De-Ossifying the Internet Transport Layer: A Survey and Future Perspectives

Giorgos Papastergiou, Gorry Fairhurst, David Ros, Anna Brunstrom, Karl-Johan Grinnemo, *Senior Member, IEEE*, Per Hurtig, Naeem Khademi, Michael Tüxen, Michael Welzl, Dragana Damjanovic, and Simone Mangiante

***Abstract*—It is widely recognized that the Internet transport layer has become ossified, where further evolution has become hard or even impossible. This is a direct consequence of the ubiquitous deployment of middleboxes that hamper the deployment of new transports, aggravated further by the limited flexibility of the application programming interface (API) typically presented to applications. To tackle this problem, a wide range of solutions have been proposed in the literature, each aiming to address a particular aspect. Yet, no single proposal has emerged that is able to enable evolution of the transport layer. In this paper, after an overview of the main issues and reasons for transport-layer ossification, we survey proposed solutions and discuss their potential and limitations. The survey is divided into five parts, each covering a set of point solutions for a different facet of the problem space: 1) designing middlebox-proof transports; 2) signaling for facilitating middlebox traversal; 3) enhancing the API between the applications and the transport layer; 4) discovering and exploiting end-to-end capabilities; and 5) enabling user-space protocol stacks. Based on this analysis, we then identify further development needs toward an overall solution. We argue that the development of a comprehensive transport layer framework, able to facilitate the integration and cooperation of specialized solutions in an application-independent and flexible way, is a necessary step toward making the Internet transport architecture truly evolvable. To this end, we identify the requirements for such a framework and provide insights for its development.**

***Index Terms*—Transport protocols, protocol-stack ossification, API, middleboxes, user-space networking stacks.**

Manuscript received March 11, 2016; revised September 16, 2016; accepted October 25, 2016. Date of publication November 8, 2016; date of current version February 22, 2017. This work was supported by the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement no. 644334 (NEAT).

G. Papastergiou was with Simula Research Laboratory, 1364 Fornebu, Norway. He is currently with Digea S.A., 15125 Athens, Greece (e-mail: gpapastergiou@digea.gr).

D. Ros is with Simula Research Laboratory, 1364 Fornebu, Norway (e-mail: dros@simula.no).

G. Fairhurst is with the University of Aberdeen, Aberdeen, AB24 3UE, U.K. (e-mail: gorry@erg.abdn.ac.uk).

A. Brunstrom, K.-J. Grinnemo, and P. Hurtig are with Karlstad University, 651 88 Karlstad, Sweden (e-mail: anna.brunstrom@kau.se; karl-johan.grinnemo@kau.se; per.hurtig@kau.se).

N. Khademi and M. Welzl are with the University of Oslo, 0316 Oslo, Norway (e-mail: naeemk@ifi.uio.no; michawe@ifi.uio.no).

M. Tüxen is with the Münster University of Applied Sciences, 48565 Steinfurt, Germany (e-mail: tuexen@fh-muenster.de).

D. Damjanovic is with Mozilla Corporation, Mountain View, CA 94041, USA (e-mail: ddamjanovic@mozilla.com).

S. Mangiante is with EMC Corporation, P31 D253 Ovens, Ireland (e-mail: simone.mangiante@emc.com).

Digital Object Identifier 10.1109/COMST.2016.2626780

I. Introduction

NETWORKS can and do vary significantly in the set of functions they offer and their ability to move data between endpoints. The transport layer operates across the network and is responsible for efficient and robust end-to-end communication between network endpoints. The term end-to-end is often associated with a principle, called the end-to-end argument [1]. This suggests that "functions placed at low levels of a system may be redundant, or of little value, when compared with the cost of providing them at that low level." This argument followed Schroeder *et al.*'s [2] earlier work on system design and security, and is now generally considered as a simple guide for which services should be realized at the transport layer.

The transport layer was designed to hide the details and variability of the network service from the applications that need to use it. The Internet's transport layer also contains other functions that are difficult or impossible to provide within a network, such as reliability, verification of delivery, flow control to prevent the application from overwhelming the remote endpoint, congestion control to prevent the application from overwhelming the network, etc. People using the Internet mostly run applications that are based on the Transmission Control Protocol, TCP [3], which provides these transport functions.

Some applications need a different set of services to those offered by TCP. For example, a Web client may wish to be able to prioritize sub-flows carrying specific objects, a multimedia flow may prefer timeliness to reliable delivery, and IP telephony can be tolerant to packet loss or in some cases to bit errors. There are many cases where TCP simply does not meet the need of applications—yet it ends up being used because it "just works," but not necessarily very well [4]. Applications that do not want the transport semantics of TCP typically just use the User Datagram Protocol, UDP [5]. While UDP provides flexibility that allows any set of services to be defined, every function needed has to be implemented at the application layer.

Some initiatives have developed alternate protocols to TCP, suited for other application types, for instance: the Datagram Congestion Control Protocol (DCCP) [6] was proposed to support streaming multimedia; the Stream Control Transport Protcol (SCTP) [7], [8] originally targeted telephony signaling; UDP-Lite [9] supports error-tolerant audio and video services over wireless links. However, despite being standardized, with

available implementations for common platforms, these transports are seldom seen in the general Internet, and TCP and UDP remain the only widely used transports.

A. Transport-Layer Ossification: Overview of Issues

Why do developers and users not adopt more modern protocols? It is not because new transports do not meet a real need. The following paragraphs examine the main reasons for this *ossification* of the transport layer.

1) Middleboxes: Since the time [1] was first published, computer operating systems and Internet equipment have evolved. In today's commercial Internet environment, there is now no market for simplicity, and each new product and improvement adds complexity—necessary to differentiate markets and to cater for the wide variety of applications supported by modern systems. New stakeholders have emerged [10]: Internet service providers seeking differentiation; new government interests; changing motivations of a growing user base; tension between the demand for trustworthy overall operation and the inability to trust individual users or operators. Most importantly, most operators have chosen to introduce these functions using middleboxes [10].

To become usable, a new transport needs to be made available to applications, requiring upgrades of both the sending and receiving endpoints. However, for a new transport to be adopted, the need to upgrade end-hosts is not the only obstacle to overcome. *Ossification of the network infrastructure* is probably the most significant barrier [11]–[14]: a transport protocol must be able to traverse the network; a new protocol is only useful if it is able to traverse paths on a larger part of the Internet. The ubiquity of middleboxes of a variety of forms (from Network Address and Port Translators (NAPTs) to firewalls, accelerators, load-balancers, and a range of portals and more exotic devices) makes it very hard to change the status quo. Blumenthal and Clark [10] also warn of the implications of this approach: "certain kinds of innovation will be stifled if the open and transparent nature of the Internet erodes." Performing advanced network functions that go beyond the network layer, middleboxes not only need to understand the semantics of transport layer protocols, but some also tamper with protocol headers and thus violate end-to-end semantics [10], [15]. As a result, any new native transport (layered directly on IP) is doomed to fail to pass through middleboxes until specific explicit support is added for that transport, while new extensions to standard transports (i.e., to TCP and UDP) are also vulnerable to potential middlebox interference [16].

If a protocol (or application) is widely used, then it is likely that there exists a business case to support the protocol. However, the motivation to support a protocol that has not yet reached wide-scale use is much weaker or non-existent. This creates a "tussle," described in [17], or similarly the "vicious circle" described in [4]. Quite simply, a new protocol will not be deployed over the Internet—because to do so would first require a business case, predicated on a user base that already have deployed the protocol. This ossification has resulted in little-to-no use of new transports for the last decade.

Even when TCP or UDP is used, middleboxes still cause significant connectivity problems to applications. For instance, since most NAPTs are built around the traditional client/server application model, they usually break end-to-end connectivity for applications that need direct communication between two arbitrary hosts, such as peer-to-peer applications [18]. While Application Layer Gateways (ALGs) are often used to embed application-specific knowledge into middleboxes to facilitate protocol traversal for particular applications, this solution has significant limitations in terms of deployment and scalability: a separate ALG is needed for each application protocol used (e.g., SIP [19], [20], FTP [21], etc.) and hence all NAPTs would require to be updated every time a new application—i.e., a new application protocol—needs to be supported. There are various forms of middlebox that perform a transport proxy function, for instance to enable multipath transports (e.g., [22] and [23]). Although able to mitigate the deployment tussle between new protocols and applications/services, proxy solutions have limited scalability and break end-to-end connectivity [24]. Security-related manipulation of TCP and UDP traffic performed by corporate firewalls and NAPTs can also cause significant connectivity problems in enterprise environments. Finally, a class of middleboxes expects only a certain application protocol like HTTP; in the face of such devices, the only solution is to tunnel connections over the supported protocol.

2) Application Programming Interfaces: A flexible and extensible API between the applications and the transport layer is essential for applications to be able to harness the benefits of new transport services [25]. Today, the socket API essentially serves as the omnipresent application networking interface. However, it has become more and more apparent that this API is contributing to the Internet transport-layer ossification problem [26]. Its simplicity may have led to its ubiquity, but has also held back the development of more enhanced APIs. This is evident in the currently ongoing standardization of the SCTP socket API—the SCTP transport protocol incorporates support for multihoming, but it is impossible to export this support through the standard socket API.

The very success of TCP and UDP has therefore led to *ossification of the API presented to applications*. These two have now become the only widely available transports. This is reflected in the implementation of the socket API, which ties applications to a priori choices of transport protocol (either TCP or UDP). An application designed to work with one of these transports will need to be changed to support any new transport protocol.

The Internet has been designed so that transports only rely on core network functions, the so-called *Best-Effort* service. This has enabled transports to work across a diverse range of networks without having to know exactly how these provide the network service. However, this does not mean that information about what the transport/application needs from the network would not be helpful to improve the efficiency of the network or to enable the application to receive the most suitable service, but the current socket API does not facilitate this.

3) Other Issues: An evolvable transport layer architecture requires that endpoints are capable of *discovering* if a new transport can be used: An endpoint initiating a communication session must know whether a transport (and any required transport options) are supported both along the end-to-end network path, and by the intended remote endpoint(s).

Except for some one-sided transport-layer mechanisms (e.g., a TCP sender's choice between a range of congestion control algorithms [27]), the choice of a transport will require not only discovering the set of transports that are available at the remote endpoint, but also when more than one is supported at both ends, there needs to be an *agreement* from both endpoints on the choice of the particular transport.

Many network paths include middleboxes, some of which can, and often will, interfere with transport protocols. Endpoints need to assess whether a particular choice of transport can be safely used over the path.

Finally, one major additional challenge to deploying a new transport protocol is whether the transport protocol is supported across multiple OS platforms (e.g., Linux, FreeBSD, Mac OS X and Windows). Modifying OS kernel code can be costly in terms of deployment effort and often requires an OS update at the sender and/or the receiver to support the new transport, making any development effort platform-dependent.

B. Scope and Structure of the Paper

A range of point solutions have been proposed in the literature to address the above issues. Each covers a different aspect of the overall problem. In this paper, we review previous and ongoing efforts in the field. Our goal is to provide a better understanding of the pertained research issues, identify the potential and limitations of existing point solutions, and identify the need for further development.

We focus on evolutionary deployment. This restricts our survey to proposals that do *not* require redesigning the Internet architecture from scratch, hence, clean-slate approaches, such as Information-Centric Networking (ICN) [28], have been ruled as out of scope. ICN is a approach to evolve the Internet infrastructure away from host-centric end-to-end communication to receiver-driven content retrieval based on "named information" (or content or data). Among different ICN proposals, Named Data Networking (NDN) [29] is designed to integrate fundamental architectural primitives: security built into data itself; inherently self-regulating network traffic (flow balance); and adaptive routing and forwarding capabilities. The NDN architecture does not have a separate transport layer and transport functionality is moved into applications, their supporting libraries, and the strategy component in the forwarding plane.

In addition to clean-state approaches like NDN, less radical solutions are being considered such as Mobile-Edge Computing (MEC) [30], [31]. This moves the transport endpoint for IT and cloud-computing capabilities closer to subscribers, moving transport connections from the network core to the edge of a cellular network, which may reduce core congestion and latency. MEC may offer opportunities to simplify transport protocols, for instance by using cross-layer signaling from the RAN (e.g., knowledge of a short path RTT, and/or throughput guidance) to optimize transport stacks for throughput and/or delay. This is still the subject of future research as the MEC concept evolves.

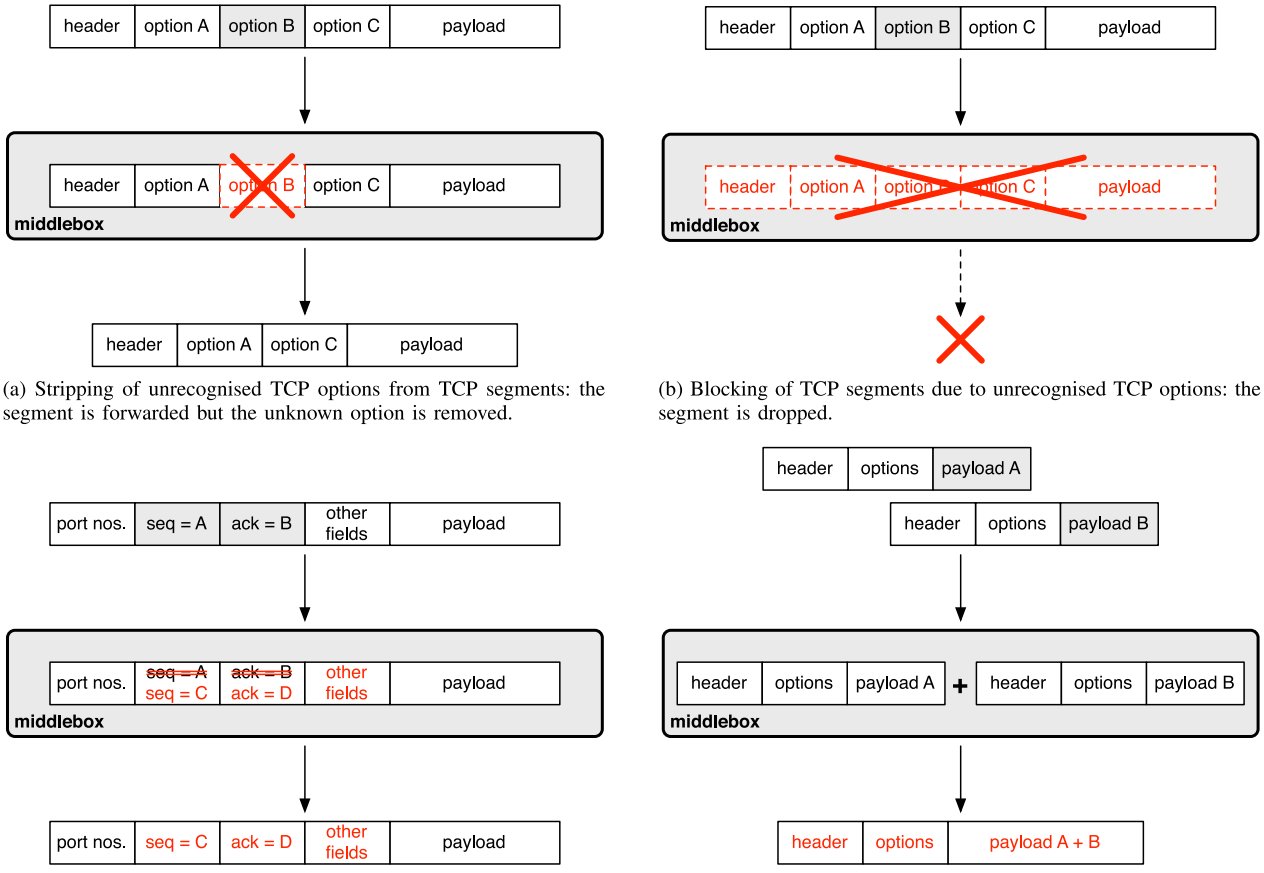
Another less radical solution is the concept of network overlays, which were seen as a promising approach to tackle ossification of the Internet architecture [32], [33]. Network overlays have employed network-layer encapsulation methods to introduce new functions not supported in the network [34]. Examples of such functions include support for IP multicast [35]–[37] and network virtualization within a data center environment [38] based on methods like NVGRE, VXLAN, GRE-in-UDP and GUE [39]. Li [40] surveys proposals to improve future scalability of the Internet, including the Locator/ID Separation Protocol (LISP) [41] which—besides helping with, e.g., route table scaling—can aid in overcoming ossification by, e.g., adding edge support for mobility and multicast [42] or enabling some form of multipath transport proxy [43], [44]. Network overlays have benefits (especially in transition to support new protocols), however, they hide the underlying network from the transport, impose homogeneity on a diverse network service and can be an obstacle to evolution of different network-transport interactions, which adds to ossification. The remainder of this survey therefore focuses on native transport protocols.

Communication middleware is also beyond the scope of this survey, because such middleware usually provides a different communication abstraction to applications, rather than offering transport services different to those of a common networking stack.

Based on our analysis, we argue that proposing solutions in isolation cannot result in an Internet transport layer architecture that is truly evolvable, and that a necessary step forward is the development of a comprehensive *transport layer framework* able to facilitate the integration and cooperation of new network and transport functions in an application-independent and flexible way. We therefore identify the requirements for such a framework and provide insights for its development.

The remainder of the paper is organized as follows: Sections II to VI provide an overview of previous and ongoing efforts to tackle ossification of the Internet transport layer, where each section covers a different aspect of the overall problem:

- Section II focuses on ways to design middlebox-proof transports, as a means to overcome the barriers imposed by middleboxes to using new transport protocols and protocol features.
- Section III is devoted to mechanisms that seek to better support end-to-end connectivity by facilitating traversal of middleboxes.
- Section IV outlines approaches that aim to enhance the API between applications and the transport layer.
- Section V examines approaches that allow endpoints to discover and agree on which protocols are supported along an end-to-end path.
- Lastly, Section VI explores techniques for enabling user-space protocol stacks.



(a) Stripping of unrecognised TCP options from TCP segments: the segment is forwarded but the unknown option is removed.

(b) Blocking of TCP segments due to unrecognised TCP options: the segment is dropped.

(c) Rewriting of TCP sequence and acknowledgement numbers. The TCP checksum has to be rewritten as well.

(d) Coalescing of small TCP segments into a single, larger segment: parts of the header are rewritten, and options in the coalesced segment may change.

Fig. 1. Examples of middlebox interactions with TCP.

Section VII summarizes the survey and taxonomy of point solutions to transport ossification presented in Sections II–VI. Next, Section VIII analyzes the requirements for an evolvable transport framework. Finally, Section IX concludes the paper by identifying future research directions that may assist work in this area.

II. Design of Middlebox-Proof Transports

There have been recent efforts to provide a richer set of transport services to applications than those provided by TCP and UDP within the constrained design space imposed by the ubiquitous deployment of middleboxes. These span two broad research directions: 1) extending TCP to provide a richer set of transport services, while guarding new extensions against potential middlebox interference, and 2) building new application-specific transports on top of UDP or TCP to ensure they transparently pass through existing middleboxes.

A. Extending TCP to Offer a Richer Set of Transport Services

TCP is an extensible protocol. It can negotiate protocol extensions during connection establishment and exchange additional control information throughout the lifetime of a connection. During the last decade, measurement studies have investigated how existing middleboxes interact, either intentionally or unintentionally, with TCP extensions, how prevalent these interactions are, and to what extent they affect TCP's extensibility [14], [16], [45]–[47]. Examples of middlebox behavior (some of which are illustrated in Fig. 1) include: blocking or stripping of unknown TCP options, modification of TCP header fields and options (such as the Initial Sequence Number (ISN) and the Maximum Segment Size (MSS) option), re-segmentation or coalescence of TCP segments, and behavior triggered by "non-stereotypical" TCP communication seen on the wire. These empirical studies provide a first demarcation of the solution space and the first guidelines for designing middlebox-proof TCP extensions [16].

Multipath TCP (MPTCP) [16], [48]–[50], Tcpcrypt [51], [52], and Gentle Aggression [53] are prominent examples of TCP extensions whose design was highly influenced by the need to account for known middlebox behavior. Techniques were adopted to guard extended operations against potential middlebox interference. For instance, a fallback strategy to plain TCP is incorporated in all approaches to handle cases where extended operations fail (e.g., when options are stripped from SYN or regular packets, or when payload modification is detected). This ability to fall back to plain TCP assures stability and is considered an

important design goal for achieving widespread deployment. Relative sequence numbers are considered when encoding sequencing information within the new options to cope with potential re-writing of sequence numbers. Other techniques include the use of an additional data-level sequence space in MPTCP that allows it to maintain consistent sequence numbering on the wire while ensuring in-order data delivery over multiple subflows. Tcpcrypt was intentionally designed to exclude fields from the authentication header that could be expected to be modified by the path.

Recent work has identified the need for TCP to infer in-path alterations of packet header fields as a way to enable deployment of new TCP functions. Craven *et al*. [47] proposed TCP HICCUPS, an enhancement that allows TCP to detect packet header manipulation at field-level granularity and take appropriate actions (such as disabling a non-compatible extension) based on the middlebox behavior observed on a path.

TCP has a limited maximum header size. This led the designers of Tcpcrypt to the exchange of encryption information within the TCP payload (i.e., the body of the INIT1 and INIT2 sub-options). This highlights a significant factor that constrains the design space of TCP extensions: The limited size constrains the number and the extent of TCP options that can be simultaneously used by a TCP connection.

Extending the TCP option space has become an active research area that faces similar middlebox-related issues. For instance, Ramaiah [54] presents several middlebox considerations for designs to increase the TCP options space and reviews approaches proposed up to 2012. More recent proposals include TCP Extended Data Offset (EDO) [55], [56], TCP SYN Extended Option Space (SYN-EOS) [57], and Inner Space [58]. TCP EDO extends the option space in all packets except the initial SYN packets (i.e., SYN and SYN/ACK) using a TCP option to override the TCP data offset field, while TCP SYN-EOS complements TCP EDO by extending the option space in SYN packets using an additional out-of-band packet during connection establishment. Inner Space uses a different strategy to extend the option space in every segment, where options are tunneled within the segment payload and a dual handshake procedure is used for assuring backwards compatibility with legacy servers. These approaches are currently under development and further work is needed to evaluate their deployability.

Experience in the design of MPTCP inspired another possible dimension to the design space: TCP "camouflaging" [59]. This suggests a new transport protocol could operate alongside TCP when the new protocol is disguised to look like TCP on the wire as in Polyversal TCP (PVTCP) [59]. Built upon the MPTCP subflow mechanism, PVTCP allows applications to explicitly customize the transport semantics of each subflow according to their requirements and assures a fallback to plain TCP or MPTCP. It remains to be seen whether the complexity of Polyversal TCP, or similar approaches, will offer a feasible path to deployment.

Although recent advances indicate that TCP continues to be extensible, more detailed and large-scale studies are needed to provide a deeper insight into the prevalence and range of middlebox behaviors. The IAB Workshop on Stack Evolution in a Middlebox Internet (SEMI) [60] identified this need and resulted in the "Measurement and Analysis for Protocols" (MAP) IRTF research group[1] that aims to be a forum for exchange and discussion of insights from such measurements [61].

B. Using Widely Deployed Transports as Substrates

The broad deployment and support of TCP and UDP in the Internet have led to the proliferation of a new design/deployment model where transport layer innovation occurs on top of these protocols. Typically, such transports are integrated into applications and aim to fulfill specific application requirements.

The choice between TCP and UDP involves a trade-off between design and implementation effort, flexibility and performance. On the one hand, UDP provides a "least-common-denominator" substrate with greater flexibility to control how data are sent over the network. However, building new transports on top of UDP often involves reinventing the wheel for services already offered by TCP (e.g., feature negotiation, congestion control, and reliability) and requires maintaining connection state in middleboxes by sending keep-alive messages that waste capacity and energy [62]. Guidelines for using UDP robustly are given in [63]. On the other hand, TCP is a feature-rich transport protocol that has undergone remarkable evolution over the past decades and can hence offer significant performance advantages over UDP [64]. However, TCP does not preserve message boundaries and is unable to support the use of only a subset of the services it provides; providing services that may not be needed can result in significant performance penalties. For example, the TCP in-order delivery service can incur increased end-to-end delays in lossy networks due to head-of-line blocking at the receiver.

The Minion suite of protocols has been proposed to address the above shortcomings of using TCP as a substrate protocol [64], [65]. This was designed to offer an unordered, message-oriented delivery alternative to UDP. Minion is wire-compatible with TCP (or TLS/TCP when secure services are needed), at the expense of using slightly increased capacity. Other facilities offered by Minion to the application include message multiplexing and priority-based data transmission. Despite its attractive features, the Minion suite has not seen wide-scale use. One reason could be that one of its greatest benefits, the ability to relax the in-order semantics of TCP, requires changes to the TCP stack, and hence is OS-dependent.

UDP can be used as a lightweight substrate and has been used since the 1990s to carry multimedia traffic with the Real-Time Transport Protocol (RTP) [66]. Characteristic examples using UDP are:

- Google's Quick UDP Internet Connections (QUIC) protocol [67], [68], a UDP-based low-latency alternative to TCP/TLS for SPDY [69] and HTTP/2 [70].
- Adobe's Real Time Media Flow Protocol (RTMFP) [71], a protocol for efficient peer-to-peer multimedia streaming.
- The Multipath Real-Time Transport Protocol (MPRTP) [72], a protocol for multipath media streaming.

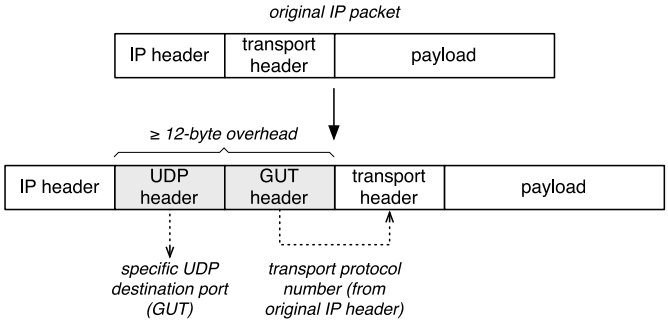
[1]Formerly known as "How Ossified is the Protocol Stack?" (HOPS).

- The widely used DTLS [73] protocol that provides stream- and datagram-oriented security services over UDP.
- The uTorrent Transport Protocol (uTP) [74], a UDP-based protocol for BitTorrent designed to offer a less-than-best-effort service for peer-to-peer file sharing applications.
- The UDP-based Data Transfer (UDT) protocol [75], [76] designed for efficient transferring of large data volumes over high-speed networks.
- The Structured Stream Transport (SST) protocol [77], a generic approach designed to offer services similar to SCTP [8], such as multistreaming and stream prioritization, over UDP.

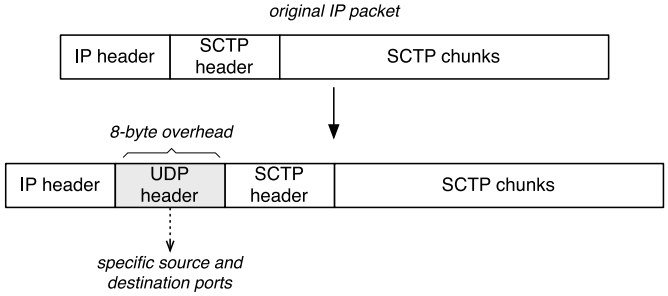
In addition to the above approaches, methods have been standardized by the IETF that encapsulate native protocols such as SCTP [8] and DCCP [6] within UDP [78], [79]. Methods have been proposed for encapsulating TCP over UDP enabling it to traverse network paths where only UDP is supported [80]. There is a large variety of (incompatible) tunnel and encapsulation frameworks that allow protocols to operate over UDP. Generic solutions have been suggested for encapsulating native IP protocols within UDP: Generic UDP Tunneling (GUT) [81] is a simple UDP encapsulation that aims to transparently tunnel native transports over a single well-known UDP port. GUT modifies native IP packets by including an appropriate UDP/GUT header, reconstructing the packets at the receiver. Generic UDP Encapsulation (GUE) [82] is similar to GUT, but focuses on leveraging the capabilities of network devices for handling UDP flows (e.g., load balancing). GUE uses a UDP source port as an inner flow identifier and permits encapsulation of layer-2 and layer-3 protocols. Although generic approaches could allow for more consistent deployment, protocol-specific designs may still be needed to ensure the functionality of the encapsulated protocol is not affected. Fig. 2 illustrates some of the UDP-based encapsulation methods just described.

Besides enabling middlebox traversal, UDP encapsulation offers an additional benefit: it allows user-space implementations of native protocols to be a part of applications without requiring special privileges to access the IP layer. The SCTP user-space implementation in [83] also offers this option. However, UDP encapsulation increases protocol overhead due to the additional UDP headers and also affects interoperability as the encapsulated protocol cannot in principle interoperate with the native one. Other potential drawbacks include: additional processing overhead, possibly redundant functionality (e.g., checksums) and increased design complexity due to an additional point of multiplexing.

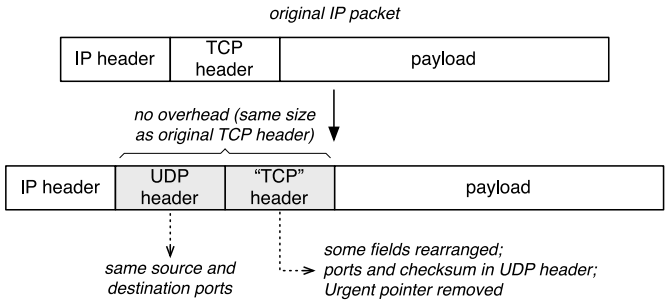
McQuistin *et al.* [84] approach the problem from a slightly different perspective and suggest, at a conceptual level, to reinterpret the semantics of TCP and UDP to support novel services. They propose reinterpretation of UDP headers as transport identification headers where port numbers become dynamic identifiers of the transport protocol carried in the payload, as well as the relaxation of TCP semantics (based on McQuistin *et al.*'s [85] earlier work on TCP Hollywood). Earlier work that "relaxes" TCP includes Time-lined TCP (TLTCP) [86] and Receiver-Centered TCP (TCP-RC) [87].



(a) Generic UDP Tunneling (GUT) [81], allowing encapsulation of an arbitrary native transport protocol.



(b) SCTP-over-UDP encapsulation [78].



(c) TCP-over-UDP encapsulation [80].

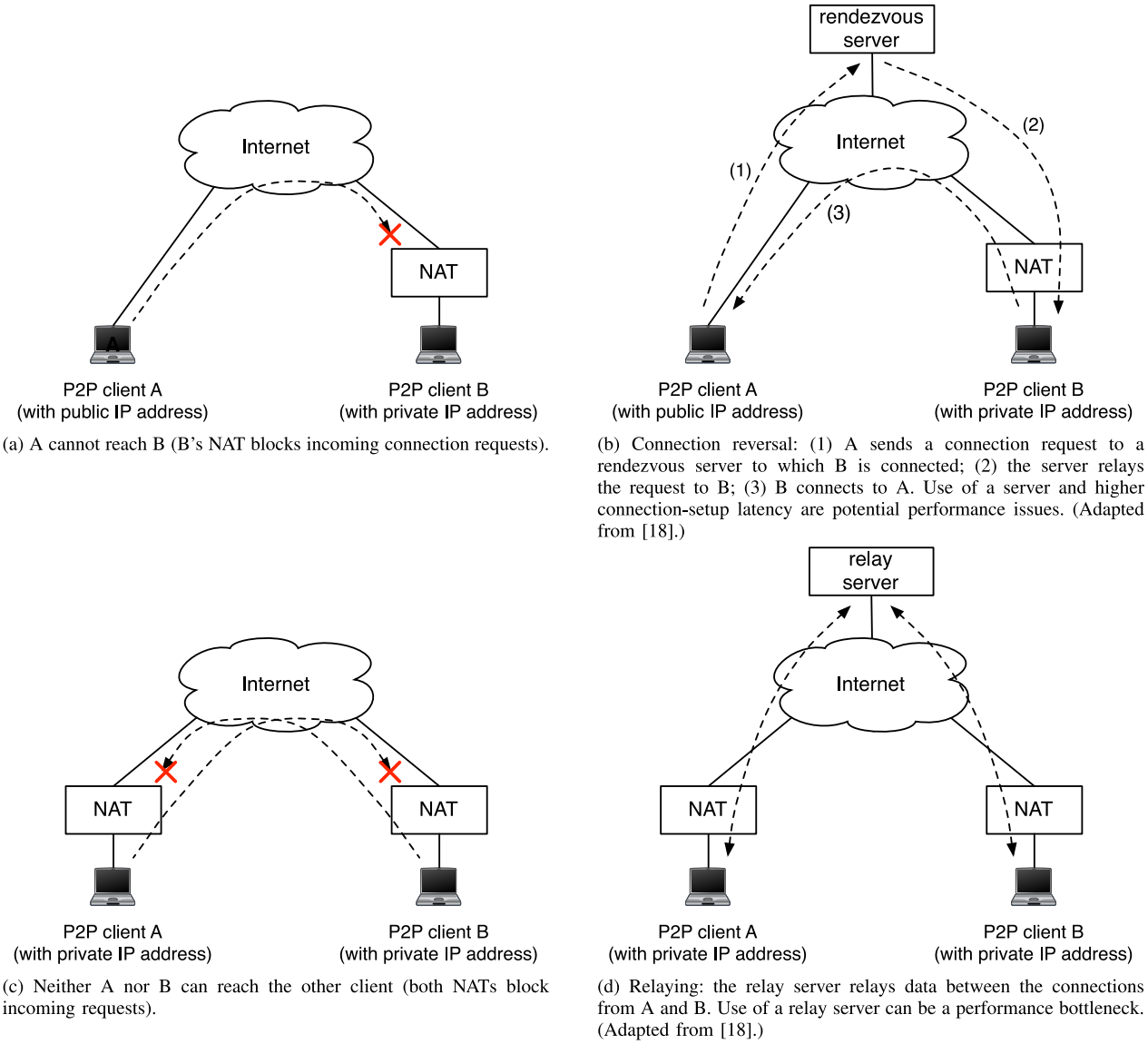
Fig. 2. Examples of transport encapsulation methods based on UDP.

The Minion suite [64] discussed above could contribute to this development.

Finally, there is ongoing work [88] to identify the suitability of the DTLS protocol [73] as a sub-transport for providing standardized security to higher-layer transports, along with services similar to that of PLUS (Section III-B), for instance signals to a middlebox to indicate the beginning or end of a flow. Huitema *et al*. [88] identified requirements that need to be fulfilled, including zero-latency setup and low overhead.

III. Signaling for Facilitating Middlebox Traversal

Even when TCP or UDP is used, middleboxes can cause significant connectivity problems to applications. For example, a NAPT can break the end-to-end connectivity for peer-to-peer applications (see Fig. 3) and applications that use control protocols such as SIP [19], [20], RTSP [89], or FTP [21] preventing them from communicating reachability information.



(a) A cannot reach B (B's NAT blocks incoming connection requests).

(b) Connection reversal: (1) A sends a connection request to a rendezvous server to which B is connected; (2) the server relays the request to B; (3) B connects to A. Use of a server and higher connection-setup latency are potential performance issues. (Adapted from [18].)

(c) Neither A nor B can reach the other client (both NATs block incoming requests).

(d) Relaying: the relay server relays data between the connections from A and B. Use of a relay server can be a performance bottleneck. (Adapted from [18].)

Fig. 3. Examples of connectivity issues due to NATs, and of implicit control techniques to address them.

A variety of support protocols and mechanisms have been proposed to improve connectivity across paths with middleboxes. These focus on ways to control middlebox behavior, methods to allow cooperation between endpoints and middleboxes, and methods to facilitate end-to-end connectivity. Such methods may be categorized as either *implicit* or *explicit*.

Implicit control solutions treat middleboxes as black boxes and trigger specific middlebox behaviors using data traffic sent to a well-known third party server. An explicit control solution allows an endpoint to explicitly interact with a middlebox to control or influence its behavior, e.g., to create NAPT mappings or to configure the lifetime for flow state.

A. *Implicit Middlebox Control*

Interactive Connection Establishment (ICE) [90], [91] seeks to increase the probability of successful connection by trying a set of implicit control techniques and selecting the one that works best. ICE was developed for middlebox traversal of UDP-based multimedia streams established by an offer/answer protocol (e.g., SIP) and is the middlebox traversal solution used in WebRTC [92]. This utilizes the Session Traversal Utilities for NAT (STUN) [93], a STUN signaling relay as a rendezvous point [19], and the Traversal Using Relays around NAT (TURN) protocol [94], a media relay. Ford *et al.* [18] describe a method for UDP hole punching. A TCP-based extension of ICE [95] adds TCP hole punching and considers UDP encapsulation as an alternative traversal solution for TCP. Techniques for TCP hole punching are presented in [18] and [96]. The IETF has defined a TURN relay for TCP [97] and DTLS [98].

No single solution is perfect in terms of applicability and performance. For instance, UDP hole punching cannot work with symmetric NATs, TURN uses a media relay server and hence can be a performance bottleneck, and TCP hole punching techniques have lower success probability because they depend on specific middlebox behaviors that are not always supported [99].

B. Explicit Middlebox Control and Cooperation

There is a range of approaches that can allow the transport to exchange control information with a middlebox, such as the Universal Plug and Play Internet Gateway Device (UPnP IGD) protocol [100], the Port Control Protocol (PCP) [101] and its precursor NAT Port Mapping Protocol (NAT-PMP) [102], the Middlebox Communication (MIDCOM) framework [103], and the NAT/Firewall NSIS Signaling Layer Protocol (NSLP) of the NSIS protocol suite [104]. Each solution has its own merits depending on network topology and security requirements, and hence there is no single solution that an application can rely upon to be universally available. For this reason, applications usually resort to use implicit control schemes that do not require additional support by middleboxes. However, no solution can always guarantee traversal.

A new form of UDP encapsulation layer could allow explicit cooperation with middleboxes [60], [105], [106]. This approach may help re-instantiate the layer boundary between a hop-by-hop network layer and an end-to-end transport layer [106], by allowing endpoints to control the information exposed to the path (encrypting everything above the UDP header), while still allowing appropriate transport semantics to be explicitly exposed to the path to assist the middlebox in establishing and maintaining state. An approach in which the transport protocol encrypts its protocol information can allow the transport to evolve without needing to consider the interference of middleboxes [69].

The Path Layer UDP Substrate (PLUS) protocol (previously called the Substrate Protocol for User Datagrams (SPUD) prototype [107]) is ongoing work that seeks to realize and facilitate middlebox traversal for new transports. PLUS groups the packets of a transport connection into a "tube" that can allow network devices on the path to understand basic session semantics (e.g., beginning and end of a flow). PLUS may also enable communication of path information to the sender, and permits explicit endpoint to/from middlebox communication.

PLUS requires support at both endpoints, and only gains benefits from middleboxes when they also implement support for the protocol. While use of encryption presents opportunities to readdress the incentives for stakeholders to declare the metadata that they use, this can not be considered a "quick-fix" solution. It has therefore been designed so that the PLUS protocol is useful as a simple encapsulation until support is enabled in middleboxes, enabling incremental deployment [107].

IV. Enhancing the API Between the Applications and the Transport Layer

The first part of this section gives an overview of the standard socket API and how it has been extended to support SCTP. The remaining parts consider ways to address some of the major inherent limitations of this API, i.e., those limitations that are believed to contribute to the ossification of the transport layer. We examine some proposed extensions to the standard socket API, and ways to address its current tight coupling between the offered transport service and the underlying transport protocol offering this service.

TABLE I
Basic TCP Socket API Functions

Function	Description
`socket`	Creates a new communication endpoint
`bind`	Binds communication endpoint to local IP address and port number
`listen`	Makes a socket listen to incoming connections
`accept`	Blocks a socket until a connection request arrives
`connect`	Makes a connection request
`send`	Sends a message over a connection
`recv`	Receives a message over a connection
`close`	Releases the connection

A. The Socket API

The socket API [108] is one of the most pervading and longest-lasting interfaces in distributed computing. After almost three decades of existence, however, novel technologies, like for instance multipath transport, are challenging the socket API's continued success [109].

Conceptually, a socket is an abstraction of a communication endpoint through which an application may send and receive data in much the same way as an open file permits an application to read and write data to a stable storage device such as a hard disk. Applications use socket descriptors to access sockets in the same way that they use file descriptors to access files.

The API was designed from the start to be independent from the underlying protocol stack, as seen in the way that a socket is created: `int socket(int` *domain*, `int` *type*, `int` *protocol*`)`. The *domain* parameter determines the communication domain or protocol family of a socket. Examples of protocol families include: `AF_INET` for the IPv4 Internet domain; `AF_INET6` for the IPv6 Internet domain; and, `AF_UNIX` for the local or Unix domain. The *type* parameter determines the type of a socket, or, more specifically, the semantics for the transport service—e.g., whether the transport service should be stream-oriented, reliable, and connection-oriented (`SOCK_STREAM`), or message-oriented, unreliable, and connectionless (`SOCK_DGRAM`). Finally, the `protocol` parameter lets an application specify which transport protocol to use to provide the transport service specified by the `type` parameter.

Although the socket API comprises a fairly large number of functions, there are less than a dozen core ones. For example, a simple connection-oriented client-server application does not need more than the eight functions listed in Table I. A server application generally executes the first four functions in the order given in the table, while a client application attempts to connect to the server after having created a socket; the `send` and `recv` functions may be called by both the client and the server. A connection that is no longer needed is closed by the client or server.

The API lets an application control the behavior of a socket through options. The set of options has expanded over time, as usage has evolved. There are essentially three ways to manipulate socket options:

1) The functions `setsockopt` and `getsockopt` provide access to the majority of available socket options.

2) The function `fcntl` is primarily used with non-blocking and asynchronous I/O.
3) The function `ioctl` has traditionally been the way to access implementation-dependent socket attributes.

The socket options accessed via `setsockopt` and `getsockopt` are divided into two levels: The first level are generic (i.e., non-protocol specific) options. For example, the sizes of the socket send (`SO_SNDBUF`) and receive (`SO_RCVBUF`) buffers are generic socket options. The second level comprises protocol-specific options such as those that control the behavior of IP, UDP, and TCP. An example of a well-known, second-level socket option is `TCP_NODELAY`, which determines whether the Nagle algorithm [110] should be enabled.

The deployment of the SCTP transport protocol [7], [8] demanded changes to the socket API. In addition to the services offered by TCP, SCTP supports both multi-homing (i.e., connections comprising several network paths) and multi-streaming (i.e., several independent logical flows over a single connection). These additions required extended versions of several existing socket API functions and a new notification mechanism to enable signaling of transport-level events to an application, such as connection status changes [111]. A good example of how SCTP extended the socket API, is the extended version of `bind`: The normal `bind` socket call only enables for a communication endpoint to bind to a single IP address. SCTP introduces the `sctp_bindx` socket call which lets an application bind to several or all IP addresses on a host.

Since SCTP has its roots in the transport of critical telephony signaling traffic, it had to be able to communicate transport-level events to an application, such as connection availability and remote operational errors. To ensure the SCTP event notification is well aligned with the rest of the socket API, events are enabled by a socket option: `SCTP_EVENTS`. Once enabled, the SCTP stack sends events as normal messages to the application. An application may distinguish between event notifications and normal messages, by a flag in event notification messages set to `MSG_NOTIFICATION`.

SCTP also extended the semantics of the socket API by supporting two types of sockets: one-to-one and one-to-many. A one-to-one socket resembles usage by TCP. A one-to-many socket makes it possible for an application to manage several SCTP connections via a single socket. This has advantages for server applications that may use a one-to-many socket to avoid the need to administer each client request through a separate socket.

The example of SCTP has shown that incorporating a transport with different techniques has required updates to the current socket API. It would seem reasonable to expect similar changes may also be needed to support any additional new transport (or technique) [109]. A significant drawback is that this also requires any application that wishes to benefit from using a new technique to be updated to use the new API.

B. More Expressive APIs/Extensions to the Socket API

Extensions to the socket API have also been proposed that change the way an application interacts with the transport layer. These may be categorized according to the abstraction level at which the underlying transport services are exposed (Fig. 4). Some proposed extensions, which we call *basic extensions*, only aim to remove perceived limitations and drawbacks of the standard socket API. For example, Msocket [112] makes it possible to have several implementations for each domain, type, and protocol assignment. These proposals provide the same exposure of the transport layer as the standard socket API.

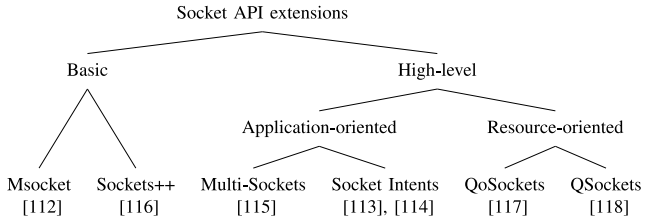


Fig. 4. Extensions to the socket API.

In contrast, *high-level extensions* hide the implementation of offered transport services from applications. These focus on ways to allow an application to express its quality-of-service (QoS) requirements to the transport layer. Examples include Socket Intents [113], [114] and Multi-Sockets [115]. High-level extensions can be further divided into *application-oriented* and *resource-oriented* extensions. Application-oriented extensions let an application express its QoS requirements in terms of application-dependent performance metrics or the characteristics of the traffic it will generate. In contrast, resource-oriented extensions focus on system-wide, network-oriented performance metrics such as packet loss, re-ordering, bitrate, or end-to-end delay. We now present each category of socket API extensions and provide examples within these categories.

1) Basic Extensions: If several protocol stacks are available, the standard socket API does not enable an application to explicitly select the one to use. The Msocket [112] extension removes this limitation by adding a stack parameter to the `socket` call. In Unix systems, the stack parameter is a device file. This does not have to be the case in other systems, and could refer to a kernel module. Backward compatibility with the standard socket API is assured by the definition of so-called default stacks: each protocol family is assigned a default stack.

Sockets++ [116] is an object-oriented basic extension that addresses a range of shortcomings with the socket API. It supports multipoint connections to enable several applications to participate in the same connection. It also supports direct forwarding allowing multimedia applications to request data to be directly forwarded from one stream to another. It seeks to minimize parameters in socket calls, e.g., combining domain and protocol parameters in the `socket` call, and to simplify socket API options. Importantly, this extension also enables applications to express their quality-of-service requirements.

2) High-Level Extensions: Intentional extensions originated in work for mobile devices with more than one available network interface. They allow applications to inform the API

about the traffic they intend to send (e.g., whether it will be latency-sensitive video conferencing traffic or throughput-dependent file transfers). This information enables the transport layer to select the most appropriate network interface, dividing the responsibility for communication between the application and the transport stack.

Intentional networking was first realized in Multi-Sockets [115], allowing an application to use labels to communicate its intents. Labels provided qualitative rather than quantitative information, e.g., to inform the API whether a message unit belongs to an interactive or non-interactive traffic flow, or whether it belongs to a flow that consumes little or much capacity. Conceptually, a multi-socket multiplexes several different labels across a single virtual connection, however, in practice, the proposal instantiated and used actual TCP connections over one or several physical network interfaces.

Socket Intents [113], [114] is a successor to Multi-Sockets, seeking to support multi-homed applications. Socket Intents replaced the labels used in Multi-Sockets by augmenting the socket API with additional socket options. An implementation of Socket Intents comprises three components: a wrapper library over the standard socket API, a policy module, and the multi-access manager—a daemon that hosts the policy module. Since creating a single policy that maps different traffic flows to different network interfaces is, in general, not feasible, the Socket Intents API was built as a generic framework with a replaceable policy module.

Resource-oriented socket API extensions offer communication between themselves and the application. For example, QoSockets [117] enables an application to negotiate its quality-of-service requirements with the transport layer, and for the transport layer to signal violations of these requirements back to the application. The requirements include loss rate, ordered or unordered delivery, end-to-end delay, and jitter. Application- and network-management functions were integrated by adding an interface to a Management Information Base (MIB), and a status interface for connections. These MIBs show how communication resources are allocated and utilized, and enable an application to detect and adapt to quality-of-service violations.

QSockets [118] is another resource-oriented socket API extension. Similar to QoSockets, QSockets also offers bidirectional communication to the application, enabling applications to obtain detailed quality-of-service feedback. It uses an extended socket API that adds a structure that contains the QoS preferences. The QoS parameters may also be set on a per-packet level by passing a structure to `sendto` calls, allowing per-packet deadlines and the setting of other flags. The API communicates with an in-kernel management module to control an in-kernel scheduler. This exposes functionality to the management module for managing scheduled packet streams. A pluggable scheduling layer allows various QoS scheduling algorithms.

Although no single approach has been adopted by the community, this body of research has shown there are benefits to enriching the transport API to express more than the traditional socket API.

C. Transparent Transport Protocol Selection

The current design of the socket API has a design that focuses on specific support for each transport protocol, each with different needs. Fairhurst *et al*. [119] provide a recent survey of the services provided by the range of IETF-standardized transports. The present design of the API makes it difficult to introduce any new protocol [120].

These limitations could be overcome by re-designing the way that the API is used, e.g., by using a protocol-independent mechanism to set parameters; by describing application requirements at a higher level of abstraction (similar to intentional methods); and by providing a service-oriented interface between applications and the transport (where applications describe the required services rather than the protocols to use). The latter would allow transport protocol selection to be dynamically handled at run-time, easing the introduction of new and alternate protocols.

A prototype implementation [121] used a service-oriented API to indicate a combination of inherent properties (reliability, security, etc.) and qualitative properties (expressing tendencies and preferences). The set of inherent transport properties was derived by examining several transport protocols (TCP/IP, UDP/IP, RDP [122], RDP/IP, XTP [123], XTP/IP, SCTP/IP). Five qualitative properties were also suggested (transmission delay, flow setup delay, network resource usage, host resource usage, and quality). A broker then matched the inherent properties with application requirements to first identify the transport to use, and then used the qualitative properties to optimize the matching.

Welzl [26] identified deployment problems resulting from the complexity of the different protocol APIs and proposed an "Adaptation Layer" that hides protocol details and exposes a common service-oriented interface. This allowed applications to specify their requirements and characteristics. An adaptation layer then sought to provide the best transport service based on available transport protocols and the current network environment. This adaptation layer could also tune protocol parameters and provide additional functions, such as buffering.

Welzl *et al*. [124] later derived a methodology for constructing a service-oriented transport API. This started with a list of all services provided by SCTP, DCCP and UDP-Lite, and iteratively pruned redundant services or services considered unnecessary, resulting in a list of 23 distinct transport services composed from six different features. This led to a strawman proposal for a protocol-independent version of the socket API, where the selected transport services could be accessed through their service number.

A similar proposal [125] expressed the desired service through a set of requirements, such as packet boundary preservation, authentication or maximum delays. Their adapted socket API used a name similar to a URI [126] to identify the communication peer, removing dependence on IP addresses.

There is a need to standardize any new service-oriented API [26], to ensure that it can have significant impact and becomes used by applications in future. This requires the community of application developers, and transport developers to reach consensus on the set of desirable interface features. Recent IETF work within the Transport Services working

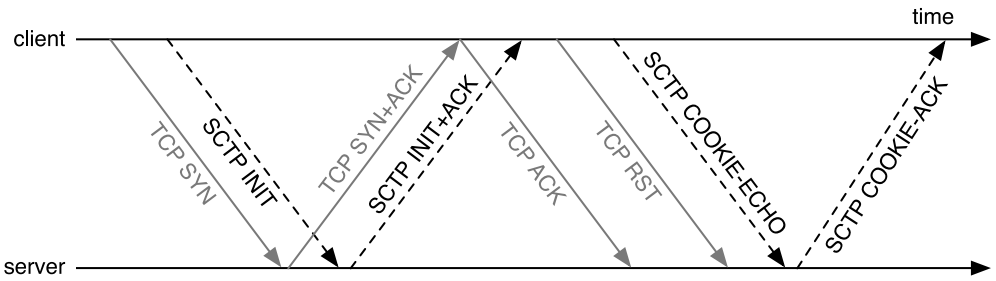


Fig. 5. "Happy eyeballs" technique for the discovery of SCTP support, with SCTP being the preferred choice. The first handshake of the SCTP association succeeds shortly after the TCP connection does, so the latter is aborted.

group (TAPS) [119], [127] provides a unique opportunity to develop this sort of consensus.

D. Enhancing the API to Allow Evolution Below the Transport Layer

There is a long history of proposals to support communication between end systems and the network. Proposed solutions can be divided into two broad classes according to their scope: 1) solutions that facilitate middlebox traversal for applications (discussed in Section III-B), 2) solutions that focus on communicating information between the network and the endpoints to improve application experience (signaling of QoS requirements, QoS reservations, and indications of capacity changes, of data corruption, of congestion, etc.). However, there are also challenges to finding suitable, scalable, secure and robust signaling mechanisms that can be deployed across the Internet (e.g., [107] and [128]–[131]). Finding appropriate methods largely remain an area of research. One issue with deploying these mechanisms is that many require applications to indicate their needs and how they expect the network to respond. The current socket API does not provide such information, nor have applications typically been designed to utilize such methods, and hence at present these are unlikely to be widely deployed.

A higher-level transport API that places the responsibility for negotiating and using network signaling below the transport API may encourage future applications to utilize new methods as the stack and network introduce them. This technique was adopted by some of the API proposals discussed [118] and could be enabled by the approaches being proposed in [132].

V. Discovery and Exploitation of End-to-End Capabilities

Some application-layer proposals provide limited support for negotiation of e.g., transport security for unicast, connection-oriented application sessions [133], [134], or transport protocol, port and IP address for multimedia sessions [135], [136]. A more generic approach is for end-points to use a negotiation protocol to exchange protocol-stack information, and to agree on a transport stack (i.e., transport and security protocols to be used, and their options), as described in [137]. This proposal focused on connection-oriented transports. Minimizing latency, by reducing the number of RTTs needed for negotiation, requires changes to the implementations of the transport protocols being negotiated.

In the absence of an explicit end-to-end signaling or a negotiation protocol, the only way for an end-host to discover and (implicitly) agree on the choice of protocol(s) is to *simultaneously try* a set of candidate methods, and choose one method that works. This "test-and-select" approach, known as *happy eyeballs* [138], has been proposed both for choosing between transports [139], [140] and between versions of the IP protocol [141]. To the best of our knowledge, only the latter has been implemented in real systems (e.g., [142]), coupled with address-selection algorithms such as [143], with a few papers (e.g., [144]–[146]) reporting on performance assessments of IP-version happy eyeballs.

Fig. 5 depicts a possible variant of happy eyeballs for a client to discover SCTP support, both at a server and along the path to the server. A drawback of this kind of technique is it increases both the number of packets sent, and (potentially) the server-side load and the amount of state created in middleboxes; hence, it does not scale well with the number of candidates to try. For instance, testing for native SCTP, SCTP-over-UDP and TCP, combined with both IPv4 and IPv6, would in principle require testing six protocol combinations (compared to two in the example). Moreover, happy eyeballs requires careful design of timers, needed to decide when to discard a trial for a given protocol choice. Also, the sequence in which trials are attempted can be important, to avoid systematic bias towards particular protocol choices.

It is important to consider the overhead in the design of a happy eyeballs algorithm, especially the overhead in terms of *added latency* for initiating a session. In general, any transport signaling or feature discovery/negotiation mechanism may incur either additional round-trip times (e.g., if connection attempts are serialized) or waiting delay (e.g., due to waiting for replies to two parallel connection requests). It is therefore essential to cache results to speed up subsequent trials. For instance, prior knowledge that protocol choice X works with destination D can be used to tune the testing process, e.g., by slightly delaying trials with protocols other than X [139]. Cached information can also inform the happy eyeballs mechanism to give preference to certain choices, e.g., ones expected to offer lower path latency [142]. Another overhead worth considering is that of *CPU and memory load* on servers. These could be, in principle, important performance metrics for transport-layer happy eyeballs, since creating transport

connections implies creating state in end hosts. However, results in [147] suggest this may not necessarily be a major issue, especially when considering the impact of caching and the overhead inherent to transport-layer security.

VI. Enabling User-Space Protocol Stacks

It is possible to run a transport as a user-space library, letting applications use the transport in user-space, rather than the one provided by an OS kernel. This can allow more portability and deployability across multiple OS and hardware platforms. This approach can enable easy introduction and ease testing of new features and protocols (e.g., a simple user-space TCP library (UTCP) used on top of MultiStack [148]).

In many systems, privileges are needed to add a new protocol and may not always be granted to the entity trying to introduce a new transport protocol. User-space transport implementations can be installed on a host machine without *root* privileges and, as pointed out in Section II-B, when run over UDP, no special privileges are needed to access the IP layer.[2] However, the use of user-space transports presents a range of challenges.

One challenge is that network I/O operations that originate in user-space can incur higher latency compared to network I/O operations handled in the kernel. MultiStack [148] offers a solution that enhances commodity operating systems with support for dedicated user-level network stacks. It can concurrently host a large number of independent stacks, and can fall back to the kernel stack if necessary. MultiStack provides high speed packet I/O at rates up to 10 Gb/s [148], by extending two components: the netmap framework [149] and the VALE software switch [150]. Using the netmap framework, Marinos *et al*. [151] show that using specialized user-level stacks can provide a substantial performance improvement compared to using generic protocol stacks.

Other libraries can help achieve fast packet I/O in user-space, such as the Data Plane Development Kit (DPDK) [152] and PACKET_MMAP [153]. DPDK is a set of libraries and drivers for fast packet processing mostly in Linux user-space. However, DPDK is not a networking stack and does not provide functions such as Layer-3 forwarding, IPsec, firewalling, etc. PACKET_MMAP seeks to provide efficient raw packet transmission and reception in the Linux kernel using a zero-copy mechanism with a configurable circular buffer, mapped in user-space to minimize the number of system calls.

In addition to user-space TCP [154], there is also a user-space SCTP implementation for all major OS platforms [83] using the FreeBSD kernel sources for SCTP. Since it is not always possible to send data directly over native SCTP (e.g., because not all middleboxes can process SCTP packets), the SCTP user-space implementation in [83] additionally supports the option of encapsulating SCTP packets in UDP.

User-space SCTP [83] is implemented using *raw sockets* in user-space. A raw socket receives or sends raw datagrams (at OSI Layer 3), whereas *packet sockets* receive or send raw packets at the device driver level (OSI Layer 2). This allows a user to implement protocol modules in user-space on top of the physical layer (e.g., PACKET_MMAP [153]).

Another technique that enables transport protocols to run in user-space is to run the entire kernel (instead of only the transport) as a user-space process, as in User-Mode Linux (UML) [155]. This permits experimenting with new transport protocols implemented in different Linux kernels without interfering with the host Linux setup. UML provides a virtual machine as a single file, potentially with more (virtual) hardware/software resources than the actual host, and can potentially provide limited access to host hardware. A similar approach is followed by LibOS [156], which runs the kernel as a library that can be called by an application. LibOS has been used by NUSE [157] to provide a Linux network stack for user-space applications.

VII. Summary of Point Solutions

Table II summarises the taxonomy of issues and point solutions to transport-layer ossification described in more detail in the previous sections. The first column recaps the four main reasons behind ossification, discussed in Section I-A:

- The first two main problems, *Middlebox-related hindrances* and *API ossification*, are those that have received the most attention by the research and standards communities; this is reflected by the number of point solutions (examined in Sections II–III and IV, respectively) that have been proposed in this space.
- For clarity, the table subdivides families of solutions for the last two types of issues, *Lack of local knowledge about path- and remote end-host support* and *End-host deployment issues* (examined in Sections V and VI, respectively), according to the different approaches taken by the reviewed proposals.

VIII. A Way Forward: A Transport-Layer Framework

The previous sections have shown that de-ossifying the Internet transport layer to re-enable its evolution is a multi-dimensional problem. This requires the enhancement of multiple components of the end-to-end communication. Several point solutions have been proposed or are underway, each aiming to address a specific aspect of the overall problem. However, there has been little effective integration of techniques that can produce an evolvable transport layer.

For instance, incorporating a new application-level transport within the application's code (e.g., QUIC) to enable new transport services would inevitably require a negotiation service, e.g., a negotiation protocol like the one described in [137] to discover if the transport is supported by the remote peer (e.g., a Web server), accompanied with a fall-back strategy for the case where the new transport is not supported.[3] Implementing more advanced transport and network functions, such as dynamic selection and configuration of a transport based on current network state and QoS negotiation, would additionally require

[2]This requires that UDP port numbers $\geq$ 1024 be used.

[3]At the time of writing, the Chrome browser (version 46.0.2490.86) does this by implementing Happy Eyeballs (see Section V) between QUIC/UDP and TCP.

TABLE II
SUMMARY OF MAIN ISSUES AND POINT SOLUTIONS TO INTERNET TRANSPORT-LAYER OSSIFICATION

Type of problem	Family of solutions	Solution approach	Examples of proposals
Middlebox-related hindrances	Design of middlebox-proof transports (§ II)	Extending TCP while guarding against middlebox interference (§ II-A)	MPTCP [16], [48], [49] Tcpcrypt [51], [52] Gentle Aggression [53] TCP HICCUPS [47] Extending TCP's option space [55]–[58] Polyversal TCP [59]
		Using widely deployed transports as substrates (§ II-B)	Minion [64], [65] Encapsulation of specific transport protocols over UDP [78]–[80] Generic encapsulation frameworks [81], [82] Reinterpreting UDP packet headers [84] Relaxation of TCP semantics [84]–[87] Use of DTLS as a substrate [88]
	Signaling for facilitating middlebox traversal (§ III)	Implicit middlebox control (§ III-A)	STUN [93] TURN [94], [97], [98] UDP hole punching [18] TCP hole punching [18], [96] ICE [91], [95]
		Explicit middlebox control (§ III-B)	UPnP IGD [100] PCP [101] MIDCOM [103] NSIS Signaling Layer Protocol [104] PLUS (previously SPUD) [107]
API ossification	Enhancing the API between the applications and the transport layer (§ IV)	More expressive APIs / Extensions to the socket API (§ IV-B)	Msocket [112] Sockets++ [116] Multi-Sockets [115] Socket Intents [113], [114] QoSockets [117] Qsockets [118]
		Transparent transport protocol selection (§ IV-C)	Reuther et al. [121] Welzl [26] Welzl et al. [124] Siddiqui and Mueller [125] TAPS [119], [127]
Lack of local knowledge about path- and remote end-host support	Discovery and exploitation of end-to-end capabilities (§ V)	Explicit negotiation protocols	BEEP [133], [134] Rosenberg and Schulzrinne [136] Ford and Iyengar [137]
		Implicit discovery / agreement	Happy eyeballs [139], [141]
End-host deployment issues	Enabling user-space protocol stacks (§ VI)	Transport protocols implemented in user-level libraries	User-space TCP [154] User-space SCTP [83]
		Enabling fast packet I/O in user-space	MultiStack [148] Specialized user-level stacks [151] DPDK [152] PACKET_MMAP [153]
		Running an OS kernel in user-space	User-Mode Linux [155] LibOS [156]

the involvement of more components, such as a policy system, measurement modules and network signaling mechanisms, that need to interact with each other.

While various solutions could be partly implemented according to *certain* application needs, this would inevitably result in an application-specific and less flexible implementation, that is neither sufficiently general to support other types of applications nor incrementally upgradable to support new transport and network functions as they become available. This would need considerable effort from application developers to re-implement common functions or services that might not be interoperable or efficient. Examples include QUIC in Chrome, RTMFP [71] in Adobe Flash Player, and proprietary protocols in Skype [158] and the WebRTC framework [92].

We argue that a truly evolvable Internet transport architecture requires a necessary step to design and develop a

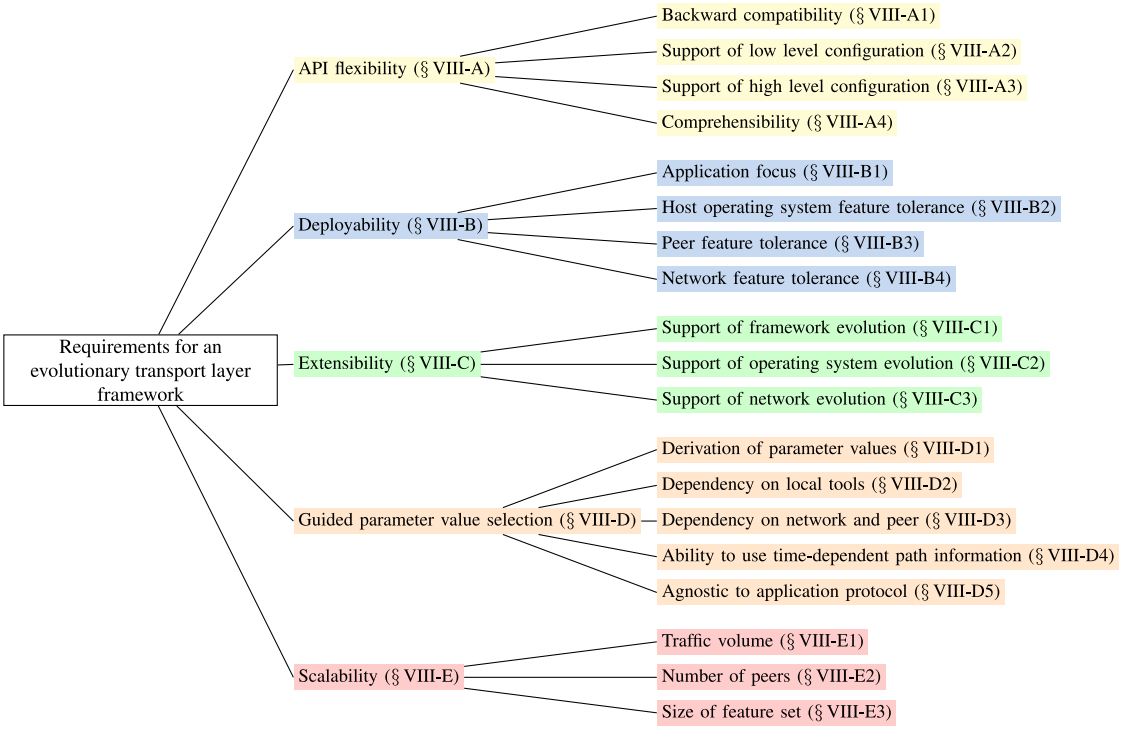


Fig. 6. Requirements for an evolutionary transport layer framework, as presented in Section VIII. Leaves and nodes in the tree correspond to requirements and their categories, respectively. Relevant sections in the text are shown in parentheses.

comprehensive and evolutionary *transport layer framework* that can facilitate integration and cooperation of transport layer solutions in an application-independent and flexible way.

This would relieve application developers from the burden of changing the application code to introduce new transport or network services and functions, breaking the vicious circle that hampers evolution.

The remainder of this section motivates the requirements for such a framework. Based on the discussion in previous sections, we identify such requirements and summarize them in five general categories: 1) API flexibility, 2) Deployability, 3) Extensibility, 4) Guided parameter value selection, and 5) Scalability. Figure 6 provides a visual guide to the requirements presented below.

A. API Flexibility

As discussed in Section IV, the ossification of the current transport API is a key obstacle that needs to be overcome. Applications using the framework should only interact with it via the API provided by the framework. This API should be able to decouple applications from a priori decisions on underlying protocols and functions. It should also allow to use the framework in the future by providing a simple way for porting existing applications to it. To this end, the API must be flexible, in the sense of the following requirements.

1) Backward Compatibility: The API provided by the framework needs to provide backward compatibility to enable evolution from previous versions of the framework without affecting the applications that use the framework.

2) Support of Low Level Configuration: The classical socket API requires detailed usage of the transport protocol stack, where the network and transport protocol need to be specified, and protocol-specific parameters chosen (when values different other than the defaults are needed). The framework should continue to permit this detailed level of configuration.

3) Support of High Level Configuration: The framework should allow configuration at a high level of abstraction. Mechanisms should describe the needs of an application in a more generic way than required by the classic socket API. Possible needs include message-orientation, preservation of message order, reliability, low latency, mobility support, relative priorities and security features.

An application may assume that it receives the requested service, but should not implicitly receive additional services. This allows the framework to make any further decisions necessary to establish optimal communication with the peer endpoint. As the framework evolves, different choices might lead to a better service without the need to change the application. Finally, multiple levels of abstraction need to be supported.

Recent advances in the development of more expressive, high-level, extensions to the socket API (e.g., Socket Intents and QSockets, Section IV-B2), and the important ongoing standardization effort of the IETF TAPS working group can provide a basis towards satisfying this requirement.

4) Comprehensibility: The framework must make low level information available to the application and to reveal the decision processes, so that applications know the concrete choices that were made to fulfill the requested abstract requirements. QoS feedback, as provided by QoSockets and QSockets (Section IV-B2), is an example of how such low level information could be of interest to an application.

B. Deployability

The framework should enable fast seamless deployment with as little disruption as possible. The deployability goals translate into the following requirements:

1) Application Focus: The evolutionary character of the framework requires support of existing host operating systems. It must be installable, usable and upgradable without specific privileges. This enables the speed of the evolution of the framework to be independent of the speed that operating systems are updated.

2) Host Operating System Feature Tolerance: The framework should not only make use of protocols and features available on the host operating system, but allow integration of additional protocols (e,g., SCTP or DCCP) and features (e.g., caching network or transport information).

To enable easy deployment of new transport protocols and/or transport protocol components, solutions that enable the deployment of user space transport stacks should be supported by the framework. Examples include support for application-level transports (such as uTP and QUIC, Section II-B), UDP encapsulation schemes (such as SCTP/UDP encapsulation and GUT, Section II-B), and user space implementations of native transports (such as SCTP and TCP, Section VI).

3) Peer Feature Tolerance: It can not be assumed either that all endpoints use the new framework. Even when the framework is supported by all endpoints, it must not be assumed that they use the same version of the framework. This allows for incremental deployment, possibly at the cost of providing less benefit. Similar robustness is required for the protocols and mechanisms used to realize the transport service.

A method that allows implicit or explicit discovery of the set of protocols/mechanisms supported by a remote endpoint could allow the framework to leverage the best common set of available features. Examples of such solutions are the negotiation protocol described in [137] and the happy eyeballs mechanisms (Section V). Feature negotiation and fallback mechanisms can be incorporated within a protocol or a mechanism itself, such as the options mechanism for negotiating TCP extensions and the fallback scheme of MPTCP (Section II-A).

4) Network Feature Tolerance: The ability to use the framework must not depend on the network support for specific features (e.g., quality of service mechanisms or middlebox interaction), but may utilize these when they are found to be supported.

Support for middlebox-proof transports (Section II) and mechanisms for implicit middlebox control (Section III-A) can be of great value for making the framework independent of the features supported by middleboxes. Additionally, support for "looser" network signaling mechanisms (e.g., PLUS, Section III-B) for interacting with network devices can enable a "best effort" use of available network features.

C. Extensibility

The framework must be able to support seamless, independent evolution of the different components.

1) Support of Framework Evolution: An evolutionary framework must permit addition of new protocols and features in the future.

2) Support of Operating System Evolution: The interface between the framework and the operating system may change over time to improve the service provided by the framework, including additional protocols and features. This allows moving implementations from the framework to the host operating systems and vice versa as they evolve.

3) Support of Network Evolution: Some middleboxes may allow an endpoint to signal its needs. Applications should not rely on signaling, but can benefit when this is available, possibly increasing the chance that a path can be used (e.g., by explicitly controlling middlebox traversal, Section III-B), or even enabling features (such as QoS support) that can benefit the transport (e.g., through the signaling of advisory metadata, Section IV-D). It should be assumed that the available methods for interacting with the network (and middleboxes) will evolve over time. The architecture of the framework must therefore allow applications using the framework to benefit from this evolution.

D. Guided Parameter Value Selection

Current transport and network stacks require explicit parameter value selection. For example, an application may choose IPv4 or IPv6 and select DCCP, SCTP, TCP, UDP-Lite or UDP. Furthermore, parameter values can be specified by explicit socket or protocol level socket options. The framework should be able to combine network-wide and local information to select the appropriate parameter values that make the best of available features for satisfying application requirements. Such guided parameter value selection corresponds to the following requirements.

1) Derivation of Parameter Values: The framework must map high-level requirements provided by the application to the low level parameter values to be used. This parameter selection should be guided by the requirements provided by each application to result in selection of the interfaces to be used, the network protocol, the transport protocol, and the setting of parameter values at each layer. Examples include the policy-based interface selection system of Socket Intents (Section IV-B2) and the run-time service broker in [121] (Section IV-C). The IETF TAPS working group is seeking to provide guidance on choosing among available protocols and mechanisms [25].

2) Dependency on Local Tools: If possible, tools included in an operating system (for example, link status supervision tools) should provide useful information to the framework when making the decisions and parameter value selections.

3) Dependency on Network and Peer: Any decision to use a particular protocol must be based on the set of protocols supported by the local and remote endpoints. A prerequisite to using a protocol is that it can communicate over the path between the endpoints, including any middleboxes employed along the path. The framework should support mechanisms for discovering characteristics of the end-to-end path and/or the remote endpoint, such as happy eyeballs, end-to-end signaling and negotiation protocols (Section V).

4) Ability to Use Time-Dependent Path Information: The final decision to use a candidate protocol can be based on

historical information such as whether a protocol or feature was previously supported on the path, but needs to also consider that path characteristics can change over both long time-scales (e.g., due to upgrades or route changes) and short time-scales (due to load balancing over alternate paths, wireless links, etc.). Use of historical information will require components for caching path properties (e.g., caching happy eyeballs results, Section V) and which will be able to efficiently store information with diverse lifetime requirements.

5) Agnostic to Application Protocol: Testing and discovery must be done by the framework and must not require any change to, or specific support by, application protocols.

E. Scalability

The framework must be scalable in a variety of ways.

1) Traffic Volume: The framework must limit the impact on CPU load and scale to support a high volume of user traffic (e.g., to support high-speed interfaces). Hardware support should be leveraged whenever possible. At the same time, the framework must not by itself produce control traffic (signaling) that limits scalability.

2) Number of Peers: The number of transport associations needed (for example TCP connections or SCTP associations) depend on the use case. The framework must efficiently support a high number of simultaneous transport associations.

3) Size of Feature Set: Finally, the framework needs to be able to support a variety of combinations of protocols, parameter settings and network interactions. The selection process must therefore be able to select from a large set of possibilities, while providing an acceptable communication setup time.

IX. Future Research Directions

To conclude, we identify ongoing and forthcoming research efforts that we expect will lead to further developments towards de-ossifying the transport layer.

Considering the approaches discussed so far, it seems that the ossification problem has two main root causes: 1) middleboxes that examine and/or manipulate the contents of packets beyond the IP header make it hard to deploy protocols that these middleboxes do not yet know; 2) the application networking interface that is exposed by the socket API ties applications (or the middleware or library that these applications are based upon) to a specific protocol choice. Both sub-problems have been addressed in various ways by research proposals. Unfortunately, some of these proposals are not new, yet it seems that present solutions have had little to no impact on the Internet: the transport layer still appears to consist of only TCP and UDP, often even further constrained to specific port numbers [12]. If anything, the situation seems to have worsened over the years.

There is however some reason for hope that we may be reaching a turning point. At the time of writing, several initiatives are focusing on making such a change possible; these initiatives point at the different open research directions in this space:

- The IETF TAPS working group seeks to specify how applications could express their transport requirements, instead of being tied to a specific protocol, and how a transport system based on such requirements specifications could be constructed.[4] This work begins with identifying the services that current IETF transport protocols provide [119], [127], [159]. An outcome of TAPS could include a new abstract API, and it will include recommendations on how to perform selection between protocols. One of the group's documents [25] provides guidance on choosing the minimal set of Transport Services that end systems should expose. Identifying this minimal set is important as not exposing some Transport Services limits the ability to benefit from protocols other than TCP and UDP. For example, SCTP can deliver delimited messages faster than TCP in case they arrive out-of-order, for applications that can tolerate such out-of-order delivery. With most of today's APIs providing a reliable byte stream, there is no way to automatize the use of this SCTP service, and just replacing TCP with SCTP does not necessarily yield much benefit (with the possible exceptions of potential gains from transparent use of multi-streaming [160], and increased resilience due to multihoming [161]).
- The IP Stack Evolution Program within the Internet Architecture Board (IAB) provides architectural guidance, and a point of coordination for work at the architectural level to improve the present situation of ossification in the Internet protocol stack.[5] This program provides a forum for discussion of design principles to make new Internet protocols deployable, based in part on RFC 5218 [162], and principles for the use of encapsulation (e.g., UDP-based). It has also organized a number of workshops and other meetings around topics related to stack evolution—e.g., the "Managing Radio Networks in an Encrypted World" (MaRNEW) workshop which focused on questions related to network management in the face of increasingly ubiquitous encryption.[6]
- Current activity around the PLUS protocol at the IETF[7] is striving for better visibility and control over the cooperation between end-points and middleboxes in a context of increasing use of encryption. "Birds-of-a-Feather" (BoF) sessions related to PLUS (and its predecessor SPUD) were held at two IETF meetings (IETF-92, Dallas, March 2015; IETF-96, Berlin, July 2016) and were well attended, with much debate on many aspects of the problem space that PLUS intends to cover, especially on the question of privacy implications of this proposal.[8]
- The IRTF "Measurement and Analysis for Protocols" Research Group[9] (MAPRG), chartered on August 2016,

[4] https://tools.ietf.org/wg/taps/charters

[5] https://www.iab.org/activities/programs/ip-stack-evolution-program/

[6] https://www.iab.org/activities/workshops/marnew/

[7] https://www.ietf.org/mailman/listinfo/spud

[8] See https://www.ietf.org/proceedings/92/minutes/minutes-92-spud and https://www.ietf.org/proceedings/96/minutes/minutes-96-plus for the sessions' minutes.

[9] https://irtf.org/maprg

serves as a forum to exchange insights derived from measuring the Internet, including the possibility to design protocols based on measured path characteristics, rather than conservatively designing just for the worst case.

- The European collaborative research project "NEAT" implements a transport system, following the requirements detailed in Section VIII, that will allow transport decisions to be made and verified *at run-time*, instead of design time, based on understanding application needs and the available transport protocols [163]—this is key to breaking the vicious circle and enabling deployment of new transports.[10] NEAT contributes to the TAPS working group [25], [119], [127], [140], [159], and the project welcomes contributions to their open-source implementation on github.[11]
- The European collaborative research project "MAMI" is set to perform a large-scale assessment of middlebox behavior [164], and to use this to inform development of an architecture for middlebox cooperation.[12] MAMI is involved in several of the activities listed above: development of the PLUS protocol, creation of the MAPRG, TAPS and the IAB Stack Evolution program.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their useful suggestions and comments.

REFERENCES

[1] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. Comput. Syst.*, vol. 2, no. 4, pp. 277–288, Nov. 1984.

[2] M. D. Schroeder, D. D. Clark, and J. H. Saltzer, "The multics kernel design project," in *Proc. 6th ACM Symp. Oper. Syst. Principles (SOSP)*, West Lafayette, IN, USA, 1977, pp. 43–56.

[3] J. Postel, "Transmission Control Protocol," Internet Eng. Task Force, Fremont, CA, USA, RFC 793 (Internet Standard), Sep. 1981. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[4] M. Handley, "Why the Internet only just works," *BT Technol. J.*, vol. 24, no. 3, pp. 119–129, 2006.

[5] J. Postel, "User Datagram Protocol," Internet Eng. Task Force, Fremont, CA, USA, RFC 768 (Internet Standard), Aug. 1980. [Online]. Available: http://www.ietf.org/rfc/rfc768.txt

[6] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)," Internet Eng. Task Force, Fremont, CA, USA, RFC 4340 (Proposed Standard), Mar. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4340.txt

[7] R. Stewart and C. Metz, "SCTP: New transport protocol for TCP/IP," *IEEE Internet Comput.*, vol. 5, no. 6, pp. 64–69, Nov./Dec. 2001.

[8] R. Stewart, "Stream Control Transmission Protocol," Internet Eng. Task Force, Fremont, CA, USA, RFC 4960 (Proposed Standard), Sep. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4960.txt

[9] L.-A. Larzon, M. Degermark, S. Pink, L.-E. Jonsson, and G. Fairhurst, "The Lightweight User Datagram Protocol (UDP-Lite)," Internet Eng. Task Force, Fremont, CA, USA, RFC 3828 (Proposed Standard), Jul. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3828.txt

[10] M. S. Blumenthal and D. D. Clark, "Rethinking the design of the Internet: The end-to-end arguments vs. the brave new world," *ACM Trans. Internet Technol.*, vol. 1, no. 1, pp. 70–109, Aug. 2001.

[11] H. Tschofenig, "The new waist of the hourglass," Internet Draft, draft-tschofenig-hourglass-00, Jul. 2012. [Online]. Available: http://tools.ietf.org/html/draft-tschofenig-hourglass-00

[12] L. Popa, A. Ghodsi, and I. Stoica, "HTTP as the narrow waist of the future Internet," in *Proc. ACM SIGCOMM Workshop Hot Topics Netw. (HotNets)*, Monterey, CA, USA, 2010, Art. no. 6.

[10] https://www.neat-project.org

[11] https://github.com/NEAT-project/neat

[12] https://mami-project.eu

[13] B. Ford and J. R. Iyengar, "Breaking up the transport logjam," in *Proc. ACM SIGCOMM Workshop Hot Topics Netw. (HotNets)*, Calgary, AB, Canada, 2008, pp. 85–90.

[14] A. Medina, M. Allman, and S. Floyd, "Measuring the evolution of transport protocols in the Internet," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 2, pp. 37–52, 2005.

[15] B. Carpenter and S. Brim, "Middleboxes: Taxonomy and issues," Internet Eng. Task Force, Fremont, CA, USA, RFC 3234 (Informational), Feb. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3234.txt

[16] M. Honda *et al.*, "Is it still possible to extend TCP?" in *Proc. ACM IMC*, Berlin, Germany, 2011, pp. 181–194.

[17] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden, "Tussle in cyberspace: Defining tomorrow's Internet," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 347–356, 2002.

[18] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-peer communication across network address translators," in *Proc. USENIX Annu. Tech. Conf. Gen. Track*, Anaheim, CA, USA, 2005, pp. 179–192.

[19] R. Sparks, "SIP: Basics and beyond," *ACM Queue*, vol. 5, no. 2, pp. 22–33, Mar. 2007.

[20] J. Rosenberg *et al.*, "SIP: Session Initiation Protocol," Internet Eng. Task Force, Fremont, CA, USA, RFC 3261 (Proposed Standard), Jun. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3261.txt

[21] J. Postel and J. Reynolds, "File Transfer Protocol," Internet Eng. Task Force, Fremont, CA, USA, RFC 959 (Internet Standard), Oct. 1985. [Online]. Available: http://www.ietf.org/rfc/rfc959.txt

[22] G. Detal, C. Paasch, and O. Bonaventure, "Multipath in the middle(box)," in *Proc. Workshop Hot Topics Middleboxes Netw. Funct. Virtualization (HotMiddlebox)*, Santa Barbara, CA, USA, Dec. 2013, pp. 1–6.

[23] A. Tachibana, Y. Yoshida, M. Shibuya, and T. Hasegawa, "Implementation of a proxy-based CMT-SCTP scheme for Android smartphones," in *Proc. IEEE 10th Int. Conf. Wireless Mobile Comput. Netw. Commun. (WiMob)*, Larnaca, Cyprus, Oct. 2014, pp. 660–665.

[24] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "Performance enhancing proxies intended to mitigate link-related degradations," Internet Eng. Task Force, Fremont, CA, USA, RFC 3135 (Informational), Jun. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3135.txt

[25] S. Gjessing and M. Welzl, "A minimal set of transport services for TAPS systems," Internet Draft, draft-gjessing-taps-minset-02, Jul. 2016. [Online]. Available: https://tools.ietf.org/html/draft-gjessing-taps-minset-02

[26] M. Welzl, "A case for middleware to enable advanced Internet services," in *Proc. Next Gener. Netw. Middleware Workshop (NGNM)*, Athens, Greece, May 2004, pp. 20/1–20/5.

[27] S.-U. Lar and X. Liao, "An initiative for a classified bibliography on TCP/IP congestion control," *J. Netw. Comput. Appl.*, vol. 36, no. 1, pp. 126–133, Jan. 2013.

[28] G. Xylomenos *et al.*, "A survey of information-centric networking research," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 2, pp. 1024–1049, 2nd Quart., 2014.

[29] Q. Chen *et al.*, "Transport control strategies in named data networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 3, pp. 2052–2083, 3rd Quart., 2016.

[30] M. Patel *et al.*, "Mobile-edge computing," White Paper, Eur. Telecommun. Standards Inst., Sophia Antipolis, France, Sep. 2014. [Online]. Available: https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge_computing_-_introductory_technical_white_paper_v1%2018-09-14.pdf

[31] Y. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—A key technology towards 5G," White Paper, Eur. Telecommun. Standards Inst., Sophia Antipolis, France, Sep. 2015. [Online]. Available: http://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp11_mec_a_key_technology_towards_5g.pdf

[32] J. S. Turner and D. E. Taylor, "Diversifying the Internet," in *Proc. IEEE GLOBECOM*, St. Louis, MO, USA, Nov. 2005, p. 6.

[33] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A blueprint for introducing disruptive technology into the Internet," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 59–64, Jan. 2003.

[34] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet impasse through virtualization," *IEEE Comput.*, vol. 38, no. 4, pp. 34–41, Apr. 2005.

[35] K. C. Almeroth, "The evolution of multicast: From the MBone to inter-domain multicast to Internet2 deployment," *IEEE Netw.*, vol. 14, no. 1, pp. 10–20, Jan./Feb. 2000.

[36] R. Boivie, N. Feldman, Y. Imai, W. Livens, and D. Ooms, "Explicit multicast (Xcast) concepts and options," Internet Eng. Task Force, Fremont, CA, USA, RFC 5058 (Experimental), Nov. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc5058.txt

[37] A. Boudani, A. Guitton, and B. Cousin, "GXcast: Generalized explicit multicast routing protocol," in *Proc. IEEE ISCC*, vol. 2. Alexandria, Egypt, 2004, pp. 1012–1017.

[38] T. Narten *et al.*, "Problem statement: Overlays for network virtualization," Internet Eng. Task Force, Fremont, CA, USA, RFC 7364 (Informational), Oct. 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7364.txt

[39] *IETF Network Virtualization Overlays (nvo3) Working Group Charter*. Accessed on Nov. 16, 2016. [Online]. Available: https://datatracker.ietf.org/doc/charter-ietf-nvo3/

[40] T. Li, "Recommendation for a routing architecture," Internet Eng. Task Force, Fremont, CA, USA, RFC 6115 (Informational), Feb. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6115.txt

[41] D. Saucez, L. Iannone, O. Bonaventure, and D. Farinacci, "Designing a deployable Internet: The locator/identifier separation protocol," *IEEE Internet Comput.*, vol. 16, no. 6, pp. 14–21, Nov./Dec. 2012.

[42] F. Coras, J. Domingo-Pascual, F. Maino, D. Farinacci, and A. Cabellos-Aparicio, "Lcast: Software-defined inter-domain multicast," *Comput. Netw.*, vol. 59, pp. 153–170, Feb. 2014.

[43] Y. Benchaïb, S. Secci, and C.-D. Phung, "Transparent cloud access performance augmentation via an MPTCP-LISP connection proxy," in *Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, Oakland, CA, USA, May 2015, pp. 201–202.

[44] M. Coudron, S. Secci, G. Maier, G. Pujolle, and A. Pattavina, "Boosting cloud communications through a crosslayer multipath protocol architecture," in *Proc. IEEE Softw. Defined Netw. Future Netw. Services (SDN4FNS)*, Trento, Italy, Nov. 2013, pp. 1–8.

[45] A. Langley. (2008). *Probing the Viability of TCP Extensions*. [Online]. Available: http://www.imperialviolet.org/binary/ecntest.pdf

[46] G. Detal, B. Hesmans, O. Bonaventure, Y. Vanaubel, and B. Donnet, "Revealing middlebox interference with tracebox," in *Proc. ACM IMC*, Barcelona, Spain, 2013, pp. 1–8.

[47] R. Craven, R. Beverly, and M. Allman, "A middlebox-cooperative TCP for a non end-to-end Internet," in *Proc. ACM SIGCOMM*, Chicago, IL, USA, 2014, pp. 151–162.

[48] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP extensions for multipath operation with multiple addresses," Internet Eng. Task Force, Fremont, CA, USA, RFC 6824 (Experimental), Jan. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc6824.txt

[49] C. Raiciu *et al*., "How hard can it be? Designing and implementing a deployable multipath TCP," in *Proc. USENIX NSDI*, vol. 12. San Jose, CA, USA, 2012, p. 29.

[50] C. Nicutar, C. Paasch, M. Bagnulo, and C. Raiciu, "Evolving the Internet with connection acrobatics," in *Proc. Workshop Hot Topics Middleboxes Netw. Funct. Virtualization (HotMiddlebox)*, Santa Barbara, CA, USA, 2013, pp. 7–12.

[51] A. Bittau, M. Hamburg, M. Handley, D. Mazières, and D. Boneh, "The case for ubiquitous transport-level encryption," in *Proc. USENIX Security Symp.*, Austin, TX, USA, 2010, pp. 403–418.

[52] A. Bittau *et al.*, "Cryptographic protection of TCP streams (tcpcrypt)," Internet Draft, draft-bittau-tcp-crypt-04, Feb. 2014. [Online]. Available: http://tools.ietf.org/html/draft-bittau-tcp-crypt-04

[53] T. Flach *et al.*, "Reducing Web latency: The virtue of gentle aggression," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 159–170, Oct. 2013.

[54] A. Ramaiah, "TCP option space extension," Internet Draft, draft-ananth-tcpm-tcpoptext-00, Mar. 2012. [Online]. Available: http://tools.ietf.org/html/draft-ananth-tcpm-tcpoptext-00

[55] J. Touch and W. Eddy, "TCP extended data offset option," Internet Draft, draft-ietf-tcpm-tcp-edo-06, Jun. 2016. [Online]. Available: http://tools.ietf.org/html/draft-ietf-tcpm-tcp-edo-06

[56] H. Trieu, J. Touch, and T. Faber, "Implementation of the TCP extended data offset option," USC Inf. Sci. Inst., Marina Del Rey, CA, USA, Tech. Rep. ISI-TR-696, Mar. 2015.

[57] J. Touch and T. Faber, "TCP SYN extended option space using an out-of-band segment," Internet Draft, draft-touch-tcpm-tcp-syn-ext-opt-04, Apr. 2016. [Online]. Available: http://tools.ietf.org/html/draft-touch-tcpm-tcp-syn-ext-opt-04

[58] B. Briscoe, "Inner space for TCP options," Internet Draft, draft-briscoe-tcpm-inner-space-01, Oct. 2014. [Online]. Available: http://tools.ietf.org/html/draft-briscoe-tcpm-inner-space-01

[59] Z. Nabi, T. Moncaster, A. Madhavapeddy, S. Hand, and J. Crowcroft, "Evolving TCP. How hard can it be?" in *Proc. ACM CoNEXT Student Workshop*, Nice, France, 2012, pp. 35–36.

[60] B. Trammell and M. Kuehlewind, "Report from the IAB workshop on stack evolution in a middlebox Internet (SEMI)," Internet Eng. Task Force, Fremont, CA, USA, RFC 7663 (Informational), Oct. 2015. [Online]. Available: http://www.ietf.org/rfc/rfc7663.txt

[61] *IRTF Measurement and Analysis for Protocols (MAP) Proposed Research Group Charter*. Accessed on Nov. 16, 2016. [Online]. Available: https://datatracker.ietf.org/doc/charter-irtf-maprg/

[62] F. Audet and C. Jennings, "Network address translation (NAT) behavioral requirements for unicast UDP," Internet Eng. Task Force, Fremont, CA, USA, RFC 4787 (Best Current Practice), Jan. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4787.txt

[63] L. Eggert, G. Fairhurst, and G. Shepherd, "UDP usage guidelines," Internet Draft, draft-ietf-tsvwg-rfc5405bis, Nov. 2015. [Online]. Available: https://tools.ietf.org/html/draft-ietf-tsvwg-rfc5405bis

[64] M. F. Nowlan, N. Tiwari, J. Iyengar, S. O. Aminy, and B. Ford, "Fitting square pegs through round pipes: Unordered delivery wire-compatible with TCP and TLS," in *Proc. USENIX NSDI*, San Jose, CA, USA, 2012, p. 28.

[65] J. Iyengar, S. Cheshire, and J. Graessley, "Minion—Wire protocol," Internet Draft, draft-iyengar-minion-protocol-02, Oct. 2013. [Online]. Available: http://tools.ietf.org/html/draft-iyengar-minion-protocol-02

[66] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," Internet Eng. Task Force, Fremont, CA, USA, RFC 3550 (Internet Standard), Jul. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3550.txt

[67] J. Roskind, "QUIC: Multiplexed stream transport over UDP," Google Working Design Document, Dec. 2013. [Online]. Available: https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/edit

[68] S. R. Das, "Evaluation of QUIC on Web page performance," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, 2014.

[69] Google. *SPDY: An Experimental Protocol for a Faster Web*. Accessed on Nov. 16, 2016. [Online]. Available: http://www.chromium.org/spdy/spdy-whitepaper

[70] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol version 2 (HTTP/2)," Internet Eng. Task Force, Fremont, CA, USA, RFC 7540 (Proposed Standard), May 2015. [Online]. Available: http://www.ietf.org/rfc/rfc7540.txt

[71] M. Thornburgh, "Adobe's Secure Real-Time Media Flow Protocol," Internet Eng. Task Force, Fremont, CA, USA, RFC 7016 (Informational), Nov. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc7016.txt

[72] V. Singh, S. Ahsan, and J. Ott, "MPRTP: Multipath considerations for real-time media," in *Proc. 4th ACM Multimedia Syst. Conf. (MMSys)*, Oslo, Norway, 2013, pp. 190–201.

[73] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security version 1.2," Internet Eng. Task Force, Fremont, CA, USA, RFC 6347 (Proposed Standard), Jan. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6347.txt

[74] A. Norberg, "uTorrent transport protocol," BitTorrent Enhancement Proposal 29, Jun. 2009. [Online]. Available: http://www.bittorrent.org/beps/bep_0029.html

[75] Y. Gu and R. L. Grossman, "UDT: UDP-based data transfer for high-speed wide area networks," *Comput. Netw.*, vol. 51, no. 7, pp. 1777–1799, 2007.

[76] Y. Gu and R. Grossman, "UDTv4: Improvements in performance and usability," in *Networks for Grid Applications*. Heidelberg, Germany: Springer, 2009, pp. 9–23.

[77] B. Ford, "Structured streams: A new transport abstraction," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 361–372, 2007.

[78] M. Tuexen and R. Stewart, "UDP encapsulation of Stream Control Transmission Protocol (SCTP) packets for end-host to end-host communication," Internet Eng. Task Force, Fremont, CA, USA, RFC 6951 (Proposed Standard), May 2013. [Online]. Available: http://www.ietf.org/rfc/rfc6951.txt

[79] T. Phelan, G. Fairhurst, and C. Perkins, "DCCP-UDP: A Datagram Congestion Control Protocol UDP encapsulation for NAT traversal," Internet Eng. Task Force, Fremont, CA, USA, RFC 6773 (Proposed Standard), Nov. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6773.txt

[80] S. Cheshire, J. Graessley, and R. McGuire, "Encapsulation of TCP and other transport protocols over UDP," Internet Draft, draft-cheshire-tcp-over-udp-00, Jul. 2013. [Online]. Available: http://tools.ietf.org/html/draft-cheshire-tcp-over-udp-00

[81] J. Manner, N. Varis, and B. Briscoe, "Generic UDP Tunnelling (GUT)," Internet Draft, draft-manner-tsvwg-gut-02, Jul. 2010. [Online]. Available: http://tools.ietf.org/html/draft-manner-tsvwg-gut-02

[82] T. Herbert, L. Yong, and O. Zia, "Generic UDP Encapsulation," Internet Draft, draft-ietf-nvo3-gue-04, Jul. 2016. [Online]. Available: http://tools.ietf.org/html/draft-ietf-nvo3-gue-04

[83] B. Penoff, A. Wagner, M. Tüxen, and I. Rüngeler, "Portable and performant userspace SCTP stack," in *Proc. 21st Int. Conf. Comput. Commun. Netw. (ICCCN)*, Munich, Germany, 2012, pp. 1–9.

[84] S. McQuistin, C. Perkins, and M. Fayed, "Implementing real-time transport services over an ossified network," in *Proc. ACM/IRTF/ISOC Appl. Netw. Res. Workshop (ANRW)*, Berlin, Germany, 2016, pp. 81–87.

[85] S. McQuistin, C. Perkins, and M. Fayed, "TCP goes to Hollywood," in *Proc. 26th Int. Workshop Netw. Oper. Syst. Support Digit. Audio Video (NOSSDAV)*, Klagenfurt, Austria, 2016, pp. 1–6.

[86] B. Mukherjee and T. Brecht, "Time-lined TCP for the TCP-friendly delivery of streaming media," in *Proc. IEEE ICNP*, Osaka, Japan, 2000, pp. 165–176.

[87] D. McCreary, K. Li, S. A. Watterson, and D. K. Lowenthal, "TCP-RC: A receiver-centered TCP protocol for delay-sensitive applications," in *Proc. Multimedia Comput. Netw. (MMCN)*, San Jose, CA, USA, 2005, pp. 126–130.

[88] C. Huitema, E. Rescorla, and J. Iyengar, "DTLS as subtransport protocol," Internet Draft, draft-huitema-tls-dtls-as-subtransport-00, Mar. 2015. [Online]. Available: http://tools.ietf.org/html/draft-huitema-tls-dtls-as-subtransport-00

[89] H. Schulzrinne, A. Rao, and R. Lanphier, "Real Time Streaming Protocol (RTSP)," Internet Eng. Task Force, Fremont, CA, USA, RFC 2326 (Proposed Standard), Apr. 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2326.txt

[90] J. Rosenberg, "Interactive Connectivity Establishment: NAT traversal for the Session Initiation Protocol," *IETF J.*, vol. 2, pp. 14–19, Nov. 2006. [Online]. Available: http://www.internetsociety.org/articles/interactive-connectivity-establishment

[91] J. Rosenberg, "Interactive Connectivity Establishment (ICE): A protocol for network address translator (NAT) traversal for offer/answer protocols," Internet Eng. Task Force, Fremont, CA, USA, RFC 5245 (Proposed Standard), Apr. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5245.txt

[92] A. Bergkvist, D. C. Burnett, C. Jennings, A. Narayanan, and B. Aboba, "WebRTC 1.0: Real-time communication between browsers," W3C, W3C Working Draft, Aug. 2016. [Online]. Available: http://www.w3.org/TR/webrtc/

[93] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, "Session Traversal Utilities for NAT (STUN)," Internet Eng. Task Force, Fremont, CA, USA, RFC 5389 (Proposed Standard), Oct. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5389.txt

[94] M. Petit-Huguenin, "Traversal Using Relays around NAT (TURN) resolution mechanism," Internet Eng. Task Force, Fremont, CA, USA, RFC 5928 (Proposed Standard), Aug. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5928.txt

[95] J. Rosenberg, A. Keranen, B. B. Lowekamp, and A. B. Roach, "TCP candidates with Interactive Connectivity Establishment (ICE)," Internet Eng. Task Force, Fremont, CA, USA, RFC 6544 (Proposed Standard), Mar. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6544.txt

[96] A. Biggadike, D. Ferullo, G. Wilson, and A. Perrig, "NATBLASTER: Establishing TCP connections between hosts behind NATs," in *Proc. ACM SIGCOMM Asia Workshop*, Beijing, China, 2005. [Online]. Available: http://www.netsec.ethz.ch/publications/papers/natblaster.pdf

[97] S. Perreault and J. Rosenberg, "Traversal Using Relays around NAT (TURN) extensions for TCP allocations," Internet Eng. Task Force, Fremont, CA, USA, RFC 6062 (Proposed Standard), Nov. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc6062.txt

[98] M. Petit-Huguenin and G. Salgueiro, "Datagram Transport Layer Security (DTLS) as transport for Session Traversal Utilities for NAT (STUN)," Internet Eng. Task Force, Fremont, CA, USA, RFC 7350 (Proposed Standard), Aug. 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7350.txt

[99] S. Guha, K. Biswas, B. Ford, S. Sivakumar, and P. Srisuresh, "NAT behavioral requirements for TCP," Internet Eng. Task Force, Fremont, CA, USA, RFC 5382 (Best Current Practice), Oct. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5382.txt

[100] "WANIPConnection: 2 service," UPnP Forum, Standardized DCP, London, U.K., Sep. 2010. [Online]. Available: http://upnp.org/specs/gw/UPnP-gw-WANIPConnection-v2-Service.pdf

[101] D. Wing, S. Cheshire, M. Boucadair, R. Penno, and P. Selkirk, "Port Control Protocol (PCP)," Internet Eng. Task Force, Fremont, CA, USA, RFC 6887 (Proposed Standard), Apr. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc6887.txt

[102] S. Cheshire and M. Krochmal, "NAT Port Mapping Protocol (NAT-PMP)," Internet Eng. Task Force, Fremont, CA, USA, RFC 6886 (Informational), Apr. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc6886.txt

[103] P. Srisuresh, J. Kuthan, J. Rosenberg, A. Molitor, and A. Rayhan, "Middlebox communication architecture and framework," Internet Eng. Task Force, Fremont, CA, USA, RFC 3303 (Informational), Aug. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3303.txt

[104] M. Stiemerling, H. Tschofenig, C. Aoun, and E. Davies, "NAT/firewall NSIS Signaling Layer Protocol (NSLP)," Internet Eng. Task Force, Fremont, CA, USA, RFC 5973 (Experimental), Oct. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5973.txt

[105] B. Trammell and J. Hildebrand, "Evolving transport in the Internet," *IEEE Internet Comput.*, vol. 18, no. 5, pp. 60–64, Sep./Oct. 2014.

[106] B. Trammell, "Architectural considerations for transport evolution with explicit path cooperation," Internet Draft, draft-trammell-stackevo-explicit-coop, Sep. 2015. [Online]. Available: http://tools.ietf.org/html/draft-trammell-stackevo-explicit-coop-00

[107] J. Hildebrand and B. Trammell, "Substrate Protocol for User Datagrams (SPUD) prototype," Internet Draft, draft-hildebrand-spud-prototype-03, Mar. 2015. [Online]. Available: http://tools.ietf.org/html/draft-hildebrand-spud-prototype-03

[108] D. Coffield and D. Shepherd, "Tutorial guide to Unix sockets for network communication," *Comput. Commun.*, vol. 10, no. 1, pp. 21–29, Feb. 1987.

[109] G. V. Neville-Neil, "Whither sockets?" *ACM Queue*, vol. 7, no. 4, p. 35, May 2009.

[110] J. Nagle, "Congestion control in TCP/IP Internetworks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 14, no. 4, pp. 11–17, Oct. 1984.

[111] R. Stewart, M. Tuexen, K. Poon, P. Lei, and V. Yasevich, "Sockets API extensions for the Stream Control Transmission Protocol (SCTP)," Internet Eng. Task Force, Fremont, CA, USA, RFC 6458 (Informational), Dec. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6458.txt

[112] R. Davoli and M. Goldweber, "Msocket: Multiple stack support for the Berkeley socket API," in *Proc. 27th Annu. ACM Symp. Appl. Comput. (SAC)*, Riva del Garda, Italy, 2012, pp. 588–593.

[113] T. Enghardt, "Socket intents: Extending the socket API to express application needs," M.S. thesis, Dept. Telecommun. Syst., Tech. Univ. Berlin, Berlin, Germany, Jul. 2013.

[114] P. S. Schmidt, T. Enghardt, R. Khalili, and A. Feldmann, "Socket Intents: Leveraging application awareness for multi-access connectivity," in *Proc. ACM CoNEXT*, Santa Barbara, CA, USA, Dec. 2013, pp. 295–300.

[115] B. D. Higgins *et al.*, "Intentional networking: Opportunistic exploitation of mobile network diversity," in *Proc. ACM Mobicom*, Chicago, IL, USA, 2010, pp. 73–84.

[116] S. Bocking, "Sockets++: A uniform application programming interface for basic level communication services," *IEEE Commun. Mag.*, vol. 34, no. 12, pp. 114–123, Dec. 1996.

[117] P. G. S. Florissi, Y. Yemini, and D. Florissi, "QoSockets: A new extension to the sockets API for end-to-end application QoS management," *Comput. Netw.*, vol. 35, no. 1, pp. 57–76, 2001.

[118] H. Abbasi, C. Poellabauer, K. Schwan, G. Losik, and R. West, "A quality-of-service enhanced socket API in GNU/Linux," in *Proc. 4th Real Time Linux Workshop*, Boston, MA, USA, Dec. 2002. [Online]. Available: https://www.osadl.org/fileadmin/events/rtlws-2002/proc/g08_abbasi.pdf

[119] G. Fairhurst, B. Trammell, and M. Kuehlewind, "Services provided by IETF transport protocols and congestion control mechanisms," Internet Draft, draft-ietf-taps-transports-11, Jul. 2016. [Online]. Available: http://tools.ietf.org/html/draft-ietf-taps-transports-11

[120] D. Henrici and B. Reuther, "Service-oriented protocol interfaces and dynamic intermediation of communication services," in *Proc. 2nd IASTED Int. Conf. Commun. Internet Inf. Technol. (CIIT)*, Scottsdale, AZ, USA, Nov. 2003. [Online]. Available: http://dspace.icsy.de:12000/dspace/bitstream/123456789/120/1/DPArchiv.0075.pdf

[121] B. Reuther, D. Henrici, and M. Hillenbrand, "DANCE: Dynamic application oriented network services," in *Proc. 30th Euromicro Conf.*, Rennes, France, 2004, pp. 298–305.

[122] D. Velten, R. Hinden, and J. Sax, "Reliable Data Protocol," Internet Eng. Task Force, Fremont, CA, USA, RFC 908 (Experimental), Jul. 1984. [Online]. Available: http://www.ietf.org/rfc/rfc908.txt

[123] W. T. Strayer, B. J. Dempsey, and A. C. Weaver, *XTP: The Xpress Transfer Protocol*. Reading, MA, USA: Addison-Wesley, 1992.

[124] M. Welzl, S. Jorer, and S. Gjessing, "Towards a protocol-independent Internet transport API," in *Proc. IEEE ICC*, Kyoto, Japan, Jun. 2011, pp. 1–6.

[125] A. A. Siddiqui and P. Mueller, "A requirement-based socket API for a transition to future Internet architectures," in *Proc. 6th Int. Conf. Innov. Mobile Internet Services Ubiquitous Comput. (IMIS)*, Palermo, Italy, Jul. 2012, pp. 340–345.

[126] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform resource identifier (URI): Generic syntax," Internet Eng. Task Force, Fremont, CA, USA, RFC 3986 (Internet Standard), Jan. 2005. [Online]. Available: http://www.ietf.org/rfc/rfc3986.txt

[127] M. Welzl, M. Tüxen, and N. Khademi, "On the usage of transport service features provided by IETF transport protocols," Internet Draft, draft-ietf-taps-transports-usage, Jul. 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-taps-transports-usage-01
[128] M. Kuehlewind and B. Trammell, "SPUD use cases," Internet Draft, draft-kuehlewind-spud-use-cases, Mar. 2016. [Online]. Available: http://tools.ietf.org/html/draft-kuehlewind-spud-use-cases-01
[129] T. Eckert, R. Penno, A. Choukir, and C. Eckel, "A framework for signaling flow characteristics between applications and the network," Internet Draft, draft-eckert-intarea-flow-metadata-framework, Oct. 2013. [Online]. Available: http://tools.ietf.org/html/draft-eckert-intarea-flow-metadata-framework-02
[130] S. Floyd, M. Allman, A. Jain, and P. Sarolahti, "Quick-start for TCP and IP," Internet Eng. Task Force, Fremont, CA, USA, RFC 4782 (Experimental), Jan. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4782.txt
[131] P. Sarolahti, M. Allman, and S. Floyd, "Determining an appropriate sending rate over an underutilized network path," *Comput. Netw.*, vol. 51, no. 7, pp. 1815–1832, May 2007.
[132] *IETF Transport Services (TAPS) Working Group Charter*. Accessed on Nov. 16, 2016. [Online]. Available: https://datatracker.ietf.org/doc/charter-ietf-taps/
[133] M. Rose, "An overview of BEEP," *Internet Protocol J.*, vol. 5, no. 2, pp. 2–11, Jun. 2002.
[134] M. Rose, "The Blocks Extensible Exchange Protocol core," Internet Eng. Task Force, Fremont, CA, USA, RFC 3080 (Proposed Standard), Mar. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3080.txt
[135] H. Schulzrinne and J. Rosenberg, "Internet telephony: Architecture and protocols—An IETF perspective," *Comput. Netw.*, vol. 31, no. 3, pp. 237–255, 1999.
[136] J. Rosenberg and H. Schulzrinne, "An offer/answer model with Session Description Protocol (SDP)," Internet Eng. Task Force, Fremont, CA, USA, RFC 3264 (Proposed Standard), Jun. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3264.txt
[137] B. Ford and J. R. Iyengar, "Efficient cross-layer negotiation," in *Proc. ACM SIGCOMM Workshop Hot Topics Netw. (HotNets)*, New York, NY, USA, 2009. [Online]. Available: http://conferences.sigcomm.org/hotnets/2009/papers/hotnets2009-final123.pdf
[138] D. Wing and A. Yourtchenko, "Improving user experience with IPv6 and SCTP," *Internet Protocol J.*, vol. 13, no. 3, pp. 16–21, Sep. 2010.
[139] D. Wing, A. Yourtchenko, and P. Natarajan, "Happy Eyeballs: Trending towards success (IPv6 and SCTP)," Internet Draft, draft-wing-http-new-tech-01, Aug. 2010. [Online]. Available: http://tools.ietf.org/html/draft-wing-http-new-tech-01
[140] K.-J. Grinnemo, A. Brunstrom, P. Hurtig, and N. Khademi, "Happy Eyeballs for transport selection," Internet Draft, draft-grinnemo-taps-he, Jul. 2016. [Online]. Available: https://tools.ietf.org/html/draft-grinnemo-taps-he-01
[141] D. Wing and A. Yourtchenko, "Happy Eyeballs: Success with dual-stack hosts," Internet Eng. Task Force, Fremont, CA, USA, RFC 6555 (Proposed Standard), Apr. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6555.txt
[142] D. Schinazi, "Apple and IPv6—Happy eyeballs," IETF, Fremont, CA, USA, Jul. 2015. [Online]. Available: https://www.ietf.org/mail-archive/web/v6ops/current/msg22455.html
[143] D. Thaler, R. Draves, A. Matsumoto, and T. Chown, "Default address selection for Internet Protocol version 6 (IPv6)," Internet Eng. Task Force, Fremont, CA, USA, RFC 6724 (Proposed Standard), Sep. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6724.txt
[144] V. Bajpai and J. Schönwälder, "Measuring TCP connection establishment times of dual-stacked Web services," in *Proc. 9th Int. Conf. Netw. Service Manag. (CNSM)*, Zürich, Switzerland, Oct. 2013, pp. 130–133.
[145] S. Ahsan, V. Bajpai, J. Ott, and J. Schönwälder, "Measuring YouTube from dual-stacked hosts," in *Proc. 16th Passive Active Meas. Conf. (PAM)*, New York, NY, USA, Mar. 2015, pp. 249–261.
[146] V. Bajpai and J. Schönwälder, "Measuring the effects of Happy Eyeballs," in *Proc. ACM/IRTF/ISOC Appl. Netw. Res. Workshop (ANRW)*, Berlin, Germany, Jul. 2016, pp. 38–44.
[147] G. Papastergiou *et al.*, "On the cost of using Happy Eyeballs for transport protocol selection," in *Proc. ACM/IRTF/ISOC Appl. Netw. Res. Workshop (ANRW)*, Berlin, Germany, Jul. 2016, pp. 45–51.
[148] M. Honda, F. Huici, C. Raiciu, J. Araujo, and L. Rizzo, "Rekindling network protocol innovation with user-level stacks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 52–58, 2014.
[149] *Netmap*. Accessed on Nov. 16, 2016. [Online]. Available: http://info.iet.unipi.it/~luigi/netmap/
[150] L. Rizzo and G. Lettieri, "VALE, a switched Ethernet for virtual machines," in *Proc. ACM CoNEXT*, Nice, France, 2012, pp. 61–72.
[151] I. Marinos, R. N. M. Watson, and M. Handley, "Network stack specialization for performance," in *Proc. ACM SIGCOMM Workshop Hot Topics Netw. (HotNets)*, College Park, MD, USA, 2013, pp. 1–7.
[152] Intel. *Data Plane Development Kit*. Accessed on Nov. 16, 2016. [Online]. Available: http://www.dpdk.org
[153] U. A. Camaró and J. Baudy. *PACKET-MMAP*. Accessed on Nov. 16, 2016. [Online]. Available: https://www.kernel.org/doc/Documentation/networking/packet_mmap.txt
[154] P. Kelsey, "Userspace networking with libuinet," presented at the Tech. BSD Conf. (BSDCan), Ottawa, ON, Canada, May 2014. [Online]. Available: https://www.bsdcan.org/2014/schedule/attachments/260_libuinet_bsdcan2014.pdf
[155] J. Dike, *User Mode Linux*, vol. 2. Englewood Cliffs, NJ, USA: Prentice-Hall, 2006.
[156] H. Tazaki, R. Nakamura, and Y. Sekiya, "Library operating system with mainline Linux network stack," presented at the NetDev 0.1, Ottawa, ON, Canada, Feb. 2015. [Online]. Available: https://www.netdev01.org/docs/netdev01-tazaki-libos.pdf
[157] *NUSE*. Accessed on Nov. 16, 2016. [Online]. Available: https://github.com/libos-nuse/linux-libos-tools
[158] S. A. Baset and H. G. Schulzrinne, "An analysis of the Skype peer-to-peer Internet telephony protocol," in *Proc. IEEE INFOCOM*, Barcelona, Spain, Apr. 2006, pp. 1–11.
[159] G. Fairhurst and T. Jones, "Features of the User Datagram Protocol (UDP) and Lightweight UDP (UDP-Lite) transport protocols," Internet Draft, draft-fairhurst-taps-transports-usage-udp, May 2016. [Online]. Available: https://tools.ietf.org/html/draft-fairhurst-taps-transports-usage-udp
[160] M. Welzl, F. Niederbacher, and S. Gjessing, "Beneficial transparent deployment of SCTP: The missing pieces," in *Proc. IEEE GLOBECOM*, Houston, TX, USA, 2011, pp. 1–5.
[161] R. W. Bickhart, "Transparent TCP-to-SCTP translation shim layer," M.S. thesis, Dept. Comput. Sci., Univ. Delaware, Newark, DE, USA, 2005. [Online]. Available: http://www.cis.udel.edu/~amer/PEL/poc/pdf/BickhartMSthesis.pdf
[162] D. Thaler and B. Aboba, "What makes for a successful protocol?" Internet Eng. Task Force, Fremont, CA, USA, RFC 5218 (Informational), Jul. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5218.txt
[163] K.-J. Grinnemo *et al.*, "Towards a flexible Internet transport layer architecture," in *Proc. IEEE LANMAN*, Rome, Italy, Jun. 2016, pp. 1–7.
[164] I. R. Learmonth, B. Trammell, M. Kuhlewind, and G. Fairhurst, "PATHspider: A tool for active measurement of path transparency," in *Proc. ACM/IRTF/ISOC Appl. Netw. Res. Workshop (ANRW)*, Berlin, Germany, 2016, pp. 62–64.

Giorgos Papastergiou received the Diploma in electrical and computer engineering and the Ph.D. degree in computer networks from the Democritus University of Thrace, Xanthi, Greece, in 2005 and 2012, respectively, and the M.Sc. degree in informatics from the Aristotle University of Thessaloniki, Greece. From 2015 to 2016, he was a Postdoctoral Researcher with Simula Research Laboratory, Norway. He is currently an IP Network Engineer with Digea S.A., Greece. He has participated in several EU and ESA funded research projects related to delay-tolerant networking, space internetworking, and the evolution of Internet's transport architecture. His research interests include various aspects of transport protocol design, implementation, and performance evaluation.

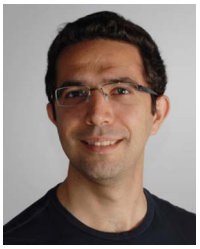
Gorry Fairhurst received the B.Sc. degree in applied physics and electronics from Durham University, Durham, U.K., and the Ph.D. degree in communications engineering from the University of Aberdeen, Aberdeen, U.K. He is currently a Professor with the School of Engineering, University of Aberdeen. His research interests include link protocol design, TCP transport, multicast transport protocols, networking techniques for low latency Internet communication, and performance evaluation of broadband systems. He has worked on a range of Internet projects funded with national, European, and ESA funding, and contributed to the Digital Video Broadcast Project on networking standards for IP transmission over DVB and the HLS for DVB-RCS2. He actively participates in developing networking standards with the Internet Engineering Task Force, where he chairs the Transport and Services Working Group and has served on the IETF Transport Directorate.

David Ros received the B.Sc. (Hons.) and M.Sc. degrees from the Simón Bolívar University, Caracas, Venezuela, both in electronics engineering, and the Ph.D. degree in computer science from the Institut National de Sciences Appliquées, Rennes, France. He is a Senior Research Scientist with Simula Research Laboratory. He was the Co-Chair of the Internet Congestion Control Research Group, Internet Research Task Force, from 2012 to 2015. He has advised or co-advised a dozen M.Sc.-level theses and six completed Ph.D. theses, and has been involved in several European and national funded research projects, as well as in scientific collaboration programs with two Latin American countries. His active research interests include transport-layer issues, congestion control, architectural and quality-of-service issues in IP networks.

Anna Brunstrom received the B.Sc. degree in computer science and mathematics from Pepperdine University, CA, USA, in 1991, and the M.Sc. and Ph.D. degrees in computer science from the College of William and Mary, VA, USA, in 1993 and 1996, respectively. She joined the Department of Computer Science, Karlstad University (KaU), Sweden, in 1996, where she is currently a Full Professor and the Research Manager with the Distributed Systems and Communications Research Group. She has authored/co-authored 10 book chapters and over 100 international journal and conference papers. Her research interests include transport protocol design, techniques for low latency Internet communication, cross-layer interactions, multipath communication, and performance evaluation of mobile broadband systems. She has lead several externally funded research projects within the above mentioned areas and served as the Principal Investigator and the Coordinator with KaU in additional national and international projects. She is currently the KaU Principal Investigator within two EU H2020 projects, the NEAT project aiming to design a new, evolutive API, and transport-layer architecture for the Internet, and the MONROE project proposing to design and operate a European transnational open platform for independent, multihomed, large-scale monitoring, and assessment of mobile broadband performance. She is the Co-Chair of the RTP Media Congestion Avoidance Techniques working group within the IETF.

Karl-Johan Grinnemo received the M.Sc. degree in computer science and engineering from the Linkoping Institute of Technology, Sweden, in 1994, the Ph.D. degree in computer science from Karlstad University, in 2006. He has worked almost 15 years as an Engineer with the telecom industry first at Ericsson and then as a Consultant with Tieto. A large part of his work has been related to Ericsson's signaling system in the mobile core and radio access network. From 2009 to 2010, he was on leave from Tieto and worked as an Acting Associate Professor with the School of Information and Communication Technology, KTH Royal Institute of Technology. From 2010 to 2014, he was an Associate Senior Lecturer with Karlstad University, Sweden, and became a Senior Lecturer in 2014. His research primarily targets application- and transport-level service quality. He has authored and co-authored around 40 conference and journal papers.

Per Hurtig received the M.Sc. and Ph.D. degrees in computer science from Karlstad University, Sweden, in 2006 and 2012, respectively. He is currently an Associate Professor with the Department of Computer Science, Karlstad University. His research interests include transport protocols, low-latency Internet communication, multipath transport, and network emulation. He has participated in several externally funded international research projects, within the above mentioned areas, and also lead a number of national projects. He is also involved in Internet standardization within the IETF.



Naeem Khademi received the bachelor's degree in software engineering from Kish University, Iran, the master's degree in computer networks from University Putra Malaysia, and the Ph.D. degree in computer networks from University of Oslo, Norway, in 2015, where he has been a Post-Doctorate Researcher with the Department of Informatics, University of Oslo, since 2015. His research interests include design, evaluation, and optimization of transport layer protocols and mechanisms in the Internet including congestion control, active queue management, and cross-layer interactions. He has also been involved in two EU-funded projects, Reducing Internet Transport Latency and A New, Evolutive API and Transport-Layer Architecture for the Internet in addition to the standardization activities in the IETF.

Michael Tüxen received the Dipl.Math. and Dr.rer.nat. degrees from the University of Göttingen in 1993 and 1996, respectively, both in mathematics. In 1997, he joined the Systems Engineering Group, ICN WN CS, Siemens AG, Munich. He has been a Professor with the Department of Electrical Engineering and Computer Science, Münster University of Applied Sciences, since 2003. His research interests include innovative transport protocols, especially SCTP, IP-based networks, and highly available systems. At the IETF, he participates in several working groups and co-chairs the TCP Maintenance and Minor Extensions Working Group.

Michael Welzl received the Ph.D. (with distinction) and the habilitation degrees from the University of Darmstadt, Germany, in 2002 and 2007, respectively, and was with the University of Linz and University of Innsbruck, Austria. He has been a Full Professor with the Department of Informatics, University of Oslo, since 2009. His habilitation thesis, the Wiley book *Network Congestion Control: Managing Internet Traffic*, is the only introductory book on network congestion control. He is active in the IETF and IRTF, as a Chair of the Internet Congestion Control Research Group and by leading the effort to form the Transport Services Working Group. He has also been participating in several European research projects, including roles such as a coordinator and a technical manager.

Dragana Damjanovic received the Ph.D. degree in computer science from the University of Innsbruck, Innsbruck, Austria, in 2010. She is a Platform Engineer with the Mozilla Corporation focused on computer network innovations for the Web. She worked as a Research Assistant with the University of Innsbruck with research interests in transport protocols and computer networks in general. She has been involved in several European research projects. She also participates in Internet standardization within the IETF.

Simone Mangiante received the Ph.D. degree in computer networks from the University of Genoa, Italy, in 2013, where he worked on carrier Ethernet management using the SDN paradigm. He is a Senior Research Scientist with Dell EMC Research Europe based, Centre of Excellence, Ireland. His research interests are focused on computer networks, software defined networking, cloud architecture, and Internet of Things. He has been involved in European projects focusing on SDN (FP7 Marie Curie SOLAS) and network transport (H2020 NEAT). He has also led the design and deployment of the Dell EMC INFINITE industrial IoT testbed.