly to favor certain interfaces over other interfaces.

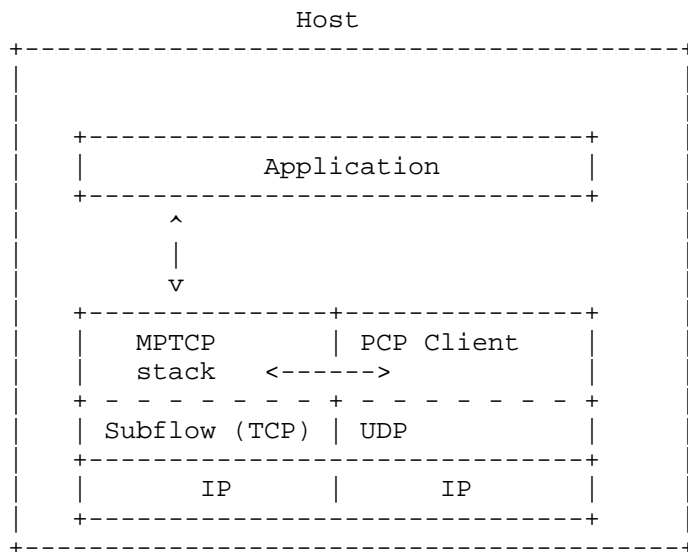
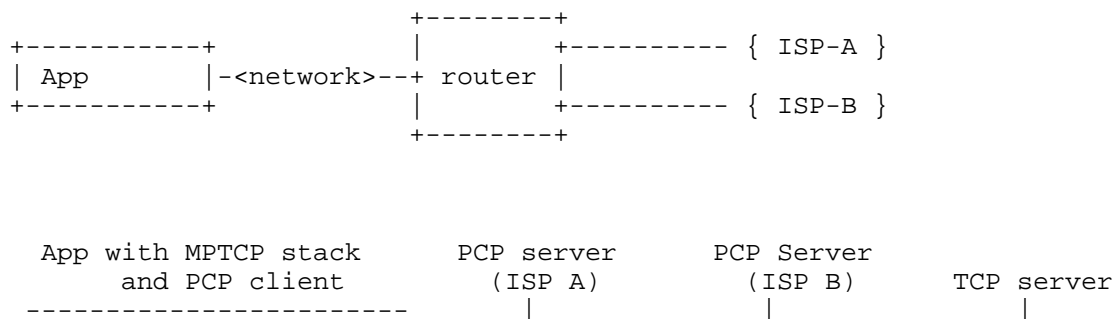


Figure 1: MPTCP stack using PCP

The below steps briefly describe how a MPTCP stack uses the PCP FLOWDATA option:

1. The application requests the MPTCP stack to setup a connection towards a server/remote peer. The MPTCP stack discovers all the available interfaces and gathers the source addresses from these interfaces. This includes addresses from different interfaces (in the case of the host having multiple interfaces), or from the same interface (Multihoming), and also confirms that PCP Flow Extensions is supported.

2. The application signals the required flow characteristics to the MPTCP stack via a API (such as the abstract API described in Appendix A of [RFC6897]). After getting the flow characteristics, the MPTCP stack uses the PCP client to send PCP MAP opcode with FILTER (section 11 of [RFC6887]) and FLOWDATA options (section 3 of [I-D.wing-pcp-flowdata]) to signal the flow characteristics like bandwidth, loss, delay, etc to multiple PCP servers.
3. After receiving the PCP Flow extension responses from multiple PCP servers, the MPTCP stack sorts the source addresses according to the link characteristics.
4. The MPTCP stack picks the source address from the above sorted list and uses the procedures explained in [RFC6824] to send a SYN with MP_CAPABLE flag set to indicate to the server (peer) that this host is MPTCP capable, in order to initiate the primary subflow.
5. If the server supports MPTCP then the stack will either choose to create subsequent subflows using the sorted source address list from step 3 for resiliency purposes, or for use in parallel with the primary subflow to exchange data at a higher throughput. The choice here will likely depend on the stack's interpretation of the application's required flow characteristics.
6. Any changes to the path characteristics that the PCP client receives will be indicated to the MPTCP stack which then may chose to migrate a subflow or consolidate subflows.
7. MPTCP stack can use PCP to communicate with PCP-controlled NAT to learn external IP address, port and advertise in ADD_ADDR MPTCP option to the remote peer. MPTCP stack can also use PCP to communicate with PCP-controlled firewall to permit incoming TCP connections from the remote peer.



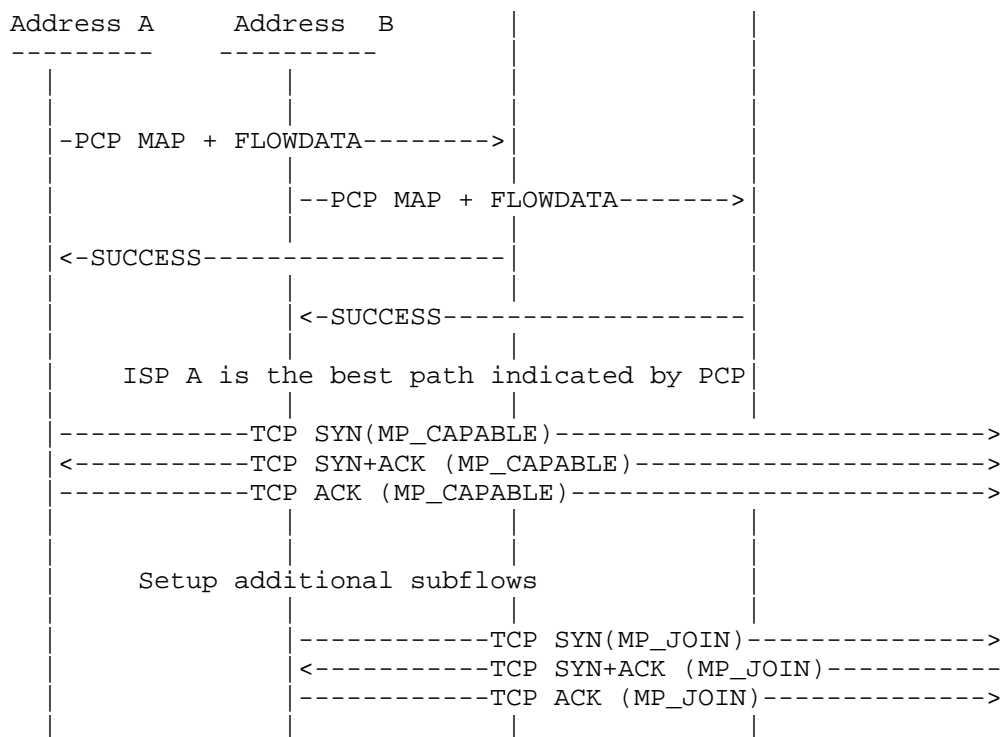


Figure 2: MPTCP stack using PCP

4. Multiple Interfaces

An MPTCP session begins similarly to a regular TCP connection. If multiple paths are available, the MPTCP stack can use PCP flow extensions [I-D.wing-pcp-flowdata] to determine the best path. The advantage is PCP can be used to select the most suitable paths instead of having MPTCP stack try out all paths. When a host has multiple interfaces available (for example 3G/4G, WiFi, VPN etc), an MPTCP application or the MPTCP stack can choose the interface for the primary subflow and interfaces for subsequent subflows according to the path characteristics, as discussed in the previous two sections.

4.1. Interface Availability

A MPTCP stack using the procedures described in [I-D.deng-mif-api-session-continuity-guide] will be notified whenever existing interfaces become unavailable or new interfaces are available. For example the MPTCP stack implementation in the Linux kernel is aware of the changes in the availability of interfaces and can react accordingly.

In such cases the MPTCP stack can use PCP to consolidate subflows or migrate an existing subflow, as described below.

4.1.1. consolidate subflows

When a new interface is discovered, the MPTCP stack can use PCP flow extensions to determine the link characteristics of the new path. If the new path can provide the required flow characteristics then MPTCP could reduce the number of subflows in use. For example, assume three subflows were in use to meet the application bandwidth demand: the primary path providing bandwidth of 2Mbps, the secondary path providing 1Mbps, and the tertiary paths 2Mbps. If PCP determines that the new path can provide 3Mbps, then one subflow can be set up in the new path and, and some of the subflows can be migrated to this new path and thus reduce the number of subflows by closing the old ones. Other factors like jitter, delay, and loss MAY also be considered in the decision to migrate subflows.

4.1.2. migrating an existing subflow

When a existing interface becomes unavailable, the MPTCP stack picks the unused interfaces and uses PCP flow extensions to determine the interfaces which can provide the required flow characteristics. MPTCP stack will follow the previously described steps to pick one or more of the unused interfaces for creating additional subflows.

5. Switch-over

It is possible that the characteristics of a link might change over time, and the MPTCP stack might want to move the subflow to a different interface. For example, if a competing high-bandwidth flow has finished, more bandwidth is available for the MPTCP flow; the DSL line rate might have improved (or degraded); the link speed may have been dynamically increased (or decreased). When link quality changes in such a fashion, a PCP server will send PCP response which could carry a FLOWDATA option where the data fields contain different values from the first response. Upon receiving PCP response, the MPTCP stack can tune its behavior (e.g., increase or decrease traffic on the interface that is now more or less favorable).

6. Using MP_PRIO mechanism of MPTCP along with PCP

MPTCP has a priority mechanism, MP_PRIO, for setting a path to be backup flow. This allows additional subflows to be set up but not used until no higher priority subflows are available, allowing fast fail-over. The MP_PRIO value of a subflow can be changed during the lifetime of the session. A PCP server could send a notification to the PCP client whenever path characteristics change, thus the PCP

client can indicate the same to the MPTCP stack which could change the MP_PRIO values for the associated subflow(s) and trigger switch-over appropriately.

7. PCP Instance ID usage in MPTCP flows

The instance identifier field in PCP flow extensions would help the PCP server to co-relate multiple subflows that are part of the same MPTCP session. The instance ID can be also be used by the service provider to co-relate all the subflows of a MPTCP session.

8. IANA Considerations

None.

9. Security Considerations

Security considerations discussed in [RFC6887] are to be taken into account.

Security considerations discussed in [RFC6824] are to be taken in to account when creating new TCP subflows.

10. References

10.1. Normative References

- [I-D.ietf-pcp-proxy]
Boucadair, M., Penno, R., and D. Wing, "Port Control Protocol (PCP) Proxy Function", draft-ietf-pcp-proxy-04 (work in progress), July 2013.
- [I-D.wing-pcp-flowdata]
Wing, D., Penno, R., and T. Reddy, "PCP Flowdata Option", draft-wing-pcp-flowdata-00 (work in progress), July 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.
- [RFC6182] Ford, A., Raiciu, C., Handley, M., Barre, S., and J. Iyengar, "Architectural Guidelines for Multipath TCP Development", RFC 6182, March 2011.

- [RFC6724] Thaler, D., Draves, R., Matsumoto, A., and T. Chown,
"Default Address Selection for Internet Protocol Version 6
(IPv6)", RFC 6724, September 2012.
- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure,
"TCP Extensions for Multipath Operation with Multiple
Addresses", RFC 6824, January 2013.
- [RFC6887] Wing, D., Cheshire, S., Boucadair, M., Penno, R., and P.
Selkirk, "Port Control Protocol (PCP)", RFC 6887, April
2013.
- [RFC6897] Scharf, M. and A. Ford, "Multipath TCP (MPTCP) Application
Interface Considerations", RFC 6897, March 2013.

10.2. Informative References

- [I-D.deng-mif-api-session-continuity-guide]
Deng, H., Krishnan, S., Lemon, T., and M. Wasserman,
"Guide for application developers on session continuity by
using MIF API", draft-deng-mif-api-session-continuity-
guide-03 (work in progress), October 2012.
- [RFC6296] Wasserman, M. and F. Baker, "IPv6-to-IPv6 Network Prefix
Translation", RFC 6296, June 2011.
- [RFC6356] Raiciu, C., Handley, M., and D. Wischik, "Coupled
Congestion Control for Multipath Transport Protocols", RFC
6356, October 2011.

Authors' Addresses

Dan Wing
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, California 95134
USA

Email: dwing@cisco.com

Ram Mohan Ravindranath
Cisco Systems, Inc.
Cessna Business Park,
Kadabeesanahalli Village, Varthur Hobli,
Sarjapur-Marathahalli Outer Ring Road
Bangalore, Karnataka 560103
India

Email: rmohanr@cisco.com

Tirumaleswar Reddy
Cisco Systems, Inc.
Cessna Business Park, Varthur Hobli
Sarjapur Marathalli Outer Ring Road
Bangalore, Karnataka 560103
India

Email: tiredddy@cisco.com

Alan Ford
Unaffiliated

Email: alan.ford@gmail.com

Reinaldo Penno
Cisco Systems, Inc.
170 West Tasman Drive
San Jose 95134
USA

Email: repenno@cisco.com