Stream: Internet Engineering Task Force (IETF)
RFC: 8803
Category: Experimental
Published: July 2020
ISSN: 2070-1721
Authors:
O. Bonaventure, Ed. *Tessares*
M. Boucadair, Ed. *Orange*
S. Gundavelli *Cisco*
S. Seo *Korea Telecom*
B. Hesmans *Tessares*

RFC 8803
0-RTT TCP Convert Protocol

Abstract

This document specifies an application proxy, called Transport Converter, to assist the deployment of TCP extensions such as Multipath TCP. A Transport Converter may provide conversion service for one or more TCP extensions. The conversion service is provided by means of the 0-RTT TCP Convert Protocol (Convert).

This protocol provides 0-RTT (Zero Round-Trip Time) conversion service since no extra delay is induced by the protocol compared to connections that are not proxied. Also, the Convert Protocol does not require any encapsulation (no tunnels whatsoever).

This specification assumes an explicit model, where the Transport Converter is explicitly configured on hosts. As a sample applicability use case, this document specifies how the Convert Protocol applies for Multipath TCP.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8803.

Copyright Notice

Table of Contents

1. Introduction

1.1. The Problem

Transport protocols like TCP evolve regularly [RFC7414]. TCP has been improved in different ways. Some improvements such as changing the initial window size [RFC6928] or modifying the congestion control scheme can be applied independently on Clients and Servers. Other improvements such as Selective Acknowledgments [RFC2018] or large windows [RFC7323] require a new TCP option or changing the semantics of some fields in the TCP header. These modifications must be deployed on both Clients and Servers to be actually used on the Internet. Experience with the latter class of TCP extensions reveals that their deployment can require many years. Fukuda reports in [Fukuda2011] results of a decade of measurements showing the deployment of Selective Acknowledgments, Window Scale, and TCP Timestamps. [ANRW17] describes measurements showing that TCP Fast Open (TFO) [RFC7413] is still not widely deployed.

There are some situations where the transport stack used on Clients (or Servers) can be upgraded at a faster pace than the transport stack running on Servers (or Clients). In those situations, Clients (or Servers) would typically want to benefit from the features of an improved transport protocol even if the Servers (or Clients) have not yet been upgraded. Some assistance from the network to make use of these features is valuable. For example, Performance Enhancing Proxies [RFC3135] and other service functions have been deployed as solutions to improve TCP performance over links with specific characteristics.

Recent examples of TCP extensions include Multipath TCP (MPTCP) [RFC8684] or tcpcrypt [RFC8548]. Those extensions provide features that are interesting for Clients such as wireless devices. With Multipath TCP, those devices could seamlessly use Wireless Local Area Network (WLAN) and cellular networks for bonding purposes, faster hand-overs, or better resiliency. Unfortunately, deploying those extensions on both a wide range of Clients and Servers remains difficult.

More recently, 5G bonding experimentation has been conducted into global range of the incumbent 4G (LTE) connectivity using newly devised Clients and a Multipath TCP proxy. Even if the 5G and 4G bonding (that relies upon Multipath TCP) increases the bandwidth, it is also crucial to minimize latency entirely between end hosts regardless of whether intermediate nodes are inside or outside of the mobile core. In order to handle Ultra Reliable Low Latency Communication (URLLC) for the next-generation mobile network, Multipath TCP and its proxy mechanism such as the one used to provide Access Traffic Steering, Switching, and Splitting (ATSSS) must be optimized to reduce latency [TS23501].

1.2. Network-Assisted Connections: The Rationale

This document specifies an application proxy called Transport Converter. A Transport Converter is a function that is installed by a network operator to aid the deployment of TCP extensions and to provide the benefits of such extensions to Clients in particular. A Transport Converter may provide conversion service for one or more TCP extensions. Which TCP extensions are eligible for the conversion service is deployment specific. The conversion service is provided by means of the 0-RTT TCP Convert Protocol (Convert), which is an application-layer protocol that uses a specific TCP port number on the Converter.

The Convert Protocol provides Zero Round-Trip Time (0-RTT) conversion service since no extra delay is induced by the protocol compared to connections that are not proxied. Particularly, the Convert Protocol does not require extra signaling setup delays before making use of the conversion service. The Convert Protocol does not require any encapsulation (no tunnels, whatsoever).

The Transport Converter adheres to the main steps drawn in Section 3 of [RFC1919]. In particular, a Transport Converter achieves the following:

- Listening for Client sessions;
- Receiving the address of the Server from the Client;
- Setting up a session to the Server;
- Relaying control messages and data between the Client and the Server;
- Performing access controls according to local policies.

The main advantage of network-assisted conversion services is that they enable new TCP extensions to be used on a subset of the path between endpoints, which encourages the deployment of these extensions. Furthermore, the Transport Converter allows the Client and the Server to directly negotiate TCP extensions for the sake of native support along the full path.

The Convert Protocol is a generic mechanism to provide 0-RTT conversion service. As a sample applicability use case, this document specifies how the Convert Protocol applies for Multipath TCP. It is out of scope of this document to provide a comprehensive list of all potential conversion services. Applicability documents may be defined in the future.

This document does not assume that all the traffic is eligible for the network-assisted conversion service. Only a subset of the traffic will be forwarded to a Transport Converter according to a set of policies. These policies, and how they are communicated to endpoints, are out of scope. Furthermore, it is possible to bypass the Transport Converter to connect directly to the Servers that already support the required TCP extension(s).

This document assumes an explicit model in which a Client is configured with one or a list of Transport Converters (statically or through protocols such as [DHC-CONVERTER]). Configuration means are outside the scope of this document.

The use of a Transport Converter means that there is no end-to-end transport connection between the Client and Server. This could potentially create problems in some scenarios such as those discussed in Section 4 of [RFC3135]. Some of these problems may not be applicable. For example, a Transport Converter can inform a Client by means of Network Failure (65) or Destination Unreachable (97) error messages (Section 6.2.8) that it encounters a failure problem; the Client can react accordingly. An endpoint, or its network administrator, can assess the benefit provided by the Transport Converter service versus the risk. This is one reason why the Transport Converter functionality has to be explicitly requested by an endpoint.

This document is organized as follows:

Section 3 provides a brief overview of the differences between the well-known SOCKS protocol and the 0-RTT TCP Convert Protocol.

Section 4 provides a brief explanation of the operation of Transport Converters.

Section 5 includes a set of sample examples to illustrate the overall behavior.

Section 6 describes the Convert Protocol.

Section 7 discusses how Transport Converters can be used to support different TCP extensions.

Section 8 then discusses the interactions with middleboxes.

Section 9 focuses on security considerations.

Appendix A describes how a TCP stack would need to support the protocol described in this document.

1.3. Applicability Scope

The 0-RTT TCP Convert Protocol specified in this document **MUST** be used in a single administrative domain deployment model. That is, the entity offering the connectivity service to a Client is also the entity that owns and operates the Transport Converter, with no transit over a third-party network.

Future deployment of Transport Converters by third parties **MUST** adhere to the mutual authentication requirements in Section 9.2 to prevent illegitimate traffic interception (Section 9.4) in particular.

2. Conventions and Definitions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Differences with SOCKSv5

Several IETF protocols provide proxy services, the closest to the 0-RTT TCP Convert Protocol being the SOCKSv5 protocol [RFC1928]. This protocol is already used to deploy Multipath TCP in some cellular networks (Section 2.2 of [RFC8041]).

A SOCKS Client creates a connection to a SOCKS Proxy, exchanges authentication information, and indicates the IP address and port number of the target Server. At this point, the SOCKS Proxy creates a connection towards the target Server and relays all data between the two proxied connections. The operation of an implementation based on SOCKSv5 (without authentication) is illustrated in Figure 1.

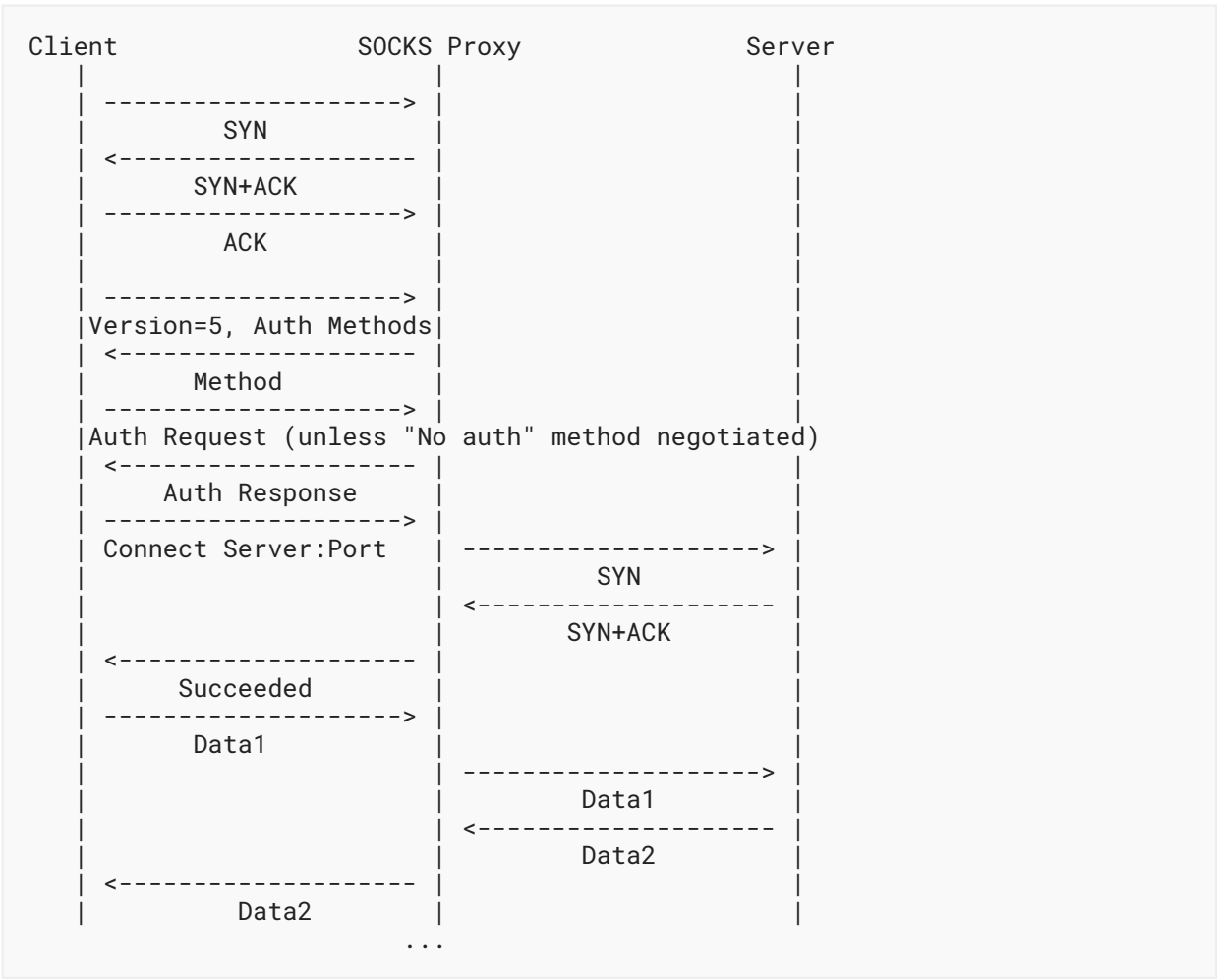

Figure 1: Establishment of a TCP Connection through a SOCKS Proxy without Authentication

When SOCKS is used, an "end-to-end" connection between a Client and a Server becomes a sequence of two TCP connections that are glued together on the SOCKS Proxy. The SOCKS Client and Server exchange control information at the beginning of the bytestream on the Client-Proxy connection. The SOCKS Proxy then creates the connection with the target Server and then glues the two connections together so that all bytes sent by the application (Client) to the SOCKS Proxy are relayed to the Server and vice versa.

The Convert Protocol is also used on TCP proxies that relay data between an upstream and a downstream connection, but there are important differences with SOCKSv5. A first difference is that the 0-RTT TCP Convert Protocol exchanges all the control information during the initial RTT. This reduces the connection establishment delay compared to SOCKS, which requires two or more round-trip times before the establishment of the downstream connection towards the final destination. In today's Internet, latency is an important metric, and various protocols have been tuned to reduce their latency [LOW-LATENCY]. A recently proposed extension to SOCKS leverages the TCP Fast Open (TFO) option [INTAREA-SOCKS] to reduce this delay.

A second difference is that the Convert Protocol explicitly takes the TCP extensions into account. By using the Convert 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 Server supports the required TCP extension(s). Neither SOCKSv5 [RFC1928] nor the proposed SOCKSv6 [INTAREA-SOCKS] provide such a feature.

A third difference is that a Transport Converter will only confirm the establishment of the connection initiated by the Client provided that the downstream connection has already been 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 [RFC8305].

A fourth difference is that the 0-RTT TCP Convert Protocol only allows the Client to specify the IP address/port number of the destination Server and not a DNS name. We evaluated an alternate design that included the DNS name of the remote peer instead of its IP address as in SOCKS [RFC1928]. However, that design was not adopted because it induces both an extra load and increased delays on the Transport Converter to handle and manage DNS resolution requests. Note that the name resolution at the Converter may fail (e.g., private names discussed in Section 2.1 of [RFC6731]) or may not match the one that would be returned by a Client's resolution library (e.g., Section 2.2 of [RFC6731]).

4. Architecture and Behaviors

4.1. Functional Elements

The Convert Protocol considers three functional elements:

- Clients
- Transport Converters
- Servers

A Transport Converter is a network function that proxies all data exchanged over one upstream connection to one downstream connection and vice versa (Figure 2). Thus, the Transport Converter maintains state that associates one upstream connection to a corresponding downstream connection.

A connection can be initiated from both sides of the Transport Converter (External realm, Internal realm).

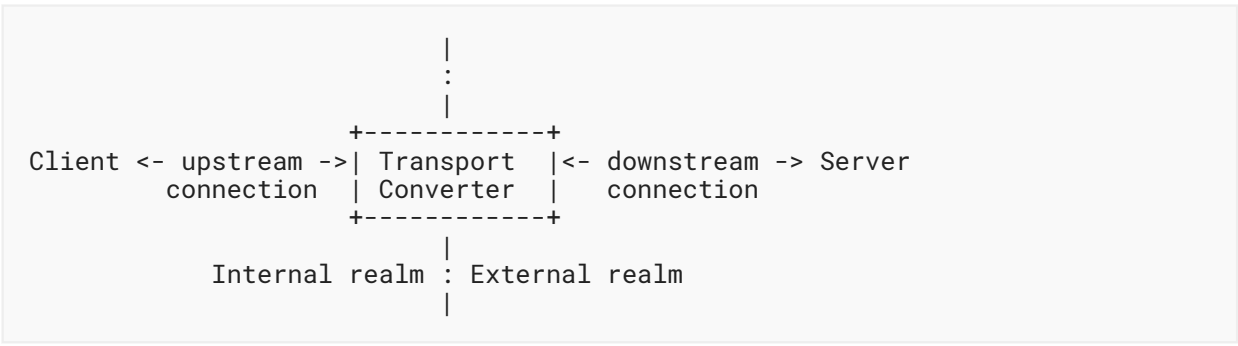

Figure 2: A Transport Converter Proxies Data between Pairs of TCP Connections

"Client" refers to a software instance embedded on a host that can reach a Transport Converter in the internal realm. The "Client" can initiate connections via a Transport Converter (referred to as outgoing connections). Also, the "Client" can accept incoming connections via a Transport Converter (referred to as incoming connections).

A Transport Converter can be embedded in a standalone device or be activated as a service on a router. How such a function is enabled is deployment specific.

The architecture assumes that new software will be installed on the Client hosts to interact with one or more Transport Converters. Furthermore, the architecture allows for making use of new TCP extensions even if those are not supported by a given Server.

A Client is configured, through means that are outside the scope of this document, with the names and/or addresses of one or more Transport Converters and the TCP extensions that they support. The procedure for selecting a Transport Converter among a list of configured Transport Converters is outside the scope of this document.

One of the benefits of this design is that different transport protocol extensions can be used on the upstream and the downstream connections. This encourages the deployment of new TCP extensions until they are widely supported, in particular, by Servers.

The architecture does not mandate anything on the Server side.

Similar to SOCKS, the architecture does not interfere with end-to-end TLS connections [RFC8446] between the Client and the Server (Figure 3). In other words, end-to-end TLS is supported in the presence of a Converter.

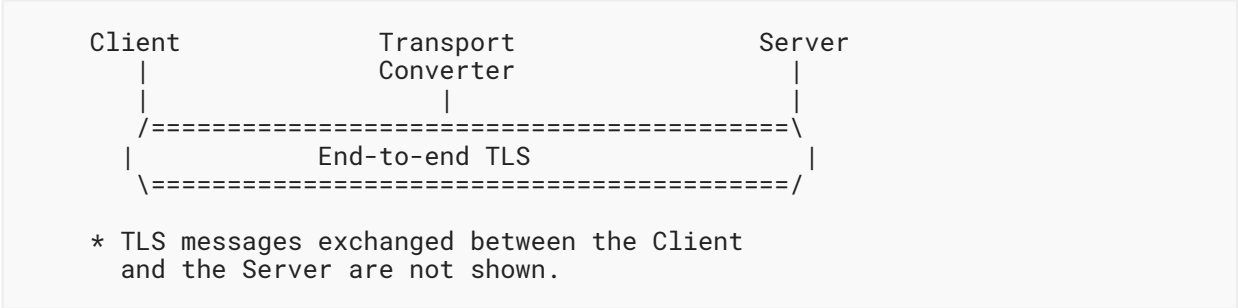

Figure 3: End-to-end TLS via a Transport Converter

It is out of scope of this document to elaborate on specific considerations related to the use of TLS in the Client-Converter connection leg to exchange Convert messages (in addition to the end-to-end TLS connection). In particular, (1) assessment of whether 0-RTT data mode discussed in Section 2.3 of [RFC8446] is safe under replay and (2) specification of a profile for its use (Appendix E.5 of [RFC8446]) are out of scope.

4.2. Theory of Operation

At a high level, the objective of the Transport Converter is to allow the use a specific extension, e.g., Multipath TCP, on a subset of the path even if the peer does not support this extension. This is illustrated in Figure 4 where the Client initiates a Multipath TCP connection with the Transport Converter (packets belonging to the Multipath TCP connection are shown with "===") while the Transport Converter uses a TCP connection with the Server.

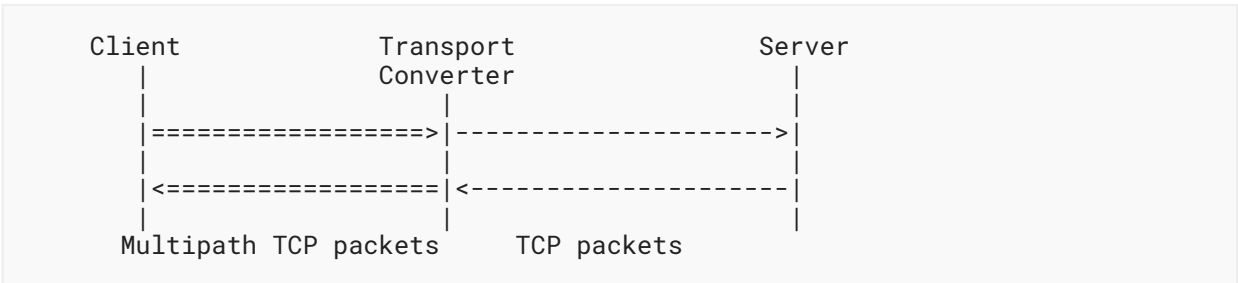

Figure 4: An Example of 0-RTT Network-Assisted Outgoing MPTCP Connection

The packets belonging to a connection established through a Transport Converter may follow a different path than the packets directly exchanged between the Client and the Server. Deployments should minimize the possible additional delay by carefully selecting the location of the Transport Converter used to reach a given destination.

When establishing a connection, the Client can, depending on local 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 (that is, the conversion service is used), the Client initiates a connection towards the Transport Converter and indicates the IP address and port number of the Server within the connection establishment packet. Doing so enables the Transport Converter

to immediately initiate a connection towards that Server without experiencing an extra delay. The Transport Converter waits until the receipt of the confirmation that the Server agrees to establish the connection before confirming it to the Client.

The Client places the destination address and port number of the Server in the payload of the SYN sent to the Transport Converter to minimize connection establishment delays. The Transport Converter maintains two connections that are combined together:

- The upstream connection is the one between the Client and the Transport Converter.
- The downstream connection is the one between the Transport Converter and the Server.

Any user data received by the Transport Converter over the upstream (or downstream) connection is proxied over the downstream (or upstream) connection.

Figure 5 illustrates the establishment of an outgoing TCP connection by a Client through a Transport Converter.

> Note: The information shown between brackets in Figure 5 (and other figures in the document) refers to Convert Protocol messages described in Section 6.

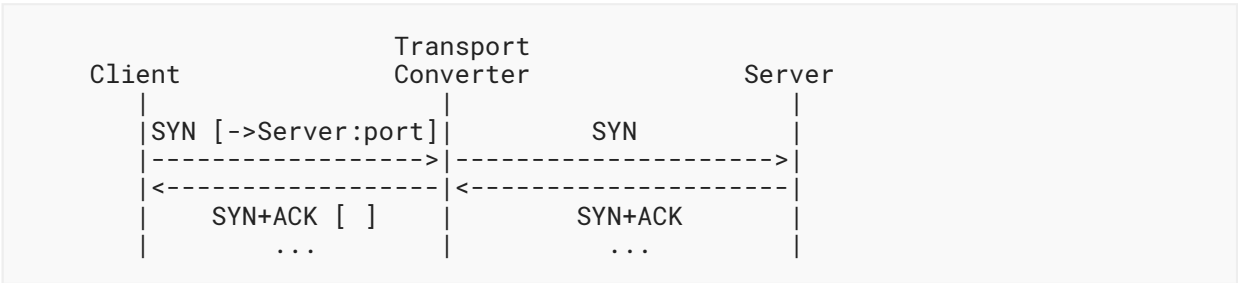

Figure 5: Establishment of an Outgoing TCP Connection through a Transport Converter

The Client sends a SYN destined to the Transport Converter. The payload of this SYN contains the address and port number of the Server. The Transport Converter does not reply immediately to this SYN. It first tries to create a TCP connection towards the target Server. If this upstream connection succeeds, the Transport Converter confirms the establishment of the connection to the Client by returning a SYN+ACK and the first bytes of the bytestream contain information about the TCP options that were negotiated with the Server. Also, a state entry is instantiated for this connection. This state entry is used by the Converter to handle subsequent messages belonging to the connection.

The connection can also be established from the Internet towards a Client via a Transport Converter (Figure 6). This is typically the case when the Client hosts an application Server that listens to a specific port number. When the Converter receives an incoming SYN from a remote host, it checks if it can provide the conversion service for the destination IP address and destination port number of that SYN. The Transport Converter receives this SYN because it is, for example, on the path between the remote host and the Client or it provides address-sharing service for the Client (Section 2 of [RFC6269]). If the check fails, the packet is silently ignored by

the Converter. If the check is successful, the Converter tries to initiate a TCP connection towards the Client from its own address and using its configured TCP options. In the SYN that corresponds to this connection attempt, the Transport Convert inserts a TLV message that indicates the source address and port number of the remote host. A transport session entry is created by the Converter for this connection. SYN+ACK and ACK will then be exchanged between the Client, the Converter, and remote host to confirm the establishment of the connection. The Converter uses the transport session entry to proxy packets belonging to the connection.

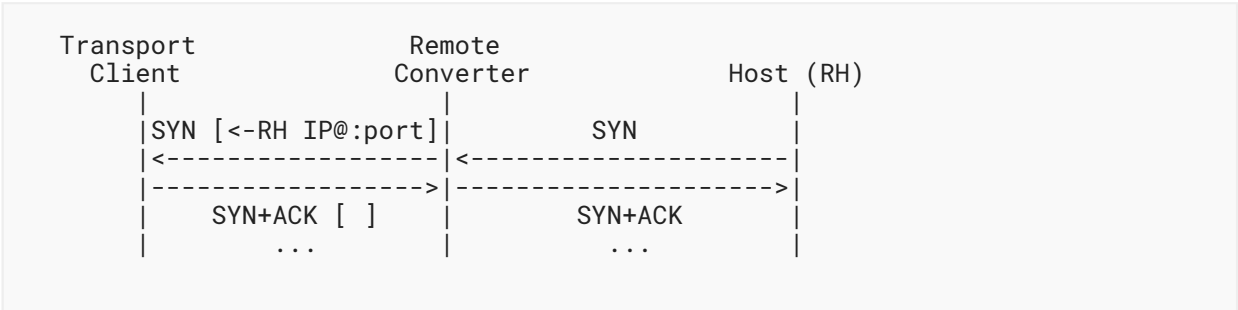

Figure 6: Establishment of an Incoming TCP Connection through a Transport Converter

Standard TCP (Section 3.4 of [RFC0793]) allows a SYN packet to carry data inside its payload but forbids the receiver from delivering it to the application until completion of the three-way-handshake. To enable applications to exchange data in a TCP handshake, this specification follows an approach similar to TCP Fast Open [RFC7413] and thus, removes the constraint by allowing data in SYN packets to be delivered to the Transport Converter application.

As discussed in [RFC7413], such change to TCP semantics raises two issues. First, duplicate SYNs can cause problems for applications that rely on TCP; whether or not a given application is affected depends on the details of that application protocol. Second, TCP suffers from SYN flooding attacks [RFC4987]. TFO solves these two problems for applications that can tolerate replays by using the TCP Fast Open option that includes a cookie. However, the utilization of this option consumes space in the limited TCP header. Furthermore, there are situations, as noted in Section 7.3 of [RFC7413], where it is possible to accept the payload of SYN packets without creating additional security risks such as a network where addresses cannot be spoofed and the Transport Converter only serves a set of hosts that are identified by these addresses.

For these reasons, this specification does not mandate the use of the TCP Fast Open option when the Client sends a connection establishment packet towards a Transport Converter. The Convert Protocol includes an optional Cookie TLV that provides similar protection as the TCP Fast Open option without consuming space in the TCP header. Furthermore, this design allows for the use of longer cookies than [RFC7413].

If the downstream (or upstream) connection fails for some reason (excessive retransmissions, reception of an RST segment, etc.), then the Converter reacts by forcing the teardown of the upstream (or downstream) connection. In particular, if an ICMP error message that indicates a hard error is received on the downstream connection, the Converter echoes the Code field of that ICMP message in a Destination Unreachable Error TLV (see Section 6.2.8) that it transmits to the Client. Note that if an ICMP error message that indicates a soft error is received on the

downstream connection, the Converter will retransmit the corresponding data until it is acknowledged or the connection times out. A classification of ICMP soft and hard errors is provided in Table 1 of [RFC5461].

The same reasoning applies when the upstream connection ends with an exchange of FIN segments. In this case, the Converter will also terminate the downstream connection by using FIN segments. If the downstream connection terminates with the exchange of FIN segments, the Converter should initiate a graceful termination of the upstream connection.

4.3. Data Processing at the Transport Converter

As mentioned in Section 4.2, the Transport Converter acts as a TCP proxy between the upstream connection (i.e., between the Client and the Transport Converter) and the downstream connection (i.e., between the Transport Converter and the Server).

The control messages (i.e., the Convert messages discussed in Section 6) establish state (called transport session entry) in the Transport Converter that will enable it to proxy between the two TCP connections.

The Transport Converter uses the transport session entry to proxy packets belonging to the connection. An implementation example of a transport session entry for TCP connections is shown in Figure 7.

```
(C,c) <--> (T,t), (S,s), Lifetime
```

Figure 7: An Example of Transport Session Entry

Where:

- C and c are the source IP address and source port number used by the Client for the upstream connection.
- S and s are the Server's IP address and port number.
- T and t are the source IP address and source port number used by the Transport Converter to proxy the connection.
- Lifetime is a timer that tracks the remaining lifetime of the entry as assigned by the Converter. When the timer expires, the entry is deleted.

Clients send packets bound to connections eligible for the conversion service to the provisioned Transport Converter and destination port number. This applies for both control messages and data. Additional information is supplied by Clients to the Transport Converter by means of Convert messages as detailed in Section 6. User data can be included in SYN or non-SYN messages. User data is unambiguously distinguished from Convert TLVs by a Transport Converter owing to the Convert Fixed Header in the Convert messages (Section 6.1). These Convert TLVs are destined to the Transport Convert and are, thus, removed by the Transport Converter when proxying between the two connections.

Upon receipt of a packet that belongs to an existing connection between a Client and the Transport Converter, the Converter proxies the user data to the Server using the information stored in the corresponding transport session entry. For example, in reference to Figure 7, the Transport Converter proxies the data received from (C,c) downstream using (T,t) as source transport address and (S,s) as destination transport address.

A similar process happens for data sent from the Server. The Converter acts as a TCP proxy and sends the data to the Client relying upon the information stored in a transport session entry. The Converter associates a lifetime with state entries used to bind an upstream connection with its downstream connection.

When Multipath TCP is used between the Client and the Transport Converter, the Converter maintains more state (e.g., information about the subflows) for each Multipath TCP connection. The procedure described above continues to apply except that the Converter needs to manage the establishment/termination of subflows and schedule packets among the established ones. These operations are part of the Multipath TCP implementation. They are independent of the Convert Protocol that only processes the Convert messages in the beginning of the bytestream.

A Transport Converter may operate in address preservation mode (that is, the Converter does not rewrite the source IP address (i.e., C==T)) or address-sharing mode (that is, an address pool is shared among all Clients serviced by the Converter (i.e., C!=T)); refer to Section 4.4 for more details. Which behavior to use by a Transport Converter is deployment specific. If address-sharing mode is enabled, the Transport Converter **MUST** adhere to REQ-2 of [RFC6888], which implies a default "IP address pooling" behavior of "Paired" (as defined in Section 4.1 of [RFC4787]) **MUST** be supported. This behavior is meant to avoid breaking applications that depend on the source address remaining constant.

4.4. Address Preservation vs. Address Sharing

The Transport Converter is provided with instructions about the behavior to adopt with regard to the processing of source addresses of outgoing packets. The following subsections discuss two deployment models for illustration purposes. It is out of the scope of this document to make a recommendation.

4.4.1. Address Preservation

In this model, the visible source IP address of a packet proxied by a Transport Converter to a Server is an IP address of the end host (Client). No dedicated IP address pool is provisioned to the Transport Converter, but the Transport Converter is located on the path between the Client and the Server.

For Multipath TCP, the Transport Converter preserves the source IP address used by the Client when establishing the initial subflow. Data conveyed in secondary subflows will be proxied by the Transport Converter using the source IP address of the initial subflow. An example of a proxied Multipath TCP connection with address preservation is shown in Figure 8.

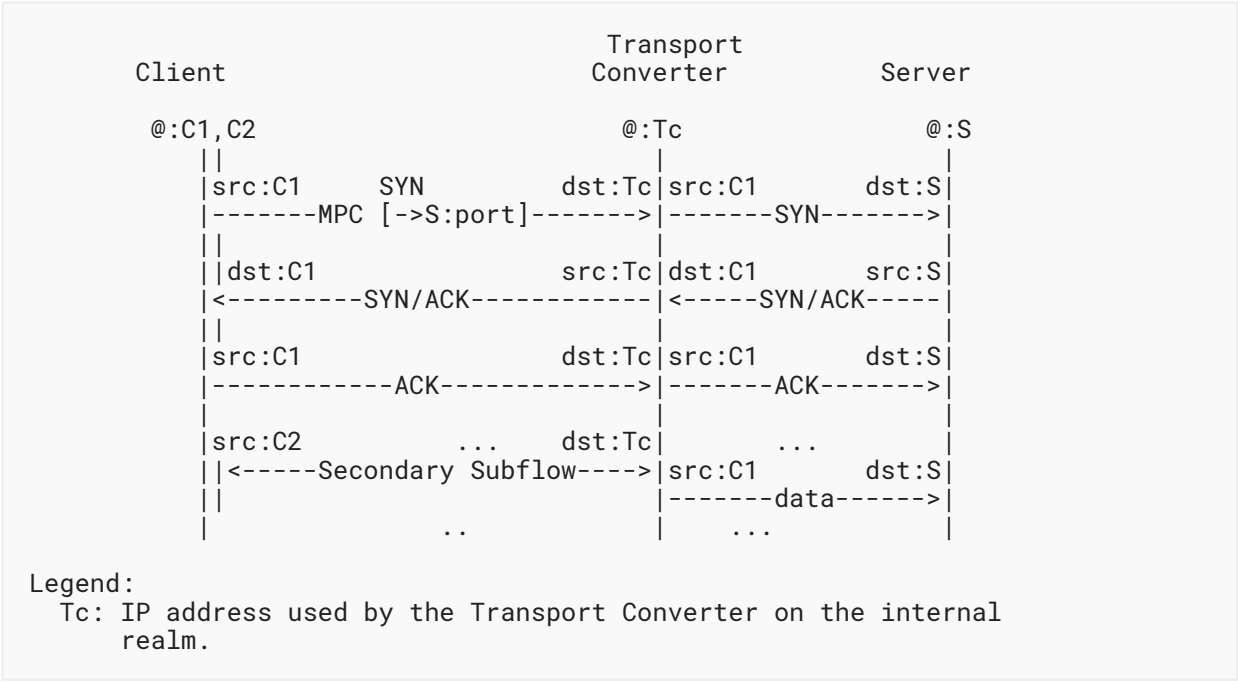

Figure 8: Example of Address Preservation

The Transport Converter must be on the forwarding path of incoming traffic. Because the same (destination) IP address is used for both proxied and non-proxied connections, the Transport Converter should not drop incoming packets it intercepts if no matching entry is found for the packets. Unless explicitly configured otherwise, such packets are forwarded according to the instructions of a local forwarding table.

4.4.2. Address/Prefix Sharing

A pool of global IPv4 addresses is provisioned to the Transport Converter along with possible instructions about the address-sharing ratio to apply (see Appendix B of [RFC6269]). An address is thus shared among multiple Clients.

Likewise, rewriting the source IPv6 prefix [RFC6296] may be used to ease redirection of incoming IPv6 traffic towards the appropriate Transport Converter. A pool of IPv6 prefixes is then provisioned to the Transport Converter for this purpose.

Adequate forwarding policies are enforced so that traffic destined to an address of such a pool is intercepted by the appropriate Transport Converter. Unlike Section 4.4.1, the Transport Converter drops incoming packets that do not match an active transport session entry.

An example is shown in Figure 9.

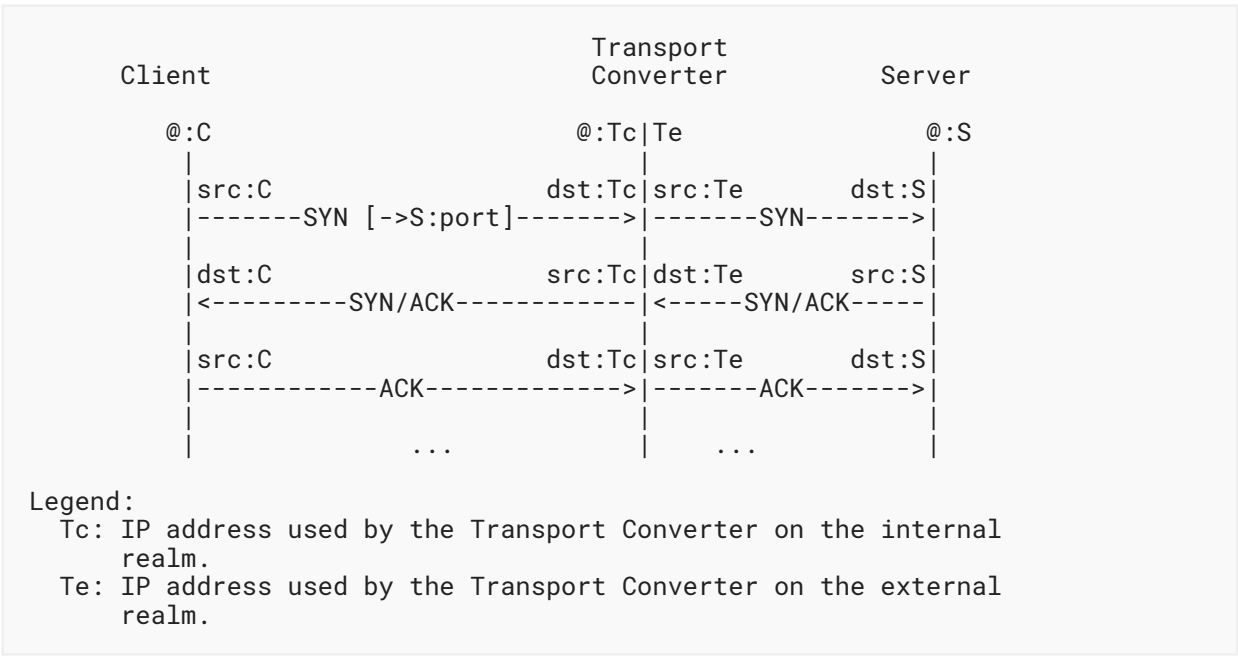

Figure 9: Address Sharing

5. Sample Examples

5.1. Outgoing Converter-Assisted Multipath TCP Connections

As an example, let us consider how the Convert Protocol can help the deployment of Multipath TCP. We assume that both the Client and the Transport Converter support Multipath TCP but consider two different cases depending on whether or not the Server supports Multipath TCP.

As a reminder, a Multipath TCP connection is created by placing the MP_CAPABLE (MPC) option in the SYN sent by the Client.

Figure 10 describes the operation of the Transport Converter if the Server does not support Multipath TCP.

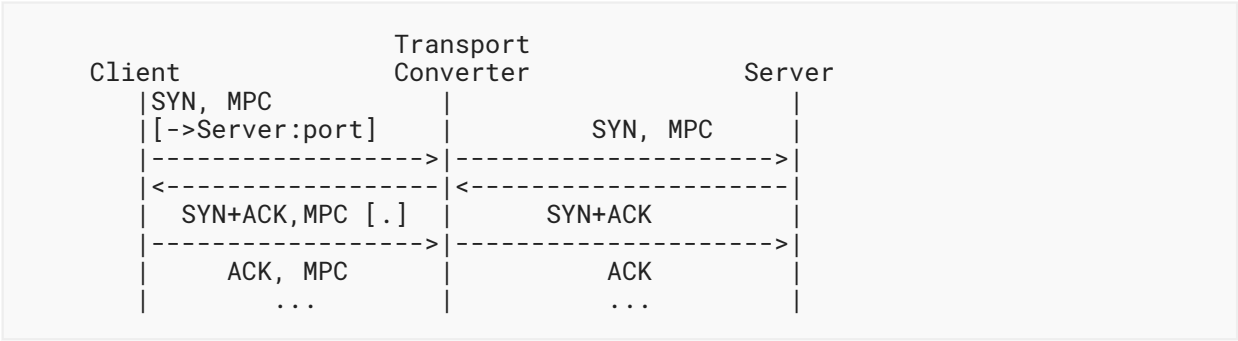

Figure 10: Establishment of a Multipath TCP Connection through a Transport Converter towards a Server That Does Not support Multipath TCP

The Client tries to initiate a Multipath TCP connection by sending a SYN with the MP_CAPABLE option (MPC in Figure 10). The SYN includes the address and port number of the target Server, that are extracted and used by the Transport Converter 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 returns the extended TCP header received from the Server to the Client.

Note that, if the TCP connection is reset for some reason, the Converter tears down the Multipath TCP connection by transmitting an MP_FASTCLOSE. Likewise, if the Multipath TCP connection ends with the transmission of DATA_FINs, the Converter terminates the TCP connection by using FIN segments. As a side note, given that with Multipath TCP, RST only has the scope of the subflow and will only close the concerned subflow but not affect the remaining subflows, the Converter does not terminate the downstream TCP connection upon receipt of an RST over a Multipath subflow.

Figure 11 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 to the Client that the Server supports Multipath TCP. With this information, the Client has discovered that the Server supports Multipath TCP. This will enable the Client to bypass the Transport Converter for the subsequent Multipath TCP connections that it will initiate towards this Server.

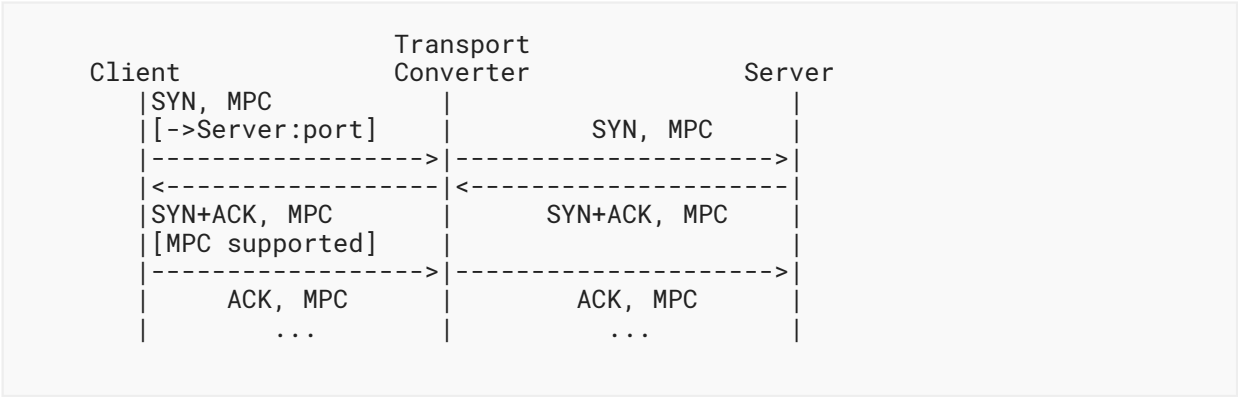

Figure 11: Establishment of a Multipath TCP Connection through a Converter towards an MPTCP-Capable Server

5.2. Incoming Converter-Assisted Multipath TCP Connection

An example of an incoming Converter-assisted Multipath TCP connection is depicted in Figure 12. In order to support incoming connections from remote hosts, the Client may use the Port Control Protocol (PCP) [RFC6887] to instruct the Transport Converter to create dynamic mappings. Those mappings will be used by the Transport Converter to intercept an incoming TCP connection destined to the Client and convert it into a Multipath TCP connection.

Typically, the Client sends a PCP request to the Converter asking to create an explicit TCP mapping for the internal IP address and internal port number. The Converter accepts the request by creating a TCP mapping for the internal IP address, internal port number, external IP address, and external port number. The external IP address, external port number, and assigned lifetime are returned back to the Client in the PCP response. The external IP address and external port number will then be advertised by the Client (or the user) using an out-of-band mechanism so that remote hosts can initiate TCP connections to the Client via the Converter. Note that the external and internal information may be the same.

Then, when the Converter receives an incoming SYN, it checks its mapping table to verify if there is an active mapping matching the destination IP address and destination port of that SYN. If no entry is found, the Converter silently ignores the message. If an entry is found, the Converter inserts an MP_CAPABLE option and Connect TLV in the SYN packet, and rewrites the source IP address to one of its IP addresses and, eventually, the destination IP address and port number in accordance with the information stored in the mapping. SYN+ACK and ACK will then be exchanged between the Client and the Converter to confirm the establishment of the initial subflow. The Client can add new subflows following normal Multipath TCP procedures.

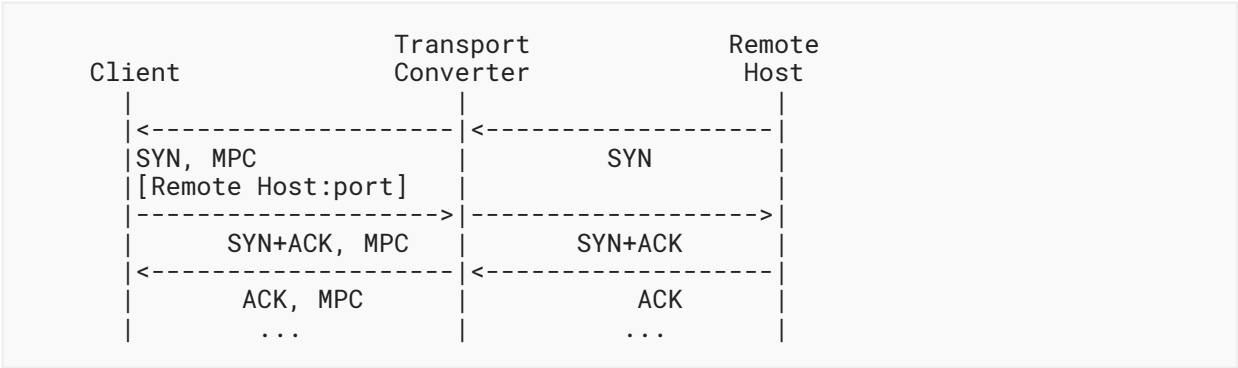

Figure 12: Establishment of an Incoming Multipath TCP Connection through a Transport Converter

It is out of scope of this document to define specific Convert TLVs to manage incoming connections (that is, TLVs that mimic PCP messages). These TLVs can be defined in a separate document.

6. The Convert Protocol (Convert)

This section defines the Convert Protocol (Convert, for short) messages that are exchanged between a Client and a Transport Converter.

The Transport Converter listens on a specific TCP port number for Convert messages from Clients. That port number is configured by an administrator. Absent any policy, the Transport Converter **SHOULD** silently ignore SYNs with no Convert TLVs.

Convert messages may appear only in SYN, SYN+ACK, or ACK.

Convert messages **MUST** be included as the first bytes of the bytestream. All Convert messages start with a fixed header that is 32 bits long (Section 6.1) followed by one or more Convert TLVs (Type, Length, Value) (Section 6.2).

If the initial SYN message contains user data in its payload (e.g., see [RFC7413]), that data **MUST** be placed right after the Convert TLVs when generating the SYN.

The protocol can be extended by defining new TLVs or bumping the version number if a different message format is needed. If a future version is defined but with a different message format, the version negotiation procedure defined in Section 6.2.8 (see "Unsupported Version") is meant to agree on a version that is supported by both peers.

> Implementation note 1: Several implementers expressed concerns about the use of TFO. As a reminder, the Fast Open Cookie protects from some attack scenarios that affect open servers like web servers. The Convert Protocol is different and, as discussed in [RFC7413], there are different ways to protect from such attacks. Instead of using a Fast Open Cookie inside the TCP options, which consumes precious space in the extended TCP header, the Convert Protocol supports the

> utilization of a Cookie that is placed in the SYN payload. This provides the same level of protection as a Fast Open Cookie in environments were such protection is required.
>
> Implementation note 2: Error messages are not included in RST but sent in the bytestream. Implementers have indicated that processing RST on Clients was difficult on some platforms. This design simplifies Client implementations.

6.1. The Convert Fixed Header

The Convert Protocol uses a fixed header that is 32 bits long sent by both the Client and the Transport Converter over each established connection. This header indicates both the version of the protocol used and the length of the Convert message.

The Client and the Transport Converter **MUST** send the fixed-sized header, shown in Figure 13, as the first four bytes of the bytestream.

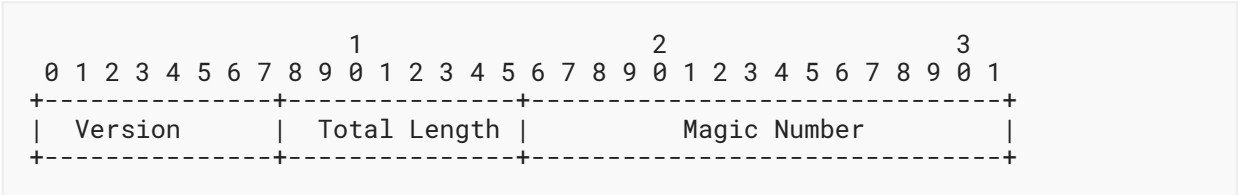

Figure 13: The Convert Fixed Header

The version is encoded as an 8-bit unsigned integer value. This document specifies version 1. Version 0 is reserved by this document and **MUST NOT** be used.

> Note: Early versions of this specification don't use a dedicated port number but only rely upon the IP address of the Converter. Having a bit set in the Version field together with the Total Length field avoids misinterpreting data in a SYN as Convert TLVs. Since the design was updated to use a specific service port, that constraint was relaxed. Version 0 would work, but given existing implementations already use Version 1, the use of Version 0 is maintained as reserved.

The Total Length is the number of 32-bit words, including the header, of the bytestream that are consumed by the Convert messages. Since Total Length is also an 8-bit 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 a header with such a total length.

The Magic Number field **MUST** be set to 0x2263. This field is meant to further strengthen the protocol to unambiguously distinguish any data supplied by an application from Convert TLVs.

The Total Length field unambiguously marks the number of 32-bit words that carry Convert TLVs in the beginning of the bytestream.

6.2. Convert TLVs

6.2.1. Generic Convert TLV Format

The Convert Protocol uses variable length messages that are encoded using the generic TLV format depicted in Figure 14.

The length of all TLVs used by the Convert Protocol is always a multiple of four bytes. All TLVs are aligned on 32-bit boundaries. All TLV fields are encoded using the network byte order.

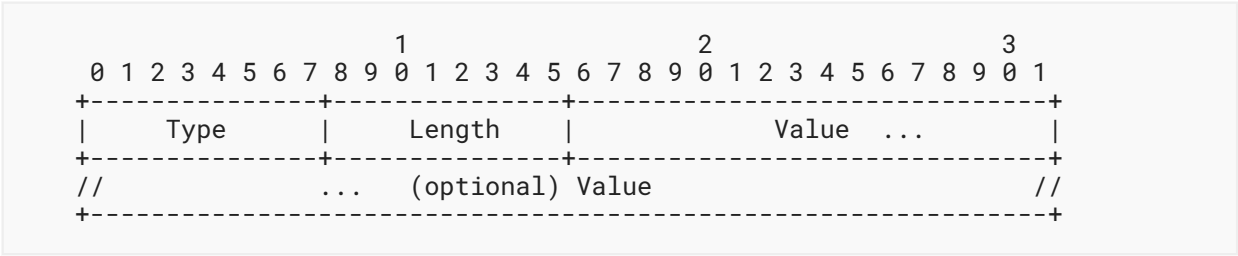

Figure 14: Convert Generic TLV Format

The Length field covers Type, Length, and Value fields. It is expressed in units of 32-bit words. If necessary, Value **MUST** be padded with zeroes so that the length of the TLV is a multiple of 32 bits.

A given TLV **MUST** only appear once on a connection. If a Client receives two or more instances of the same TLV over a Convert connection, it **MUST** reset the associated TCP connection. If a Converter receives two or more instances of the same TLV over a Convert connection, it **MUST** return a Malformed Message Error TLV and close the associated TCP connection.

6.2.2. Summary of Supported Convert TLVs

This document specifies the following Convert TLVs:

Type	Hex	Length	Description
1	0x1	1	Info TLV
10	0xA	Variable	Connect TLV
20	0x14	Variable	Extended TCP Header TLV
21	0x15	Variable	Supported TCP Extensions TLV
22	0x16	Variable	Cookie TLV
30	0x1E	Variable	Error TLV

Table 1: The TLVs Used by the Convert Protocol

Type 0x0 is a reserved value. If a Client receives a TLV of type 0x0, it **MUST** reset the associated TCP connection. If a Converter receives a TLV of type 0x0, it **MUST** return an Unsupported Message Error TLV and close the associated TCP connection.

The Client typically sends, in the first connection it established with a Transport Converter, the Info TLV (Section 6.2.3) to learn its capabilities. Assuming the Client is authorized to invoke the Transport Converter, the latter replies with the Supported TCP Extensions TLV (Section 6.2.4).

The Client can request the establishment of connections to Servers by using the Connect TLV (Section 6.2.5). If the connection can be established with the final Server, the Transport Converter replies with the Extended TCP Header TLV (Section 6.2.6). If not, the Transport Converter **MUST** return an Error TLV (Section 6.2.8) and then close the connection. The Transport Converter **MUST NOT** send an RST immediately after the detection of an error to let the Error TLV reach the Client. As explained later, the Client will send an RST regardless upon reception of the Error TLV.

6.2.3. The Info TLV

The Info TLV (Figure 15) is an optional TLV that can be 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. Assuming a Client is entitled to invoke the Transport Converter, the latter replies with the Supported TCP Extensions TLV described in Section 6.2.4.

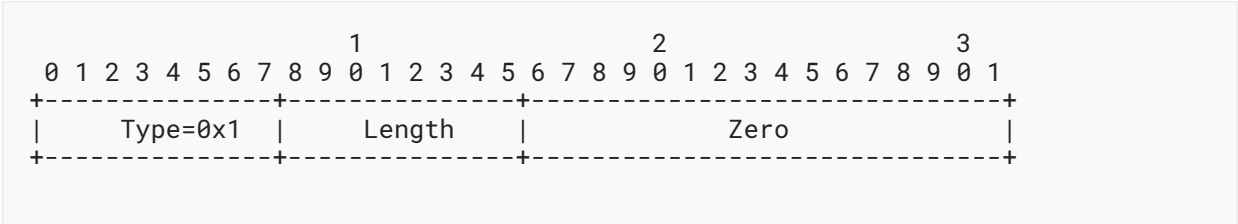

Figure 15: The Info TLV

6.2.4. Supported TCP Extensions TLV

The Supported TCP Extensions TLV (Figure 16) is used by a Transport Converter to announce the TCP options for which it provides a conversion service. A Transport Converter **SHOULD** include in this list the TCP options that it supports in outgoing SYNs.

Each supported TCP option is encoded with its TCP option Kind listed in the "Transmission Control Protocol (TCP) Parameters" registry maintained by IANA [IANA-CONVERT]. The Unassigned field **MUST** be set to zero by the Transport Converter and ignored by the Client.

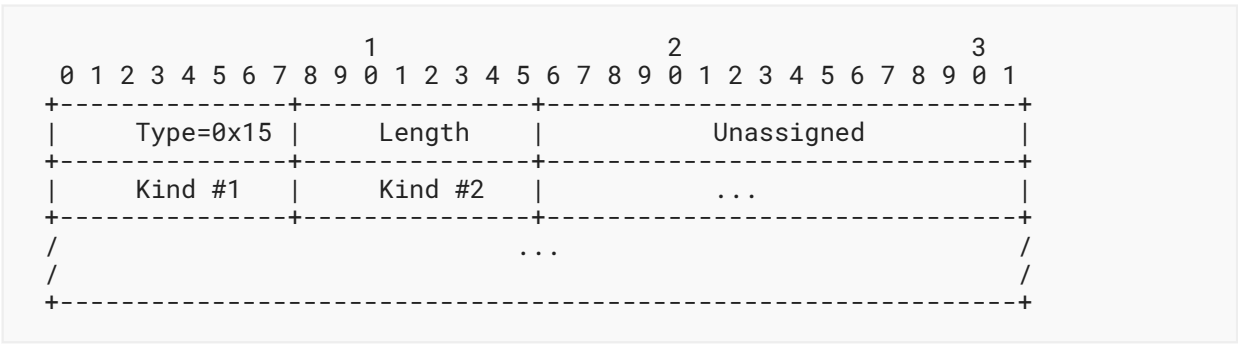

Figure 16: The Supported TCP Extensions TLV

TCP option Kinds 1 and 2 defined in [RFC0793] are supported by all TCP implementations and thus, **MUST NOT** appear in this list.

The list of Supported TCP Extensions is padded with 0 to end on a 32-bit boundary.

For example, if the Transport Converter supports Multipath TCP, Kind=30 will be present in the Supported TCP Extensions TLV that it returns in response to the Info TLV.

6.2.5. Connect TLV

The Connect TLV (Figure 17) is used to request the establishment of a connection via a Transport Converter. This connection can be from or to a Client.

The Remote Peer Port and Remote Peer IP Address fields contain the destination port number and IP address of the Server, for outgoing connections. For incoming connections destined to a Client serviced via a Transport Converter, these fields convey the source port number and IP address of the SYN packet received by the Transport Converter from the Server.

The Remote Peer IP Address **MUST** be encoded as an IPv6 address. IPv4 addresses **MUST** be encoded using the IPv4-mapped IPv6 address format defined in [RFC4291]. Further, the Remote Peer IP Address field **MUST NOT** include multicast, broadcast, or host loopback addresses [RFC6890]. If a Converter receives a Connect TLV with such invalid addresses, it **MUST** reply with a Malformed Message Error TLV and close the associated TCP connection.

We distinguish two types of Connect TLV based on their length: (1) the Base Connect TLV has a length set to 5 (i.e., 20 bytes) and contains a remote address and a remote port (Figure 17), and (2) the Extended Connect TLV spans more than 20 bytes and also includes the optional TCP Options field (Figure 18). This field is used to request the advertisement of specific TCP options to the Server.

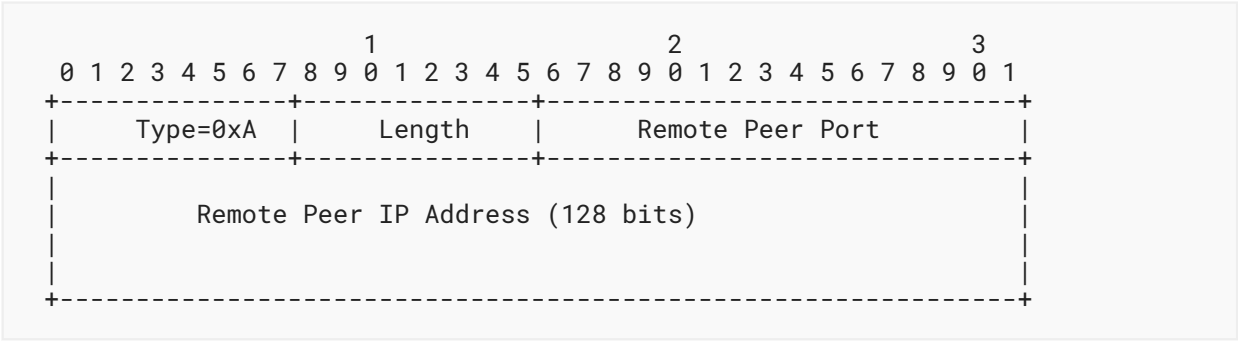

Figure 17: The Base Connect TLV

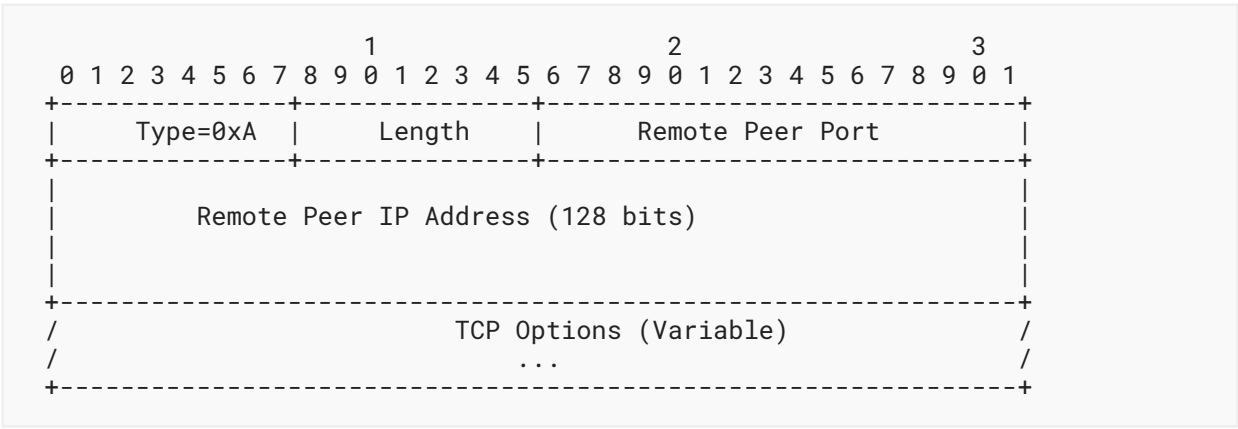

Figure 18: The Extended Connect TLV

The TCP Options field is a variable length field that carries a list of TCP option fields (Figure 19). Each TCP option field is encoded as a block of 2+n bytes where the first byte is the TCP option Kind and the second byte is the length of the TCP option as specified in [RFC0793]. The minimum value for the TCP option Length is 2. The TCP options that do not include a length sub-field, i.e., option types 0 (EOL) and 1 (NOP) defined in [RFC0793] **MUST NOT** 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 2 indicates the absence of the Value field. The TCP options field always ends on a 32-bit boundary after being padded with zeros.

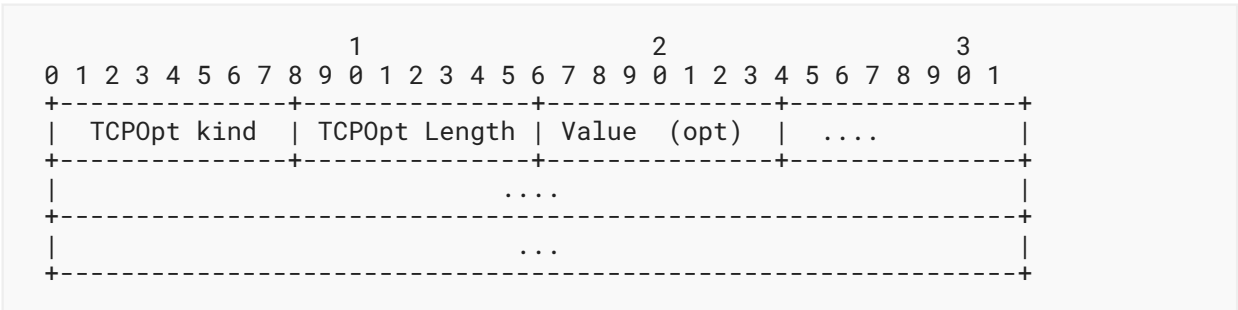

Figure 19: The TCP Options Field

Upon reception of a Base Connect TLV, and absent any policy (e.g., rate-limit) or resource exhaustion conditions, a Transport Converter attempts to establish a connection to the address and port that it contains. The Transport Converter **MUST** use by default the TCP options that correspond to its local policy to establish this connection.

Upon reception of an Extended Connect TLV, a Transport Converter first checks whether or not it supports the TCP Options listed in the TCP Options field. If not, it returns an error TLV set to "Unsupported TCP Option" (Section 6.2.8). If the above check succeeded, and absent any rate-limit policy or resource exhaustion conditions, a Transport Converter **MUST** attempt to establish a connection to the address and port that it contains. It **MUST** include in the SYN that it sends to the Server the options listed in the TCP Options subfield and the TCP options that it would have used according to its local policies. For the TCP options that are included in the TCP Options field without an optional value, the Transport Converter **MUST** generate its own value. For the TCP options that are included in the TCP Options field with an optional value, it **MUST** copy the entire option in the SYN sent to the remote Server. This procedure is designed with TFO in mind. Particularly, this procedure allows to successfully exchange a Fast Open Cookie between the Client and the Server. See Section 7 for a detailed discussion of the different types of TCP options.

The Transport Converter may refuse a Connect TLV request for various reasons (e.g., authorization failed, out of resources, invalid address type, or unsupported TCP option). An error message indicating the encountered error is returned to the requesting Client (Section 6.2.8). In order to prevent denial-of-service attacks, error messages sent to a Client **SHOULD** be rate-limited.

6.2.6. Extended TCP Header TLV

The Extended TCP Header TLV (Figure 20) is used by the Transport Converter to return to the Client the TCP options that were returned by the Server in the SYN+ACK packet. A Transport Converter **MUST** return this TLV if the Client sent an Extended Connect TLV and the connection was accepted by the Server.

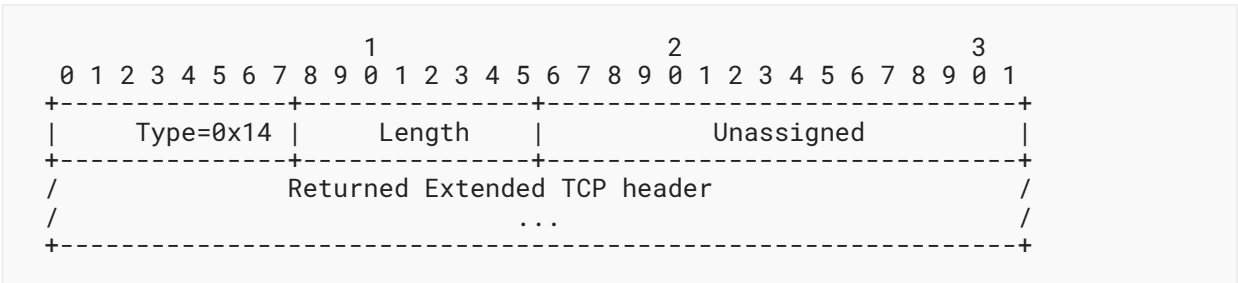

Figure 20: The Extended TCP Header TLV

The Returned Extended TCP header field is a copy of the TCP Options that were included in the SYN+ACK received by the Transport Converter.

The Unassigned field **MUST** be set to zero by the sender and ignored by the receiver.

6.2.7. The Cookie TLV

The Cookie TLV (Figure 21) is an optional TLV that is similar to the TCP Fast Open Cookie [RFC7413]. A Transport Converter may want to verify that a Client can receive the packets that it sends to prevent attacks from spoofed addresses. This verification can be done by using a Cookie that is bound to, for example, the IP address(es) of the Client. This Cookie can be configured on the Client by means that are outside of this document or provided by the Transport Converter.

A Transport Converter that has been configured to use the optional Cookie TLV **MUST** verify the presence of this TLV in the payload of the received SYN. If this TLV is present, the Transport Converter **MUST** validate the Cookie by means similar to those in Section 4.1.2 of [RFC7413] (i.e., IsCookieValid). If the Cookie is valid, the connection establishment procedure can continue. Otherwise, the Transport Converter **MUST** return an Error TLV set to "Not Authorized" and close the connection.

If the received SYN did not contain a Cookie TLV, and cookie validation is required, the Transport Converter **MAY** compute a Cookie bound to this Client address. In such case, the Transport Converter **MUST** return an Error TLV set to "Missing Cookie" and the computed Cookie and close the connection. The Client will react to this error by first issuing a reset to terminate the connection. It also stores the received Cookie in its cache and attempts to reestablish a new connection to the Transport Converter that includes the Cookie TLV.

The format of the Cookie TLV is shown in Figure 21.

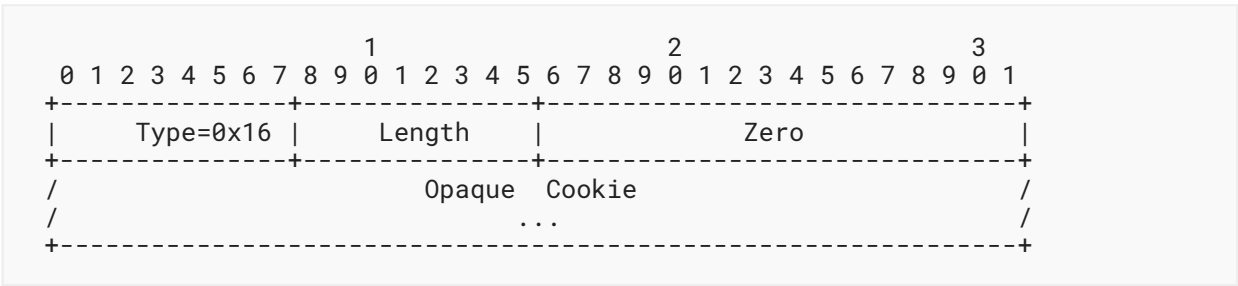

Figure 21: The Cookie TLV

6.2.8. Error TLV

The Error TLV (Figure 22) is meant to provide information about some errors that occurred during the processing of a Convert message. This TLV has a variable length. Upon reception of an Error TLV, a Client **MUST** reset the associated connection.

An Error TLV can be included in the SYN+ACK or an ACK.

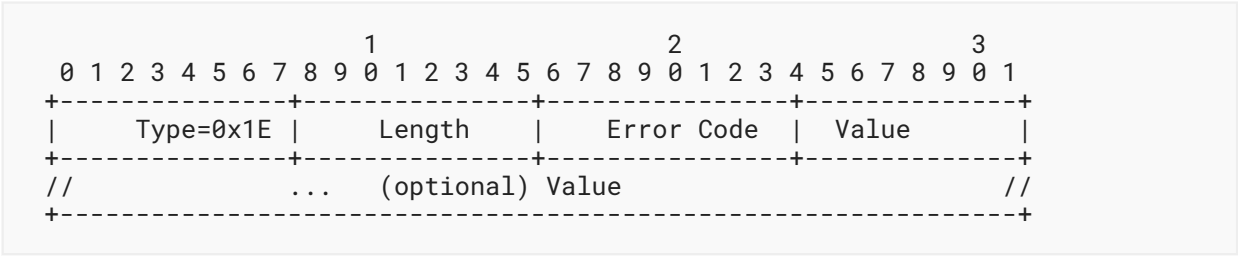

Figure 22: The Error TLV

Different types of errors can occur while processing Convert messages. Each error is identified by an Error Code represented as an unsigned integer. Four classes of error codes are defined:

Message validation and processing errors (0-31 range):
: Returned upon reception of an invalid message (including valid messages but with invalid or unknown TLVs).

Client-side errors (32-63 range):
: The Client sent a request that could not be accepted by the Transport Converter (e.g., unsupported operation).

Converter-side errors (64-95 range):
: Problems encountered on the Transport Converter (e.g., lack of resources) that prevent it from fulfilling the Client's request.

Errors caused by the destination Server (96-127 range):
: The final destination could not be reached or it replied with a reset.

The following error codes are defined in this document:

Unsupported Version (0):
: The version number indicated in the fixed header of a message received from a peer is not supported.

  This error code **MUST** be generated by a peer (e.g., Transport Converter) when it receives a request having a version number that it does not support.

  The Value field **MUST** be set to the version supported by the peer. When multiple versions are supported by the peer, it includes the list of supported versions in the Value field; each version is encoded in 8 bits. The list of supported versions **MUST** be padded with zeros to end on a 32-bit boundary.

  Upon receipt of this error code, the remote peer (e.g., Client) checks whether it supports one of the versions returned by the peer. The highest commonly supported version number **MUST** be used by the remote peer in subsequent exchanges with the peer.

Malformed Message (1):
: This error code is sent to indicate that a message received from a peer cannot be successfully parsed and validated.

  Typically, this error code is sent by the Transport Converter if it receives a Connect TLV enclosing a multicast, broadcast, or loopback IP address.

  To ease troubleshooting, the Value field **MUST** echo the received message using the format depicted in Figure 23. This format allows keeping the original alignment of the message that triggered the error.

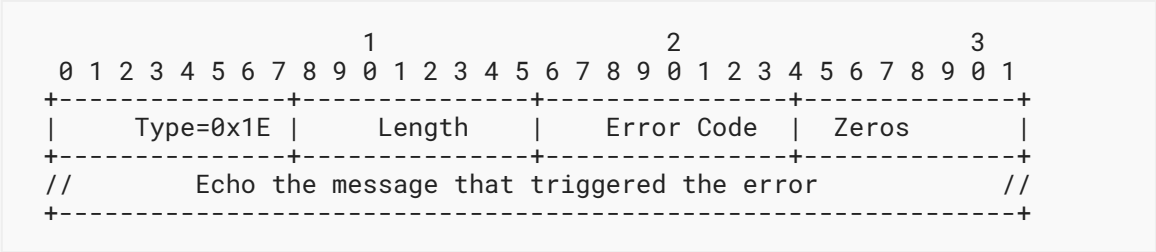

Figure 23: Error TLV to Ease Message Correlation

Unsupported Message (2):
: This error code is sent to indicate that a message type received from a Client is not supported.

  To ease troubleshooting, the Value field **MUST** echo the received message using the format shown in Figure 23.

Missing Cookie (3):
: If a Transport Converter requires the utilization of Cookies to prevent spoofing attacks and a Cookie TLV was not included in the Convert message, the Transport Converter **MUST** return this error to the requesting Client only if it computes a cookie for this Client. The first byte of the Value field **MUST** be set to zero and the remaining bytes of the Error TLV contain the Cookie computed by the Transport Converter for this Client.

  A Client that receives this error code **SHOULD** cache the received Cookie and include it in subsequent Convert messages sent to that Transport Converter.

Not Authorized (32):
: This error code indicates that the Transport Converter refused to create a connection because of a lack of authorization (e.g., administratively prohibited, authorization failure, or invalid Cookie TLV). The Value field **MUST** be set to zero.

  This error code **MUST** be sent by the Transport Converter when a request cannot be successfully processed because the authorization failed.

Unsupported TCP Option (33):
: A TCP option that the Client requested to advertise to the final Server cannot be safely used.

  The Value field is set to the type of the unsupported TCP option. If several unsupported TCP options were specified in the Connect TLV, then the list of unsupported TCP options is returned. The list of unsupported TCP options **MUST** be padded with zeros to end on a 32-bit boundary.

Resource Exceeded (64):
: This error indicates that the Transport Converter does not have enough resources to perform the request.

  This error **MUST** be sent by the Transport Converter when it does not have sufficient resources to handle a new connection. The Transport Converter may indicate in the Value field the suggested delay (in seconds) that the Client **SHOULD** wait before soliciting the Transport Converter for a new proxied connection. A Value of zero corresponds to a default delay of at least 30 seconds.

Network Failure (65):
: This error indicates that the Transport Converter is experiencing a network failure to proxy the request.

  The Transport Converter **MUST** send this error code when it experiences forwarding issues to proxy a connection. The Transport Converter may indicate in the Value field the suggested delay (in seconds) that the Client **SHOULD** wait before soliciting the Transport Converter for a new proxied connection. A Value of zero corresponds to a default delay of at least 30 seconds.

Connection Reset (96):
: This error indicates that the final destination responded with an RST segment. The Value field **MUST** be set to zero.

Destination Unreachable (97):
: This error indicates that an ICMP message indicating a hard error (e.g., destination unreachable, port unreachable, or network unreachable) was received by the Transport Converter. The Value field **MUST** echo the Code field of the received ICMP message.

  As a reminder, TCP implementations are supposed to act on an ICMP error message passed up from the IP layer, directing it to the connection that triggered the error using the demultiplexing information included in the payload of that ICMP message. Such a demultiplexing issue does not apply for handling the "Destination Unreachable" Error TLV because the error is sent in-band. For this reason, the payload of the ICMP message is not echoed in the Destination Unreachable Error TLV.

Table 2 summarizes the different error codes.

Error	Hex	Description
0	0x00	Unsupported Version
1	0x01	Malformed Message
2	0x02	Unsupported Message
3	0x03	Missing Cookie
32	0x20	Not Authorized
33	0x21	Unsupported TCP Option
64	0x40	Resource Exceeded
65	0x41	Network Failure
96	0x60	Connection Reset
97	0x61	Destination Unreachable

Table 2: Convert Error Values

7. Compatibility of Specific TCP Options with the Conversion Service

In this section, we discuss how several deployed Standards Track TCP options can be supported through the Convert Protocol. The other TCP options will be discussed in other documents.

7.1. Base TCP Options

Three TCP options were initially defined in [RFC0793]: End-of-Option List (Kind=0), No-Operation (Kind=1), and Maximum Segment Size (Kind=2). The first two options are mainly used to pad the TCP header. There is no reason for a Client to request a Transport Converter to specifically send these options towards the final destination.

The Maximum Segment Size option (Kind=2) is used by a host to indicate the largest segment that it can receive over each connection. This value is a function of the stack that terminates the TCP connection. There is no reason for a Client to request a Transport Converter to advertise a specific Maximum Segment Size (MSS) value to a remote Server.

A Transport Converter **MUST** ignore options with Kind=0, 1, or 2 if they appear in a Connect TLV. It **MUST NOT** announce them in a Supported TCP Extensions TLV.

7.2. Window Scale (WS)

The Window Scale (WS) option (Kind=3) is defined in [RFC7323]. As for the MSS option, the window scale factor that is used for a connection strongly depends on the TCP stack that handles the connection. When a Transport Converter opens a TCP connection towards a remote Server on behalf of a Client, it **SHOULD** use a WS option with a scaling factor that corresponds to the configuration of its stack. A local configuration **MAY** allow for a WS option in the proxied message to be a function of the scaling factor of the incoming connection.

From a deployment viewpoint, there is no benefit in enabling a Client of a Transport Converter to specifically request the utilization of the WS option (Kind=3) with a specific scaling factor towards a remote Server. For this reason, a Transport Converter **MUST** ignore option Kind=3 if it appears in a Connect TLV. The Transport Converter **MUST NOT** announce a WS option (Kind=3) in a Supported TCP Extensions TLV.

7.3. Selective Acknowledgments

Two distinct TCP options were defined to support Selective Acknowledgment (SACK) in [RFC2018]. This first one, SACK-Permitted (Kind=4), is used to negotiate the utilization of Selective Acknowledgments during the three-way handshake. The second one, SACK (Kind=5), carries the Selective Acknowledgments inside regular segments.

The SACK-Permitted option (Kind=4) **MAY** be advertised by a Transport Converter in the Supported TCP Extensions TLV. Clients connected to this Transport Converter **MAY** include the SACK-Permitted option in the Connect TLV.

The SACK option (Kind=5) cannot be used during the three-way handshake. For this reason, a Transport Converter **MUST** ignore option Kind=5 if it appears in a Connect TLV. It **MUST NOT** announce it in a TCP Supported Extensions TLV.

7.4. Timestamp

The Timestamp option [RFC7323] can be used during the three-way handshake to negotiate the utilization of timestamps during the TCP connection. It is notably used to improve round-trip-time estimations and to provide Protection Against Wrapped Sequences (PAWS). As for the WS option, the timestamps are a property of a connection and there is limited benefit in enabling a Client to request a Transport Converter to use the timestamp option when establishing a connection to a remote Server. Furthermore, the timestamps that are used by TCP stacks are specific to each stack and there is no benefit in enabling a Client to specify the timestamp value that a Transport Converter could use to establish a connection to a remote Server.

A Transport Converter **MAY** advertise the Timestamp option (Kind=8) in the TCP Supported Extensions TLV. The Clients connected to this Transport Converter **MAY** include the Timestamp option in the Connect TLV but without any timestamp.

7.5. Multipath TCP

The Multipath TCP options are defined in [RFC8684], which defines one variable length TCP option (Kind=30) that includes a sub-type field to support several Multipath TCP options. There are several operational use cases where Clients would like to use Multipath TCP through a Transport Converter [IETFJ16]. However, none of these use cases require the Client to specify the content of the Multipath TCP option that the Transport Converter should send to a remote Server.

A Transport Converter that supports Multipath TCP conversion service **MUST** advertise the Multipath TCP option (Kind=30) in the Supported TCP Extensions TLV. Clients serviced by this Transport Converter may include the Multipath TCP option in the Connect TLV but without any content.

7.6. TCP Fast Open

The TCP Fast Open Cookie option (Kind=34) is defined in [RFC7413]. There are two different usages of this option that need to be supported by Transport Converters. The first utilization of the TCP Fast Open Cookie option is to request a cookie from the Server. In this case, the option is sent with an empty cookie by the Client, and the Server returns the cookie. The second utilization of the TCP Fast Open Cookie option is to send a cookie to the Server. In this case, the option contains a cookie.

A Transport Converter **MAY** advertise the TCP Fast Open Cookie option (Kind=34) in the Supported TCP Extensions TLV. If a Transport Converter has advertised the support for TCP Fast Open in its Supported TCP Extensions TLV, it needs to be able to process two types of Connect TLV.

If such a Transport Converter receives a Connect TLV with the TCP Fast Open Cookie option that does not contain a cookie, it **MUST** add an empty TCP Fast Open Cookie option in the SYN sent to the remote Server. If the remote Server supports TFO, it responds with a SYN-ACK according to the procedure in Section 4.1.2 of [RFC7413]. This SYN-ACK may contain a Fast Open option with a cookie. Upon receipt of the SYN-ACK by the Converter, it relays the Fast Open option with the cookie to the Client.

If such a Transport Converter receives a Connect TLV with the TCP Fast Open Cookie option that contains a cookie, it **MUST** copy the TCP Fast Open Cookie option in the SYN sent to the remote Server.

7.7. TCP-AO

The TCP Authentication Option (TCP-AO) [RFC5925] provides a technique to authenticate all the packets exchanged over a TCP connection. Given the nature of this extension, it is unlikely that the applications that require their packets to be authenticated end to end would want their connections to pass through a converter. For this reason, we do not recommend the support of

the TCP-AO by Transport Converters. The only use cases where it could make sense to combine TCP-AO and the solution in this document are those where the TCP-AO-NAT extension [RFC6978] is in use.

A Transport Converter **MUST NOT** advertise the TCP-AO (Kind=29) in the Supported TCP Extensions TLV. If a Transport Converter receives a Connect TLV that contains the TCP-AO, it **MUST** reject the establishment of the connection with error code set to "Unsupported TCP Option", except if the TCP-AO-NAT option is used. Nevertheless, given that TCP-AO-NAT is Experimental, its usage is not currently defined and must be specified by some other document before it can be used.

8. Interactions with Middleboxes

The Convert Protocol is designed to be used in networks that do not contain middleboxes that interfere with TCP. Under such conditions, it is assumed that the network provider ensures that all involved on-path nodes are not breaking TCP signals (e.g., strip TCP options, discard some SYNs, etc.).

Nevertheless, and in order to allow for a robust service, this section describes 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 focus the middleboxes that modify the payload since the Convert Protocol places its messages at the beginning of the bytestream.

Consider a middlebox that removes the SYN payload. The Client can detect this problem by looking at the acknowledgment number field of the SYN+ACK if returned by the Transport Converter. The Client **MUST** stop to use this Transport Converter given the middlebox interference.

Consider now a middlebox that drops SYN/ACKs with a payload. The Client won't be able to establish a connection via the Transport Converter. The case of a middlebox that removes the payload of SYN+ACKs or from the packet that follows the SYN+ACK (but not the payload of SYN) can be detected by a Client. This is hinted by the absence of a valid Convert message in the response.

As explained in [RFC7413], some Carrier Grade NATs (CGNs) can affect the operation of TFO if they assign different IP addresses to the same end host. Such CGNs could affect the operation of the cookie validation used by the Convert Protocol. As a reminder, CGNs that are enabled on the path between a Client and a Transport Converter must adhere to the address preservation defined in [RFC6888]. See also the discussion in Section 7.1 of [RFC7413].

9. Security Considerations

An implementation **MUST** check that the Convert TLVs are properly framed within the boundary indicated by the Total Length in the fixed header (Section 6.1).

Additional security considerations are discussed in the following subsections.

9.1. Privacy & Ingress Filtering

The Transport Converter may have access to privacy-related information (e.g., subscriber credentials). The Transport Converter is designed to not leak such sensitive information outside a local domain.

Given its function and location in the network, a Transport Converter is in a position to observe all packets that it processes, to include payloads and metadata, and has the ability to profile and conduct some traffic analysis of user behavior. The Transport Converter **MUST** be as protected as a core IP router (e.g., Section 10 of [RFC1812]).

Furthermore, ingress filtering policies **MUST** be enforced at the network boundaries [RFC2827].

This document assumes that all network attachments are managed by the same administrative entity. Therefore, enforcing anti-spoofing filters at these networks is a guard that hosts are not sending traffic with spoofed source IP addresses.

9.2. Authentication and Authorization Considerations

The Convert Protocol is **RECOMMENDED** for use in a managed network where end hosts can be securely identified by their IP address. If such control is not exerted and there is a more open network environment, a strong mutual authentication scheme **MUST** be defined to use the Convert Protocol.

One possibility for mutual authentication is to use TLS to perform mutual authentication between the Client and the Converter. That is, use TLS when a Client retrieves a Cookie from the Converter and rely on certificate-based, pre-shared key-based [RFC4279], or raw public key-based Client authentication [RFC7250] to secure this connection. If the authentication succeeds, the Converter returns a cookie to the Client. Subsequent Connect messages will be authorized as a function of the content of the Cookie TLV. An attacker from within the network between a Client and a Transport Converter may intercept the Cookie and use it to be granted access to the conversion service. Such an attack is only possible if the attacker spoofs the IP address of the Client and the network does not filter packets with source-spoofed IP addresses.

The operator that manages the various network attachments (including the Transport Converters) has various options for enforcing authentication and authorization policies. For example, a non-exhaustive list of methods to achieve authorization is provided hereafter:

- The network provider may enforce a policy based on the International Mobile Subscriber Identity (IMSI) to verify that a user is allowed to benefit from the TCP converter service. If that authorization fails, the Packet Data Protocol (PDP) context/bearer will not be mounted. This method does not require any interaction with the Transport Converter for authorization matters.
- The network provider may enforce a policy based upon Access Control Lists (ACLs), e.g., at a Broadband Network Gateway (BNG) to control the hosts that are authorized to communicate with a Transport Converter. These ACLs may be installed as a result of RADIUS exchanges,

e.g., [TCPM-CONVERTER]. This method does not require any interaction with the Transport Converter for authorization matters.

- A device that embeds a Transport Converter may also host a RADIUS Client that will solicit a AAA Server to check whether or not connections received from a given source IP address are authorized [TCPM-CONVERTER].

A first safeguard against the misuse of Transport Converter resources by illegitimate users (e.g., users with access networks that are not managed by the same provider that operates the Transport Converter) is the Transport Converter to reject Convert connections received in the external realm. Only Convert connections received in the internal realm of a Transport Converter will be accepted.

In deployments where network-assisted connections are not allowed between hosts of a domain (i.e., hairpinning), the Converter may be instructed to discard such connections. Hairpinned connections are thus rejected by the Transport Converter by returning an Error TLV set to "Not Authorized". Otherwise, absent explicit configuration, hairpinning is enabled by the Converter (see Figure 24).

```
           <===Network Provider===>

    +----+ from X1:x1 to X2':x2'   +-----+ X1':x1'
    | C1 |>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>--+---
    +----+                         |  v  |
                                   |  v  |
                                   |  v  |
                                   |  v  |
    +----+ from X1':x1' to X2:x2   |  v  | X2':x2'
    | C2 |<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<--+---
    +----+                         +-----+
                                  Converter

    Note: X2':x2' may be equal to
          X2:x2
```

Figure 24: Hairpinning Example

9.3. Denial of Service

Another possible risk is 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 attacks, the Transport Converter **SHOULD** rate-limit the number of pending requests for a given Client. It **SHOULD** also avoid sending SYNs that are significantly longer than the SYN received from the Client, to remote Servers. Finally, the Transport Converter **SHOULD** only retransmit a SYN to a Server after having received a retransmitted SYN from the corresponding Client. Means to protect against SYN flooding attacks should also be enabled (e.g., Section 3 of [RFC4987]).

Attacks from within the network between a Client and a Transport Converter (including attacks that change the protocol version) are yet another threat. Means to ensure that illegitimate nodes cannot connect to a network should be implemented.

9.4. Traffic Theft

Traffic theft is a risk if an illegitimate Converter is inserted in the path. Indeed, inserting an illegitimate Converter in the forwarding path allows traffic interception and can therefore provide access to sensitive data issued by or destined to a host. Converter discovery and configuration are out of scope of this document.

9.5. Logging

If the Converter is configured to behave in the address-sharing mode (Section 4.4.2), the logging recommendations discussed in Section 4 of [RFC6888] need to be considered. Security-related issues encountered in address-sharing environments are documented in Section 13 of [RFC6269].

10. IANA Considerations

10.1. Convert Service Name

IANA has assigned a service name for the Convert Protocol from the "Service Name and Transport Protocol Port Number Registry" available at <https://www.iana.org/assignments/service-names-port-numbers>.

Service Name:	convert
Port Number:	N/A
Transport Protocol(s):	TCP
Description:	0-RTT TCP Convert Protocol
Assignee:	IESG <iesg@ietf.org>
Contact:	IETF Chair <chair@ietf.org>
Reference:	RFC 8803

Clients may use this service name to feed the procedure defined in [RFC2782] to discover the IP address(es) and the port number used by the Transport Converters of a domain.

10.2. The Convert Protocol (Convert) Parameters

IANA has created a new "TCP Convert Protocol (Convert) Parameters" registry.

The following subsections detail new registries within the "Convert Protocol (Convert) Parameters" registry.

The designated expert is expected to ascertain the existence of suitable documentation as described in Section 4.6 of [RFC8126] and to verify that the document is permanently and publicly available. The designated expert is also expected to check the clarity of purpose and use of the requested code points.

Also, criteria that should be applied by the designated experts includes determining whether the proposed registration duplicates existing functionality, whether it is likely to be of general applicability or useful only for private use, and whether the registration description is clear. All requests should be directed to the review mailing list. For both the "Convert TLVs" and "Convert Errors" subregistries, IANA must only accept registry updates in the 128-191 range from the designated experts. It is suggested that multiple designated experts be appointed. In cases where a registration decision could be perceived as creating a conflict of interest for a particular expert, that expert should defer to the judgment of the other experts.

10.2.1. Convert Versions

IANA has created the "Convert Versions" subregistry. New values are assigned via IETF Review (Section 4.8 of [RFC8126]).

The initial values of the registry are as follows:

Version	Description	Reference
0	Reserved	RFC 8803
1	Assigned	RFC 8803

Table 3: Current Convert Versions

10.2.2. Convert TLVs

IANA has created the "Convert TLVs" subregistry. The procedures for assigning values from this registry are as follows:

1-127: IETF Review

128-191: Specification Required

192-255: Private Use

The initial values of the registry are as follows:

Code	Name	Reference
0	Reserved	RFC 8803
1	Info TLV	RFC 8803
10	Connect TLV	RFC 8803
20	Extended TCP Header TLV	RFC 8803
21	Supported TCP Extension TLV	RFC 8803
22	Cookie TLV	RFC 8803

Code	Name	Reference
30	Error TLV	RFC 8803

Table 4: Initial Convert TLVs

10.2.3. Convert Error Messages

IANA has created the "Convert Errors" subregistry. Codes in this registry are assigned as a function of the error type. Four types are defined; the following ranges are reserved for each of these types:

0-31: Message validation and processing errors

32-63: Client-side errors

64-95: Transport Converter-side errors

96-127: Errors caused by destination Server

The procedures for assigning values from this subregistry are as follows:

0-127: IETF Review

128-191: Specification Required

192-255: Private Use

The initial values of the registry are as follows:

Error	Description	Reference
0	Unsupported Version	RFC 8803
1	Malformed Message	RFC 8803
2	Unsupported Message	RFC 8803
3	Missing Cookie	RFC 8803
32	Not Authorized	RFC 8803
33	Unsupported TCP Option	RFC 8803
64	Resource Exceeded	RFC 8803
65	Network Failure	RFC 8803
96	Connection Reset	RFC 8803

Error	Description	Reference
97	Destination Unreachable	RFC 8803

Table 5: Initial Convert Error Codes

11. References

11.1. Normative References

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

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, DOI 10.17487/RFC2018, October 1996, <https://www.rfc-editor.org/info/rfc2018>.

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

[RFC2827] Ferguson, P. and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", BCP 38, RFC 2827, DOI 10.17487/RFC2827, May 2000, <https://www.rfc-editor.org/info/rfc2827>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/info/rfc4291>.

[RFC4787] Audet, F., Ed. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, DOI 10.17487/RFC4787, January 2007, <https://www.rfc-editor.org/info/rfc4787>.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <https://www.rfc-editor.org/info/rfc4987>.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <https://www.rfc-editor.org/info/rfc5925>.

[RFC6888] Perreault, S., Ed., Yamagata, I., Miyakawa, S., Nakagawa, A., and H. Ashida, "Common Requirements for Carrier-Grade NATs (CGNs)", BCP 127, RFC 6888, DOI 10.17487/RFC6888, April 2013, <https://www.rfc-editor.org/info/rfc6888>.

[RFC6890] Cotton, M., Vegoda, L., Bonica, R., Ed., and B. Haberman, "Special-Purpose IP Address Registries", BCP 153, RFC 6890, DOI 10.17487/RFC6890, April 2013, <https://www.rfc-editor.org/info/rfc6890>.

[RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, DOI 10.17487/RFC7323, September 2014, <https://www.rfc-editor.org/info/rfc7323>.

[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>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8684] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 8684, DOI 10.17487/RFC8684, March 2020, <https://www.rfc-editor.org/info/rfc8684>.

11.2. Informative References

[ANRW17] Trammell, B., Kuehlewind, M., De Vaere, P., Learmonth, I., and G. Fairhurst, "Tracking transport-layer evolution with PATHspider", Applied Networking Research Workshop 2017 (ANRW17), July 2017.

[DHC-CONVERTER] Boucadair, M., Jacquenet, C., and T. Reddy.K, "DHCP Options for 0-RTT TCP Converters", Work in Progress, Internet-Draft, draft-boucadair-tcpm-dhc-converter-03, 7 October 2019, <https://tools.ietf.org/html/draft-boucadair-tcpm-dhc-converter-03>.

[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.

[HOT-MIDDLEBOX13] Detal, G., Paasch, C., and O. Bonaventure, "Multipath in the Middle (Box)", HotMiddlebox'13, DOI 10.1145/2535828.2535829, December 2013, <https://inl.info.ucl.ac.be/publications/multipath-middlebox>.

[IANA-CONVERT] IANA, "TCP Convert Protocol (Convert) Parameters", <https://www.iana.org/assignments/tcp-convert-protocol-parameters/tcp-convert-protocol-parameters.xhtml>.

[IETFJ16] Bonaventure, O. and S. Seo, "Multipath TCP Deployments", IETF Journal, Vol. 12, Issue 2, November 2016.

[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 , DOI 10.1145/2068816.2068834, November 2011, <https://doi.org/10.1145/2068816.2068834>.

[INTAREA-SOCKS] Olteanu, V. and D. Niculescu, "SOCKS Protocol Version 6", Work in Progress, Internet-Draft, draft-olteanu-intarea-socks-6-10, 13 July 2020, <https://tools.ietf.org/html/draft-olteanu-intarea-socks-6-10>.

[LOW-LATENCY] Arkko, J. and J. Tantsura, "Low Latency Applications and the Internet Architecture", Work in Progress, Internet-Draft, draft-arkko-arch-low-latency-02, 30 October 2017, <https://tools.ietf.org/html/draft-arkko-arch-low-latency-02>.

[MPTCP-PLAIN] Boucadair, M., Jacquenet, C., Bonaventure, O., Behaghel, D., Secci, 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", Work in Progress, Internet-Draft, draft-boucadair-mptcp-plain-mode-10, March 2017, <https://tools.ietf.org/html/draft-boucadair-mptcp-plain-mode-10>.

[MPTCP-TRANSPARENT] Peirens, B., Detal, G., Barre, S., and O. Bonaventure, "Link bonding with transparent Multipath TCP", Work in Progress, Internet-Draft, draft-peirens-mptcp-transparent-00, 8 July 2016, <https://tools.ietf.org/html/draft-peirens-mptcp-transparent-00>.

[RFC1812] Baker, F., Ed., "Requirements for IP Version 4 Routers", RFC 1812, DOI 10.17487/RFC1812, June 1995, <https://www.rfc-editor.org/info/rfc1812>.

[RFC1919] Chatel, M., "Classical versus Transparent IP Proxies", RFC 1919, DOI 10.17487/RFC1919, March 1996, <https://www.rfc-editor.org/info/rfc1919>.

[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, <https://www.rfc-editor.org/info/rfc1928>.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, <https://www.rfc-editor.org/info/rfc2782>.

[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, <https://www.rfc-editor.org/info/rfc3135>.

[RFC4279] Eronen, P., Ed. and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, DOI 10.17487/RFC4279, December 2005, <https://www.rfc-editor.org/info/rfc4279>.

[RFC5461] Gont, F., "TCP's Reaction to Soft Errors", RFC 5461, DOI 10.17487/RFC5461, February 2009, <https://www.rfc-editor.org/info/rfc5461>.

[RFC6269] Ford, M., Ed., Boucadair, M., Durand, A., Levis, P., and P. Roberts, "Issues with IP Address Sharing", RFC 6269, DOI 10.17487/RFC6269, June 2011, <https://www.rfc-editor.org/info/rfc6269>.

[RFC6296] Wasserman, M. and F. Baker, "IPv6-to-IPv6 Network Prefix Translation", RFC 6296, DOI 10.17487/RFC6296, June 2011, <https://www.rfc-editor.org/info/rfc6296>.

[RFC6731] Savolainen, T., Kato, J., and T. Lemon, "Improved Recursive DNS Server Selection for Multi-Interfaced Nodes", RFC 6731, DOI 10.17487/RFC6731, December 2012, <https://www.rfc-editor.org/info/rfc6731>.

[RFC6887] Wing, D., Ed., Cheshire, S., Boucadair, M., Penno, R., and P. Selkirk, "Port Control Protocol (PCP)", RFC 6887, DOI 10.17487/RFC6887, April 2013, <https://www.rfc-editor.org/info/rfc6887>.

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

[RFC6978] Touch, J., "A TCP Authentication Option Extension for NAT Traversal", RFC 6978, DOI 10.17487/RFC6978, July 2013, <https://www.rfc-editor.org/info/rfc6978>.

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <https://www.rfc-editor.org/info/rfc7250>.

[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, <https://www.rfc-editor.org/info/rfc7414>.

[RFC8041] Bonaventure, O., Paasch, C., and G. Detal, "Use Cases and Operational Experience with Multipath TCP", RFC 8041, DOI 10.17487/RFC8041, January 2017, <https://www.rfc-editor.org/info/rfc8041>.

[RFC8305] Schinazi, D. and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency", RFC 8305, DOI 10.17487/RFC8305, December 2017, <https://www.rfc-editor.org/info/rfc8305>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8548] Bittau, A., Giffin, D., Handley, M., Mazieres, D., Slack, Q., and E. Smith, "Cryptographic Protection of TCP Streams (tcpcrypt)", RFC 8548, DOI 10.17487/RFC8548, May 2019, <https://www.rfc-editor.org/info/rfc8548>.

[TCPM-CONVERTER] Boucadair, M. and C. Jacquenet, "RADIUS Extensions for 0-RTT TCP Converters", Work in Progress, Internet-Draft, draft-boucadair-opsawg-tcpm-converter-01, 28 February 2020, <https://tools.ietf.org/html/draft-boucadair-opsawg-tcpm-converter-01>.

[TS23501] 3GPP (3rd Generation Partnership Project), "Technical Specification Group Services and System Aspects; System architecture for the 5G System; Stage 2 (Release 16)", 2019, <https://www.3gpp.org/ftp/Specs/archive/23_series/23.501/>.

Appendix A. Example Socket API Changes to Support the 0-RTT TCP Convert Protocol

A.1. Active Open (Client Side)

On the Client side, the support of the 0-RTT Converter protocol does not require any other changes than those identified in Appendix A of [RFC7413]. Those modifications are already supported by multiple TCP stacks.

As an example, on Linux, a Client can send the 0-RTT Convert message inside a SYN by using sendto with the MSG_FASTOPEN flag as shown in the example below:

```
s = socket(AF_INET, SOCK_STREAM, 0);

sendto(s, buffer, buffer_len, MSG_FASTOPEN,
               (struct sockaddr *) &server_addr, addr_len);
```

The Client side of the Linux TFO can be used in two different modes depending on the host configuration (sysctl tcp_fastopen variable):

0x1: (client) enables sending data in the opening SYN on the Client.

0x4: (client) enables sending data in the opening SYN regardless of cookie availability and without a cookie option.

By setting this configuration variable to 0x5, a Linux Client using the above code would send data inside the SYN without using a TFO option.

A.2. Passive Open (Converter Side)

The Converter needs to enable the reception of data inside the SYN independently of the utilization of the TFO option. This implies that the Transport Converter application cannot rely on the Fast Open Cookies to validate the reachability of the IP address that sent the SYN. It must rely on other techniques, such as the Cookie TLV described in this document, to verify this reachability.

[RFC7413] suggested the utilization of a TCP_FASTOPEN socket option to enable the reception of SYNs containing data. Later, Appendix A of [RFC7413] mentioned:

> Traditionally, accept() returns only after a socket is connected. But, for a Fast Open connection, accept() returns upon receiving a SYN with a valid Fast Open cookie and data, and the data is available to be read through, e.g., recvmsg(), read().

To support the 0-RTT TCP Convert Protocol, this behavior should be modified as follows:

> Traditionally, accept() returns only after a socket is connected. But, for a Fast Open connection, accept() returns upon receiving a SYN with data, and the data is available to be read through, e.g., recvmsg(), read(). The application that receives such SYNs with data must be able to validate the reachability of the source of the SYN and also deal with replayed SYNs.

The Linux Server side can be configured with the following sysctls:

0x2: (server) enables the Server support, i.e., allowing data in a SYN packet to be accepted and passed to the application before a 3-way handshake finishes.

0x200: (server) accepts data-in-SYN w/o any cookie option present.

However, this configuration is system wide. This is convenient for typical Transport Converter deployments where no other applications relying on TFO are collocated on the same device.

Recently, the TCP_FASTOPEN_NO_COOKIE socket option has been added to provide the same behavior on a per-socket basis. This enables a single host to support both Servers that require the Fast Open Cookie and Servers that do not use it.

Acknowledgments

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.

We would like to thank Raphael Bauduin, Stefano Secci, Anandatirtha Nandugudi, and Gregory Vander Schueren for their help in preparing this document. Nandini Ganesh provided valuable feedback about the handling of TFO and the error codes. Yuchung Cheng and Praveen Balasubramanian helped to clarify the discussion on supplying data in SYNs. Phil Eardley and Michael Scharf helped to clarify different parts of the text. Thanks to Éric Vyncke, Roman Danyliw, Benjamin Kaduk, and Alexey Melnikov for the IESG review, and Christian Huitema for the Security Directorate review.

Many thanks to Mirja Kühlewind for the detailed AD review.

This document builds upon earlier documents that proposed various forms of Multipath TCP proxies: [MPTCP-PLAIN], [MPTCP-TRANSPARENT], and [HOT-MIDDLEBOX13].

From [MPTCP-PLAIN]:

Many thanks to Chi Dung Phung, Mingui Zhang, Rao Shoaib, Yoshifumi Nishida, and Christoph Paasch for their valuable comments.

Thanks to Ian Farrer, Mikael Abrahamsson, Alan Ford, Dan Wing, and Sri Gundavelli for the fruitful discussions at IETF 95 (Buenos Aires).

Special thanks to Pierrick Seite, Yannick Le Goff, Fred Klamm, and Xavier Grall for their input.

Thanks also to Olaf Schleusing, Martin Gysi, Thomas Zasowski, Andreas Burkhard, Silka Simmen, Sandro Berger, Michael Melloul, Jean-Yves Flahaut, Adrien Desportes, Gregory Detal, Benjamin David, Arun Srinivasan, and Raghavendra Mallya for their input.

Contributors

Bart Peirens contributed to an early draft version of this document.

As noted above, this document builds on two previous documents.

The authors of [MPTCP-PLAIN] were:

- Mohamed Boucadair
- Christian Jacquenet
- Olivier Bonaventure
- Denis Behaghel
- Stefano Secci
- Wim Henderickx
- Robert Skog
- Suresh Vinapamula
- SungHoon Seo
- Wouter Cloetens
- Ullrich Meyer
- Luis M. Contreras
- Bart Peirens

The authors of [MPTCP-TRANSPARENT] were:

- Bart Peirens
- Gregory Detal
- Sebastien Barre

- Olivier Bonaventure

Authors' Addresses

Olivier Bonaventure (EDITOR)
Tessares
Avenue Jean Monnet 1
B-1348 Louvain-la-Neuve
Belgium
Email: Olivier.Bonaventure@tessares.net

Mohamed Boucadair (EDITOR)
Orange
Clos Courtel
35000 Rennes
France
Email: mohamed.boucadair@orange.com

Sri Gundavelli
Cisco
170 West Tasman Drive
San Jose, CA 95134
United States of America
Email: sgundave@cisco.com

SungHoon Seo
Korea Telecom
151 Taebong-ro
Seocho-gu, Seoul, 06763
Republic of Korea
Email: sh.seo@kt.com

Benjamin Hesmans
Tessares
Avenue Jean Monnet 1
B-1348 Louvain-la-Neuve
Belgium
Email: Benjamin.Hesmans@tessares.net