Workgroup: QUIC Working Group
Internet-Draft: draft-piraux-quic-plugins-00
Published: 9 March 2020
Intended Status: Experimental
Expires: 10 September 2020
Authors: M. Piraux
UCLouvain
Q. De Coninck
UCLouvain
F. Michel
UCLouvain
F. Rochet
UCLouvain
O. Bonaventure
UCLouvain

QUIC Plugins

Abstract

The extensibility of Internet protocols is a key factor to their success. Yet, their implementations are often not designed with agility in mind. In this document, we leverage the features of the QUIC protocol and propose a solution to dynamically extend QUIC implementations. Our solution relies on QUIC Plugins that allow tuning and extending the QUIC protocol on a per-connection basis. These platform-independent plugins are executed inside a sandboxed environment which can be included in QUIC implementations. We describe how such plugins can be used in different use cases.

This document is a straw-man proposal. It aims at sparking discussions on the proposed approach.

Note to Readers

Discussion of this document takes place on the QUIC Working Group mailing list (quic@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/quic/.

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

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 10 September 2020.

Table of Contents

1. Introduction

Internet hosts rely on protocols to efficiently exchange information. Most protocols are designed assuming a layered model. In such a layered protocol model, a protocol implementation is often represented as a black-box that uses the service provided by the underlying layer to offer an enhanced Application Programming Interface (API) to the upper layer. This is illustrated in Figure 1.

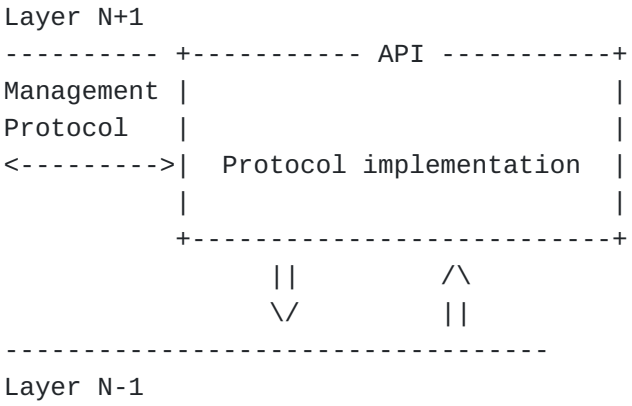

Figure 1: A protocol implementation exposes an API to the upper layer

Different protocols expose different APIs. In the transport layer, the socket API is the most popular one. It was originally defined with the TCP and UDP protocols in mind, but has been extended to support SCTP [RFC6458] and Multipath TCP [RFC6897]. The TAPS working group is currently defining new APIs for the transport services [I-D.ietf-taps-interface].

An important benefit of having standardized APIs is that if two implementations expose the same API, it is possible to replace one by the other without changing anything in the upper layers.

Besides exposing an API to the layer above, many protocol implementations also expose an API to management protocols such as SNMP, IPFIX or NETCONF. For example, a TCP implementation exposes the variables and tables defined in the TCP part of the MIB-2 [RFC1213][RFC4022]. There is ongoing work in developing a YANG model for TCP implementations [I-D.scharf-tcpm-yang-tcp]. This is illustrated in the left part of Figure 1.

These management APIs expose abstract configuration parameters and statistics about the key operations performed by a protocol implementation to offer more control to the upper layer. They have been useful in configuring and operating a wide range of protocol implementations. As different implementations expose the same abstraction, it becomes possible for operators to configure and manage different implementations by using the same tool in a unified manner.

A protocol implementation exchanges messages with other implementations by leveraging the service provided by the underlying layer. For example, a TCP implementation interacts through the socket API and exchanges TCP segments with remote hosts through the underlying IP protocol. Protocol designers usually define transport and application protocols in two parts:

*a series of messages that are encoded using a specific format.

*a Finite State Machine that defines how hosts react to commands from the layer above or to the reception of a specific message.

This model has been used to represent a wide range of Internet protocols. Typically, the Finite State Machine and the syntax of the messages are described in a single or a series of documents that are heavily discussed within IETF working groups before reaching a consensus and getting a final specification. The ongoing work on the finalization of the first version of the QUIC specification is a recent example of such efforts [I-D.ietf-quic-transport].

Once the first stable version of the specification of a protocol has been approved, IETF working groups typically observe the deployment of the protocol and improve it based on the received feedback. If the protocol is successful, it often triggers suggestions for extensions or improvements. These extensions are important for these successful protocols, but they often take a lot of time to be deployed. Experience with TCP shows that extensions, such as selective acknowledgments [RFC2018], support for large windows and timestamps [RFC7323] or more recently Multipath TCP [RFC6824] took more than a decade to be widely deployed. There are several considerations that can explain the difficulty of deploying TCP extensions, ranging from the fact that TCP is often part of operating system kernels, which evolve at a slower pace than applications, to middlebox interference.

Looking at other IETF protocols, we rarely observe successful protocols that have not been extended over the years. Thus, the need for extensibility could be seen as an invariant for a successful Internet protocol. In addition to the extensions that were accepted by the IETF and eventually deployed, there are many other extensions

that correspond to specific applications or more restricted use cases. These extensions would be very valuable in specific environments, but their proponents never managed to convince the relevant IETF working group and implementers of their benefits.

Besides the protocol extensions, we also observe that there are some protocol behaviors that can be difficult to precisely express using a set of parameters that are exchanged inside a message. Here are two recent examples that illustrate this difficulty.

A first example is the transmission of acknowledgments in reliable transport protocols. There is a trade-off between the feedback provided by the acknowledgments and the resources (bandwidth and CPU) required to generate and process them. Various heuristics have been proposed in TCP to generate these ACKs [RFC1122],[RFC5681], [RFC3449],[RFC5690]. These heuristics are deployed independently on receivers, but for some of them the senders need to adapt by at least taking into account the fact that some acknowledgments were delayed while measuring the round-trip-time. A similar discussion has started for QUIC. Given the flexibility of QUIC, researchers have proposed to define an acknowledgment strategy as a set of parameters that are exchanged in a new QUIC frame over each connection [I-D.fairhurst-quic-ack-scaling],[I-D.iyengar-quic-delayed-ack]. This brings more flexibility than in TCP where the limited size of the header made it impossible to exchange such information.

A second example in the application layer is the support for stream priorities in HTTP/2 [RFC7540]. Since HTTP/2 provides parallel streams, some application developers have expressed their need for prioritizing some streams over others. The HTTP/2 protocol defines such priorities, but they are not widely used and some have proposed to deprecate them [I-D.peon-httpbis-h2-priority-one-less]. In spite of this, a proposal for stream prioritization for HTTP/3 already exists [I-D.kazuho-httpbis-priority]. [LNBIP2020] evaluates stream scheduling for HTTP/3 and claims that performances are heavily impacted by the adopted stream scheduling.

These examples illustrate the difficulty of precisely expressing complex behaviors in a few parameters that are exchanged inside packets.

In this document, we propose a different approach to specify and implement protocols to better address the extensibility requirement. We focus on the QUIC protocol in this document, but similar ideas could be applied to other IETF protocols or networking systems. We leverage the recent results in extending operating system kernels [eBPF] or web-browsers [WebAssembly] with virtual machines that execute bytecodes to propose a new approach to define complex

behaviors inside protocols. We first describe the general architecture of the proposed approach in Section 2. We then provide a few examples showing how such an approach could be applied to the next version of the QUIC protocol in Section 3.

2. Dynamically extensible protocol implementations

Experience with successful Internet protocols shows that once a protocol gets (widely) deployed, it attracts new use cases and proposals to extend and improve it. Most of the key Internet protocols, including IP, TCP, HTTP, DNS, BGP, OSPF, IS-IS, ... have been improved over the years. Today, the developer of an implementation of any of these protocols need to consult dozens of RFCs to find their complete specification.

Despite the importance of extensions to those key Internet protocols, we still do not design them under the assumption that they will evolve over decades and that their implementations should be made agile. In this document, we propose a new organization for protocol implementations. Instead of trying to pack as many features as possible inside a protocol implementation that is considered as a blackbox, we consider a protocol implementation as an open system which can be extended to support new features in a safe and agile manner. Our vision is that such an implementation exposes an internal Plugin API which can be leveraged by code extensions that we call Plugins in this document. This is illustrated in Figure 2.

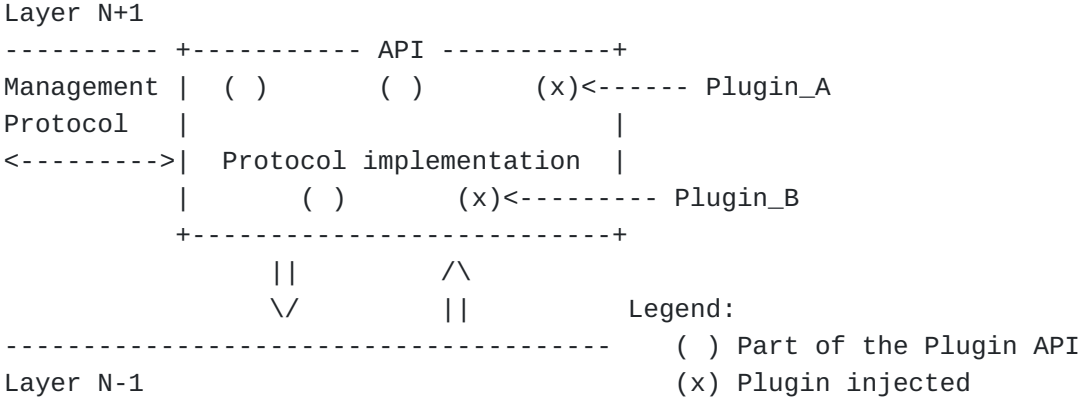

Figure 2: A pluginized protocol implementation exposes a Plugin API that enables plugins to dynamically extend its operation

These Plugins extend the protocol by modifying its messages or its Finite State Machine. Consider Figure 3 as an example which illustrates a simple receiver with three states, i.e. Receiving, Data rcvd and Ack needed. This receiver waits until either two data packets have been received or 10 milliseconds have passed after receiving a data packet before sending an acknowledgment. Its

implementation consists of three actions, receive(), send_ack() and wait().

An application using this receiver could benefit from tuning this acknowledgment policy. For instance, another state could be chained between Data rcvd and Ack needed, so that acknowledgments can be batched for more than two data packets. Similarly, a more advanced heuristic could be added to generate acknowledgments, e.g. taking into account the size of data packets to detect transmission tails.

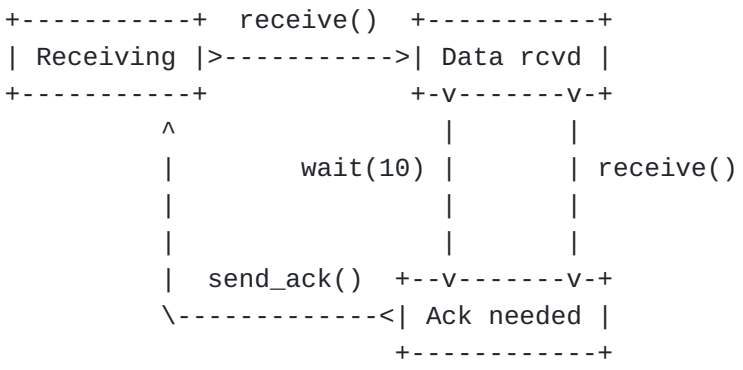

Figure 3: Finite State Machine of a simple receiver

Using this Plugin API, a protocol can be extended to add new messages and to modify its Finite State Machine, i.e. modify its states and transitions. This document defines such protocol as a pluginized protocol. Figure 4 illustrates how this simple receiver could be extended. In this example a new state is added so that a single acknowledgment can be sent for three received data packets.

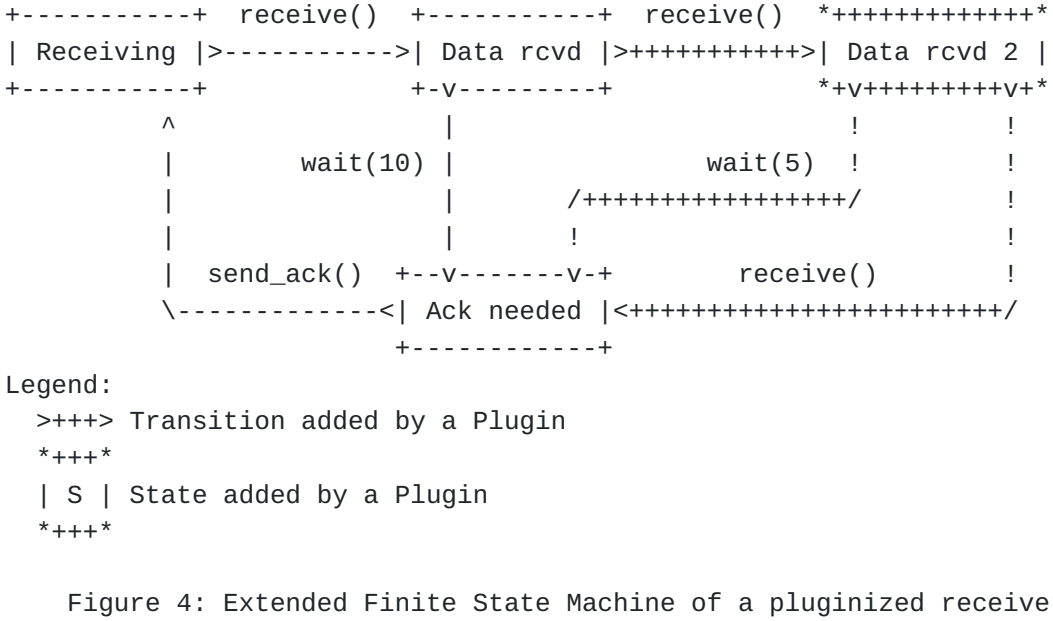

Figure 4: Extended Finite State Machine of a pluginized receiver

Due to space constraints, the case of adding a new action is not depicted in Figure 4. In this example, a new action data_not_full() could be added by a Plugin. This action could be taken whenever the last received data packet is not full, indicating the end of a series of packets. Transitions from the states Data rcvd and Data rcvd 2 to the state Ack needed with this action could be added by this Plugin.

2.1. Requirements

There are different ways of implementing the idea of dynamically extending protocols by using plugins. We list here some requirements that any solution should support:

REQ1: Different implementations should expose the same Plugin API, e.g. as different implementations expose the same SNMP MIB.

REQ2: It should be possible to dynamically attach a plugin to one instance of an implementation. For instance, for a connection-oriented protocol, it should be possible to associate a plugin to a given connection. Different connections could have different sets of plugins.

REQ3: It should be possible to execute the same plugin on different interoperable implementations of the same protocol.

REQ4: It should be possible for a protocol implementation to restrict the operations that a given plugin can execute.

REQ5: Plugins should be sandboxed and the application should be safe using them, for instance regarding memory corruption and runtime traps.

One possible way to realize this new architecture is to include a virtual machine inside each protocol implementation and expose a small Plugin API accessible through the virtual machine. Several efficient virtual machines have been proposed and used in related environments [eBPF] [WebAssembly]. They provide a sandbox controlling the operations a plugin can execute and the memory that it can access. Since the same virtual machine can be provided on different platforms, it becomes possible to execute the same plugin on different implementations of a given protocol exposing the same Plugin API.

The idea of extending protocols through plugins can be applied to different Internet protocols. In this document, we focus on adding plugins to QUIC since it is a recent and flexible protocol that includes useful security features. A similar approach has been applied to OSPF and BGP [ICNP] and partially to TCP [TCP-Options-BPF]. Other networking systems, such as the Tor network [FAN] or

Cryptocurrency networks, may also benefit from extending their protocols through plugins. In some cases, the extensibility through plugins may help to solve fundamental security issues [DROPPING] linked to both compliance to the Postel principle and slow deployment processes.

3. Pluginizing QUIC

Conceptually, we break down a QUIC implementation execution flow into generic subroutines. These are specified functions called protocol operations. These protocol operations implement a given part of the QUIC protocol, for example the acknowledgment generation or the computation of the round-trip-time. Some are generic and depend on a parameter, for instance parsing a QUIC frame is a generic operation that depends on the type of QUIC frame. This version of the document does not elaborate exhaustively on the protocol operations composing a Pluginized QUIC implementation. The next versions of this document will work on defining a set of protocol operations.

A QUIC Plugin consists of platform-independent bytecode which modifies or extends the behavior of a QUIC implementation. Adding the functionality of a QUIC Plugin consists in adding or replacing a set of protocol operations implemented by this bytecode. This action of adding a QUIC Plugin to a QUIC connection is referred to as injecting a QUIC Plugin. Injecting a plugin is limited to a given QUIC connection.

Its bytecode is run inside a sandboxed execution environment. It has restricted access to the state of a QUIC connection through the Plugin API.

The scope of a QUIC Plugin is restricted by both the limitations of this execution environment, e.g. in terms of instruction set and quantity, and by the surface exposed by the Plugin API, e.g. the quantity of state that can be read from or written to by a plugin. The Plugin API also defines a set of protocol operations to which QUIC Plugins can be injected. For instance, a QUIC implementation might restrict QUIC Plugins injection to its acknowledgment generation policy, e.g. the protocol operation deciding whether sending an ACK frame is needed.

A QUIC Plugin can be injected by several means. The application can inject plugins to tune its underlying QUIC connection. QUIC peers can exchange and inject plugins over a QUIC connection, as described in Section 4. Users can set a default configuration injecting plugins on their devices.

Considering again Figure 3 applied to QUIC Plugins, the actions are the protocol operations. The states of the FSM are defined by the QUIC connection state, which can be modified and extended during the execution of a protocol operation through the Plugin API. The transitions are calls to protocol operations.

4. Exchanging QUIC Plugins

Injecting QUIC Plugins locally allows the application to tune the underlying QUIC implementation to its needs. But some use-cases requires adapting the peer behavior. In those cases, being able to exchange plugins helps to fill this gap.

QUIC offers both data multiplexing and encryption. Using those mechanisms, the QUIC Plugins used for a given connection could be safely transferred over this connection in a new dedicated stream, akin to the crypto stream. This does not impact the application data transfer, as illustrated in Figure 5. In this example, an HTTP/3 request is interleaved with the transfer from the server to the client of a QUIC Plugin controlling the acknowledgment policy.

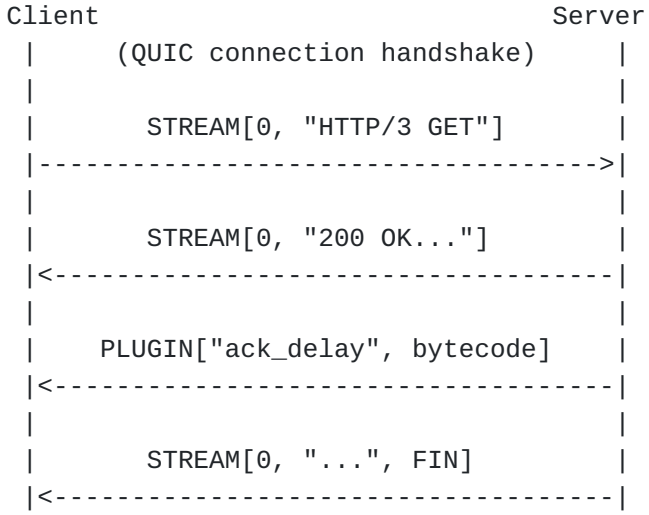

Figure 5: Interleaving a QUIC Plugin transfer over a QUIC connection

A local policy could restrict the type of plugins that can be exchanged and injected, e.g. in terms of connection state accessed and in terms of protocol operations affected. This version of document does not describe further how the negotiation of QUIC Plugins takes place.

QUIC Plugins could also be transferred out of band. The application using QUIC can then inject these plugins locally.

5. Examples of use-cases

There exist several cases in which being able to modify the behavior of a QUIC peer, either locally or remotely, is beneficial. This first version of the document focus on enabling the extension of simple QUIC mechanisms. Three of them are documented in this section. Contributions to this document regarding new uses-cases are welcomed.

5.1. Tunable acknowledgments policy

The large diversity of Internet paths also counts networks involving a significant path asymmetry. Those paths require the receiver to adapt its feedback rate to the sender. The performance of transport protocols such as TCP on those paths has been studied in depth and reported in [RFC3449]. A method for controlling the rate of receiver feedback of TCP, i.e. the rate of ACKs, has been proposed in [RFC5690].

Proposals for controlling the acknowledgment policy of QUIC already exist. [I-D.fairhurst-quic-ack-scaling] proposes a change to the default policy for asymmetric paths. [I-D.iyengar-quic-delayed-ack] describes a QUIC extension enabling a QUIC endpoint to control the acknowledgment policy of its peer. For that purpose, they model the acknowledgment policy depending on a few parameters and introduce a new QUIC frame to signal new values for those parameters.

A QUIC Plugin could allow implementing a new acknowledgment with a fine granularity. For example, in the case of an application that generates bursty traffic, such as a real-time video streaming application, a QUIC Plugin allows embedding application knowledge, i.e. the characteristics of such bursts, inside the acknowledgment generation policy. Exchanging and injecting this plugin allows controlling the other peer behavior.

5.2. Pluggable congestion controller

There exists many congestion control algorithms. Each of them has been designed for a given context, i.e. a range of applications and a range of Internet paths. For instance, NewReno [RFC6582] has been designed for optimizing the web use-case on common Internet paths. Westwood [Westwood] is a modification of NewReno to better accommodate Internet paths with a high bandwidth-delay product, such as satellite links.

Efforts to restructure congestion controllers within a common framework have been presented in past works such as [CCP]. Such a framework eases the development and maintenance of those algorithms. It also enables rapid prototyping and A/B testing.

[TCP-Options-BPF] proposes a new TCP Option, leveraging the TCP-BPF framework, to negotiate the congestion controller to use. The QUIC specification does not specify a similar mechanism. Negotiating the congestion controller used allows one endpoint to tune the other, provided that it implements the algorithm. QUIC Plugins could allow the application to directly plug the required congestion controller and to exchange it with the other peer. This flexibility allows the application to choose the best congestion controller for its requirements.

5.3. Application-driven stream scheduling

QUIC streams allow the application to multiplex several bytestreams over a single QUIC connection. Yet, the QUIC specification does not provide a mechanism for exchanging prioritization information nor for indicating the relative priority of streams. As described in Section 2.3 of [I-D.ietf-quic-transport], "A QUIC implementation should provide ways in which an application can indicate the relative priority of streams". A QUIC implementation could allow QUIC Plugins to extend or override its stream scheduler.

For other applications using QUIC with a broad range of requirements, a flexible approach for defining the stream scheduling policy is key to best fit their needs. QUIC Plugin offer a flexible way to embed application knowledge inside the QUIC implementation.

5.4. More advanced QUIC extensions

Exposing more protocol operations through the API proposed to plugins by the QUIC implementation allows implementing more advanced QUIC Plugins. Each protocol operation offers flexibility over the QUIC implementation. [PQUIC] demonstrates how this approach can be used to implement Multipath QUIC [I-D.deconinck-quic-multipath] and [QUIC-FEC] entirely using plugins.

6. QUIC Plugins Authenticity

When exchanging and injecting QUIC Plugins, guaranteeing there authenticity and safety is important. This section describes two possible approaches for this purpose, the first guarantees the authenticity of the QUIC Plugins, the second guarantee both their authenticity and an open set of security properties with regard to QUIC Plugins.

6.1. Central Authorities

This first approach leverages the central authorities commonly used to secure HTTPS. In this approach, each QUIC Plugin could be associated to some level of trust regarding its origin. A QUIC Plugin may be authenticated using a certificate, itself certified by

a central authority. Consequently, a QUIC implementation supporting QUIC plugins may restrict their exchange and only accept plugins authenticated using the same certificate used for establishing the QUIC connection. This approach only guarantees the authenticity of a QUIC Plugin.

6.2. Plugin Transparency

This second approach is presented in the [PQUIC] research paper, and suggests going beyond the restrictive approach of a centralized trust model. The centralized trust model obliges the server to update manually their list of supported plugins. It also prevents the client from injecting a plugin that the server has not marked as supported, e.g. because the server is unaware of its existence, or because its list has not been updated.

The suggested design, called Plugin Transparency, proposes a methodology to transparently distribute plugins created by independent developers and verified by freely selected plugin validators. Those validators endorse verifying some publicly known safety or security properties. A QUIC endpoint can announce a set of conditions to accept a plugin as a first order logic formula bound to plugin validators. Whenever the other peer is willing to inject a plugin, it sends a (unforgeable) proof fulfilling the requirements expressed by this logic formula. If the requirements are met, then the endpoint may safely accept the plugin and could update its list of supported plugin. Compared to the central authority approach, supported plugins are updated as part of the protocol design or as a consequence of any change to the default logic formula bound to plugin acceptance.

6.3. Comparing Certificate Transparency and Plugin Transparency

Certificate Transparency (CT) [RFC6962] is an attempt to address the structural issues hidden within the central trust assumption and prevent mistakes, rogue certificates and rogue authorities from weakening the system. Plugin Transparency bears similarities to Certificate Transparency. First, its motivations are drawn from the same conclusions regarding the danger of central trust models. Second, similar to CT, it is based on distributing trust assumptions to secure the system. However, Plugin Transparency offers stronger properties and eliminates the independent monitoring entities which hold the resource endowment to continuously monitor the CT log on the behalf of certificate owners. Indeed, our design offers the independent developers checking for spurious plugins in O(log(N)) with N the size of the log (instead of O(N) in CT's design). Our design also offers secure human-readable plugin names that unambiguously authenticate them and non-equivocation from rogue plugin validators. Our design is more resilient to failure by

offering several validators that can be trusted within the logic formula. For example, a QUIC peer may request a proof bound to any combination of plugin validators. The detail of Plugin Transparency, including performance considerations and security proofs are available in [PQUIC].

7. Security Considerations

The next versions of this document will elaborate on security considerations following the guidelines of [RFC3552]. Moreover, this document will consider privacy as part of those considerations.

8. IANA Considerations

This document has no IANA actions.

9. Informative References

[PQUIC] De Coninck, Q., Michel, F., Piraux, M., Rochet, F., Given-Wilson, T., Legay, A., Pereira, O., and O. Bonaventure, "Pluginizing QUIC", Proceedings of SIGCOMM'19 , August 2019, <https://pquic.org>.

[I-D.kazuho-httpbis-priority] Oku, K. and L. Pardue, "Extensible Prioritization Scheme for HTTP", Work in Progress, Internet-Draft, draft-kazuho-httpbis-priority-04, 20 November 2019, <http://www.ietf.org/internet-drafts/draft-kazuho-httpbis-priority-04.txt>.

[LNBIP2020] Marx, R., De Decker, T., Quax, P., and W. Lamotte, "Resource Multiplexing and Prioritization in HTTP/2 over TCP versus HTTP/3 over QUIC", 2020.

[CCP] Narayan, A., Cangialosi, F., Raghavan, D., Goyal, P., Narayana, S., Mittal, R., Alizadeh, M., and H. Balakrishnan, "Restructuring Endpoint Congestion Control", Proc. SIGCOMM 2018 , August 2018.

[I-D.ietf-quic-transport] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-27, 21 February 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-27.txt>.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <https://www.rfc-editor.org/info/rfc1122>.

[RFC1213] McCloghrie, K. and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", STD 17, RFC 1213, DOI 10.17487/RFC1213, March 1991, <https://www.rfc-editor.org/info/rfc1213>.

[RFC4022] Raghunarayan, R., Ed., "Management Information Base for the Transmission Control Protocol (TCP)", RFC 4022, DOI 10.17487/RFC4022, March 2005, <https://www.rfc-editor.org/info/rfc4022>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/info/rfc5681>.

[RFC3449] Balakrishnan, H., Padmanabhan, V., Fairhurst, G., and M. Sooriyabandara, "TCP Performance Implications of Network Path Asymmetry", BCP 69, RFC 3449, DOI 10.17487/RFC3449, December 2002, <https://www.rfc-editor.org/info/rfc3449>.

[RFC5690] Floyd, S., Arcia, A., Ros, D., and J. Iyengar, "Adding Acknowledgement Congestion Control to TCP", RFC 5690, DOI 10.17487/RFC5690, February 2010, <https://www.rfc-editor.org/info/rfc5690>.

[RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", RFC 6962, DOI 10.17487/RFC6962, June 2013, <https://www.rfc-editor.org/info/rfc6962>.

[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/info/rfc7540>.

[RFC6897] Scharf, M. and A. Ford, "Multipath TCP (MPTCP) Application Interface Considerations", RFC 6897, DOI 10.17487/RFC6897, March 2013, <https://www.rfc-editor.org/info/rfc6897>.

[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>.

[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>.

[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>.

[RFC6458] Stewart, R., Tuexen, M., Poon, K., Lei, P., and V. Yasevich, "Sockets API Extensions for the Stream Control Transmission Protocol (SCTP)", RFC 6458, DOI 10.17487/RFC6458, December 2011, <https://www.rfc-editor.org/info/rfc6458>.

[RFC6582] Henderson, T., Floyd, S., Gurtov, A., and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 6582, DOI 10.17487/RFC6582, April 2012, <https://www.rfc-editor.org/info/rfc6582>.

[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/info/rfc3552>.

[I-D.fairhurst-quic-ack-scaling]
Fairhurst, G., Custura, A., and T. Jones, "Changing the Default QUIC ACK Policy", Work in Progress, Internet-Draft, draft-fairhurst-quic-ack-scaling-01, 5 March 2020, <http://www.ietf.org/internet-drafts/draft-fairhurst-quic-ack-scaling-01.txt>.

[I-D.iyengar-quic-delayed-ack]
Iyengar, J. and I. Swett, "Sender Control of Acknowledgement Delays in QUIC", Work in Progress, Internet-Draft, draft-iyengar-quic-delayed-ack-00, 23 January 2020, <http://www.ietf.org/internet-drafts/draft-iyengar-quic-delayed-ack-00.txt>.

[I-D.deconinck-quic-multipath]
Coninck, Q. and O. Bonaventure, "Multipath Extensions for QUIC (MP-QUIC)", Work in Progress, Internet-Draft, draft-deconinck-quic-multipath-04, 5 March 2020, <http://www.ietf.org/internet-drafts/draft-deconinck-quic-multipath-04.txt>.

[I-D.peon-httpbis-h2-priority-one-less]
Thomson, M. and R. Peon, "Deprecation of HTTP/2 Priority Signaling Hints", Work in Progress, Internet-Draft, draft-peon-httpbis-h2-priority-one-less-00, 25 July 2019,

<http://www.ietf.org/internet-drafts/draft-peon-httpbis-h2-priority-one-less-00.txt>.

[I-D.ietf-taps-interface]
Trammell, B., Welzl, M., Enghardt, T., Fairhurst, G., Kuehlewind, M., Perkins, C., Tiesel, P., Wood, C., and T. Pauly, "An Abstract Application Layer Interface to Transport Services", Work in Progress, Internet-Draft, draft-ietf-taps-interface-05, 4 November 2019, <http://www.ietf.org/internet-drafts/draft-ietf-taps-interface-05.txt>.

[I-D.scharf-tcpm-yang-tcp]
Scharf, M., Murgai, V., and M. Jethanandani, "YANG Model for Transmission Control Protocol (TCP) Configuration", Work in Progress, Internet-Draft, draft-scharf-tcpm-yang-tcp-04, 24 February 2020, <http://www.ietf.org/internet-drafts/draft-scharf-tcpm-yang-tcp-04.txt>.

[QUIC-FEC] Michel, F., De Coninck, Q., and O. Bonaventure, "QUIC-FEC: Bringing the benefits of Forward Erasure Correction to QUIC", IFIP Networking 2019 , May 2019.

[TCP-Options-BPF] Tran, VH. and O. Bonaventure, "Beyond socket options: making the Linux TCP stack truly extensible", IFIP Networking 2019 , May 2019.

[eBPF] Matt Fleming, ., "A thorough introduction to eBPF", Linux Weekly News , December 2017, <https://old.lwn.net/Articles/740157/>.

[Westwood] Casetti, C., Gerla, M., Mascolo, S., Sanadidi, M.Y., and R. Wang, "TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks", Wireless Networks 8 , 2002.

[WebAssembly] Haas, A., Rossberg, A., Schuff, D.L., Titzer, B.L., Holman, M., Gohman, D., Wagner, L., Zakai, A., and J.F. Bastien, "Bringing the web up to speed with WebAssembly", Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation , June 2017.

[ICNP] Wirtgen, T., Denos, C., De Coninck, Q., Jadin, M., and O. Bonaventure, "The Case for Pluginized Routing Protocols", 2019 IEEE 27th International Conference on Network Protocols (ICNP) , October 2019.

[FAN] Rochet, F., Bonaventure, O., and O. Pereira, "Flexible Anonymous Network", HotPETs , July 2019.

[DROPPING]
Rochet, F. and O. Pereira, "Dropping on the Edge: Flexibility and Traffic Confirmation in Onion Routing Protocols", Proceedings on Privacy Enhancing Technologies , July 2018.

Acknowledgments

The authors of Pluginizing QUIC are thanked again for their work that initiated this proposal. This work was partially supported by the MQUIC project.

Authors' Addresses

Maxime Piraux
UCLouvain

Email: maxime.piraux@uclouvain.be

Quentin De Coninck
UCLouvain

Email: quentin.deconinck@uclouvain.be

Francois Michel
UCLouvain

Email: francois.michel@uclouvain.be

Florentin Rochet
UCLouvain

Email: florentin.rochet@uclouvain.be

Olivier Bonaventure
UCLouvain

Email: olivier.bonaventure@uclouvain.be