           Making Multipath TCP robust for stateless webservers
                    draft-paasch-mptcp-syncookies-01

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.

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 6, 2016.

Copyright Notice

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. As the client is initiating the data
   transfer to the server, and this data is being delivered reliably,
   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. 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. The server behaves
in a stateless manner and thus has generated it's own key in a
verifiable fashion (e.g., as a hash of the 4-tuple, sequence number
and a local secret - similar to what is done for the TCP-sequence
number in case of SYN-cookies [RFC4987]). It is thus able to verify
whether it is indeed the originator of the key echoed back in the
MP_CAPABLE_ACK 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.

```
                    1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +---------------+---------------+-------+----------------------+
   |     Kind      |   Length=24   |Subtype|Version|A|B|C|D|E|F|G|H|
   +---------------+---------------+-------+----------------------+
   |                   Sender's Key (64 bits)                     |
   |                                                              |
   +---------------+---------------+-------+----------------------+
   |                  Receiver's Key (64 bits)                    |
   |                                                              |
   +--------------------------------------------------------------+
   | Data-Level Length (2 octets)  | Checksum (2 octets, optional)|
   +--------------------------------------------------------------+
```
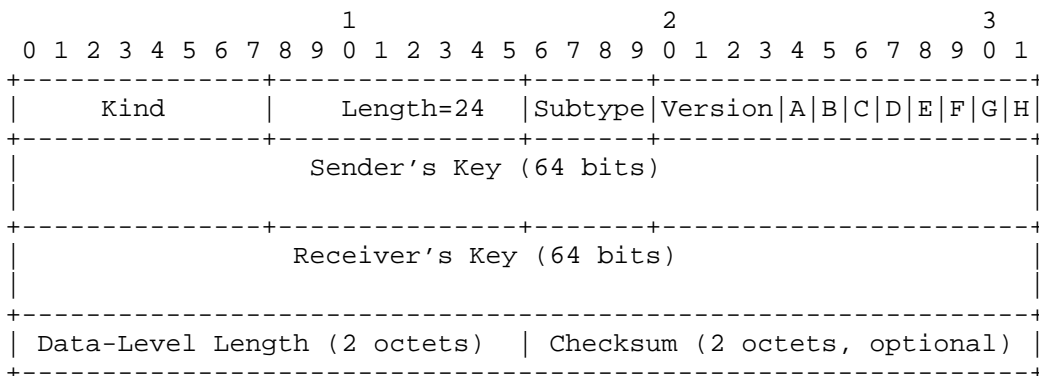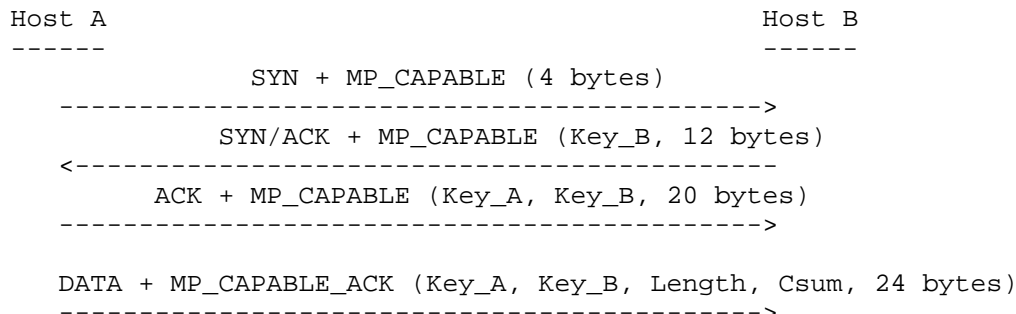
                  Format of the MP_CAPABLE_ACK option.

                                Figure 1

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

third 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. Thus, the
MP_CAPABLE handshake suggested within this document is as follows:

```
   Host A                                          Host B
   ------                                          ------
              SYN + MP_CAPABLE (4 bytes)
        --------------------------------------------->
              SYN/ACK + MP_CAPABLE (Key_B, 12 bytes)
        <---------------------------------------------
          ACK + MP_CAPABLE (Key_A, Key_B, 20 bytes)
        --------------------------------------------->

       DATA + MP_CAPABLE_ACK (Key_A, Key_B, Length, Csum, 24 bytes)
        --------------------------------------------->
```

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

                             Figure 2

It must be said that if TCP Fastopen [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. This is not an issue as the
DATA_ACK is optional in the DSS option. However, the client
receiving this data will have to acknowledge it with a DATA_ACK. If
the client has no data to send itself, this DATA_ACK must also
include the MP_CAPABLE option. This is necessary, because it must be
made sure that the server receives the MP_CAPABLE option. The client
has to combine DATA_ACK with MP_CAPABLE option until he receives a
DATA_ACK from the server, which confirms that the server correctly
received the keys from the client inside the MP_CAPABLE option.
Thus, after reception of the MP_CAPABLE, the server is required to
reply with a DATA_ACK, to signal to the client that it successfully
created the MPTCP-state. Combining the MP_CAPABLE with the DATA_ACK
will require 20 + 8 bytes, which still leaves 12 bytes for the TCP
timestamp option. 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.

Further, as the MP_CAPABLE_ACK option is combined with data, a client
can piggyback data on the 3rd ACK using the MP_CAPABLE_ACK option

instead of the MP_CAPABLE option on this ACK. The reliability of the
MP_CAPABLE_ACK ensures that the server can still reconstruct the
state.

3.1.1. 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.

3.1.2. 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.

3.1.3. 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:

o 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.

o 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). 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]. However, 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.

o 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.

o 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
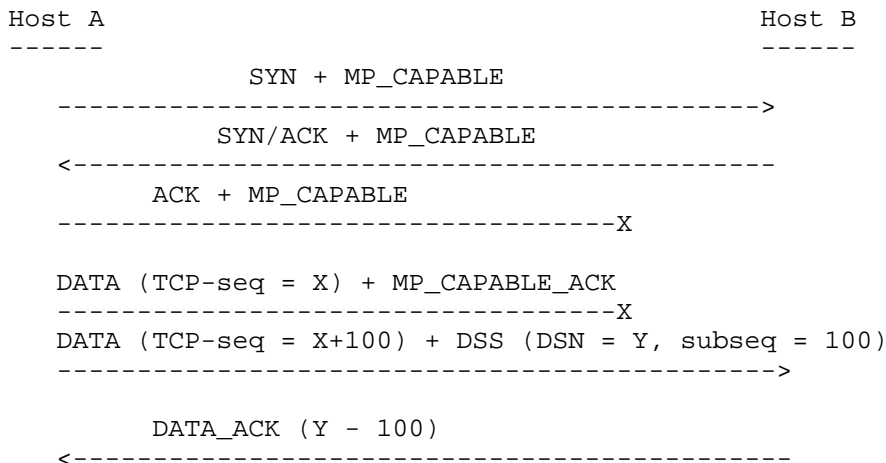   as defined in [RFC6824].

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.

```
        Host A                                        Host B
        ------                                        ------
                    SYN + MP_CAPABLE
            --------------------------------------------->
                   SYN/ACK + MP_CAPABLE
            <---------------------------------------------
              ACK + MP_CAPABLE
            ---------------------------------X

          DATA (TCP-seq = X) + MP_CAPABLE_ACK
            ---------------------------------X
          DATA (TCP-seq = X+100) + DSS (DSN = Y, subseq = 100)
            --------------------------------------------->

              DATA_ACK (Y - 100)
            <---------------------------------------------
```

   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.

7. Acknowledgments

   We would like to thank Olivier Bonaventure 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