

MPTCP Working Group
Internet-Draft
Intended status: Experimental
Expires: January 4, 2018

O. Bonaventure
Tessares
July 03, 2017

0-rtt TCP converters
draft-bonaventure-mptcp-converters-00

Abstract

This document proposes the utilisation of Transport Converters to aid the deployment of TCP extensions such as Multipath 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 January 4, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
2. Architecture	4
2.1. Differences with SOCKSv5	7
3. The Converter protocol	10
3.1. The Fixed Header	10
3.2. The TLV messages	10
3.2.1. The Connect TLV	11
3.2.2. Extended TCP Header TLV	12
3.2.3. Error TLV	13
3.2.4. The Bootstrap TLV	14
3.2.5. Supported TCP Options TLV	14
4. Examples	16
4.1. Bootstrap	16
4.2. Multipath TCP	17
4.3. TCP Fast Open (TFO)	18
5. Interactions with middleboxes	23
6. Security Considerations	24
7. IANA Considerations	25
8. Conclusion	26
9. Acknowledgements	27
10. References	28
10.1. Normative References	28
10.2. Informative References	28
Author's Address	30

1. Introduction

Transport protocols like TCP evolve regularly [RFC7414]. Given the end-to-end nature of those protocols, a new feature can only be used once it has been deployed on both clients and servers. Experience with TCP extensions reveals that the deployment of a new TCP extension requires several years or more [Fukuda2011].

There are some situations where the transport stack used on clients (resp. servers) can be upgraded at a faster pace than the transport stack running on servers (resp. clients). In those situations, clients would typically want to benefit from the features of an improved transport protocol even if the servers have not yet been upgraded and conversely. In the past, Performance Enhancing Proxies have been proposed and deployed [RFC3135] as solutions to improve TCP performance over links/networks with specific characteristics.

Recent examples of TCP extensions include Multipath TCP [RFC6824] or TCPINC [I-D.ietf-tcpinc-tcpcrypt]. Those extensions provide features that are interesting for clients such as smartphones. With Multipath TCP, smartphones could seamlessly use WiFi and cellular networks, either for bonding purposes or for faster handovers. Unfortunately, deploying those extensions on both a wide range of clients and a wide range of servers remains difficult.

In this document, we propose the utilisation of Transport Converter. A Transport Converter is a network function that is installed by a network operator to aid the deployment of TCP extensions. A Transport Converter operates entirely in the transport layer and supports one or more TCP extensions. Their main advantage is that they enable new TCP extensions to be used on a subset of the end-to-end path, which encourages the deployment of this extension.

This document is organised as follows. We first provide a brief explanation of the operation of Transport Converters in Section 2. We compare them in Section 2.1 with SOCKS proxies that are already used to deploy Multipath TCP in cellular networks [IETFJ16]. We then describe the Converter protocol in Section 3 and illustrate its usage with a few examples in Section 4. We then discuss the interactions with middleboxes (Section 5) and the security considerations (Section 6).

2. Architecture

Our architecture considers three types of hosts:

- o Client endhosts
- o Transport Converters
- o Server endhosts

We do not mandate anything on the server side. Our architecture assumes that new software will be installed on the Client hosts and on Transport Converters.

A Transport Converter is a network function that relays all data exchanged over one upstream connection to one downstream connection and vice versa. The converter thus maintains state that associates one upstream connection to a corresponding downstream connection. The main benefit of the Transport Converter is that different transport protocol extensions can be used on the upstream and the downstream connection. This encourages the deployment of new TCP extensions until they are supported by all Servers.

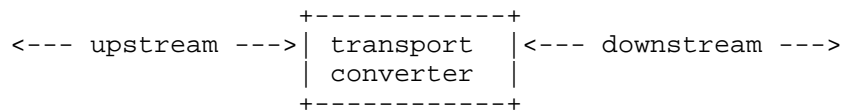


Figure 1: A Transport Converter relays data between pairs of transport connections

Transport converters can be operated by either the network operator or third parties. The Client is configured, through means that are outside the scope of this document, with the names and/or the addresses of one or more Transport Converters. The packets belonging to a transport connection that pass through a transport converter will typically follow a different path than the packets directly exchanged between the Client and the Server. Deployments should minimise this additional delay by carefully selecting the location of the Transport Converter used to reach a given destination.

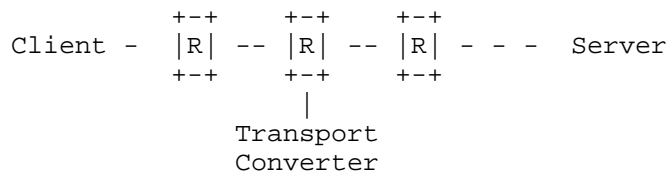


Figure 2: A Transport Converter can be installed anywhere in the network

When establishing a transport connection, the Client can, depending on its policies, either contact the Server directly (e.g. by sending a TCP SYN towards the Server) or create the connection via a Transport Converter. In the latter case, the Client initiates a connection towards the Transport Converter and indicates the address of the final Server inside the connection establishment packet (shown between brackets in Figure 3). This enables the Transport Converter to immediately initiate a connection towards the final Server. The Transport Converter waits until the confirmation that the Server agrees to establish the connection before confirming it to the Client. Figure 3 illustrates the establishment of a TCP connection by the Client through a Transport Converter. The information shown between brackets is part of the Converter protocol described later in this document.

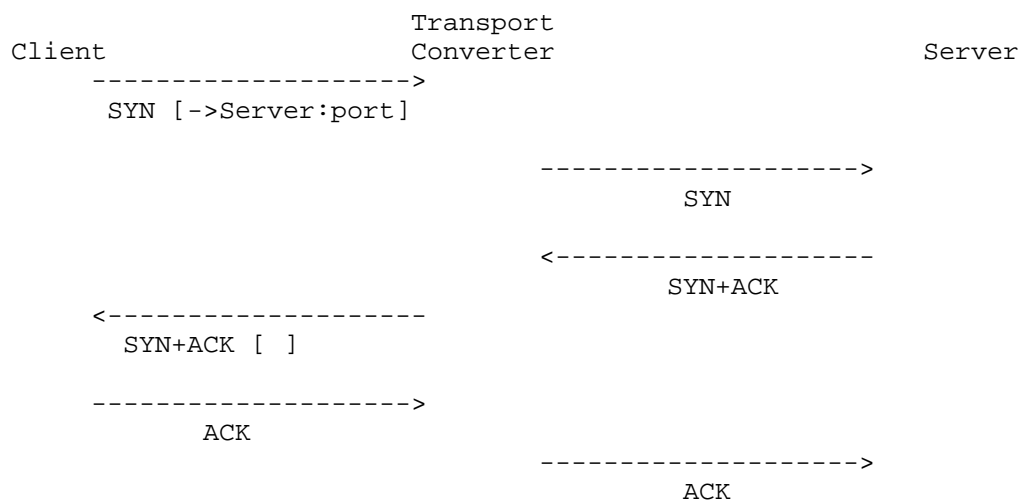


Figure 3: Establishment of a TCP connection through a Converter

As shown in Figure 3, the Converter protocol places its information

inside the handshake packets. This information is encoded in a way that separates this information from the user data that can also be carried inside the payload of such packets [RFC7413]. With TCP, the Converter protocol places the destination address and ports of the final Server inside the SYN. The SYN+ACK packet returned by the Transport Converter to the Client contains information that confirms the establishment of the connection between the Transport Converter and the final Server. It is important to note that the Transport Converter maintains two transport connections that are combined together. The upstream connection is the one between the Client and the Transport Converter. The downstream connection is between the Transport Converter and the final Server. Any user data received by the Transport Converter over the upstream (resp. downstream) connection is relayed over the downstream (resp. upstream) connection to give to the Client the illusion of an end-to-end connection.

As an example, let us consider how such a protocol can help the deployment of Multipath TCP [RFC6824]. We assume that both the Client and the Transport Converter support Multipath TCP but consider two different cases depending on whether the Server supports Multipath TCP or not. A Multipath TCP connection is created by placing the MP_CAPABLE (MPC) option inside the SYN sent by the Client. Figure 4 describes the operation of the Transport Converter if the Server does not support Multipath TCP.

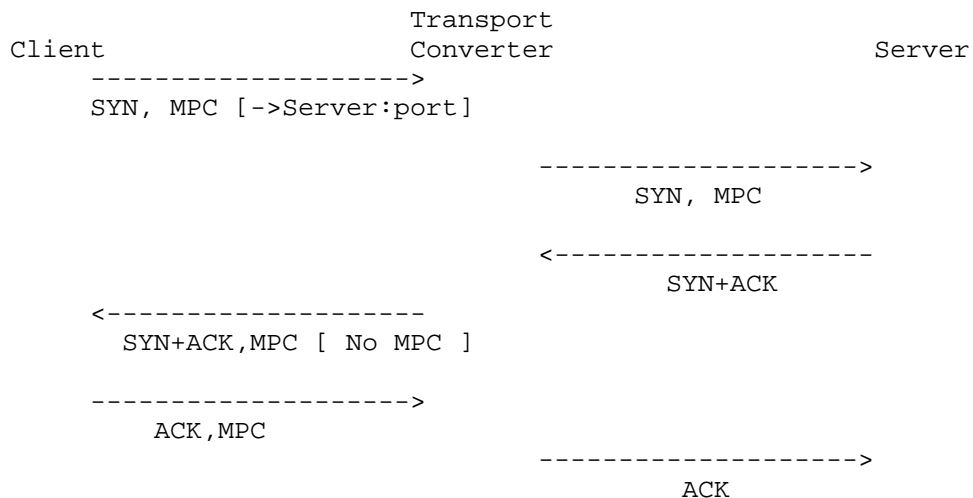


Figure 4: Establishment of a Multipath TCP connection through a Converter

The Client tries to initiate a Multipath TCP connection by sending a

SYN with the MP_CAPABLE option (MPC in Figure 4). The SYN includes the address and port of the final Server and the Transport Converter attempts to initiate a Multipath TCP connection towards this Server. Since the Server does not support Multipath TCP, it replies with a SYN+ACK that does not contain the MP_CAPABLE option. The Transport Converter notes that the connection with the Server does not support Multipath TCP and informs the Client that the remote Server does not support Multipath TCP.

Figure 5 considers a Server that supports Multipath TCP. In this case, it replies to the SYN sent by the Transport Converter with the MP_CAPABLE option. Upon reception of this SYN+ACK, the Transport Converter confirms the establishment of the connection to the Client and indicates in the SYN+ACK packet sent to the Client that the Server supports Multipath TCP. With this information, the Client has discovered that the Server supports Multipath TCP natively. This will enable it to bypass the Transport Converter for the next Multipath TCP connection that it will initiate towards this Server.

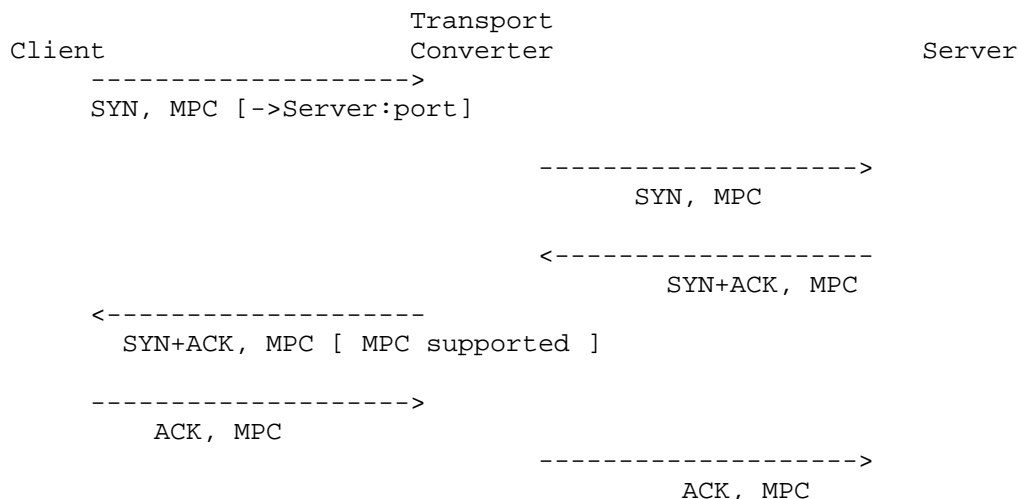


Figure 5: Establishment of a Multipath TCP connection through a converter

2.1. Differences with SOCKSv5

The description above is a simplified description of the Converter protocol. At a first glance, the proposed solution could seem similar to the SOCKS v5 protocol [RFC1928]. This protocol is used to proxy TCP connections. The Client creates a connection to a SOCKS proxy, exchanges authentication information and indicates the destination address and port of the final server. At this point, the

SOCKS proxy creates a connection towards the final server and relays all data between the two proxied connections. The operation of SOCKS v5 is illustrated in Figure 6.

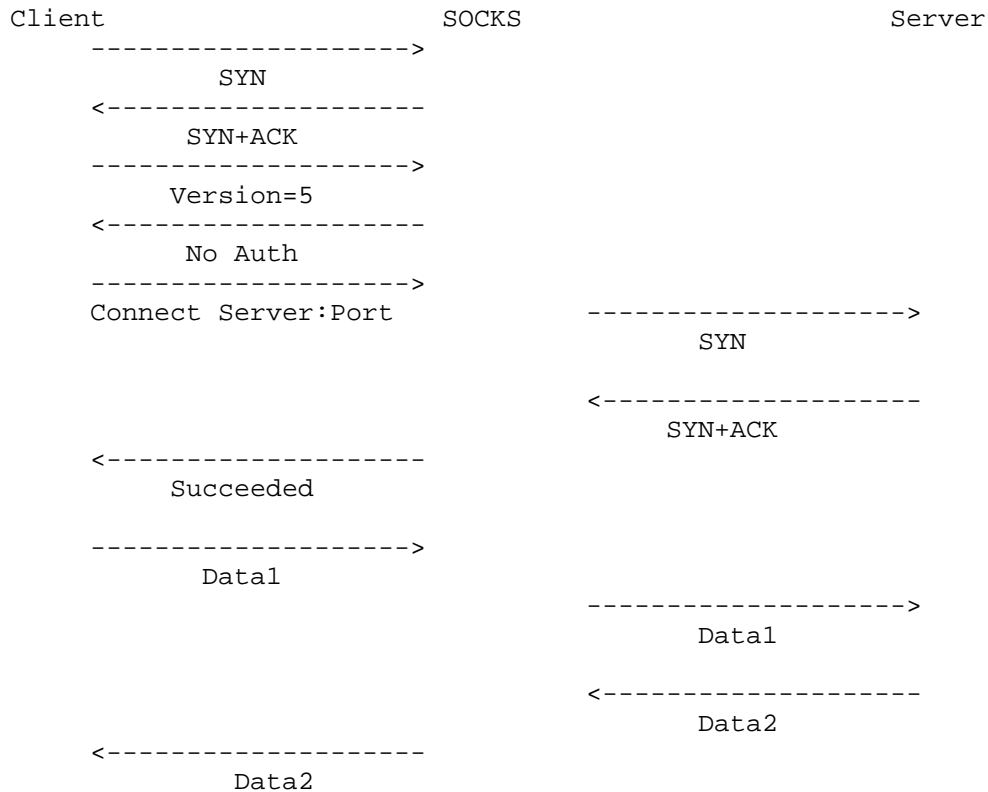


Figure 6: Establishment of a TCP connection through a SOCKS proxy without authentication

The converter protocol proposed in this document also relays data between an upstream and a downstream connection, but there are important differences with SOCKS v5.

A first difference is that the converter protocol leverages the TFO option [RFC7413] to place all its control information inside the SYN and SYN+ACK packets. This reduces the connection establishment delay compared to SOCKS that requires two or more round-trip-times before the establishment of the downstream connection towards the final destination. A recently proposed extension to SOCKS also leverages the TFO option [I-D.olteanu-intarea-socks-6].

A second difference is that the converter protocol takes the TCP extensions explicitly into account. With the converter protocol, the Client can learn whether a given TCP extension is supported by the destination Server. This enables the Client to bypass the Transport Converter when the destination supports the required TCP extension. Neither SOCKSv5 [RFC1928] nor the proposed SOCKS v6 [I-D.olteanu-intarea-socks-6] provide such feature.

A third difference is that a Transport Converter will only accept the connection initiated by the Client provided that the downstream connection is accepted by the Server. If the Server refuses the connection establishment attempt from the Transport Converter, then the upstream connection from the Client is rejected as well. This feature is important for applications that check the availability of a Server or use the time to connect as a hint on the selection of a Server [RFC6555]. This is illustrated in Figure 7.

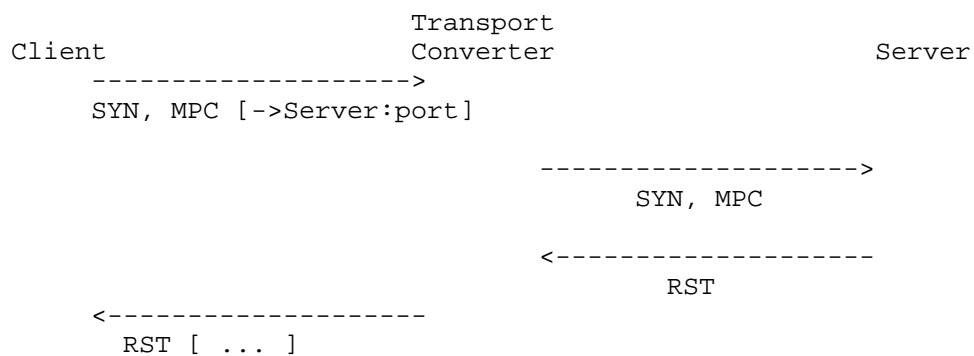


Figure 7: Establishment of a Multipath TCP connection through a converter

These three differences between SOCKS and the Converter protocol imply that a Transport Converter cannot be implemented as a regular user-space application like a SOCKS proxy. A Transport Converter needs to interact with the underlying TCP implementation more closely than the regular socket APIs used by the SOCKS proxy.

3. The Converter protocol

We now describe in details the messages that are exchanged between a Client and a Transport Converter. The Converter protocol (CP) leverages the TCP Fast Open extension defined in [RFC7413].

The Converter Protocol (CP) uses a 32 bits long fixed header that is sent by both the Client and the Transport Converter. This header indicates both the version of the protocol used and the length of the CP messages.

3.1. The Fixed Header

When a Client initiates a connection to a Transport Converter using the Converter Protocol, it **MUST** send the fixed-sized header shown in Figure 8 as the first four bytes of the bytestream.

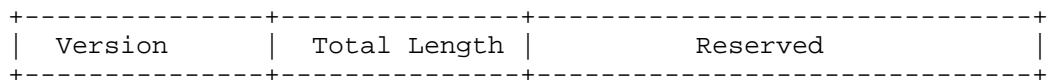


Figure 8: The fixed-sized header of the Converter Protocol

The Version is encoded as an 8 bits unsigned integer value. This document specifies version 1. The Total Length is the number of 32 bits word, including the header, of the bytestream that are consumed by the Converter Protocol messages. Since Total Length is also an 8 bits unsigned integer, those messages cannot consume more than 1020 bytes of data. This limits the number of bytes that a Transport Converter needs to process. A Total Length of zero is invalid and the connection **MUST** be reset upon reception of such a header. The Reserved field **MUST** be set to zero in this version of the protocol.

3.2. The TLV messages

The Converter protocol uses variable length messages that are encoded using a TLV format to simplify the parsing of the messages and leave room to extend the protocol in the future. A given TLV can only appear once on a Converter connection. If two or more copies of the same TLV are exchanged over a Converter connection, the associated TCP connections **MUST** be closed.

Five TLVs are defined in this document. They are listed in Table 1.

Type	Length	Description
1	1	Bootstrap TLV
10	Variable	Connect TLV
20	Variable	Extended TCP Header TLV
21	Variable	Supported TCP Options TLV
30	Variable	Error TLV

Table 1: The TLVs used by the Converter protocol

3.2.1. The Connect TLV

This TLV is used to request the Transport Converter to establish a connection towards the Server address and port included in the TLV. The Server Address is always encoded as an IPv6 address. IPv4 addresses are encoded using the IPv4-Mapped IPv6 Address format defined in [RFC4291]. The optional TCP Options field is used to specify how some TCP Options are advertised by the Transport Converter to the final destination. If this field is empty, then the Transport Converter uses the standard TCP options that correspond to its local policy.

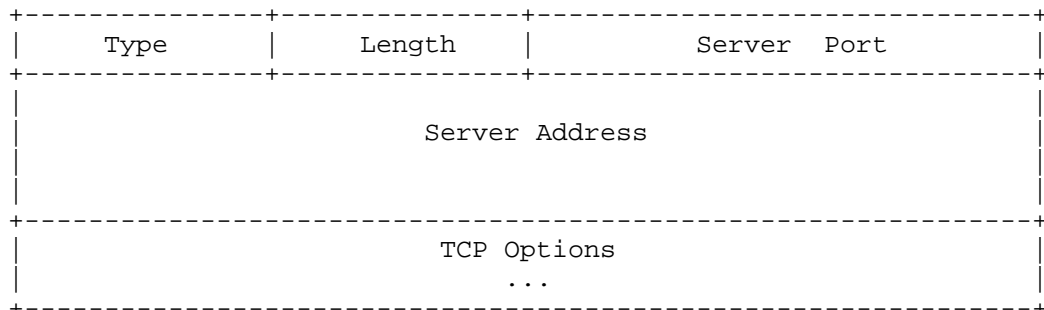


Figure 9: The Connect TLV

The TCP Options field is a variable length field that carries a list of TCP Option fields. Each TCP Option field is encoded as a block of 2+n bytes where the first byte is the TCP Option Type and the second byte is the length of the TCP Option as specified in [RFC0793]. The minimum value for the TCPOpt Length is 2. The TCP Options that do not include a length subfield, i.e. option types 0 (EOL) and 1 (NOP)

defined in [RFC0793] cannot be placed inside the TCP Options field of the Connect TLV. The optional Value field contains the variable-length part of the TCP option. A length of two indicates the absence of the Value field. The TCP Options field always ends on a 32 bits boundary after being padded with zeros.

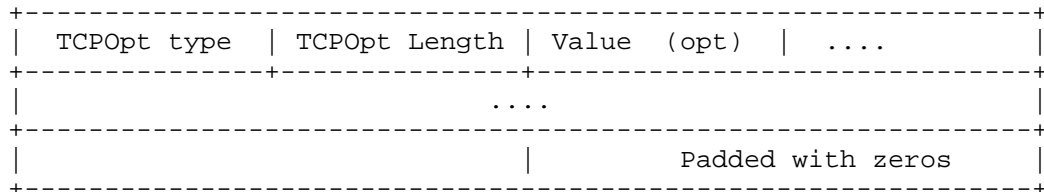


Figure 10: The TCP Options field

If a Transport Converter receives a Connect TLV with an empty TCP Options field, it shall place in the SYN that it sends towards the Server the TCP Options that it would have used according to its local policy.

If a Transport Converter receives a Connect TLV with a non-empty TCP Options field, it shall place in the SYN that it sends towards the destination Server the TCP Options that it would have used according to its local policies and the options that are listed in the TCP Options field. For the TCP Options that are listed without an optional value, it will generate its own value. For the TCP Options that are included in the TCP Options field with an optional value, it shall copy the entire option in the SYN sent to the Server. This feature is required to support TCP Fast Open as explained in Section 4.3.

3.2.2. Extended TCP Header TLV

The Extended TCP Header TLV is used by the Transport Converter to return to the Client the extended TCP header that was returned by the Server in the SYN+ACK packet. This TLV is only present if the Client has sent a Connect TLV to request the establishment of a connection.

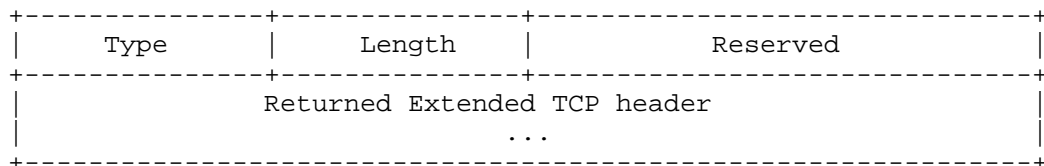


Figure 11: The Extended TCP Header TLV

The Returned Extended TCP header field is a copy of the extended header that was received in the SYN+ACK by the Transport Converter. The Reserved field is set to zero by the transmitter and ignored by the receiver.

3.2.3. Error TLV

This optional TLV can be used by the Transport Converter to provide information about some errors that occurred during the processing of a request to convert a connection. This TLV will appear after the Converter header in a RST segment returned by the Transport Converter if the error is fatal and prevented the establishment of the connection. If the error is not fatal and the connection could be established with the final destination, then the error TLV will be placed in the SYN/ACK packet.

+-----+ Type +-----+	+-----+ Length +-----+	+-----+ Error +-----+	+-----+ Value +-----+
--------------------------------	----------------------------------	---------------------------------	---------------------------------

Figure 12: The Error TLV

The following fatal errors are defined in this document:

- o Administratively prohibited (1). This error indicates that the Converter refused to create a connection towards the specific Server Address Destination or Port. The Value field is set to zero.
- o Connection reset by final destination (2). This Error indicates that the final destination responded with a RST packet. The Value field is set to zero.
- o Destination unreachable (3). This Error indicates that an ICMP destination unreachable, port unreachable or network unreachable was received by the Transport Converter. The Value field contains the Code field of the received ICMP message.
- o Invalid Converter message (4). This Error indicates that the Transport Converter received an unknown TLV. The Value field is set to the type of the unknown TLV. If several unknown TLVs were received, only one of them is reported in the error.
- o Resource Exceeded (5). This Error indicates that the Transport Converter does not have enough resources to perform the Client request.

The following non-fatal errors are defined in this document:

- o Unsupported TCP Option (128). A TCP Option that the Client requested to advertise to the final Server is not supported by the Transport Converter. The Value field is set to the type of the unsupported TCP Option. If several unsupported TCP Options were specified in the Connect TLV, only one of them is returned in the Value.

Table 2 summarises the different error messages.

Error	Description
1	Administratively prohibited
2	Connection reset by final destination
3	Destination unreachable
16	Invalid Converter message
32	Resource Exceeded
128	Error TLV

Table 2: The different error types

3.2.4. The Bootstrap TLV

The Bootstrap TLV is sent by a Client to request the TCP Extensions that are supported by a Transport Converter. It is typically sent on the first connection that a Client establishes with a Transport Converter to learn its capabilities. The Transport Converter replies with the Supported TCP Options TLV described in Section 3.2.5.

Type	Length	Zero
------	--------	------

Figure 13: The Bootstrap TLV

3.2.5. Supported TCP Options TLV

The Supported TCP Options TLV is used by a Converter to announce the TCP options that it supports. Each supported TCP Option is encoded with its TCP option Kind listed in the TCP Parameters registry maintained by IANA. TCP option Kinds 0, 1 and 2 defined in [RFC0793]

are supported by all TCP implementations and thus cannot appear in this list. The list of supported TCP Options is padded with 0 to end on a 32 bits boundary.

+	-----	+	-----	+	-----	+
	Type		Length		Reserved	
+	-----	+	-----	+	-----	+
	Kind #1		Kind #2		...	
+	-----	+	-----	+	-----	+
/						/
/						/
+	-----	+	-----	+	-----	+
					Kind #n	
					Zero	
+	-----	+	-----	+	-----	+

Figure 14: The Supported Options TLV

4. Examples

This section provides some examples of the utilisation of the Transport Converter. We consider the following network to illustrate the operation of the Converter protocol.

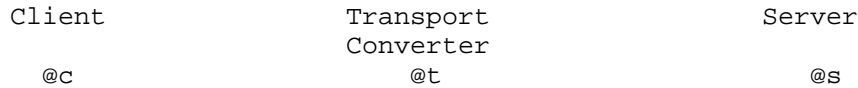


Figure 15: Simple scenario

4.1. Bootstrap

The Converter protocol defined in this document assumes the utilisation of TCP Fast Open between the Client and the Transport Converter. To be able to place data inside the SYN packet that it sends, the Client first needs to obtain a TFO cookie from the Transport Converter. This is achieved by simply establishing a TCP connection to the Transport Converter without requesting the establishment of a connection towards a Server.

To perform this bootstrap operation, the Client sends the following SYN packet :

- o source IP address: @c
- o destination IP address: @t
- o TCP Options : MSS, TFO (2 bytes), NOP, NOP
- o Payload : empty

The Converter replies with the following SYN+ACK packet:

- o source IP address: @t
- o destination IP address: @c
- o TCP Options : MSS, TFO (including @tcookie), NOP, NOP
- o Payload : empty

At this point, the Client has learned the TFO cookie (@tcookie) that needs to be used to interact with this Transport Converter. For the examples in this section, we assume TFO cookies that contain 4 bytes of information. Other cookie lengths are possible as well [RFC7413].

The Client sends the third acknowledgement to conclude the three-way handshake. It then sends in a Data packet the fixed header and the Bootstrap TLV to query the TCP options that are supported by the Converter. This message spans 8 bytes (4 for the fixed header and 4 for the Bootstrap TLV). The Converter replies with a fixed header and a TCP Options TLV that indicates the TCP extensions that it supports.

4.2. Multipath TCP

The MP_CAPABLE Option defined in [RFC6824] allows to negotiate the utilisation of Multipath TCP. Consider a Client that uses the Transport Converter to create a connection on port 123 with a Server that supports [RFC6824].

For this, the Client sends the following SYN packet :

- o source IP address: @c
- o destination IP address: @t
- o TCP Options : MSS, TFO (@t cookie), MP_CAPABLE(key@c)
- o Payload :
 - * Converter Header, Total Length=7 32 bits words
 - * Connect :
 - + Length=6
 - + Port=123
 - + Address: @s
 - + TCP Options:
 - TCPOpt type: 30 (Multipath TCP)
 - TCPOpt Length: 2 (no value)
 - Padding: zero (2 bytes)

Upon reception of this packet, the Transport Converter creates a SYN packet and sends it to the destination Server:

- o source IP address: @t
- o destination IP address: @s
- o TCP Options : MSS, MP_CAPABLE(key@ts)
- o Payload : empty

The Server replies with the following SYN+ACK:

- o source IP address: @s
- o destination IP address: @t
- o TCP Options : MSS, MP_CAPABLE(key@ts,key@s)
- o Payload : empty

The Transport Converter then confirms the establishment of the connection to the Client with the following SYN+ACK:

- o source IP address: @t
- o destination IP address: @c
- o TCP Options : MSS, MP_CAPABLE(key@c,key@tc)
- o Payload:
 - * Converter Header, Total Length=8
 - * TCP Extended Header TLV:
 - + Length=7
 - + Value: MSS, MP_CAPABLE(key@ts,key@s)

Upon reception of this packet, the Client has the confirmation that the Multipath TCP connection has been established through the Transport Converter. By parsing the TCP Extended Header TLV, it detects that Server @s supports Multipath TCP and will thus be able to bypass the Transport Converter for future connections towards this Server.

4.3. TCP Fast Open (TFO)

The TCP Fast Open (TFO) option is defined in [RFC7413]. In this section, we show how a Client can use TFO with a remote Server

through a Transport Converter. We consider two TCP connections to this Server. The Client has already received the cookie of the Transport Converter (@t cookie).

For the first connection, the Client sends the following SYN packet:

- o source IP address: @c
- o destination IP address: @t
- o TCP Options : MSS, TFO (@t cookie)
- o Payload :
 - * Converter Header, Total Length=8
 - * Connect:
 - + Length=7
 - + Port=123
 - + Address: @s
 - + TCP Options:
 - TCPOpt type: 34 (TFO)
 - TCPOpt Length: 2 (no value)
 - Padding: zero (2 bytes)

The TFO option of the SYN packet contains the cookie chosen by the Transport Converter. The Transport Converter then issues the following SYN packet towards the Server:

- o source IP address: @t
- o destination IP address: @s
- o TCP Options : MSS, TFO (empty), NOP, NOP
- o Payload : empty

The Server replies with its own TFO cookie (@s cookie) in the SYN+ACK packet:

- o source IP address: @s
- o destination IP address: @t
- o TCP Options : MSS, TFO (@s cookie)
- o Payload : empty

The Converter confirms the establishment of the TCP connection to the Client by sending the following SYN+ACK packet:

- o source IP address: @t
- o destination IP address: @c
- o TCP Options : MSS, NOP, NOP
- o Payload :
 - * Converter Header, Total Length=8
 - * TCP Extended Header TLV:
 - + Length=7
 - + Value: MSS, TFO (@s cookie)
 - * some data

The Client can extract the Server cookie from the TCP Extended Header TLV and initiate future connections to this Server as follows (assuming that it prefers to establish it via the Transport Converter instead of contacting directly the final destination).

- o source IP address: @c
- o destination IP address: @t
- o TCP Options : MSS, TFO (@t cookie)
- o Payload :
 - * Converter Header, Total Length=4
 - * Connect:
 - + Length=8

- + Port=123
- + Address: @s
- + TCP Options:
 - TCPOpt type: 34 (TFO)
 - TCPOpt Length: 6 (value is @s cookie)
 - Padding: zero (2 bytes)

The Transport Converter then initiates the connection towards the final destination by sending the following SYN packet:

- o source IP address: @t
- o destination IP address: @s
- o TCP Options : MSS, TFO (@s cookie), NOP, NOP
- o Payload : some data

The Server verifies the TFO option and accepts the data in the SYN. It replies with the following SYN+ACK packet:

- o source IP address: @s
- o destination IP address: @t
- o TCP Options : MSS, NOP, NOP
- o Payload : more data

The Server confirms the establishment of the TCP connection to the Client by sending the following SYN+ACK packet:

- o source IP address: @t
- o destination IP address: @c
- o TCP Options : MSS, NOP, NOP
- o Payload :

* Converter Header, Total Length=3

- * Report:
 - + Length=2
 - + Value: MSS, NOP, NOP
- * more data

The Client has thus been able to use TFO with a remote Server through the Transport Converter.

5. Interactions with middleboxes

The Converter protocol was designed to be used in networks that do not contain middleboxes that interfere with TCP. We describe in this section how a Client can detect middlebox interference and stop using the Transport Converter affected by this interference.

Internet measurements [IMC11] have shown that middleboxes can affect the deployment of TCP extensions. In this section, we only discuss the middleboxes that modify SYN and SYN+ACK packets since the Converter protocol places its messages in such packets.

Let us first consider a middlebox that removes the TFO Option from the SYN packet. This interference will be detected by the Client during the bootstrap procedure described in section Section 4.1. A Client should not use a Transport Converter that does not reply with the TFO option during the Bootstrap.

Consider a middlebox that removes the SYN payload after the bootstrap procedure. The Client can detect this problem by looking at the acknowledgement number field of the SYN+ACK returned by the Transport Converter. The Client should stop to use this Transport Converter given the middlebox interference.

As explained in [RFC7413], some carrier-grade NATs can affect the operation of TFO if they assign different IP addresses to the same endhost. Such carrier-grade NATs could affect the operation of the TFO Option used by the Converter protocol. See also the discussion in section 7.1 of [RFC7413].

6. Security Considerations

Given its function and its location in the network, a Transport Converter has access to the payload of all the packets that it processes. As such, it must be protected as a core IP router.

The Converter protocol is intended to be used in managed networks where endhosts can be identified by their IP address. Thanks to the Bootstrap procedure described in section Section 4.1, the Transport Converter can verify that the Client correctly receives packets sent by the Converter. Stronger authentication schemes should be defined to use the Converter protocol in more open network environments.

Upon reception of a SYN that contains a valid TFO Cookie and a Connect TLV, the Transport Converter attempts to establish a TCP connection to a remote Server. There is a risk of denial of service attack if a Client requests too many connections in a short period of time. Implementations should limit the number of pending connections from a given Client.

Another possible risk are the amplification attacks since a Transport Converter sends a SYN towards a remote Server upon reception of a SYN from a Client. This could lead to amplification attacks if the SYN sent by the Transport Converter were larger than the SYN received from the Client or if the Transport Converter retransmits the SYN. To mitigate such attack,s the Transport Converter should first limit the number of pending requested for a given Client. It should also avoid sending to remote Servers SYNs that are significantly longer than the SYN received from the Client. In practice, Transport Converters should not advertise to a Server TCP Options that were not specified by the Client in the received SYN. Finally, the Transport Converter should only retransmit a SYN to a Server after having received a retransmitted SYN from the corresponding Client.

7. IANA Considerations

This document requests the allocation of a reserved service name and port number for the converter protocol.

This document specifies version 1 of the Converter protocol. Five types of Converter messages are defined:

- o 1: Bootstrap TLV
- o 10: Connect TLV
- o 20: Extended TCP Header TLV
- o 21: Supported TCP Options TLV
- o 30: Error TLV

Furthermore, it also defines 6 types of errors.

- o 1: Administratively prohibited
- o 2: Connection reset by final destination
- o 3: Destination unreachable
- o 16: Invalid Converter message
- o 32: Resource Exceeded -128: Error TLV

8. Conclusion

We have proposed the utilisation of Transport Converters to aid the deployment of TCP extensions such as Multipath TCP. Compared with deployed solutions such as SOCKS proxies, the Transport Converters provide several benefits. First, they do not increase the connection establishment time. Second, they are compatible and benefit from the TCP Fast Open extension. Third, clients benefit from the Transport Converter when the Server does not support the required extension. Furthermore, they can easily detect when the Server supports the required extension and thus bypass the Transport Converter to contact those Servers.

9. Acknowledgements

This document builds upon earlier documents that proposed various forms of Multipath TCP proxies [I-D.boucadair-mptcp-plain-mode], [I-D.peirens-mptcp-transparent] and [HotMiddlebox13b].

We would like to thank Bart Peirens, Raphael Bauduin and Anand Nandugudi for their help in preparing this draft.

Although they could disagree with the contents of the document, we would like to thank Joe Touch and Juliusz Chroboczek whose comments on the MPTCP mailing list have forced us to reconsider the design of the solution several times.

10. References

10.1. 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>>.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <<http://www.rfc-editor.org/info/rfc4291>>.
- [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>>.
- [RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <<http://www.rfc-editor.org/info/rfc7413>>.

10.2. Informative References

- [Fukuda2011] Fukuda, K., "An Analysis of Longitudinal TCP Passive Measurements (Short Paper)", Traffic Monitoring and Analysis. TMA 2011. Lecture Notes in Computer Science, vol 6613. , 2011.
- [HotMiddlebox13b] Detal, G., Paasch, C., and O. Bonaventure, "Multipath in the Middle(Box)", HotMiddlebox'13 , December 2013, <<http://inl.info.ucl.ac.be/publications/multipath-middlebox>>.
- [I-D.boucadair-mptcp-plain-mode] Boucadair, M., Jacquenet, C., Bonaventure, O., Behaghel, D., stefano.secci@lip6.fr, s., Henderickx, W., Skog, R., Vinapamula, S., Seo, S., Cloetens, W., Meyer, U., Contreras, L., and B. Peirens, "Extensions for Network-Assisted MPTCP Deployment Models", draft-boucadair-mptcp-plain-mode-10 (work in progress), March 2017.
- [I-D.ietf-tcpinc-tcpcrypt] Bittau, A., Giffin, D., Handley, M., Mazieres, D., Slack, Q., and E. Smith, "Cryptographic protection of TCP Streams (tcpcrypt)", draft-ietf-tcpinc-tcpcrypt-06 (work in progress), March 2017.

- [I-D.olteanu-intarea-socks-6]
Olteanu, V. and D. Niculescu, "SOCKS Protocol Version 6",
draft-olteanu-intarea-socks-6-00 (work in progress),
June 2017.
- [I-D.peirens-mptcp-transparent]
Peirens, B., Detal, G., Barre, S., and O. Bonaventure,
"Link bonding with transparent Multipath TCP",
draft-peirens-mptcp-transparent-00 (work in progress),
July 2016.
- [IETFJ16] Bonaventure, O. and S. Seo, "Multipath TCP Deployment",
IETF Journal, Fall 2016 , n.d..
- [IMC11] Honda, K., Nishida, Y., Raiciu, C., Greenhalgh, A.,
Handley, M., and T. Hideyuki, "Is it still possible to
extend TCP ?", Proceedings of the 2011 ACM SIGCOMM
conference on Internet measurement conference , 2011.
- [RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and
L. Jones, "SOCKS Protocol Version 5", RFC 1928,
DOI 10.17487/RFC1928, March 1996,
<<http://www.rfc-editor.org/info/rfc1928>>.
- [RFC3135] Border, J., Kojo, M., Griner, J., Montenegro, G., and Z.
Shelby, "Performance Enhancing Proxies Intended to
Mitigate Link-Related Degradations", RFC 3135,
DOI 10.17487/RFC3135, June 2001,
<<http://www.rfc-editor.org/info/rfc3135>>.
- [RFC6555] Wing, D. and A. Yourtchenko, "Happy Eyeballs: Success with
Dual-Stack Hosts", RFC 6555, DOI 10.17487/RFC6555,
April 2012, <<http://www.rfc-editor.org/info/rfc6555>>.
- [RFC7414] Duke, M., Braden, R., Eddy, W., Blanton, E., and A.
Zimmermann, "A Roadmap for Transmission Control Protocol
(TCP) Specification Documents", RFC 7414, DOI 10.17487/
RFC7414, February 2015,
<<http://www.rfc-editor.org/info/rfc7414>>.

Author's Address

Olivier Bonaventure
Tessares

Email: Olivier.Bonaventure@tessares.net

Internet Area Working Group
Internet-Draft
Intended status: Experimental
Expires: December 30, 2017

V. Olteanu
D. Niculescu
University Politehnica of Bucharest
June 28, 2017

SOCKS Protocol Version 6
draft-olteanu-intarea-socks-6-00

Abstract

The SOCKS protocol is used primarily to proxy TCP connections to arbitrary destinations via the use of a proxy server. Under the latest version of the protocol (version 5), it takes 2 RTTs (or 3, if authentication is used) before data can flow between the client and the server.

This memo proposes SOCKS version 6, which reduces the number of RTTs used, takes full advantage of TCP Fast Open, and adds support for 0-RTT authentication.

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 30, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	2
2. Requirements language	3
3. Mode of operation	3
4. Connection Requests	5
5. SOCKS Options	7
5.1. Authentication options	7
6. Authentication Replies	8
7. Operation Replies	9
7.1. Handling CONNECT	10
7.2. Handling BIND	10
7.3. Handling UDP ASSOCIATE	11
8. Security Considerations	11
9. IANA Considerations	11
10. Acknowledgements	11
11. References	11
11.1. Normative References	11
11.2. Informative References	12
Authors' Addresses	12

1. Introduction

Versions 4 and 5 [RFC1928] of the SOCKS protocol were developed two decades ago and are in widespread use for circuit level gateways or as circumvention tools, and enjoy wide support and usage from various software, such as web browsers, SSH clients, and proxifiers. However, their design needs an update in order to take advantage of the new features of transport protocols, such as TCP Fast Open [RFC7413], or to better assist newer transport protocols, such as MPTCP [RFC6824].

One of the main issues faced by SOCKS version 5 is that, when taking into account the TCP handshake, method negotiation, authentication, connection request and grant, it may take up to 5 RTTs for a data exchange to take place at the application layer. This is especially costly in networks with a large delay at the access layer, such as 3G, 4G, or satellite.

The desire to reduce the number of RTTs manifests itself in the design of newer security protocols. TLS version 1.3 [I-D.ietf-tls-tls13] defines a zero round trip (0-RTT) handshake mode for connections if the client and server had previously communicated.

TCP Fast Open [RFC7413] is a TCP option that allows TCP to send data in the SYN and receive a response in the first ACK, and aims at obtaining a data response in one RTT. The SOCKS protocol needs to concern itself with at least two TFO deployment scenarios: First, when TFO is available end-to-end (at the client, at the proxy, and at the server); second, when TFO is active between the client and the proxy, but not at the server.

This document describes the SOCKS protocol version 6. The key improvements over SOCKS version 5 are:

- o The client sends as much information upfront as possible, and does not wait for the authentication process to conclude before requesting the creation of a socket.
- o The connection request also mimics the semantics of TCP Fast Open [RFC7413]. As part of the connection request, the client can supply the payload for the initial SYN that is sent out to the server.
- o The protocol can be extended via options without breaking backward-compatibility.
- o The protocol can leverage the aforementioned options to support 0-RTT authentication schemes.

2. Requirements language

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

3. Mode of operation

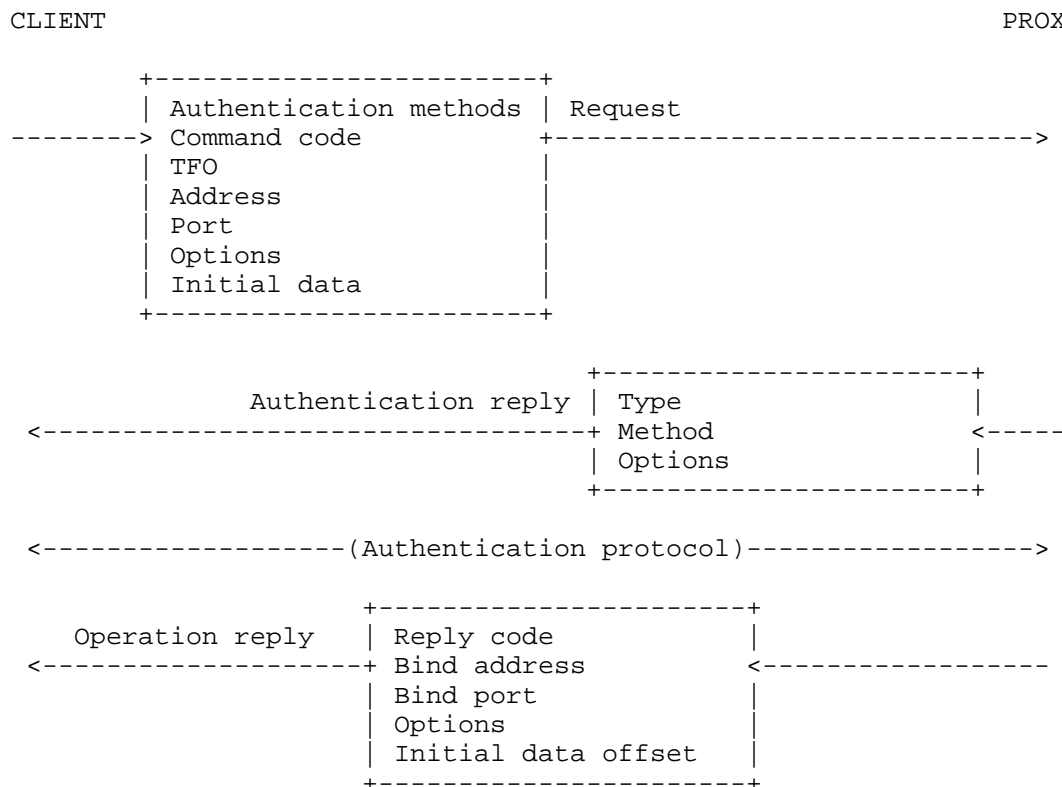


Figure 1: The SOCKS version 6 protocol message exchange

When a TCP-based client wishes to establish a connection to a server, it must open a TCP connection to the appropriate SOCKS port on the SOCKS proxy. The client then enters a negotiation phase, by sending the request in figure Figure 1, that contains, in addition to fields present in SOCKS 5 [RFC1928], fields that facilitate low RTT usage and faster authentication negotiation.

Next, the server sends an authentication reply. If the request did not contain the necessary authentication information, the proxy indicates an authentication method that must proceed. This may trigger a longer authentication sequence that could include tokens for ulterior faster authentications. The part labeled "Authentication protocol" is specific to the authentication method employed and is not expected to be employed for every connection between a client and its proxy server. The authentication protocol typically takes up 1 RTT or more.

If the authentication is successful, an operation reply is generated by the proxy. It indicates whether the proxy was successful in creating the requested socket or not.

In the fast case, when authentication is properly set up, the proxy attempts to create the socket immediately after the receipt of the request, thus achieving an operational connection in one RTT (provided TFO functionality is available at the client, proxy, and server).

4. Connection Requests

The client starts by sending a request to the proxy.

Version		Number of Methods		Methods
Major	Minor			
1	1	1	Variable	

Command Code	TFO	Address Type	Address	Port
1	1	1	Variable	2

Number of Options	Options	Initial Data Size	Initial Data
1	Variable	2	Variable

Figure 2: SOCKS 6 Request

- o Version: The major byte MUST be set to 0x06, and the minor byte MUST be set to 0x00.
- o Number of Methods: The number of supported authentication methods that the client wishes to advertise.
- o Methods: One byte per advertised method. Method numbers are assigned by IANA.
- o Command Code:
 - * 0x00 AUTH: authenticate the client and do nothing.

- * 0x01 CONNECT: requests the establishment of a TCP connection.
- * 0x02 BIND: requests the establishment of a TCP port binding.
- * 0x03 UDP ASSOCIATE: requests a UDP port association.
- o TFO:
 - * 0x00 indicates that the proxy MUST NOT attempt to use TFO in case of a CONNECT command, or accept TFO in case of a BIND command. In case of an AUTH or UDP ASSOCIATE command, this field MUST be set to 0x00.
 - * 0x01 indicates that the proxy SHOULD attempt to use TFO in case of a CONNECT command, or accept TFO in case of a BIND command.
- o Address Type:
 - * 0x01: IPv4
 - * 0x03: Domain Name
 - * 0x04: IPv6
- o Address: this field's format depends on the address type:
 - * IPv4: a 4-byte IPv4 address
 - * Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated.
 - * IPv6: a 16-byte IPv6 address
- o Port: the port in network byte order.
- o Number of Options: the number of SOCKS options that appear in the Options field.
- o Options: see section Section 5.
- o Initial Data Size: A two-byte number in network byte order. In case of AUTH, BIND or UDP ASSOCIATE, this field MUST be set to 0. In case of CONNECT, this is the number of bytes of initial data that are supplied in the following field.
- o Initial Data: The first octets of the data stream.

Clients MUST support the "No authentication required" method. Clients MAY omit advertising the "No authentication required" option.

Clients SHOULD NOT issue AUTH commands unless they advertise authentication methods with support for 0-RTT authentication.

The server MAY truncate the initial data to an arbitrary size and disregard the rest.

5. SOCKS Options

SOCKS options have the following format:

+-----+-----+			
Kind	Length	Option Data	
+-----+-----+			
1	1	Variable	
+-----+-----+			

Figure 3: SOCKS 6 Option

- o Kind: MUST be allocated by IANA. (See section Section 9.)
- o Length: The length of the option data.
- o Option Data: the contents are specific to each option kind.

5.1. Authentication options

Authentication options have the following format:

Kind	Length	Method	Authentication Data
1	1	1	Variable

Figure 4: Authentication Option

- o Kind: MUST be allocated by IANA. (See section Section 9.)
- o Length: the length of the option data.
- o Method: the number of the authentication method. These numbers are assigned by IANA.

- o Authentication Data: the contents are specific to each method.

All proxy implementations MUST support authentication method options. Clients MAY omit advertising authentication methods for which they have included at least an authentication option.

6. Authentication Replies

Upon receipt of a request, the proxy sends an Authentication Reply:

Version		Type	Method	Number of	Options
Major	Minor			Options	
1	1	1	1	1	Variable

Figure 5: SOCKS 6 Authentication Reply

- o Version: The major byte MUST be set to 0x06, and the minor byte MUST be set to 0x00.
- o Type:
 - * 0x00: authentication successful.
 - * 0x01: further authentication needed.
- o Method: The chosen authentication method.
- o Number of Options: the number of SOCKS options that appear in the Options field.
- o Options: see section Section 5.

Multihomed clients SHOULD cache the chosen method on a per-interface basis and SHOULD NOT include authentication options related to any other methods in further requests originating from the same interface.

If the server signals that further authentication is needed and selects "No Acceptable Methods", the client MUST close the connection.

The client and proxy begin a method-specific negotiation. During such negotiations, the proxy MAY supply information that allows the client to authenticate a future request using an authentication

option. Descriptions of such negotiations are beyond the scope of this memo.

If the cliend issued an AUTH command, the client MUST close the connection after the negotiation is complete.

7. Operation Replies

After the authentication negotiations are complete, the server sends an Operation Reply:

Version		Reply	Address	Bind	Bind
Major	Minor	Code	Type	Address	Port
1	1	1	1	Variable	2

Number of	Options	Initial Data
Options		Offset
1	Variable	2

Figure 6: SOCKS 6 Operation Reply

- o Version: The major byte MUST be set to 0x06, and the minor byte MUST be set to 0x00.
- o Reply Code:
 - * 0x00: Succes
 - * 0x01: General SOCKS server failure
 - * 0x02: Connection not allowed by ruleset
 - * 0x03: Network unreachable
 - * 0x04: Host unreachable
 - * 0x05: Connection refused
 - * 0x06: TTL expired
 - * 0x07: Command not supported

- * 0x08: Address type not supported
- o Address Type:
 - * 0x01: IPv4
 - * 0x03: Domain Name
 - * 0x04: IPv6
- o Bind Address: the proxy bound address in the following format:
 - * IPv4: a 4-byte IPv4 address
 - * Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated.
 - * IPv6: a 16-byte IPv6 address
- o Bind Port: the proxy bound port in network byte order.
- o Number of Options: the number of SOCKS options that appear in the Options field.
- o Options: see section Section 5
- o Initial Data Offset: A two-byte number in network byte order. In case of BIND or UDP ASSOCIATE, this field MUST be set to 0. In case of CONNECT, it represents the offset in the plain data stream from which the client is expected to continue sending data.

If the proxy returns a reply code other than "Success", the client MUST close the connection.

7.1. Handling CONNECT

In case the client has issued a CONNECT request, data can now pass. The client MUST resume the data stream at the offset indicated by the Initial Data Offset field.

7.2. Handling BIND

In case the client has issued a BIND request, it must wait for a second Operation reply from the proxy, which signifies that a host has connected to the bound port. The Bind Address and Bind Port fields contain the address and port of the connecting host. Afterwards, application data may pass.

7.3. Handling UDP ASSOCIATE

The relay of UDP packets is handled exactly as in SOCKS 5 [RFC1928].

8. Security Considerations

Given the format of the request message, a malicious client could craft a request that is in excess of 100 KB and proxies could be prone to DDoS attacks.

To mitigate such attacks, proxy implementations SHOULD be able to incrementally parse the requests. Proxies MAY close the connection to the client if:

- o the request is not fully received after a certain timeout, or
- o the number of options exceeds an imposed hard cap, or
- o the total size of the options exceeds an imposed hard cap, or
- o the size of the initial data exceeds a hard cap.

Further, the server MAY choose not to buffer any initial data beyond what would fit in a TFO SYN's payload.

9. IANA Considerations

This document requests that IANA allocate option codes for SOCKS 6 options. Further, this document requests an option code for authentication options.

10. Acknowledgements

The protocol described in this draft builds upon and is a direct continuation of SOCKS 5 [RFC1928].

11. References

11.1. 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>>.

11.2. Informative References

- [I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-20 (work in progress), April 2017.
- [RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 1928, DOI 10.17487/RFC1928, March 1996, <<http://www.rfc-editor.org/info/rfc1928>>.
- [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>>.
- [RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <<http://www.rfc-editor.org/info/rfc7413>>.

Authors' Addresses

Vladimir Olteanu
University Politehnica of Bucharest

Email: vladimir.olteanu@cs.pub.ro

Dragos Niculescu
University Politehnica of Bucharest

Email: dragos.niculescu@cs.pub.ro