A New Transport Encapsulation for Middlebox Cooperation

Brian Trammell, Mirja Kühlewind, and Elio Gubser
Networked Systems Group
ETH Zurich, Switzerland

Joe Hildebrand
Cisco Systems, Inc.
San Jose, CA, USA

***Abstract*—Large-scale pervasive surveillance of Internet traffic has led to a rapidly expanding deployment of encryption. Network operators increasingly rely on the use of in-network functionality provided by middleboxes to make their networks manageable and economically viable. Emerging applications such as real-time video conferencing need further support that the transport layer, having become ossified between narrow interfaces and an inability to universally deploy new transports or new options, is unable to provide. To address the collision of these three forces, we present in this paper a new encapsulation and signalling protocol, called the Substrate Protocol for User Datagrams (SPUD), that provides a shim layer for middlebox cooperation.**

While the protocol itself must be standardized to allow for successful deployment and cooperation, in this paper we additionally assess research questions on how this new protocol can be used to expose information to middleboxes to enable low latency support for new services. We describe our approach based on a prototype implementation available to the community, and show through a small initial measurement study that the use of UDP as a multiplexing encapsulation for SPUD, which allows for NAT traversal as well as rapid deployment in userspace, is a valid strategy in the present Internet. Finally we discuss open issues and research challenges in SPUD's development as an input for both, the standardization community as well as for academic researchers.

I. Introduction

The Internet's transport layer has ossified, squeezed between narrow interfaces (from BSD sockets to pseudo-transport over Secure Hypertext Transfer Protocol (HTTPS)) and increasing in-network modification of traffic by middleboxes that make assumptions about the protocols running through them. This ossification makes it difficult to innovate in the transport layer at the same time as emerging applications require functionality that existing protocols can provide only inefficiently, if at all.

The increased deployment of encryption by applications and service providers, driven in large part by revelations of large-scale passive surveillance of Internet traffic and metadata in 2013, shows one way to resolve this issue. Middleboxes cannot meddle with what crypto keeps them from seeing; indeed, the widespread use of Transport Layer Security (TLS) already threatens the functionality of a wide variety of Deep Packet Inspection (DPI) devices and application layer gateways. Simply extending this trend, and encapsulating transport headers in Datagram Transport Layer Security (DTLS) [14] would restore the ability to innovate end to end at the transport layer. However, it would simply also not work in the present Internet, breaking network address translators, firewalls, and any number of other middleboxes that make the network more manageable.

We present the Substrate Protocol for User Datagrams (SPUD), a new approach to selective information exposure designed to support transport evolution as described in [19]. Since the widespread deployment of Network Address Translation (NAT) has in effect moved the network-transport layer header boundary to the other side of the source and destination port numbers, and Transmission Control Protocol (TCP) brings a complex set of semantics over which new transports may not necessarily be mappable, the current state of the Internet demands that new transport protocol deployment occur over User Datagram Protocol (UDP). Therefore, SPUD is realized as a shim between UDP and an (encrypted) transport protocol, and provides minimal sub-transport functionality, of use to devices on path:

- Grouping of packets together into *tubes*.
- Signaling of the start and end of a tube, assisting in state setup and teardown along the path.
- An extensible signaling mechanism based on type-value encoding in the Concise Binary Object Representation (CBOR) [1] for associating properties with individual packets or all packets in a tube.

This last feature allows endpoints to declare traffic properties to devices along the path, as well as for path devices to declare properties of the path to the endpoints. In contrast to previous path signaling approaches, SPUD places the following general constraints on the vocabulary of these declarations, in order to support deployability of SPUD and applications based on top of it.

Information exposure is declarative.
: There is no mechanism for property matching, negotiation, or reservation using SPUD. Path elements and endpoints expose properties independently and should not make any assumption about the action that might follow.

All entities may trust but verify.
: Declarations are designed to be verifiable by their receivers. No trust relationship is necessary: the best way to prevent cheating is to remove the incentives to do so.

Information must be incrementally useful.
: Information exchange should not require all nodes on a path to support SPUD to be useful. A sender must always assume its exposed information are

> not understood. As there is no minimal set of mandatory vocabulary defined, an element that does not understand a given declaration may neither remove it nor filter the packet containing it.

These constraints were chosen to address problems with related work in the area. Declarative information exposure without defined or necessarily predictable effects, such that there are weak or nonexistent incentives for either endpoints or middleboxes to make false declarations, should not suffer the same problems with lack of trust as previous signaling mechanisms.

Clearly, standardization is key to the success of any such approach. SPUD is being developed within the Internet Architecture Board's IP Stack Evolution program, with an eventual goal the standardization of a protocol meeting its requirements within the Internet Engineering Task Force (IETF) Transport and Services (TSV) area. It bridges open research questions as to the most efficient way to implement new transport protocols atop UDP and work to define signals between endpoints and devices on path in an untrusted environment such as the Internet with current standardization efforts.

In this paper we present the design of SPUD and discuss its applicability to a specific use case in real-time applications, supporting a low-latency transport service over the Internet. We then answer the most basic question about any new encapsulation proposed for general Internet usage – "will it actually work in the Internet?" – with a simple measurement study. Finally we discuss open issues and research challenges in SPUD's development as an input for both, the standardization community as well as for academic researchers.

A. Related Work

SPUD is by no means the first proposal for application-to-network or network-to-application signaling. Work on Quality of Service (QoS) has led to diverse signaling approaches such as Differentiated Services [10] and NSIS [4]. These proposals have seen very little interdomain or end-to-end deployment, however, due to a lack of trust across network boundaries. More success in signaling in an interdomain context has been seen in the case of NAT traversal, such as the Interactive Connectivity Establishment (ICE) [15] protocol.

As the ability to extend TCP is dependent on network conditions [7], [2], the creation of transport protocols that use UDP as an encapsulation layer is a widespread practice. Examples of special-purpose transport protocols using UDP as include the uTorrent protocol [11], a scavenging transport protocol using the LEDBAT congestion control mechanism [17] and Google's Quick UDP Internet Connections [16]. In both cases the ability to implement in unprivileged userspace is a specific goal of UDP encapsulation: here, UDP provides a user-space raw socket service with port multiplexing. The present work can be seen as an effort to generalize this design pattern in a middlebox-cooperative way.

B. Low Latency Service Differentiation

Most networks in the Internet are presently optimized for high throughput and low loss. In contrast, many emerging services are more latency sensitive and relatively robust to packet loss. For high bit rate services, such as WebRTC [6], the need for the network to support a low latency service is unavoidable. The deployment of these applications inevitable.

Sevice differentiation, offering a new low latency service alongside current "best-effort" service, has long been proposed in the literature [13], [8], [3]. The basic approach is to serve current low loss services (with potentially high delay) out of a separate queue (or at least provide a different treatment) from service offering a small maximum queueing delay (with potentially high loss). Here, there is a clear tradeoff which removes the incentive to lie.

In an unencrypted world, DPI is often applied to traffic classification, and could be used to assign each traffic flow to a service type. This is not possible when the traffic is encrypted. Other approaches for explicit endpoint signaling to select between low-loss and low-latency queues include redefinition of DSCP [10] or implicitly using ECN support as a classifier [9]. In this work we instead propose the use of SPUD, as this is one of the core use cases of endpoint to network signaling for which SPUD is designed.

II. SPUD Prototype

The Substrate Protocol for User Datagrams (SPUD) is a new transport encapsulation built atop UDP which provides three basic facilities. First, packets are grouped into *tubes* – groups of packets identified by a cryptographically random number. Tubes allow transport protocols using SPUD to multiplex multiple streams over a given UDP socket, and to assign different lifetimes and properties to each. Different tubes on the same socket can be used by different applications or different streams within an application. Second, SPUD contains a facility to signal the start and end of a tube, in order to assist NAT devices and other middleboxes along the path with state setup and teardown. Third, arbitrary data can be added to each packet, in order to allow endpoints to make declarations about packets and the tube, and to allow middleboxes to make declarations about the path with respect to a tube.

Tubes are started by an "initiator" expressing an interest in communicating with a "responder", and may be closed by either endpoint. The signaling for path open and close is separate from any state signaling used by the transport protocol SPUD encapsulates, and represents the minimal information about transport stream or session state that must be exposed to middleboxes for state maintenance.

A. Design Principles and Deployment Concept

As noted above, SPUD is designed to support a restricted but extensible vocabulary for endpoint-middlebox signaling which addresses interdomain deployability issues in previous sigaling approaches. This leads to design principles for the protocol itself, the most fundamental of which is that SPUD is *explicitly advisory*. In contrast to other signaling protocols, a future SPUD standard will define no parts of the vocabulary which are mandatory to implement, and define no mandatory state handling at middleboxes. Middleboxes must only associate that state with a tube which benefit their own processing of the packets, and may freely manage state according to their requirements, treating tube open and close as advisory signals.

As on day zero there will be no SPUD-aware routers and middleboxes in the network, SPUD applications will initially use only the tube facility between endpoints for multiplexing user-space transport encapsulation over UDP. We envision the first middleboxes to support SPUD be Network Function Virtualization (NFV) implementations within provider networks, where selective exposure can replace complicated key management infrastructure required to provide in-network functionality in an encrypted environment. Following the deployment of SPUD-based transports at the endpoints, there will be an incentive for other middleboxes to use SPUD signaling in an interdomain context as hints to existing in-network services. We also anticipate the standardization of SPUD (or SPUD-like) protocols to be a driving factor in final adoption of the approach.

The SPUD protocol [5] as we have defined and implemented it to date is intended primarily for experimentation with these mechanisms and vocabularies. Some parts of the experimental protocol are well established, such as the need for UDP-based encapsulation and tube multiplexing. Others are still under active discussion discussion, such as the handling of tube start and end signaling, and the details of the signaling encoding. Discussions are ongoing within the Internet standardization community, Internet Architecture Board (IAB) IP Stack Evolution program. We discuss the design below, by examining the SPUD header and the current implementation.

B. The SPUD Header

Figure 1 shows the SPUD header along with the UDP header encapsulating it.

The magic field is a 4-byte magic number `0xd80000d8` chosen to allow on-path devices to quickly recognize that a packet within a UDP flow is a SPUD packet. The magic number has the property that it is invalid UTF-8, UTF-16, and UTF-32 (both big-and little-endian), and will not appear as the first four bytes of any valid Domain Name Service (DNS), Network Time Protocol (NTP), DTLS, Dynamic Host Configuration Protocol (DHCP), Simple Network Management Protocol (SNMP), Trivial File Transfer Protocol (TFTP), BitTorrent uTP, or netbios-ns datagram, reducing the chance of false positive magic number matches.

The Tube ID is a cryptographically random, 64 bit number scoped to the UDP 5-tuple. Randomness guards against tube ID spoofing, ensuring that possession of the Tube ID indicates with a high probability that an entity has seen a packet associated with that tube: it is an endpoint or a device along the path.

The command field indicates whether a packet opens a tube, closes a tube, or acknowledges an open tube; tube closure need not be acknowledged. These commands generally map to the connection establishment and teardown mechanisms in the upper level transport protocol which SPUD encapsulates, and are exposed as hints to help devices on path to efficiently track the state of the tube.

The A and P flags mark a SPUD message as being an application (endpoint) or a path (middlebox) declaration. Declaration messages are used to bind the properties to a tube ID.

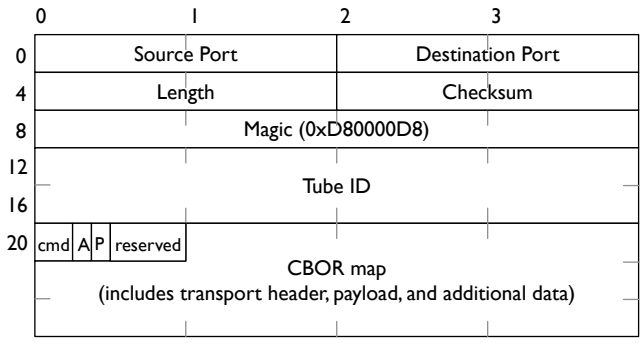

Fig. 1. SPUD header.

The rest of a SPUD datagram is a CBOR [1] map. CBOR provides an efficient generic encoding of lists and maps of primitive datatypes, and was selected as a standard representation for SPUD content due to its ease of implementation in constrained environments. Work is in progress to define a reduced profile of CBOR for use in SPUD which is also efficiently hardware-implementable.

The integer key 0 in the CBOR map identifies the message payload. Other keys in the CBOR map are used for attaching property declarations to a specific packet (if neither A or P bit is set) or associating properties with a tube (if either or both of the A or P bits are set). SPUD can also be used to accumulate and reflect information along both the upstream and downstream paths between two endpoints. If both the A and P bits are set, the receiving endpoint will strip the A flag and reflect the SPUD datagram back to the sender.

C. SPUD Implementation

An initial implementation of SPUD is available as BSD-licensed open-source code on github at https://github.com/iptube/SPUDlib. This code allows an endpoint to manage a number of tubes as either the initiator or responder. The responder listens on a given port number for incoming SPUD messages and the initiator can use the same socket for SPUD messages with different tube IDs. Events are fired for state transitions, data or declarations being received, etc. Eventually, this library will grow to encompass several transport mechanisms encapsulated by SPUD, all using a similar API, as a basis for experimentation.

Current work on the implementation includes integration of existing transport protocols over SPUD, starting with user-space implementations of TCP and Stream Control Transmission Protocol (SCTP) [18].

Figure 2 shows, based on our wireshark SPUD plug-in, a generated example SPUD data packet with one CBOR key/value pair (value=0 for data) and 212 Bytes of dummy data.

III. Signaling for Low Latency

The SPUD protocol can be used to signal low latency requirements from an endhost to the network or expose the existence of support for such services from the network to the host.

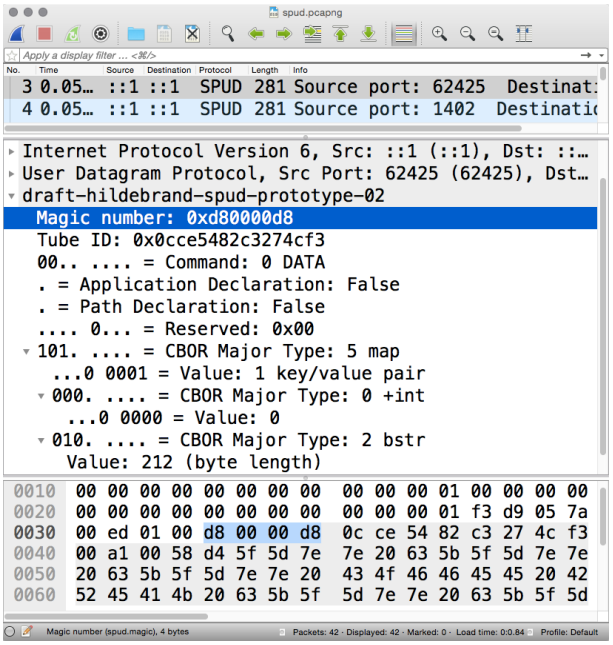

Fig. 2. SPUD data packet in wireshark.

Current approaches to traffic classification often try to detect real-time media traffic and provide them with more appropriate quality of service. However, in the absence of any other information, traffic is treated by the network as being more sensitive to loss than latency. A more latency-sensitive service, implemented through the use of smaller queues and/or different AQM parameters, is often more appropriate for such traffic. Here we propose to provide four SPUD signals: a *latency sensitivity flag*, a signal to *yield to another tube*, an application preference for a *maximum single queue delay*, and a facility to discover the *maximum possible single queue length* along the path.

First, a boolean SPUD application declaration signal can explicitly define a tube as loss or latency sensitive. Applications and endpoints setting the latency sensitivity flag on a tube must be prepared to experience relatively higher loss rates on that tube, and might use techniques such as Forward Error Correction (FEC) to cope with these losses. A middlebox may use this signal to assign packet to the approrpiate queue/service if different services are implemented at this middlebox.

With the second signal, the application can indicate that a given tube is of lower priority than another tube sent by the same endpoint; preferential drop in case of congestion can then be implemented by on-path devices by translating yield instructions into priority queueing or other intradomain signals (e.g. Differentiated Services Codepoint (DSCP)). For interactive conferencing applications, for example, normal video data yields to interframes yields to audio. Future integration of codec and transport technology can use even finer grained more priority levels to provide automatic graceful degradation of service within the network itself.

With the third signal, the application can indicate a maximum acceptable single-hop queueing delay per tube, expressed in milliseconds. SPUD-aware routers can then drop any packet for that tube which would be placed in a queue that has more than the given delay at that point in time, before queue admission, thereby reducing overall congestion. While this mechanism does not guarantee that sent packets will experience less than the requested delay due to queueing delay, it can significantly reduce the amount of traffic uselessly sitting in queues, since at any given instance only a small number of queues along a path (usually only zero or one) will be full.

Finally, SPUD's declaration reflection capability allows a "queue trace" of SPUD-aware routers. Here, the application sends a SPUD "probe", setting the current maximum queue delay to 0; SPUD-aware network nodes will then overwrite the value in the CBOR map if they may impart a higher queueing delay than the currently declared maximum. This information can then be used by the application to estimated the maximum jitter and configure its streaming buffers appropriately.

IV. Feasibility of UDP Encapsulation

A basic assumption made by the current prototype is that UDP encapsulation is necessary, both to enable non-privileged userspace implementations of new transports over SPUD (e.g. in web browsers) as well as to cross firewalls and middleboxes in the first place. However, the configuration of the current Internet may make assumptions about the properties of UDP traffic, subjecting such traffic to different treatment, and many firewalls may be configured to block UDP traffic.

Therefore, we ran a simple measurement comparing connectivity and loss over the same paths using both UDP and TCP to transfer the same data, in order to estimate different packet treatment along these paths. Given that we know from Honda et al [7] that TCP options support is port-dependent, we ran these tests on five UDP ports (80, 443, 1228, 12345, and 54321[1]) to five vantage points (in Amsterdam, London, New York, San Francisco, and Singapore) on a commercial cloud services provider, from ten sites, including:

1) a small/independent residential ISP in Switzerland
2) a large residential cable ISP in Switzerland
3) the 3G network of a major mobile operator in Switzerland
4) the 4G network of a different major mobile operator in Switzerland
5) the campus network at ETH Zurich
6) a residential cable ISP in Germany
7) a campus network at an east-coast US university
8) a campus network at a west-coast US university
9) a major residential cable ISP in the US, and
10) a large enterprise network.

For the most part, these tests gave us unsurprising results: the enterprise network blocked all UDP traffic, presumably as a matter of firewall policy, but otherwise we saw no evidence of blocking of UDP traffic versus TCP traffic based only on IP protocol at any site. We did observe some port-and-protocol-dependent filtering at other sites:

- All udp/80 traffic from our servers in Singapore and San Francisco to all sites was blocked, but not from the other three servers. tcp/80 traffic was not blocked.

[1]Note that we avoided ports on which well-known UDP services run in this selection, because we want to test UDP-specific behavior as opposed to application-specific behavior

- tcp/443 traffic was blocked from one campus network, though udp/443 was not.

UDP and TCP loss rates were uncorrelated. The loss rate in TCP is a function of transient network conditions and the congestion control algorithm, while the UDP test traffic we sent was constant packet rate traffic at or below the mean packet rate achieved by TCP on the same endpoint pair. We noticed only one path of the 45 observed, between site 6 and the Singapore measurement server, that suffered approximately 20% packet loss with UDP but without significant loss in TCP, which may be evidence of a queue favoring TCP on a transiently congested link.

The present SPUD prototype test software only sends bare UDP packets in SPUD format, so there should be no existing middlebox that could understand the SPUD protocol to treat it differently. Indeed, we ran simple connectivity tests using the SPUD prototype software itself, over UDP port 1402, from sites 1 and 4. No connectivity or loss issues were noted.

This basic measurement study provides a sanity check only: on the non-enterprise networks we observed, UDP-based encapsulations would be fundamentally supported, especially on unpopular and unallocated ports. Future work is necessary to get information about more than the fifty paths we tested, and about a more diverse set of network conditions. The wholesale blocking of UDP on the enterprise network is unsurprising; indeed, one of the aims of SPUD is to make it possible to open corporate firewalls to UDP in a scalable manner.

V. Challenges and Future Work

Even though we have confidence that the UDP encapsulation is a valid approach for introducing the SPUD shim layer into the transport stack, we need further measurements to quantify any performance penalties that could be experienced, as well as the risk of non-reachability due to the use of UDP. A much larger measurement study (on the order of 100k paths) using the RIPE Atlas active measurement platform and the PlanetLab testbed is currently underway.

These measurements might indicate the need for more or less elaborate encapsulation and/or fallback schemes to ensure SPUD-based transports are universally deployable. It may also be possible and necessary to realize SPUD over a TCP substrate, as is the case with TCP Minion [12]. However, path probing and fallback mechanisms generally increase start-up latency; simultaneous open, similar to the "Happy Eyeballs" approach to dual-stack IPv6 usage, could be implemented to probe path availability without such a latency penalty. Both generic encapsulation as well as methods and studies on middlebox measurements are current topics in standardization that rely on the input from the research community (e.g see the How Ossified is the Protocol Stack (HOPS) proposed research group within the Internet Research Task Force (IRTF)[2]).

This work is only beginning; large parts of the eventual stack of SPUD-enabled transports remains undefined, including a final API, both for the SPUD layer and for transports above it. The implementation of transports over this substrate, currently underway, may indicate necessary design changes. Integrity protection for part or all of the CBOR map, for example, will be necessary to prevent on-path modification of exposed information, and is not yet a feature of the base prototype. This, in turn, may indicate the need to more tightly bind SPUD to the upper layer protocol providing cryptographic protection, as a source of identity. Finally, minimizing complexity and demonstrating the implementation feasibility of our approach are important preconditions for successful standardization. The process of standardization itself also might influence final design decisions.

Service discovery, encapsulation negotiation, and transport protocol selection are also left to future work. Presently, the assumption is that an application implies a specific transport protocol over SPUD, and that the application will be identified by UDP port number. However, work on dynamic protocol selection schemes such as those under investigation in the IETF Transport Services (TAPS) working group implies that a mechanism for transport protocol selection between endpoints atop SPUD may prove useful.

The set of commands currently defined are intended to map well to present TCP and SCTP handshakes. However, as future transport protocols have low setup latency as a core requirement, it may be necessary to revisit these commands to support zero-RTT connection setup.

Future work on SPUD's binding to CBOR includes defining and standardizing a reduced profile of CBOR for efficient hardware implementation as well as a set of rules for the arrangement of keys in a SPUD CBOR map, to facilitate fast-path processing of SPUD information.

Finally, the restrictions on vocabulary for SPUD declarations will avoid problems with lack of trust among elements, but it is not yet clear that the set of declarations that can still be made under these restrictions are universally applicable to a wide variety of use cases. Defining these vocabularies and experimenting with them is a topic for future research.

With respect to low latency support, the realization of the Active Queue Management (AQM) used in a one or two queue system as well as the respective scheduling algorithm is a broader field of research. As low latency support translates into smaller queues or AQM mechanism with very low thresholds, congestion control should also be further optimized for this use case.

All SPUD use cases, including low latency support, need separate discussion on attack risks and privacy as well as clear deployment incentives before a decision can be made which information should be exposed by SPUD. This is on-going work where further middlebox measurements can bring more insights on middlebox behavior and requirements.

The SPUD prototype defined in [5] and described in this document is intended to help us answer these questions; this design is intended to be replaced with a new protocol as the result of this experiment.

VI. Conclusion and Outlook

In this work we have presented SPUD, an encapsulation shim layer supporting selective exposure of transport- and

[2]https://www.ietf.org/mailman/listinfo/hops

application-layer data for transport evolution, and defined a set of constraints on the vocabulary for declarations that can be made with it in support of incremental deployment of SPUD. While the basic SPUD design is focus on enabling firewall traversal of UDP based traffic, we have further shown how this layer can be used to support a low-latency service for future Internet applications. Therefore we proposed a set of information elements that could be exposed to enable service differentiation as well as preferential drop. Further, we provide an open-source prototype implementation of SPUD to encourage experimentation in research. As a first step on the evaluation of the feasibility of the proposed UDP-based encapsulations we provided a very basic sanity check with respect to reachability. As both TCP and UDP traffic can be blocked in the Internet, a flexible encapsulation will be necessary to make SPUD deployment successful. The intent of this work is to begin an effort to standardize a new signaling and encapsulation protocol for transport stack evolution.

VII. Acknowledgments

This work is partially supported by the European Commission under grant agreement FP7-318627 mPlane; this support does not imply endorsement of the content. Thanks to the participants of the IAB/ISOC workshop on Stack Evolution in a Middlebox Internet (SEMI) and the IETF 92 SPUD BoF session for the discussions leading to several of the concepts in this work.

References

[1] C. Bormann and P. Hoffman. Concise Binary Object Representation (CBOR). RFC 7049, IETF, October 2013.

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

[3] V. Firoiu and X. Zhang. Best Effort Differentiated Services: Trade-off Service Differentiation for Elastic Applications. In *Proc. of the IEEE International Conference on Telecommunications (ICT)*, 2000.

[4] R. Hancock, G. Karagiannis, J. Loughney, and S. V. den Bosch. Next steps in signaling (NSIS): Framework. RFC 4080, IETF, June 2005.

[5] J. Hildebrand and B. Trammell. Substrate protocol for user datagrams (SPUD) prototype. Internet-Draft draft-hildebrand-spud-prototype-03 (work in progress), IETF, March 2015.

[6] C. Holmberg, S. Hakansson, and G. Eriksson. Web real-time communication use cases and requirements. RFC 7478, IETF, March 2015.

[7] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda. Is it still possible to extend TCP? In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, pages 181–194, New York, NY, USA, 2011. ACM.

[8] P. Hurley, J.-Y. Le Boudec, P. Thiran, and M. Kara. ABE: providing a low-delay service within best effort. *IEEE Network*, 15(3):60–69, 2001.

[9] M. Kuhlewind, D. P. Wagner, J. M. R. Espinosa, and B. Briscoe. Using data center TCP (DCTCP) in the Internet. In *Globecom Workshops (GC Wkshps), 2014*, pages 583–588, Dec 2014.

[10] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated services field (DS Field) in the IPv4 and IPv6 headers. RFC 2474, IETF, December 1998.

[11] A. Norberg. utorrent transport protocol. Standards Track (Draft) 29, BEP, October 2012.

[12] M. F. Nowlan, N. Tiwari, J. Iyengar, S. O. Aminy, and B. Fordy. Fitting square pegs through round pipes: Unordered delivery wire-compatible with TCP and TLS. In *USENIX Conf. on Networked Systems Design and Impl. (NSDI)*, 2012.

[13] M. Podlesny and S. Gorinsky. RD network services: differentiation through performance incentives. *SIGCOMM Comput. Commun. Rev.*, 38(4):255–266, Aug. 2008.

[14] E. Rescorla and N. Modadugu. Datagram transport layer security. RFC 4347, IETF, April 2006.

[15] J. Rosenberg. Interactive connectivity establishment (ICE): A protocol for network address translator (NAT) traversal for offer/answer protocols. RFC 5245, IETF, April 2010.

[16] J. Roskind. QUIC (quick udp internet connections) - multiplexed stream transport over udp. Technical report, Google, December 2013.

[17] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind. Low Extra Delay Background Transport (LEDBAT). RFC 6817, IETF, December 2012.

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

[19] B. Trammell and J. Hildebrand. Evolving transport in the Internet. *Internet Computing, IEEE*, 18(5):60–64, Sept 2014.