MPTCP C. Paasch
Internet-Draft O. Bonaventure
Intended status: Experimental UCLouvain
Expires: August 15, 2014 February 11, 2014

A generic control stream for Multipath TCP
draft-paasch-mptcp-control-stream-00

Abstract

Multipath TCP's extensive use of TCP options to exchange control information consumes a significant part of the TCP option space. Extending MPTCP to add more control information into the session becomes cumbersome as the TCP option space is limited to 40 bytes.

This draft introduces a control stream that allows to send control information as part of the subflow's payload. The control stream is mapped into a separate sequence number space and uses a TLV-format for maximum extensibility. It is left to future documents to specify how the TLV-format might be used to exchange control information. As the control stream is sent as part of the subflow's payload, it is not subject to the 40 bytes limitation of the TCP option space.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Multipath TCP [RFC6824] uses the TCP options to exchange control information between the communication hosts. [RFC6824] defines several new TCP options that are used during the three-way handshake and the data transfer. Using options is the standard method to extend the TCP protocol. Unfortunately, the maximum length of the TCP options field is 40 bytes. This severely limit the utilisation of options to exchange control information between communicating hosts. During the three-way handshake, the TCP options space is further limited by the other TCP options that are also included in the SYN and SYN+ACK segments. [RFC6824] did its best to minimize the size of the MP_CAPABLE option inside the SYN and SYN+ACK segments given the presence of other options (typically MSS, timestamp, selective acknowledgements and window scale). However, this has been at the cost of a reduced security due to the utilization of security keys that are too short.

The security requirements for MPTCP ask for a strong authentication of additional subflows [RFC6181]. Given the restriction in the size of the MPTCP options, it seems very difficult to provide strong security by relying only on TCP options that cannot be longer than 40 bytes and are not exchanged reliably. Although a design to overcome these problems would probably be possible, it would add a lot of complexity to the protocol.

Furthermore, today’s MPTCP control information is sent in an unreliable manner. This means that control information like MP_PRIO, ADD_ADDR or REMOVE_ADDRESS might get lost, resulting in potential suboptimal performance of Multipath TCP.

In this document, we show that another design is possible. Instead of using only TCP options to exchange control information, we show how it is possible to define a control stream in parallel with the data stream that is used to exchange data over the established subflows. By using this control stream, two MPTCP hosts can reliably exchange control information without being restricted by TCP option space. The control stream can be used to exchange cryptographic material to authenticate the handshake of additional subflows or for any other purpose.

Together with the control stream, we propose to modify the MPTCP-handshake so that no crypto information is exchanged within the TCP options. We suggest to use the control stream instead. Within the control stream, different key-negotiation schemes can be specified (e.g., reuse SSL-key, tcpcrypt-style, Diffie-Hellman,...)

This document is structured as follows. First, we define how the control stream can be used within an MPTCP session. Section 3 presents the modified MPTCP handshake of the initial subflow, while Section 4 specifies the handshake of additional subflows. Section 5 gives some example use-cases for the key negotiation through the control stream. Finally, Section 6 gives another example on how to use the control stream to conduct the MPTCP session.

2. The control stream

In contrast with SCTP [RFC4960], TCP and Multipath TCP [RFC6824] only support one data stream. SCTP uses chunks to allow the communicating hosts to exchange control information of almost unlimited size. As explained earlier, having a control stream in Multipath TCP would enable a reliable delivery of the control information without strict length limitations.

This section defines a control stream that allows to exchange MPTCP control information of arbitrary length besides the regular data stream. The control stream holds data in a TLV-format and thus any type of data can be added to it. Further, the control stream provides a reliable and in-order delivery of the control data.

The control stream is sent within the payload of the TCP segments. This ensures a reliable delivery of the TLVs exchanged in the control stream. Further, a separate control-sequence number space is defined for the control stream to ensure in-order delivery of the control stream. The Initial Control stream Sequence Number (ICSN) is the same as the IDSN in the respective directions. A DSS-mapping is used within the TCP option space to signal the control stream sequence numbers as well as a control stream acknowledgement. This DSS-mapping option is the same as the one defined in [RFC6824]. To differentiate the control stream from the data stream, we use the last bit of the 'reserved' field of the MPTCP DSS option. We call this bit the Stream (S) bit. When the DSS option is used to map regular data, this bit is set to 0. When the DSS option is used to map one TLV on the control stream, it is set to 1 (see Figure Figure 1)

```
 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     |Subtype|(reserved)|S|F|m|M|a|A|
+---------------+---------------+-------+----------------------+
|        Control ACK (4 or 8 octets, depending on flags)       |
+--------------------------------------------------------------+
|Control sequence number (4 or 8 octets, depending on flags)   |
+--------------------------------------------------------------+
|              Subflow Sequence Number (4 octets)              |
+-------------------------------+------------------------------+
|Control-Level Length (2 octets)|      Checksum (2 octets)     |
+-------------------------------+------------------------------+
```

The S bit of the 'reserved' field is set to 1 when sending on the control stream.

Figure 1

The control information exchanged in the control stream is encoded by using a TLV format, where the type and length are 16-bit values. This allows for maximum extensibility and to use very long data within the control stream. The format of the TLV option is shown in Figure 2

```
 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
+---------------+---------------+-------+----------------------+
|              Type             |            Length            |
+---------------+---------------+-------+----------------------+
|                      Value (Length - 4)                      |
+-------------------------------+------------------------------+
```

The TLV option format

Figure 2

2.1. Window considerations

MPTCP uses the receive-window to do flow-control at the receiver. The receive-window within MPTCP is being used at the data sequence level, however any segment sent on a subflow must obey to the last window-announcement received on this particular subflow with respect to the subflow-level sequence number.

The control stream is no different with respect to this last point. The subflow-sequence numbers used for control stream data must fit within the window announced over this specific subflow. However, to avoid issues of receive-window handling at the control stream sequence number level, a host may never have more than one unacknowledged TLV-field in-flight. This effectively limits the amount of memory required to support the control-stream down to 64KB (the maximum size of a TLV-field).

TCP uses the congestion-window to limit the amount of unacknowledged in-flight data within a TCP connection. The control stream must also obey to this limitation. As the control stream uses regular TCP sequence numbers, the congestion-window limitations apply too.

3. Connection initiation

The control stream allows to negotiate the crypto material to authenticate new subflows. Thus, the handshake of the initial subflow does not need anymore to send the 64-bit key in plaintext. The suggested modification to the initial handshake is detailled in this section.

MultiPath TCP uses the MP_CAPABLE option in the handshake for the initial subflow. This handshake was designed to meet several requirements. When designing another variant of the Multipath TCP handshake, it is important to have these requirements in mind. These requirements are :

1. Detect whether the peer supports MultiPath TCP.

2. Exchange locally unique tokens that unambiguously identify the Multipath TCP connection

3. Agree on an Initial Data Sequence Number to initialize the MPTCP state on each direction of the Multipath TCP connection

Before discussing the proposed handshake, it is important to have in mind how [RFC6824] meets the three requirements above.

The first requirement is simply met by using a Multipath TCP specific option, like all TCP extensions.

To meet the second requirement, a simple solution would have been to encode the token inside the MP_CAPABLE option. However, this would have increased its size. This would have limited the possibility of extending Multipath TCP later by adding new TCP options that require space inside the SYN segments. To minimize the number of option bytes consummed in the SYN segment, [RFC6824] uses a hash function to compute the token based on the keys exchanged in clear. However, using hash functions implies that implementations must handle the possible collisions which increases the complexity of implementing the Multipath TCP handshake.

In this document we suggest a simplified handshake that meets the above three goals. This simplified handshake avoids negotiating the crypto-material during the three-way handshake. Instead, security information is exchanged reliably by relying on the control stream. The figure below provides an overview of the proposed handshake.

```
             Host A                              Host B
            ----------                          ----------
            Address A1                          Address B1
            ----------                          ----------
                |                                   |
                |  SYN+MP_CAPABLE(Token-A)   |
                |---------------------------------->|
                |                                   |
                |SYN/ACK+MP_CAPABLE(Token-B) |
                |<----------------------------------|
                |                                   |
                |  ACK+MP_CAPABLE(Token-A, Token-B)  |
                |                                   |
                |---------------------------------->|
```

Handshake of the initial subflow.

Figure 3

MPTCP's establishment of the initial subflow follows TCP's regular 3-way handshake, but the SYN, SYN/ACK and ACK packets contain the MP_CAPABLE-option. The proposed MP_CAPABLE option contains one 32 bits token in the SYN and SYN/ACK segments. The third ACK includes an MP_CAPABLE option that contains the two tokens. Echoing all the information back in the third ACK allows stateless operation of the server. The tokens are used to explicitly exchange the identifiers of the Multipath TCP connection.

It is required that the server, upon reception of the SYN generates a token different from the client's token. This is necessary to protect against reflection attacks when establishing additional subflows.

The format of the proposed MP_CAPABLE option is proposed in the figures below.

```
                     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     |Subtype|Version|A|B|C|D|E|F|G|H|
+---------------+---------------+-------+-------+---------------+
|                      Sender's Token (32 bits)                 |
+---------------------------------------------------------------+
```

Format of the MP_CAPABLE-option in the SYN and SYN/ACK packets

Figure 4

```
                     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     |Subtype|Version|A|B|C|D|E|F|G|H|
+---------------+---------------+-------+-------+---------------+
|                      Sender's Token (32 bits)                 |
+---------------------------------------------------------------+
|                      Receiver's Token (32 bits)               |
+---------------------------------------------------------------+
```

Format of the MP_CAPABLE-option in the third ACK of the handshake

Figure 5

The format of the MP_CAPABLE option is shown in Figure 4. To indicate that this MP_CAPABLE contains tokens numbers and not keys (as in [RFC6824]), the Version-field is set to 1. The message format of the third ACK's MP_CAPABLE option is show in Figure 5.

The Initial Data Sequence Number (IDSN) serves to initialize the MPTCP state on the end-hosts in the same way as TCP's sequence numbers do during the 3-way handshake. There is one IDSN for each direction of the data-stream. The IDSN for the data from the client to the server is the 64 low-order bits of the hash (SHA1) of the concatenation of the tokens (Token-A || Token-B). For the data from server to client, the IDSN is 64 low-order bits of the hash (SHA1) of the reverse concatenation (Token-B || Token-A). The tokens should be generated with sufficient randomness so that they are hard to guess. Recommendations for generating random numbers are given in [RFC4086].

The meaning of the other fields and behavior of the end-hosts during the MP_CAPABLE exchange is the same as specified in [RFC6824].

4. Starting a new subflow

The handshake for the establishment of a new subflow is similar to the one specified in [RFC6824]. There are three important differences. First, the HMAC is computed by using the keys negotiated over the control stream. Second, the token and the client's random numbers are included inside the third ack to allow stateless operation of the passive opener of an additional subflow. Finally, the token is used within the message of the HMAC. This protects against reflection attacks, as the HMAC cannot be sent in the reverse direction anymore, because the tokens are ensured to be different on both end-hosts.

```
          Host A                                   Host B
         ----------                              ----------
         Address A2                              Address B2
         ----------                              ----------
             |                                        |
             |   SYN + MP_JOIN(Token-B, R-A)          |
             |--------------------------------------->|
             |                                        |
             |   SYN/ACK + MP_JOIN(HMAC-B, R-B)       |
             |<---------------------------------------|
             |                                        |
             |  ACK + MP_JOIN(Token-B, R-A, HMAC-A)   |
             |--------------------------------------->|

         HMAC-A = HMAC(Key, Msg=(Token-B+R-A+R-B))
         HMAC-B = HMAC(Key, Msg=(Token-B+R-B+R-A))
```

Handshake of a new subflow.

Figure 6

In order to allow the Token-B and R-A inside the third ack, the HMAC-A must also be a truncated version of the 160-bit HMAC-SHA1. Thus, HMAC-A is the truncated (leftmost 128 bits) of the HMAC as shown in Figure 6.

The message-format of the MP_JOIN-option in the SYN and the SYN/ACK is the same as in [RFC6824]. As the third ACK includes the Token and the random nonce, the MP_JOIN message format of the third ack is as shown in Figure 7. The length of the MP_JOIN-option in the third ACK is 28 bytes. Thus, there remains enough space to insert the timestamp option in the third ACK.

```
                     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    |Subtype|     |B|   Address ID  |
+---------------+---------------+-------+-------+---------------+
|                                                               |
|                Sender's Truncated HMAC (128 bits)             |
|                                                               |
+---------------------------------------------------------------+
|                Sender's Random Number (32 bits)               |
+---------------------------------------------------------------+
|                     Receiver's Token (32 bits)                |
+---------------------------------------------------------------+
```

Format of the MP_JOIN-option

Figure 7

The semantics of the backup-bit "B" and the Address ID are the same as in [RFC6824].

5. Examples of key negotiation through the control stream

The control stream's primary goal is to negotiate the crypto-material to authenticate additional subflows. Both hosts must agree on which key-negotiation scheme to use over the control stream. The option "key select" of the control stream is of type 1 and it negotiates the available key-negotiation schemes. The value-field of the "key select"-option contains a bitmask of available key-negotiation schemes. The bitmask remains to be defined as the schemes are being defined. The bits within the bitmask are numbered, starting from the leftmost as being '1'.

The key-select must be initiated by one host and answered by the other one. During the initiation, the host offers the available schemes, and the answering host selects one of the offered ones. The hosts need thus to ensure an order among themself of who initiates the "key select" option. A possibility would be that the host with the smaller token initiates the "key select" option.

The following are examples of how the control stream could be used to negotiate the cryptographic material. A proper specification is probably needed for each of them.

5.1. Reusing the application’s TLS key

Within the "key select"-option, this negotiation scheme takes the bit number 1. It signals to the peer that the connection should use a derivate of TLS’s master key to authenticate new subflows with this "MPTCP key". It is required that indeed TLS is being used within the data stream.

As TLS allows to modify the key being used during a TLS session, the control stream might be used to ensure that both end hosts agree on the "MPTCP key" being used at a specific moment in time through the exchange of the hash of the "MPTCP key".

5.2. TLS-like key exchange

It enables a key-negotiation in an TLS-like manner, thus authenticating the client/server through a certificate.

5.3. Tcpcrypt-like key exchange

It uses the control stream, to exchange a secret key in a tcpcrypt-like manner. Optionally, it may include a data-sequence number to define from which moment on the data stream should be encrypted.

6. Other example use cases of the control stream

This shows one example of how the control stream can be used within MPTCP.

6.1. Address signaling

In RFC6824, the address-signaling is achieved through the ADD_ADDRESS and REMOVE_ADDRESS options. These options are sent within the TCP options-space and thus do not benefit from reliable delivery. Further, security-concerns have rosen concerning the ADD_ADDRESS-option. Using the control stream to signal the addition or removal of addresses allows to make these options reliable and provides the space to add any kind of cryptographic material to enhance their security.

7. Security Considerations

TBD

8. Acknowledgments

This work is supported by the European FP7 Project "Trilogy2" under grant agreement 317756.

9. Informative References

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.

[RFC6181] Bagnulo, M., "Threat Analysis for TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6181, March 2011.

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

Authors' Addresses

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

Email: christoph.paasch@uclouvain.be

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

Email: olivier.bonaventure@uclouvain.be