Workgroup: Network Working Group
Internet-Draft: draft-piraux-tcpls-03
Published: 21 October 2022
Intended Status: Experimental
Expires: 24 April 2023
Authors: M. Piraux F. Rochet O. Bonaventure
UCLouvain University of Namur UCLouvain

TCPLS: Modern Transport Services with TCP and TLS

Abstract

This document specifies a protocol leveraging TCP and TLS to provide modern transport services such as multiplexing, connection migration and multipath in a secure manner.

Discussion Venues

This note is to be removed before publishing as an RFC.

Source for this draft and an issue tracker can be found at https://github.com/mpiraux/draft-piraux-tcpls.

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 24 April 2023.

Copyright Notice

Table of Contents

1. Introduction

The TCP/IP protocol stack continuously evolves. In the early days, most applications were interacting with the transport layer (mainly TCP, but also UDP) using the socket API. This is illustrated in Figure 1.

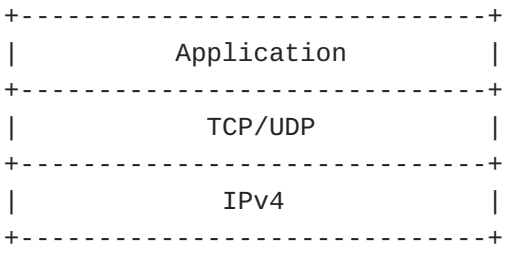

Figure 1: The classical TCP/IP protocol stack

The TCP/IP stack has slowly evolved and the figure above does not anymore describe current Internet applications. IPv6 is now widely deployed next to IPv4 in the network layer. In the transport layer, protocols such as SCTP [RFC4960] or DCCP [RFC6335] and TCP extensions including Multipath TCP [RFC8684] or tcpcrypt [RFC8548] have been specified. The security aspects of the TCP/IP protocol suite are much more important today than in the past [RFC7258]. Many applications rely on TLS [RFC8446] and their stack is similar to the one shown in Figure 2.

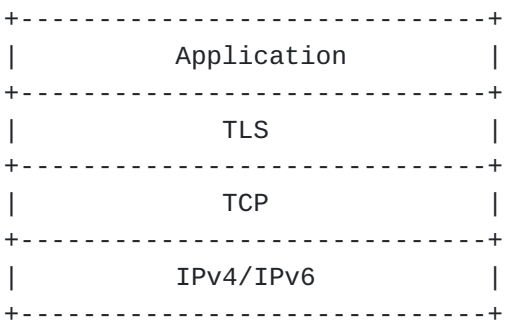

Figure 2: Today's TCP/IP protocol stack

Recently, the IETF went one step further in improving the transport layer with the QUIC protocol [RFC9000]. QUIC is a new secure transport protocol primarily designed for HTTP/3. It includes the reliability and congestion control features that are part of TCP and integrates the security features of TLS 1.3 [RFC8446]. This close

integration between the reliability and security features brings a lot of benefits in QUIC. QUIC runs above UDP to be able to pass through most middleboxes and to be implementable in user space. While QUIC reuses TLS, it does not strictly layer TLS on top of UDP as DTLS [I-D.ietf-tls-dtls13]. This organization, illustrated in Figure 3 provides much more flexibility than simply layering TLS above UDP. For example, the QUIC migration capabilities enable an application to migrate an existing QUIC session from an IPv4 path to an IPv6 one.

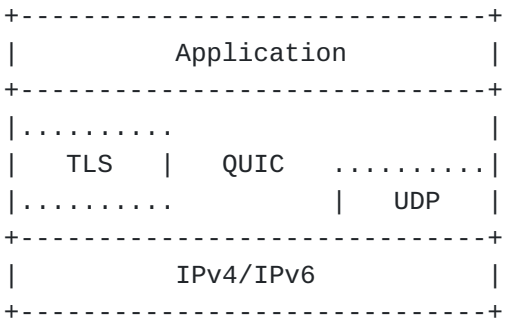

Figure 3: QUIC protocol stack

In this document, we revisit how TCP and TLS 1.3 can be used to provide modern transport services to applications. We apply a similar principle and combine TCP and TLS 1.3 in a protocol that we call TCPLS. TCPLS leverages the security features of TLS 1.3 like QUIC, but without begin simply layered above a single TCP connection. In addition, TCPLS reuses the existing TCP stacks and TCP's wider support in current networks. A preliminary version of the TCPLS protocol is described in [CONEXT21].

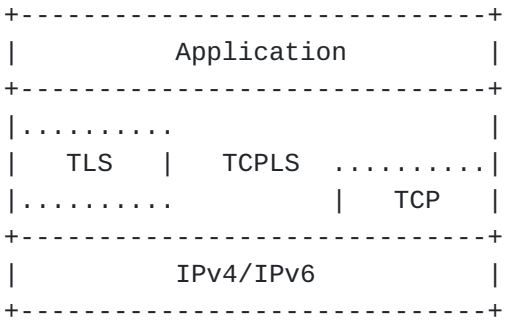

Figure 4: TCPLS in the TCP/IP protocol stack

In this document, we use the term TLS/TCP to refer to the TLS 1.3 protocol running over one TCP connection. We reserve the word TCPLS for the protocol proposed in this document.

This document is organized as follows. First, Section 3 summarizes the different types of services that modern transports expose to application. Section 4 gives an overview of TCPLS and how it supports these services. Finally, Section 5 describes the TCPLS in more details and the TLS Extensions introduced in this document.

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.

2.1. Notational conventions

This document uses the same conventions as defined in Section 1.3 of [RFC9000].

This document uses network byte order (that is, big endian) values. Fields are placed starting from the high-order bits of each byte.

3. Modern Transport Services

Application requirements and the devices they run on evolve over time. In the early days, most applications involved single-file transfer and ran on single-homed computers with a fixed-line network. Today, web-based applications require exchanging multiple objects, with different priorities, on devices that can move from one access network to another and that often have multiple access networks available. Security is also a key requirement of applications that evolved from only guaranteeing the confidentiality and integrity of application messages to also preventing pervasive monitoring.

With TCP and TLS/TCP, applications use a single connection that supports a single bytestream in each direction. Some TCP applications such as HTTP/2 [RFC7540] use multiple streams, but these are mapped to a single TCP connection which leads to Head-of-Line (HoL) blocking when packet losses occur. SCTP [RFC4960] supports multiple truly-concurrent streams and QUIC adopted a similar approach to prevent HoL blocking.

Modern transport services also changed the utilization of the underlying network. With TCP, when a host creates a connection, it is bound to the IP addresses used by the client and the server during the handshake. When the client moves and receives a different IP address, it has to reestablish all TCP connections bound to the previous address. When the client and the server are dual-stack, they cannot easily switch from one address family to another. Happy

Eyeballs [RFC8305] provides a partial answer to this problem for web applications with heuristics that clients can use to probe TCP connections with different address families. With Multipath TCP, the client and the server can learn other addresses of the remote host and combine several TCP connections within a single Multipath TCP connection that is exposed to the application. This supports various use cases [RFC8041]. QUIC [RFC9000] enables applications to migrate from one network path to another, but not to simultaneously use different paths.

4. TCPLS Overview

In order for TCPLS to be widely compatible with middleboxes that inspect TCP segments and TLS records, TCPLS does not modify the TCP connection establishment and only adds a TLS extension to the TLS handshake. Figure 5 illustrates the opening of a TCPLS session which starts with the TCP 3-way handshake, followed by the TLS handshake. In the Extensions of the ClientHello and in the server EncryptedExtensions, the tcpls TLS Extension is introduced to announce the support of TCPLS.

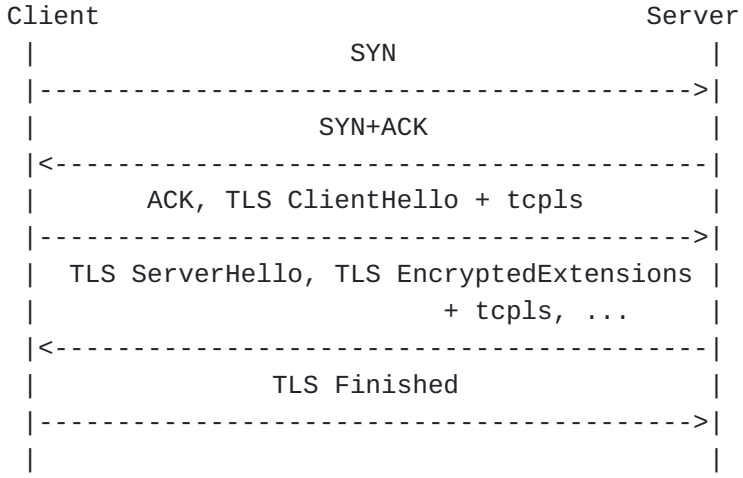

Figure 5: Starting a TCPLS session

TCP/TLS offers a single encrypted bytestream service to the application. To achieve this, TLS records are used to encrypt and secure chunks of the application bytestream and are then sent through the TCP bytestream. TCPLS leverages TLS records differently. TCPLS defines its own framing that allows encoding application data and control information. A TCPLS frame is the basic unit of information for TCPLS. A TCPLS frame always fits in a single record. One or more TCPLS frames can be encoded in a TLS record. This TLS record is then reliably transported by a TCP connection. Figure 6 illustrates the relationship between TCPLS frames and TLS records.

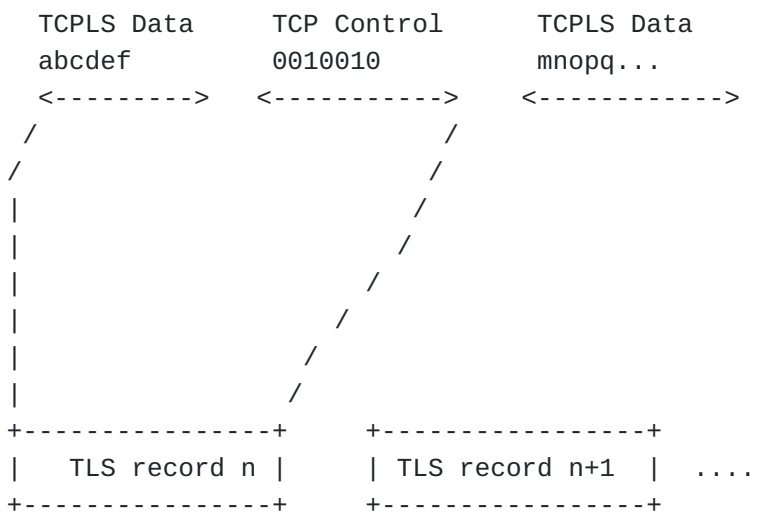

Figure 6: The first TLS record contains two TCPLS frames

4.1. Multiple Streams

TCPLS extends the service provided by TCP with streams. Streams are independent bidirectional bytestreams that can be used by applications to concurrently convey several objects over a TCPLS session. Streams can be opened by the client and by the server.

Streams are identified by a 32-bit unsigned integer. The parity of this number indicates the initiator of the stream. The client opens even-numbered streams while the server opens odd-numbered streams. Streams are opened in sequence, e.g. a client that has opened stream 0 will use stream 2 as the next one.

Data is exchanged using Stream frames whose format is described in Section 5.2.3. Each Stream frame carries a chunk of data of a given stream. Applications can mark the end of a stream to close it.

TCPLS enables the receiver to decrypt and process TLS records in zero copy similarly to TLS 1.3 under circumstances discussed in Section 4.5.

Similarly to HTTP/2 [RFC7540], conveying several streams on a single TCP connection introduces Head-of-Line (HoL) blocking between the streams. To alleviate this, TCPLS provides means to the application to choose the degree of HoL blocking resilience it needs for its application objects by spreading streams among different underlying TCP connections.

4.2. Multiple TCP connections

TCPLS is not restricted to using a single TCP connection to exchange frames. A TCPLS session starts with the TCP connection that was used to transport the TLS handshake. After this handshake, other TCP

connections can be added to a TCPLS session, either to spread the load or for failover. TCPLS manages the utilization of the underlying TCP connections within a TCPLS session.

Multipath TCP enables both the client and the server to establish additional TCP connections. However, experience has shown that additional subflows are only established by the clients. TCPLS focuses on this deployment and only allows clients to create additional TCP connections.

Using Multipath TCP, a client can try establishing a new TCP connection at any time. If a server wishes to restrict the number of TCP connections that correspond to one Multipath TCP connection, it has to respond with RST to the in excess connection attempts.

TCPLS takes another approach. To control the number of connections that a client can establish, a TCPLS server supplies unique tokens. A client includes one of the server supplied tokens when it attaches a new TCP connection to a TCPLS session. Each token can only be used once, hence limiting the amount of additional TCP connections.

TCPLS endpoints can advertise their local addresses, allowing new TCP connections for a given TCPLS session to be established between new pairs of addresses. When an endpoint is no more willing new TCP connections to use one of its advertised addresses, it can remove this address from the TCPLS session.

4.2.1. Joining TCP connections

The TCPLS server provides tokens to the client in order to join new TCP connections to the TCPLS session. Figure 7 illustrates a client and server first establishing a new TCPLS session as described in Section 4. Then the server sends a token over this connection using the New Token frame. Each token has a sequence number (e.g. 1) and a value (e.g. "abc"). The client uses this token to open a new TCP connection and initiates the TCPLS handshake. It adds the token inside the TCPLS Join TLS extension in the ClientHello.

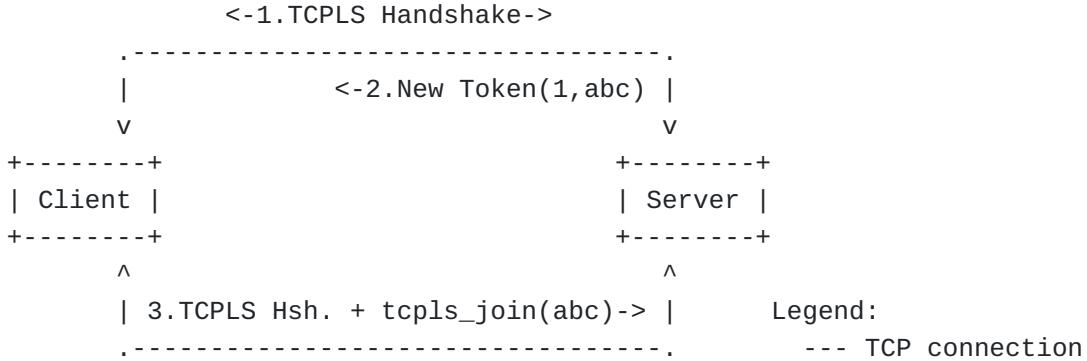

Figure 7: Joining a new TCP connection

When receiving a TCPLS Join Extension, the server validates the token and associates the TCP connection to the TCPLS session.

Each TCP connection that is part of a TCPLS session is identified by a 32-bit unsigned integer called its Connection ID. The first TCP connection of a session corresponds to Connection ID 0. When joining a new connection, the sequence number of the token, i.e. 1 in our example, becomes the Connection ID of the connection. The Connection ID enables the Client and the Server to identify a specific TCP connection within a given TCPLS session.

4.2.2. Robust session establishment

The TCPLS protocol also supports robust session establishment, where a multihomed or dual-stack client can establish a TCPLS session when at least one network path to the server can be established. This guarantees robustness against network failures and lowers the overall latency of a session establishment.

Figure 8 illustrates a dual-homed client that robustly establish a TCPLS session over two local addresses. In this example, the path from IP b towards the server exhibits a higher delay.

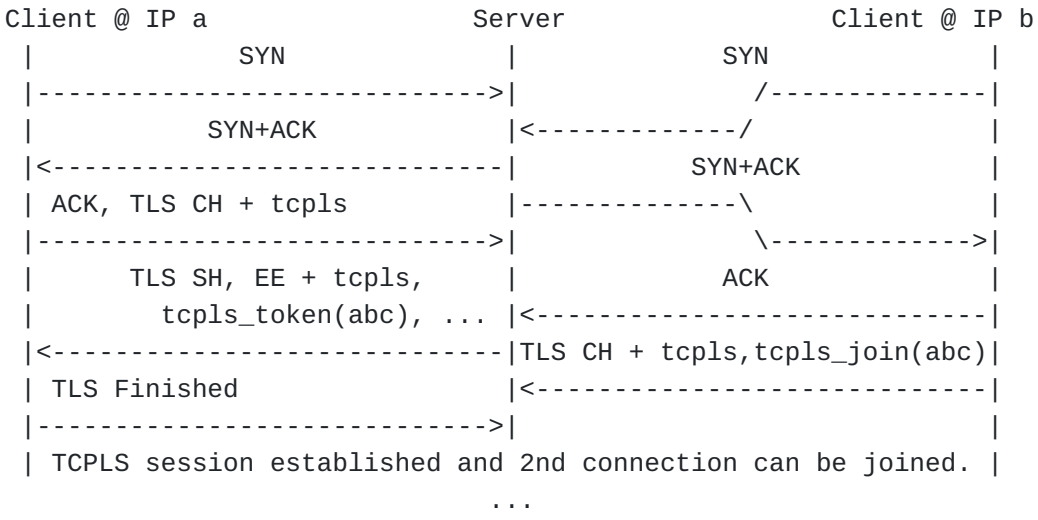

Figure 8: Robust session establishment example

The client starts by opening a TCP connection on from each of its local addresses. The TCP connection from IP "a" towards the server completes faster than the other. The client then starts a TCPLS Handshake on this connection. When the other connection is established, the client waits for receiving a TCPLS token allowing to join it to the session being established. In addition to the New Token frame, the TCPLS protocol enables the server to provide one

such token during the handshake using the TCPLS Token TLS extension. The server uses this extension when sending its EncryptedExtensions over the faster connection to provide the TCPLS token "abc". As soon as the client has received this token, it uses it over the other connection to join it to the session. When the TLS handshake completes over the fastest connection, the TCPLS session is established and the other connection can be joined to the session.

4.2.3. Failover

TCPLS supports two types of failover. In make-before-break, the client creates a TCP connection using the procedure described in Section 4.2.1 but only uses it once the initial connection fails.

In break-before-make, the client creates the initial TCP connection and uses it for the TCPLS handshake and the data. The server advertises one or more tokens over this connection. Upon failure of the initial TCP connection, the client initiates a second TCP connection using the server-provided token.

In both cases, some records sent by the client or the server might be in transit when the failure occurs. Some of these records could have been partially received but not yet delivered to the TCPLS layer when the underlying TCP connection fails. Other records could have already been received, decrypted and data of their frames could have been delivered to the application. To prevent data losses and duplication, TCPLS includes its own acknowledgments.

A TCPLS receiver acknowledges the received records using the ACK frame. Records are acknowledged after the record protection has been successfully removed. This enables the sender to know which records have been received. TCPLS enables the endpoint to send acknowledgments for a TCP connection over any connections, e.g. not only the receiving connection.

4.2.4. Migration

To migrate from a given TCP connection, an endpoint stops transmitting over this TCP connection and sends the following frames on other TCP connections. It leverages the acknowledgments to retransmit the frames of TLS records that have not been yet acknowledged.

When an endpoint abortfully closes a TCP connection, its peer leverages the acknowlegments to retransmit the TLS records that were not acknowlegded.

4.2.5. Multipath

TCPLS also supports the utilization of different TCP connections, over different paths or interfaces, to improve throughput or spread stream frames over different TCP connections. When the endpoints have opened several TCP connections, they can send frames over the connections. TCPLS can send all the stream frames belonging to a given stream over one or more underlying TCP connections. The latter enables bandwidth aggregation by using TCP connections established over different network paths.

4.3. Record protection

When adding new TCP connections to a TCPLS session, an endpoint does not complete the TLS handshake. TCPLS provides a nonce construction for TLS record protection that is used for all connections of a session. This reduces the cryptographic cost of adding connections. The endpoints **SHOULD** send TLS messages to form an apparent complete TLS handshake to middleboxes.

In order to use the TLS session over multiple connections, TCPLS adds a record sequence number space per connection that is maintained independently at both sides. Each record sent over a TCPLS session is identified by the Connection ID of its connection and its record sequence number. Each record nonce is constructed as defined in Figure 9.

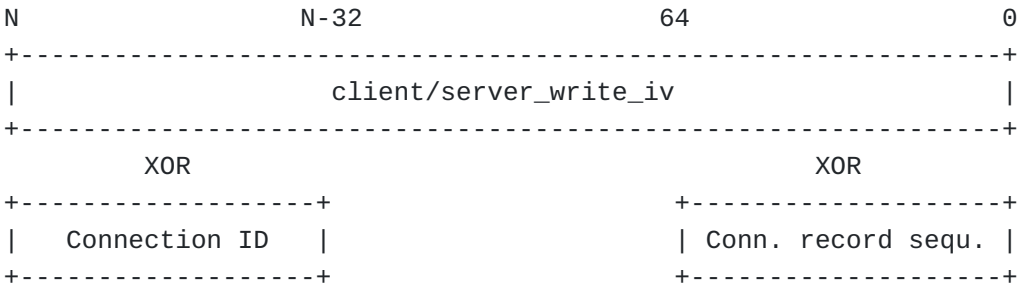

Figure 9: TCPLS TLS record nonce construction

This construction guarantees that every TLS record sent over the TLS session is protected with a unique nonce. As in TLS 1.3, the per-connection record sequence is implicit.

4.4. Closing a TCPLS session

Endpoints notify their peers that they do not intend to send more data over a given TCPLS session by sending a TLS Alert "close_notify". The alert can be sent over one or more TCP connections of the session. The alert **MUST** be sent before closing

the last TCP connection of the TCPLS session. The endpoint **MAY** close its side of the TCP connections after sending the alert.

When all TCP connections of a session are closed and the TLS Alert "close_notify" was exchanged in both directions, the TCPLS session is considered as closed.

We leave defining an abortful and idle session closure mechanisms for future versions of this document.

4.5. Zero-Copy Receive Path

TCPLS enables the receiver to process TLS records in a zero-copy manner under several conditions. When they are met, the application data carried in TCPLS frames can be decrypted at the right place in the application buffers.

First, zero-copy can be achieved when Stream frames of a given stream arrive in order. When using several TCP connections, out-of-order Stream frames cannot be processed in zero copy. A Multipath scheduling algorithm may target the minimization of out-of-order packets.

Second, the composition of TCPLS frames in a TLS record is impactful. The sender **SHOULD** encode a single Stream Data frame as the first frame of the record, followed by control-related frames if needed. When the sender encodes several Stream frames, the frame at the start of the record **SHOULD** be the largest, in order to maximise the use of zero copy. When several Stream frames are included in a record, they **SHOULD** belong to different streams.

5. TCPLS Protocol

5.1. TCPLS TLS Extensions

This document specifies three TLS extensions used by TCPLS. The first, "tcpls", is used to announce the support of TCPLS. The second, "tcpls_join", is used to join a TCP connection to a TCPLS session. The third, "tcpls_token", is used to provide a token to the client before the handshake completes. Their types are defined as follows.

```
enum {
    tcpls(TBD1),
    tcpls_join(TBD2),
    tcpls_token(TBD3),
    (65535)
} ExtensionType;
```

The table below indicates the TLS messages where these extensions can appear. "CH" indicates ClientHello while "EE" indicates EncryptedExtensions.

Extension	Allowed TLS messages
tcpls	CH, EE
tcpls_join	CH
tcpls_token	EE

Table 1: TLS messages allowed to carry TCPLS TLS Extensions

5.1.1. TCPLS

The "tcpls" extension is used by the client and the server to announce their support of TCPLS. The extension contains no value. When it is present in both the ClientHello and the EncryptedExtensions, the endpoints **MUST** use TCPLS after completing the TLS handshake.

5.1.2. TCPLS Join

```
struct {
    opaque token<32>;
} TCPLSJoin;
```

The "tcpls_join" extension is used by the client to join the TCP connection on which it is sent to a TCPLS session. The extension contains a token provided by the server. The client **MUST NOT** send more than one "tcpls_join" extension in its ClientHello. When receiving a ClientHello with this extension, the server checks that the token is valid and joins the TCP connection to the corresponding TCPLS session. When the token is not valid, the server **MUST** abort the handshake with an illegal_parameter alert.

5.1.3. TCPLS Token

```
struct {
    opaque token<32>;
} TCPLSToken;
```

The "tcpls_token" extension is used by the server to provide a token to the client during the TLS handshake. When receiving this extension, the client associates the token value as the first token of the TCPLS session, i.e. with a sequence number of 1. The server **MUST NOT** send this extension when the corresponding ClientHello contains a "tcpls_join" extension.

5.2. TCPLS Frames

TCPLS uses TLS Application Data records to exchange TCPLS frames. After decryption, the record payload consists of a sequence of TCPLS frames. Figure 10 illustrates the manner in which TCPLS frames are parsed from a decrypted TLS record. The receiver processes the frames starting from the last one to the first one. The fields of each frames are also parsed from the end towards the beginning of the TLS Application Data content. The parsing of a frame starts with the last byte indicating the frame type and then with type-specific fields preceeding it, forming a Type-Value unit. Such ordering enables a zero-copy processing of the type-specific fields as explained in Section 4.5.

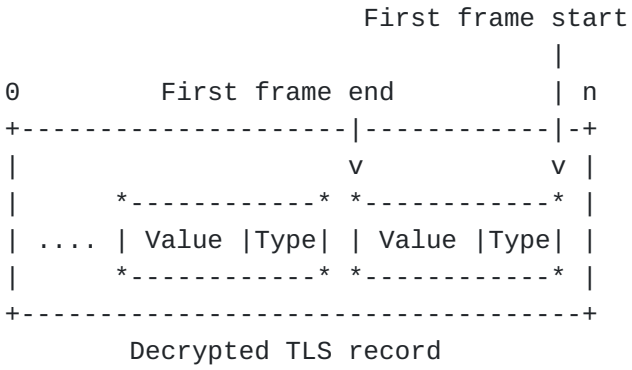

Figure 10: Parsing TCPLS frames inside a TLS record starts from the end.

Table 2 lists the frames specified in this document.

Type value	Frame name	Rules	Definition
0x00	Padding	N	Section 5.2.1
0x01	Ping		Section 5.2.2
0x02-0x03	Stream		Section 5.2.3
0x04	ACK	N	Section 5.2.4
0x05	New Token	S	Section 5.2.5
0x06	Connection Reset		Section 5.2.6
0x07	New Address		Section 5.2.7
0x08	Remove Address		Section 5.2.8
0x09	Stream Change	Cs	Section 5.2.9

Table 2: TCPLS frames

The "Rules" column in Table 2 indicates special requirements regarding certain frames.

N: Non-ack-eliciting. Receiving this frame does not elicit the sending of a TCPLS acknowledgment.

S:
Server only. This frame **MUST NOT** be sent by the client.

Cs: Connection-specific semantics. This frame is not idempotent and has specific semantics based on the TCP connection over which it is exchanged.

5.2.1. Padding frame

This frame has no semantic value. It can be used to mitigate traffic analysis on the TLS records of a TCPLS session. The Padding frame has no content.

```
Padding frame {
    Type (8) = 0x00,
}
```

Figure 11: Padding frame format

5.2.2. Ping frame

This frame is used to elicit an acknowledgment from its peer. It has no content. When an endpoint receives a Ping frame, it acknowledges the TLS record that contains this frame. This frame can be used by an endpoint to check that its peer can receive TLS records over a particular TCP connection.

```
Ping frame {
    Type (8) = 0x01,
}
```

Figure 12: Ping frame format

5.2.3. Stream frame

This frame is used to carry chunks of data of a given stream.

```
Stream frame {
    Stream Data (...),
    Length (16),
    Offset (64),
    Stream ID (32),
    FIN (1),
    Type (7) = 0x01,
}
```

Figure 13: Stream frame format

FIN:
The last bit of the frame type bit indicates that this Stream frame ends the stream when its value is 1. The last byte of the stream is at the sum of the Offset and Length fields of this frame.

Stream ID: A 32-bit unsigned integer indicating the ID of the stream this frame relates to.

Offset: A 64-bit unsigned integer indicating the offset in bytes of the carried data in the stream.

Length: A 16-bit unsigned integer indicating the length of the Stream Data field.

5.2.4. ACK frame

This frame is sent by the receiver to acknowledge the receipt of TLS records on a particular TCP connection of the TCPLS session. Although the reliability of the data exchange on a connection is handled by TCP, there are situations such as the failure of a TCP connection where a sender does not know whether the TLS frames that it sent have been correctly received by the peer. The ACK frame allows a TCPLS receiver to indicate the highest TLS record sequence number received on a specific connection. The ACK frame can be sent over any TCP connection of a TCPLS session.

```
ACK frame {
    Highest Record Sequence Received (64),
    Connection ID (32),
    Type (8) = 0x04,
}
```

Figure 14: ACK frame format

Connection ID: A 32-bit unsigned integer indicating the TCP connection for which the acknowledgment was sent.

Highest Record Sequence Received: A 64-bit unsigned integer indicating the highest TLS record sequence number received on the connection indicated by the Connection ID.

5.2.5. New Token frame

This frame is used by the server to provide tokens to the client. Each token can be used to join a new TCP connection to the TCPLS session, as described in Section 4.2.1. Clients **MUST NOT** send New Token frames.

```
New Token frame {
    Token (256),
    Sequence (8),
    Type (8) = 0x05,
}
```

Figure 15: New Token frame format

Sequence: A 8-bit unsigned integer indicating the sequence number of this token

Token: A 32-byte opaque value that can be used as a token by the client.

By controlling the amount of tokens given to the client, the server can control the number of active TCP connections of a TCPLS session. The server **SHOULD** replenish the tokens when TCP connections are removed from the TCPLS session.

5.2.6. Connection Reset frame

This frame is used by the receiver to inform the sender that a TCP connection has been reset.

```
Connection Reset frame {
    Connection ID (32)
    Type (8) = 0x06,
}
```

Figure 16: Connection Reset format

Connection ID: A 32-bit unsigned integer indicating the ID of the connection that failed.

5.2.7. New Address frame

This frame is used by an endpoint to add a new local address to the TCPLS session. This address can then be used to establish new TCP connections. The server advertises addresses that the client can use as destination when adding TCP connections. The client advertises address that it can use as source when adding TCP connections.

```
New Address frame {
    Port (16),
    Address (..),
    Address Version (8),
    Address ID (8),
    Type (8) = 0x07,
}
```

Figure 17: New Address format

Address ID: A 8-bit identifier for this address. For a given Address ID, an endpoint receiving a frame with a content that differs from previously received frames **MUST** ignore the frame. An endpoint receiving a frame for an Address ID that was previously removed **MUST** ignore the frame.

Address Version: A 8-bit value identifying the Internet address version of this address. The number 4 indicates IPv4 while 6 indicates IPv6.

Address: The address value. Its size depends on its version. IPv4 addresses are 32-bit long while IPv6 addresses are 128-bit long.

Port: A 16-bit value indicating the TCP port used with this address.

5.2.8. Remove Address frame

This frame is used by an endpoint to announce that it is not willing to use a given address to establish new TCP connections. After receiving this frame, a client **MUST NOT** establish new TCP connections to the given address. After receiving this frame, an endpoint **MUST** close all TCP connections using the given address.

```
Remove Address frame {
    Address ID (8),
    Type (8) = 0x08,
}
```

Figure 18: Remove Address format

Address ID: A 8-bit identifier for the address to remove. An endpoint receiving a frame for an address that was nonexistent or already removed **MUST** ignore the frame.

5.2.9. Stream Change frame

This frame is used by a sender to announce the Stream ID and Offset of the next record over a given TCP connection. It can be used to

make explicit a change in stream scheduling over a connection to the receiver, enabling a zero-copy receive path as explained in Section 4.5. The hint contained in this frame relates to the connection over which it was exchanged.

```
Stream Change frame {
    Type (8) = 0x09,
    Next Offset (64),
    Next Record Stream ID (32),
}
```

Figure 19: Stream Change format

Next Record Stream ID: A 32-bit unsigned integer indicating the Stream ID of the Stream frame in the next record .

Next Offset: A 64-bit unsigned integer indicating the Offset of the Stream frame in the next record.

6. Security Considerations

When issuing tokens to the client as presented in Section 4.2.1, the server **SHOULD** ensure that their values appear as random to observers and cannot be correlated together for a given TCPLS session.

The security considerations for TLS apply to TCPLS. The next versions of this document will elaborate on other security considerations following the guidelines of [RFC3552].

7. IANA Considerations

IANA is requested to create a new "TCPLS" heading for the new registry described in Section 5.2. New registrations in TCPLS registries follow the "Specification Required" policy of [RFC8126].

7.1. TCPLS TLS Extensions

IANA is requested to add the following entries to the existing "TLS ExtensionType Values" registry.

Value	Extension Name	TLS 1.3	Recommended	Reference
TBD1	tcpls	CH, EE	N	This document
TBD2	tcpls_join	CH	N	This document
TBD3	tcpls_token	EE	N	This document

Table 3

Note that "Recommended" is set to N as these extensions are intended for uses as described in this document.

7.2. TCPLS Frames

IANA is requested to create a new registry "TCPLS Frames Types" under the "TCPLS" heading.

The registry governs an 8-bit space. Entries in this registry must include a "Frame name" field containing a short mnemonic for the frame type. The initial content of the registry is present in Table 2, without the "Rules" column.

8. References

8.1. Normative References

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

[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/rfc/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/rfc/rfc8174>.

[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/rfc/rfc8446>.

8.2. Informative References

[CONEXT21] Rochet, F., Assogba, E., Piraux, M., Edeline, K., Donnet, B., and O. Bonaventure, "TCPLS - Modern Transport Services with TCP and TLS", Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT'21) , December 2021.

[I-D.ietf-tls-dtls13] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-43, 30 April 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-dtls13-43>.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/rfc/rfc3552>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/rfc/rfc4960>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/rfc/rfc6335>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <https://www.rfc-editor.org/rfc/rfc7258>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/rfc/rfc7540>.

[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/rfc/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/rfc/rfc8305>.

[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/rfc/rfc8548>.

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

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

Appendix A. Alternative Designs

In this section, we discuss alternatives to mechanisms defined in this document.

A.1. Securing new TCP connections with New Session Tickets

When adding a new TCP connection to a TCPLS session, endpoints derive a Initial Vector based on the technique presented in Section 4.3. An alternative is to use New Session Tickets to derive separate cryptographic protection materials based on a pre-shared key sent by the server. In this mode of operation, the server provides New Session Tickets to control the amount of additional TCP connections that can be opened by the client. The server could encode the Connection ID in the ticket value.

Using this mechanism differs from our proposal in several ways. While it enables to use an existing TLS mechanism for this purpose, it has a number of differences. First, it requires the server to compute pre-shared keys which could be more costly than computing the TCPLS tokens defined in Section 4.2.1. Second, when used to establish a TLS session, additional TLS messages must be computed and exchanged to complete the handshake, which the current mechanism does not require. Third, TLS New Session Tickets have a lifetime that is separated from the session they are exchanged over. This is unneeded in the context of TCPLS and may require additional protocol specification and guidance to implementers.

Acknowledgments

This work has been partially supported by the ``Programme de recherche d'interet general WALINNOV - MQUIC project (convention number 1810018)'' and European Union through the NGI Pointer programme for the TCPLS project (Horizon 2020 Framework Programme, Grant agreement number 871528). The authors thank Quentin De Coninck and Louis Navarre for their comments on the first version of this draft.

Change log

Since draft-piraux-tcpls-02

- Adds the TCPLS Token TLS extension to enable fast robust session establishment.

Since draft-piraux-tcpls-01

- Change frames and fields order to enable zero-copy receiver.

Since draft-piraux-tcpls-00

- Added the addresses exchange mechanism with New Address and Remove Address frames.

Authors' Addresses

Maxime Piraux
UCLouvain

Email: maxime.piraux@uclouvain.be

Florentin Rochet
University of Namur

Email: florentin.rochet@unamur.be

Olivier Bonaventure
UCLouvain

Email: olivier.bonaventure@uclouvain.be