This project has received funding from the European Unions Horizon 2020
research and innovation programme under grant agreement No 688421.

Measurement and Architecture for a Middleboxed Internet

H2020-ICT-688421

Transport and Network Signaling Mechanisms Implementation Report

Author(s):	ETH	Brian Trammell, M. Kühlewind
	UNIABDN	Gorry Fairhurst (ed.), Ana Custura, Tom Jones
	ZHAW	Stephan Neuhaus
	ALCATEL	Thomas Fossati
	U3CM	Pedro Aranda

Document Number:	D3.3
Internal Reviewer:	Benoit Donnet
Due Date of Delivery:	31 December 2018
Actual Date of Delivery:	17 December 2018
Dissemination Level:	Public

Disclaimer

The information, documentation and figures available in this deliverable are written by the MAMI consortium partners under EC co-financing (project H2020-ICT-688421) and does not necessarily reflect the view of the European Commission.

The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The user uses the information at its sole risk and liability.

European Commission | Horizon 2020 European Union funding for Research & Innovation

Contents

European Commission | Horizon 2020 European Union funding for Research & Innovation

Executive Summary

WP3 of the MAMI project has the main objective to define a middlebox cooperation architecture. This document therefore describes research leading to a flexible transport stack for this architecture and respective implementation efforts enabling experimental evaluation of the proposed design and mechanisms.

D3.2 specified the Path Layer UDP Substrate (PLUS), a UDP-based shim-layer protocol for sender-to-path and path-to sender signalling, as a core mechanism for middlebox cooperation. In this deliverable this effort is complemented by a report on our proof-of-concept implementation of PLUS for QUIC as well as a description of the use of UDP options as an alternative approach for path signaling that can more easily be deployed as an extension to UDP. Experimental implementation of both of these has been completed by the MAMI project and made available as open source software.

This deliverable further reports on implementation and evaluation efforts for transport stack mechanisms that introduce signaling between the transport and network layers into existing protocols. These mechanisms are usually specific to a certain function only, however, their deployment strategy is more imminent as they utilize the extension mechanisms provided by these already deployed protocols, such as IPv6 options, DCSP, or TCP options.

Together with D3.2, which specifies PLUS as well as Post Socket, a protocol-independent API that supports flexible path and protocol selection in the transport layer, this deliverable presents the final outcomes of the WP3 activities at the conclusion of the MAMI project. While work on PLUS will be completed with the end of the project, other work in standardization, including follow-up work on a new transport API, will continue beyond the end of the project.

1 Introduction

The work in WP3 of the MAMI project focused on the development of an architecture for middlebox cooperation that eases the deployment of new services on the Internet. The project thereby utilised a measurement-based approach for the design of mechanisms and protocols in such a flexible transport stack [17]. The approaches and mechanisms described in this deliverable were therefore developed in cooperation with measurements performed in WP1, to detect network capabilities and flaws, as well as experiments performed in WP2, to evaluate functionality and performance.

As the core of a more flexible transport stack the project developed an explicit middlebox cooperations protocol, the Path Layer UDP Substrate (PLUS) protocol, which as been specified and discussed in D3.2. In this deliverable we report also on our (QUIC over) PLUS proof-of-concept reference implementation. However, given the deployment challenges of a completely new protocol and diverging options in the IETF, while standardization would have been a precondition for any deployment, the project decided to re-focus its effort on the extension and use of existing protocols and protocol mechanisms.

One such an example is UDP options which an extension to existing protocol (UDP [32]) and also provide header space for generic transport and network signaling, very similar to the shim layer approach taken by PLUS. UDP options have been discussed as an alternative signalling scheme to the explicit path layer approach as taken by PLUS from the beginning, e.g. also in the IETF PLUS BoF that was held in July 2016. In this deliverable we therefore also present and discuss an implementation of as well as testing and experimentation with UPD options, as an alternative middlebox cooperation and signalling approach.

More generally, the architectural approach taken by the project aims at introducing flexibility into the transport system that can enable easier Internet deployment of new mechanisms and protocols. To support this goal the project also developed and/or experimentally evaluated mechanisms or extension of various existing protocols that introduce signaling between the transport and network layer for a specific set of information. E.g. mechanisms for endpoints to detect and measure the properties of the end-to-end path such as the PMTU while using different signalling schemes, e.g. either UDP options or IPv6 HBH options.

This deliverable concludes with a summary of the main outcomes of WP3.

2 Core Flexible Transport Stack Implementation

In this section we present the results of an experimental evaluation of integrating Path Layer UDP Substrate (PLUS) [24, 25] as a shim layer to QUIC, as well as our experimental implementation of UDP options, as a lightweight alternative for transport and path signalling. However, the use of UDP options compared to the approach taken by PLUS trades-off some security goals, specifically data integrity protection and thereby better endpoint control about information exposure, with the desire to achieve more immediate deployment as it provides the ability to introduce signaling and transport parameters below any transport layer protocol using UDP.

Further, building on the exploration of Post Sockets (see Section 5 of D3.2 [24]) this section also describes on-going work to develop a protocol-independent abstract interface as needed for an architecture that provides flexible protocol and path selection as a transport feature. The current activity in the IETF on taps combines the initial work in this project with contributions from other project and various industry stakeholders to realise a set of specifications for an abstract API, as further detailed below.

2.1 PLUS Reference Implementation

This section describes a reference implementation of the PLUS protocol, as specified in D3.2 and published in [25], called plus-lib[1]. PLUS is a shim layer protocol between transport and network but layered on top of UDP for deployment purposes. The encryption context, needed in PLUS to protect the information carried in the header, is provided by the upper layer transport, as PLUS, as a shim layer protocol, does only enable signalling but does not implement own end-to-end functionality. Further, the transport protocol can be utilized to feed back information provided by the network from the receiver to the sender, if supported. QUIC was used as a reference transport protocol as it can provide an encryption context and also precisely as it addresses one of the goals of the project: enabling transport innovation [8].

PLUS for QUIC[2] is a fork of quic-go - an implementation of QUIC in Go. In our implementation QUIC has been extended with a new frame type, which we called a PLUS Feedback Frame, to handle feedback within PLUS.

Experience developing this reference implementation for PLUS complemented the security analysis for middlebox cooperation mechanisms in Section 4 of D3.2 [24]. This security analysis was extended into a white paper which explores middlebox cooperation specifically from the point of privacy [19]. The white paper has been published for industry exploitation on the MAMI webpage and concluded that PLUS specifically or Middlebox Cooperation Protocols in general will likely not enable completely new attacks, although may facilitate already existing attacks.

2.1.1 A PLUS API

In our implementation, we aimed to provide an API that facilitates easy integration into existing code. This required to make the PLUS interface look like the network layer interface as much as possible, such that it could serve as a drop-in replacement for other networking APIs and

[1] https://github.com/mami-project/plus-lib
[2] https://github.com/mami-project/puic-go

enable fully transparent use of PLUS.

However, PLUS-aware transport protocols might desire more control over the signalling mechanism PLUS provided, such as setting specific flags or access to packet numbers.

Existing code could also make use of connection multiplexing through a datagram socket, in which case there would be two layers of connection multiplexing. This implies an approach may need to provide its users with different ways of interfacing with PLUS.

Our PLUS API be accessed in three modes:

- Low Level: Low level functions to send and receive PLUS packets.
- Semi Transparent: Connection-style interface with various callbacks as well as support for connection multiplexing.
- Fully Transparent: Very similar to semi transparent but also handles feedback autonomously without support of the upper layer protocol.

Plus-lib uses an underlying packet connection, but could also be used as a pure state machine with no connection. This gives users more control, because they can directly use a connection of their choice with full access to it and then feed the data received into the state machine of plus-lib.

Since the Go implementation of QUIC already handled multiplexing, the low level interface of our PLUS library was used to integrate PLUS support into QUIC.

2.1.2 Open Source Status

The reference implementation of PLUS for QUIC for endpoints is made available as open source at `https://github.com/mami-project/puic-go`. This source code has been used in the project for internal experimentation and as a proof-of-concept implementation but will not be maintained beyond the end of the MAMI Project.

This implementation was used to conducted a series of tests in order to explore the effectiveness of the approach, and verify that the required functionality could be realised. However, our implementation has not be optimized for performance or robustness and as such these aspects have not been systematically evaluated. Still, without any optimizations and under non-impaired conditions (fast network, little load) our implementation of PLUS over QUIC added only about 7% overhead over a raw QUIC connection, while if there is additional protocol overhead, for example with H2QUIC, the overhead from PLUS became negligible.

In addition, an open source implementation of PLUS support for gopacket[3], which provided packet decoding for Go, was realised and is available from `https://github.com/mami-project/plus-gopacket`.

2.2 A Transport Stack with UDP Options

UDP Options [37] is an experimental extension to UDP to provide a mechanism that allows transport options to be added as a trailer to any UDP datagram. A generic syntax for UDP

[3] `https://github.com/google/gopacket`

options is defined assuming a stateless, unreliable message protocol. The method is currently an IETF work-in-progress within the TSVWG working group targeting publication after 2018.

UDP Options does not rely upon or modify the payload of a UDP datagram and hence could be used as a shim layer with an unencrypted or an encrypted end-to-end transport protocol. The current method does not provide cryptographic protection of the options information (compared to the use of encryption/integrity checks in PLUS, Section 2 of D3.2 [24]). UDP options therefore explicitly exposes end-to-end information visible on the path, but does not expect the path to modify this information [37].

UDP Options adds the opportunity for feedback at the UDP transport layer, i.e., the return of transport information from the receiver back to the sender. Protocol mechanisms proposed [37] include the ability for an endpoint to inject a protocol-independent timestamp to allow measurement of the path RTT (both for latency measurement and to assist in setting appropriate timers for transport and upper layer protocols), signaling to advertise the remote maximum packet size, probe mechanisms for a challenge/response mechanism that could be used to validate an operational end-to-end path, and support for fragmentation and reassembly.

Like other MCP methods proposed by the MAMI project [24], use of the UDP Option method does not require changes to applications to enable deployment of new transport mechanisms. Since the approach is also extensible, further options can communicate other endpoint information or additional signaling from an application.

2.2.1 Use of UDP Options over Middleboxed Paths

MAMI Deliverable D1.3 describes measurements to understand the transparency of paths when packets carry UDP Options. As expected, these results show a variety of middlebox behaviours. Some middleboxes on the path incrementally update the UDP checksum [22] and are therefore transparent to use of UDP Options, but there are also middleboxes that discard UDP packets that can not validate a UDP checksum. For example, a default behavior for a stateful firewall is to parse a packet, but to not parse packets with an invalid checksum. This will necessarily discard all datagrams where the UDP checksum can not be verified.

A complication was discovered, since measurements showed that some middlebox and endpoint implementations did not verify the checksum using the UDP length (as required in the RFC series), but instead used an updated IP length to perform the UDP checksum calculation [15]. A solution to this middlebox impairment has been proposed by the project to increase the deployability of this new transport [37]. This new method was implemented and experiments were performed by the MAMI project (described in D2.2).

A small number of on-path middleboxes were also found to verify that the UDP datagram length was exactly consistent with the IP payload size (D1.3). Although the RFC series does not require this check, this verification is consistent with a firewall trying to prevent a covert-channel being exposed. These middleboxes consequentially discard any datagrams that carry UDP Options. From the perspective of deploying UDP Options on Internet paths, this presents an unwanted ossification, and middleboxes would need to be updated to present a transparent path.

2.2.2 UDP Options Socket API

UDP Options can be used to perform tasks that UDP applications are commonly required to, to make the most efficient use of the network. Path MTU Discovery is a common requirement for UDP applications that wish to use packet sizes larger than the minimum without suffering the disadvantages of network fragmentation.

UDP Options offers an API to application protocols so they may delegate common tasks. The UDP Options API extends the services offered by the UDP and UDP-lite APIs [13] with new features that can only be offered by adding transport options to UDP.

2.2.2.0.1 PLPMTU API The PLPMTU API enables an application to delegate Datagram Packetization Layer Path MTU Discovery to the UDP stack. The PLPMTUD service is only available on sockets that are connected. The API offers the following actions:

- `perform plpmtud (set|get)` Enable performing PLPMTUD for this flow.
- `search step size (set|get)` Set or retrieve the step search size for PLPMTUD.
- `target plpmtu (set|get)` Set or retrieve a target PLPMTU for the DPLPMTUD search. In some cases an application will know the maximum MTU required, this allows the application to provide a hint to the search algorithm.
- `plpmtu (get)` Retrieve the PLPMTU discovered by DPLPMTUD.

2.2.2.0.2 Socket API Examples The above APIs are implemented as extensions to the Socket API in the FreeBSD Operating System. Get and Set operations are made available to applications via the `getsockopt` and `setsockopt` Socket API calls.

As UDP Options inherently adds soft state to UDP, UDP Options calls are only valid on connect UDP Sockets, that is sockets where a `connect` call has been performed. For ICMP errors to be reported to a UDP application it must have connected its sending UDP socket.

Applications must register to use UDP Options, this is performed by a call to `setsockopt`. UDP Options use can also be configured system wide and is controlled by the `net.inet.udp.process_options` sysctl system control.

Applications must register their socket before the stack will respond to UDP Options REQ messages with UDP Options RES messages, this is performed by making a call to `setsockopt`.

Listing 2.1 shows the Socket API calls required to perform Datagram PLPMTUD with UDP Options.

Listing 2.1: Example of usage PLPMTUD from the socket API

```
int plpmtu, optval = 1;
int sockfd = socket(AF_INET6, SOCK_DGRAM, IPPROTO_UDP);

bind(sockfd, (struct sockaddr *)&sa, sizeof(struct sockaddr));
connect(sockfd, (struct sockaddr *)&sa, sizeof(struct sockaddr));

setsockopt(sockfd, IPPROTO_UDP, UDP_OPT, &optval, sizeof(int))
setsockopt(sockfd, IPPROTO_UDP, UDP_OPT_ECHO, &optval, sizeof(int))
setsockopt(sockfd, IPPROTO_UDP, UDP_OPT_PLPMTUD, &optval, sizeof(int))

sendto(sockfd, plpmtu, sendsize, 0, p->ai_addr, p->ai_addrlen))
...
getsockopt(sockfs, IPPROTO_UDP, UDP_OPT_PLPMTU, &plpmtu, sizeof(int));
```

2.2.3 Open Source Status

The UDP Options implementation developed by the MAMI project is available as an open source reference implementation on the FreeBSD operating system [4]. This is provided as an extension of the current UDP implementation, adding a small number of socket API calls to enable their use. The reference implementation is being maintained current to the draft. Work is being conducted now to write an automatic regression suite to provide a continual measure of implementation correctness.

We expect the UDP Options specification to be completed and finally published by the IETF after conclusion of the MAMI project. A published RFC is expected to allow implementations of UDP Options to be upstreamed to open source operating systems distributions, enabling future exploitation of the method by application developers.

2.3 A Standards-based Abstract Interface for Transport Services

There are many variables that determine whether a transport stack (with or without an explicit signaling mechanism) may be used on a given path – e.g. application layer requirements, availability of transport features from an endpoint's kernel, impairments to the use of transport protocols or extensions by middleboxes along the path, etc – *transport agility* is a key feature needed of any flexible transport layer. However, a facility supporting dynamic selection of transport protocols and features is useless for application development, unless it can present a unified interface to the application for each transport stack.

To solve this problem, the MAMI Project proposed Post Sockets [39] (Chapter 5 of D3.2). This provided a basis for the project to work together with partners from the now concluded H2020 NEAT project [5] to standardize an *abstract interface* to the transport layer in the IETF TAPS

[4] https://github.com/uoaerg/freebsd
[5] https://www.neat-project.org/

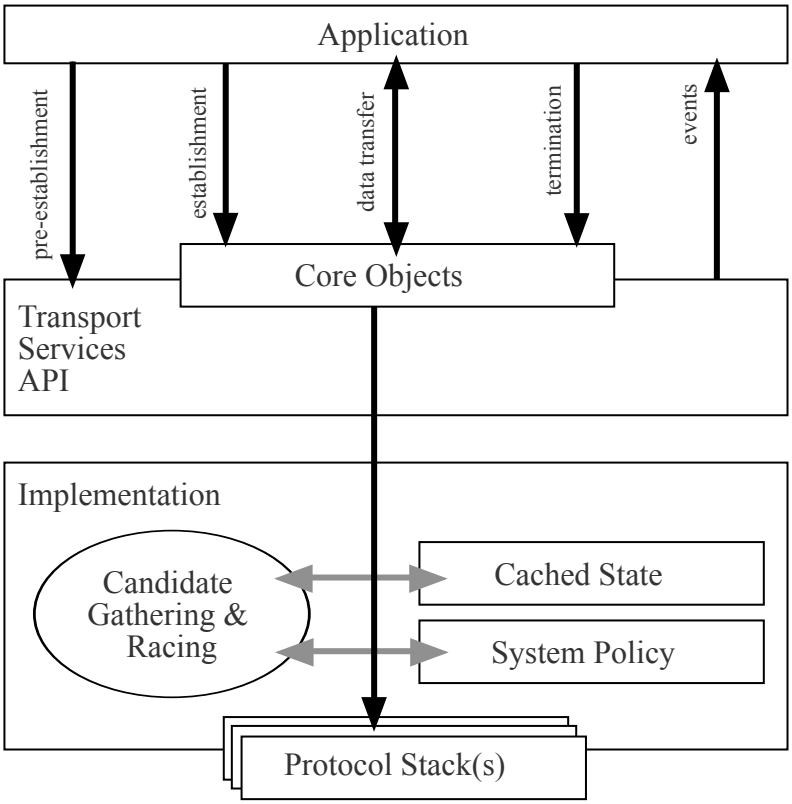


Figure 1: The TAPS architecture. An abstract API wraps a transport system implementation, which selects and maintains connections bound to protocol stack instances.

working group. This effort can be seen as the synthesis of a number of contributions: The Post Sockets work, focused on a clean API to a set of transport-independent transport layer semantics; the NEAT API, focused on clean dynamic selection of transport protocol stacks at runtime, based on application requirements, system policies, and network conditions; with additional input from Socket Intents [34], providing a higher-level view of the implementation of application policies; and Apple's Network. Framework[6], which provides implementation experience with a transport-independent, TAPS-like stack.

The remainder of this section summarizes the current state of the TAPS work, focusing on changes relative to Post Sockets. The descriptions are drawn in part from the TAPS architecture [31] and interface [40] documents, co-authored by MAMI Project participants.

The TAPS architecture is detailed in [31], and shown in Figure 1. The design of the abstract interface within this architecture is based on a small set of first principles, largely matching those in Post Sockets:

- **Common functions have a common interface, while functions specific to transport stacks remain accessible.** The interface can be used in a variety of application design patterns, independent of the properties of the application and the transport stacks that will be used at runtime. All common specialized features of these protocol stacks are

[6] `https://developer.apple.com/documentation/network`

made available to the application as necessary in a transport-independent way, to enable applications written to a single API to make use of transport protocols in terms of the features they provide.

- **Data transfer takes place in terms of messages,** objects of a finite length that are transmitted atomically whenever possible, using application-assisted framing and deframing where the underlying transport does not provide these.
- **All operations involving a remote peer are asynchronous,** allowing most application interactions with the transport layer to be Event-driven, in line with developments in modern platforms and programming languages.
- **Security properties of transport stacks are integrated into the interface,** in contrast to the current transport interface architecture that treats security as an optional "add-on".
- **Explicit support for multistreaming and multipath transport protocols,** allowing applications to take full advantage of new transport protocols supporting these features.
- **The interface definition remains abstract**, describing the "shape" of interactions between the application and the network stack, as opposed to defining specific entry points and interaction methods that

The interaction between the application and the transport system is split into five *phases*:

- **Pre-Establishment** encompasses the properties that an application can pass to describe its intent, requirements, prohibitions, and preferences for its networking operations. For any system that provides generic Transport Services, these properties should primarily be defined to apply to multiple transports. Properties may have a large impact on the rest of the aspects of the interface: they can modify how establishment occurs, they can influence the expectations around data transfer, and they determine the set of events that will be supported.
- **Establishment** focuses on the actions that an application takes on a pre-connection objects to prepare for data transfer.
- **Data Transfer** consists of how an application represents data to be sent and received, the functions required to send and receive that data, and how the application is notified of the status of its data transfer.
- **Event Handling** defines the set of properties about which an application can receive notifications during the lifetime of transport objects. Events can also provide opportunities for the application to interact with the underlying transport by querying state or updating maintenance options.
- **Termination** focuses on the methods by which data transmission is stopped, and state is torn down in the transport.

Details of the objects, actions, and events available for interaction from the application, and the generic and specific properties available on are given in the "Abstract Application Layer Interface to Transport Services" draft [40].

2.3.1 Status of TAPS Standardisation

This section has provided a snapshot of an ongoing effort, which will continue beyond the end of the MAMI project; the TAPS Implementation [6], Architecture [31] and API [40] documents should be consulted for up-to-date information about TAPS. These details are expected to change rapidly as the documents are completed. Final publication of the TAPS interface as a proposed standard is scheduled for 2019.

3 Specific Transport Signaling Mechanisms and Implementation

This chapter describes transport mechanisms developed by the MAMI project that allow the transport system to signal to the path or probe the path to influence the way middleboxes forward packets, utilising or extending existing protocols and mechanisms. Example mechanisms have already been described in Section 3 of D3.2. This deliverable, in addition, reports on implementation and experimentation to explore the design options and expected drawbacks/ benefits of each proposed approach. Final conclusions are drawn from exploratory research performed in various testbeds (also see D2.2) as well as measurements (see D1.3 more details on the measurement setup), enabling the final design to be robust to path impairments.

Two types of signaling are described: Explicit signaling to network devices on the path by the sender, which can change the way a path forwards the packets it receives from the endpoint transport; and explicit signaling by network devices on the path to the sender, which can influence how a transport sends packets and allow the endpoint to adapt to feedback received from the network.

3.1 Explicit Sender-to-Path Signaling

Many existing protocols already have explicit sender-to-path signaling mechanism that provides unencrypted, possibly authenticated, information to the network path. This signaling can be used to influence the forwarding behaviour of the packets within the network or to reveal information about the transport usage of the network. Usually on-path devices can read these signals, but not verify their authenticity. (Example use cases for the mechanisms have been described in section 1.2 of D3.2).

Usually, the path signals can be read by network devices on the path, but are not expected to be modified by on-path devices. In modern protocols (e.g., the spin-bit in QUIC), modification to these signals can be detected by the receiver by verifying the end-to-end integrity of the information. In other cases (e.g., DSCP marking), the information may be modified by network devices on the path, usually with the intention to impact network treatment further down the path.

3.1.1 Differentiated Services Code Point (DSCP)

The Differentiated Services (DiffServ) is an existing architecture introduced as a scalable solution for providing consistent treatment of traffic classes (traffic with similar QoS requirements) along a path. A Differentiated Services Code Point (DSCP) mark in the IP network header is used to request that the packet receives a specific forwarding treatment (PHB) at each network device on the path. The IETF also defines a range of standard PHBs and associates each with a well-known DSCP or collection of DSCPs, including the Assured Forwarding (AF) class and Expedited Forwarding (EF) class. Since its initial specification it has been integrated across all mail network technologies and all major operating systems.

DiffServ is commonly used for prioritising traffic in VoIP networks within a domain, with leading

network equipment vendors supporting this feature. There are several other ways an operator can use DSCP inside their domain, depending on the type and scope of network.

Although DiffServ has been used in networks under the same administrative domain, there are also opportunities for using DiffServ across multiple domains along an Internet path, and IETF debates show an interest in maintaining compatible DSCP markings across interconnected domains. To explore the potential to increase inter-domain interoperability requires analysis of the prevalent remarking behaviour for DSCP values. In [10] we characterized the DSCP modification pathologies in a variety of operator networks, including mobile paths.

While there is evidence of operator configuration using DiffServ, much of the observed remarking appears to arise from routers configured to use historic ToS semantics, which in some cases results in priority inversion. Compared to the core of the Internet, mobile networks aggresively remark DSCP codepoints.

The DiffServ architecture is extensible, and new forwarding behaviours (PHBs) can be defined by the IETF. Remarking pathologies are important in identifiying a suitbale to DSCP to associate wit any new PHB. Presentation of an analysis of DSCP-remarking [35] by the MAMI project informed the decision to the change the IANA registration procedure [12] to allow a more transparent DSCP to be used for a new Lower Effort, LE PHB [3] designed to support scavanger traffic.

3.1.1.1 Current Status

The results suggest, there are no significant downsides to applications and transport stacks enabling the use of codepoints by applications as there is very little evidence of loss due to the usage of a specific codepoint, both in the core of the Internet and in edge networks. As older or misconfigured routers functioning on ToS semantics are retired by network operators, there is potential for realizing end-to-end QoS using DSCP.

Following the choice of codepoint, the LE PHB specification is now progressing in the IETF TSVWG working group, with the expectation to publish in 2019. This demonstrates the influence of measurements on the directions of standardisation and the continued desire to add to the existing DiffServ architecture.

3.1.2 The LoLa Signaling Mechanism

Loss-Latency Tradeoff (LoLa) is a signaling mechanism defined by the MAMI project that provides an sender-to-path signal that allows an endpoint to indicate the treatment that packets in a flow are expected to receive from network devices along the path to the remote endpoint. This is especially interesting for mobile network where bufferbloat is still prevalent. On this kinds of paths, LoLa would help interactive applications to better behave, and optimise its use of capacity.

One example where LoLa would be useful is a mobile terminal running a WebRTC session (or another interactive / real-time application) while at the same time performing a file download – a pretty typical “rich app” scenario. Using LoLa to route the real-time flow onto a dedicated, low-latency Evolved Packet System (EPS) bearer would increase its chance to stay within the expected delay budget at the risk of increasing the probability of losing packets on the path.

Segregating the real-time flow from its throughput-seeking twin would decrease its end-to-end delay by avoiding mixing both into the same (deep) buffer allocated to the User Equipment (UE) at the base station.

Taking advantage of LoLa requires a mobile network to trust the sender-to-path signal and act upon it. While the trust aspects have been sorted out in D3.2 and PLUS-red-team and need not be restated (in summary, there is no trust issue both ways), However, LoLa is explicitly designed as a tradeoff signal where an endpoint does not have an advantage to lie about, given there is only a benefit if the traffic

acting upon the signal needs some extra configuration steps in the mobile segment for setting up a dedicated low-latency EPS bearer per UE in addition to the default one.

This could be done preventively at attachment or triggered opportunistically by matching a LoLa-specific Traffic Flow Template (TFT) either at the packet Gateway (PGW) (downlink) or on the UE itself (uplink). However, this is not the current operational practice; therefore, a primary objective for this work item is to provide compelling evidence of the advantages of LoLa over the status-quo to the network operators and their users in order to modify the inertia. This is where the LoLa testbed, itself an instance of the GSMA Content Classification (CC) experiment [18], plays a key role. The experiment is currently on hold at the 5TONIC lab due to a missing feature in the LTE core. We are not sure whether it will be installed during the lifetime of the MAMI project. Therefore, we are currently translating the experiment into NS-3 and expect to disclose and discuss the results of the simulation at IETF 103 and next GSMA IG meeting (both due by end of November 2018). In addition, all the elements of the LoLa experiment, implemented as Virtual Network Functions, have been transferred to 5TONIC and will be used (and evolved) in the context of future research projects.

3.1.2.1 Current Status

When and if the LoLa hypothesis is validated, a further step will be to identify where the signal needs to be set: at the IP layer (as a DSCP), at the transport substrate (as a UDP Option) or at the transport itself (e.g., as a part of the QUIC header). Each option has pros and cons, and the best answer may be that there is not one true answer; instead any of these strategies can be used by the applications either alone or together on the same or different paths.

3.1.3 Explicit Support for Passive Latency Measurement in QUIC

The latency spin signal has been defined by the MAMI Project. In the absence of a middlebox cooperation protocol, the latency spin signal, initially documented in Section 3.2 of Deliverable 3.2, allows passive latency measurement even on encrypted transport protocols. This provides one RTT sample per RTT to passive observation points, even to network devices that can only see traffic in one direction.

Recent work includes enhancement of the signal to improve operation in challenging network conditions (i.e., loss and reordering up to the tolerances of the transport protocol itself) and implementation both on QUIC and TCP, as described in a paper to appear at ACM IMC 2018 [41][1]; and the continued effort to add the spin signal to the specification of the QUIC protocol under

[1] The description of the signal in this section is excerpted from this paper

1 $spin_{next}, vec_{next}, t_{last}, PN_{max} \leftarrow 0, 1, 0, 0$
2 **Function** `OnPacketReceive()`**:**
3 | **if** $PN > PN_{max}$ **then**
4 | | **if** $spin_{next} \neq spin_{rcv}$ **then**
5 | | | $vec_{next} \leftarrow min(vec_{rcv} + 1,\ 3)$
6 | | | $t_{last} \leftarrow t_{sys}$
7 | | **if** *is_client* **then** $spin_{next} \leftarrow \neg spin_{rcv}$
8 | | **else** $spin_{next} \leftarrow spin_{rcv}$
9 | | $PN_{max} \leftarrow PN$
10 **Function** `OnPacketSend()`**:**
11 | $spin_{snd} \leftarrow spin_{next}$
12 | **if** $t_{sys} - t_{last} > delay_{max}$ **then**
13 | | $vec_{snd} \leftarrow min(vec_{next},\ 1)$
14 | **else** $vec_{snd} \leftarrow vec_{next}$
15 | $vec_{next} \leftarrow 0$

Algorithm 1: Logic of the spin signal. t_{sys} is the current system time, PN_{max} the currently highest packet number.

standardization in the IETF.

The latency spin signal is composed of two parts: a spin *bit* that changes once each RTT, and a 2-bit Valid Edge Counter (*VEC*) that indicates the validity of the latency information between two spin bit toggle events, which are called edges. These three bits appear on every packet sent by each side of a transport connection. The generation of the signal does not require the generation of additional packets not otherwise sent by the transport, and does not interfere with the transport's own transmission scheduling algorithms. The mechanism is lightweight, and can be added to a transport protocol with minimal effort.

The spin bit itself is the part of the spin signal that is used to actually monitor the Round-Trip Time (RTT) of a flow, as it is toggling once per RTT. A passive on-path observer can log the period between two transitions and thereby extract a flow's RTT.

This toggling behavior is illustrated in Figures 3.2(a) to 3.2(c). When the client initiates a connection, it will start sending packets with spin 0 (Figure 3.2(a)). Once the server starts sending packets, it will echo back the spin value it last received from the server (Figure 3.2(b)). Conversely, once the client receives a packet from the server, it will set the spin of its outgoing packets to the opposite of the spin value it last received from the server (Figure 3.2(c)). This asymmetric behavior will cause the transition point of the spin values to 'spin' through the network, resulting in exactly one spin edge per RTT at any point in the network and thereby exposing the flow's RTT between two edges. Furthermore, when logging the duration between two edges on different flow directions, the delays from the observation point up- and downstream to each of the endpoints can be measured separately — we refer to these as *component RTT*. This part of the algorithm is described in Lines 1, 7, 8 and 11 (yellow) of Algorithm 1.

To provide resilience to packet reordering (Figure 3.2(d)), the algorithm ignores all packets that were received out of order. Figures 3.2(e) and 3.2(f) illustrate how this mechanism removes distortions in the spin bit sequence. In Lines 3 and 9 (green) of Algorithm 1 we assume that every packet has a unique, in-order sequence number (PN) which is at least visible to the

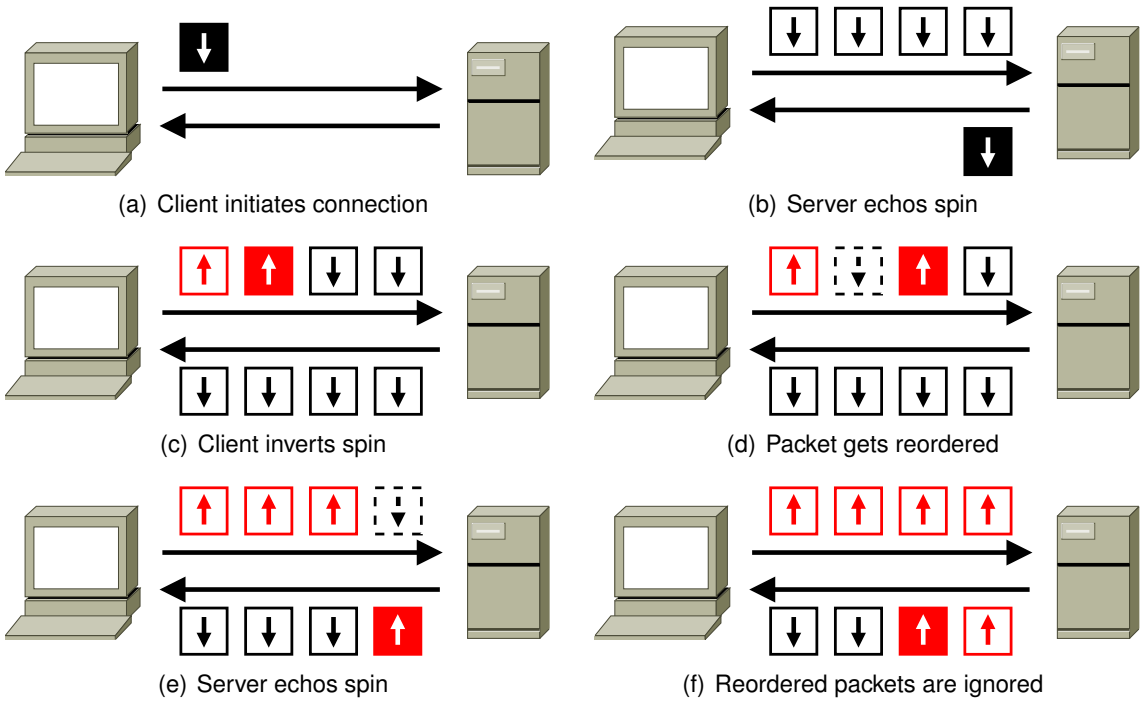


Figure 2: The spin bit mechanism: arrows indicate spin values, filled packets have a non-zero VEC value, and dashed lines indicate reordered packets.

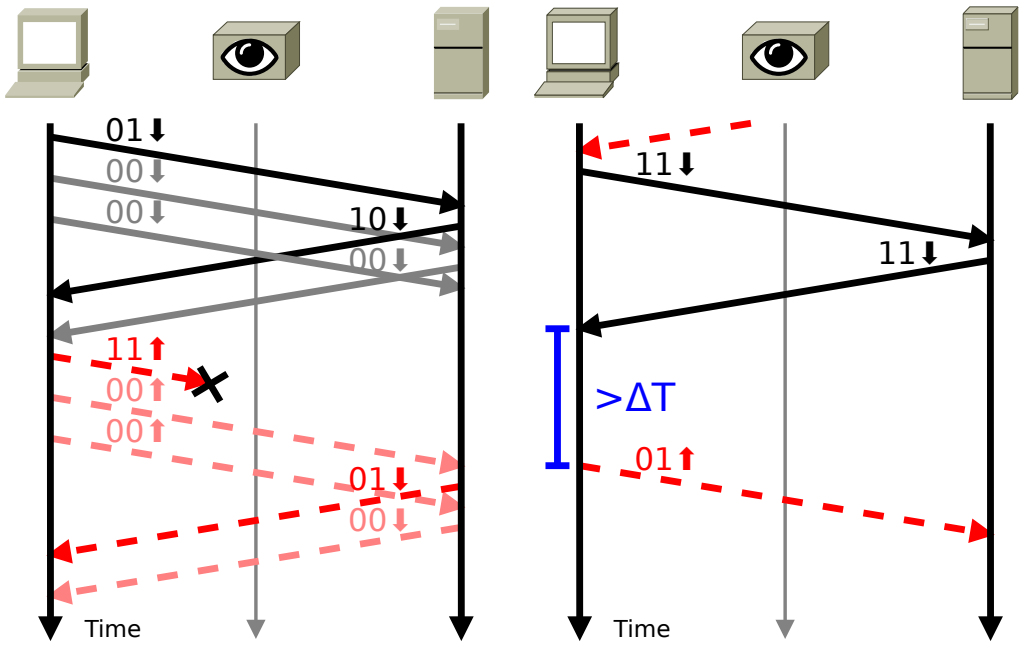
endpoints, such as the sequence number in TCP or packet number in QUIC.[2]

While the endpoints can trivially detect reordering by observing the packet sequence number, this does not always hold for on-path observers, because sequence information may be encrypted (as it is the case with the QUIC packet number). An observer considering only the spin bit would then incorrectly report two very short RTT samples when observing a reordered packet, as illustrated in Figure 3.2(d). To address this problem, we introduce the VEC, a two bit signal that explicitly marks packets that carry a valid spin bit edge set by the endpoint, and increases its value with the number of non-distorted transitions of the spin signal. Figure 3.3(a) illustrates this mechanism which is described in Lines 4, 5, 14 and 15 (blue) of Algorithm 1. Non-edge packets carry a VEC of zero. All spin edges carry a nonzero VEC value, which is set to one plus the VEC value of the packet that triggered the edge transition, clamped to a maximum of three. Thereby, the VEC indicates the number of network transitions during which the spin signal has not been distorted. With this mechanism, even an observer that can only monitor one direction of a flow can also detect distortions that occurred in the other flow direction.

As observer can estimate the validity of their measurements based on the VEC, the end hosts can also use the VEC to signal when they *know* that the spin bit does *not* carry valid RTT information. This situation occurs when an endpoint introduces excessive delay between receiving and transmitting a packet carrying a spin edge (e.g., because of application-limited traffic). This is done by the remaining (uncolored) lines in Algorithm 1, and is illustrated in Figure 3.3(b). The endpoint will reset the VEC to its initial value of '01'.

Endpoints can also explicitly indicate that they have opted out of RTT measurement by set-

[2]The correctness of the spin bit is shown in the following thesis report [41].



(a) Increment VEC (with edge loss) (b) Reset VEC on invalid edge

Figure 3: The VEC mechanism: VEC values are represented in binary format; spin values as arrows.

ting the VEC to '00' on all packets, or control the sample rate by probabilistically marking an otherwise non-'00' edge as '00'.

3.1.3.1 Current Status

The latency spin signal was accepted as an experimental addition [38] to the QUIC transport protocol at the IETF 101 meeting in London in March 2018[3], and as an addition to the standard at the IETF 103 meeting in Bangkok in November 2018. The effort to build consensus within the IETF to add the spin bit to the protocol defintion was led by the MAMI project, in concert with other telecommunications equipment vendors and browser implementors.

The fine details will be worked out within the working group; however, the consensus agreed in Bangkok is that:

- the one-bit variant of the spin signal will be added to the protocol; the VEC is not included, as the working group felt that simple heuristic detection of loss and reordering of edges was good enough for production use, and the two extra bits required for VEC signaling therefore represented too much overhead
- Spin-bit implementation and participation are optional, and endpoints should be user-configurable to expose the signal or not. However, endpoints implementing the spin bit should probabalistically (as a proportion of flows and/or peer connections) refuse to par-

[3] see `https://www.youtube.com/watch?v=TQq6Z4_HBaY&feature=youtu.be&t=1276` for video of this discussion within the QUIC WG

ticipate, in order to ensure a large anonymity set in which clients and servers that opt out can hide.

3.2 Explicit Path-to-Sender Signaling

Path-to-sender signaling mechanisms provide unencrypted information from on-path network devices that can influence the way sending endpoint controls the sending of packets to the network.

Often, first a path-to-receiver signaling mechanism is utilized, initiated by the sender, to provide ”scratch space” in the packet that is writable by on-path devices (also see section 1.2 of D3.2 for use cases for this information). Depending on the usage, middlebox may explicitly indicates support or, alternatively, the endpoint may need to probe the path to discover if a feature is supported. In practice a combination of the two approaches is often desired where the endpoint explicitly validates that the path behaves as expected.

3.2.1 IPv6 Hop-by-Hop MTU Option

There is currently revived interest in the IETF in defining an improved network signal for Path MTU discovery. The HBH MTU option is one such effort to add a mechanism to provide sender to path signaling by defining a new IPv6 Option to the IPv6 Hop-by-Hop extension header.

IPv6 extension headers [11] encode optional Internet-layer information between the IPv6 and upper (e.g transport) layer. A Hop-by-Hop (HBH) Options Extension header must be examined by every hop that a packet visits en route to its destination and can be used to signal data to every router on the path. The HBH extension header contains options which can be useful to routers' control and forwarding planes, such as indicating Jumbo payloads [4], or router alerts [16].

The HBH architecture is extensible, and new options can be defined by the IETF. Traversability of packets carrying extension headers and router handling behaviour must be observed before these can be used in practice. There are issues with the implementation of IPv6 extension headers. Middleboxes providing in-network functionality must parse the headers before inspecting the upper layer protocol. This has historically resulted in failure as the options did not have a uniform TLV format and were numbered outside of the IP header protocol space [7].

The MAMI project has collaborated on an Internet Draft within 6MAN working group proposing the new HBH option to be used in recording the minimum MTU along a path between source and destination nodes [20].

The new option signals the minimum MTU a link supports. When a router sees the MTU option it compares the MTU field in the option with its own forward MTU, replacing the MTU in the option if smaller. This mechanism could provide a hint that a larger MTU may be supported and inform Packetization Layer PMTUD about a suggested size to probe. If successful, this significantly reduces the time to discover a larger workable PMTU. We have explored this by providing a proof-of-concept implementation of the new option in FreeBSD [4].

[4] `https://github.com/adventureloop/freebsd/tree/thj/hbhmtu`

3.2.1.1 Preliminary experimentation

The MAMI project has carried out Hop-By-Hop experiments using PATHspider to see how HBH options perform in the Internet. The authoritative DNS servers for the top 1M domains were tested with DNS queries with and without HBH options. For the 16057 servers tested the success rate dropped from 99.1% to only 0.2% when using HBH options. The results suggest there are still many routers and stacks which drop packets with the Hop-by-Hop extension header.

While HBH options are not usable in the current Internet, they can be deployed in data centers or under the same administrative domain. These measurements validate the intended goal of draft-hinden-6man-mtu-option to add sender to path signals in data center environments.

3.2.2 The Datagram PLPMTUD Mechanism

Path MTU discovery is the technique used by an endpoint to discover the largest size of packet that may be sent along a network path. The method relies on reception of the network-to-path signals sent by on-path network devices, known as ICMP PTB mesages. Middleboxes within the network can make this signaling unreliable and prevent the algorithm functioning, resulting in black holing of packets larger than the actual PMTU.

3.2.2.1 The Need for Robust PMTU Discovery

MAMI research has completed an initial analysis of the short-comings of the currently standardised PMTUD algorithm. This analysis reveals fundamental issues with a network-layer solution. Despite attempts to revive work on PMTUD for IPv6 [36], it is now becoming accepted amongst network equipment vendors that the current approaches are unable to supply the required robustness needed. Not only are PMTUD methods unreliable because of middleboxes and routers not returning ICMP messages, they also face vulnerabilities to off-path attack.

A common current work-around for TCP is for on-path middleboxes to change the TCP control information specifically clamping the advertised MSS option to avoid senders trying to send packets that are larger than a path can sustain. This approach appears to be widely used (in various forms) and is what currently offers stability for applications using TCP. However, the approach has a number of fundamental pitfalls. First, it relies on all packets for a connection flowing through the same middlebox an unnecessary constraint on routing/forwarding decisions. Second, it requires the middlebox to parse the end-to-end transport header to locate the MSS Option (if any) and change the value. Some devices even insert a MSS option when this was not originally present.

Evidence shows that in many cases, the equipment reducing the MSS is not the one presenting the bottleneck on the path [29], but instead the clamping is done by another provider seeking to avoid connectivity failure [9]. In this respect, MSS Clamping presents an unwanted ossification of the network. It prevents an application from ever being able to send a TCP segment larger than the clamped MSS size, even when a has been upgraded to support this.

The MSS Clamping approach is clearly also unable to work with an encrypted transport protocol using UDP, since even if a UDP version of the MSS is negotiated in some way, the negotiation

could be encrypted and is not observable by a middlebox.

3.2.2.2 A Datagram PLPMTUD Method

PLPMTUD [30] describes a standards-track mechanism to perform PMTUD at the packetization layer (PL) without reliance on ICMP PTB messages, primarily targeting TCP. This method uses path probing techniques to allow an application to confirm an acceptable maximum packet size. When the path can provide explicit feedback signals (PTB messages) from on path network devices, these can be utilised, and the path properties confirmed by the method. Current implementations of TCP do not by default support PLPMTUD. The existing PLPMTUD method also does not specify how to use this with UDP-based transports.

An Internet Draft describes work to develop a PLPMTUD for datagrams [14]. This provides a solution to these problems that allows a UDP-based protocol to discover a suitable maximum datagram size. The method was designed based on requirements derived in the EU H2020 NEAT Project, and benefiting from widescale measurements performed in WP1 of the MAMI Project, leading to implementation of the current specification in WP3.

DPLPMTUD evaluation is being performed on lab testbeds and real networks. This has resulted in the project producing three implementations of DPLPMTUD. One is a test tool that uses a custom application protocol over UDP, one is an experimental implementation for QUIC, and a final one is an implementation using UDP Options.

3.2.2.3 Current Status

The DPLPMTUD specification is a work item of the TSVWG working group, and the base algorithm is now complete. Additional work is required to evaluate the robustness and protocol modifications to IETF transport protocols, with expected final publication of a proposed standard expected in 2019/2020.

3.2.3 Explicit Congestion Signaling

Explicit Congestion Notification (ECN) [33] is a standardised TCP/IP mechanism for signaling congestion information from the network to the endpoints. Using this method, network devices on the path use a field in the IP header (network layer) to indicate incipient congestion. This information is then observed by the receiving endpoint and fed back to the data sender on the transport layer. The sender then uses the feedback information as input for congestion control, regulating its sending rate on the congested link. RFC3168 only defines a feedback scheme for TCP that provides only one congestion indication per end-to-end RTT. Given congestion control always operates on a per-RTT basis, this was seen as sufficient when ECN was first specified. However, modern congestion control also take the extent of congestion (how many signals where observed in each RTT, see e.g. DCTCP) into account in determining their congestion reaction. In 2018, RFC8311 [2] reopened the topic of how ECN can be used across the Internet, enabling new ECN-based congestion reactions to be research and ultimately standardised.

3.2.3.1 Accurate ECN Signaling from the Path

Accurate ECN (AccECN) [5] is an alternative experimental feedback scheme for TCP that also provide information about how many indications where observed in each RTT. However, the exact representation of this information can be optimized differently for different use cases, e.g. is it also important to know the exact order of markings or just how many have been there? These trade-offs led to continuous discussion in the IETF TCPM working group starting before the beginning of the project with involvement of project partner.

Work to provide a similarly accurate feedback mechanism for QUIC is on-going with the input and involvement of the project based on the experience from designing AccECN for TCP.

While initial deployment problems for a long time hindered wide-spread support and use of ECN, the project has supported the increasing deployment of ECN on endpoints that we see todays by providing continuous measurements of potentially ECN implications, with a server-side support rate of more then 80 percent now and default probabilistic use of ECN on MacOS/iOS and Linux hosts. Still, as networks aim to minimize loss, often the benefit of just replacing loss with ECN marks is not seen as overly beneficial. Providing more accurate feedback in both, TCP and QUIC, will likely foster ECN deployment as a way to experiment with and deploy more advances congestion control schemes.

3.2.3.2 Congestion Exposure to the Path

MAMI Deliverable D3.2 described how ECN information can be fed back into the network as a network-visible signal to estimate the current on-path congestion level, as specified by the Congestion Expose (ConEx) protocol for IPv6 [23]. Implementing a similar signaling scheme was also discussed for QUIC.

3.2.3.3 Current Status

At the conclusion of the MAMI Project, ECN continues to be an active topic for network operators and protocol developers. This is expected to continue beyond 2019.

Members of the MAMI consortium were also a part of the design team that has introduced the initial design of an ECN feedback mechanism for the QUIC protocol. It is expected that new versions of the QUIC protocol will add support for ECN in a similar way as AccECN does for TCP, and new transport protocol methods will emerged that effectively utilise the services offered by advanced ECN methods.

3.2.4 Explicit Capacity Signals

This subsection describes Mobile Assisted Rate Control (MARC) an experimental TCP congestion control algorithm produced by the MAMI Project, based on explicit path-to-sender signals from on-path network devices. The method is designed to overcome the poor performance of standard TCP in cellular networks. A MARC TCP sender quickly adapts its sending rate to the capacity available to a mobile user by exploiting signaling from the base station. This adaptation is performed while seeking to also keep queueing delay under a small threshold.

Our preliminary evaluation using simulation shows that this outperforms existing TCP variants subject to a varying mobile capacity.

3.2.4.1 The Need for Explicit Capacity Information

The loss-based variants of TCP do not perform well over cellular networks [27, 28, 42, 43]. There are several key contributing factors. First, channel conditions in cellular networks fluctuate rapidly and TCP is inherently slow in adapting its sending rate to the available capacity. Second, mobile providers typically configure large buffers at base stations to deal with time-varying channel conditions. As a result, a (loss-based) TCP sender is allowed to increase its sending rate although it has already overshot the bottleneck capacity, creating long end-to-end self-inflicted delays. The victims of such delays are real-time and low-latency applications (e.g., WebRTC and conferencing applications).

To address these challenges, several cellular-specific transport protocols have been introduced [27, 42, 43]. However, these either require special hardware in the mobile provider's network (e.g., performance-enhancing proxy) and/or are sub-optimal due to imperfect estimation of available capacity [28]. The recently proposed CQIC [28] follows a cross-layer design principle. CQIC attempts to estimate the available bandwidth of a mobile user (UE) by exploiting the UE's physical channel quality information (CQI). Although this approach is promising, it has several drawbacks: (1) inaccurate capacity estimation due to a lack of awareness of traffic load at the base station; (2) being limited to flows within the scope of the operator network; (3) assuming that the UE has a single flow at a time. The latter does not hold with modern smartphones and tablets where multitasking is common.

3.2.4.2 MARC Congestion Control

MAMI research has designed and simulated a new TCP congestion control, which exploits explicit signaling from the radio access network (RAN). The RAN is aware of the physical, network and transport layer information of all UEs as well as how radio resources are distributed among them. This pursues the following key goals: (1) full utilisation of scarce radio resources; (2) low queueing delay; and (3) coexistence with deployed TCP variants.

The core idea is that MARC splits the air-interface capacity between competing long flows and considers short flows as noise. The reasoning is that if there are no long flows, short flows can complete their data delivery swiftly; if there are competing long flows, the aggregate of their sending rates is close to the available capacity, so short flows can still deliver their data quickly.

This requires two pieces of information from the base station: (1) the capacity available to a UE; and (2) the number of long flows competing at the UE's bottleneck link. This information – referred to as Throughput Guidance (TG) – is explicitly signalled to the MARC sender periodically (e.g., every 100ms) [21].

This signaling could also in future be exploited by adaptive bit rate applications (e.g., a media application can quickly adjust its video encoding to the advertised capacity).

MARC modifies the additive increase function and the Slow Start, Slow Start (SS), of TCP congestion control. In addition, it introduces a periodic congestion window reduction (PWR) mechanism. The algorithm is:

1. For each ACK, increase congestion window (CWND) (w): $w \leftarrow w + \frac{\alpha}{w}$
2. For each loss, decrease CWND (w): $w \leftarrow \frac{w}{2}$
3. For each TG, update target rate (r): $r \leftarrow (\frac{RTT_{min} \times Capacity}{\#LongFlows})$
4. Every window of data (roughly an RTT):
   (a) Estimate queueing delay (q): $q \leftarrow (\frac{\sum_{i=1}^{n} SRTT_i}{n}) - RTT_{min}$
   (b) Update aggressiveness of the increase function (α):
      i. if $(q \leq k)$ for θ RTTs, $\alpha \leftarrow 20$, else $\alpha \leftarrow 1$
   (c) Decrease CWND (w) only if $(w \geq r$ && $q > k)$: $w \leftarrow r$

The value α dictates the aggressiveness of CWND increase and is updated once per window of data. The target rate (r) is calculated upon arrival of feedback from the base station (r equals the bandwidth-delay product when the UE has only one flow). Note that r is less critical for short flows because they typically complete their data delivery within the SS phase. $SRTT_i$ is the i-th smoothed RTT sample, n is the total number of packets in a window and RTT_{min} is the minimum RTT. Constant k is the queueing delay budget. Constant θ ensures that MARC does not react to transient RTT variations (e.g., when the available capacity is under-utilised for 3 consecutive RTTs, MARC becomes aggressive).

Figure 4 shows a simple representation of how the algorithm adjusts after cwnd is reset to the target rate (r) based on the signaling information. During connection setup, the sender receives feedback from the remote endpoint relaying signaling received from the base station. It then sets the slow start threshold (ssthresh) to the estimated target rate (r). This prevents unnecessary packet loss due to buffer overflow during SS. From the start of data transmission, MARC begins estimating the queueing delay (q) every RTT. Once the flow enters the congestion avoidance phase (CWND $\geq$ ssthresh), and q exceeds the delay budget (k), MARC reduces its CWND to the target rate (r).

Figure 4 shows the congestion window, CWND, sawtooth resulting from PWR. The PWR mechanism regulates packets in the bottleneck queue, seeking to ensure that the packet queueing delay does not exceed the specified threshold.

Because the queue is periodically drained, RTT_{min} may update to the lowest round-trip propagation delay (when there is no queueing), resulting in an accurate estimation of the target rate. A correctly estimated target rate can improve fairness among contending flows and controls better the queue occupancy.

Finally, when a MARC flow is in the congestion avoidance phase, two scenarios arise: (1) CWND exceeds the target rate (r) in which case it behaves similarly to TCP ($\alpha = 1$) with PWR; and (2) CWND is smaller than target rate (r) in which case MARC may increase α to a large value (if it sees an opportunity). It is possible that a MARC flow starts losing packets even before reaching its target rate and also the estimate of queueing delay would prevent the sender aggressively increasing rate. This condition may happen when MARC competes with loss-based TCP, like NewReno. In such conditions, MARC behaves similar to NewReno to acquire a reasonable share of capacity from competing TCP flows.

MARC has been prototyped in NS3 with simulation parameters: θ = 3, k = 1ms, RTT = 60ms, queue size = 100 packets, the base station sends feedback every 100ms. Figures 3.5(a)

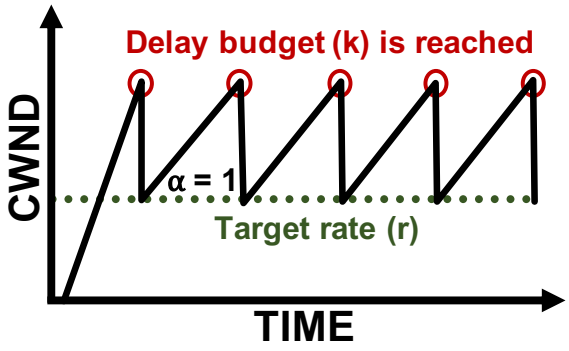


Figure 4: Simple showcase of the MARC algorithm

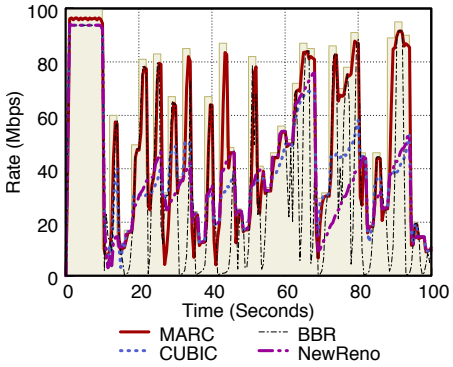
and 3.5(b) show the performance of MARC, CUBIC, BBR and NewReno when the UE's capacity varies every 2s, starting from time 10s (shaded region), and the UE has one flow. MARC outperforms other schemes while maintaining a low queue occupancy. MARC achieves 13%, 27%, and 18% higher overall throughput compared to CUBIC, BBR and NewReno. At 90th percentile, MARC's queue occupancy is 13 packets whereas CUBIC, BBR and NewReno is 82, 67, and 68 respectively.

Figure 3.5(c) shows simulation results that explore how quickly flows converge to their fair share of capacity. In this experiment, a new MARC flow joins the shared 100 Mbps bottleneck every 10s. The MARC flows quickly converge to their target rate while keeping their queuing delay within the delay budget, regardless of the number of competing flows.

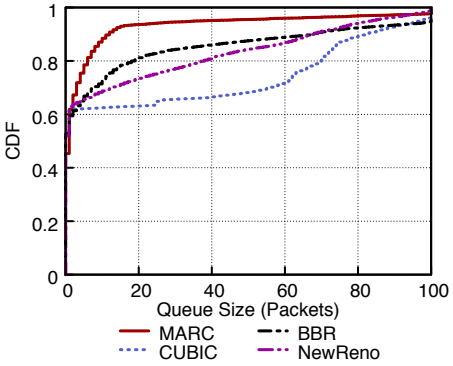
Figure 3.5(d) shows that MARC is not severely penalised when competing with buffer filling flows at a bottleneck link in the Internet. Two CUBIC flows and a MARC flow join the bottleneck link at 10s. Although CUBIC causes packet losses across all flows and forces the MARC flow to stay below its target rate, the MARC flow constantly tries to grab a fair capacity share. MARC behaves opportunistically when the CUBIC flows are throttled, by quickly increasing its sending rate as it detects available capacity.

3.2.4.3 Current Status

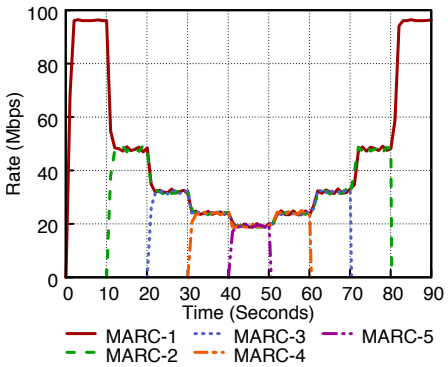
The core contribution that MARC brings to MAMI is that the simulation results demonstrate that explicit path to endpoint signaling can assist a transport protocol in finding an acceptable sending rate for an end to end path that includes mobile links. Further work will be required to understand how trust relationships needed and with actual implementation to evaluate the impact of actual mobile channels on the final performance. The opportunities for this research will continue after the conclusion of the MAMI project.



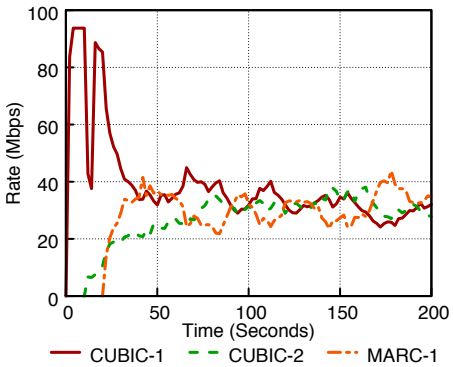
(a) Capacity variation



(b) Queue size distribution



(c) Convergence (intra-protocol)



(d) Coexistence (inter-protocol)

4 Conclusion

This deliverable reports the final status of the work completed in WP3.

The first part of this deliverable describes implementation efforts and experiments with PLUS for QUIC and UDP Options, both provided as a proof-of-concept to explore the challenges for developing a new transport stack, as well as progress on standardisation of a common abstract application-transport interface, based on Post Sockets as introduced in D3.2.

The second part of the deliverable presents the project's current efforts to apply middlebox co-operation mechanisms to existing and new transport protocols, as also already initially outlined in D3.2.

The technical work performed in WP3 must also be seen in the context of the dissemination activities of the project as described in D4.4. Especially, the research performed in WP3 directly impacts work in multiple standardization organizations as well as informed dissemination in industry through the MAMI Management and Measurement Summit (M3S), held in January 2018 in London, with three resulting white papers on "Challenges in Network Management with Encrypted Traffic" [26], "Analysis and Consideration on Management of Encrypted Traffic" [1], and "Security and Privacy Implications of Middlebox Cooperation Protocols" [19]. These three white papers summarize the finding of the project presented in form that is suitable as a basis for further discussion and work in industry and standardization. Current standardization efforts as described in this deliverables and explicitly listed in D4.4 will continue beyond end of the project in order to enables more and explicit middlebox cooperations.

References

[1] P. A. Aranda, D. Lpez, and T. Fossati. Analysis and consideration on management of encrypted traffic. cs.NI arxiv:1812.04834, 2018.

[2] D. L. Black. Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation. RFC 8311, Jan. 2018.

[3] R. Bless. A lower effort per-hop behavior (le phb). Internet-Draft draft-ietf-tsvwg-le-phb-05, IETF Secretariat, July 2018. http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-le-phb-05.txt.

[4] D. Borman, S. Deering, and R. Hinden. Ipv6 jumbograms. RFC 2675, RFC Editor, August 1999.

[5] B. Briscoe, M. Kuehlewind, and R. Scheffenegger. More accurate ecn feedback in tcp. Internet-Draft draft-ietf-tcpm-accurate-ecn-07, IETF Secretariat, July 2018. http://www.ietf.org/internet-drafts/draft-ietf-tcpm-accurate-ecn-07.txt.

[6] A. Brunstrom, T. Pauly, T. Enghardt, K.-J. Grinnemo, T. Jones, P. Tiesel, C. Perkins, and M. Welzl. Implementing interfaces to transport services. Internet-Draft draft-ietf-taps-impl-01, IETF Secretariat, July 2018. http://www.ietf.org/internet-drafts/draft-ietf-taps-impl-01.txt.

[7] B. Carpenter and S. Jiang. Transmission and processing of ipv6 extension headers. RFC 7045, RFC Editor, December 2013.

[8] Y. Cui, T. Li, C. Liu, X. Wang, and M. Kühlewind. Innovating transport with quic: Design approaches and research challenges. *IEEE Internet Computing*, 21(2):72–76, 2017.

[9] A. Custura, G. Fairhurst, and I. Learmonth. Exploring usable path mtu in the internet. In *Network Traffic Measurement and Analysis Conference (TMA 2018)*, 2018.

[10] A. Custura, R. Secchi, and G. Fairhurst. Exploring dscp modification pathologies in the internet. *Computer Communications*, 127:86–94, 9 2018.

[11] S. E. Deering and R. M. Hinden. Internet protocol, version 6 (ipv6) specification. RFC 2460, RFC Editor, December 1998. http://www.rfc-editor.org/rfc/rfc2460.txt.

[12] G. Fairhurst. Update to IANA Registration Procedures for Pool 3 Values in the Differentiated Services Field Codepoints (DSCP) Registry. RFC 8436, Aug. 2018.

[13] G. Fairhurst and T. Jones. Transport features of the user datagram protocol (udp) and lightweight udp (udp-lite). RFC 8304, RFC Editor, February 2018.

[14] G. Fairhurst, T. Jones, M. Txen, and I. Ruengeler. Packetization Layer Path MTU Discovery for Datagram Transports. Internet-Draft draft-ietf-tsvwg-datagram-plpmtud-05, Internet Engineering Task Force, Oct. 2018.

[15] G. Fairhurst, T. Jones, and R. Zullo. A Tale of Two Checksums, Nov. 2018. Work in Progress.

[16] F. L. Faucheur. Ip router alert considerations and usage. BCP 168, RFC Editor, October 2011.

[17] G. Fauirhurst, M. Khlewind, and D. R. Lopez. Measurement-based protocol design. In *European Conference on Networks and Communications (EuCNC'2017)*, 2017.

[18] T. Fossati. Content classification. Technical Report Document No IG.01, rev 1.0, GSM Association (GSMA), 2018.

[19] T. Fossati, R. Muentener, S. Neuhaus, and B. Trammell. Security and privacy implications of middlebox cooperation protocols. cs.NI arXiv:1812.05437, 2018. ETH TIK Technical Report 370.

[20] B. Hinden and G. Fairhurst. IPv6 Minimum Path MTU Hop-by-Hop Option. Internet-Draft draft-hinden-6man-mtu-option-00, Internet Engineering Task Force, Oct. 2018. Work in Progress.

[21] A. Jain, A. Terzis, H. Flinck, N. Sprecher, S. Arunachalam, K. Smith, V. Devarapalli, and R. B. Yanai. Mobile Throughput Guidance Inband Signaling Protocol. Internet-Draft draft-flinck-mobile-throughput-guidance-04, Internet Engineering Task Force, Mar. 2017. Work in Progress.

[22] C. Jennings and F. Audet. Network Address Translation (NAT) Behavioral Requirements for Unicast UDP. RFC 4787, Jan. 2007.

[23] S. Krishnan, M. Kuehlewind, B. Briscoe, and C. Ralli. IPv6 Destination Option for Congestion Exposure (ConEx). RFC 7837 (Experimental), May 2016.

[24] M. Kuehlwind, B. Trammell, G. Fairhurst, T. Jones, S. Neuhaus, R. Muentener, and T. Fossati. Middlebox cooperation protocol specification and analysis. Deliverable 3.2, Measurement and Architecture for a Middleboxed Internet (MAMI), March 2018.

[25] M. Kühlewind, T. Bühler, B. Trammell, R. Müntener, S. Neuhaus, and G. Fairhurst. A Path Layer for the Internet: Enabling Network Operations on Encrypted Protocols. In *Proceedings of the International Conference on Network and Service Management (CNSM)*. IEEE, 2017.

[26] M. Khlewind, B. Trammell, T. Bhler, G. Fairhurst, and V. Gurbani. Challenges in network management of encrypted traffic. cs.NI arXiv:1810.09272, 2018. ETH TIK Technical Report 369.

[27] W. K. Leong, Z. Wang, and B. Leong. Tcp congestion control beyond bandwidth-delay product for mobile cellular networks. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '17, pages 167–179, New York, NY, USA, 2017. ACM.

[28] F. Lu, H. Du, A. Jain, G. M. Voelker, A. C. Snoeren, and A. Terzis. Cqic: Revisiting cross-layer congestion control for cellular networks. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, HotMobile '15, pages 45–50, New York, NY, USA, 2015. ACM.

[29] M. Ma wski. Path MTU discovery in practice. *Cloudflare Blog*, May 2015. (https://blog.cloudflare.com/path-mtu-discovery-in-practice/).

[30] M. Mathis and J. Heffner. Packetization Layer Path MTU Discovery. RFC 4821, RFC Editor, March 2007. `http://www.rfc-editor.org/rfc/rfc4821.txt`.

[31] T. Pauly, B. Trammell, A. Brunstrom, G. Fairhurst, C. Perkins, P. Tiesel, and C. Wood. An architecture for transport services. Internet-Draft draft-ietf-taps-arch-01, IETF Secretariat, July 2018. `http://www.ietf.org/internet-drafts/draft-ietf-taps-arch-01.txt`.

[32] J. Postel. User datagram protocol. STD 6, RFC Editor, August 1980. `http://www.rfc-editor.org/rfc/rfc768.txt`.

[33] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, RFC Editor, September 2001. `http://www.rfc-editor.org/rfc/rfc3168.txt`.

[34] P. S. Schmidt, T. Enghardt, R. Khalili, and A. Feldmann. Socket Intents: Leveraging Application Awareness for Multi-Access Connectivity,. In *ACM CoNEXT*.

[35] R. Secchi, A. Venne, and A. Custura. Measurements concerning the DSCP for a LE PHB , Aug. 2017. IETF 99.

[36] J. M. <>, S. E. D. <>, J. M. <>, and B. Hinden. Path MTU Discovery for IP version 6. RFC 8201, July 2017.

[37] J. Touch. Transport options for udp. Internet-Draft draft-ietf-tsvwg-udp-options-05, IETF Secretariat, July 2018. `http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-udp-options-05.txt`.

[38] B. Trammell and M. Kuehlewind. The QUIC Latency Spin Bit. Internet-Draft draft-ietf-quic-spin-exp-00, IETF Secretariat, April 2018. `http://www.ietf.org/internet-drafts/draft-ietf-quic-spin-exp-00.txt`.

[39] B. Trammell, C. Perkins, T. Pauly, M. Khlewind, and C. A. Wood. Post Sockets, An Abstract Programming Interface for the Transport Layer. Internet-Draft draft-trammell-taps-post-sockets-03, Internet Engineering Task Force, Oct. 2017. Work in Progress.

[40] B. Trammell, M. Welzl, T. Enghardt, G. Fairhurst, M. Kuehlewind, C. Perkins, P. Tiesel, and C. Wood. An abstract application layer interface to transport services. Internet-Draft draft-ietf-taps-interface-01, IETF Secretariat, July 2018. `http://www.ietf.org/internet-drafts/draft-ietf-taps-interface-01.txt`.

[41] P. D. Vaere, T. Bhler, M. Khlewind, and B. Trammell. Three bits suffice: Explicit support for passive measurement of internet latency in quic and tcp. In *Internet Measurement Conference (IMC) 2018*, 2018.

[42] K. Winstein, A. Sivaraman, and H. Balakrishnan. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pages 459–472, Berkeley, CA, USA, 2013. USENIX Association.

[43] Y. Zaki, T. Pötsch, J. Chen, L. Subramanian, and C. Görg. Adaptive congestion control for unpredictable cellular networks. *SIGCOMM Comput. Commun. Rev.*, 45(4):509–522, Aug. 2015.