draft-piraux-tcplls

TCPLS: Modern Transport Services with TCP and TLS

Maxime Piraux, Olivier Bonaventure,
UCLouvain
Florentin Rochet
University of Edinburgh
Content

- Introduction
- Using TLS for transport protocol extensibility
- Opportunities for the transport stack
- The TCPLS protocol
- Conclusion
The design of MPTCP

- In 2009, the mptcp WG formed with an initial design involving TCP Options
- In 2013, v0 shipped and enabled
  - Bandwidth aggregation of several TCP subflows
  - Failover in case of network failure
  - Backwards compatibility with TCP
- MPTCP has several issues
  - Address exchange is not secure, improved in MPTCP v1
  - TCP is prone to middlebox interference
  - Can be difficult to implement
    - 7-year journey from specification to mainline Linux
The design of QUIC

- In 2016, the quic WG formed to design an UDP-based transport protocol
- In 2021, QUIC v1 shipped and enabled:
  - Stream multiplexing
  - Connection migration, failover
- TLS secures most of the QUIC header and all QUIC payloads
- QUIC can be implemented in user-space and shipped with applications
Using TLS for transport protocol extensibility

- TLS is the most used protocol atop TCP
- TLS version 1.3 used encryption to extend the protocol
  - Encrypted TLS records and Encrypted Extensions allows securely exchanging control and application data
- TCP support in the network and in operating systems remains wider
- Given the ubiquity of TLS, can we provide new transport services with TCP and TLS?
Opportunities for the transport stack

● Build an encrypted transport protocol
  ○ Stream multiplexing
    ■ App-chosen HoL blocking resilience
  ○ Connection Migration
    ■ Based on app triggers and network conditions
  ○ Multipath
    ■ Scheduling at the TLS record level

● More efficient than the HTTP/2+TLS+MPTCP stack
  ○ Built on a strict layering assumption

● Clean slate for other transport extensions
The TCPLS protocol

- Session establishment
- Exchanging application and control data
- Adding TCP connections
- Record acknowledgements
- Modern Transport Services
  - Stream multiplexing
  - Failover
  - Bandwidth aggregation
Session establishment

- TCPLS does not modify the TCP and TLS handshake
- `tcpls` is a TLS Extension indicating the support of TCPLS
- Compatible with TCP TFO and TLS 0-RTT Handshake
Stream multiplexing

- Streams provide concurrent bytestreams to applications
- TCPLS manages the streams and multiplexes them
Exchanging data

- Application and control data can then be sent in TLS encrypted records using TCPLS frames
- Frames compose TLS records
Example: A TLS record containing a TCPLS Stream frame

<table>
<thead>
<tr>
<th>OpaqueType = AppData</th>
<th>Version = 0x303</th>
<th>Length</th>
</tr>
</thead>
</table>

TLS Ciphertext header | TLS Encrypted record

- **TCPLS**
- **TLS Ciphertext**
- **TLS Cleartext**
Example: A TLS record containing a Stream frame

<table>
<thead>
<tr>
<th>TCPLS</th>
<th>Content</th>
<th>Type = AppData</th>
</tr>
</thead>
</table>

TLS Application Data record
Example: A TLS record containing a Stream frame

<table>
<thead>
<tr>
<th>FrameType</th>
<th>Stream ID</th>
<th>Offset</th>
<th>Length</th>
<th>FIN</th>
<th>Stream data</th>
</tr>
</thead>
</table>

Stream frame

- TCPLS
- TLS Ciphertext
- TLS Cleartext
Stream multiplexing

- Streams provide concurrent bytestreams to applications
- TCPLS manages the streams and multiplexes them
- Streams multiplexed on a single connection are subject to HoL blocking
Stream multiplexing

- TCPLS manages TCP connections and schedules the TLS records.
- By mapping streams to connections, the app chooses the streams it wants to protect, and the ones that are bound together.
Adding TCP connections

- Server gives tokens to the client
- Each token can be used by the client to open and join an additional TCP connection
- Server can limit the connections by limiting the tokens
- The Sequence number of the Token becomes the Connection ID

Both endpoint can use TCPLS

\[\text{TLS AppData} = \{\text{NewToken} = [1, \text{ “abc”}]\}\]
Adding TCP connections

- The client put the token in the `tcplsJoin` TLS Extension
- The server validates the token and joins the TCP connection to the session
Adding TCP connections

- Each TLS record is encrypted with a unique nonce
- Record sequence is kept implicit
- The record sequence cannot be shared among TCP connections
- **We do not want to do a full TLS handshake, which is costly**

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client used token “abc”</td>
<td>What crypto material should be used?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>N-32</th>
<th>64</th>
<th>0</th>
</tr>
</thead>
</table>

**TLS 1.3 Initial Vector**

**TLS Per-record Nonce**
Adding TCP connections

- Each TLS record is encrypted with a unique nonce
- Record sequence is kept implicit
- The record sequence cannot be shared among TCP connections
- **We XOR the Connection ID to the nonce and add a per-connection record sequence**

```
<table>
<thead>
<tr>
<th>Client used token “abc”</th>
</tr>
</thead>
<tbody>
<tr>
<td>What crypto material should be used?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>N-32</th>
<th>64</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOR</td>
<td>XOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connection ID</td>
<td>Conn. record seq.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TLS 1.3 Initial Vector**

**TCPLS Per-record Nonce**
Failover

- Endpoints can reinject frames from lost records onto other TCP connections
- They know which records have been received
Record acknowledgements

If CID 0 fails, how to know whether this record was received?
Record acknowledgements

- Each record is identified by:
  - its TLS record sequence number
  - the Connection ID (=Token sequence number) it was sent on
- ACK frame indicates the sequence number of the latest record received over a connection
- As TCP delivers data in sequence, only cumulative ACKs are needed
Record acknowledgements

If CID 0 fails, how to know whether this record was received?

Client
CID 0

Server
CID 1

TLS AppData={ } (Seq=3)

If CID 0 fails, how to know whether this record was received?

TLS AppData={ } (Seq=4)

TLS AppData=[
  ACK={CID=0, Seq=3}]
(Seq=0)

TLS AppData={ } (Seq=0)

TLS AppData=[
  ACK={CID=0, Seq=3},
  ACK={CID=1, Seq=0}]
(Seq=1)
Bandwidth aggregation

- Endpoints can send Stream frames of a given stream on several TCP connections, benefiting from bandwidth aggregation.
draft-piraux-tcplsls

- It describes the protocol presented here
- We welcome feedback and comments on the draft
  - For both the protocol and the use-cases
- We will continue working on improving the protocol
- Some parts will be discussed in future versions
  - Congestion control
  - Flow control
- Followed a preliminary version of the TCPLS protocol presented at CoNEXT’21 [1]

Prototype

- We implemented *draft-piraux-tcpls-01* on top of *picotls*, a TLS 1.3 implementation in C
- We modified 50 lines of *picotls* for the required TCPLS interface
- The prototype implements stream multiplexing, failover and multipath
- It consists of 2.5k lines of C
- We will release the prototype under an open-source license
Conclusion

● TCPLS is a secure, user-space, transport protocol bringing
  ○ Stream multiplexing
  ○ Connection migration, Failover
  ○ Multipath

● TCPLS leverages in-kernel high performance TCP implementations

● We implemented a prototype in 2.5k lines of C
  ○ We will publish the code

● We are interested pursuing this work within the IETF
  ○ Should we start with a dedicated mailing list?
Backup – HTTP/2

- HTTP/2+TLS+MPTCP is built on strict layering assumption
- TCPLS offers more control to the application over the TCP connections of the session

TCPLS offers a better control and leverages several TCP connections

How to divide in:
- Frame?
- Record?
- Packets?

Diagram:
- Application
  - HTTP/2
  - TLS
  - MPTCP
  - IPv4/IPv6
- TCPLS
  - TLS
  - TCP
  - TCP
  - IPv4/IPv6