TCPM Working Group   O. Bonaventure, Ed.
Internet-Draft   Tessares
Intended status: Experimental   M. Boucadair, Ed.
Expires: April 6, 2020   Orange
                                                          S. Gundavelli
                                                                   Cisco
                                                                  S. Seo
                                                           Korea Telecom
                                                             B. Hesmans
                                                                Tessares
                                                        October 04, 2019

                         0-RTT TCP Convert Protocol
                       draft-ietf-tcpm-converters-12

Abstract

   This document specifies an application proxy, called Transport
   Converter, to assist the deployment of TCP extensions such as
   Multipath TCP. This proxy is designed to avoid inducing extra delay
   when involved in a network-assisted connection (that is, 0-RTT).

   This specification assumes an explicit model, where the proxy is
   explicitly configured on hosts.

   -- Editorial Note (To be removed by RFC Editor)

   Please update these statements with the RFC number to be assigned to
   this document: [This-RFC]

   Please update TBA statements with the port number to be assigned to
   the 0-RTT TCP Convert Protocol.

Status of This Memo

   This Internet-Draft is submitted in full conformance with the
   provisions of BCP 78 and BCP 79.

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF). Note that other groups may also distribute
   working documents as Internet-Drafts. The list of current Internet-
   Drafts is at https://datatracker.ietf.org/drafts/current/.

   Internet-Drafts are draft documents valid for a maximum of six months
   and may be updated, replaced, or obsoleted by other documents at any
   time. It is inappropriate to use Internet-Drafts as reference
   material or to cite them other than as "work in progress."

   This Internet-Draft will expire on April 6, 2020.

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 to change 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 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
would typically want to benefit from the features of an improved
transport protocol even if the servers have not yet been upgraded and
conversely. 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 [RFC6824] or
TCPINC [RFC8548]. Those extensions provide features that are
interesting for clients such as wireless devices. With Multipath
TCP, those devices could seamlessly use WLAN (Wireless Local Area
Network) 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 the 4G
bonding relying upon Multipath TCP increases the bandwidth, it is as
well crucial to minimize latency for all the way between endhosts
regardless of whether intermediate nodes are inside or outside of the
mobile core. In order to handle URLLC (Ultra Reliable Low Latency
Communication) 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. A Transport
Converter may provide conversion service for one or more TCP
extensions. Which TCP extensions are eligible to the conversion
service is deployment-specific. The conversion service is provided
by means of the 0-RTT TCP Convert Protocol (Convert), that is an

application-layer protocol which uses TCP port number TBA
(Section 8).

The Convert 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. 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 principles drawn in
[[RFC1919](RFC1919)]. In particular, a Transport Converter achieves the
following:

o Listen for client sessions;

o Receive from a client the address of the final target server;

o Setup a session to the final server;

o Relay control messages and data between the client and the server;

o Perform 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 to 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 [I-D.boucadair-tcpm-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 4.2.8) 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. First, Section 3 provides a
   brief explanation of the operation of Transport Converters. Then,
   Section 4 describes the Convert Protocol. Section 5 discusses how
   Transport Converters can be used to support different TCP extensions.
   Section 6 then discusses the interactions with middleboxes, while
   Section 7 focuses on the security considerations.

   Appendix B describes how a TCP stack would need to support the
   protocol described in this document. Appendix C records some
   considerations that impacted the design of the protocol. Appendix E
   provides a comparison with SOCKS proxies that are already used to
   deploy Multipath TCP in some cellular networks (Section 2.2 of
   [RFC8041]).

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.

   The information shown between brackets in the figures refers to
   Convert Protocol messages described in Section 4.

   Only the exchange of control messages is depicted in the figures.

3. Architecture

3.1. Functional Elements

   The Convert Protocol considers three functional elements:

   o Clients;

   o Transport Converters;

   o 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 1). The Transport Converter, thus, maintains
   state that associates one upstream connection to a corresponding
   downstream connection.

   A connection can be initiated from both sides of the Transport
   Converter (Internet-facing interface, customer-facing interface).

```
                              |
                              :
                              |
                  +------------+
   Client <- upstream ->| Transport  |<- downstream -> Server
          connection  | Converter  |   connection
                  +------------+
                              |
    customer-facing interface : Internet-facing interface
                              |
```
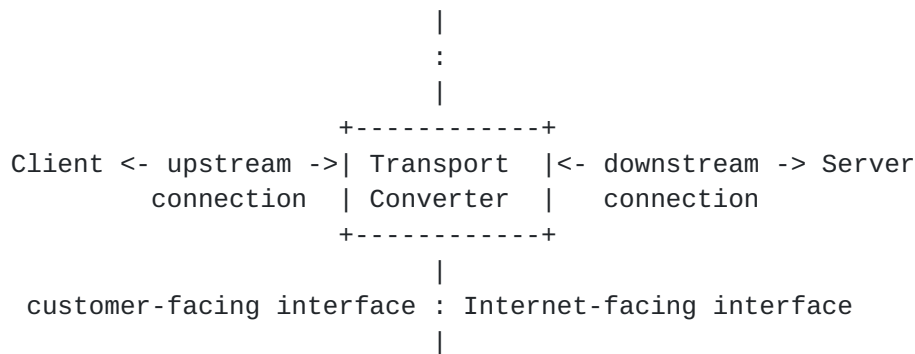
        Figure 1: A Transport Converter Proxies Data between Pairs of TCP
                             Connections

   "Client" refers to a software instance embedded on a host that can
   reach a Transport Converter via its customer-facing interface. The
   "Client" can initiate connections via a Transport Converter (referred
   to as outgoing connections (Section 3.4)). Also, the "Client" can
   accept incoming connections via a Transport Converter (referred to as
   incoming connections (Section 3.5)).

   Transport Converters can be operated by network operators or third
   parties. Nevertheless, this document focuses on the single
   administrative deployment case where the entity offering the
   connectivity service to a client is also the entity which owns and
   operates the Transport Converter.

A Transport Converter can be embedded in a standalone device or be
activated as a service on a router. How such function is enabled is
deployment-specific. A sample deployment is depicted in Figure 2.

```
              +-+     +-+     +-+
   Client -  |R| -- |R| -- |R| - - -  Server
              +-+     +-+     +-+
                       |
                      +-+
                      |R|
                      +-+
                       |
                 +---------+
                 |Transport|
                 |Converter|
                 +---------+
   R: Router
```
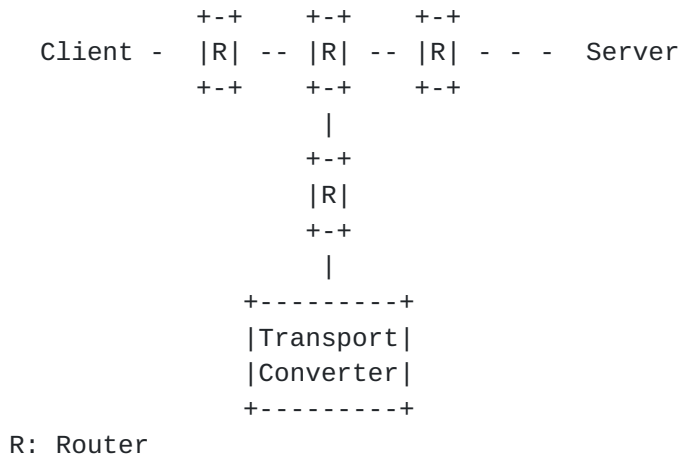
         Figure 2: A Transport Converter Can Be Installed Anywhere in the
                                 Network

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 the 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 by servers, in particular.

The architecture does not mandate anything on the Server side.

Similar to address sharing mechanisms, 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.
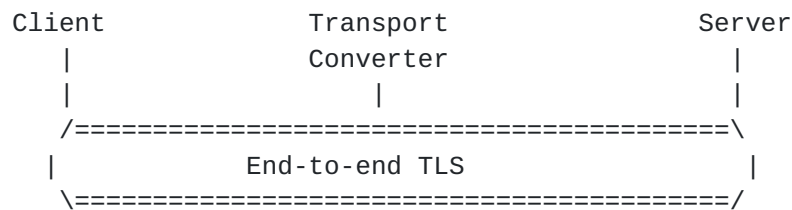
```
     Client                 Transport                    Server
        |                   Converter                       |
        |                       |                           |
      /=========================================\
      |              End-to-end TLS              |
      \=========================================/


        * TLS messages exchanged between the Client
          and the Server are not shown.

            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).

3.2. 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 regular TCP connection with the Server.
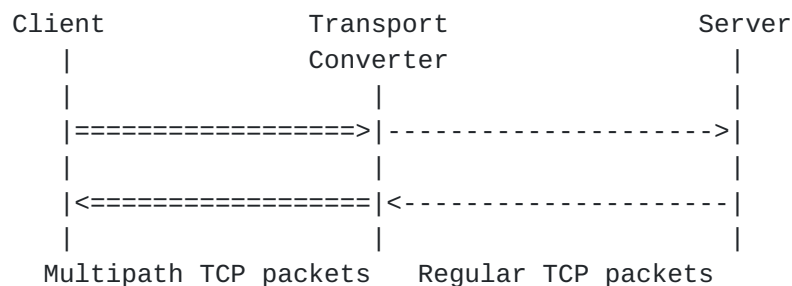
```
     Client                 Transport                    Server
        |                   Converter                       |
        |                       |                           |
        |==================>|-------------------->|
        |                       |                           |
        |<==================|<--------------------|
        |                       |                           |
      Multipath TCP packets   Regular TCP packets

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

The packets belonging to the pair of connections between the Client
and Server passing 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:

o the upstream connection is the one between the Client and the
   Transport Converter.

o the downstream connection is 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. In particular, if the initial SYN message
contains data in its payload (e.g., [RFC7413]), that data MUST be
placed right after the Convert TLVs when generating the SYN.

The Converter associates a lifetime with state entries used to bind
an upstream connection with its downstream connection.

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

```
                        Transport
     Client             Converter                 Server
        |                   |                        |
        |SYN [->Server:port]|          SYN           |
        |------------------>|----------------------->|
        |<-----------------|<-----------------------|
        |    SYN+ACK [ ]    |          SYN+ACK       |
        |       ...         |          ...           |
```
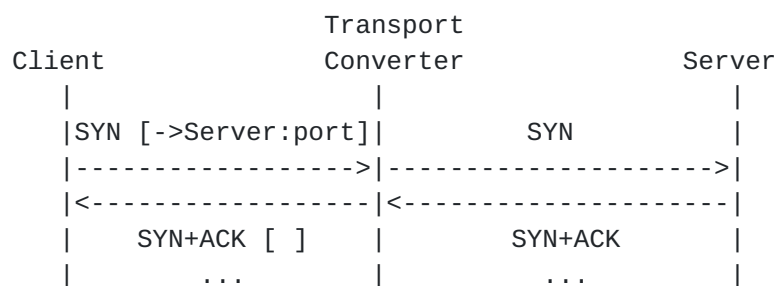
        Figure 5: Establishment of an Outgoing TCP Connection Through a
                        Transport Converter

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 an application on the Client listens to a specific port
(the Client hosts an application server, typically). 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. If the check fails, the
packet is silently ignored by the Converter. If the check is
successful, the Converter inserts the source IP address and source
port number in the SYN packet, rewrites the source IP address to one
of its IP addresses and, eventually (i.e., only when the Converter is
configured in an address sharing mode), the destination IP address
and port number in accordance with any information stored locally.
That SYN is then forwarded to the next hop. A transport session
entry is created by the Converter for this connection. SYN+ACK and
ACK will be then 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.

```
                        Transport              Remote
     Client             Converter            Host (RH)
        |                   |                    |
        |SYN [<-RH IP@:port]|         SYN        |
        |<------------------|<-------------------|
        |------------------>|------------------->|
        |    SYN+ACK [ ]    |        SYN+ACK     |
        |        ...        |          ...       |
```
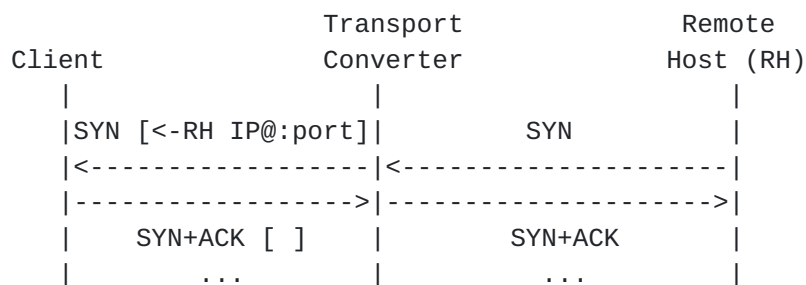
             Figure 6: Establishment of an Incoming TCP Connection Through a
                        Transport Converter

Standard TCP ([RFC0793], Section 3.4) 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 semantic raises two
issues. First, duplicate SYNs can cause problems for some
applications that rely on TCP. 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 extended TCP header.
In particular, this design allows for the use of longer cookies.

If the downstream (or upstream) connection fails for some reason
(excessive retransmissions, reception of an RST segment, etc.), then
the Converter should force the tear-down of the upstream (or
downstream) connection.

The same reasoning applies when the upstream connection ends. In
this case, the Converter should 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.

3.3. Data Processing at the Transport Converter

As mentioned in Section 3.2, 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, discussed in Section 4, 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
```

Where:
   * C and c are the source IP address and source port number
     used by the Client for the usptream 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 the validity lifetime of the entry as assigned
     by the Converter.

       Figure 7: An Example of Transport Session Entry (TCP)

Note that for the Multipath TCP case, the Transport Converter
identifies an MPTCP connection by means, e.g., of the token assigned
to the MPTCP connection. Subflows can be added or deleted during the
lifetime of an MPTCP connection between a Client and a Transport
Converter. The identification of each subflow that belongs to an
MPTCP connection are also part of the transport session entry. An
implementation example of a transport session entry maintained by a
Transport Converter for Multipath TCP connections is shown in
Figure 7. As a reminder, the Convert TLVs are only exchanged during
the establishment of the initial subflow.

```
     token, {(C1,c1), .., (Cn, cn)} <--> (T,t), (S,s), Lifetime
```

Where:
   * Token is a locally unique identifier given to a (upstream)
     multipath connection by the Transport Converter.
   * Ci and ci are the source IP address and source port number
     used by the Client for a subflow of a (upstream) Multipath TCP
     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 the validity lifetime of the entry as assigned
     by the Converter.

   Figure 8: An Example of Transport Session Entry (Multipath TCP)

Clients send packets bound to connections eligible to the conversion
service to the provisioned Transport Converter using TBA as
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 4.
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 Total Length field in the Convert messages

(Section 4.1). 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 Non-SYN (or a secondary subflow for Multipath TCP)
on port number TBA by the Transport Converter from a Client, the
Converter checks if the packet matches an active transport session
entry. If no entry is found, the Transport Converter MUST silently
ignore the packet. If an entry is found, the user data is proxied to
the Server using the information stored in the corresponding
transport session entry. For example,

o 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.

o In reference to Figure 8, the Transport Converter proxies the data
   received from a new subflow of an existing Multipath TCP
   connection (Cn, cn) 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.

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
Appendix D 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.

3.4. Sample Examples of 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 the Server supports Multipath TCP or not.

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

Figure 9 describes the operation of the Transport Converter if the
Server does not support Multipath TCP.

```
                         Transport
    Client               Converter                 Server
        |SYN,                 |                       |
        |MPC [->Server:port]|         SYN, MPC        |
        |------------------>|---------------------->|
        |<-----------------|<--------------------|
        |   SYN+ACK,MPC [.]  |         SYN+ACK          |
        |------------------>|---------------------->|
        |       ACK, MPC       |           ACK           |
        |         ...          |           ...           |
```
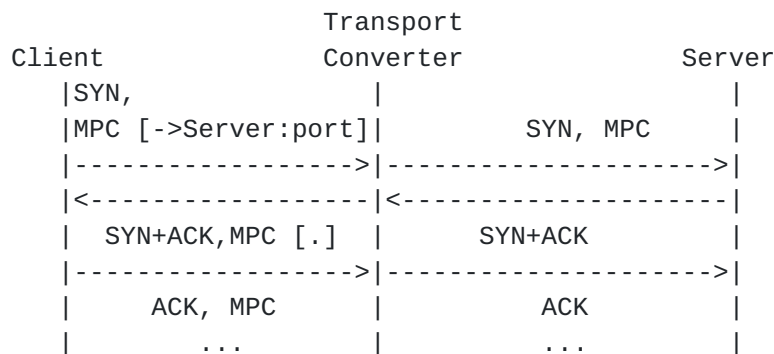
      Figure 9: Establishment of a Multipath TCP Connection Through a
    Transport Converter towards a Server that Does Not Support Multipath
                                   TCP

The Client tries to initiate a Multipath TCP connection by sending a
SYN with the MP_CAPABLE option (MPC in Figure 9). 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 fails for some reason, the Converter
tears down the Multipath TCP connection by transmitting a
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 TCP connection upon receipt of
an RST over a Multipath subflow.

Figure 10 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 natively. This will enable the Client to
bypass the Transport Converter for the subsequent Multipath TCP
connections that it will initiate towards this Server.

```
                          Transport
        Client            Converter                Server
           |SYN,                |                      |
           |MPC [->Server:port]|        SYN, MPC       |
           |------------------>|--------------------->|
           |<------------------|<---------------------|
           |SYN+ACK,            |      SYN+ACK, MPC     |
           |MPC [MPC supported]|                      |
           |------------------>|--------------------->|
           |      ACK, MPC      |        ACK, MPC       |
           |        ...         |          ...          |
```
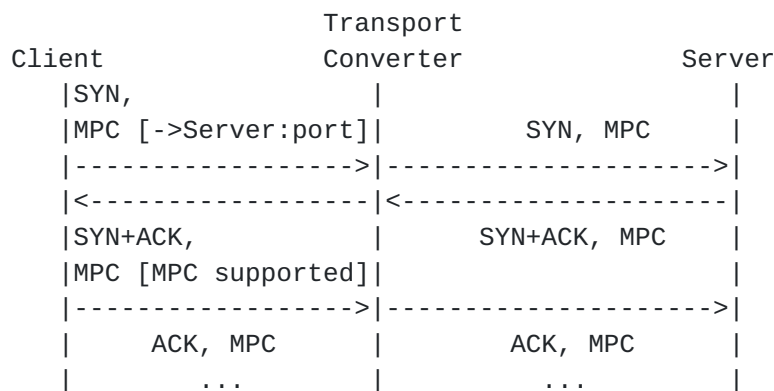
         Figure 10: Establishment of a Multipath TCP Connection Through a
                   Converter Towards an MPTCP-capable Server

**3.5. Sample Example of Incoming Converter-Assisted Multipath TCP
    Connection**

   An example of an incoming Converter-assisted Multipath TCP connection
   is depicted in Figure 11. In order to support incoming connections
   from remote hosts, the Client may use 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 (internal IP address, internal
   port number). The Converter accepts the request by creating a TCP
   mapping (internal IP address, internal port number, external IP
   address, external port number). The external IP address and external
   port number will be then advertised 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, 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 be then 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.

```
                          Transport              Remote
        Client            Converter               Host
           |                  |                    |
           |<-------------------|<-------------------|
           |SYN,               |         SYN        |
           |MPC[Remote Host:port]|                   |
           |------------------->|------------------->|
           |       SYN+ACK, MPC  |      SYN+ACK       |
           |<-------------------|<-------------------|
           |         ACK, MPC    |         ACK        |
           |        ...         |         ...        |
```
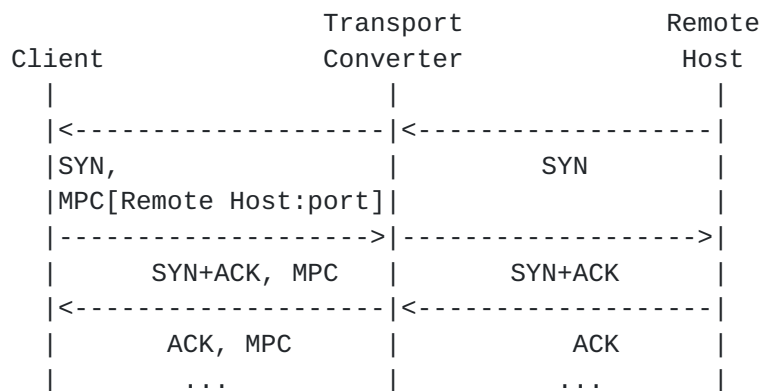
        Figure 11: Establishment of an Incoming Multipath TCP Connection
                       through a Transport Converter

   It is out of scope of this document to define specific Convert TLVs
   to manage incoming connections. These TLVs can be defined in a
   separate document.

4. The Convert Protocol (Convert)

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

   By default, the Transport Converter listens on TCP port number TBA
   for Convert messages from Clients.

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

   Convert messages MUST be included as the first bytes of the
   bytestream. All Convert messages starts with a 32 bits long fixed
   header (Section 4.1) followed by one or more Convert TLVs (Type,
   Length, Value) (Section 4.2).

4.1. The Convert Fixed Header

   The Convert Protocol uses a 32 bits long fixed header that is 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 12, as the first four bytes of the
   bytestream.

```
                            1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +---------------+---------------+-------------------------------+
    |   Version     |  Total Length |            Unassigned         |
    +---------------+---------------+-------------------------------+
```
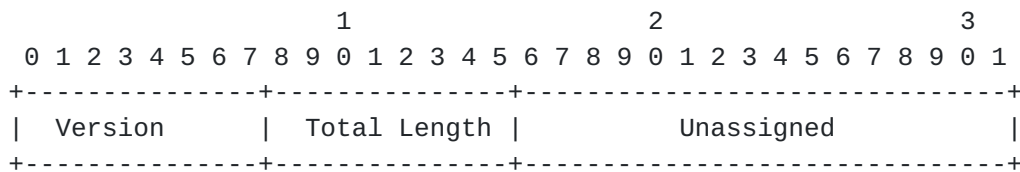
         Figure 12: The Fixed-Sized Header of the Convert Protocol

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

   The Total Length is the number of 32 bits word, including the header,
   of the bytestream that are consumed by the Convert messages. Since
   Total Length is also an 8 bits unsigned integer, those messages
   cannot consume more than 1020 bytes of data. This limits the number
   of bytes that a Transport Converter needs to process. A Total Length
   of zero is invalid and the connection MUST be reset upon reception of
   a header with such total length.

   The Unassigned field MUST be set to zero in this version of the
   protocol. These bits are available for future use [RFC8126].

   Data added by the Convert protocol to the TCP bytestream is
   unambiguously distinguished from payload data by the Total Length
   field in the Convert messages.

4.2. Convert TLVs

4.2.1. Generic Convert TLV Format

   The Convert protocol uses variable length messages that are encoded
   using the generic TLV format depicted in Figure 13.

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

```
                            1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +---------------+---------------+-------------------------------+
    |     Type      |     Length    |            Value  ...         |
    +---------------+---------------+-------------------------------+
    //              ...   (optional) Value                         //
    +---------------------------------------------------------------+
```
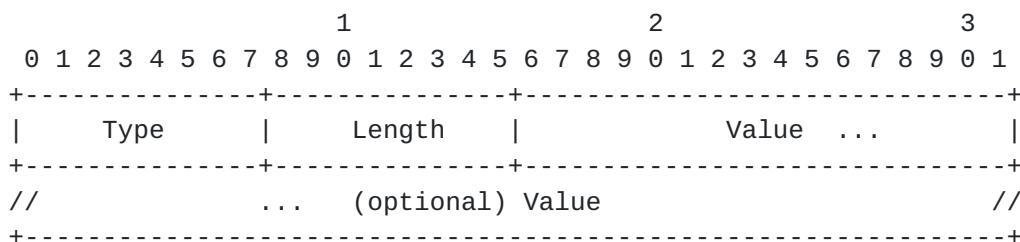
               Figure 13: Convert Generic TLV Format

The Length field covers Type, Length, and Value fields. It is
expressed in units of 32 bits 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 two or more
instances of the same TLV are exchanged over a Convert connection,
the associated TCP connections MUST be closed.

4.2.2. 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                              |
+------+-----+----------+----------------------------------------+
```

Figure 14: The TLVs used by the Convert Protocol

Type 0x0 is a reserved valued. Implementations MUST discard messages
with such TLV.

The Client typically sends in the first connection it established
with a Transport Converter the Info TLV (Section 4.2.3) to learn its
capabilities. Assuming the Client is authorized to invoke the
Transport Converter, the latter replies with the Supported TCP
Extensions TLV (Section 4.2.4).

The Client can request the establishment of connections to servers by
using the Connect TLV (Section 4.2.5). If the connection can be
established with the final server, the Transport Converter replies
with the Extended TCP Header TLV (Section 4.2.6). If not, the
Transport Converter returns an Error TLV (Section 4.2.8) and then
closes the connection.

When an error is encountered an Error TLV with the appropriate error
code MUST be returned by the Transport Converter.

4.2.3. The Info TLV

   The Info TLV (Figure 15) is an optional TLV which 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 4.2.4.

```
                         1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +--------------+--------------+------------------------------+
    |    Type=0x1  |    Length    |             Zero             |
    +--------------+--------------+------------------------------+
```
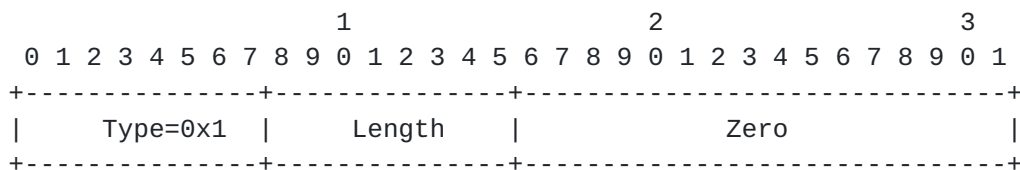
                        Figure 15: The Info TLV

4.2.4. 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 accepts from Clients; these options are
   included by the Transport Converter in the SYN packets that it sends
   to initiate connections.

   Each supported TCP option is encoded with its TCP option Kind listed
   in the "TCP Parameters" registry maintained by IANA.

```
                         1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +--------------+--------------+------------------------------+
    |    Type=0x15 |    Length    |          Unassigned          |
    +--------------+--------------+------------------------------+
    |    Kind #1   |    Kind #2   |             ...              |
    +--------------+--------------+------------------------------+
    /                             ...                             /
    /                                                             /
    +-------------------------------------------------------------+
```
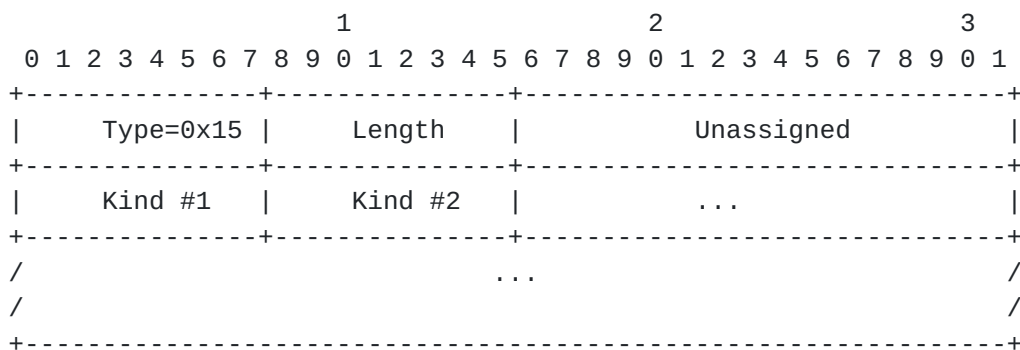
              Figure 16: The Supported TCP Extensions TLV

   TCP option Kinds 0, 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 bits 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 Info TLV.

4.2.5. 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.

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, Remote Peer IP address field MUST NOT include multicast, broadcast, and host loopback addresses [RFC6890]. Connect TLVs witch such messages MUST be discarded by the Transport Converter.

We distinguish two types of Connect TLV based on their length: (1) the base Connect TLV has a length of 20 bytes and contains a remote address and a remote port, (2) the extended Connect TLV spans more than 20 bytes and also includes the optional 'TCP Options' field. This field is used to specify how specific TCP options should be advertised by the Transport Converter to the server.

```
                        1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +---------------+---------------+-------------------------------+
   |    Type=0xA   |    Length     |       Remote Peer Port        |
   +---------------+---------------+-------------------------------+
   |                                                               |
   |          Remote Peer IP Address (128 bits)                    |
   |                                                               |
   |                                                               |
   +---------------------------------------------------------------+
   /                      TCP Options (Variable)                   /
   /                             ...                               /
   +---------------------------------------------------------------+
```

Figure 17: The Connect TLV

The 'TCP Options' field is a variable length field that carries a
list of TCP option fields (Figure 18). 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
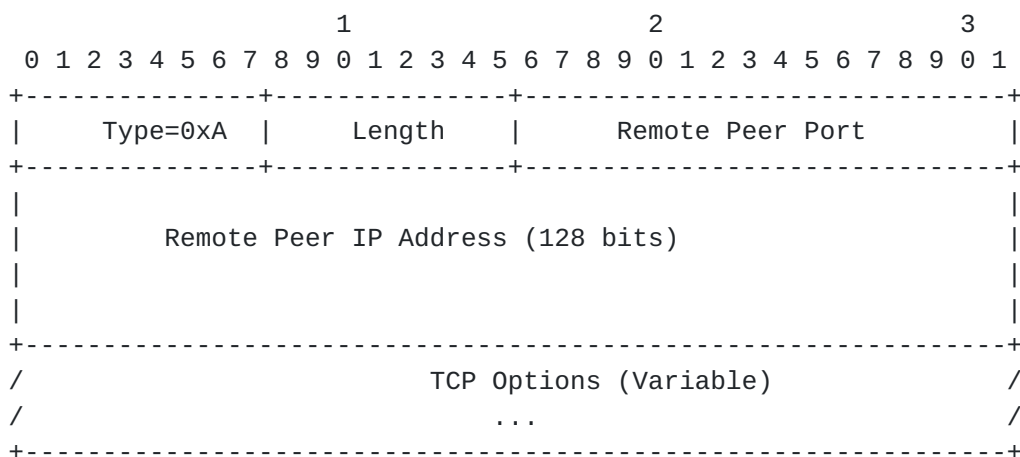length of two indicates the absence of the Value field. The TCP
options field always ends on a 32 bits boundary after being padded
with zeros.

```
                        1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +---------------+---------------+---------------+---------------+
   |  TCPOpt kind  | TCPOpt Length | Value  (opt)  |  ....         |
   +---------------+---------------+---------------+---------------+
   |                             ....                              |
   +---------------------------------------------------------------+
   |                             ...                               |
   +---------------------------------------------------------------+
```
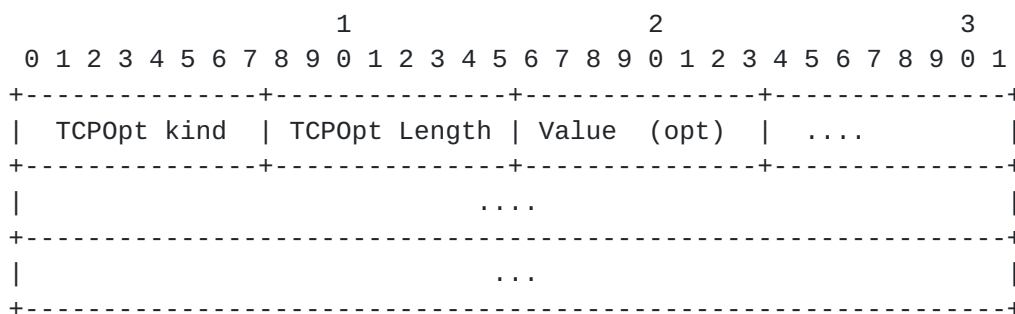
Figure 18: The TCP Options Field

Upon reception of a 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. These are the options that it advertises in the
Supported TCP Extensions TLV.

Upon reception of an extended Connect TLV, 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 the options of the 'TCP Options' sub-field
in the SYN sent to the Server in addition to the TCP options that it
would have used according to its local policies. For the TCP options
that are listed 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 for use in the connection with the destination peer.
This feature is required to support TCP Fast Open.

The Transport Converter may discard a Connect TLV request for various
reasons (e.g., authorization failed, out of resources, invalid
address type). An error message indicating the encountered error is

returned to the requesting Client (Section 4.2.8). In order to
prevent denial-of-service attacks, error messages sent to a Client
SHOULD be rate-limited.

4.2.6. Extended TCP Header TLV

The Extended TCP Header TLV (Figure 19) is used by the Transport
Converter to send to the Client the extended TCP header that was
returned by the Server in the SYN+ACK packet. This TLV is only sent
if the Client sent a Connect TLV to request the establishment of a
connection.

```
                       1                   2                   3
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
  +---------------+---------------+-------------------------------+
  |   Type=0x14   |    Length     |           Unassigned          |
  +---------------+---------------+-------------------------------+
  /                Returned Extended TCP header                   /
  /                            ...                                /
  +---------------------------------------------------------------+
```
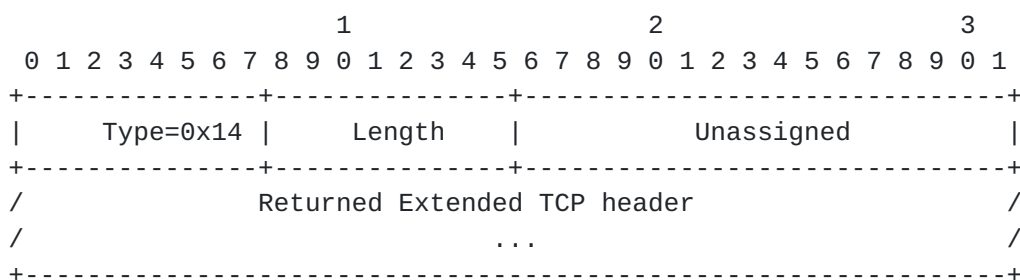
Figure 19: The Extended TCP Header TLV

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

The Unassigned field MUST be set to zero by the sender and ignored by
the receiver. These bits are available for future use [RFC8126].

4.2.7. The Cookie TLV

The Cookie TLV (Figure 20 is an optional TLV which use 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 as follows.

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 should compute a
Cookie bound to this Client address and return a Convert message
containing the fixed header, an Error TLV set to "Missing Cookie" and
the computed Cookie and close the connection. The Client will react
to this error by storing the received Cookie in its cache and attempt
to reestablish a new connection to the Transport Converter that
includes the Cookie TLV.

The format of the Cookie TLV is shown in Figure 20.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+-------------------------------+
|   Type=0x16   |    Length     |             Zero              |
+---------------+---------------+-------------------------------+
/                       Opaque  Cookie                          /
/                          ...                                  /
+---------------------------------------------------------------+
```
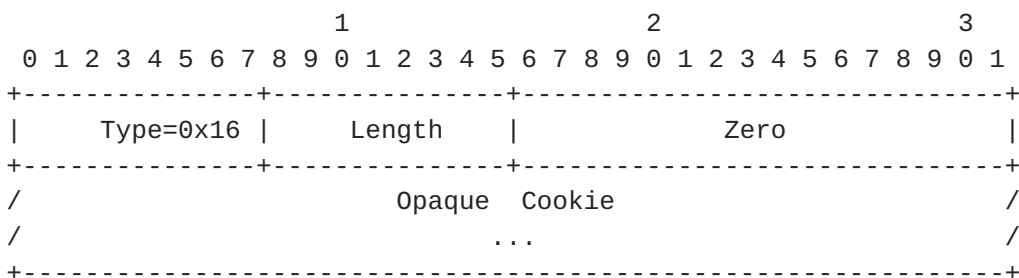
Figure 20: The Cookie TLV

4.2.8. Error TLV

The Error TLV (Figure 21) 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 close the associated connection.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+--------------+
|   Type=0x1E   |    Length     |   Error Code  |    Value     |
+---------------+---------------+---------------+--------------+
//             ...   (optional) Value                         //
+---------------------------------------------------------------+
```

Figure 21: 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:

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

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

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

o 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:

o 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 Transport Converter (or
   Client) 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
   Transport Converter (or Client). When multiple versions are
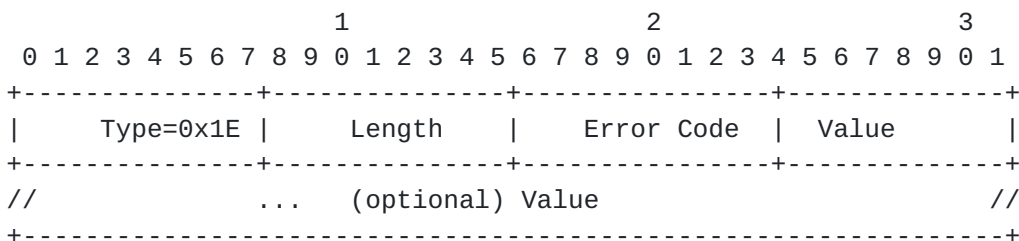   supported by the Transport Converter (or Client), it includes the
   list of supported version in the value field; each version is
   encoded in 8 bits. The list of supported versions should be
   padded with zeros to end on a 32 bits boundary.

   Upon receipt of this error code, the Client (or Transport
   Converter) checks whether it supports one of the versions returned
   by the Transport Converter (or Client). The highest common
   supported version MUST be used by the Client (or Transport
   Converter) in subsequent exchanges with the Transport Converter
   (or Client).

o Malformed Message (1): This error code is sent to indicate that a
   message received from a peer is can not 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 shifted by one byte to keep to original alignment of the
   message.

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

To ease troubleshooting, the value field MUST echo the received
message shifted by one byte to keep to original alignment of the
message.

o 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. 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 which receives this error code MUST cache the received
   Cookie and include it in subsequent Convert messages sent to that
   Transport Converter.

o 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, invalid Cookie TLV, etc.). 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.

o 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 bits boundary.

o 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.

o 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.

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

o Destination Unreachable (97): This error indicates that an ICMP
   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.

Figure 22 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                        |
+-------+------+------------------------------------------------+
```

Figure 22: Convert Error Values

5. Compatibility of Specific TCP Options with the Conversion Service

In this section, we discuss how several standard track TCP options
can be supported through the Convert protocol. The non-standard
track options and the experimental options will be discussed in other
documents.

5.1. 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 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 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.

5.2. 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 WS option in the
proxied message to be function of the scaling factor of the incoming
connection.

There is no benefit from a deployment viewpoint 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. It MUST NOT announce
it in a Supported TCP Extensions TLV.

5.3. Selective Acknowledgments

Two distinct TCP options were defined to support selective
acknowledgments 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.

5.4. Timestamp

The Timestamp option was initially defined in [RFC1323] and later
refined in [RFC7323]. It 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 sequence numbers (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.

5.5. Multipath TCP

The Multipath TCP options are defined in [RFC6824]. [RFC6824]
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 which 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.

5.6. 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 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.

5.7. TCP User Timeout

The TCP User Timeout option is defined in [RFC5482]. The associated TCP option (Kind=28) does not appear to be widely deployed.

5.8. TCP-AO

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 option 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 option (Kind=29) in the Supported TCP Extensions TLV. If a Transport Converter receives a Connect TLV that contains the TCP-AO option, 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.

5.9. TCP Experimental Options

The TCP Experimental options are defined in [RFC4727]. Given the variety of semantics for these options and their experimental nature, it is impossible to discuss them in details in this document.

6. **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 only discuss
   the middleboxes that modify SYN and SYN+ACK packets since the Convert
   Protocol places its messages in such packets.

   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 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 (but not
   the payload of SYN) can be detected by a Client. This is hinted by
   the absence of an Error or Extended TCP Header TLV in a response. If
   an Error was returned by the Transport Converter, a message to close
   the connection would normally follow from the Converter. If no such
   message is received, the Client may continue to use this Converter.

   As explained in [RFC7413], some CGNs (Carrier Grade NATs) 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, 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].

7. **Security Considerations**

7.1. 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 its location in the network, a Transport
Converter has access to the payload of all the packets that it
processes. As such, it MUST be protected as a core IP router (e.g.,
[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 network ensures that hosts are not sending traffic
with spoofed source IP addresses.

7.2. Authorization

The Convert Protocol is intended to be used in managed networks where
end hosts can be identified by their IP address.

Stronger mutual authentication schemes MUST be defined to use the
Convert Protocol in more open network environments. One possibility
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 client
authentication, 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.

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". Absent explicit configuration otherwise,
hairpinning is enabled by the Converter (see Figure 23.
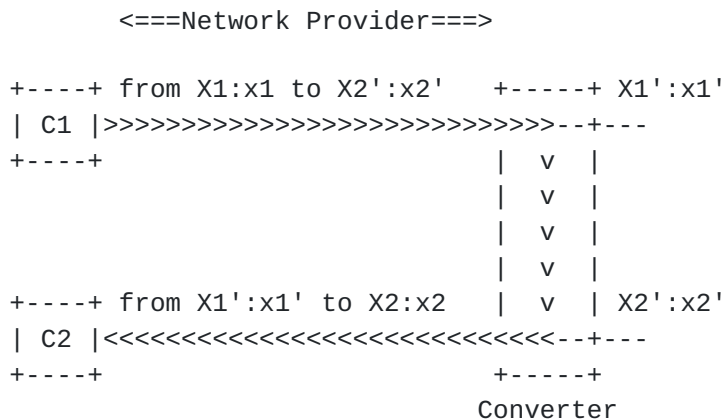
```
          <===Network Provider===>

   +----+ 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 23: Hairpinning Example

   See below for authorization considerations that are specific for
   Multipath TCP.

7.3. Denial of Service

   Another possible risk is the amplification attacks since a Transport
   Converter sends a SYN towards a remote Server upon reception of a SYN
   from a Client. This could lead to amplification attacks if the SYN
   sent by the Transport Converter were larger than the SYN received
   from the Client or if the Transport Converter retransmits the SYN.
   To mitigate such attacks, the Transport Converter SHOULD rate limit
   the number of pending requests for a given Client. It SHOULD also
   avoid sending to remote Servers SYNs that are significantly longer
   than the SYN received from the Client. 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 MUST also be enabled [RFC4987].

7.4. 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.

7.5. Multipath TCP-specific Considerations

   Multipath TCP-related security threats are discussed in [RFC6181] and
   [RFC6824].

   The operator that manages the various network attachments (including
   the Transport Converters) can enforce authentication and
   authorization policies using appropriate mechanisms. For example, a
   non-exhaustive list of methods to achieve authorization is provided
   hereafter:

   o 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 Multipath 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.

   o 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., [I-D.boucadair-radext-tcpm-converter].
      This method does not require any interaction with the Transport
      Converter for authorization matters.

   o A device that embeds a Transport Converter may also host a RADIUS
      client that will solicit an AAA server to check whether
      connections received from a given source IP address are authorized
      or not [I-D.boucadair-radext-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 Multipath TCP connections
   received on its Internet-facing interfaces. Only Multipath TCP
   connections received on the customer-facing interfaces of a Transport
   Converter will be accepted.

8. IANA Considerations

8.1. Convert Service Port Number

   IANA is requested to assign a TCP port number (TBA) 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-names-port-numbers.xhtml.

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

8.2. The Convert Protocol (Convert) Parameters

   IANA is requested to create a new "The Convert Protocol (Convert)
   Parameters" registry.

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

8.2.1. Convert Versions

   IANA is requested to create the "Convert versions" sub-registry. New
   values are assigned via IETF Review (Section 4.8 of [RFC8126]).

   The initial values to be assigned at the creation of the registry are
   as follows:

```
    +---------+--------------------------------------+-------------+
    | Version | Description                          | Reference   |
    +---------+--------------------------------------+-------------+
    |    0    | Reserved by this document            | [This-RFC]  |
    |    1    | Assigned by this document            | [This-RFC]  |
    +---------+--------------------------------------+-------------+
```

8.2.2. Convert TLVs

   IANA is requested to create the "Convert TLVs" sub-registry. The
   procedure for assigning values from this registry is as follows:

   o The values in the range 1-127 can be assigned via IETF Review.

   o The values in the range 128-191 can be assigned via Specification
      Required.

   o The values in the range 192-255 can be assigned for Private Use.

   The initial values to be assigned at the creation of the registry are
   as follows:

```
+---------+------------------------------------+-------------+
|  Code   | Name                               | Reference   |
+---------+------------------------------------+-------------+
|    0    | Reserved                           | [This-RFC]  |
|    1    | Info TLV                           | [This-RFC]  |
|   10    | Connect TLV                        | [This-RFC]  |
|   20    | Extended TCP Header TLV            | [This-RFC]  |
|   21    | Supported TCP Extension TLV        | [This-RFC]  |
|   22    | Cookie TLV                         | [This-RFC]  |
|   30    | Error TLV                          | [This-RFC]  |
+---------+------------------------------------+-------------+
```

8.2.3. Convert Error Messages

IANA is requested to create the "Convert Errors" sub-registry. 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:

o Message validation and processing errors: 0-31

o Client-side errors: 32-63

o Transport Converter-side errors: 64-95

o Errors caused by destination server: 96-127

The procedure for assigning values from this sub-registry is as
follows:

o 0-127: Values in this range are assigned via IETF Review.

o 128-191: Values in this range are assigned via Specification
   Required.

o 192-255: Values in this range are assigned for Private Use.

The initial values to be assigned at the creation of the registry are
as follows:

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

                   Figure 24: The Convert Error Codes

9. References

9.1. 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>.

[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>.

[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>.

[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>.

[RFC4727] Fenner, B., "Experimental Values In IPv4, IPv6, ICMPv4,
           ICMPv6, UDP, and TCP Headers", RFC 4727,
           DOI 10.17487/RFC4727, November 2006,
           <https://www.rfc-editor.org/info/rfc4727>.

[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>.

   [RFC5482] Eggert, L. and F. Gont, "TCP User Timeout Option",
              RFC 5482, DOI 10.17487/RFC5482, March 2009,
              <https://www.rfc-editor.org/info/rfc5482>.

   [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>.

   [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure,
              "TCP Extensions for Multipath Operation with Multiple
              Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013,
              <https://www.rfc-editor.org/info/rfc6824>.

   [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>.

   [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>.

   [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>.

9.2. Informative References

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

   [Fukuda2011]
              Fukuda, K., "An Analysis of Longitudinal TCP Passive
              Measurements (Short Paper)", Traffic Monitoring and
              Analysis. TMA 2011. Lecture Notes in Computer Science, vol
              6613. , 2011.

   [HotMiddlebox13b]
              Detal, G., Paasch, C., and O. Bonaventure, "Multipath in
              the Middle(Box)", HotMiddlebox'13 , December 2013,
              <http://inl.info.ucl.ac.be/publications/multipath-
              middlebox>.

   [I-D.arkko-arch-low-latency]
              Arkko, J. and J. Tantsura, "Low Latency Applications and
              the Internet Architecture", draft-arkko-arch-low-
              latency-02 (work in progress), October 2017.

   [I-D.boucadair-mptcp-plain-mode]
              Boucadair, M., Jacquenet, C., Bonaventure, O., Behaghel,
              D., stefano.secci@lip6.fr, s., Henderickx, W., Skog, R.,
              Vinapamula, S., Seo, S., Cloetens, W., Meyer, U.,
              Contreras, L., and B. Peirens, "Extensions for Network-
              Assisted MPTCP Deployment Models", draft-boucadair-mptcp-
              plain-mode-10 (work in progress), March 2017.

   [I-D.boucadair-radext-tcpm-converter]
              Boucadair, M. and C. Jacquenet, "RADIUS Extensions for
              0-RTT TCP Converters", draft-boucadair-radext-tcpm-
              converter-02 (work in progress), April 2019.

   [I-D.boucadair-tcpm-dhc-converter]
              Boucadair, M., Jacquenet, C., and R. K, "DHCP Options for
              0-RTT TCP Converters", draft-boucadair-tcpm-dhc-
              converter-02 (work in progress), April 2019.

   [I-D.nam-mptcp-deployment-considerations]
              Boucadair, M., Jacquenet, C., Bonaventure, O., Henderickx,
              W., and R. Skog, "Network-Assisted MPTCP: Use Cases,
              Deployment Scenarios and Operational Considerations",
              draft-nam-mptcp-deployment-considerations-01 (work in
              progress), December 2016.

[I-D.olteanu-intarea-socks-6]
          Olteanu, V. and D. Niculescu, "SOCKS Protocol Version 6",
          draft-olteanu-intarea-socks-6-07 (work in progress), July
          2019.

[I-D.peirens-mptcp-transparent]
          Peirens, B., Detal, G., Barre, S., and O. Bonaventure,
          "Link bonding with transparent Multipath TCP", draft-
          peirens-mptcp-transparent-00 (work in progress), July
          2016.

[IETFJ16] Bonaventure, O. and S. Seo, "Multipath TCP Deployment",
          IETF Journal, Fall 2016 , n.d..

[IMC11]   Honda, K., Nishida, Y., Raiciu, C., Greenhalgh, A.,
          Handley, M., and T. Hideyuki, "Is it still possible to
          extend TCP?", Proceedings of the 2011 ACM SIGCOMM
          conference on Internet measurement conference , 2011.

[RFC1323] Jacobson, V., Braden, R., and D. Borman, "TCP Extensions
          for High Performance", RFC 1323, DOI 10.17487/RFC1323, May
          1992, <https://www.rfc-editor.org/info/rfc1323>.

[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>.

[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>.

[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>.

   [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>.

   [RFC6181] Bagnulo, M., "Threat Analysis for TCP Extensions for
              Multipath Operation with Multiple Addresses", RFC 6181,
              DOI 10.17487/RFC6181, March 2011,
              <https://www.rfc-editor.org/info/rfc6181>.

   [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>.

   [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>.

   [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>.

   [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>.

   [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. Change Log

   This section to be removed before publication.

   o 00 : initial version, designed to support Multipath TCP and TFO
      only

   o 00 to -01 : added section Section 5 describing the support of
      different standard tracks TCP options by Transport Converters,
      clarification of the IANA section, moved the SOCKS comparison to
      the appendix and various minor modifications

   o 01 to -02: Minor modifications

   o 02 to -03: Minor modifications

   o 03 to -04: Minor modifications

   o 04 to -05: Integrate a lot of feedback from implementors who have
      worked on client and server side implementations. The main
      modifications are the following :

* TCP Fast Open is not strictly required anymore. Several
  implementors expressed concerns about this requirement. The
  TFO 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 TFO cookie
  inside the TCP options, which consumes precious space in the
  extended TCP header, this version supports the utilization of a
  Cookie that is placed in the SYN payload. This provides the
  same level of protection as a TFO Cookie in environments were
  such protection is required.

* the Bootstrap procedure has been simplified based on feedback
  from implementors

* Error messages are not included in RST segments anymore but
  sent in the bytestream. Implementors have indicated that
  processing such segments on clients was difficult on some
  platforms. This change simplifies client implementations.

* Many minor editorial changes to clarify the text based on
  implementors feedback.

o 05 to -06: Many clarifications to integrate the comments from the
   chairs in preparation to the WGLC:

   * Updated IANA policy to require "IETF Review" instead of
     "Standard Action"

   * Call out explicitly that data in SYNs are relayed by the
     Converter

   * Reiterate the scope

   * Hairpinning behavior can be disabled (policy-based)

   * Fix nits

o 07:

   * Update the text about supplying data in SYNs to make it clear
     that a constraint defined in RFC793 is relaxed following the
     same rationale as in RFC7413.

   * Nits

   * Added Appendix A on example Socket API changes

   o 08:

      * Added short discussion on the termination of connections

   o 09:

      * Address various comments received during last call

[Appendix B](#). Example Socket API Changes to Support the 0-RTT Convert Protocol

[B.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 TCP TFO can be used in two different
   modes depending on the host configuration (sysctl tcp_fastopen
   variable):

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

   o 0x4: (client) send 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.

[B.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 TFO
   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
the 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
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 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:

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

o 0x200: (server) accept 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 TFO cookie and
servers that do not use it.

Appendix C. Some Design Considerations

Several implementors expressed concerns about the use of TFO. As a
reminder, the TFO 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 TFO 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 TFO Cookie in environments were such protection is
required.

Error messages are not included in RST segments but sent in the
bytestream. Implementors have indicated that processing such
segments on clients was difficult on some platforms. This change
simplifies client implementations.

Appendix D. Address Preservation vs. Address Sharing

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

D.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.

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 25.

```
                                    Transport
            Client                  Converter          Server

            @:C1,C2                     @:Tc                  @:S
             ||                          |                     |
             |src:C1      SYN        dst:Tc|src:C1        dst:S|
             |-------MPC [->S:port]------->|-------SYN------->|
             ||                          |                     |
             ||dst:C1               src:Tc|dst:C1        src:S|
             |<---------SYN/ACK------------|<-----SYN/ACK-----|
             ||                          |                     |
             |src:C1                dst:Tc|src:C1        dst:S|
             |------------ACK------------->|-------ACK------->|
             |                          |                     |
             |src:C2          ...    dst:Tc|       ...         |
             ||<-----Secondary Subflow---->|src:C1        dst:S|
             ||                          |-------data------>|
             |                 ..         |     ...           |
```
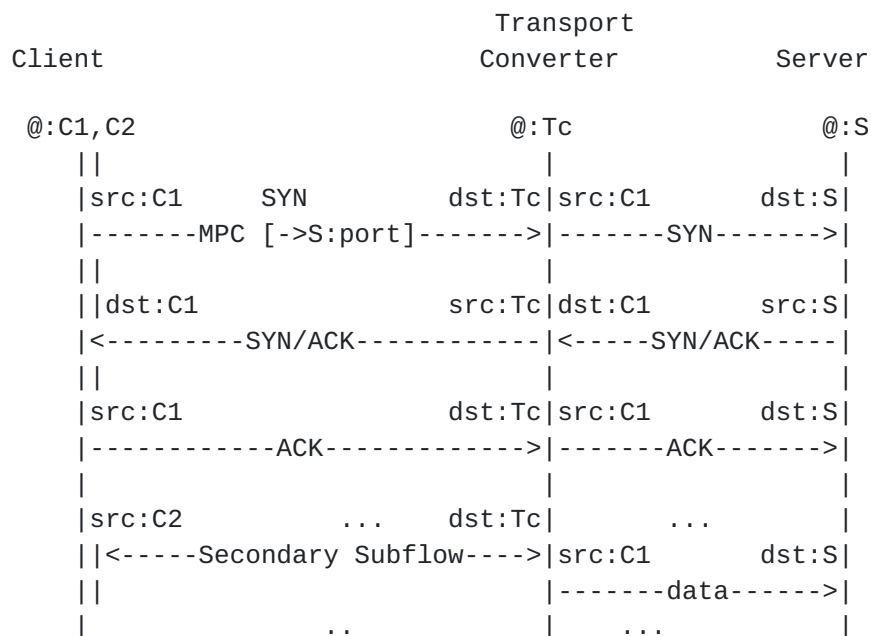
Legend:
   Tc: IP address used by the Transport Converter on its customer-facing
       interface.

                    Figure 25: 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.

D.2. IPv4 Address 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 pool is intercepted by the appropriate Transport

Converter. Unlike Appendix D.1, the Transport Converter drops
incoming packets which do match an active transport session entry.

An example is shown in Figure 26.

```
                                   Transport
          Client                   Converter              Server

           @:C                       @:Tc|Te                   @:S
            |                           |                  |
            |src:C                 dst:Tc|src:Te      dst:S|
            |-------SYN [->S:port]------->|-------SYN------->|
            |                           |                  |
            |dst:C                 src:Tc|dst:Te      src:S|
            |<---------SYN/ACK------------|<-----SYN/ACK-----|
            |                           |                  |
            |src:C                 dst:Tc|src:Te      dst:S|
            |------------ACK------------->|-------ACK------->|
            |                           |                  |
            |            ...            |    ...           |
```

Legend:
   Tc: IP address used by the Transport Converter for its customer-facing
       interface.
   Te: IP address used by the Transport Converter for its Internet-facing
       interface.

                        Figure 26: Address Sharing

Appendix E. Differences with SOCKSv5

At a first glance, the solution proposed in this document could seem
similar to the SOCKS v5 protocol [RFC1928] which is used to proxy TCP
connections. The Client creates a connection to a SOCKS proxy,
exchanges authentication information and indicates the destination
address and port of the final server. At this point, the SOCKS proxy
creates a connection towards the final server and relays all data
between the two proxied connections. The operation of an
implementation based on SOCKSv5 is illustrated in Figure 27.

```
   Client                       SOCKS Proxy                      Server
        ------------------->
             SYN
        <-------------------
             SYN+ACK
        ------------------->
             ACK

        ------------------->
        Version=5, Auth Methods
        <-------------------
             Method
        ------------------->
           Auth Request (unless "No auth" method negotiated)
        <-------------------
           Auth Response
        ------------------->
        Connect Server:Port           ------------------->
                                              SYN

                                      <-------------------
                                          SYN+ACK
        <-------------------
             Succeeded

        ------------------->
            Data1
                                      ------------------->
                                          Data1

                                      <-------------------
                                          Data2
        <-------------------
               Data2
```
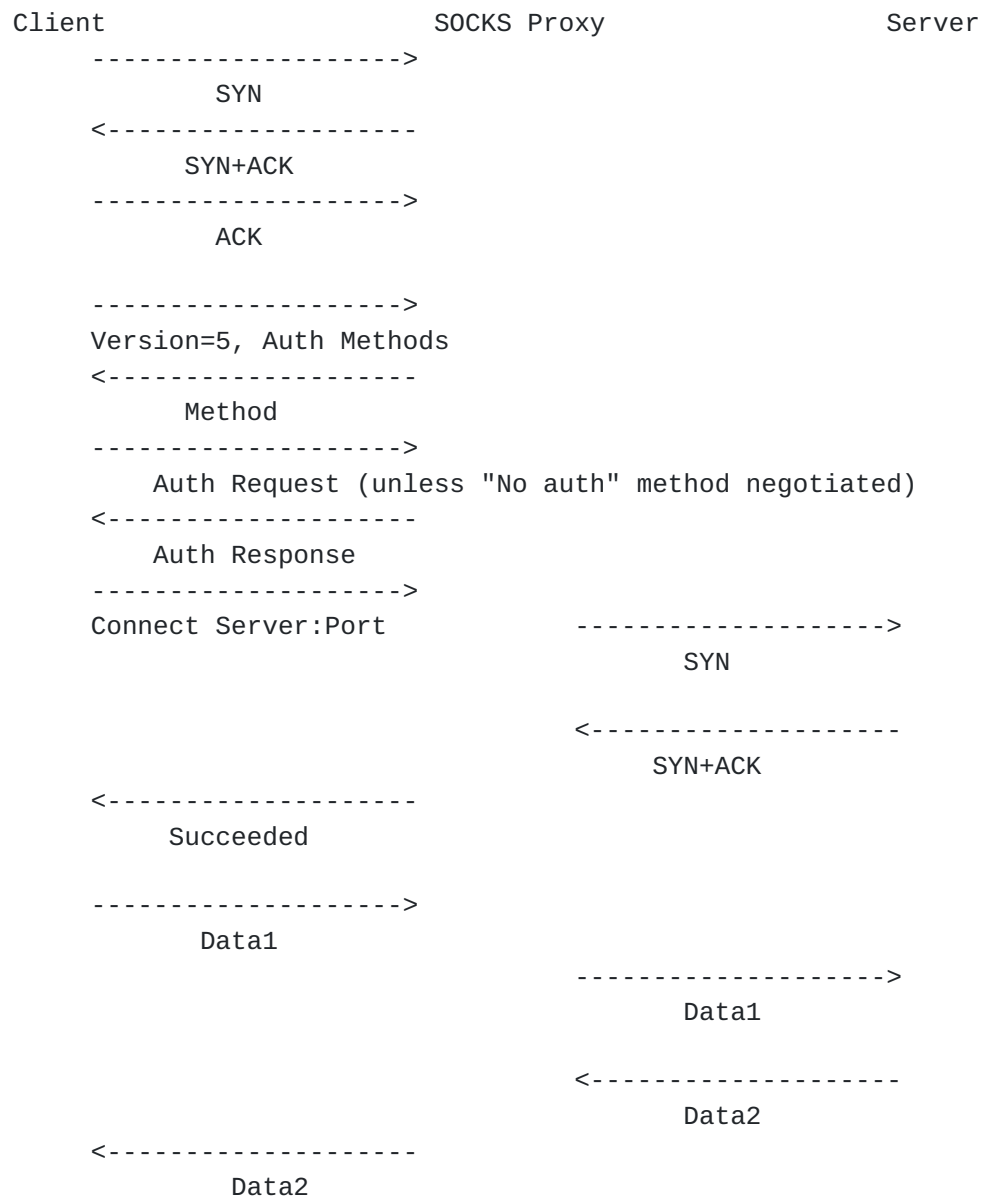
        Figure 27: Establishment of a TCP connection through a SOCKS proxy
                          without authentication

   The Convert protocol also relays data between an upstream and a
   downstream connection, but there are important differences with
   SOCKSv5.

   A first difference is that the Convert protocol exchanges all control
   information during the three-way handshake. This reduces the
   connection establishment delay compared to SOCKS that requires two or
   more round-trip-times before the establishment of the downstream
   connection towards the final destination. In today's Internet,
   latency is a important metric and various protocols have been tuned

to reduce their latency [I-D.arkko-arch-low-latency]. A recently
proposed extension to SOCKS leverages the TFO option
[I-D.olteanu-intarea-socks-6].

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 destination supports the required TCP extension.
Neither SOCKS v5 [RFC1928] nor the proposed SOCKS v6
[I-D.olteanu-intarea-socks-6] provide such a feature.

A third difference is that a Transport Converter will only accept the
connection initiated by the Client provided that the downstream
connection is accepted by the Server. If the Server refuses the
connection establishment attempt from the Transport Converter, then
the upstream connection from the Client is rejected as well. This
feature is important for applications that check the availability of
a Server or use the time to connect as a hint on the selection of a
Server [RFC8305].

A fourth difference is that the Convert protocol only allows the
client to specify the address/port of the destination server and not
a DNS name. We evaluated an alternate design for the Connect TLV
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.

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's helped to clarify different
parts of the text.

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

From [I-D.boucadair-mptcp-plain-mode]:

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 in IETF#95 (Buenos
Aires).

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

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 the discussion.

Contributors

Bart Peirens contributed to an early version of the document.

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

The authors of [I-D.boucadair-mptcp-plain-mode] were:

o Mohamed Boucadair

o Christian Jacquenet

o Olivier Bonaventure

o Denis Behaghel

o Stefano Secci

o Wim Henderickx

o Robert Skog

o Suresh Vinapamula

o SungHoon Seo

o Wouter Cloetens

o Ullrich Meyer

o Luis M. Contreras

   o Bart Peirens

   The authors of [I-D.peirens-mptcp-transparent] were:

   o Bart Peirens

   o Gregory Detal

   o Sebastien Barre

   o Olivier Bonaventure

Authors' Addresses

   Olivier Bonaventure (editor)
   Tessares

   Email: Olivier.Bonaventure@tessares.net

   Mohamed Boucadair (editor)
   Orange
   Clos Courtel
   Rennes 35000
   France

   Email: mohamed.boucadair@orange.com

   Sri Gundavelli
   Cisco

   Email: sgundave@cisco.com

   SungHoon Seo
   Korea Telecom

   Email: sh.seo@kt.com

   Benjamin Hesmans
   Tessares

   Email: Benjamin.Hesmans@tessares.net