A $\LaTeX$ Template for SIGCOMM 18

Paper # XXX, XXX pages

1 INTRODUCTION [BUDGET=1P]

The Transmission Control Protocol (TCP) [42] is one of the most critical protocols in today's Internet. A wide range of applications that require reliable delivery use it. During the last four decades, TCP evolved under the pressure of competing protocols. During the 1980s, software-based TCP implementations were considered too slow. Researchers proposed new transport protocols such as XTP [46] which could be implemented in hardware. TCP implementations got a considerable speed boost [13], and XTP did not succeed. However, the TCP speed boost and usage triggered the development of various important TCP extensions, including, timestamps and large windows [7] to scale to the gigabit link speed or Selective Acknowledgments [19].

During the late nineties, early 2000s, transport protocol researchers explored other alternatives to TCP . Two of these approaches were adopted and standardized within the IETF: DCCP [33] and SCTP [49]. We rarely use DCCP today. Despite its benefits (support for multihoming, better design, and extensibility), only a few niche applications use SCTP [9]. This limited deployment is probably due to two different factors. First, SCTP required changes to the applications to replace TCP . Second, operators have deployed middleboxes (NAT, firewalls) that often block packets that do not carry TCP or UDP [27].

SCTP initially supported multihoming by switching from one path to another. It was later extended to be able to use different paths continuously [31]. Multipath TCP [20, 43] brought similar multihoming capabilitiy to TCP , and included a coupled congestion control scheme [58], later brought to SCTP as well. This particular succession of events shows how different designs can collobarate to advance each others. Multipath TCP is now deployed, notably on smartphones [6]. Other recent TCP extensions include TCP Fast Open [12] or TCPCrypt [5].

In the mid-nineties, the Secure Socket Layer protocol was proposed to secure emerging e-commerce websites [17]. This protocol evolved in different versions of the Transport Layer Security (TLS) protocol, the most recent one being version 1.3 [44]. Many details of the TLS protocol have changed since the first version of SSL [34]. Nowadays, TLS is almost ubiquitous on web servers [26] thanks to the availability of various TLS implementations and automated certificate authorities [1]. Furthermore, many non-web applications also rely on TLS [2].

Transport protocols continued to evolve in parallel. QUIC started as a proprietary protocol used by Google to speed up web transfers [35, 45]. During the last years, it evolved into a complete transport protocol whose standardization is being finalized within the IETF [30]. QUIC combines the functions that are usually found in TCP , TLS , and HTTP/2. A key characteristic of QUIC is that it encrypts almost all the packets, including most of their headers. Although QUIC is essentially a new transport protocol, it does not run directly above IP in contrast with SCTP, TCP , or DCCP. QUIC runs above UDP. This choice is mainly motivated by the desire to avoid as much as possible middlebox interference. QUIC's clean architecture has attracted researchers who have already proposed various extensions to the protocol [14, 15, 29, 38, 41, 50, 55].

Does the finalization of version 1 of the QUIC specification mark the end of the TCP era and move all transport research on this new protocol? We do not think so. History tells us that TCP has evolved with competing transport protocols. QUIC is today's competitor, but there is still plenty of room to improve TCP .

In this paper, we take a step back. As QUIC benefits from a closer integration between the reliability and the security mechanisms, we reconsider the separation between TCP and TLS . TLS brings security features, but TLS 1.3 can do much more. Thanks to the TLS 1.3 messages and records' extensibility, TLS can provide a secondary channel that enables hosts to exchange more control information and structured data. Furthermore, since TLS records are encrypted, middleboxes cannot easily interfere with the data exchanged over this new channel.

We combine TCP and TLS in a protocol that we call **TCPLS** . We describe in Section **??** a first design for TCPLS with the goals of *(i)* solving extensibility issues in TCP . *(ii)* Exporting complex transport features to the application and *(iii)* drawing a path to make TCP /TLS a good challenger to QUIC with modern appications. Then, we discuss how TLS ' flexible record layer can be used to provide a new channel to exchange information between TCPLS implementations. The design presentation concludes with an overview of the API to interact with the application. Our second contribution is the ongoing implementation of a TCPLS prototype on Linux by extending `picotls`, a TLS 1.3 implementation. We use it in Section 4 to illustrate the benefits of TCPLS with a multihoming connection migration use case. Finally, we analyze in Section **??** some of the research questions that TCPLS opens.

2 BACKGROUND [BUDGET=0.75P]

Here comes background on TCP/TLS/UDP/QUIC/MIDDLEBOXES/BPF

3 TCPLS DESIGN [BUDGET=3P]

TCPLS offers a cross-layer interface to TLS and TCP with the motivation to do more than securing the transport layer. Merging the stacks benefits both protocols and the application using this new approach. First, TCP suffers from a lack of extensibility due to size restrictions in its header and due to potential middlebox interferences [27]. TCPLS aims to solve TCP 's extensibility issue in the long run by offering a secure control channel to exchange TCP options without suffering from middlebox interferences and size restrictions in TCP headers. Second, TLS does not have a clear view of the transport protocol, and offering one with TCPLS brings opportunity for performance improvement (e.g., avoiding records fragmentation by matching the record size to the congestion window), and for connection reliability (e.g., failover). Third, applications are becoming more complex, which appeals to exposing transport-level functionalities and letting them tune the underlying transport to their use case. Essentially, this last motivation discusses a novel manner to expose transport-level functionalities that are encrypted, authenticated, reliable, extensible and adapted to complex application-level requirements.

3.1 Overview

TCP separates control information and data by placing the control information in the packet header and the data in the payload. This separation worked well until middleboxes started to interfere with TCP [16, 27, 37]. On a fraction of Internet paths, including e.g., some enterprise and cellular networks, some middleboxes interfere by adding, removing, or changing TCP options [27, 56, 59] and, in some cases, also transparently terminating TCP connections. These middleboxes have slowed down the evolution of TCP in recent years. TCPLS also uses the packet header to exchange TCP control information, but it leverages TLS to create a second and secure control channel. In a nutshell, TCPLS leverages the extensibility of TLS 1.3 to place control information such as TCP options inside the TLS handshake messages and new TLS records. Since this information is encrypted and authenticated, the communicating hosts can exchange new control information without encountering middlebox interference. We describe several examples of these new types of control information in Section 3.2 and Section 4.3.1.

In our current prototype, a TCPLS session starts with a classic TCP handshake. Immediately after, the client sends the ClientHello TLS message. The server replies with a ServerHello message which can contain encrypted data but also encrypted control information. For example, a dual-stack server may advertise its IPv6 address in the encrypted ServerHello message when contacted over its IPv4 address. We highlight one of our roadmap features in Section **??** to enable a 0-RTT TCPLS which would enable TCP to catchup the QUIC design regarding fast connection establishment. We also describe in Section 4 how our current prototype uses this information to support connection migration, failover, and other features.

Once the TCPLS session has been established, TCPLS sends TLS records. Most of these records contain application data transmitted by the client or the server. The control channel between the client and the server enables TCPLS to support new features, such as streams. Indeed, applications such as HTTP/2 support multiple streams mapped to a single TCP connection. However, there are situations, e.g., to prevent head-of-line blocking, where different streams should be mapped over other underlying TCP connections. With TCPLS , the client and the server can establish different datastreams over a single TCPLS session. The data from all these streams is encrypted using TLS . Furthermore, thanks to the TCPLS API, the client and the server can map each data stream to an underlying TCP connection. Thus, a TCPLS session can be composed of one or more TCP connections similarly as a Multipath TCP connection gathers subflows. Note that, if the multipath mode is enabled, then a lost packet over one TCP connection may create HOL blocking since packets received on other streams may have to wait for the lost packet to be properly reordered and delivered to the application. In the case of the multipath mode not enabled, this problem would not happen, but the application has to be careful to map application objects per stream, and not to mix these objects among several streams since the ordering would not be guaranteed.

To support data from a given datastream to be exchanged over several TCP connections, TCPLS includes its sequence numbers. A client and server can also enable acknowledgments. Thanks to these TCPLS acknowledgments, a TCPLS session can react to the failure of the underlying TCP connection by reestablishing a new TCP connection to continue the transfer of data and replay the records that have been lost.

A TCP connection ends with the exchange of FIN or RST packets. However, some middleboxes force the termination of TCP connections by sending RST packets [18, 57]. TCPLS can preserve established connections by automatically restarting the underlying TCP connection upon reception of a spurious reset. TCPLS defines the connection termination at the stream level: closing the last stream attached to a TCP connection allows clients and servers to securely terminate the TCPLS session.

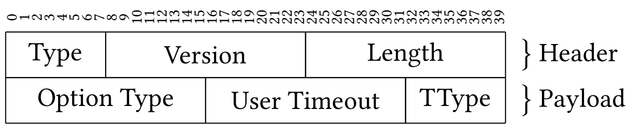

Figure 1: A new type of TLS Record containing a TCP option.

3.2 The Secure Control Channel

TLS 1.3 [44] has been designed with careful consideration for potential extensions. It supports the EncryptedExtensions message sent by the server alongside the ServerHello. Any extension sent with the ServerHello message is encrypted with the handshake key, and is not part of the context used to derive the eventual application key.

A reasonable approach to designing extensibility mechanisms in today's Internet is to avoid leaking any information that could help an on-path attacker recognize specific users or applications. Indeed, censorship [10, 23, 39] can be easily implemented when protocol messages can be distinguished, and avoiding trivial opportunities to implement censorship should become the bare minimum in designing a new protocol. TCPLS 's control protocol considers those problems by avoiding unencrypted data within the ClientHello.

In our design, the client indicates its willingness to use TCPLS with a transport parameter in the ClientHello. Upon reception of this parameter, the server can opportunistically send lightweight TCPLS data and TCP options as EncryptedExtensions. If the client does not support some extension, it echoes back an alert with the value of the option it does not recognize, but the connection continues.

The server or the client can also send TCPLS control messages after the handshake. These control messages take advantage of the TLS 1.3 content-type extensibility feature to avoid middlebox interference. Indeed, in TLS 1.3, the Record Protocol ensures that any new message appears as an APPDATA message type while the true content type (TType) is stored at the end of the encrypted payload. As an example, Figure 1 shows the TCPLS control message structure that carries the TCP User Timeout [22] option. $TType$ is the true type of this record (TCP_OPTION), while its Type is set to $APPDATA$.

3.3 Multipath in a Modern Transport Protocol

3.4 Datastreams and TCP Connections

In TCPLS , each stream has its own cryptographic context. They use the same key but derive the blockcipher IV such that nonce-misuse cannot happen while the record sequence number within each stream starts at 0. Only one application-level key is used for N streams, for each direction. The reason behind this design choice is to avoid security degradation with the usage of multiple keys (by a factor k with k keys) [11].

Having a separate cryptographic context means that TCPLS can do concurrent encryption and decryption between streams while maintaining decryption correctness and security, and potentially also use this capability to process streams over multiple cores. Finally, if we have multiple streams over the same TCP connection, TCPLS does not explicitely know which received data belongs to which stream. To obtain this information, we either require to modify the associated information within TLS records to add a stream id (these associated data are not encrypted but the AEAD cipher authenticates them). This choice means potential middlebox interference, which we chose to avoid. The other option is to leverage the AEAD cipher to check the authentication tag of the incoming record until we find the stream that properly verifies the tag). This operation is lightweight: it does not require full decryption of the record because TLS 1.3 uses AEAD ciphers doing Encrypt then MAC (and MAC then Decrypt), and looking for the right stream needs to be performed once each time the application writes to another stream over the same TCP connection.

Note that, security-wise, each failed decryption is considered as a forgery attempt. However, we have large limits on the confidentiality and integrity with all AEAD ciphers [24, 36] before a successful iorgery may be considered as a non-negligeable probability.

TCPLS enables the client or the server to associate new TCP connections to an existing TCPLS connection. This is similar to what Mutipath TCP does [20, 43], but with some differences. First, Multipath TCP supports only one bytestream. Second, TCPLS does not suffer from the same security limitations as Multipath TCP. To secure the attachment of additional subflows, Multipath TCP hosts exchange keys in plaintext during the handshake [20, 21]. These keys are then used later to authenticate the attachment of subflows to a connection. An attacker that has observed the initial handshake can attach any subflow to an existing Multipath TCP connection [3].

TCPLS securely solves this "connection join" problem. For example, consider a client connecting to a dual-stack server. Figure 2 depicts the TLS messages exchanged. The client starts with a ClientHello. This includes the TCPLS extension to negotiate TCPLS . The server replies with a ServerHello containing several important and encrypted control information α. First, the server announces its IPv4 and IPv6 addresses. Second, it associates one connection identifier. This identifier uniquely identifies the connection on the server. Third, the server provides a list of cookies that enable the client to attach additional TCP connections to the TCPLS connection. To attach a new connection, e.g., using the server's

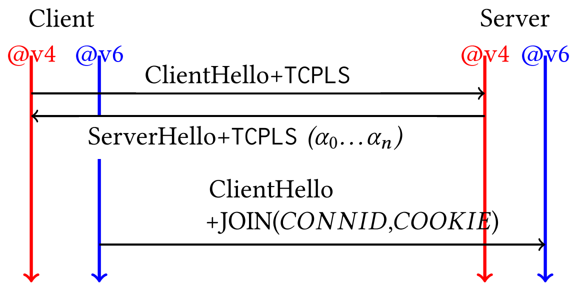

Figure 2: TCPLS supports the attachment of additional TCP connections to a TCPLS connection. Each α_i is encrypted with the handshake key.

IPv6 address, the client opens a TCP connection and sends a ClientHello message containing the connection identifier (*CONNID*) and one of the cookies supplied (*COOKIE*) by the server.

The Connection identifier allows the server to attach the new TCP connection to the right TCPLS session, assuming the received cookie is valid. The Connection identifier and the cookie play that same role as Multipath TCP's token. However, the cookie is longer, encrypted in the ServerHello message, and one-time use (i.e., when the server receives a valid cookie, it accepts the connection, attaches it to the right TCPLS session, and discard the cookie). Thanks to the cookies, the server can limit the number of TCP connections that a client can attach to a TCPLS connection. This prevents some denial of service attacks that are possible with Multipath TCP.

3.5 Secure Connection Closing

4 TCPLS PROTOTYPE[BUDGET=2.5P]

This section describes several of the possible benefits of TCPLS compared to keeping TCP and TLS isolated. We provide some use cases and experiment with the connection application-level connection migration offered by our API. Other user cases described in Section **??** are flagged to the roadmap and we expect them to further demonstrate the strength of a more intertwined TLS /TCP transport protocol.

Our current implementation offers: *(i)* An experimental API that wraps TLS and TCP and enables applications to handle multihoming, multipathing, and various transport layer mechanisms. *(ii)* An improved TCP extensibility mechanism that sends TCP options through the secure TCPLS channel. We currently support the TCP User Timeout option. Supporting another TCP option is only a matter of extending the sender's API and processing the option on the receiver side. TCPLS 's internal machinery can already send any TCP option during or after the handshake. *(iii)* The ability for the server to send eBPF bytecode over the secure channel to upgrade the client's TCP congestion control scheme or tune other TCP mechanisms [8, 53]. *(iv)* The support of parallel streams and multiplexing over TCP connections with different cryptographic context.

4.1 The TCPLS API

The API that applications use to interact with a protocol plays an important role in enabling them to leverage all the protocol features. The most popular API to interact with the transport layer remains the BSD socket API. Researchers and the IETF have explored new ways to expose a transport API [25, 28, 40, 48, 54].

In this spirit, application-level developers would only be required to configure a TCPLS context and register function callbacks. To illustrate TCPLS API's flexibility, we consider a simple use case inspired by Happy Eyeballs [47]. This technique is used by web browsers when interacting with dual-stack servers. They try to establish TCP connections using IPv4 and IPv6 and prefer the one that offers the lowest latency. This avoids problems when an address family is broken on a path but not the other and sometimes results in lower latency [4].

Figure 3 shows an example of our current API workflow. The API can handle explicit multipath techniques such as Happy Eyeball by chaining `tcpls_connect()` with an appropriate timeout of 50ms, as shown in the Figure. TCPLS lets the application explicitly choose the multipath mesh by calling several times `tcpls_connect(src, dest, timeout)`;. The application may configure callbacks to connection events that would occur within TCPLS , such as a connection establishment, a stream attachment, a multipath join, the reception of a TCP option to tune TCP, and more. When multiple streams are attached to multiple TCP connections, the application may configure various TCPLS behaviours. Among them, we support HOL-blocking avoidance, aggregation of bandwidth with multipathing, connection failover, and connection migration. Note that, HOL-blocking avoidance is incompatible with the aggregation of bandwidth with multipathing (the application can do either one but not both at the same time).

4.2 On TCP extensibility

The TCP specification limits the size of the entire TCP header (including options) to 64 bytes. Unfortunately, the TCP designers did not foresee that so many TCP extensions would be standardized. Today, the size of the TCP header becomes a constraint. For example, it severely limits the number of gaps that can be covered by selective acknowledgments. This gets worse with extensions such as Multipath TCP [20] that consume more space in the TCP header. The IETF has discussed

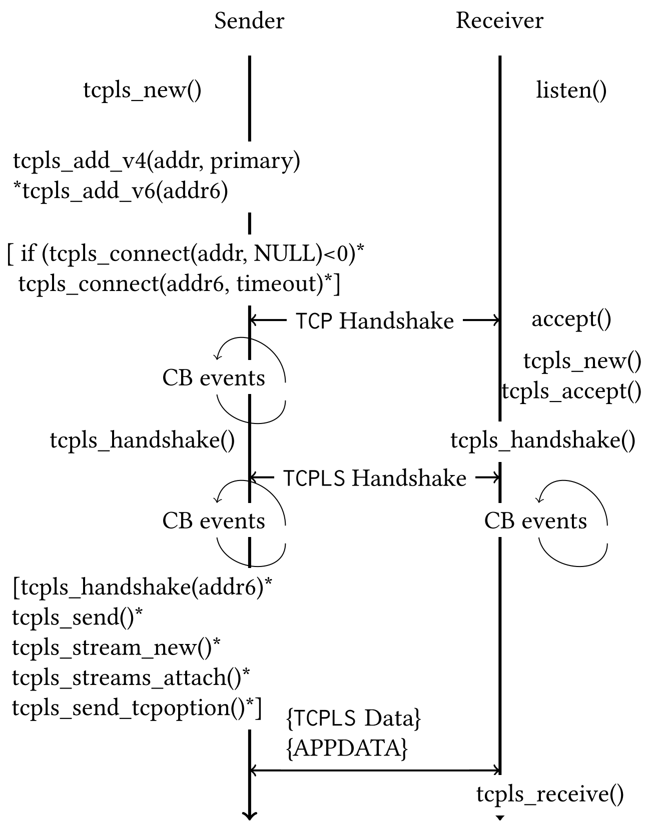

Figure 3: API Workflow example. * means optional call, [ ] means optional call flow, and { } means encrypted.

this problem for several years, but the latest attempt to solve it [52] has not yet been implemented by major TCP stacks.

TCPLS provides more space for some TCP options. First, with TCPLS , TCP options can be negotiated during the TLS handshake. Since the TLS messages are included in the TCP payload, there is more space to carry them. Another advantage of this approach is that the TCP options are secured by TLS . This implies that they cannot be modified by middleboxes. This could be an advantage, but could also prevent TCPLS from correctly working through some types of transparent TCP proxies.

Second, we can also carry TCP options inside TLS records. For example, we used this feature to implement the TCP User Time Out option [22]. A client can use this option to set the maximum value of the retransmission timer on a server. Linux TCP has a socket option that allows setting this timer locally, but it does not implement the option. With TCPLS , the client sends the option inside a TLS record, the server extracts it and performs the required `setsockopt`.

4.3 Multipathing

4.3.1 Application-level Connection Migration. Given the availability of multiple IP paths, connection migration might be a powerful tool to improve the application connection's reliability. We implement Connection migration and Failover as two distinct measures to handle two different inquiries: 1) The application expects to take advantage of multiple IP paths. 2) The application expects to be resilient to a network outage. In the first case, we implement connection migration and multipathing from a protocol viewpoint, as the same exchange of messages and API calls from TCPLS . It is left for the application to decide and program through the API calls whether it wants to move all the traffic from one path to another or split the traffic among the available paths according to any scheduling policy. The second inquiry focuses on simply configuring TCPLS to automatically move the traffic to another available IP-level path if a network outage is detected.

4.3.2 Failover. describing Failover

4.3.3 Data Aggregation. Describing orderings and schedulers

4.4 0-RTT Connections

5 EVALUATION[BUDGET=3P]

5.1 General Methodology

5.2 Capability Comparison Between QUIC, TCP and TLS /TCP

Table 1 compares the features supported by TCP , TLS /TCP , QUIC and TCPLS . QUIC and TCPLS are very similar in their capabilities. They mainly differ in their semantic. TCPLS 's semantic is to let the applications make the decision, and we design its API to fulfill this goal. That is, the meaning of TCPLS is to offer advanced, extensible and secure transport-layer functionalities on top of TCP , while exposing a simple but powerful API to let the application composes the properties its transport should have. One example is further demonstrated in Section 4.3.1, in which TCPLS 's simple API allows the application to take advantage of path aggregation (in multipath mode) and connection migration to obtain a smooth handover between networks.

Note that several of the features suggested by TCPLS are also suggested on TCP or QUIC via research works such as a new socket API for explicit multipath for TCP [25], or eBPF plugins in QUIC [15].

5.3 Application-level migration

Detailler pourquoi on a besoin du controle applicatif pour la migration, et à quels cas du monde réels ils s'appliquent

	TCP	TLS /TCP	QUIC	TCPLS
Transport reliability	✓	✓	✓	✓
Message conf. and auth.	✗	✓	✓	✓
Connection reliability	✗	✗	✓	(✓)
0-RTT	✓	(✗)	✓	✓
Session Resumption	✗	✓	✓	✓
Connection Migration	✗	✗	✓	✓
Application-exposed features				
Streams	✗	✗	✓	✓
Happy eyeballs	✗	✗	✗	✓
Explicit Multipath	✗	✗	✗	✓
App-level Con. migration	✗	✗	✗	✓
Pluginization	✗	✗	✗	(✓)
Resilience to HOL blocking	✗	✗	✓	(✓)
Secure Connection Closing	✗	✗	✓	(✓)

Table 1: Protocol features comparison. (✗) means that the feature is available, but not straightforward to use. (✓) means that the feature is partially available and under development.

Figure 4 shows the result of an Application-level connection migration demo using the API (i.e., it is left to the application to decide when to migrate, and we expose a simplistic code flow to perform it). In this experiment, we use an IPMininet network [32, 51] composed of a client and a server with a dual-stack of IPs. One path within the network is composed of OSPF routers with IPv4 only, and one path is composed of OSPF6 routers IPv6 only. We configure the bandwidth to 30Mbps, the lowest delay to the v4 link. Our application downloads a 60 MB file from a server and migrates to the v6 connection in the middle of the download.

Triggering the connection migration involves chaining 5 API calls: first, `tcpls_handshake()` configured with handshake properties announcing a JOIN over the v6 connection id. Then, the creation of a new stream `tcpls_stream_new()` for the v6 connection id, finally followed by the attachment of this new stream `tcpls_streams_attach()` and the secure closing of the v4 TCP connection using `tcpls_stream_close()`. Following these events, the server seamlessly switches the path while looping over `tcpls_send` to send the file content. Note that all the events trigger callbacks on the server side, to let the server react appropriately if other requirements need to be fulfilled.

TCPLS 's application connection migration takes advantage of multipath to offer a smooth handover to applications, which QUIC cannot do at the moment.

5.4 Failover

1) analyse du temps de recovery pour different type de cassure, et comparaison avec mptcp 2) discuter une propriété

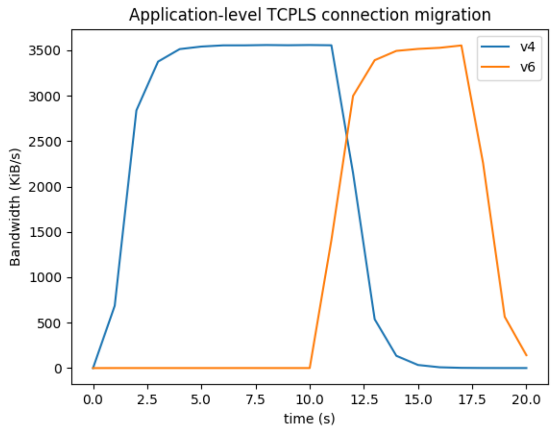

Figure 4: Application-level connection migration during a 60MB file download

de "connection reliability" => ca casse, on restabilise le plus vite possible 3) montrer que le path manager est important pour cette propriété, et que ce n'est pas encore au point pour mptcp, mpquic, etc

Mptcp overhead: 1.07449979782104449 TCPLS overhead: 1.0994282363439873 MPTCP overhead/TCPLS overhead: 0.97732599755138

5.5 Middlebox Interferences

Montrer que TCPLS handshake et JOIN handshake passent/-passent pas les middlebox, discuter les implications

5.6 Congestion Control Injection

Injecter un control de congestion, montrer que les perf s'améliorent

5.7 Performance

TCPLS vs QUIC vs TLS/TCP

Bar chart or/and tabular with throughput performance

6 CONCLUSION[BUDGET=0.5]

SOFTWARE ARTEFACTS

A TCPLS reference documentation and implementation is under active development. The current specifications and code are available on https://pluginized-protocols.org/tcpls, forked from a fast and full TLS 1.3 implementation written in C. Our TCPLS prototype adds about 5k lines of C code to `picotls` latest version based on the latest specification of TLS 1.3.

ACKNOWLEDGMENTS

REFERENCES

[1] Josh Aas, Richard Barnes, Benton Case, Zakir Durumeric, Peter Eckersley, Alan Flores-López, J Alex Halderman, Jacob Hoffman-Andrews,

James Kasten, Eric Rescorla, et al. 2019. Let's Encrypt: An Automated Certificate Authority to Encrypt the Entire Web. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2473–2487.

[2] Blake Anderson and David McGrew. 2019. TLS Beyond the Browser: Combining End Host and Network Data to Understand Application Behavior. In *Proceedings of the Internet Measurement Conference*. 379–392.

[3] M. Bagnulo. [n. d.]. Threat Analysis for TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6181. ([n. d.]). https://tools.ietf.org/html/rfc6181

[4] Vaibhav Bajpai and Jürgen Schönwälder. 2019. A longitudinal view of dual-stacked websites—failures, latency and happy eyeballs. *IEEE/ACM Transactions on Networking* 27, 2 (2019), 577–590.

[5] Andrea Bittau, Daniel B. Giffin, Mark J. Handley, David Mazieres, Quinn Slack, and Eric W. Smith. 2019. Cryptographic Protection of TCP Streams (tcpcrypt). RFC 8548. (May 2019). https://doi.org/10.17487/RFC8548

[6] Olivier Bonaventure and S Seo. 2016. Multipath TCP deployments. *IETF Journal* 12, 2 (2016), 24–27.

[7] David A. Borman, Robert T. Braden, and Van Jacobson. 1992. TCP Extensions for High Performance. RFC 1323. (May 1992). https://doi.org/10.17487/RFC1323

[8] Lawrence Brakmo. 2017. Tcp-bpf: Programmatically tuning tcp behavior through bpf. *NetDev 2.2* (2017).

[9] Łukasz Budzisz, Johan Garcia, Anna Brunstrom, and Ramon Ferrús. 2012. A taxonomy and survey of SCTP research. *ACM Computing Surveys (CSUR)* 44, 4 (2012), 1–36.

[10] Zimo Chai, Amirhossein Ghafari, and Amir Houmansadr. 2019. On the Importance of Encrypted-SNI (ESNI) to Censorship Circumvention. In *Free and Open Communications on the Internet*. USENIX. https://www.usenix.org/system/files/foci19-paper_chai_update.pdf

[11] Sanjit Chatterjee, Alfred Menezes, and Palash Sarkar. 2011. Another look at tightness. In *International Workshop on Selected Areas in Cryptography*. Springer, 293–319.

[12] Yuchung Cheng, Jerry Chu, Sivasankar Radhakrishnan, and Arvind Jain. 2014. TCP Fast Open. RFC 7413. (Dec. 2014). https://doi.org/10.17487/RFC7413

[13] David D Clark, Van Jacobson, John Romkey, and Howard Salwen. 1989. An analysis of TCP processing overhead. *IEEE Communications magazine* 27, 6 (1989), 23–29.

[14] Yong Cui, Zhiwen Liu, Hang Shi, Jie Zhang, Kai Zheng, and Wei Wang. 2020. *Deadline-aware Transport Protocol*. Internet-Draft draft-shi-quic-dtp-01. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-shi-quic-dtp-01 Work in Progress.

[15] Quentin De Coninck, François Michel, Maxime Piraux, Florentin Rochet, Thomas Given-Wilson, Axel Legay, Olivier Pereira, and Olivier Bonaventure. 2019. Pluginizing quic. In *Proceedings of the ACM Special Interest Group on Data Communication*. 59–74.

[16] Gregory Detal, Benjamin Hesmans, Olivier Bonaventure, Yves Vanaubel, and Benoit Donnet. 2013. Revealing Middlebox Interference with Tracebox. In *Proceedings of the 2013 ACM SIGCOMM conference on Internet measurement conference*. ACM.

[17] Dr. Taher Elgamal and Kipp E.B. Hickman. 1995. *The SSL Protocol*. Internet-Draft draft-hickman-netscape-ssl-00. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-hickman-netscape-ssl-00 Work in Progress.

[18] Sally Floyd. 2002. Inappropriate TCP Resets Considered Harmful. RFC 3360. (Aug. 2002). https://doi.org/10.17487/RFC3360

[19] Sally Floyd, Jamshid Mahdavi, Matt Mathis, and Dr. Allyn Romanow. 1996. TCP Selective Acknowledgment Options. RFC 2018. (Oct. 1996). https://doi.org/10.17487/RFC2018

[20] Alan Ford, Costin Raiciu, Mark J. Handley, and Olivier Bonaventure. 2013. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824. (Jan. 2013). https://doi.org/10.17487/RFC6824

[21] Alan Ford, Costin Raiciu, Mark J. Handley, Olivier Bonaventure, and Christoph Paasch. 2020. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 8684. (March 2020). https://doi.org/10.17487/RFC8684

[22] Fernando Gont and Lars Eggert. 2009. TCP User Timeout Option. RFC 5482. (March 2009). https://doi.org/10.17487/RFC5482

[23] Devashish Gosain, Anshika Agarwal, Sahil Shekhawat, H. B. Acharya, and Sambuddho Chakravarty. 2017. Mending Wall: On the Implementation of Censorship in India. In *SecureComm*. Springer. https://censorbib.nymity.ch/pdf/Gosain2017a.pdf

[24] F Günther, M. Thomson, and C.A. Wood. [n. d.]. Usage Limits on AEAD Algorithms. Internet-Draft. ([n. d.]). https://tools.ietf.org/html/draft-wood-cfrg-aead-limits-00 Work in Progress.

[25] Benjamin Hesmans and Olivier Bonaventure. 2016. An enhanced socket API for Multipath TCP. In *Proceedings of the 2016 applied networking research workshop*. 1–6.

[26] Ralph Holz, Johanna Amann, Abbas Razaghpanah, and Narseo Vallina-Rodriguez. 2019. The Era of TLS 1.3: Measuring Deployment and Use with Active and Passive Methods. *arXiv preprint arXiv:1907.12762* (2019).

[27] Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. 2011. Is it still possible to extend TCP?. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. 181–194.

[28] Tomas Hruby, Teodor Crivat, Herbert Bos, and Andrew S Tanenbaum. 2014. On Sockets and System Calls: Minimizing Context Switches for the Socket API. In *2014 Conference on Timely Results in Operating Systems (TRIOS 14)*.

[29] Christian Huitema. 2020. *Quic Timestamps For Measuring One-Way Delays*. Internet-Draft draft-huitema-quic-ts-02. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-huitema-quic-ts-02 Work in Progress.

[30] Jana Iyengar and Martin Thomson. 2020. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Internet-Draft draft-ietf-quic-transport-28. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-28 Work in Progress.

[31] Janardhan R Iyengar, Paul D Amer, and Randall Stewart. 2006. Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths. *IEEE/ACM Transactions on networking* 14, 5 (2006), 951–964.

[32] Mathieu Jadin, Olivier Tilmans, Maxime Mawait, and Olivier Bonaventure. 2020. Educational Virtual Routing Labs with IPMininet. In *ACM SIGCOMM Education Workshop 2020*.

[33] Eddie Kohler, Mark Handley, and Sally Floyd. 2006. Designing DCCP: Congestion control without reliability. *ACM SIGCOMM Computer Communication Review* 36, 4 (2006), 27–38.

[34] Platon Kotzias, Abbas Razaghpanah, Johanna Amann, Kenneth G Paterson, Narseo Vallina-Rodriguez, and Juan Caballero. 2018. Coming of age: A longitudinal study of tls deployment. In *Proceedings of the Internet Measurement Conference 2018*. 415–428.

[35] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. 2017. The quic transport protocol: Design and internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 183–196.

[36] Atul Luykx and Kenneth G Paterson. 2015. Limits on authenticated encryption use in TLS. *Personal webpage: http://www. isg. rhul. ac. uk/˜kp/TLS-AEbounds. pdf* (2015).

[37] Alberto Medina, Mark Allman, and Sally Floyd. 2005. Measuring the Evolution of Transport Protocols in the Internet. *SIGCOMM Comput. Commun. Rev.* (April 2005), 37–52. https://doi.org/10.1145/1064413.1064418

[38] François Michel, Quentin De Coninck, and Olivier Bonaventure. 2019. QUIC-FEC: Bringing the benefits of Forward Erasure Correction to QUIC. In *2019 IFIP Networking Conference (IFIP Networking)*. IEEE, 1–9.

[39] Mehrab Bin Morshed, Michaelanne Dye, Syed Ishtiaque Ahmed, and Neha Kumar. 2017. When the Internet Goes Down in Bangladesh. In *Computer-Supported Cooperative Work and Social Computing*. ACM. https://nehakumardotorg.files.wordpress.com/2014/03/p1591-bin-morshed.pdf

[40] Tommy Pauly, Brian Trammell, Anna Brunstrom, Gorry Fairhurst, Colin Perkins, Philipp S. Tiesel, and Christopher A. Wood. 2020. *An Architecture for Transport Services*. Internet-Draft draft-ietf-taps-arch-07. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-taps-arch-07 Work in Progress.

[41] Michele Polese, Federico Chiariotti, Elia Bonetto, Filippo Rigotto, Andrea Zanella, and Michele Zorzi. 2019. A survey on recent advances in transport layer protocols. *IEEE Communications Surveys & Tutorials* 21, 4 (2019), 3584–3608.

[42] Jon Postel. 1981. Transmission Control Protocol. RFC 793. (Sept. 1981). https://doi.org/10.17487/RFC0793

[43] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. 2012. How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. 399–412.

[44] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. (Aug. 2018). https://doi.org/10.17487/RFC8446

[45] Jim Roskind. 2013. QUIC, Quick UDP Internet Connections. https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/preview. (2013).

[46] Robert M Sanders and Alfred C Weaver. 1990. The Xpress transfer protocol (XTP)— a tutorial. *ACM SIGCOMM Computer Communication Review* 20, 5 (1990), 67–80.

[47] David Schinazi and Tommy Pauly. 2017. Happy Eyeballs Version 2: Better Connectivity Using Concurrency. RFC 8305. (Dec. 2017). https://doi.org/10.17487/RFC8305

[48] Philipp S Schmidt, Theresa Enghardt, Ramin Khalili, and Anja Feldmann. 2013. Socket intents: Leveraging application awareness for multi-access connectivity. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. 295–300.

[49] Randall R. Stewart. 2007. Stream Control Transmission Protocol. RFC 4960. (Sept. 2007). https://doi.org/10.17487/RFC4960

[50] Ian Swett, Marie-Jose Montpetit, Vincent Roca, and François Michel. 2020. *Coding for QUIC*. Internet-Draft draft-swett-nwcrg-coding-for-quic-04. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-swett-nwcrg-coding-for-quic-04 Work in Progress.

[51] Olivier Tilmans and Mathieu Jadin. [n. d.]. IPMininet. ([n. d.]). https://github.com/cnp3/ipmininet, Accessed Feb-20-2020.

[52] Joseph D. Touch and Wesley Eddy. 2018. *TCP Extended Data Offset Option*. Internet-Draft draft-ietf-tcpm-tcp-edo-10. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-tcp-edo-10 Work in Progress.

[53] Viet-Hoang Tran and Olivier Bonaventure. 2019. Beyond socket options: making the Linux TCP stack truly extensible. In *2019 IFIP Networking Conference (IFIP Networking)*. IEEE, 1–9.

[54] Michael Tüxen, Vladislav Yasevich, Peter Lei, Randall R. Stewart, and Kacheong Poon. 2011. Sockets API Extensions for the Stream Control Transmission Protocol (SCTP). RFC 6458. (Dec. 2011). https://doi.org/10.17487/RFC6458

[55] Tobias Viernickel, Alexander Froemmgen, Amr Rizk, Boris Koldehofe, and Ralf Steinmetz. 2018. Multipath QUIC: A deployable multipath transport protocol. In *2018 IEEE International Conference on Communications (ICC)*. IEEE, 1–7.

[56] Zhaoguang Wang, Zhiyun Qian, Qiang Xu, Zhuoqing Mao, and Ming Zhang. 2011. An untold story of middleboxes in cellular networks. *ACM SIGCOMM Computer Communication Review* 41, 4 (2011), 374–385.

[57] Nicholas Weaver, Robin Sommer, and Vern Paxson. 2009. Detecting Forged TCP Reset Packets.. In *NDSS*.

[58] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. 2011. Design, Implementation and Evaluation of Congestion Control for Multipath TCP.. In *NSDI*, Vol. 11. 8–8.

[59] Xing Xu, Yurong Jiang, Tobias Flach, Ethan Katz-Bassett, David Choffnes, and Ramesh Govindan. 2015. Investigating transparent web proxies in cellular networks. In *International Conference on Passive and Active Network Measurement*. Springer, 262–276.