MPTCP Working Group S. Barre
Internet-Draft G. Detal
Intended status: Informational Tessares
Expires: May 31, 2018 O. Bonaventure
UCLouvain and Tessares
C. Paasch
Apple
November 27, 2017

TFO support for Multipath TCP
draft-barre-mptcp-tfo-02

Abstract

TCP Fast Open (TFO) is a TCP extension that allows sending data in the SYN, instead of waiting until the TCP connection is established. This document describes what parts of Multipath TCP must be adapted to support it, and how TFO and MPTCP can operate together.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 2
2. TFO cookie request with MPTCP 3
3. Data sequence mapping under TFO 4
4. Early context creation in server 4
5. Using TFO to avoid useless MPTCP negotiations 5
6. Using TFO with MP_JOIN 6
7. Connection establishment examples 6
8. Middlebox interactions 8
9. Security considerations 9
10. Conclusion 10
11. Acknowledgements 10
12. Informative References 10
Appendix A. Implementation status 10
Authors' Addresses 11

1. Introduction

TCP Fast Open, described in [RFC7413], has been introduced with the objective of gaining one RTT before transmitting data. This is considered a valuable gain as very short connections are very common, especially for HTTP request/response schemes. MPTCP, on the other hand, has been defined in [I-D.ietf-mptcp-rfc6824bis] to add multipath support to TCP, where a TCP flow is divided in several TCP subflows. Given that MPTCP can be applied transparently to any TCP socket, without the application knowing, it should be able to support TCP fast open when the application asks for it.

When doing that, one important thing to examine is the option length consumed in segments that would carry both a TFO and an MPTCP option. The handling of MPTCP data sequence mappings must also be updated to take into account the data that is sent together with the SYN or the SYN+ACK. A third issue to handle is the state creation in the server: TFO allows the server to create TCP state as soon as a SYN is received. With MPTCP, even more state is created, and it may be useful to avoid this in a situation where MPTCP does not work but TFO does.

The rest of this document is organized as follows:

Section 2 describes the TFO cookie request, in the case of a Multipath TCP flow. Section 3 proposes a way to map SYN data to the data sequence number space, while taking middleboxes into account.

In Section 4, it is explained that the MP_CAPABLE option is no longer always necessary in the third ack of the three-way handshake. Section 5 presents two ways to avoid useless MPTCP context creations in the server, one for client implementations, the other for server implementations, as a TFO extension. Section 6 takes the MP_JOIN case into consideration. Finally, we describe middlebox interactions in Section 8, and security considerations in Section 9.

2. TFO cookie request with MPTCP

When a TFO client first connects to a server, it cannot immediately include data in the SYN for security reasons [RFC7413]. Instead, it requests a cookie that will be used in subsequent connections. This is done with the TCP cookie request/response options, of resp. 2 bytes and 6-18 bytes (depending on the chosen cookie length).

TFO and MPTCP can be combined provided that the total length of their options does not exceed the maximum 40 bytes possible in TCP:

- In the SYN: MPTCP uses a 4-bytes long MP_CAPABLE option. The MPTCP and TFO options sum up to 6 bytes. [I-D.ietf-mptcp-rfc6824bis] mentions in Appendix A that SYN packet options typically sum up to 19 bytes, or 24 bytes where implementations pad each option up to a word boundary. Even in the worst case, this fits the maximum option space.

- In the SYN+ACK: MPTCP uses a 12-bytes long MP_CAPABLE option, but now TFO can be as long as 18 bytes. Since the maximum option length may be exceeded, it is up to the server to solve this by using a shorter cookie or pad the whole option block instead of each option separately. Alternatively, the server may decide to fallback to MPTCP-only (by not giving a cookie at all), or to TFO-only. As an example, if we consider that 19 bytes are used for classical TCP options, the maximum possible cookie length would be of 7 bytes. The consequence of this, from a security viewpoint, is explored in Section 9. Note that the same limitation applies to subsequent connections, for the SYN packet (because the client then echoes back the cookie to the server). Finally, if the security impact of reducing the cookie size is not deemed acceptable, the server can reduce the amount of other TCP-options by omitting the TCP timestamps. Indeed, as outlined in [I-D.ietf-mptcp-rfc6824bis] in Appendix A, an MPTCP connection could also avoid the use of TCP timestamps thanks to MPTCP's use of 64-bit sequence numbers which already provides protection against wrapped sequence numbers.

- In the third ACK: Nothing special compared to MPTCP, since no TFO option is used there.

Once the cookie has been successfully exchanged, the rest of the connection is just regular MPTCP. The rest of this document assumes that the cookie request has been exchanged, and that data can be included in the SYN.

3. Data sequence mapping under TFO

MPTCP [I-D.ietf-mptcp-rfc6824bis] uses, in the TCP establishment phase, a key exchange that is used to generate the Initial Data Sequence Numbers (IDSNs). More precisely, [I-D.ietf-mptcp-rfc6824bis] states in section 3.1 that "The SYN with MP_CAPABLE occupies the first octet of data sequence space, although this does not need to be acknowledged at the connection level until the first data is sent". With TFO, one way to handle the data sent together with the SYN would be to consider an implicit DSS mapping that covers that SYN segment (since there is not enough space in the SYN to include a DSS option). The problem with that approach is that if a middlebox modifies the TFO data, this will not be noticed by MPTCP because of the absence of a DSS-checksum. For example, a TCP (but not MPTCP)-aware middlebox could insert bytes at the beginning of the stream and adapt the TCP checksum and sequence numbers accordingly. With an implicit mapping, this would give to client and server a different view on the DSS-mapping, with no way to detect this inconsistency as the DSS checksum is not present. One way to solve this is to simply consider that the TFO data is not part of the Data Sequence Number space: the SYN with MP_CAPABLE still occupies the first octet of data sequence space, but then the first non-TFO data byte occupies the second octet. This guarantees that, if the use of DSS-checksum is negotiated, all data in the data sequence number space is checksummed. We also note that this does not entail a loss of functionality, because TFO-data is always sent when only one path is active.

4. Early context creation in server

In order to enable the server to receive and send data before the end of the three-way handshake, TFO allows creating state on the server as soon as the SYN is received if a valid cookie is provided. The MPTCP state should then also be created upon SYN reception (see exceptions for that in Section 5).

DISCUSSION: Doing that allows relaxing the MPTCP MP_CAPABLE exchange, in that the sender's and receiver's keys are no longer required in the third ack of the three-way handshake, because their role was precisely to compensate for the absence of server state until the end of the establishment. The consequence is that the MP_CAPABLE option can simply be removed from the third ack. However, an MPTCP option must still be present when concluding the three-way handshake, to confirm to the server that its own MP_CAPABLE option (in the SYN+ACK) has been correctly received by the client. The DSS option can replace the MP_CAPABLE option, while simultaneously allowing the transmission of more data in the third ack. Moreover, providing a DSS option to the server early allows faster establishment of new subflows (see [I-D.ietf-mptcp-rfc6824bis], Section 3.1).

In order to decide whether it can send a third ack with DSS-only instead of MP_CAPABLE, a client must verify if the TFO data has been at least partially acknowledged. If the SYN+ACK only acknowledges the SYN, TFO may be not supported in the server, or the cookie may have been filtered by the network. There is no guarantee that the MPTCP state has been created, and the third ack should contain the MP_CAPABLE option, with the client and server keys.

5. Using TFO to avoid useless MPTCP negotiations

The TFO cookie, sent in a SYN, indicates that a previous connection has been successfully established, and that TCP state can safely be created. It does not however say anything about whether the MPTCP options are filtered or not in the network. It is thus possible that a server creates an MPTCP context upon SYN+TFO cookie reception, then actually needs to discard it after having discovered that the MPTCP options are filtered.

One way to solve this would be for the client to cache destinations that do support MPTCP. TFO allows sending data together with the SYN starting at the second connection. The first one is used to learn the cookie from the server. It could also be used to learn whether MPTCP can be used with the peer.

DISCUSSION: The other, compatible way to solve the problem is to extend TFO and cache the Multipath Capability in the cookie generated by the server. The server could modify its cookie computation, to include multipath capability information in the cookie. Then, upon SYN+TFO cookie reception, the server could easily determine if the initial TFO flow was a successful MPTCP connection or not. The problem with this approach is that the server does not know yet whether the flow is multipath-capable when sending the TFO cookie. It could then send a first pessimistic cookie, as GetCookie(IP_Address, mp_capable=false) (adapted from [RFC7413], Section 4.1.2). Then, when it is determined that the flow is Multipath Capable (third ack received with an MPTCP option), a new cookie=GetCookie(IP_Address, mp_capable=true) can be generated and sent in the FIN to ensure reliable delivery.

6. Using TFO with MP_JOIN

TFO must not be used when establishing joined subflows. Doing that would be in contradiction with [I-D.ietf-mptcp-rfc6824bis], that states in section 3.2 that "It is not permitted to send data while in the PRE_ESTABLISHED state". Using TFO with joined subflows would mean that data is sent even before getting to the PRE_ESTABLISHED state.

7. Connection establishment examples

In this section we show a few examples of possible TFO+MPTCP establishment scenarios. For representing segments, we use the Tcpdump syntax.

Before a client can send data together with the SYN, it must request a cookie to the server, as shown in Figure 1. This is done by simply combining the TFO and MPTCP options.

```
client                                                          server
  |                                                               |
  |    S  0(0) <MP_CAPABLE>,<TFO cookie request>                  |
  | ------------------------------------------------------------> |
  |                                                               |
  |    S. 0(0) ack 1 <MP_CAPABLE>,<TFO cookie>                    |
  | <------------------------------------------------------------ |
  |                                                               |
  |    .  0(0) ack 1 <MP_CAPABLE>                                 |
  | ------------------------------------------------------------> |
  |                                                               |
```

Figure 1: Cookie request

Once this is done, the received cookie can be used for TFO, as shown in Figure 2. The MP_CAPABLE is no longer required for the third ack, as explained in Section 4. Note that the last segment in the figure has a TCP sequence number of 21, while the DSS subflow sequence number is 1 (because the TFO data is not part of the data sequence number space, as explained in Section 3.

```
client                                                        server
   |                                                             |
   |     S  0(20) <MP_CAPABLE>,<TFO cookie>                      |
   | ----------------------------------------------------------> |
   |                                                             |
   |     S. 0(0) ack 21 <MP_CAPABLE>                             |
   | <---------------------------------------------------------- |
   |                                                             |
   |     .  1(100) ack 21 <DSS ack=1 seq=1 ssn=1 dlen=100>       |
   | <---------------------------------------------------------- |
   |                                                             |
   |     .  21(20) ack 101 <DSS ack=101 seq=1 ssn=1 dlen=20>     |
   | ----------------------------------------------------------> |
   |                                                             |
```

Figure 2: The server supports TFO

In Figure 3, the server does not support TFO. The client detects that no state is created in the server (as no data is acked), and now sends the MP_CAPABLE in the third ack, in order for the server to build its MPTCP context at then end of the establishment. Now, the tfo data, retransmitted, becomes part of the data sequence mapping because it is effectively sent (in fact re-sent) after the establishment.

```
client                                                        server
   |                                                             |
   |     S  0(20) <MP_CAPABLE>,<TFO cookie>                      |
   | ----------------------------------------------------------> |
   |                                                             |
   |     S. 0(0) ack 1 <MP_CAPABLE>                              |
   | <---------------------------------------------------------- |
   |                                                             |
   |     .  21(0) ack 1 <MP_CAPABLE>                             |
   | ----------------------------------------------------------> |
   |                                                             |
   |     .  1(20) ack 1 <DSS ack=1 seq=1 ssn=1 dlen=20>          |
   | ----------------------------------------------------------> |
   |                                                             |
   |     .  0(0) ack 21 <DSS ack=21 seq=1 ssn=1 dlen=0>          |
   | <---------------------------------------------------------- |
```

Figure 3: The server does not support TFO

It is also possible that the server acknowledges only part of the TFO data, as illustrated in Figure 4.

```
client                                                        server
   |                                                             |
   |  S  0(1000) <MP_CAPABLE>,<TFO cookie>                       |
   | ----------------------------------------------------------> |
   |                                                             |
   |  S. 0(0) ack 501 <MP_CAPABLE>                               |
   | <---------------------------------------------------------- |
   |                                                             |
   |   .  501(500) ack 1 <DSS ack=1 seq=1 ssn=1 dlen=500>        |
   | ----------------------------------------------------------> |
   |                                                             |
```

Figure 4: Partial data acknowledgement

8. Middlebox interactions

[I-D.ietf-mptcp-rfc6824bis], Section 6, describes middlebox interactions for Multipath TCP. This document does not define any new option compared to MPTCP or TFO. It defines a combination of them.

TFO also defines how an implementation should react when the TFO SYN is lost (fallback to regular TCP, [RFC7413] Section 4.2.1).

We propose to remove the MP_CAPABLE option from the third ack when TFO is used, based on the assumption that the context has been created already in the server upon SYN reception. Should the server actually not create this state, it would not be able to create its MPTCP state and would fallback to regular TCP. The state is not created in the server if it has no TFO support or the cookie is invalid, but in that case only the SYN is acknowledged, and the client does send the MP_CAPABLE option.

The other case where the server does not create MPTCP state is when the cookie includes a "mp_capable=false" information. In that case, regular TCP is used to take into account middleboxes that prevent correct MPTCP operation.

Even though this document presents mechanisms for collaboration between MPTCP and TFO, the filtering of one will not stop the other from working. For example, if a TFO option is dropped, MPTCP will fallback to sending MP_CAPABLE in third_ack, because no TFO data is acked. If the server stores MPTCP information in the cookie, this will be completely opaque to the network, and even to the client. Should that cookie be transformed or lost, it would not be accepted anymore by the server, which would fallback to regular MPTCP communication, or regular TCP if MPTCP options are also filtered or modified.

The problem of middleboxes that alter the TFO data is solved by the fact that TFO data is not part of the Data Sequence Number space, as explained in Section 3.

9. Security considerations

Compared to using TFO or MPTCP alone, implementing the present combination could lead to more state created in the server, since MPTCP now creates state as soon as the first SYN is received. This is however not considered as a problem, for the following reasons:

- o The server will only create state when a valid TFO cookie is received. This guarantees that a successful TCP connection has been previously established with the same peer.

- o It remains possible that a useless MPTCP context is created upon SYN reception (due to TFO support but MPTCP options being filtered by the network). This is more an optimization issue than a security issue given the TFO cookie protection already present. Section 5 still proposes a solution to avoid creating MPTCP state in that case.

- o When under memory pressure, a server always has the option to refuse the client cookie. In that case, the session establishment will happen without data, and the client will send the MP_CAPABLE option in the third ack so that the server can create the MPTCP context at that time.

As mentioned in Section 2, it may be required to reduce the length of the cookie when MPTCP and TFO are used together. This can become a security issue when attackers and networks become fast enough for a brute force attack to be successful. An option to solve this would be to use TCP payload to store additional options, as suggested in [I-D.ietf-mptcp-rfc6824bis], Section 5. Another way would be to allow longer TCP options by using an "Extended Data Offset Option" [I-D.touch-tcpm-tcp-edo]. The problem with this is that the most problematic segment in the present case is the SYN (with long TFO cookie and MP_CAPABLE MPTCP option), for which it is more difficult to apply the Extended Data Offset Option ([I-D.touch-tcpm-tcp-edo], Section 7.7).

10. Conclusion

In this document, we have proposed minor extensions to MPTCP and TFO to allow them to operate together. In particular, we proposed excluding the TFO data from the data sequence number space. We explained that TFO allows to relax the MPTCP establishment in that the MP_CAPABLE option of the third ack can be removed in some cases. We also emphasized that such a combination augments the size of the TCP options, already quite large, although the combination is still possible with common TCP options and limited cookie length. We also proposed a way to cache multipath capability information in the client or in the TFO cookie. Finally, we examined potential middlebox interaction problems, or security problems that would arise from that combined operation.

11. Acknowledgements

This work was supported by the FP7-Trilogy2 project and by the Belgian Walloon Region under its FIRST Spin-Off Program (RICE project).

12. Informative References

[I-D.ietf-mptcp-rfc6824bis]
Ford, A., Raiciu, C., Handley, M., Bonaventure, O., and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses", draft-ietf-mptcp-rfc6824bis-09 (work in progress), July 2017.

[I-D.touch-tcpm-tcp-edo]
Touch, J. and W. Eddy, "TCP Extended Data Offset Option", draft-touch-tcpm-tcp-edo-03 (work in progress), July 2014.

[MultipathTCP-Linux]
Paasch, C., Barre, S., and . et al, "Multipath TCP in the Linux kernel", n.d., <http://www.multipath-tcp.org>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.

Appendix A. Implementation status

In this section, we present the report of the implementation of this draft inside the Linux reference implementation of Multipath TCP [MultipathTCP-Linux]. The support of TFO in the MPTCP stack has been implemented on the 3.14 kernel (MPTCP v0.89).

The main design choices of this implementation are the following:

- o Minimize the modification to the current MPTCP and TFO stacks, i.e. let the TFO stack deal with sending data, receiving data inside the SYN.
- o Create the MPTCP state when receiving a SYN with a valid token on the server side as defined in Section 4.
- o Map the remaining data segments in the receive and send buffers to MPTCP data sequence numbers.

This latter point needs further explanation. First, in the current reference implementation of MPTCP, the MPTCP state is created upon reception of the SYN+ACK on the client-side. The implementation however did the MPTCP state allocation before processing the actual acknowledgement at the subflow level. This means that data (even acknowledged by the SYN+ACK) remains in the send buffer at the time of the allocation (which contained only the SYN in the case of regular MPTCP). We modified this behaviour to ensure that only unacknowledged data remains in the send buffer when allocating the state. Moreover, as the data was initially sent over the regular TCP flow, they had no MPTCP sequence numbers (the MPTCP state did not exist during the initial sendto() call). After the allocation of the MPTCP state, we modify these sequence numbers such that they are mapped starting at "IDSN + 1". This effectively gives the data sequence number "IDSN + 1" to the first byte following the establishement, since the acknowledged TFO data has been removed fromt the queues at this point. This data will then follow the same path as for data sent via a regular write() call.

As is the case for unacknowledged data on the client-side, the server-side can also have data in the receive buffer (the data sent in the SYN). We perform the same operation by mapping this data from TCP to MPTCP sequence numbers. TFO data is then mapped ahead of the IDSN, so as to ensure, again, that the first byte following the establishment has the data sequence number "IDSN + 1".

As of this writing, the implementation still generates a regular third acknowledgment with a MP_CAPABLE option (see Section 4) and it does not take benefit from the TFO cache to avoid useless MPTCP negotiation (see Section 5).

Authors' Addresses

Sebastien Barre
Tessares

Email: sebastien.barre@tessares.net

Gregory Detal
Tessares

Email: gregory.detal@tessares.net

Olivier Bonaventure
UCLouvain and Tessares

Email: Olivier.Bonaventure@tessares.net

Christoph Paasch
Apple

Email: cpaasch@apple.com