A Path Layer for the Internet: Enabling Network Operations on Encrypted Protocols

Mirja Kühlewind*, Tobias Bühler*, Brian Trammell*, Roman Müntener†, Stephan Neuhaus†, and Gorry Fairhurst‡
*ETH Zurich, {mirjak,buehlert,briant}@ethz.ch;
†Zurich Univ. of Applied Sciences, {munt, neut}@zhaw.ch;
‡Univ. of Aberdeen, gorry@erg.abdn.ac.uk

***Abstract*—The deployment of encrypted transport protocols imposes new challenges for network management. Key in-network functions such as those implemented by firewalls and passive measurement devices currently rely on information exposed by the transport layer. Encryption, in addition to improving privacy, helps to address ossification of network protocols caused by middleboxes that assume certain information to be present in the clear. However, "encrypting it all" risks diminishing the utility of these middleboxes for the network management tasks for which they were designed. A middlebox cannot use what it cannot see.**

We propose an architectural solution to this issue, by introducing a new "path layer" for transport-independent, in-band signaling between Internet endpoints and network elements on the paths between them, and using this layer to reinforce the boundary between the hop-by-hop network layer and the end-to-end transport layer. We define a path layer header on top of UDP to provide a common wire image for new, encrypted transports. This path layer header provides information to a transport-independent on-path state machine that replaces stateful handling currently based on exposed header flags and fields in TCP; it enables explicit measurability of transport layer performance; and offers extensibility by sender-to-path and path-to-receiver communications for diagnostics and management. This provides not only a replacement for signals that are not available with encrypted traffic, but also allows integrity-protected, enhanced signaling under endpoint control. We present an implementation of this wire image integrated with the QUIC protocol, as well as a basic stateful middlebox built on Vector Packet Processing (VPP) provided by FD.io.

I. Introduction

New Internet transport protocols, such as QUIC [1], [2], leverage encapsulation and encryption of transport-layer headers, in order to improve end-user privacy in the face of pervasive surveillance as well as to facilitate deployment in the Internet. Prior experience with new transport protocols such as SCTP [3], or new TCP extensions such as TCP Fast Open [4], shows that these protocols can be difficult to deploy due in part to incorrect modification and dropping of packets with unknown headers by middleboxes.

On the one hand, future transport protocols that leverage encryption assist in the restoration of the end-to-end nature of the Internet by returning complex processing to the endpoints. On the other hand, many of these middlebox functions, such as network address translation (NAT), firewalling, or passive network performance measurement, are essential to network operations and troubleshooting. Since middleboxes cannot modify what they cannot see, a decision to use pervasive transport header encryption, whatever the motive, will have implications for network operations and design.

This tussle between middleboxes which need to inspect transport headers and end-to-end transport protocols which encrypt their headers can lead to an arms race: middlebox vendors attempt to reverse-engineer protocols in order to support the inference-based functions their current devices perform on non-encrypted protocols such as TCP; protocol designers tighten their designs to provide stronger resistance to said reverse-engineering, and so on.

The transport layer is the lowest layer at which end-to-end interactions across the Internet happen. Transport protocols, being layered directly over the network service, are sent in the payload of network-layer packets. However, this simple architectural view hides one of the core functions of the transport - to discover and adapt to the properties of the Internet path that is currently being used. The design of Internet transport protocols is as much about trying to avoid the unwanted side effects of congestion on a flow and other flows sharing capacity, avoiding congestion collapse, adapting to changes in the path characteristics, etc., as it is about end-to-end feature negotiation, flow control and optimising for performance of a specific application.

In this work we propose to separate these two sets of functions, arguing that it is not enough to have an end-to-end transport layer directly above the network layer, and introducce a *path layer* between them. We can identify three classes of protocol header information, each of which can be empty in any given packet: (1) fields for the exclusive use of the remote endpoint, such as transport capability negotiation or information needed for synchronising endpoint state; (2) fields that on-path devices are intended to inspect, but must not modify; and (3) fields that on-path devices are intended or assumed to modify. Fields in the first class belong in the transport layer, and since experience shows that it is not effective for an RFC to ask nicely that on-path devices not modify transport-layer headers, these should be encrypted. Fields in the second and third class, on the other hand, belong in the path layer, and access to them cryptographically controlled.

Indeed, the most recent interesting example of a new transport protocol, QUIC [1], follows this approach. UDP encapsulation provides port numbers, which are effectively in the third class due to the widespread deployment of network address translation (NAT) on access networks [5]; almost everything else is encrypted. This approach does not support the second class of transport headers. TCP sequence numbers and flags, for example, contain no semantic information from the application layer, but are widely used for performance measurement and on-path state maintenance, respectively.

Currently, stateful on-path devices often manage forwarding state by using an internal model of the TCP state machine to determine when a TCP flow starts and ends; instead, UDP flows must rely on timeouts that are generally short relative to those for TCP [6], requiring UDP-encapsulated transports either to generate unproductive keep-alive traffic for long-lived sessions, or to tolerate connectivity problems and reconnect after loss of forwarding state. Despite these efforts to duplicate the TCP state machine, on-path devices are not necessarily interested in whether a TCP packet sets the SYN or FIN or RST flag. Rather, they are interested in whether the packet is the first or last in a flow, because then they need to allocate or deallocate memory to manage that flow. The actual flags in the TCP header are *incidental*, and that the middlebox merely uses these flags to infer *essential* information about the flow to which the packet belongs. If a packet could explicitly expose essential information to the path, then the incidental information contained in the transport header could remain encrypted, and thus protected from tampering.

This paper presents the design principles of a Path Layer UDP Substrate (PLUS) header, designed to support in-band signaling for transport-independent management of in-network state and add explicit support for enhanced in-network services. PLUS is intended to be deployed together with an encrypted transport protocol such as QUIC [1] to protect confidentiality of headers and payloads, and also to protect the integrity of information in PLUS that is exposed to the network. PLUS therefore provides a common "wire image" for new, encrypted transport protocols.

This work is based on an ongoing experimental effort by the authors on the topic of in-band signaling and transport protocol evolution [7], [8], [9], [10], [11], further developed through discussion at two Birds of a Feather (BoF) sessions within the Internet Engineering Task Force (IETF): SPUD in March 2015 and PLUS in July 2016. The present work defines design goals for path layer signaling that enables endpoints to safely send information to the path on which they travel, and vice versa (Section II), on the background of a well-defined, limited set of applications (Sections III and IV), and based on wider discussions about this ongoing effort. It then specifies PLUS, a new protocol providing an integrity-protected and extensible common wire image for all future encrypted transports (Section V), and reports on a first reference implementation of PLUS for both the endpoint, integrated with QUIC, and middleboxes implemented using the FD.io Vector Packet Processor [12], focusing on how this implementation differs from a traditional, fully-layered approach (Section VI).

A. Related Work

As the ability to extend TCP is dependent on network conditions [13], [14], transport protocols are emerging that use UDP as an encapsulation layer. PLUS applies lessons learned from a great deal of prior work on application-to-network and/or network-to-application signaling to this practice. Work on Quality of Service (QoS) has led to standardized signaling approaches such as NSIS [15]. However, due to deployment challenges of a separate signaling protocol these proposals have seen very little inter-domain deployment.

Other approaches for network signaling operate at the session layer, e.g. as proposed by Kissel et al [16] or e.g. Session Initiation Protocol (SIP) [17] for initiation of media sessions. These protocols expose specific application semantics, while PLUS is independent of application or transport protocol. The IETF IPPM working group has produced a network-layer specification [18] defining an IPv6 Destination Option to carry timestamp and packet serial number information, enabling loss and latency measurements. However, its reliance on IPv6 extension headers may limit deployment on the Internet at large [19].

II. Requirements and Design Goals

In this section, we begin with design principles, derived from a set of functional requirements, observations about devices currently deployed in the Internet, and current best practices in protocol design and implementation:

1) **An endpoint should be able to explicitly expose any signals used by on-path devices**. We refer to this as sender-to-path signaling.
2) **An endpoint should be able to request signals from devices on the path**. We refer to this as path-to-receiver signaling.
3) **An on-path device should not be able to forge, change, or remove a signal sent by an endpoint.** This implies a need for end-to-end integrity protection for these signals.
4) **The endpoint should control signaling between endpoints and the path, or from one on-path device to another**. The use of signaling on a given packet or flow is therefore optional. An on-path device should not be able to force an endpoint to send a signal, or to use the mechanism to send a signal of its own volition without explicit cooperation from the endpoints. This can be achieved by integrity protection over a scratch space allowing devices on path to send specified signals to receivers.
5) **It should be possible for an endpoint to request and receive signals from a previously unknown on-path device**. This implies that, in the absence of a cryptographic introduction protocol or public-key infrastructure for on-path devices in the Internet – which we consider to be impractical – authentication of signals from these on-path devices is not possible.
6) **The mechanism should present no significant surface for amplification attacks.** We can achieve this by specifying that no signal can request transmission of a packet beyond forwarding the packet carrying the signal. This relies on in-band signaling, piggybacked on higher-layer traffic.

Our design assumes that transport and application layer information is encrypted, and is independent from encryption at layer 2 or layer 3. The former is hop-to-hop, and is therefore independent of on-path usage of the path layer header. The latter is generally applied via tunneling. As with all tunnels, nodes along the path forwarding the tunneled traffic cannot inspect or modify traffic inside the tunnel. We therefore understand an "on-path" device to be one which, in the current Internet architecture, could inspect or modify the transport-layer headers.

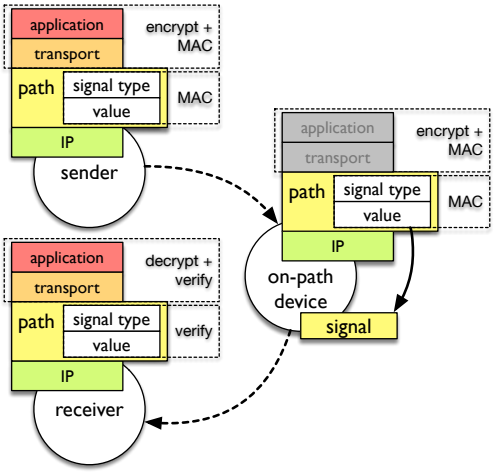

Fig. 1. Sender-to-path signaling: signals are readable by on path-devices, but modification is detected by the receiver

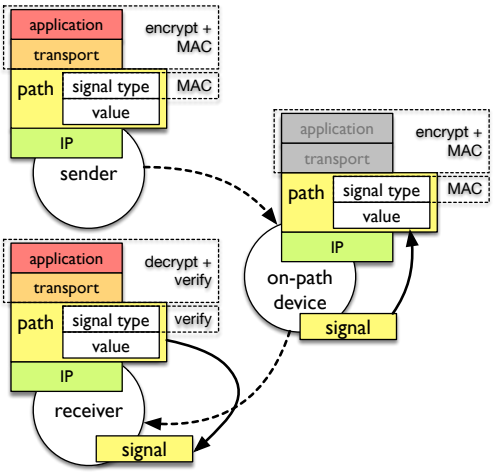

Fig. 2. Path-to-receiver signaling: signal content is writable by on-path devices, but signal presence is protected end-to-end. Signals can neither be added nor removed on path.

Sender-to-path signaling, as shown in Figure 1, is unencrypted but with integrity protection, relying on the receiver to verify integrity. On-path devices can read these signals, but not verify their authenticity. Modifications to these signals will be detected by the receiver, which can respond by raising an error to the transport and/or application layer.

Path-to-receiver signaling, as shown in Figure 2, is also initiated by the sender. The sender places the requested signaling type in the packet and creates a "scratch space" writable by on-path devices in the packet. The length of the scratch space is fixed by the sender, to avoid problems with downstream loss and/or fragmentation due to packet size changes along the path. Path accumulation signals, such as those proposed in IPIM [20] and further specified in Section V-D, can also use this mechanism; in this case, the scratch space is initialized by the sender to some value, and each device aware of the signal along the path updates it according to an algorithm specified by the type, until the final accumulated value is received at the receiver. Since some path-to-receiver signals may be of more use to the sender than the receiver, path-to receiver signaling can be augmented by **feedback**, the return of the final value of a signal from the path by the receiver back to the sender.

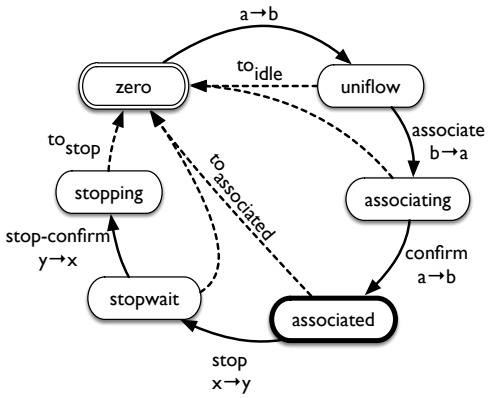

Fig. 3. A transport-independent path layer state machine.

III. A Transport-Independent On-Path State Machine

Our most basic signaling requirement is to support on-path recognition of session establishment and disestablishment, as provided by passive observation of the TCP three-way handshake and the FIN and/or RST flags on connection shutdown. These are visible on the path between endpoints, and therefore are often used by those network devices for state management by devices such as NATs and firewalls.

The most important signals derived from these observations concern flow lifetime and association of the two directions of a bi-directional flow, associating a uniflow with its reverse counterpart. Firewalls and other stateful network elements can use this association to assume endpoint consent to communicate. A stop signal that indicates the last packet of a flow can also assist state management, and allows for a longer timeout for active flows than presently possible for UDP [6], reducing the rate at which keep-alive traffic must be sent to avoid loss of on-path state. Stateful network devices can apply stricter policies for a flow which does not provide these signals, such as shorter timeouts to remove flow state, or rules to drop unknown traffic.

Figure 3 shows a transport-independent state machine that details the basic operation of a stateful on-path device.

The state machine has the following properties:

1) A device on path that can see traffic in both directions knows each side of an association wishes that association to continue. This allows firewalls to delegate policy decisions about accepting or continuing an association to the servers they protect.
2) A device on path that can see traffic in both directions knows that each device can receive traffic at the source address it provides. This allows firewalls to protect against trivially spoofed packets, implemented by associating and associated states.
3) Both endpoints confirm the desire to end communication, providing resistance against early state expiry

attacks (e.g. TCP RST injection) by on-path devices, implemented by the stopwait and stopping states.

The first two properties hold with current firewalls and network address translation devices observing the flags and sequence/acknowledgment numbers exposed by TCP. Detailed signaling for this state machine is defined in sections V-B and V-B. The abstract state machine is described in further detail in [9].

IV. Network Performance Measurement

Passive performance measurement is a second important application of information presently available in the unencrypted TCP header. Basic metrics suffice for most performance measurement tasks: transmission rate, latency/jitter, and packet loss rate. Transmission rate can be trivially measured by counting bytes associated with a traffic flow, whether the headers are encrypted or not, then dividing this count by the observation time interval. We therefore focus on latency and loss.

Latency measurement, as round-trip time, requires the ability to match packets in one direction with packets in the opposite direction, for both directions of the flow. Current inference-based passive measurement approaches [21], [22] generally use the TCP Timestamp Option [23], when available [13], [24]. Loss measurement, on the other hand, requires inference about the loss detection and retransmission algorithms in use by the sender [25].

PLUS is designed to support explicit passive measurability of latency and loss to replace inference-based approaches. We follow the principles proposed by Allman et al [20] that measurement should be explicit, in-band, visible, cooperative, and under the absolute control of the endpoint. Indeed, we find that designing signals expressly for measurement, much of the messy inference involved in turning a TCP packet stream into metric samples becomes unnecessary.

To measure latency, we propose a simplification of the arrival information primitive in [20]: a simple packet serial number (PSN) that increases by one with every packet sent, including retransmissions and control packets (unlike TCP), and a packet serial echo (PSE) that reflects the last packet number seen in the opposite direction. This is simple to implement, and provides adequate information for the calculation of latency between each endpoint and a passive observation point on both the upstream and downstream path between them (e.g., at a network border or CPE gateway, where these measurements are usually deployed).

Given two endpoints a and b, an observation point c can measure the time interval between seeing a given PSN on a packet sent by a and an equal or greater PSE on a packet sent by b to get a latency estimate on the path $c \leftrightarrow b$. By reversing the measurement, it can likewise estimate $c \leftrightarrow a$, and $a \leftrightarrow b = c \leftrightarrow b + c \leftrightarrow a$.

Loss estimation on the path $a \rightarrow c$ for packets in direction $a \rightarrow b$ is supported by our PSN/PSE latency measurement facility by counting gaps in the sequence of PSNs sent by a. However, this gives no visibility into loss on the path $c \rightarrow b$.

Since PLUS has a goal of transport protocol independence, it cannot simply expose transport protocol internals: requiring observation points and measurement analysis to infer behavior of endpoints based upon an identification of the transport protocol is precisely the current situation we want to avoid. Therefore, we must rely on the receiver to expose information about the loss and congestion experienced. We take design inspiration from the Congestion Exposure (ConEx) protocol [26], and define a sender-to-path signal for periodically exposing counts of detected packet loss and congestion signals to provide a delayed but accurate loss metric to the path.

Each sender emits a periodic sender-to-path signal containing a two-tuple $\{l, m\}$ where l is a cumulative count of the number of detected losses l since the beginning of the flow and m is a cumulative count of the number of detected ECN [27] marks since the beginning of the flow. A measurement point can receive and analyze the series of signals to derive various loss statistics: $l_{t2} - l_{t1}$, for example, gives the number of losses detected by the sender in the time interval $(t1 - owd(a \rightarrow c), t2 - owd(a \rightarrow c))$, where a first-order estimate of the one-way delay from the sender $owd(a \rightarrow c)$ can be derived from RTT measurements[1]. Comparing this to the observed packet transmission rate during the same interval, as well as to the upstream loss information available from the PSN series, allows a measurement point to estimate upstream as well as downstream loss rates. The addition of ECN marking information, similarly, together with an analysis of observed CE codepoints at the IP layer, allows the estimation of upstream and downstream congestion for ECN-enabled flows.

V. Path Layer UDP Substrate (PLUS)

Based on our design goals, requirements, and desired signaling focused on network management and measurement, we present the Path Layer UDP Substrate (PLUS)[2]. PLUS is implemented as a header between the UDP header and an encrypted transport header. The PLUS header explicitly exposes information intended by the sender for use by devices along the path independent of the transport protocol, hiding everything else with transport encryption.

PLUS defines two headers, a Basic Header carrying information for state management, basic measurability, and simple packet treatment; and an Extended Header carrying a path communication field, described in Section V-C. The Basic Header supports sender-to-path signaling for state maintenance and basic measurement. The Extended Header supports both sender-to-path and path-to-receiver signaling, for carrying information for which there is no room in the Basic Header, as well as for accumulated path information as in section 4.3 of [20].

A. Basic Header

The format of the PLUS header, together with the UDP header, is shown in Figure 4. The magic field is a 28-bit number that identifies this packet as carrying a PLUS header. This magic number can be used by PLUS-aware devices on

[1] Path asymmetry, of course, reduces the accuracy of this estimate. We leave accurate OWD estimation through timestamp exposure and synchronization as a topic for future work.

[2] An Internet-Draft specifying the PLUS header [11] will be kept in sync with the content of this section; as of this writing, this paper represents the latest specification.

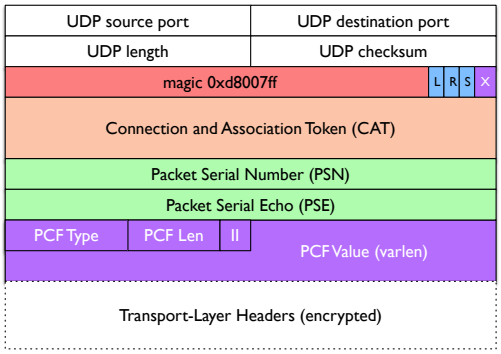

Fig. 4. The PLUS header format. The Basic Header (X=0) omits the PCF fields; the Extended Header (X=1) has all fields.

path to distinguish PLUS packets from non-PLUS packets, since PLUS cannot be identified by UDP port number. It is chosen to avoid collision with possible values of the first four bytes of any packet in widely deployed protocols on UDP. This both minimizes the chance of accidental classification of a non-PLUS protocol as PLUS, and makes it difficult to cause spurious PLUS packets to be generated by reflection or amplification.

The next four bits provide flags: two per-packet Quality of Service (QoS) signals, a stop signal, and an extended header bit. The (L)oLa flag, when set, indicates that the packet is latency sensitive and prefers drop to delay. The (R)oI flag, when set, indicates that the packet is not sensitive to reordering and thus does not need to be given the same treatment or routing as other packets of the same flow. These two signals address many QoS challenges operators face in current networks but cannot be reliably provided end-to-end by existing mechanisms, such as the use of the Differentiated Services Codepoint (DSCP) field in the IP header [28]. The (S)top flag indicates a stop or stop confirmation signal when set; see Section V-B. When the E(X)tended Header bit is set, the Extended Header is present; see Section V-C.

The Connection/Association Token (CAT) is a 64-bit token identifying this association. The CAT is chosen randomly by the connection initiator, and allows both multiplexing of flows over a 5-tuple as well as fast rebinding: a PLUS packet sharing one endpoint (source address/port pair, or destination address/port pair) and the CAT with an existing flow is taken to belong to that flow, since the other endpoint identifier might change due to a mobility event or address translation change.

The Packet Serial Number (PSN) is a 32-bit serial number for this packet. The initial PSN for each direction in a flow is chosen randomly. Each subsequently sent packet in a flow increments the PSN by one, wrapping around. Respectively, the Packet Serial Echo (PSE) is the most recent PSN seen by the sender in the opposite direction before this packet was sent. If no packet has yet been seen by the sender (i.e., the packet is the first packet in a bidirectional flow), the sender sets the PSE to zero. The CAT and PSE together serve as an indication that a packet was actually seen by the remote endpoint, and used for confirmation and stop-confirmation signals. PSN and PSE together are used for latency measurement as in Section IV.

Since PLUS is designed to be used for UDP-encapsulated, encrypted transport protocols, overlying transports are presumed to provide encryption and integrity protection for their own headers, and need to provide integrity protection for the PLUS Basic Header. This implies an interface between PLUS and the transport layer to support this protection, discussed in more detail in Section VI-A1.

B. State Establishment and Maintenance

The PLUS Basic Header provides the signaling required for the abstract state machine described in Section III. A PLUS-aware on-path device forwarding a packet with a PLUS Basic Header with a 5-tuple and CAT that it does not have state for moves that flow to the uniflow state. It will move the flow back to zero state after not seeing a packet on the same flow in the same direction with the same CAT within an idle timeout interval, but may continue forwarding packets in that direction (the a $\rightarrow$ b direction in Figure 3).

A PLUS-aware on-path device forwarding a packet with a PLUS Basic Header with a matching 5-tuple and CAT as a flow in the uniflow state, but in the opposite direction (the b $\rightarrow$ a direction in Figure 3), moves that flow to the associating state. It stores the PSN of the packet that caused this transition, and waits for a packet in the a $\rightarrow$ b direction containing a PSE indicating that that packet has been received. When it sees that packet, it transitions the flow to the associated state. Otherwise, it drops state after an idle timeout interval.

Once a flow has moved to the associated state, it will remain in that state for an associated timeout interval. It resets the timer for every packet with a PLUS Basic Header it forwards in either direction for this flow, identified by the 5-tuple and CAT.

A PLUS-aware on-path device forwarding a packet for a flow in the associated state with an S flag set moves that flow to the stopwait state. It stores the PSN on the packet causing the transition, and continues forwarding packets as if in associated state, dropping state after expiration of the associated timeout interval. When it sees a packet in the opposite direction with the S flag set and the PSE set to exactly the stored PSN, it transitions the flow to closing state. The device will forward packets in both directions for flows in the stopping state within a stop timeout interval; these packets will not reset the timer.

C. Extended Header

When the Extended Header bit is 1, the PLUS Extended Header is present. This header adds a Path Communication Field (PCF) to the Basic Header, as shown in Figure 4. The Extended Header has an 8-bit PCF type field, a 6-bit PCF length field, a 2-bit Integrity Indicator, and a variable-length PCF value field, protected by the overlying transport encryption.

The PCF Type field defines the structure and semantics of the PCF value. The PCF Length field defines the length of the PCF value field in bytes, from 0 to 63 bytes. The PCF Integrity Indicator (II) is used to implement path-to-receiver as well as sender-to-path signaling, implemented by treating a portion of the field as if all its bits are zero for purposes of integrity protection and integrity verification. This protects the type and length of path-to-receiver signals, but allows part or all of the PCF Value to be writable by devices on path.

If the Integrity Indicator is 00, the PCF value is not integrity protected; if it is 11, the PCF value is integrity protected in its entirety. The other two values provide for partial integrity protection: 01 indicates that the the first quarter of the PCF value is protected; 10 indicates that the first half of the PCF value is protected. The integrity protected range is always rounded up to the nearest byte.

D. Extended Header Types

This section defines a set of initial Extended Header types to illustrate the flexibility provided by the Extended Header. Other PCF types are reserved for future use.

The PCF is used to expose loss and congestion markings to the path. PCF Type 1 indicates the Loss and Congestion Exposure field, a field which is 2, 4, 8, or 16 bytes long, and contains two 1, 2, 4, or 8 byte unsigned integers in network byte order. The first is a total count of packets detected as lost by the sender since the start of the connection. The second is a count of the total number of congestion markings (ECN CE codepoints) received by the sender since the start of the connection. Its integrity indicator is always 11, since the information cannot be modified by on-path devices. PCF Type 1 packets can be emitted as often as once per RTT as estimated by the sender; lower rates result in less overhead, but lower fidelity measurements by devices on path.

The PCF is also used for path accumulation; to be useful this requires deployment of future PLUS-aware middleboxes. PCF Type 2 carries a two byte value reflecting the Path Maximum Transmission Unit (PMTU), initialized to the MTU of the link on which the packet is initially sent. When a PLUS-aware middlebox forwards a packet with this PCF, it updates the value with the minimum of the value present and the MTU of the link over which it will forward the packet. The integrity indicator for PCF Type 2 is always 00, since the path can modify the entire value.

PCF Type 3 carries an 8 byte field for path tracing, similar to the Path Changes mechanism in section 4.3 of [20]. The sender chooses a random 8-byte value for each flow, and initializes the PCF value field with this value for each packet with PCF Type 3 in the flow. Each on-path PLUS-aware device forwards the packet fills the result of XORing the received value with an 8-byte device identifier. The same randomly chosen value must be used for all path trace accumulator signals. Packets traversing the same set of PLUS-aware forwarding devices are therefore received with the same accumulated value, and changes to the set of devices on path can be detected by the receiver. The integrity indicator for PCF Type 3 is always 00, since the path can modify the entire value.

Devices on path must recognize that they are requested to participate in signaling by understanding the Extended Header and the semantics of each type. However, there is no guarantee that they will participate or reply honestly. Therefore all signals are advisory only. In general, endpoints still need to implement mechanisms to handle incorrect or incomplete exposed information, as today when default values are assumed (e.g., the accumulated MTU should not be directly used to set the segment size, but rather as input to a packetization layer path MTU discovery process [29]).

E. Encrypted Feedback

Path-to-receiver signals transmit information requested from the path by the sender to the receiver; however, the sender often also needs the information that was requested. The transport protocol running atop PLUS must therefore provide a feedback channel for the full PLUS Extended Header on any packet received containing a PCF value where the integrity indicator is not 11. Returning the entire header allows the sender to associate the value fed back with the original packet sent; feedback of the entire header depending only on the II value allows receivers to feed back PCF values they do not understand. We assume the overlying transport provides end-to-end encryption, and that signals fed back do not need to be exposed to the reverse path. The feedback channel interface must be designed on a per-transport basis. Section VI-A illustrates how this works in QUIC.

F. Transport layer API

The information presently supported by PLUS does not require any application interaction: CAT, PSN/PSE, and the loss/congestion PCF all expose either information available within PLUS, or that is readily available from the internals of the overlying transport. Similarly, MTU accumulation is only useful at the transport layer, and path change detection is, in its initial form, more suited to diagnostic tools in the spirit of traceroute than online use by an application. The transport-application interface therefore needs no change when a given transport is running over PLUS. However, there can be further benefits if an application is aware of PLUS. The decision to enable or disable PLUS on a given flow or node may be driven not only by the needs of the transport protocol but also the application, or even the user, following the principles of transparency and endpoint control.

VI. Insights from Implementation

For evaluating the fitness of PLUS, we implement both a PLUS-enabled endpoint using QUIC, a new encrypted transport protocol, and a PLUS-aware middlebox that uses PLUS information for state management and passive measurement.

As an encapsulation layer requiring some integration between transport and path layer, PLUS raises questions of overhead and performance. Of course PLUS adds additional bits to each packet, however, Extended Header fields need not be present on every packet in a flow. So each PCF type can be tuned for a fidelity/overhead tradeoff by the sender. Further, a close integration between the transport protocol and PLUS, especially if the transport protocols supports optional extensibility mechanisms, makes it possible that PLUS Basic Header fields, since they are integrity protected, can replace equivalent fields in the overlying transport, reducing per-packet overhead in bytes.

Questions on the overall performance of PLUS are closely coupled to the use case. Some PCFs must only be sent once, other need to be repeated frequently. Depending on the information provided, PLUS adds a certain bit overhead but still can provide input to a network function that can improve the overall performance, e.g. providing low latency for latency-sensitive applications. However, based on our initial implementation, we can provide some qualitative pointers

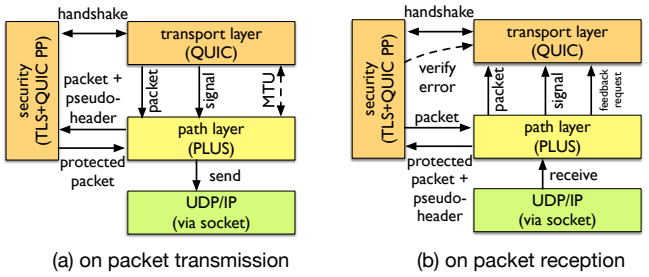

Fig. 5. Interfaces between QUIC and PLUS

toward answers. Since PLUS is designed to work over UDP, advances in technologies for user-space stacks (including fd.io, on which our middlebox implementation is based) make it possible to build highly performant PLUS-capable transport protocol implementations that do not cross a kernel context switch boundary. In our selected use case to enable basic measurements based on information in PLUS Basic Header, the performance penalty is negligible.

A. Endpoint Implementation: QUIC

QUIC is a UDP-encapsulated, encrypted transport protocol initially developed by Google and currently being standardized within the IETF[3]. Although this provides a natural choice for the overlying transport, the protocol is as-yet a moving target. Our work is based a openly available `quic-go` implementation[4] of a recent Google-internal version of QUIC, from which the IETF version of QUIC has diverged. The most crucial change was the replacement of a QUIC-specific cryptographic protocol with TLS. Despite tracking a work in progress, the design of the interface between QUIC and PLUS has been fairly stable that we describe in more detail below.

1) A Two-way Interface: PLUS, as QUIC, requires a different interaction with its cryptographic protocol than the traditional layering concept used for TLS over TCP. Further, PLUS can benefit from a tight coupling between the two protocols, especially in the case of QUIC. The service PLUS provides to its overlying transport is transmission of packets over a UDP socket with a PLUS Basic or Extended Header. PLUS requires the overlying transport to provide: integrity protection of the PLUS pseudoheader (the PLUS header with any bytes marked as unprotected by the integrity indicator replaced by zeroes), and feedback of path-to-receiver information to the sender along with delivery of this information to the application, as necessary.

2) Integrity Protection of the PLUS header: Each QUIC packet header is integrity protected, but not encrypted, using Authenticated Encryption with Additional Data (AEAD) [30]. The interface between PLUS and QUIC achieves integrity protection by extending this additional data to also cover the adjacent PLUS packet pseudoheader.

Figure 5(a) shows the flow for sending a packet: QUIC directly interacts with the TLS component [31] for the TLS handshake enabling the initial exchange of cryptographic information, without using encryption, or other control action. However, all payload packets that are to be encrypted are handed directly from QUIC to PLUS, along with an indication of the QUIC header length to be integrity protected. PLUS adds its header, and passes the packet to TLS+QUIC packet processing for encryption of the QUIC payload including its internal control information and integrity protection of the public QUIC and PLUS headers. The resulting packet is returned to PLUS for transmission over a UDP socket.

An additional interface allows QUIC and PLUS to exchange further signalling information. PLUS must inform QUIC about the MTU minus the length of the PLUS header, since the length of the Extended Header can vary on a per-packet basis. Other information can be exchanged e.g. to coordinate the QUIC connection and packet number with the PLUS CAT and PSN, or to provide input for information signalled using a certain PCF. Our implementation allows this information to either be stable for the lifetime of a flow, such as the L flag (e.g. utilising information from the system or socket configuration), or could dynamically change based on the state of the transport protocol or application on top, such as a request to send or increase the frequency of Extended Headers to perform measurements.

On packet reception, this process runs in reverse, as shown in Figure 5(b); here, the packet protection component also signals QUIC if the integrity-protected part of the QUIC or integrity-protected part of the PLUS header was tampered with.

3) The PLUS Feedback Frame: QUIC is a multistreaming transport that splits its payload and control data into *frames* associated with *streams* [1]. Stream 0 is reserved for control traffic and use of the embedded TLS 1.3. PLUS feedback information can be carried in a new feedback frame added to QUIC on the control stream. The interface between QUIC and PLUS (Figure 5), includes a feedback request for PLUS to pass to QUIC, for inclusion in a feedback frame returned to the sender. As noted in Section V-E, the feedback frame contains the entire PLUS Extended Header on the received packet with a path-to-receiver signal.

B. Middlebox implementation: FD.io – VPP

The Fast Data – Input/Output (FD.io) project [12] collects multiple projects and libraries to provide fast and programmable IO services for networking and storage. FD.io runs on a variety of architectures and development environments and automatically uses features like DPDK [32]. A core component of FD.io is the Vector Packet Processing (VPP) library [33]. A PLUS-aware middlebox can be implemented in one (or multiple) VPP nodes.

As described in Section V-A, a PLUS header cannot be identified based on a specific UDP port. Instead, we have to look for the corresponding magic number. For each UDP packet, the "IP4 UDP lookup" node has to fetch the four bytes (respectively 28 bits) directly following the UDP header and compare these to the PLUS magic number. This indicates the placement of the PLUS node in the VPP tree as shown in Figure 6. If a PLUS header is present, the packet is forwarded to the PLUS node and analyzed there. Afterwards, we move the packet pointer to the start of the next header; the following node may therefore not even notice the PLUS header. However,

[3]See https://github.com/quicwg/base-drafts
[4]https://github.com/lucas-clemente/quic-go

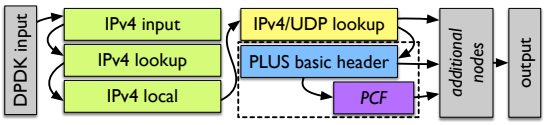

Fig. 6. Placement of a PLUS nodes in the VPP tree, assuming an IPv4 header.

if the following node wants to access fields in the underlying headers, it has to be aware of the additional offset due to the PLUS header. We include this information in the metadata exchanged between nodes.

A PLUS-aware middlebox aims to also support flows between endpoints whose IP addresses change during the flow, e.g. due to NAT re-binding. Therefore, the PLUS node cannot blindly compare the observed five-tuple and CAT with the existing flow states. We apply a two-step approach. First, the PLUS node tries to find an existing state for the CAT value. In a second step, the five-tuples are compared. If the observed packet shares at least one endpoint (source address/port respectively destination address/port) with the existing flow state, the packet is assigned to this flow and the state may be updated. Otherwise, the five-tuple and CAT build state for a new flow.

Since each PLUS packet can carry only one PCF at a time, each distinct PCF value is processed in a separate VPP node: the first PLUS node handles flow state and, if the Extended Header is present, it forwards the packet to the correct PCF node based on the PCF Type field. This approach allows new Extended Header types to be added easily. PCF nodes may read or modify certain parts of the PCF value field depending on the PCF type. For PCF type 1, we only read the provided information and compare the number of congestion marked packets to the total number of packets seen to estimated the current congestion level, providing passive measurement information for troubleshooting. For type 2, we compare current value to the MTU provided on the next hop and overwrite it if our new value is lower. For type 3, we also read the value, XOR it with a fingerprint randomly chosen at startup time, and write the result.

VII. Deploying PLUS

As shown in the previous section, our implementation experience to date confirms the basic viability of our approach; however, a few challenges remain to deployability in the Internet. Assuming that overhead is negligible, why would endpoint implementers and network operators deploy PLUS?

The transport-independent state machine in section III provides an incentive for initial adoption. Replacing UDP traffic whose state must be managed by idle timeouts with traffic exposing per-flow start and end signals, distinguishable by a magic number that is probabalistically unlikely to be generated by reflected traffic, removes an incentive to stop blocking UDP on the 3-5% of Internet-connected networks that currently do so [34], [35]. PLUS presents an opportunity to unify in-network state management for all traffic. From the endpoint point of view, exposing state information similar to TCP can reduce keep-alive traffic for long-lived, sparse flows, saving endpoint resources such as battery consumption, e.g., when the radio needs to reconnect on a mobile device.

As a generic facility, PLUS offers an powerful primitive for signaling between middleboxes and endpoints, making it possible to deploy new service models. Operators can enhance network performance while empowering endpoints to control the usage of these services. The more valuable an offered service is to the endpoint, the higher are the chances for adoption. However, not all such models are desirable. For example, an operator could require an endpoint to present a token within a PCF as proof of payment or viewing of an ad to get certain “fast lane” network treatment.

More generally, the signaling mechanism provided by PLUS could be abused as a side channel; for example, a middlebox on an access network could place Personally Identifiable Information (PII), a so called “supercookie”, about a subscriber on packet for tracking purposes. These potential abuses are addressed by the third and fourth design principles in section II: since only the sender can add PCFs and determine integrity protection on them, side-channel abuse is not feasible in the general case, providing stronger security support than most protocols deployed today, including TCP.

Coercion by a user’s sole access provider is unfortunately impossible to combat. However, the fact that the existence of a PCF is protected end-to-end and its value is visible to all nodes along the path as well as the receiver means that such coercion would at least be widely transparent; a coercive access provider would presumably seek a more secretive way to exercise such coercion, such as at layer 2 on its own network or out-of-band. PLUS therefore presents no additional surface for this attack.

Further, restrictions on PCF vocabulary, most importantly the restriction that all PCF information be treated by both endpoints and on-path devices as advisory, is designed to reduce the set of attacks available to uncooperative endpoints and middleboxes. As such PLUS provides a higher protection than deployed protocols today and enables encryption of all end-to-end information in the transport layer while providing a clear split of information needed to maintain current network management practices, as an deployment incentive to both endpoints and middleboxes.

VIII. Next Steps

Our work to date provides confidence that the design and flexibility of PLUS will be suitable to address a large variety of path layer use cases, and we are currently focused on demonstrating the feasibility of the functional mechanisms required by our design goals, and the scalability of our approach for fast packet processing for in-network functions. Our next steps focus on detailed evaluation of these use cases, showing that the benefits provided by each make a compelling case to deploy PLUS. For example, we are currently investigating the use of the L flag for simplified mobile QoS, as input for bearer selection and active queue management (AQM) and enhanced scheduling in mobile networks. As PLUS provides a generic path layer, the benefits it provides increase with every new PCF type added and each middlebox that understands them. We intend PLUS as a foundational mechanism for research and experimentation with uses for an Internet-deployable path layer, and new methods for network management leveraging cooperation between endpoints and on-path devices.

IX. Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688421, and was supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 15.0268. The opinions expressed and arguments employed reflect only the authors' views. The European Commission is not responsible for any use that may be made of that information. Further, the opinions expressed and arguments employed herein do not necessarily reflect the official views of the Swiss Government.

We thank the proponents and participants in the SPUD and PLUS BoFs, whose feedback and ideas have been instrumental in developing this work; especially Joe Hildebrand, Ted Hardie, Natasha Rooney, and Aaron Falk.

References

[1] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," IETF, Internet-Draft draft-ietf-quic-transport-02, 2017.

[2] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," in *Proceedings of ACM SIGCOMM 2017 Conference (to appear)*, Los Angeles, California, USA, August 2017.

[3] R. Stewart, "Stream Control Transmission Protocol," IETF, RFC 4960, September 2007.

[4] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain, "TCP Fast Open," Internet Requests for Comments, RFC Editor, RFC 7413, December 2014, http://www.rfc-editor.org/rfc/rfc7413.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7413.txt

[5] G. Maier, F. Schneider, and A. Feldmann, "Nat usage in residential broadband networks," in *Proceedings of the 12th International Conference on Passive and Active Measurement*, ser. PAM'11. Atlanta, GA: Springer-Verlag, 2011, pp. 32–41. [Online]. Available: http://dl.acm.org/citation.cfm?id=1987510.1987514

[6] S. Hatonen, A. Nyrhinen, L. Eggert, S. Strowes, P. Sarolahti, and M. Kojo, "An experimental study of home gateway characteristics," in *Proc. ACM IMC*, 2010.

[7] B. Trammell and J. Hildebrand, "Evolving transport in the Internet," *Internet Computing, IEEE*, vol. 18, no. 5, pp. 60–64, Sept 2014.

[8] B. Trammell, M. Kuehlewind, E. Gubser, and J. Hildebrand, "A new transport encapsulation for middlebox cooperation," in *2015 IEEE Conference on Standards for Communications and Networking (CSCN)*, Tokyo, Japan, Oct 2015, pp. 187–192.

[9] M. Kuehlewind, B. Trammell, and J. Hildebrand, "Transport-Independent Path Layer State Management," Working Draft, IETF Secretariat, Internet-Draft draft-trammell-plus-statefulness-03, March 2017, http://www.ietf.org/internet-drafts/draft-trammell-plus-statefulness-03.txt. [Online]. Available: http://www.ietf.org/internet-drafts/draft-trammell-plus-statefulness-03.txt

[10] B. Trammell, "Abstract Mechanisms for a Cooperative Path Layer under Endpoint Control," IETF, Internet-Draft draft-trammell-plus-abstract-mech-00, Sep 2016.

[11] B. Trammell and M. Kuehlewind, "Path Layer UDP Substrate Specification," Working Draft, IETF Secretariat, Internet-Draft draft-trammell-plus-spec-01, March 2017, http://www.ietf.org/internet-drafts/draft-trammell-plus-spec-01.txt. [Online]. Available: http://www.ietf.org/internet-drafts/draft-trammell-plus-spec-01.txt

[12] A Linux Foundation Project, "The Fast Data Project (FD.io)," 2017, https://fd.io/.

[13] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP?" in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11. New York, NY, USA: ACM, 2011, pp. 181–194. [Online]. Available: http://doi.acm.org/10.1145/2068816.2068834

[14] R. Craven, R. Beverly, and M. Allman, "Middlebox-cooperative TCP for a non end-to-end Internet," in *Proceedings of ACM SIGCOMM 2014 Conference*, Chicago, IL, USA, August 2014.

[15] R. Hancock, G. Karagiannis, J. Loughney, and S. V. den Bosch, "Next Steps in Signaling (NSIS): Framework," IETF, RFC 4080, June 2005.

[16] E. Kissel and M. Swany, "The eXtensible Session Protocol: A Protocol for Future Internet Architectures," Indiana University, Tech Report TR700, Feb 2012. [Online]. Available: http://www.cs.indiana.edu/cgi-bin/techreports/TRNNN.cgi?trnum=TR700

[17] J. Rosenberg *et al.*, "SIP: Session Initiation Protocol," IETF, RFC 3261, June 2001.

[18] N. Elkins, R. Hamilton, and M. Ackermann, "IPv6 Performance and Diagnostic Metrics (PDM) Destination Option," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-ippm-6man-pdm-option-10, May 2017, http://www.ietf.org/internet-drafts/draft-ietf-ippm-6man-pdm-option-10.txt. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-ippm-6man-pdm-option-10.txt

[19] F. Gont, J. Linkova, T. Chown, and W. Liu, "Observations on the Dropping of Packets with IPv6 Extension Headers in the Real World," Internet Requests for Comments, RFC Editor, RFC 7872, June 2016.

[20] M. Allman, R. Beverly, and B. Trammell, "Principles for Measurability in Protocol Design," *ACM SIGCOMM Computer Commuication Review*, vol. 47, no. 2, April 2017.

[21] S. D. Strowes, "Passively measuring TCP round-trip times," *Commun. ACM*, vol. 56, no. 10, pp. 57–64, Oct. 2013. [Online]. Available: http://doi.acm.org/10.1145/2507771.2507781

[22] B. Trammell, D. Gugelmann, and N. Brownlee, "Inline Data Integrity Signals for Passive Measurement," in *Proc. Sixth Int. Wksp. on Traffic Measurement and Analysis*, London, England, April 2014.

[23] D. Borman, B. Braden, V. Jacobson, and R. Scheffenegger, "TCP Extensions for High Performance," Internet Requests for Comments, RFC Editor, RFC 7323, September 2014.

[24] M. Kühlewind, S. Neuner, and B. Trammell, "On the State of ECN and TCP Options on the Internet," in *Proceedings of the 14th International Conference on Passive and Active Measurement*, ser. PAM'13. Hong Kong, China: Springer-Verlag, 2013, pp. 135–144. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-36516-4_14

[25] M. Allman, W. M. Eddy, and S. Ostermann, "Estimating Loss Rates with TCP," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 3, pp. 12–24, Dec. 2003. [Online]. Available: http://doi.acm.org/10.1145/974036.974038

[26] B. Briscoe, R. Woundy, and A. Cooper, "Congestion Exposure (ConEx) Concepts and Use Cases," IETF, RFC 6789, 2012.

[27] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," Internet Requests for Comments, RFC Editor, RFC 3168, September 2001, http://www.rfc-editor.org/rfc/rfc3168.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3168.txt

[28] J. You, M. Welzl, B. Trammell, M. Kuehlewind, and K. Smith, "Latency Loss Tradeoff PHB Group," IETF, Internet-Draft draft-you-tsvwg-latency-loss-tradeoff-00, March 2016.

[29] M. Mathis and J. Heffner, "Packetization Layer Path MTU Discovery," Internet Requests for Comments, RFC Editor, RFC 4821, March 2007, http://www.rfc-editor.org/rfc/rfc4821.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4821.txt

[30] D. McGrew, "An Interface and Algorithms for Authenticated Encryption," Internet Requests for Comments, RFC Editor, RFC 5116, January 2008, http://www.rfc-editor.org/rfc/rfc5116.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5116.txt

[31] M. Thomson and S. Turner, "Using Transport Layer Security (TLS) to Secure QUIC," IETF, Internet-Draft draft-ietf-quic-tls-02, 2017.

[32] A Linux Foundation Project, "Data Plane Development Kit (DPDK)," 2017, http://dpdk.org/.

[33] "Vector Packet Processing (VPP)," 2017, https://wiki.fd.io/view/VPP/What_is_VPP%3F.

[34] K. Edeline, M. Kuehlewind, B. Trammell, E. Aben, and B. Donnet, "Using UDP for Internet Transport Evolution," arXiv, cs.NI arXiv:1612.07816, 2016, ETH TIK Technical Report 366. [Online]. Available: http://arxiv.org/abs/1612.07816

[35] I. Swett, "QUIC Deployment Experiment @ Google," Proceedings of IETF 96, July 2016. [Online]. Available: https://www.ietf.org/proceedings/96/slides/slides-96-quic-3.pdf