Multipath TCP
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2016

R. Barik
University of Oslo
S. Ferlin
Simula Research Laboratory
M. Welzl
University of Oslo

October 19, 2015

A Linked Slow-Start Algorithm for MPTCP
draft-barik-mptcp-lisa-00

Abstract

This document describes the LISA (Linked Slow-Start Algorithm) for Multipath TCP (MPTCP). Currently during slow-start, subflows behave like independent TCP flows making MPTCP behave unfairly to cross-traffic and causing more congestion in the bottleneck, which yields more losses among the MPTCP subflows. LISA couples the initial windows (IW) of MPTCP subflows during the initial slow-start phase to remove this adverse behavior.

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 21, 2016.

Copyright Notice

Table of Contents

1. Introduction

MPTCP is an ongoing standardization effort that aims to extend TCP by allowing multiple paths to be used simultaneously. The current MPTCP implementation provides multiple congestion control algorithms, which aim to provide fairness to TCP flows at the shared bottlenecks. However, in RFC 6356 [RFC6356], the subflows' slow-start phase remains unchanged to RFC 5681 [RFC5681], and all the subflows at this stage behave like independent TCP flows. Following the development of IW as per [RFC6928], each MPTCP subflow starts with IW = 10. With an increasing number of subflows, the subflows' collective behavior during the initial slow-start phase can temporarily be very aggressive towards a concurrent regular TCP flow at the shared bottleneck.

According to [UIT02], most of the TCP sessions in the Internet consist of short flows, e.g., HTTP requests, where TCP will likely never leave slow-start. Therefore, the slow-start behavior becomes of critical importance for the overall performance.

To mitigate the adverse effect during initial slow-start, we introduce LISA, the "Linked Slow-Start Algorithm". LISA's design is

based on initial congestion window sharing of MPTCP subflows, hence, providing coupling in the window increase.

1.1. Definitions

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 RFC 2119 [RFC2119].

Acronyms used in this document:

IW -- Initial Window

RTT -- Round Trip Time

CWND -- Congestion Window

Inflight -- MPTCP subflow’s inflight data

old_subflow.CWND -- Congestion Window of the subflow having largest sending rate

new_subflow.CWND -- New incoming subflow’s Congestion Window

Ignore_ACKs -- a boolean variable indicating whether ACKs should be ignored

ACKs_To_Ignore -- the number of ACKs for which old_subflow.CWND stops increasing during slow-start

compound CWND -- sum of CWND of the subflows in slow-start

2. MPTCP Slow-Start Problem Description

Given that it takes 1 RTT for the sender to receive any feedback on a given TCP connection, sending an additional segment after every ACK is rather aggressive. Therefore in slow-start, all subflows independently doubling their CWND as in regular TCP, results in MPTCP also doubling its compound CWND.

2.1. Example of current MPTCP slow-start problem

We illustrate the MPTCP slow-start behavior with an example: Consider an MPTCP connection consisting of 2 subflows. The first subflow starts with IW = 10, and after 2 RTTs the CWND becomes 40 and a new subflow joins, again with IW = 10. Then, the compound CWND becomes 40+10 = 50. With an increasing number of subflows, the compound CWND in MPTCP becomes larger than that of a concurrent TCP flow.

For example, MPTCP with eight subflows (as recommended in [DCMPTCP11] for datacenters) will have a compound CWND of 110 (40+7*10). As a result, MPTCP would behave unfairly to a concurrent TCP flow sharing the bottleneck. This aggressive behavior of MPTCP also affects the performance of MPTCP. If multiple subflows share a bottleneck, each of them doubling their rate every RTT, will cause excessive losses at the bottleneck. This makes MPTCP enter the congestion avoidance phase earlier and thereby increases the completion time of the transfer.

3. Linked Slow-Start Algorithm

3.1. Description of LISA

The idea behind LISA is that each new subflow takes a 'packet credit' from an existing subflow in slow-start for its own IW. We design the mechanism such that a new subflow has 10 segments as the upper limit [RFC6928] and 3 segments as the lower limit [RFC3390]. This is based on [RFC6928], [RFC3390] and the main reason behind it is to let these subflows compete reasonably with other flows. We also divide the CWND fairly in order to give all subflows an equal chance when competing with each other.

LISA first finds the subflow with the largest sending rate measured over the last RTT. Depending on the subflow's CWND, between 3 and 10 segments are taken from it as packet credit and used for the new subflow's IW. The packet credit is realized by reducing the CWND from the old subflow and halting its increase for ACKs_To_Ignore number of ACKs.

We clarify LISA with the example given in Section 2.1. After 2 RTTs, the old_subflow.CWND = 40 and a new_subflow joins the connection. Since old_subflow.CWND >= 20 (refer to Section 3.2), 10 packets can be taken by the new_subflow.CWND, resulting in old_subflow.CWND = 30 and new_subflow.CWND = 10. Hence, MPTCP's compound CWND, whose current size is 40, should ideally become 60+20 = 80 after 1 RTT. (Linux sends ACKs for every segment in slow-start.) However, if 40 segments from old_subflow.CWND are already in flight, the compound CWND becomes in fact 70+20 = 90. Here, LISA keeps old_subflow.CWND from increasing for the next 10 ACKs. In comparison, MPTCP without LISA would have 80+20=100 after 1 RTT.

3.2. Algorithm

Below, we describe the LISA algorithm. LISA is invoked before a new subflow sends its IW.

1. Before computing the new_subflow.CWND, Ignore_ACKs = False and ACKs_To_Ignore = 0.

2. Then, ignoring the new_subflow, the subflow in slow-start with the largest sending rate (old_subflow.CWND, measured over the last RTT) is selected.

3. If there is no such subflow, the IW of the new_subflow.CWND = 10 Otherwise, the following steps are executed:

```
if old_subflow.CWND >= 20

   old_subflow.CWND -= 10

   new_subflow.CWND = 10

   Ignore_ACKs = True

else if old_subflow.CWND >= 6

   new_subflow.CWND -= old_subflow.CWND / 2

   old_subflow.CWND -= new_subflow.CWND

   Ignore_ACKs = True

else

   new_subflow.CWND = 3
```

4. if Ignore_ACKs and Inflight > old_subflow.CWND

```
   // do not increase CWND when ACKs arrive

   ACKs_To_Ignore = Inflight - old_subflow.CWND
```

4. Implementation Considerations

LISA is implemented as a patch to the Linux kernel 3.14.33+ and within MPTCP's v0.89.5.

5. Conclusions

We identify the adverse effect of MPTCP's uncoupled slow-start on the performance of MPTCP itself and on concurrent TCP traffic. We propose LISA, a linked slow-start algorithm for MPTCP that couples MPTCP subflows during slow-start phase. LISA was implemented as a patch to the Linux kernel and evaluated in both emulated and real

testbeds [lisa]. In this evaluation, we observed that TCP (CUBIC) completes its transmission earlier than MPTCP without LISA. This is due to the large overshoot when an additional subflow joins, causing more retransmissions. LISA solves this problem.

6. Acknowledgements

This work was part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The authors also would like to thank David Hayes (UiO) for his comments. The views expressed are solely those of the authors.

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

9. Change History

Changes made to this document:

00->00 : First version

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, October 2002.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <http://www.rfc-editor.org/info/rfc5681>.

[RFC6356] Raiciu, C., Handley, M., and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols", RFC 6356, DOI 10.17487/RFC6356, October 2011, <http://www.rfc-editor.org/info/rfc6356>.

[RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, April 2013.

10.2. Informative References

[DCMPTCP11]
Raiciu, C., Barre, S., Pluntke, C., Greenhalgh, A., Wischik, D., and M. Handley, "Improving datacenter performance and robustness with multipath TCP", ACM SIGCOMM p266-277, August 2011.

[UIT02] Brownlee, N. and K. Claffy, "Understanding internet traffic streams: Dragonflies and tortoises", IEEE Communications Magazine p110-117, 2002.

[lisa] Barik, R., Welzl, M., Ferlin, S., and O. Alay, "LISA: A Linked Slow-Start Algorithm for MPTCP", the paper will be available as soon as possible , 2015.

Authors' Addresses

Runa Barik
University of Oslo
PO Box 1080 Blindern
Oslo N-0316
Norway

Email: runabk@ifi.uio.no

Simone Ferlin
Simula Research Laboratory
P.O.Box 134
Lysaker 1325
Norway

Email: ferlin@simula.no

Michael Welzl
University of Oslo
PO Box 1080 Blindern
Oslo N-0316
Norway

Phone: +47 2285 2420
Email: michawe@ifi.uio.no

Multipath TCP
Internet-Draft
Intended status: Standards Track
Expires: December 29, 2016

R. Barik
University of Oslo
S. Ferlin
Simula Research Laboratory
M. Welzl
University of Oslo

June 27, 2016

A Linked Slow-Start Algorithm for MPTCP
draft-barik-mptcp-lisa-01

Abstract

This document describes the LISA (Linked Slow-Start Algorithm) for Multipath TCP (MPTCP). Currently during slow-start, subflows behave like independent TCP flows making MPTCP unfair to cross-traffic and causing more congestion at the bottleneck. This also yields more losses among the MPTCP subflows. LISA couples the initial windows (IW) of MPTCP subflows during the initial slow-start phase to remove this adverse behavior.

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 December 29, 2016.

Copyright Notice

Table of Contents

1. Introduction

The current MPTCP implementation provides multiple congestion control algorithms, which aim to provide fairness to TCP flows at the shared bottlenecks. However, in RFC 6356 [RFC6356], the subflows' slow-start phase remains unchanged to RFC 5681 [RFC5681], and all the subflows at this stage behave like independent TCP flows. Following the development of IW as per [RFC6928], each MPTCP subflow can start with IW = 10. With an increasing number of subflows, the subflows' collective behavior during the initial slow-start phase can temporarily be very aggressive towards a concurrent regular TCP flow at the shared bottleneck.

According to [UIT02], most of the TCP sessions in the Internet consist of short flows, e.g., HTTP requests, where TCP will likely never leave slow-start. Therefore, the slow-start behavior becomes of critical importance for the overall performance.

To mitigate the adverse effect during initial slow-start, we introduce LISA, the "Linked Slow-Start Algorithm". LISA shares the congestion window MPTCP subflows in slow start whenever a new subflow joins.

1.1. Definitions

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 RFC 2119 [RFC2119].

Acronyms used in this document:

IW -- Initial Window

RTT -- Round Trip Time

CWND -- Congestion Window

Inflight -- MPTCP subflow's inflight data

old_subflow.CWND -- Congestion Window of the subflow having largest sending rate

new_subflow.CWND -- New incoming subflow's Congestion Window

Ignore_ACKs -- a boolean variable indicating whether ACKs should be ignored

ACKs_To_Ignore -- the number of ACKs for which old_subflow.CWND stops increasing during slow-start

compound CWND -- sum of CWND of the subflows in slow-start

2. MPTCP Slow-Start Problem Description

Since it takes 1 RTT for the sender to receive any feedback on a given TCP connection, sending an additional segment after every ACK is rather aggressive. Therefore, in slow-start, all subflows independently doubling their CWND as in regular TCP results in MPTCP also doubling its compound CWND. The MPTCP aggregate only diverges from this behavior when the number of subflows changes. Coupling of CWND is therefore not necessary in slow-start except when a new subflow joins.

2.1. Example of current MPTCP slow-start problem

We illustrate the problematic MPTCP slow-start behavior with an example: Consider an MPTCP connection consisting of 2 subflows. The first subflow starts with IW = 10, and after 2 RTTs the CWND becomes 40 and a new subflow joins, again with IW = 10. Then, the compound CWND becomes 40+10 = 50. With an increasing number of subflows, the compound CWND in MPTCP becomes larger than that of a concurrent TCP flow.

For example, MPTCP with eight subflows (as recommended in [DCMPTCP11] for datacenters) will have a compound CWND of 110 (40+7*10). As a result, MPTCP would behave unfairly to a concurrent TCP flow sharing the bottleneck. This aggressive behavior of MPTCP also affects the performance of MPTCP. If multiple subflows share a bottleneck, each of them doubling their rate every RTT, will cause excessive losses at the bottleneck. This makes MPTCP enter the congestion avoidance phase earlier and thereby increases the completion time of the transfer.

This problem, and the improvement attained with LISA, are documented in detail in [lisa].

3. Linked Slow-Start Algorithm

3.1. Description of LISA

The idea behind LISA is that each new subflow takes a ’packet credit’ from an existing subflow in slow-start for its own IW. We design the mechanism such that a new subflow has 10 segments as the upper limit

[RFC6928] and 3 segments as the lower limit [RFC3390]. This is based on [RFC6928], [RFC3390] and the main reason behind it is to let these subflows compete reasonably with other flows. We also divide the CWND fairly in order to give all subflows an equal chance when competing with each other.

LISA first finds the subflow with the largest sending rate measured over the last RTT. Depending on the subflow's CWND, between 3 and 10 segments are taken from it as packet credit and used for the new subflow's IW. The packet credit is realized by reducing the CWND from the old subflow and halting its increase for ACKs_To_Ignore number of ACKs.

We clarify LISA with the example given in Section 2.1. After 2 RTTs, the old_subflow.CWND = 40 and a new_subflow joins the connection. Since old_subflow.CWND >= 20 (refer to Section 3.2), 10 packets can be taken by the new_subflow.CWND, resulting in old_subflow.CWND = 30 and new_subflow.CWND = 10. Hence, MPTCP's compound CWND, whose current size is 40, should ideally become 60+20 = 80 after 1 RTT (assuming a receiver without delayed ACKs). However, if 40 segments from old_subflow.CWND are already in flight, the compound CWND becomes in fact 70+20 = 90. Here, LISA keeps old_subflow.CWND from increasing for the next 10 ACKs. In comparison, MPTCP without LISA would have a compound CWND of 80+20=100 after 1 RTT.

3.2. Algorithm

Below, we describe the LISA algorithm. LISA is invoked before a new subflow sends its IW.

1. Before computing the new_subflow.CWND, Ignore_ACKs = False and ACKs_To_Ignore = 0.

2. Then, ignoring the new_subflow, the subflow in slow-start with the largest sending rate (old_subflow.CWND, measured over the last RTT) is selected.

3. If there is no such subflow, the IW of the new_subflow.CWND = 10 Otherwise, the following steps are executed:

```
if old_subflow.CWND >= 20 // take IW(10) packets

   old_subflow.CWND -= 10

   new_subflow.CWND = 10
```

```
            Ignore_ACKs = True
         else if old_subflow.CWND >= 6 // take half the packets
            new_subflow.CWND -= old_subflow.CWND / 2
            old_subflow.CWND -= new_subflow.CWND
            Ignore_ACKs = True
         else
            new_subflow.CWND = 3 // can't take from old_subflow
   4.  if Ignore_ACKs and Inflight > old_subflow.CWND
         // do not increase CWND when ACKs arrive
         ACKs_To_Ignore = Inflight - old_subflow.CWND
```

4. Implementation Status

LISA is implemented as a patch to the Linux kernel 3.14.33+ and within MPTCP's v0.89.5. It is meant for research and provided by the Unviersity of Oslo and Simula Research Laboratory, and available for download from http://heim.ifi.uio.no/runabk/lisa This code was used to produce the test results that are reported in [lisa].

5. Acknowledgements

This work was part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The authors also would like to thank David Hayes (UiO) for his comments. The views expressed are solely those of the authors.

6. IANA Considerations

This memo includes no request to IANA.

7. Security Considerations

8. Change History

Changes made to this document:

00->01 : Some minor text improvements and updated a reference.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, DOI 10.17487/RFC3390, October 2002, <http://www.rfc-editor.org/info/rfc3390>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <http://www.rfc-editor.org/info/rfc5681>.

[RFC6356] Raiciu, C., Handley, M., and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols", RFC 6356, DOI 10.17487/RFC6356, October 2011, <http://www.rfc-editor.org/info/rfc6356>.

[RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <http://www.rfc-editor.org/info/rfc6928>.

9.2. Informative References

[DCMPTCP11] Raiciu, C., Barre, S., Pluntke, C., Greenhalgh, A., Wischik, D., and M. Handley, "Improving datacenter performance and robustness with multipath TCP", ACM SIGCOMM p266-277, August 2011.

[UIT02] Brownlee, N. and K. Claffy, "Understanding internet traffic streams: Dragonflies and tortoises", IEEE Communications Magazine p110-117, 2002.

[lisa] Barik, R., Welzl, M., Ferlin, S., and O. Alay, "LISA: A Linked Slow-Start Algorithm for MPTCP", IEEE ICC 2016,

Kuala Lumpur, Malaysia , 2016.

Authors' Addresses

Runa Barik
University of Oslo
PO Box 1080 Blindern
Oslo N-0316
Norway

Email: runabk@ifi.uio.no

Simone Ferlin
Simula Research Laboratory
P.O.Box 134
Lysaker, 1325
Norway

Email: ferlin@simula.no

Michael Welzl
University of Oslo
PO Box 1080 Blindern
Oslo, N-0316
Norway

Phone: +47 2285 2420
Email: michawe@ifi.uio.no

MPTCP Working Group
Internet-Draft
Expires: April 21, 2016

F. Duchene
UCLouvain
C. Paasch
Apple
O. Bonaventure
UCLouvain

October 19, 2015

Multipath TCP MIB
draft-duchene-mptcp-mib-00

Abstract

This memo proposes a simple Management Information Base (MIB) that gathers statistics and counters about the operation of a Multipath TCP implementation.

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 21, 2016.

Table of Contents

1. Introduction

Multipath TCP is an extension to TCP [RFC0793] that was specified in [RFC6824]. Multipath TCP allows hosts to use multiple paths to send and receive the data belonging to one connection. For this, a Multipath TCP is composed of several TCP connections that are called subflows.

This memo proposes a simple Management Information Base (MIB) that gathers statistics and counters about the operation of a Multipath TCP implementation. They are designed to give a better understanding of Multipath TCP operations. In particular, the MIB covers the different failure conditions that might occur and would trigger a fallback to regular TCP of the MPTCP connection or the failure of a subflow. These failures have important operational implications. Further, several counters are defined to track the transmission and reception of data at the MPTCP-layer. These counters might help to understand the performance of MPTCP.

2. The MPTCP Group

```
mptcp OBJECT IDENTIFIER ::= { mib-2 TBD } - TBD

mptcpPassiveConn OBJECT-TYPE
```

```
   SYNTAX     Counter
   UNITS      "connections"
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "The number of MPTCP connections that made a direct
          transition to the ESTABLISHED state from the
          SYN-RECEIVED + MP_CAPABLE state. In this MIB,
      an MPTCP connection
      is defined as a TCP connection where both the
      SYN and the SYN + ACK segments
          contained the MP_CAPABLE options."
   ::= { mptcp 1 }

   mptcpActiveConn OBJECT-TYPE

   SYNTAX     Counter
   UNITS      "connections"
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "The number of MPTCP connections that made a direct
           transition to the ESTABLISHED state from the
           SYN-SENT + MP_CAPABLE state."
   ::= { mptcp 2 }

   mptcpPassiveCsumEnabled OBJECT-TYPE

   SYNTAX     Counter
   UNITS      "connections"
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "The number of MPTCP connections that made a direct
           transition to the ESTABLISHED state from the
           SYN-RECEIVED + MP_CAPABLE state with the DSS CHECKSUM
           enabled."
   ::= { mptcp 3 }

   mptcpActiveCsumEnabled OBJECT-TYPE
```

```
   SYNTAX     Counter
   UNITS      "connections"
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "The number of MPTCP connections that made a direct
           transition to the ESTABLISHED state from the
           SYN-SENT + MP_CAPABLE state with the DSS CHECKSUM
           enabled."
   ::= { mptcp 4 }

   mptcpPassiveRemovedSubflows OBJECT-TYPE

   SYNTAX     Counter
   UNITS      "connections"
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "The number of MPTCP subflows that made a transition to the
           CLOSED state from the ESTABLISHED state upon reception of
           a segment containing REM_ADDR or RST."
   ::= { mptcp 5 }

   mptcpActiveRemovedSubflows OBJECT-TYPE

   SYNTAX     Counter
   UNITS      "connections"
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "The number of MPTCP subflows that made a transition to the
           CLOSED state from the ESTABLISHED state upon emission of
           a segment containing REM_ADDR or RST."
   ::= { mptcp 6 }

   mptcpPassiveAddedSubflows OBJECT-TYPE

   SYNTAX     Counter
   UNITS      "connections"
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "The number of MPTCP subflows that made a direct
           transition to the ESTABLISHED state from the
           SYN-RECEIVED + MP_JOIN state."
   ::= { mptcp 7 }

   mptcpActiveAddedSubflows OBJECT-TYPE
```

```
   SYNTAX     Counter
   UNITS      "connections"
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "The number of MPTCP subflows that made a
           transition to the ESTABLISHED state from the
           SYN-SENT + MP_JOIN state through the
       PRE_ESTABLISHED state."
   ::= { mptcp 8 }

   mptcpFailedToEstablishInitialSubflows OBJECT-TYPE

 SYNTAX     Counter
 UNITS      "connections"
 MAX-ACCESS read-only
 STATUS     current
 DESCRIPTION
        "The number of initial MPTCP subflows (i.e. the initial SYN
     segment contained the MP_CAPABLE option) that could not
         transition to the ESTABLISHED state from the SYN-RECEIVED
         or SYN-SENT states. The reason being one of:
         - the SYN+ACK didn't contain a MP_CAPABLE
         - the first ACK didn't contain a DATA_ACK or the first
     data-segment did not contain a DSS mapping
         - 4-way handshake didn't complete (SYN+ACK or ACK not received)
     Given these reasons, a connection could not get established or fell
     back to regular TCP. They are most likely due to middleboxes
     interfering with the connection."
 ::= { mptcp 9 }

   mptcpFailedToEstablishSubflows OBJECT-TYPE
```

```
   SYNTAX     Counter
   UNITS      "connections"
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "The number of non-initial MPTCP subflows (i.e. subflows
       started with a SYN containing the MP_JOIN option)
       that could not transition to the ESTABLISHED state
       from the SYN-RECEIVED or SYN-SENT states. The reason being:
           - the SYN+ACK didn't contain a MP_JOIN
           - the first ACK didn't contain a DATA_ACK or the first
       data-segment did not contain a DSS mapping
           - The connection had already falled back to TCP
           - the 4-way handshake didn't complete (SYN+ACK or ACK
       not received)
       Given, these reasons a subflow could not get established.
       They are most likely due to middleboxes interfence."
   ::= { mptcp 10 }

   mptcpFallbackEstablishedConnections OBJECT-TYPE

   SYNTAX     Counter
   UNITS      "connections"
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "The number of MPTCP connections that fell back to regular TCP
           while being already ESTABLISHED. The reason being one of:
           - Reception of more than a window worth of data without DSS
           - Reception of a segment with an incorrect DSS checksum
       This happens when a middlebox is interfering with the data
       flow after the connection has been successfully established."
   ::= { mptcp 11 }

   mptcpOtherFailures OBJECT-TYPE

  SYNTAX     Counter
  UNITS      "connections"
  MAX-ACCESS read-only
  STATUS     current
  DESCRIPTION
         "The number of MPTCP connections were an invalid segment was
      received
      - Bad DSS-mapping (aka, the specified DSS-mapping does not map the
            TCP sequence numbers)
      - All other possible failures"
  ::= { mptcp 12 }
```

```
   mptcpInvalidJoinReceived OBJECT-TYPE

   SYNTAX     Counter
   UNITS      "segments"
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "The number of SYN+MP_JOIN segments that were received but
       discarded due to :
           - Error in the HMAC
           - Token not found"
   ::= { mptcp 13 }

   mptcpFailRX OBJECT-TYPE

   SYNTAX     Counter
   UNITS      "connections"
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "The number of MPTCP connections where the remote host
       initiated a fallback. This counter is triggered by the
       reception of a MP_FAIL."
   ::= { mptcp 14 }

   mptcpRetrans OBJECT-TYPE

   SYNTAX     Counter
   UNITS      "segments"
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "The number of segments that where retransmitted
           at the MPTCP (meta) level, including all the types
       of reinjections."
   ::= { mptcp 15 }

   mptcpFastCloseRX OBJECT-TYPE

   SYNTAX     Counter
   UNITS      "segments"
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "The number of MP_FASTCLOSE received."
   ::= { mptcp 16 }

   mptcpFastCloseTX OBJECT-TYPE
```

```
   SYNTAX     Counter
   UNITS      "segments"
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "The number of MP_FASTCLOSE emitted."
   ::= { mptcp 17 }

   mptcpReceivedInOrder OBJECT-TYPE

   SYNTAX     Counter
   UNITS      "segments"
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "The number of segments that were received in order at the
           MPTCP (meta) level."
   ::= { mptcp 18 }

   mptcpReceivedOutOfOrder OBJECT-TYPE

   SYNTAX     Counter
   UNITS      "segments"
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "The number of segments that were received out of order at the
           MPTCP (meta) level."
   ::= { mptcp 19 }

   mptcpSentSegments OBJECT-TYPE

   SYNTAX     Counter
   UNITS      "segments"
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "The number of segments that were emitted at the
           MPTCP (meta) level."
   ::= { mptcp 20 }

   END
```

3. IANA considerations

The MIB module in this document uses the following IANA-assigned OBJECT IDENTIFIER values recorded in the SMI Numbers registry:

Descriptor	OBJECT IDENTIFIER value
mptcp	{ mib-2 TBD}

RFC Editor: The IANA is requested to assign a value for "TBD" under the 'mib-2' subtree and to record the assignment in the SMI Numbers registry. When the assignment has been made, the RFC Editor is asked to replace "TBD" (here and in the MIB module) with the assigned value and to remove this note.

4. Conclusion

This document has proposed a simple Management Information Base (MIB) to manage Multipath TCP.

5. Acknowledgements

This work was partially supported by the FP7 Trilogy-2 project funded by the EC.

6. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <http://www.rfc-editor.org/info/rfc6824>.

Authors' Addresses

Fabien Duchene
UCLouvain

Email: fabien.duchene@uclouvain.be

Christoph Paasch
Apple

Email: cpaasch@apple.com

Olivier Bonaventure
UCLouvain

Email: Olivier.Bonaventure@uclouvain.be

MPTCP Working Group
Internet-Draft
Intended status: Experimental
Expires: March 10, 2016

C. Paasch
G. Greenway
Apple, Inc.
A. Ford
Pexip

September 7, 2015

Multipath TCP behind Layer-4 loadbalancers
draft-paasch-mptcp-loadbalancer-00

Abstract

Large webserver farms consist of thousands of frontend proxies that serve as endpoints for the TCP and TLS connection and relay traffic to the (sometimes distant) backend servers. Load-balancing across those server is done by layer-4 loadbalancers that ensure that a TCP flow will always reach the same server.

Multipath TCP's use of multiple TCP subflows for the transmission of the data stream requires those loadbalancers to be aware of MPTCP to ensure that all subflows belonging to the same MPTCP connection reach the same frontend proxy. In this document we analyze the challenges related to this and suggest a simple modification to the generation of the MPTCP-token to overcome those challenges.

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 March 10, 2016.

Table of Contents

1. Introduction

Internet services rely on large server farms to deliver content to the end-user. In order to cope with the load on those server farms they rely on a large, distributed load-balancing architecture at different layers. Backend servers are serving the content from within the data center to the frontend proxies. These frontend proxies are the ones terminating the TCP connections from the clients. A server farm relies on a large number of these frontend proxies to provide sufficient capacity. In order to balance the load on those frontend proxies, layer-4 loadbalancers are installed in front of these. Those loadbalancers ensure that a TCP-flow will always be routed to the same frontend proxy. For resilience and capacity reasons the data-center typically deploys multiple of these loadbalancers [Shuff13] [Patel13].

These layer-4 loadbalancers rely on consistent hashing algorithms to ensure that a TCP-flow is routed to the appropriate frontend proxy. The consistent hashing algorithm avoids state-synchronization across the loadbalancers, making sure that in case a TCP-flow gets routed to a different loadbalancer (e.g., due to a change in routing) the TCP-flow will still be sent to the appropriate frontend proxy [Greenberg13].

Multipath TCP uses different TCP flows and spreads the application's data stream across these [RFC6824]. These TCP subflows use a different 4-tuple in order to be routed on a different path on the Internet. However, legacy layer-4 loadbalancers are not aware that these different TCP flows actually belong to the same MPTCP connection.

The remainder of this document explains the issues that arise due to this and suggests a possible change to MPTCP's token-generation algorithm to overcome these issues.

2. Problem statement

In an architecture with a single layer-4 loadbalancer but multiple frontend proxies, the layer-4 loadbalancer will have to make sure that the different TCP subflows that belong to the same MPTCP connection are routed to the same frontend proxy. In order to achieve this, the loadbalancer has to be made "MPTCP-aware", tracking the keys exchanged in the MP_CAPABLE handshake. This state-tracking allows the loadbalancer to also calculate the token associated with the MPTCP-connection. The loadbalancer thus creates a mapping (token, frontend proxy), stored in memory for the lifetime of the MPTCP connection. As new TCP subflows are being created by the client, the token included in the SYN+MP_JOIN message allows the loadbalancer to ensure that this subflow is being routed to the appropriate frontend proxy.

However, as soon as the data center employs multiple of these layer-4 loadbalancers, it may happen that TCP subflows that belong to the same MPTCP connection are being routed to different loadbalancers. This implies that the loadbalancer needs to share the mapping-state it created for all MPTCP connections among all other loadbalancers to ensure that all loadbalancers route the subflows of an MPTCP connection to the same frontend proxy. This is substantially more complicated to implement, and would suffer from latency issues.

Another issue when MPTCP is being used in a large server farm is that the different frontend proxies may generate the same token for different MPTCP connections. This may happen because the token is a truncated hash of the key, and hash collisions may occur. A server farm handling millions of MPTCP connections has actually a very high chance of generating those token-collisions. A loadbalancer will thus no more be able to accurately send the SYN+MP_JOIN to the correct frontend proxy in case a token-collision happened for this MPTCP connection.

3. Proposals

The issues described in Section 2 have their origin due to the undeterministic nature in the token-generation. Indeed, if it becomes possible for the loadbalancer to infer the frontend proxy to forward this flow to, MPTCP becomes deployable in such kinds of environments.

The suggested solutions have their basis in a token from which a loadbalacer can glean routing information in a stateless manner. To allow the loadbalancer to infer the proxy based on the token, the proxies each need to be assigned to a range of unique integers. When the token falls within a certain range, the loadbalancer knows to which proxy to forward the sufblow. Using a contiguous range of integers makes the frontend very vulnerable to attackers. Thus, a reversible function is needed that makes the token random-looking. A 32-bit block-cipher (e.g., RC5) provides this random-looking reversible function. Thus, for both proposals we assume that the frontend proxies and the layer-4 loadbalancer share a local secret Y, of size 32 bits. This secret is only known to the server-side data center infrastructure. If X is an integer from within the range associated to the proxy, the proxy will generate the token by encypting X with secret Y. The loadbalancer will simply decrypt the token with the secret Y, which provides it the value of X, allowing it to forward the TCP flow to the appropriate proxy.

This approach also ensures that the tokens generated by different servers are unique to each server, eliminating the token-collision issue outlined in the previous section.

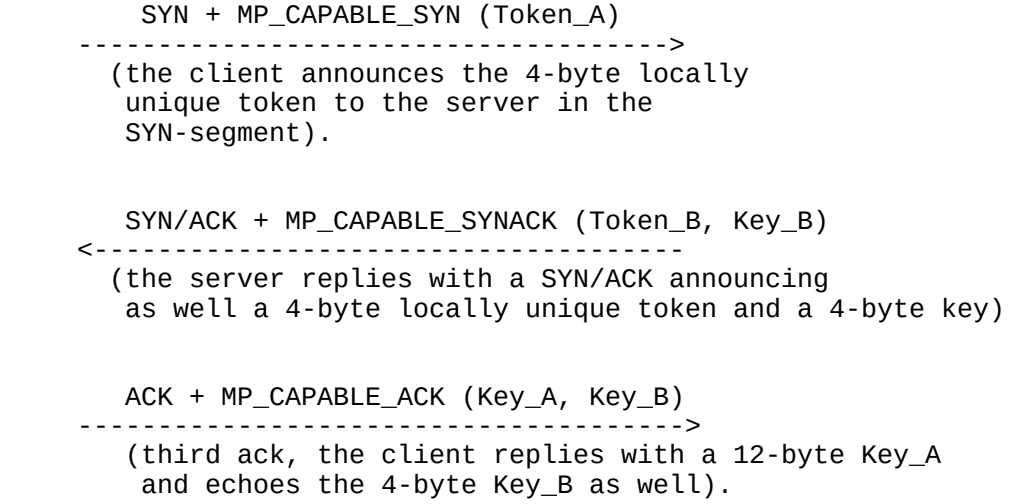
In the following we outline two different approaches to handle the above described problems, using this approach. The two proposals provide different ways of communicating the token over to the peer during the MP_CAPABLE handshake. We would like these proposals to serve as a discussion basis for the design of the definite solution.

3.1. Explicitly announcing the token

One way of communicating the token to simply announce it in plaintext within the MP_CAPABLE handshake. In order to allow this, the wire-format of the MP_CAPABLE handshake needs to change however.

One solution would be to simply increase the size of the MP_CAPABLE by 4 bytes, giving space for the token to be included in the SYN and SYN/ACK as well as adding it to the third ACK. However, due to the scarce TCP-option space this solution would suffer deployment difficulties.

If the solution proposed in [I-D.paasch-mptcp-syncookies] is being deployed, the MP_CAPABLE-option in the SYN-segment has been reduced to 4 bytes. This gives us space within the option-space of the SYN-segment that can be used. This allows the client to announce its token within the SYN-segment. To allow the server to announce its token in the SYN/ACK, without bumping the option-size up to 16 bytes, we reduce the size of the server's key down to 32 bits, which gives space for the server's token. To avoid introducing security-risks by reducing the size of the server's key, we suggest to bump the client's key up to 96 bits. This provides still a total of 128 bits of entropy for the HMAC computation. The suggested handshake is outlined in Figure 1.



The suggested handshake explicitly announces the token.

Figure 1

Reducing the size of the server's key down to 32 bits might be considered a security risk. However, one might argue that neither parties involved in the handshake (client and server) have an interest in compromising the connection. Thus, the server can have confidence that the client is going to generate a 96 bits key with sufficient entropy and thus the server can safely reduce its key-size down to 32 bits.

However, this would require the server to act statefully in the SYN exhcnage if it wanted to be able to open connections back to the client, since the token never appears again in the handshake.

3.2. Changing the token generation

Another suggestion is based on a less drastic change to the MP_CAPABLE handshake. We suggest to infer the token based on the key provided by the host. However, in contrast to [RFC6824], the token is not a truncated hash of the keys. The token-generation uses rather the following scheme: If we define Z as the 32 high-order bits and K the 32 low-order bits of the MPTCP-key generated by a host, we suggest to generate the token as the encryption of Z with key K by using a 32-bit block-cipher (the block-cipher may for example be RC5 - it remains to be defined by the working-group which is an appropriate block-cipher to use for this case). The size of the MPTCP-key remains unchanged and is actually the concatenation of Z with K. Both, K and Z are different for each and every connection, thus the MPTCP-key still provides 64 bits of randomness.

Using this approach, a frontend proxy can make sure that a loadbalancer can derive the identity of the backend server solely through the token in the SYN-segment of the MP_JOIN exchange, without the need to track any MPTCP-related state. To achieve this, the frontend proxy needs to generate K and Z in a specific way. Basically, the proxy derives the token through the method described at the beginning of this Section 3. This gives us the following relation:

token = block_cipher(proxy_id, Y) (Y is the local secret)

However, as described above, at the same time we enforce:

token = block_cipher(Z, K)

Thus, the proxy simply generates a random number K, and can thus generate Z by decrypting the token with key K. It is TBD what number of bits of a token could be used for conveying routing information. Exlcuding those bits, the token would be random, and the key K is random as well, so Z will be random as well. An attacker evesdropping the token cannot infer anything on Z nor on K. However, prolonged gathering of token data could lead to building up some data about the key K.

4. Conclusion

In order to be deployable at a large scale, Multipath TCP has to evolve to accomodate the use-case of distributed layer-4 loadbalancers. In this document we explained the different problems that arise when one wants to deploy MPTCP in a large server farm. We followed up with two possible approaches to solve the issues around the non-deterministic nature of the token. We argue that it is

important that the working group considers this problem and strives to find a solution.

5. IANA Considerations

No IANA considerations.

6. References

6.1. Normative References

[I-D.paasch-mptcp-syncookies]
Paasch, C., Biswas, A., and D. Haas, "Making Multipath TCP robust for stateless webservers", draft-paasch-mptcp-syncookies-00 (work in progress), April 2015.

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

6.2. Informative References

[Greenberg13]
Greenberg, A., Lahiri, P., Maltz, D., Parveen, P., and S. Sengupta, "Towards a Next Generation Data Center Architecture: Scalability and Commoditization", 2018, <http://dl.acm.org/citation.cfm?id=1397732>.

[Patel13] Parveen, P., Bansal, D., Yuan, L., Murthy, A., Maltz, D., Kern, R., Kumar, H., Zikos, M., Wu, H., Kim, C., and N. Karri, "Ananta: Cloud Scale Load Balancing", 2013, <http://dl.acm.org/citation.cfm?id=2486026>.

[Shuff13] Shuff, P., "Building A Billion User Load Balancer", 2013, <https://www.youtube.com/watch?v=MKgJeqF1DHw>.

Authors' Addresses

Christoph Paasch
Apple, Inc.
Cupertino
US

Email: cpaasch@apple.com

Greg Greenway
Apple, Inc.
Cupertino
US

Email: ggreenway@apple.com

Alan Ford
Pexip

Email: alan.ford@gmail.com

MPTCP Working Group
Internet-Draft
Intended status: Experimental
Expires: April 16, 2016

C. Paasch
A. Biswas
D. Haas
Apple, Inc.
October 14, 2015

Making Multipath TCP robust for stateless webservers
draft-paasch-mptcp-syncookies-02

Abstract

This document proposes a modification of the MPTCP handshake that allows it to work efficiently with stateless servers. We first identify the issues around stateless connection establishment using SYN-cookies. Further, we suggest an extension to Multipath TCP to overcome these issues and discuss alternatives.

As a side-effect, the proposed modification to the handshake opens the door to reduce the size of the MP_CAPABLE option in the SYN. This reduces the growing pressure on the TCP-option space in the SYN-segment, giving space for future extensions to TCP.

Table of Contents

1. Introduction

During the establishment of a TCP connection, a server must create state upon the reception of the SYN [RFC0793]. Specifically, it needs to generate an initial sequence number, and reply to the options indicated in the SYN. The server typically maintains in-memory state for the embryonic connection, including state about what options were negotiated, such as window scale factor [RFC7323] and the maximum segment size. It also maintains state about whether SACK [RFC2018] and TCP Timestamps were negotiated during the 3-way handshake.

Attackers exploit this state creation on the server through the SYN-flooding attack. Indeed, an attacker only needs to emit SYN segments with different 4-tuples (source and destination IP addresses and port numbers) in order to make the server create the state and thus consume its memory, while the attacker itself does not need to maintain any state for such an attack [RFC4987].

A common mitigation of this attack is to use a mechanism called SYN-cookies. SYN-cookies rely on the fact that a TCP-connection echoes back certain information that the server puts in the SYN/ACK during the three-way handshake. Notably, the sequence-number is echoed back in the acknowledgment field as well as the TCP timestamp value inside the timestamp option. When generating the SYN/ACK, the server generates these fields in a verifiable fashion. Typically, servers use the 4-tuple, the client's sequence number plus a local secret (which changes over time) to generate the initial sequence number by applying a hashing function to the aforementioned fields. Further, setting certain bits either in the sequence number or the TCP timestamp value allows to encode for example whether SACK has been negotiated and what window-scaling has been received [M08]. Upon the reception of the third ACK, the server can thus verify whether the acknowledgment number is indeed the reply to a SYN/ACK it has generated (using the 4-tuple and the local secret). Further, it can decode from the timestamp echo reply the required information concerning SACK, window scaling and MSS-size.

In case the third ACK is lost during the 3-way handshake of TCP, stateless servers only work if it's the client who initiates the communication by sending data to the server - which is commonly the case in today's application-layer protocols. As the data segment includes the acknowledgement number for the original SYN/ACK as well as the TCP timestamp value, the server is able to reconstruct the connection state even if the third ACK is lost in the network. If the very first data segment is also lost, then the server is unable to reconstruct the connection state and will respond to subsequent data sent by the client with a TCP Reset.

Multipath TCP (MPTCP [RFC6824]) is unable to reconstruct the MPTCP level connection state if the third ack is lost in the network (as explained in the following section). If the first data segment from the client reaches the server, the server can reconstruct the TCP state but not the MPTCP state. Such a server can fallback to regular TCP upon the loss of the third ACK. MPTCP is also prone to the same problem as regular TCP if the first data segment is also lost.

In the following section a more detailed assessment of the issues with MPTCP and TCP SYN-cookies is presented. Section 3 then shows how these issues might get solved.

2. Problem statement

Multipath TCP adds additional state to the 3-way handshake. Notably, the keys must be stored in the state so that later on new subflows can be established as well as the initial data sequence number is known to both hosts. In order to support stateless servers,

Multipath TCP echoes the keys in the third ACK. A stateless server thus can generate its own key in a verifiable fashion (similar to the initial sequence number), and is able to learn the client's key through the echo in the third ACK. The generation of the key is implementation-specific. An example of such a key-generation would be: Key_Server = Hash(5-tuple, server's subflow sequence number, local_secret). The reliance on the third ACK however implies that if this segment gets lost, then the server cannot reconstruct the state associated to the MPTCP connection. Indeed, a Multipath TCP connection is forced to fallback to regular TCP in case the third ACK gets lost or has been reordered with the first data segment of the client, because it cannot infer the client's key from the connection and thus won't be able to generate a valid HMAC to establish new subflows nor does it know the initial data sequence number. In the remainder of this document we refer to the aforementioned issue as "Loss of the third ACK".

Stateless servers also are unable to recover connection state when the third ack and the first data segment are lost. This issue, outlined hereafter, happens even when regular TCP is being used. In case the client is sending multiple segments when initiating the connection, it might be that the third ack as well as the first data segment get lost. Thus, the server only receives the second data segment and will try to reconstruct the state based on this segment's 4-tuple, sequence number and timestamp value. However, as this segment's sequence number has already gone beyond the client's initial sequence number, it will not be able to regenerate the appropriate SYN-cookie and thus the verification will fail. The server effectively cannot infer that the sequence number in the segment has gone beyond TCP's initial sequence number. This will make the server send a TCP reset as it appears to the server that it received a segment for which no SYN cookie was ever generated.

3. Proposal

This section shows how the above problems might be solved in Multipath TCP.

3.1. Loss of the third ACK

In order to make Multipath TCP robust against the loss of the third ACK when SYN-cookies are being deployed on servers, we must make sure that the state-information relevant to Multipath TCP reaches the server in a reliable way. If the client is initiating the data transfer to the server (this data is being delivered reliably through TCP) the state-information could be delivered together with this data and thus is implicitly reliably sent to the server - when the data reaches the server, the state-information reaches the server as well.

We achieve this by adding another variant to the MP_CAPABLE option, differentiated by the length of it (we call this option MP_CAPABLE_ACK in the remainder of this document). It is solely sent on the very first data segment from the client to the server. This option serves the dual purpose of conveying the client's and server's key as well as the DSS mapping which would otherwise have been sent in a DSS option on the first data segment.

Making the MP_CAPABLE in the third ACK reliable opens the door for another improvement in MPTCP. In fact, the client doesn't need to send its own key in the SYN anymore (it will send it reliably in the MP_CAPABLE_ACK). Thus, the MP_CAPABLE option in the SYN segment can avoid adding the key, reducing the option-space requirement of the MP_CAPABLE down to 4 bytes. This is a major improvement as the option-space in the SYN segment is very limited, and allows a TCP connection to negotiate future extensions in the SYN.

As this change is a major extension to Multipath TCP, we require that the version number of the MP_CAPABLE is increased. Further details on the negotiation are presented in Section 3.1.3. The following is a detailled description of the option format and the suggested handshake.

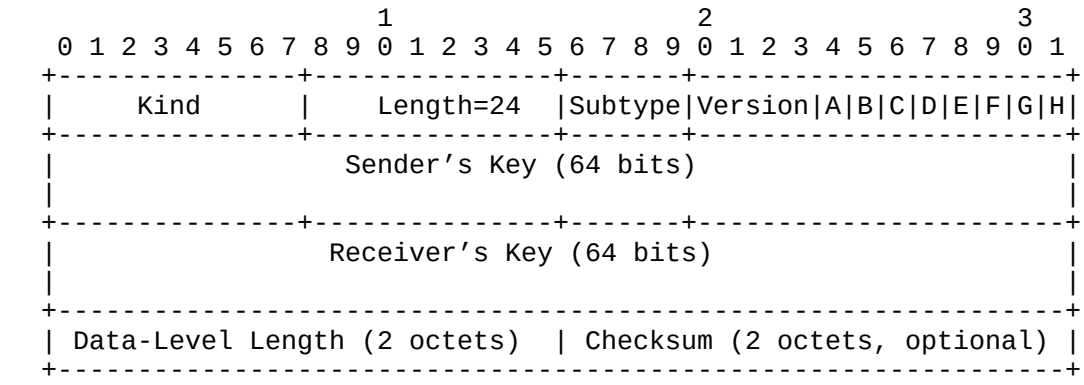
3.1.1. MP_CAPABLE_ACK specification

We suggest to remove the key from the MP_CAPABLE in the SYN-segment. The format of the MP_CAPABLE remains the same (with the bits A to H as well as the version number), with the difference that the key is no more present. Hosts are able to differentiate between the different MP_CAPABLE options through the length-field of the TCP-option.

The MP_CAPABLE option in the SYN/ACK as well as the third ACK (which does not contain any data) remain unmodified from RFC6824.

The MP_CAPABLE_ACK option (shown in Figure 1) contains the same set of bits A to H as well as the version number, like the MP_CAPABLE option. Further, the option includes the data-level length as well as the checksum (in case it has been negotiated during the 3-way handshake). This allows the server to reconstruct the mapping and deliver the data to the application. It must be noted that the information inside the MP_CAPABLE_ACK is less explicit than a DSS option. Notably, the data-sequence number, data acknowledgment as well as the relative subflow-sequence number are not part of the MP_CAPABLE_ACK. Nevertheless, the server is able to reconstruct the mapping because the MP_CAPABLE_ACK is guaranteed to only be sent on the very first data segment. Thus, implicitly the relative subflow-

sequence number equals 1 as well as the data-sequence number, which is equal to the initial data-sequence number.

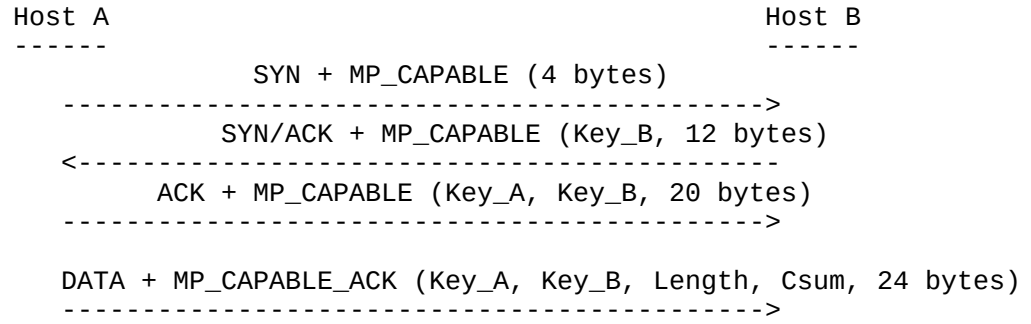


Format of the MP_CAPABLE_ACK option.

Figure 1

The handshake (depicted in Figure 2) starts with the client sending the MP_CAPABLE option to the server inside the SYN. The client is not required to having generated its key already at this point.

Upon reception of this SYN-segment, a stateful server generates a random key and replies with a SYN/ACK. If the server behaves in a stateless manner it has to generate it's own key in a verifiable fashion. This verifiable way of generating the key can be done by using a hash of the 4-tuple, sequence number and a local secret (similar to what is done for the TCP-sequence number [RFC4987]). It will thus be able to verify whether it is indeed the originator of the key echoed back in the MP_CAPABLE_ACK option. When generating this verifiable key, the server has to ensure that the token derived from this key is locally unique (Section 3.1 of RFC6824). If there is already an MPTCP-session with such a token, the server must fallback to regular TCP by not sending an MP_CAPABLE in the third ACK.



The modified MPTCP-handshake only consumes 4 bytes in the SYN.

Figure 2

To complete the three-way handshake, the client has to reply with a third ACK and the MP_CAPABLE option (with both keys as defined in RFC6824). If the client has already data to send, it can even avoid sending the third ACK, and immediately send the data together with the MP_CAPABLE_ACK. Otherwise, the client sends the MP_CAPABLE_ACK as soon as the application writes data on the socket.

The goal of the third ACK (with the MP_CAPABLE) as well as the MP_CAPABLE_ACK is to convey the client's key to the server. An indication for the client that the server received the key is when the server issues a DATA_ACK inside the DSS-option (even if this DATA_ACK does not acknowledge any data). Thus, as long as the client has not sent an MP_CAPABLE_ACK with data, it must add the MP_CAPABLE option in each (non-data) segment sent to the server. It must do this until it either did send an MP_CAPABLE_ACK or until it received a DATA_ACK from the server. The reason for this is explained in Section 3.1.2 and Section 3.1.5. Combining the MP_CAPABLE with the DATA_ACK will require 20 + 8 bytes, which still leaves 12 bytes for the TCP timestamp option.

Finally, the server must send a duplicate acknowledgment to the client upon reception of the client's key. This, to convey to the client that it successfully received the MP_CAPABLE(_ACK) option. It must be noted that this 4-way handshake does not prevent the client to send data before the reception of this fourth acknowledgment.

This mechanism of sending the MP_CAPABLE with a DATA_ACK until the server acknowledges it, introduces additional complexity to the handshake. However, we consider the gain of reducing the MP_CAPABLE option in the SYN-segment as significant enough, that it is worth to accept this added complexity.

3.1.2. TCP Fast Open

If TCP Fast Open [RFC7413] is being used in combination with Multipath TCP [I-D.barre-mptcp-tfo], the server is allowed to send data right after the SYN/ACK, without the need to wait for the third ACK. The server sending this data cannot include a DATA_ACK option inside the DSS option as it does not yet know the client's key. This is not an issue as the DATA_ACK is optional in the DSS option.

However, the client receiving this data sent by the server will have to acknowledge it with a DATA_ACK. As specified above, the client must also send an MP_CAPABLE option on this acknowledgment as it didn't yet receive a DATA_ACK from the server.

3.1.3. Negotiation

We require a way for the hosts to negotiate support for the suggested handshake. As we modify the size of the MP_CAPABLE, our proposal relies on a new version of MPTCP. The client requests this new version of MPTCP during the MP_CAPABLE exchange (it remains to be defined by the IETF which version of MPTCP includes the MP_CAPABLE_ACK option). If the server supports this version, it replies with a SYN/ACK including the MP_CAPABLE and indicating this same version.

3.1.4. DATA_FIN

As the MP_CAPABLE_ACK option includes the same bitfields as the regular MP_CAPABLE, there is no space to indicate a DATA_FIN as is done in the DSS option. This implies that a client cannot send a DATA_FIN together with the first segment of data. Thus, if the server requests the usage of MP_CAPABLE_ACK through the C-bit, the client must send a separate segment with the DSS-option, setting the DATA_FIN-flag to 1, after it has sent the data-segment that includes the MP_CAPABLE_ACK option.

3.1.5. Server sending data

The MP_CAPABLE_ACK version can only be sent by the client if it actually has data to send. One question that this raises is how the proposal will work when the server is the first one to send data to the client. In the following we describe how the handshake will still work when servers behave in a stateless and stateful manner.

For stateless servers the same issue arises as well for regular TCP. Upon loss of the third ACK, the server cannot complete the three-way handshake. Thus, stateless servers that begin the application level protocol by emitting data rely on the fact that the third ACK is

received (irregardless of whether MPTCP is used or not). Thus, this implies that the server also will receive the MP_CAPABLE with this third ACK.

Stateful servers will retransmit the SYN/ACK until the third ACK (including the MP_CAPABLE) has been received. This will thus provide to the server the client's key. When the client eventually sends its own first data segment to the server, it actually does not has to use the MP_CAPABLE_ACK option because the server already did send a DATA_ACK to the client.

3.1.6. Middlebox considerations

Multipath TCP has been designed with middleboxes in mind and so the MP_CAPABLE_ACK option must also be able to go through middleboxes. The following middlebox behaviors have been considered and MP_CAPABLE_ACK acts accordingly across these middleboxes:

- Removing MP_CAPABLE_ACK-option: If a middlebox strips the MP_CAPABLE_ACK option out of the data segment, the server receives data without a corresponding mapping. As defined in Section 3.6 of [RFC6824], the server must then do a seamless fallback to regular TCP.

- Coalescing segments: A middlebox might coalesce the first and second data segment into one single segment. While doing so, it might remove one of the options (either MP_CAPABLE_ACK or the DSS-option of the second segment because of the limited 40 bytes TCP option space). There are two cases to consider:

  * If the DSS-option is not included in the segment, the second half of the payload is not covered by a mapping. Thus, the server will do a seamless fallback to regular TCP as defined by [RFC6824] in Section 3.6. This fallback will trigger because RFC6824 specifies that during the beginning of a connection (as long as the path has not been proven to let the MPTCP-options unmodified in both directions) a seamless fallback to regular TCP must be done by stopping to send DATA_ACKs to the client.

  * If the MP_CAPABLE_ACK option is not present, then the DSS-option provides an offset of the TCP sequence number. As the server behaves statelessly it can only assume that the present mapping belongs to the first byte of the payload (similar to what is explained in detail in Section 3.2). As this however is not true, it will calculate an incorrect initial TCP sequence number and thus reply with a TCP-reset as the SYN-cookie is invalid. As such kind of middleboxes are very rare we consider this behavior as acceptable.

- Splitting segments: A TCP segmentation offload engine (TSO) might split the first segment in smaller segments and copy the MP_CAPABLE_ACK option on each of these segments. Thanks to the data-length value included in the MP_CAPABLE_ACK option, the server is able to detect this and correctly reconstructs the mapping. In case the first of these splitted segments gets lost, the server finds itself in a situation similar to the one described in Section 2. The TCP sequence number doesn't allow anymore to verify the SYN-cookie and thus a TCP reset is sent. This behavior is the same as for regular TCP.

- Payload modifying middlebox: In case the middlebox modifies the payload, the DSS-checksum included in the MP_CAPABLE_ACK option allows to detect this and will trigger a fallback to regular TCP as defined in [RFC6824].

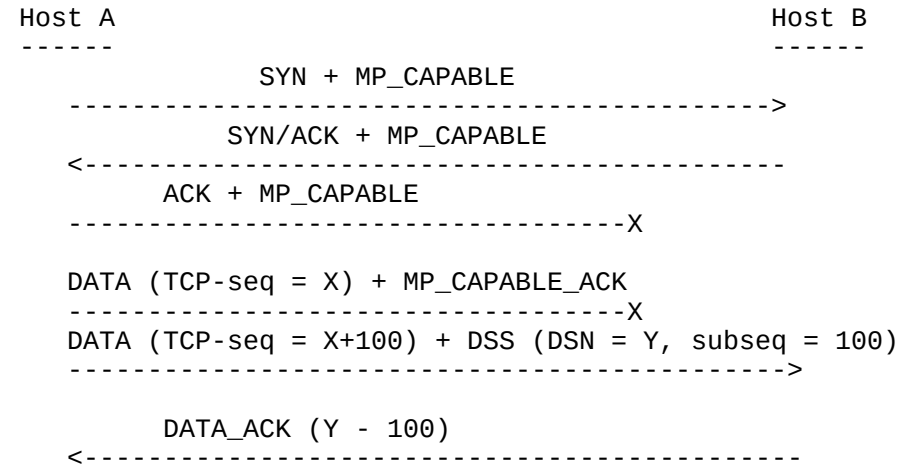
3.2. Loss of the first data segment

Section 2 described the issue of losing the first data segment of a connection while TCP SYN-cookies are in use. The following outlines how Multipath TCP actually allows to fix this particular issue.

Consider the packet-flow of Figure 3. Upon reception of the second data segment, the included data sequence mapping allows the server to actually detect that this is not the first segment of a TCP connection. Indeed, the relative subflow sequence number inside the DSS-mapping is actually 100, indicating that this segment is already further ahead in the TCP stream. This allows the server to actually reconstruct the initial sequence number based on the sequence number in the TCP-header ((X+100) - 100) that has been provided by the client and verify whether its SYN-cookie is correct. Thus, no TCP-reset is being sent - in contrast to regular TCP, where the server cannot verify the SYN-cookie. The server knows that the received segment is not the first one of the data stream and thus it can store it temporarily in the out-of-order queue of the connection. It must be noted that the server is not yet able to fully reconstruct the MPTCP state. In order to do this it still must await the MP_CAPABLE_ACK option that is provided in the first data segment.

The server responds to the out-of-order data with a duplicate ACK. The duplicate ACK may also have SACK data if SACK was negotiated. However, if this duplicate ACK does not have an MPTCP level Data ACK, the client may interpret this as a fallback to TCP. This is because the client cannot determine if an option stripping middlebox removed the MPTCP option on TCP segments after connection establishment. So even though the server has not fully recreated the MPTCP state at this point, it should respond with a Data ACK set to the Data Sequence Number Y-100. The client's TCP implementation may

retransmit the first data segment after a TCP retransmit timeout or it may do so as part of an Early Retransmit that can be triggered by an ACK arriving from the server.



Multipath TCP's DSS option allows to handle the loss of the first data segment as the host can infer the initial sequence number.

Figure 3

4. Alternative solutions

An alternative solution to creating the MP_CAPABLE_ACK option would have been to emit the MP_CAPABLE-option together with the DSS-option on the first data segment. However, as the MP_CAPABLE option is 20 bytes long and the DSS-option (using 4-byte sequence numbers) consumes 16 bytes, a total of 36 bytes of the TCP option space would be consumed by this approach. This option has been dismissed as it would prevent any other TCP option in the first data segment, a constraint that would severely limit TCP's extensibility in the future.

5. IANA Considerations

Our proposal requires the change of the MPTCP-version number.

6. Security Considerations

Sending the keys in a reliable way after the three-way handshake implies that there is a larger window during which an on-path attacker might modify the keys that are being sent in the MP_CAPABLE_ACK. However, we do not think that this can actually be

considered as a security issue. If an attacker modifies the keys, the outcome will be that the client and the server won't agree anymore on the data-sequence numbers. The data-flow will thus stall. Considering that the attacker has to be an active on-path attacker to launch this attack, he has already other means of interfering with the connection. Thus, this attack is considered as irrelevant.

Further, if servers implement the proposal from Section 3.2, to handle the scenario where the first data-segment is lost, the incoming segments need to be stored in the out-of-order queue. The server will store these segments without having verified the key that the client provides in the MP_CAPABLE option. This might be considered as a security risk where an attacker could consume buffer space in the server. It must be noted however that in order to achieve this, the attacker needs to correctly guess the SYN-cookie so that the verification described in Section 3.2 is successful. As MPTCP does not try to be more secure than regular TCP, this thread can be considered acceptable, as it uses the same level of security as regular TCP's SYN-cookies. Nevertheless, servers are free to avoid storing those segments in the out-of-order queue if the thread is considered important enough.

7. Acknowledgments

We would like to thank Olivier Bonaventure, Yoshifumi Nishida and Alan Ford for their comments and suggestions on this draft.

8. References

8.1. Normative References

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, August 2007.

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

8.2. Informative References

[I-D.barre-mptcp-tfo] Barre, S., Detal, G., and O. Bonaventure, "TFO support for Multipath TCP", draft-barre-mptcp-tfo-01 (work in progress), January 2015.

[M08] McManus, P., "Improving syncookies", 2008, <http://lwn.net/Articles/277146/>.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.

[RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, "TCP Extensions for High Performance", RFC 7323, September 2014.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, December 2014.

Authors' Addresses

Christoph Paasch
Apple, Inc.
Cupertino
US

Email: cpaasch@apple.com

Anumita Biswas
Apple, Inc.
Cupertino
US

Email: anumita_biswas@apple.com

Darren Haas
Apple, Inc.
Cupertino
US

Email: dhaas@apple.com

INTERNET-DRAFT
Intended Status: Experimental
Expires: Apr 19, 2016

A. Palanivelan
Verizon Labs
Chetan Harsha
Verizon Labs
Senthil Sivakumar
Cisco Systems
Oct 17, 2015

MPTCP Enhancement Opportunities
draft-palanivelanchetansenthil-mptcp-enhancements-00

Abstract

MPTCP Intends to address a wide range of issues, with minimal implementation tweaks. Though this works in a range of use cases,there are some use cases, where some standard implementation recommendations could help. The Purpose of this draft is to document Opportunities, where Enhancements to MPTCP can translate to more wider deployments.

Table of Contents

1 Introduction

The Scope of the use cases discussed is limited to impact on end-user experience only and recommended updates at SP (PE Router). The initial versions of this draft would document findings from tests covering various end-user use cases in detail, that presents mptcp enhancement opportunities. The later versions of the document would strive to provide solutions for the documented usecase scenarios.

1.1 Terminology

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 RFC 2119 [RFC2119].

2 MPTCP Enhancement Opportunities - End user Use cases

2.1 Short Flows vs Long Flows

Internet traffic MUST have Security, Throughput, Reliability,.. taken care across different network conditions, modes of access and flows. Data access can be categorized into short or long flows.

Too many Small Flows => Higher Number of Transactions. But, much less Bandwidth Consumption

Can we achieve Low latency for short flows?

Average completion of flow with mptcp can be higher than completion time without mptcp With Bunch of Short Flows, MPTCP may negatively impact throughput

Even a single lost packet can force an entire connection to wait for an RTO.

Far Lesser Long Flows => Lesser Number of Transactions. But, higher Bandwidth Consumption

Can we achieve higher Throughput for Long Flows Without compromising on performance?

How do we maintain Reliability? How do we manage tolerance to sudden and high bursts of traffic?

In Summary, Both long and short flows are important from the enduser perspective. We need to come up with appropriate definition and clear demarcation for short and long flows, from MPTCP Perspective. These need be dealt differently (Probably with multiple profiles).

2.2 Application based Path selection and Adaptive buffering

How much of benefit it would be when we consider different type of applications, for better mptcp profiling. Typical internet applications are categorized as Elastic and InElastic.

Elastic vs Inelastic Applications..How does it matter to MPTCP?

MPTCP performance is impacted :

When the size of the receive buffer is limited.

Path with high RTT may result in the receive buffer size growing beyond the allowed maximum

Diversified RTT

Different ways of handling packets => Better Performance.

In Summary, Application based Path Selection and Adaptive Buffering can help with the above scenarios. Tweaking the buffer sizes based on the type of application and/or network condition can positively impact the flows.

2.3 Path Selection Enhancements

Path Selection is one of the important part of MPTCP. Though there are existing tools that help diagnose issues in the path, there still is scope to fine tune it further flexible based on certain factors.

Usecases where MPTCP path selection can be enhanced:

For High packet loss and High latency networks?

Multiple profiles to dynamically switch (move across) the networks?

Roaming scenarios

In Summary, The best optimal path is ever changing in the Internet. Frequent switching may cause unnecessary overheads and can impact performance. Enhanced yet controlled Path Selection and Path Switching can help get better performance out of the network.

2.4 Optimal number of paths

The best and effective path selection is critical to the effect of MPTCP for the client application. How about the optimal number of sub-flows? Can we improve client experience by controlling number of sub flows based on certain factors?

Controlling the number of sub flows getting created:

How many is too many?

Can this be controlled? What Inputs to Consider?

- Based on Network Characteristics
- Historic data (region wise)

In Summary, MPTCP being not too strict as well as not too flexible, Certain profiling based on detailed analysis of data can positively impact MPTCP experience

3 UseCase Scenarios (Simulated in Lab) and Results

Data for enhancement opportunities are derived from our lab tests. These tests are done in a reasonably populated, yet contained test network. The initial set of tests are more focused on the throughput side and covers simulated Near, Mid and Far cell network conditions. The Intention is to get detailed data from set of tests to cover different types of data access (short/long or elastic/inelastic applications, mobile network conditions,..etc) as well as different mptcp profiles (for eg. number of sub flows). The detailed analysis and summary would be presented in the later sections of the document, followed by design/implementation recommendations for the SPs.

3.1 MPTCP Enabled Client Uploads data from Non-MPTCP Capable Server

<Shared in IETF-94 WG Discussion.. Will be updated here>

3.2 MPTCP Enabled Client Uploads data from MPTCP Enabled Server

<Shared in IETF-94 WG Discussion.. Will be updated here>

3.3 MPTCP Enabled Client Uploads data from Non-MPTCP Capable Server with Intermediate MPTCP Enabled devices (proxy?)

<Shared in IETF-94 WG Discussion.. Will be updated here>

3.4 MPTCP Enabled Client Uploads data from MPTCP Enabled Server with Intermediate MPTCP Enabled devices (proxy?)

<Shared in IETF-94 WG Discussion.. Will be updated here>

3.5 MPTCP Enabled Client Downloads data from Non-MPTCP Capable Server

<Shared in IETF-94 WG Discussion.. Will be updated here>

3.6 MPTCP Enabled Client Downloads data from MPTCP Enabled Server

<Shared in IETF-94 WG Discussion.. Will be updated here>

3.7 MPTCP Enabled Client Downloads data from Non-MPTCP Capable Server with Intermediate MPTCP Enabled devices (proxy?)

<Shared in IETF-94 WG Discussion.. Will be updated here>

3.8 MPTCP Enabled Client Downloads data from MPTCP Enabled Server with Intermediate MPTCP Enabled devices (proxy?)

<Shared in IETF-94 WG Discussion.. Will be updated here>

4 Security Considerations

None

5 IANA Considerations

None

6 References

6.1 Normative References

[RFC2119] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

6.2 Informative References

[RFC6182] Ford, A., Raiciu, C., Handley, M., Barre, S., and J.Iyengar, "Architectural Guidelines for Multipath TCP Development", RFC 6182, March 2011.

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

Author’s Addresses

Palanivelan Appanasamy
Manager Technology, Verizon Labs
Bangalore, India
Email: palanivelan.appanasamy@verizon.com

Chetan Harsha
R&D Software Engineer, Verizon Labs
Bangalore, India
Email: chetan.harsha@verizon.com

Senthil sivakumar
Principal Engineer, Cisco systems
Durham, NC
Email: ssenthil@cisco.com