Computer Networks 173 (2020) 107211

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

On the usability of transport protocols other than TCP: A home gateway and internet path traversal study

Runa Barik[a,*], Michael Welzl[a], Gorry Fairhurst[b], Ahmed Elmokashfi[c], Thomas Dreibholz[c], Stein Gjessing[a]

[a] Institutt for informatikk, University of Oslo, Oslo, Norway
[b] School of Engineering, University of Aberdeen, Aberdeen, Scotland
[c] Simula Metropolitan Centre for Digital Engineering, Simula Research Laboratory, Oslo, Norway

ARTICLE INFO

Keywords:
Protocol testing
SCTP
DCCP
UDP-Lite
NAT
Internet

ABSTRACT

Network APIs are moving towards protocol agility, where applications express their needs but not a static protocol binding, and it is up to the layer below the API to choose a suitable protocol. The IETF Transport Services (TAPS) Working Group is standardizing a protocol-independent transport API and offering guidance to implementers. Apple's recent "Network.framework" is specifically designed to allow such late and dynamic binding of protocols. When the network stack autonomously chooses and configures a protocol, it must first test which protocols are locally available and which work end-to-end ("protocol racing"). For this, it is important to know the set of available options, and which protocols should be tried first: Does it make sense to offer unchecked payload delivery, as with UDP-Lite? Is a UDP-based protocol like QUIC always a better choice, or should native SCTP be tried? This paper develops answers to such questions via (i) a NAT study in a local testbed, (ii) bidirectional Internet tests, (iii) a large scale Internet measurement campaign. The examined protocols are: SCTP, DCCP, UDP-Lite, UDP with a zero checksum and three different UDP encapsulations.

1. Introduction

Latency has become the dominant metric that most application developers and network designers strive to optimize today. In some cases this has made TCP less suitable, even when data must be reliably transferred. For example, multiplexing application streams over TCP with HTTP/2 can produce head-of-line blocking delay; this is one of several issues that the "Quick UDP Internet Connections" (QUIC) protocol addresses [1]. BitTorrent's µTP (with LEDBAT congestion control [2]) is another example of a protocol that was developed to bypass a shortcoming of TCP: its inability to offer a lower-than-best-effort (LBE) service that would suit a bulk data transfer, while limiting the delay incurred on other traffic. Generally, applications now have much more diverse needs than TCP can support.

To provide a broader service set to applications while facilitating the use of non-TCP protocols, the IETF Transport Services (TAPS) Working Group defines an API that eliminates the static compile-time binding between applications and transport protocols [3]. This enables a network stack to *try* one mechanism or protocol but fall back to a "safe" default protocol (usually TCP or UDP) in case of failure. Such opportunistic ways of using protocols and protocol mechanisms are increasingly common; they can be seen, for example, in iOS devices, in the form of "Happy-Eyeballing" (also called "racing") between IPv6 and IPv4 [4]. The Chrome browser falls back from QUIC (over UDP) to TCP if the connection setup fails with QUIC, and WebRTC allows to opportunistically set the DiffServ Code Points (DSCP) [5]. Here, measurement studies have shown that such DSCP usage is generally not harmful, but it is recommendable to fall back to DSCP zero in case of permanent failure ("black-holing") [6,7]. Rather than hard-coding a specific racing method in each application, a TAPS transport system uniformly takes care of protocol racing and configuration below the API. The NEAT library [8] is a prototypical implementation of a fully-fledged TAPS system that eliminates the static binding between applications and transport protocols or network mechanisms (e.g., DSCP value).

To support this increasing protocol agility of network stacks, this paper investigates how well protocols other than TCP could work across Internet paths. Specifically, our focus is on the native transport protocols SCTP, DCCP and UDP-Lite, as it is frequently assumed that these protocols do not work (e.g., in IETF conversations), while, to the best of our knowledge, no measurement study exists that could either confirm or reject this claim. We would like to understand whether a TAPS transport system should try to make use of UDP-Lite or DCCP at all; whether,

[*] Corresponding author.
E-mail addresses: runabk@ifi.uio.no (R. Barik), michawe@ifi.uio.no (M. Welzl), gorry@erg.abdn.ac.uk (G. Fairhurst), ahmed@simula.no (A. Elmokashfi), dreibh@simula.no (T. Dreibholz), steing@ifi.uio.no (S. Gjessing).

https://doi.org/10.1016/j.comnet.2020.107211
Received 31 October 2019; Received in revised form 10 January 2020; Accepted 11 March 2020
Available online 12 March 2020
1389-1286/

in case UDP-Lite does not work, UDP with a zero checksum would be a viable alternative; whether native or UDP-encapsulated SCTP should be tried first, or whether SCTP is hopeless anyway, making protocols such as QUIC or RTMFP that run over UDP without requiring a specific port number strictly preferable even if an application does not need all of their new features.

In the next section, we begin with an investigation of the potentially most troublesome part of an end-to-end Internet path: home gateways with their common NAT functionality. As we will see, there is only very little explicit support for these protocols in the tested off-the-shelf equipment as well as the Linux and FreeBSD Operating Systems. NAT software could be upgraded over time if these protocols are found to be useful, but this first requires a significant level of deployment and positive experience with them—which, in turn, may be prevented by current NAT behavior. It is therefore important to assess how these protocols would operate through *current* NAT software.

After assessing how SCTP, DCCP, UDP-Lite and UDP with a zero checksum (which might be a possible alternative to UDP-Lite) operate over some NAT devices and the Linux and FreeBSD Operating Systems in a local testbed, Section 3 presents results from bidirectional Internet tests where we see NAT behaviors "in the wild". In Section 4, we take a further look at Internet path traversal with a large-scale one-sided Internet measurement campaign using a customized version of Tracebox [9]. The latter study, while unable to traverse middleboxes such as NATs, provides insights into the drop-or-forward behavior of ASes. We discuss related work in Section 5, and Section 6 concludes.

2. NAT interference: local tests

We begin this section with a brief description of NAT-relevant aspects of the tested transport protocols; Appendix A gives a brief overview of the general operation of NATs and Network Address and Port Translators (NAPTs).

SCTP uses port numbers just like TCP and UDP. However, SCTP is multi-homed, allowing connections between different IP addresses but using the same port number pair to jointly be controlled as part of an "association". Adding and removing connections is done using an "ASCONF" message. Each SCTP packet contains a so-called "verification tag" (Vtag), which identifies connections within an SCTP association [10, Section 8.5]. The Vtag is negotiated during connection establishment with "INIT" and "INIT_ACK" messages. By making connections fully identifiable, the Vtag eliminates the need for a pseudo header. The SCTP checksum is thus not affected by IP layer changes.

DCCP has the same port number and checksum fields as TCP or UDP, but incorporates a partial integrity check as in UDP-Lite (which we discuss below). DCCP is unreliable yet connection-oriented. "DCCP-Request" and "DCCP-Response" packets are used to establish a connection, and connection termination is done with "DCCP-CloseReq", "DCCP-Close" and "DCCP-Reset" packets. The corresponding NAPT procedures involve port number changes as well as checksum updates [11]. Like for TCP and UDP, the checksum is calculated over a pseudo-header including the IP addresses. Since NAPT devices with DCCP support are very uncommon, UDP encapsulation for DCCP has been defined [12]. In this case, a common UDP NAPT can handle encapsulated DCCP traffic.

While the **UDP** checksum covers the whole UDP packet (plus pseudo header), or nothing in case of a zero checksum, **UDP-Lite** provides a configurable checksum coverage. The UDP-Lite header and the pseudo header are always covered by the checksum. Like UDP, UDP-Lite uses port numbers; a NAPT therefore needs to recompute the checksum.

2.1. Test setup

To understand how NATs handle SCTP, DCCP and UDP-Lite, we set up a local testbed in which we studied a number of off-the-shelf gateways as well as a host configured to forward packets, running either Linux or FreeBSD. Our testbed consists of a multi-homed Linux client and a multi-homed Linux server, interconnected via a NAT as shown in Fig. 1. Being multi-homed on both sides lets us emulate situations where multiple physical clients behind the same NAT communicate with one or multiple servers. In addition, this set up allows us to test SCTP multi-homing. Except for the SCTP multi-homing tests, for which we ran a specific SCTP test program[1], the client and the server ran the *fling* tool. *fling* [13], which is available from http://fling-frontend.nntb.no, allows to test whether an arbitrary sequence of packets can be exchanged between a client and a server using pre-defined pcap and json files. This makes it easy to exchange sequences of packets that are crafted for a particular measurement purpose. We will describe *fling* in more detail in Section 3, where it is used for Internet tests.

To assess how a NAT handles several end-to-end transport flows for UDP-Lite, SCTP and DCCP, we ran the following tests for each of these protocols:

a) A client with a single interface, communicating with a single-interface server listening on port 443;
b) Two clients communicating at the same time with a single-interface server listening on port 443;
c) Two clients communicating at the same time with two servers (each listening on a different interface, on port 443).

For reference, Table 1 summarizes all NAT observations that we made, both in local tests and Internet tests. In the following, we will refer to these observations by their number.

2.2. Off-the-shelf equipment tests

Table 2 lists the NAT boxes used in our first set of tests. We reset each device and used their existing and the latest versions of firmwares in our measurements. We also installed OpenWRT, DD-WRT and Tomato Linux based firmware distributions on some of these devices.

Every device consistently showed the same behavior irrespective of the transport protocol. We wanted to better understand if this uniform per-device behavior was specific to the protocol, or if it is generic when a protocol is unknown. To this end, we also crafted a test where we transmitted packets with random IP payload, using the unassigned IP protocol number 143.

In test-case (a), we observed that except *dl7, js1, js3, tp1, tp2, ng1, ng2, ng3* and *ls6*, all boxes perform NAT (but *not* NAPTing!) for all the tested transport protocols. All transport protocols, except SCTP use a pseudo-header in their checksum calculation and hence need their header checksum to be updated by the NAT. However, our NATs did not update this checksum, which means that they will cause DCCP or UDP-Lite packets to be dropped at the receiver. Because the SCTP checksum does not include a pseudo-header, at this point, we do not know whether the NATs explicitly support SCTP. The NATs *dl7, js1, js3, tp1*, and *tp2* do not provide address translation for these protocols, i.e. they forward packets with their private addresses. The NATs *ng1, ng2, ng3* and *ls6* drop packets from any of these transport protocols. These were manufactured by different vendors, but all were based on the VXWorks OS.

In test-case (b), we observed a similar result to test-case (a), but for all tested protocols, only the first client was able to successfully use the NAT. The failure of the second client indicates IP-level NAT'ing, in which case the second client's arrival causes a collision with an already existing entry in the NAT table. Finally, test-case (c) failed for DCCP and UDP-Lite, but both tests succeeded for SCTP for all cases of IP-level NATing (observation 2 in Table 2). This means that multiple clients behind a NAT could use SCTP when the clients communicate with different remote addresses.

In principle, the functionality of UDP-Lite could be emulated by using UDP with a zero checksum, effectively disabling the checksum calculation in accordance with the standard. Checksums over a subset of

[1] https://www.petanode.com/blog/posts/sctp-multi-homing-in-linux.html.

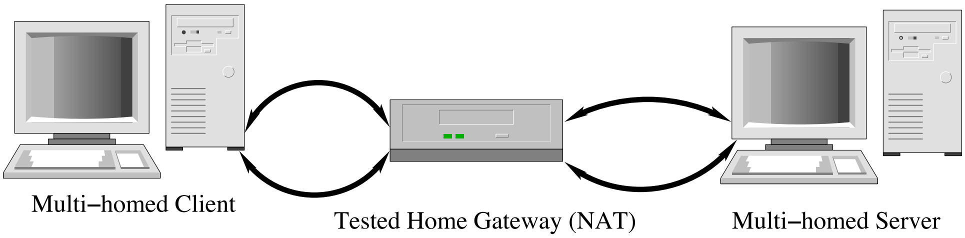


Fig. 1. Our local testbed for NAT analysis.

Table 1
NAT observations.

Obs.	Description
1	NA(P)Ting, a UDP zero checksum remains unchanged
2	IP layer NAT'ing, i.e. no transport header update
3	Forwarding of packets without NA(P)T'ing
4	Dropping
5	NAPTing, which updates the transport header checksum, but a wrong checksum remains wrong
6	Correct NAPTing, which updates the transport header checksum for protocols using a pseudo-header

Table 2
26 home gateway devices and the observed behavior with SCTP, DCCP, UDP-Lite and UDP with a zero checksum (the table contains 31 lines to account for tested firmware updates). For UDP with a zero checksum, in all these tests, we saw observation 1: NA(P)Ting with a zero checksum. * means no idle-timeout due to Obs-3 and Obs-4.

Vendor	Model	Firmware	OS	Tag	Obs.	timeout (s)
Dlink	DIR 655 (A3)	v1.10EU	Linux	*dl1*	Obs-2	> 1200
		v1.37NA	Linux	*dl2*	Obs-2	> 1200
	DIR 655 (A2)	v1.35EU	Linux	*dl3*	Obs-2	> 1200
	DIR 655 (A4)	v1.31EU	Linux	*dl4*	Obs-2	> 1200
	DIR 655 (B1)	v2.09	Linux	*dl5*	Obs-2	50
	DIR 619 (Ax)	v1.00	Linux	*dl6*	Obs-2	90
	DI-614+ (B2)	v3.44	ThreadX	*dl7*	Obs-3	*
Jensen	AL WBR 7954 v3	2.11.3	Unknown	*js1*	Obs-3	*
	AL 1000Gv2 (A)	v1.16	Linux	*js2*	Obs-2	90
	AL WBR 7954 v2	v3.1.0	Unknown	*js3*	Obs-3	*
Linksys	E2500	v1.0.07	Linux	*ls1*	Obs-2	600
		v2.0.00	Linux	*ls2*	Obs-2	600
	WRT54G/	v24-sp2(ddwrt)	Linux	*ls3*	Obs-2	600
	GL/GS	v4.30.18	Linux	*ls4*	Obs-2	600
	v1.1	v1.28(tomato)	Linux	*ls5*	Obs-2	600
	WRT54G	v7	VxWorks	*ls6*	Obs-4	*
	E4200	v2	Linux	*ls7*	Obs-2	600
Netgear	WGR 614v7	v1.0.13	VxWorks	*ng1*	Obs-4	*
		v2.0.30		*ng2*	Obs-4	*
	WGR 614v9	v1.2.30	VxWorks	*ng3*	Obs-4	*
	WNDR3400	v1.0.0.38	Linux	*ng4*	Obs-2	600
Topcom	WBR 254G	v1.3.1e	Unknown	*tp1*	Obs-3	*
	BR 604	v1.10		*tp2*	Obs-3	*
TP-LINK	TL-MR3020 v1	3.17.2 Build	Linux	*tl1*	Obs-2	600
		OpenWrt	Linux	*tl2*	Obs-2	600
	TL-WR703N	OpenWrt	Linux	*tl3*	Obs-2	600
3G modem	WR3G050-02	v4.34	Linux	*3g*	Obs-2	600
ZyXEL	P8702N	1.00(AAJX.14)	Linux	*zy1*	Obs-2	600
	P-2812HNU-F3	V3.11 (BLN.21)	Linux	*zy2*	Obs-2	600
Edimax	BR-6574N (A)	v1.24	Linux	*em1*	Obs-2	120
Xiaomi	Router 3C	v2.14.37	OpenWRT MiWiFi	xi1	Obs-2	600

the data could be implemented in the payload; in the case of IPv6, this would have to include additional mechanisms and/or restrictions, e.g. to prevent leakage of traffic from one UDP application to another [14]. NATs should not modify a zero checksum of the UDP header [15]. To check whether our NATs operate in line with this rule, we ran UDP tests with a correct checksum and a zero checksum, and found that the NAT boxes indeed kept the zero checksum intact, and communication worked flawlessly in all three test-cases.

Table 2 summarizes our findings in the controlled measurements, using the observation numbers from Table 1. There is only one observation number per line because the behavior was uniform for all protocols. None of the NAT devices fully support SCTP, DCCP or UDP-Lite (i.e. perform transport-layer functions such as port translation or, in the case of SCTP, correct mapping of connections to a table based on the Vtag). With all devices, the behavior appears generic, because it also matched the behavior observed when using an unassigned protocol number (143).

For IP layer NATing (observation 2), the chance a host suceeds to use a protocol also depends on the idle-timeout—the time during which the NAT retains a mapping after seeing a packet with an IP address. To determine the value of this timeout, we conducted an experiment where a multi-homed client (with two IP addresses) tries to establish an SCTP connection to a server. First, it sends an SCTP INIT packet using its first

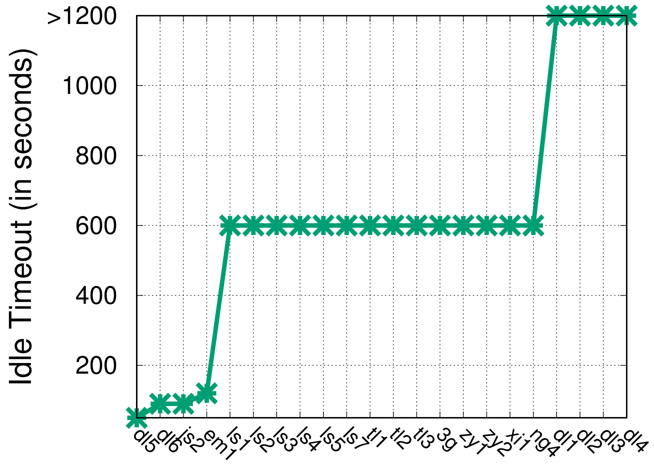


Fig. 2. Idle-timeout of NAT devices that carry out IP layer NAT'ing, prohibiting later connections from other local hosts during this interval.

source IP address. In response, it receives an ABORT message from the server (because there was no application listening to SCTP there). Then, once per second, the client sends an SCTP INIT packet using its second source IP address, until it receives an ABORT message. The difference between the two ABORT messages is the upper bound of idle-timeout, shown in Fig. 2.

Only four devices (from vendors Dlink, Jensen and Edimax) showed an idle-timeout of less than two minutes. All the IP-level-NAT'ing devices from Linksys have an idle-timeout of ten minutes, which is the default value in their Linux kernel. We also notice that most Dlink NAT devices set a higher timeout value (greater than 1200 s). In the case of device *ng4*, during this timeout interval, response packets for any attempt by later clients are wrongly delivered to the first client, causing an update of the timer for the first client.

Table 2 shows that many of the tested NAT boxes use a Linux kernel. To better understand how manually enabling support for these protocols could play out in a NAT box, we now take a closer look at the two open source OSes at our disposal: Linux and FreeBSD.

2.3. Linux

We installed our own customized Linux kernel (version 3.18.109 for MIPS architecture) in the TP-Link TL-MR3020 NAT box, using OpenWRT. The NAT needs two modules: CONNTRACK (connection tracking) for creating a NAT entry, and NAT, which is responsible for setting a new port and performing the checksum update. When a NAT box receives a packet from its local network, then, if the CONNTRACK module supports the protocol, it knows the packet formats and tracks the state of the connection. In this case, CONNTRACK first verifies the packet based on a policy (e.g., correctness of checksum or some field value) and creates a NAT entry. Then, if the NAT also supports NAT for that protocol, it can provide the port-mapping and checksum update. In the following, we examine the effects of various combinations of support for a protocol *X* (SCTP, DCCP or UDP-Lite) by the CONNTRACK_PROTO_X and NAT_PROTO_X modules; we will simply refer to them as CONNTRACK and NAT. The results are summarized in Table 3; this table includes various cases with artificially constructed errors because they will be useful later, in Section 3.

2.3.1. ***No* conntrack *and no* nat**

In all tests (SCTP, DCCP, UDP-Lite and an unknown protocol number), the **NAT treated protocol *X*as unknown and created a NAT entry, based only on the IP layer (observation 2).** Thus, for packets of protocol *X*, Linux performs NAT'ing for the first IP address. Packets arriving later from a different local address to the same server are dropped while the first client is active and during a following idle-timeout interval of 10 minutes; this is consistent with RFC 4787[16], which recommends a default value of five minutes or more. We obtained this idle-timeout value from the source code.

2.3.2. **conntrack *but no* nat**

Enabling CONNTRACK means that the NAT knows the transport protocol format and can create a NAPT entry based on the transport protocol. When the protocol is not supported *X* in NAT no port-mapping will be created and the checksum will not be updated. A collision of ports can then result in packet drop. Because only IP-level NATing works, this behavior is also categorized as observation 2. CONNTRACK verifies the packet's transport header before creating a NAT entry. If the verification fails, no NAT entry is created and no NAT mapping is set up (the packet will be forwarded with a private address). **NAPT entries are created for correct initial UDP-Lite, SCTP and DCCP packets.** Indicating support for these protocols by the CONNTRACK module, no NAPT (or even NAT!) entries were created in the following erroneous cases:

- UDP-Lite: invalid checksum coverage, zero checksum (not permitted [17]), or an invalid checksum (but having the *nf_conntrack_checksum* `sysctl` variable set).
- SCTP: non-zero Vtag ([10] requires INIT packets to contain a zero Vtag).
- DCCP: invalid checksum (but having the *nf_conntrack_checksum* `sysctl` variable set).

As expected, when the protocol was unknown, NAT'ing was the same as without CONNTRACK, i.e., based on only the IP layer.

2.3.3. CONNTRACK and NAT

For correct initial packets from all the tested transport protocols, the **NAT correctly updates the port and checksum values.** As before, for unknown protocols, it performs NAT'ing at the IP layer. The NAT appears to track the state of DCCP: unexpected incoming packets are dropped and unexpected outgoing packets are forwarded without NAT'ing. For example, after the NAT forwards a DCCP-Reset packet from the server to the client, it does not allow any further packets from the server to the client except for another DCCP-Reset packet. Similarly, in another case, the client sends a request packet and the NAT maintains the state. Until it sees the response packet in the reverse direction, no further packets are NAT'ed, except for another request packet from the client, e.g., an ACK packet sent at this point, was forwarded without NAT'ing.

With SCTP, the NAT translates the ports and recomputes the checksum. This port translation is, however, not uniformly applied across an entire association. This is a problem for SCTP multi-homing [18–21].

2.4. FreeBSD

We installed FreeBSD 11.2 in a x86_64 PC to make a NAT and tested IPF, PF and IPFW (the three different firewall variants of FreeBSD [22, Chapter 30]). For PF, we used the FreeBSD distribution PFSENSE[2] separately to study the NAT behavior. Table 4 presents the behavior of transport protocols with IPFW, IPF and PF.

IPF: SCTP, DCCP and UDP-Lite are considered as unknown protocols by IPF [22, Section 30.5]. As with Linux, the NAT only performs IP layer NAT'ing, serving the first client and dropping packets from other clients when there is a collision (observation 4), detected within an idle-timeout of 65 s. This short interval goes against the "five minutes or more" recommendation in RFC 4787 [16], but is explicitly allowed for destination ports in the range 0–1023; thus, this behavior is also in line with this RFC. After the 65 s interval, the next communication attempt succeeds, creating a new NAT entry and starting a new 65 second interval.

[2] PFSENSE: https://www.pfsense.org/.

Table 3
Linux behavior of UDP, DCCP, SCTP and UDP-Lite, with various artificially created errors; cases of correct protocol usage are shown in boldface text. *nf_conntrack_checksum* is a `sysctl` variable that enables/disables validating the checksum.

CONNTRACK	nf_conntrack_checksum	NAT	Test description	Obs.
✗	-	✗	**All tests; also: a packet with unknown proto. number in IP header**	Obs-2
√	✗	✗	**All tests**	
	√	✗	**All tests** except (UDP, UDP-Lite, DCCP) with invalid checksum	
	✗	√	(UDP, DCCP) with invalid checksum UDP-Lite with valid checksum coverage and invalid checksum	Obs-5
	√	✗/√	(UDP, UDP-Lite, DCCP) with invalid checksum	Obs-3
	✗/√	✗/√	SCTP with non-zero Vtag UDP-Lite with invalid checksum coverage or zero-checksum	
	✗/√	√	**(UDP, UDP-Lite, DCCP) with correct checksum; SCTP with zero Vtag**	Obs-6
			UDP with zero checksum	Obs-1

Table 4
Behavior of transport protocols across FreeBSD NAT firewalls for first / later clients (*: all response packets are forwarded to the last client).

Protocols	IPFW	PF	IPF
UDP, zero checksum	Obs-1 / Obs-1	Obs-1 / Obs-1	Obs-1 / Obs-1
DCCP, UDP-Lite	Obs-2 / Obs-2*	Obs-2 / Obs-4	Obs-2 / Obs-3
SCTP	Obs-6 / Obs-6	Obs-2 / Obs-4	Obs-2 / Obs-3

PF: Similar to IPF, PF [22, Section 30.3] considers SCTP, DCCP and UDP-Lite as unknown protocols. Again, the NAT only serves the first client. However, after a collision it forwards the packets with their private addresses for later clients, with an idle-timeout of 30 s (observation 3).

IPFW: For IPFW [22, Section 30.4], we loaded the *ipfw_nat* module, which loads the *libalias* (the in-kernel NAT support) module. The *libalias* module implements the code that supports specific transport protocols. We observed that only SCTP support[3] is available, and enabled by default. Because IPFW is the only FreeBSD NAT module that explicitly supports SCTP, we further tested SCTP's multi-homing capabilities with it (as we mentioned before, SCTP support in Linux re-writes port numbers per connection rather than association, which breaks SCTP's multi-homing).

2.4.1. Multi-homing with IPFW

We first set up one multi-homed client and one multi-homed server with no NAT in between to confirm that the end-hosts successfully use SCTP multi-homing. We refrained from testing multi-homed clients through a single NAT because such a scenario requires the NAT to use two public IP addresses [20]. Instead, we added a second NAT box with the same FreeBSD setup running IPFW, such that each of the two connections of the SCTP association goes through a different NAT. The client is connected to one NAT called NAT A via address A1, and to another NAT called NAT B via address B1. The server is similarly connected to each NAT.

The sequence diagrams in Fig. 3 show the client uses IP address A1 to create the first connection of the SCTP association to the server through NAT A, with a 4-way handshake. It then attempts to set up the second connection of the same association with its second IP address, B1, via NAT B, to investigate the SCTP multi-homing support in the presence of NATs. The leftmost sequence diagram shows that the current implementation of SCTP in the Linux client sends the ASCONF message to NAT A instead of NAT B. IPFW/*libalias* in NAT A forwards the packet without verifying the IP address, B1, placed inside the ASCONF message. However, the response packet, ASCONF_ACK, will never reach the client because address B1 is a *private* address. We observed that the first packet reaching NAT B is a HEARTBEAT message. NAT B directly responds to the client with a *missing state error cause* message, using the Vtag and the destination address (as the source address) from the HEARTBEAT message, failing multi-homing support.

To support SCTP multi-homing, the most recent IETF proposal [21] recommends that the client should add each IP address using ASCONF chunks sent via their respective NATs and wildcard addresses to be filled in the address fields in the ASCONF message (see the middle sequence diagram in Fig. 3). However, this requires an update to the existing NAT code. Moreover the SCTP implementation in the client does not allow it to send ASCONF(0) via NAT B. Using *fling*, we tried ASCONF(0) on the second NAT, but then received a *missing state error cause* message.

We explored how to perform SCTP multi-homing with the existing IPFW NAT code. Instead of the ASCONF message, another INIT message could be sent via the second path after the first 4-way handshake (this idea deviates from the SCTP specification). There are two ways to make the new path a part of the existing association and allow the server to validate it. The first is the client sending the internal Vtag. The second is to use a value from an arithmetic operation (e.g. XOR) of internal and external Vtags, allowing the server to validate the connection as part of the existing association. This method (see Fig. 3) would require changes to both the SCTP client and server. Using *fling*, we found our approach to work through the IPFW/*libalias* NAT.

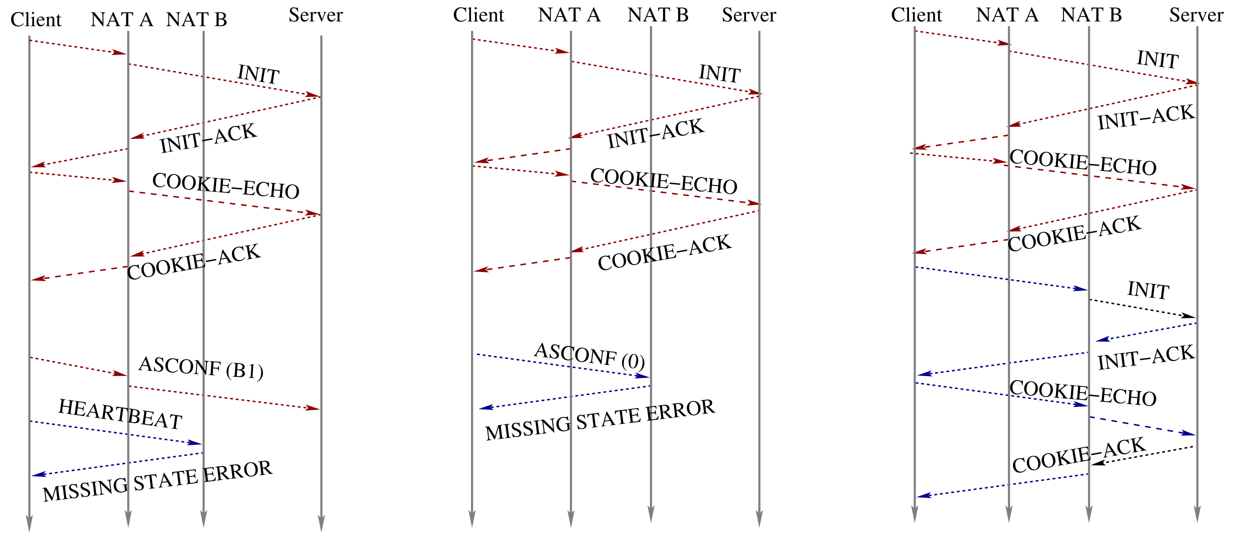
As with the other FreeBSD firewalls, support for DCCP and UDP-Lite is not implemented in IPFW. Thus, packets from these protocols are treated as unknown by the NAT. Similar to Linux, for unknown protocols, IPFW acts like a NAT (not NAPT!) for all outgoing packets from all clients behind it. The IP layer NAT state is kept for an idle-timeout of 60 seconds. However, this interval does not play a role because the NAT does not correctly treat incoming traffic (it always sends all DCCP or UDP-Lite responses to the last active client). We confirmed this behavior by checking the FreeBSD kernel code. This is different from IPF, PFSENSE/PF and Linux, where later communication attempts during the idle-timeout interval deterministically failed.

2.5. Discussion

Before turning to Internet measurements, we briefly summarize our key findings from local tests with home gateways:

- Neither DCCP nor UDP-Lite are supported by any of our studied NAT boxes—packets from these protocols are NATed at the IP layer only (which then causes a failure of the transport checksum), forwarded without NATing, or dropped.
- All the measured NAT boxes, as well as Linux and FreeBSD correctly work with UDP, even when the checksum is zero. Single-homed SCTP often works only because of a common default behaviour to apply IP-level NAT'ing in the face of an unknown protocol.
- For IP-level NAT'ing, Linux (which appears to be the OS used by most commercial middleboxes) and FreeBSD with IPF and PF correctly handle (i.e. reject) a second connection request when the first connection is ongoing, and within an ensuing idle-timeout. This idle-timeout ranges between 30 seconds (FreeBSD PF) and 10 minutes

[3] SCTP Over NAT Adaptation (SONATA) [23]: http://caia.swin.edu.au/urp/sonata/downloads/INSTALL.txt.



Linux end systems with
IPFW/libalias NAT

Proposed method in Internet–draft
with IPFW/libalias NAT

Our proposed method with
IPFW/libalias NAT

Fig. 3. Sequence diagrams showing SCTP multi-homing support, using Linux end systems and a FreeBSD IPFW/*libalias* NAT.

(Linux); the latter value was a particularly common choice in our tested NAT boxes in Table 2 as well.

Altogether, our local tests give us **some hope about the occasional usefulness of SCTP as well as UDP with a zero checksum—and little hope about other protocols** whenever a NAT with default configuration is in use. **This rules out DCCP and UDP-Lite from the list of recommendable choices for protocol racing when a NAT is in use.** This does not provide any information about whether these protocols pass through a real Internet path, in particular using IPv6, where our NAT results may not matter. Also, we have not yet considered UDP encapsulations, because UDP generally works through the tested devices.

We also tested the multi-homing code in the FreeBSD IPFW/*libalias* implementation. We found that this works with a slightly altered end system behavior, where an INIT (instead of ASCONF) message is sent on the second connection of the association. A Linux NAT, however, can also change SCTP port numbers, which breaks a fundamental SCTP assumption and will make multi-homing fail. In an additional test, we confirmed that multi-homing (following [21], which avoids inserting an address into the SCTP chunks) also works for one client with both Linux and FreeBSD (with IPF or PF). This is where SCTP is treated as an unknown protocol and the NAT only operates at the IP layer. Thus, while turning on SCTP support in the NAT can enable multiple clients to talk to the same server at the same time, with the current Linux and FreeBSD (IPFW/*libalias*) code base, it comes at the disadvantage of harming SCTP's multi-homing support.

At this point, we note that these conclusions are derived from one-sided connection establishment attempts because our study does not consider NAT traversal techniques, such as Interactive Connectivity Establishment (ICE) [24] that are common for peer-to-peer communication, e.g. for WebRTC.

3. Bidirectional internet tests

We used our generic client-server reachability test tool *fling* to better understand how the tested protocols and UDP encapsulations operate using Internet paths. A *fling* test specification consists of a json file and a pcap file. These files are uploaded to *fling* servers. *fling* clients are preconfigured to regularly pull new tests from *fling* servers via HTTPS/TCP and run the tests. Each json file describes a communication sequence (using packets in the pcap file), which is repeated three times in the case of a failure. These repetitions enabled us to detect "random" misbehavior that could occur sporadically; for instance, a drop that was caused by a temporary outage, or was due to congestion. We consider a test to "fail" if any packet of the test was consistently dropped in all three iterations. When a test is finished (or failed), paths are probed from both sides, similar to Tracebox [9], and the final client test result is submitted to the server via the HTTPS/TCP signaling channel. A detailed description of *fling* is given in [13].

We hosted *fling* servers on 36 IPv4 and 19 IPv6 nodes. All of them except two IPv4 nodes, belong to a subset of the NorNet[4] testbed called NorNet Core, covering 4 countries (v4/v6: 25/15 Norway, 4/2 Germany, 3/1 China, 2/1 USA). The remaining two IPv4 servers were in India and the USA. We ran the *fling* client tool from 186 IPv4 / 69 IPv6 Ark[5], PlanetLab[6] and NorNet Core nodes. 138 of the IPv4 addresses are public and 48 addresses (14 PlanetLab and 34 Ark) are behind NAT boxes. 114 of the clients (114 IPv4 and 50 IPv6 addresses) belong to CAIDA's Ark platform. These nodes, located in people's homes, universities and offices, are spread across 6 geographic regions (v4/v6: 46/20 North America, 33/15 Europe, 13/3 Africa, 9/6 Asia, 7/5 Oceania, 6/1 South America). 36 IPv4 nodes (located in universities) belong to PlanetLab. With all clients connecting to all servers, our measurements, which were run in May 2017, covered a total of 6696 (IPv4)/1311 (IPv6) bidirectional paths. In all tests, *fling*'s HTTPS/TCP based signaling channel worked flawlessly, ensuring connectivity on all paths.

We tested SCTP, DCCP, UDP-Lite and UDP, using communication sequences that emulate client-server connection attempts. For SCTP, our tests perform association establishment. In the DCCP test, the client sends a DCCP *Request* packet, and the server answers with a *Response* packet; the client then answers with an *Ack* and a *DataAck* packet containing 256 bytes of data, and the server responds with an *Ack*. We designed two types of UDP-Lite tests: in test 1, we check whether the protocol "works" through the path, by sending a UDP-Lite packet with *checksum coverage* = 0, containing 12 bytes of data. In test 2, we try to deduce if UDP-Lite works due to a lucky accident or genuine support of the protocol: the client sends a UDP-Lite packet with an illegal *checksum coverage* (7, which is less than the allowed minimum of 8). In both tests, the server responds with a similar UDP-Lite packet (but flipped ports).

[4] NorNet: https://www.nntb.no.
[5] Ark: https://www.caida.org/projects/ark/.
[6] PlanetLab: https://www.planet-lab.org.

Table 5
Observed transport protocol pathologies across 48 NATs on Internet paths (*: The remaining 4 NATs randomly switched between computing the full checksum and keeping it unmodified).

Protocol number with transport header chksum	Obs-1	Obs-2	Obs-3 / Obs-4	Obs-5	Obs-6
UDP (all UDP encapsulations)	–	–	–	–	48
UDP with zero chksum	44^*	–	–	–	–
SCTP	N/A	31	17	–	–
DCCP with correct chksum	N/A	26	17	–	5
DCCP with invalid chksum	N/A	26	20	2	–
UDP-Lite with correct chksum coverage	N/A	27	19	–	2
UDP-Lite with illegal chksum coverage	N/A	27	21	–	–

All UDP tests consist of single-packet requests (but, as with all tests, trying up to three times in case of a timeout) that are answered with a single packet having flipped ports. We included three types of UDP encapsulations: (i) SCTP over UDP, which uses UDP port 6511 [25]; (ii) DCCP over UDP, which uses UDP port 9899 [12]; (iii) Four random bytes over UDP, with destination port 443, with a normal checksum and with a zero checksum. This is meant to represent any protocol that is encapsulated in UDP without requiring a specific port mapping—for example, QUIC [26], RTMFP [27], an alternate proposal to encapsulate DCCP [28], and a proposal to use TCP over UDP [29]. Such common ports can sometimes represent many more applications [30].

Since we did not test different types of UDP content, our measurements implicitly assume that middleboxes do not parse the UDP payload. This assumption may well be wrong, but testing this would require extensive measurements at the application layer, which is beyond the scope of our measurement campaign. Since UDP packets with absolutely no payload would be exceptionally strange, yet larger payloads would unnecessarily increase the load on the network in our tests, for the tests with raw UDP and no encapsulated packet header in its payload, we added artificial random payload of the smallest possible size that fits in a packet without padding (4 bytes). All tests used 48001 as the source port, and initial client requests used 443 as the destination port in all the other tests above (SCTP, DCCP, UDP-Lite).

3.1. *NAT interference*

First, complementing our local NAT measurements, we test the interference of NATs with the studied transport protocols (there were 48 NATs; all the 48 clients behind NATs had different public addresses), using the observation numbers from Table 1. We present an overview of the results in Table 5. If a packet is forwarded by a NAT box without address translation (observation 3) or dropped (observation 4) before reaching the destination, then this behavior can not be differentiated from the packet being dropped by that NAT box. In this case, even the Tracebox-like-test in *fling* can not detect the location of the drop because the ICMP responses will never arrive at the *fling* host. Table 5 does therefore not distinguish between observations 3 and 4.

Table 5 contains an interesting surprise: there were some cases of DCCP and UDP-Lite seemingly working through a NAT (as shown by the Obs-5 and Obs-6 columns). In these cases, different from plain IP-level NAT'ing which ignores the transport header, the transport header checksum was updated by the NAT. In our local measurements in the previous section, we have seen that Linux can enable support for UDP-Lite and DCCP. The transport header checksum update for these protocols therefore gives us a hint that the NAT may run a Linux OS. We therefore decided to obtain a further indication of the NAT OS running Linux, as this may indicate *true* support for DCCP and UDP-Lite.

The result for UDP with an invalid checksum provides this hint, as Table 3 shows. The `sysctl` variable *nf_conntrack_checksum* is disabled, causing Linux to perform NAPT'ing on such packets, updating the checksum but resulting in a checksum error (observation 5). We also saw this behavior with FreeBSD. If, however, *nf_conntrack_checksum* is enabled, a Linux host will forward the packet without any form of NAT'ing (observation 3). Therefore, the lack of a response to an invalid UDP packet indicates that the NAT probably runs Linux rather than FreeBSD (or any of its derivatives).

How do NATs interfere with UDP?

All the NAT boxes performed NAPTing when sent UDP packets with a correct UDP checksum (observation 6). When we set packets with a UDP checksum of zero, 44 NATs kept this zero value intact. The remaining 4 NATs reacted strangely to the UDP packets with a zero checksum. They sometimes passed packets with a zero checksum, but also sometimes they recomputed it. This was done randomly for the same source and destination— noting that every test was carried out 3 times, plus we ran Tracebox, and saw multiple different results in the ICMP responses. This recomputation did not use a checksum adjustment algorithm, because that algorithm would have produced a wrong result. To obtain a stronger hint that this behavior indeed stems from single devices, we confirmed that these NATs were all one TTL hop away from their *fling* clients and consistently exhibited the same MAC address in the L2 header of their responses to the clients (which all were Ark nodes).

Do NATs block protocols other than UDP?

DCCP, UDP-Lite and SCTP packets were either dropped (observation 4), or were forwarded without address translation (observation 3) by 17 NATs, corresponding to 14 PlanetLab and 3 Ark nodes. Another 2 NATs also followed this behavior with UDP-Lite packets, but not with DCCP and SCTP packets. This leaves 29 NATs requiring further study for UDP-Lite and 31 NATs requiring a further analysis for DCCP and SCTP. In the case of SCTP, all the the remaining NATs exhibited observation 2 (IP-level NATing), which permits SCTP to work to some degree, as discussed in the previous section.

Are the NATs aware of DCCP?

31 NATs passed DCCP packets. 26 of the NATs (the Ark nodes) followed observation 2, which means that the DCCP packets were correctly received by our *fling* peer, but would normally be dropped at the receiver due to a checksum error. 5 of the 31 NATs updated the checksum correctly along with address translation (observation 6). To obtain an indication of whether these 5 NATs run Linux or some other OS, we tested both DCCP and UDP with a invalid checksum. 3 of the 5 NATs obeyed observations 3 or 4 in this case, which matches Linux with an enabled *nf_conntrack_checksum* variable. The remaining 2 NATs forwarded with NAPT'ing with an invalid checksum (observation 5), which could mean that they run Linux with a disabled *nf_conntrack_checksum* variable (default); as shown in Table 4, FreeBSD would only forward these packets with IP-level NAT'ing (observation 2).

Are NATs aware of UDP-Lite?

29 NATs passed UDP-Lite packets. 27 of them forwarded them with IP-level NAT'ing, without updating the transport checksum (observation 2—as with DCCP, this would normally provoke a drop at the receiver). The remaining 2 NATs updated the checksum correctly along with address translation (observation 6). Another measurement explored the result when clients sent packets with an illegal checksum coverage value of 7. This found that these two NAT boxes then exhibit observation 3 or 4. This behavior matches that of Linux in Table 3; however, an additional test using UDP with an invalid checksum produced no conclusive results (i.e., if the NATs ran Linux, the *nf_conntrack_checksum* variable

was disabled). Either way, because the correct checksum coverage value of 0 would be incorrect if the packets were interpreted as UDP, this result is a strong hint that these two NATs indeed support the UDP-Lite protocol.

How do these findings compare with local tests?

The previous study [31] that documented interference of NATs with SCTP and DCCP protocols examined 34 NAT devices from 15 vendors. 18 of the 34 devices (∼ 53%) passed SCTP packets with only IP-level NATing (observation 2). In our local tests in Section 2, counting devices with updated firmware as new devices, 22 out of 31 devices from 10 vendors passed SCTP packets with observation 2 (∼ 71%). In our *fling* study, 31 of the 48 NATs (∼ 65%) passed them with observation 2. Altogether, SCTP worked in this limited fashion in more than half of the cases in all tests. None of the 18 devices in [31] passed DCCP packets, and we found that all of our 31 devices followed observation 2 in Section 2, eliminating DCCP and UDP-Lite support because they break the transport checksum (UDP-Lite was not tested in [31]). Somewhat surprisingly, in our Internet study, we observed that DCCP worked correctly through 5 NATs, and obtained a strong hint that 2 of these 5 NATs support UDP-Lite.

3.2. Non-NAT middlebox interference

All but two of the *fling* clients with public IP addresses were able to reach the *fling* servers with the tested transport protocols. One of the affected clients only has an IPv4 address, and it belongs to AS12816 (the Leibniz-Rechenzentrum network; this Autonomous System (AS) is known to have various restrictions [32]). We observed that SCTP, DCCP and UDP-Lite packets were blocked at a distance of 2 hops (the router at $TTL = 2$ responded with a "time exceeded" ICMP error message; for $TTL \geq 3$, the same router responded with a "destination unreachable" ("port unreachable") ICMP error message). The other affected client was blocked on its IPv6 interface only, and we found this blocking to occur in its access network (AS680, an educational network in Germany).

In the reverse direction, i.e. the server responding to the client, we saw some restrictions that were not visible in the client-server direction. With IPv4, we noticed that 22 PlanetLab nodes were able to receive SCTP and DCCP packets, but UDP-Lite and all UDP encapsulations were dropped. At 6 Ark nodes, incoming SCTP and UDP-Lite packets were blocked, and DCCP was blocked for 5 of these 6 nodes. Additionally, none of the tested protocols (including all UDP encapsulations) worked on the reverse path for one Ark node, and this node exhibited the same behavior with IPv6.

With IPv6 (only available on Ark and NorNet), two more Ark nodes experienced dropping of all packets in all protocol tests, including all UDP encapsulations. A node for which SCTP, UDP-Lite and DCCP were blocked in IPv4 also saw blocking of (only) SCTP with IPv6. Three more nodes experienced dropping with IPv6 only, affecting SCTP, DCCP and UDP-Lite in two cases and SCTP, and UDP-Lite in one case. For all tests, with IPv4 and IPv6, we confirmed that packets were able to pass up to (and including) the neighboring ASes of the client AS for PlanetLab nodes and the client AS for all other (Ark and NorNet) nodes.

We observe that the number of clients experiencing protocol-specific failures that were not due to a NAT is less than 20.3% with IPv4 and less than 10.1% with IPv6, although the IPv6 tests exclude PlanetLab nodes where IPv6 was not available, which resulted in significant IPv4 dropping of the UDP variants. In conclusion, we now have an indication that, in the absence of NATs, even native DCCP or UDP-Lite may work in some special cases—and the number of paths with per-protocol limitations, in particular along the backward path, is significant.

In the outset, we have mentioned that the IETF TAPS Working Group is defining an API for an agile transport system [3]. Such a system offers the broader set of services that applications may need, while hiding the complexity of testing, configuring and using the various possible transport protocols from the application programmer. Protocol racing (testing which protocols work) is an important element of this machinery which has to be backed up by an informed policy. The results in this section provides important insight on how to configure the racing policy for use in the present Internet.

We now turn to measurements that will show how common protocol-specific behavior is in the wider Internet when we do not initiate connections at the client.

4. Large-scale internet path traversal tests

To carry out measurements at a larger scale for SCTP, DCCP, UDP-Lite and UDP, we implemented the packet header formats and constructed the headers in libcrafter[7]. We then modified Tracebox to build the packets and invoke the following tests (again using destination port 443 except for the UDP encapsulations):

- TCP (as a baseline): A SYN packet.
- SCTP: A SCTP INIT packet.
- DCCP: A DCCP Request packet.
- UDP-Lite: A UDP-Lite packet.
- UDP0: A UDP packet with a zero checksum.
- UDP: A general UDP packet (e.g., QUIC).
- SCTP over UDP: A SCTP INIT packet in a UDP packet with UDP source and destination ports set to 6511 [25].
- DCCP over UDP: A DCCP Request packet in a UDP packet with UDP source and destination ports set to 9899 [12].

The goal of these measurements is to understand the path traversal behavior of Internet ASes regarding these packets. Thus, to identify Tracebox destinations, we obtained 52112 different ASes using the Whois database (IP-to-AS mapping) from the dataset collected in [33] and retrieved one IP address per AS from this dataset.

In the period from January until April 2019, we ran our Tracebox tests from 15 IPv4 NorNet Core nodes (out of the set of NorNet Core nodes mentioned in Section 3), which were located in China, Korea, Germany and Norway. We then collected the data and extracted all the IPv4 addresses, the ICMP types and codes including the responses from the destinations IPs. For IP-to-AS mapping, we used the Whois database provided by Team Cymru.[8] We also took care not to analyze bogons[9] in our dataset. We retrieved neighboring ASes and AS-relationships (peer, provider or customer), from a CAIDA dataset.[10]

4.1. AS-level analysis

In the following, we will explain how we had to gradually reduce the number of considered paths in our dataset. For better clarity, Table 6 gives an overview of all these reductions, and presents the different total path numbers.

To avoid biasing the dataset due to filtering behavior in the ASes of the vantage points, we began our study with an analysis of the first-hop AS; this includes the border link to the second-hop AS. As a result, we only found 141 paths (source- destination IP address pairs) with protocol-specific policies in some first-hop ASes when the destination ASes belong to peer or customer networks of the client ASes or the second-hop ASes. For example, one first-hop AS only forwarded TCP and SCTP packets when the destination IP address belongs to a specific customer's AS. We removed these 141 paths from our dataset.

There was a larger number of paths that we had to remove due to overall communication failures. On 18,376 paths, we did not receive ICMP responses after a few hops from the source for our baseline protocol, TCP. We have also observed 9393 paths where we received no ICMP responses for any of the protocols outside the source's ASes. On

[7] libcrafter: https://github.com/runabk/libcrafter.

[8] Team Cymru: https://www.team-cymru.org/IP-ASN-mapping.html.

[9] We used this list to identify bogons: http://data.caida.org/datasets/bogon/.

[10] AS relationships: http://data.caida.org/datasets/as-relationships/.

Table 6
Overview of the number of paths used throughout the paper. For reference, the total number of paths from all measurements was 625907.

Line	# of paths considered	Section (page)	Comments
1	625766	Section 4.1 (9)	141 removed from total: protocol-specific policies observed in some first-hop ASes when the destination ASes belong to peer or customer networks of the measurement source's ASes or the second-hop ASes.
2	607390	Sectinon 4.1 (9)	18376 removed from line 1: No ICMP responses received after a few hops from the vantage points for the baseline protocol, TCP.
3	597997	Section 4.1 (9)	9393 removed from line 2: No ICMP responses received for any of the tested protocols outside the measurement source's ASes.
4	541421	Section 4.1 (10)	56576 removed from line 3: Encountered communication failures that were not related to the tested protocols (e.g. routing errors, as indicated with an ICMP "network unreachable" message).
5	510708	Section 4.1 (10)	30713 removed from line 4: Appeared for some but not all of the different protocol tests. This may be due to missing ICMP messages, but it may also indicate the effects of router load balancing (e.g. ECMP).
6	510050	Sectiob 4.1 (10)	Paths with three or more AS-hops.
7	456142	Section 4.1 (11)	Paths with four or more AS hops.
8	289464	Section 4.2 (11)	Destinations reachable with at least one protocol.

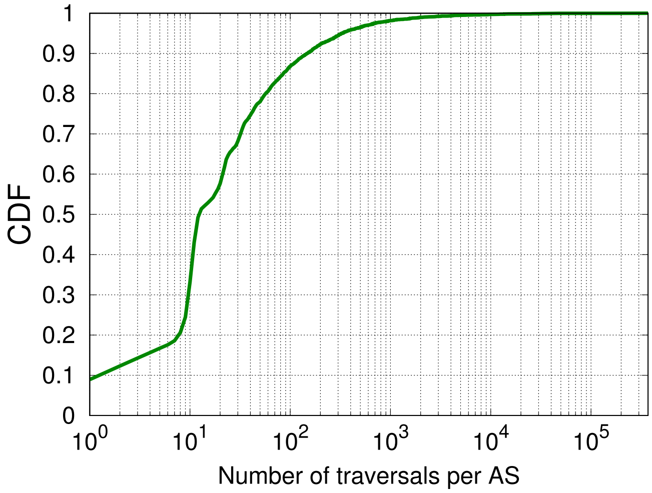


Fig. 4. The number of times an AS is contained in the 510,708 paths.

56,576 paths, we encountered communication failures that were not related to the type of protocol chosen (e.g. routing errors, as indicated with an ICMP "network unreachable" message). After removing all of these 843,45 paths, we were left with a total of 541,421 paths to analyze further.

These paths include a total of 56,494 ASes. Seven of these ASes contain the 15 nodes from which the tests were run; we call them "source ASes". We call the 50,473 ASes that contain the destinations "destination ASes". Many ASes were destination ASes in one test, but appeared on the way towards the destination AS in another test; we call them "transit ASes".

In 30,713 out of our 541,421 paths, we found ASes to appear for some but not all of the different protocol tests. This may be due to missing ICMP messages, but it may also indicate the effects of router load balancing (e.g. ECMP), which might let us draw wrong conclusions as we compare our results per tested protocol. We therefore removed these paths from our dataset, leaving us with a total of 510,708 paths covering 50,969 ASes. ASes in all of these paths appear in the same sequence, irrespective of the transport protocol. Fig. 4 shows that most of these ASes were transit ASes—e.g., 70% of the ASes were traversed by 10 or more paths.

Almost all of the remaining paths (99.9%) traversed three or more ASes. Fig. 5 shows successful AS traversal, categorizing ASes as either "destination" or "transit". All protocols were able to traverse the first-hop AS in our filtered dataset. When a packet is dropped, it is not necessarily accompanied by an ICMP error message, so we cannot reliably know the AS of the router performing the drop. However, we can observe successful traversal of an AS when the next AS is reached. Accordingly, in Fig. 5, a case of successful traversal of the destination AS means that the packet reached the destination; successful traversal of the transit AS means that the packet reached any later AS (e.g., the destination AS, and possibly the destination too, in the case of a path with 3 AS-hops).

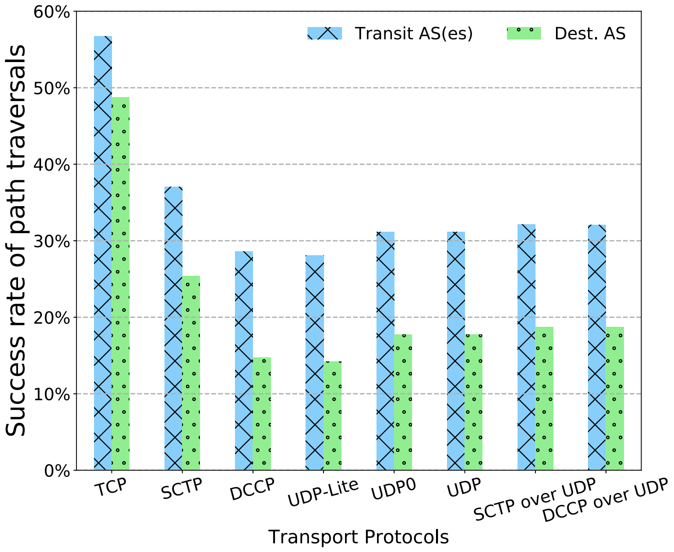


Fig. 5. Success of traversing different types of ASes (percentage of 510050 paths with three or more AS-hops, covering 50,969 ASes). All protocols had a 100% success rate of traversing the first-hop AS ("source ASes"). The results are a lower limit: lack of a response does not always mean that a path could not be traversed.

We found 100% traversal for the source AS, but reduced probability of traversal further along the path. The number of successful traversed ASes is smaller for destination ASes—50% to 60% of packets that reached the destination AS also verifiably reached the destination. Only TCP and SCTP are positive exceptions, respectively with close to half and roughly a quarter of destinations responding.

The results in Fig. 5 present a *lower limit* for successful AS traversal, indicated by receiving a response from a node further along the path. Drops may be silent, so it is incorrect to assume that the 37% success rate for SCTP transit AS traversal indicates that 63% of packets had been dropped. The low fraction of successful traversal for UDP to destination ASes could seem like a significant result for QUIC—however, these are cases where the destination host did not respond to UDP. This may be a natural behavior when no application is listening on UDP port 443 (and our destinations did not necessarily operate a server). We also note that the pre-defined IP address dataset that we used (one IP address per routable BGP prefix) may contain a large number of unreachable destinations: the authors of [33] report only 75% ping reachability for this dataset. This makes the TCP column in Fig. 5 a baseline for our tests rather than indicating true “Internet reachability”.

To better understand where drops occured, we took a closer look at the transit ASes of paths that contain four or more AS hops (456,142 paths). The tests showed that 97.8% of all transit ASes could be traversed, with the exception for all protocols being the last AS. This is a clear indication that **the majority of drops were either in the destination AS or the AS immediately prior to the destination**. This could be an indication that our tested destinations do not have an active server configured for the protocol being tested. If that is the case, then it would seem reasonable that when such a service is provided, the host may respond, and that any intervening middlebox (firewall, virus-scanner, load-balancer, etc) would have the policy updated to reflect the new service being offered.

To further analyze the location where a drop occurred using a particular protocol, we sought to predict the router that followed the router that sent a TTL-Exceeded message in the transit AS. This utilised the router address obtained with a successful protocol using the same path (source-destination IP address pair). For example, if SCTP was dropped in the penultimate AS, we could observe the following router from measurements using TCP. From this analysis, we found that on 72% of the paths the following router belonged to the destination AS.

It is tricky to identify the router that drops a packet when the drop is silent—in contrast, some routers explicitly notify the drop by sending ICMP “Communication prohibited”, “Protocol unreachable” and “Port unreachable” messages. For most protocols, we received ICMP responses in 25–29% of cases (but only 4.2% with TCP, and 17.6% with SCTP). With the exception of TCP, where, according to the received ICMP messages, roughly half the drops were in the last and penultimate AS, 70% or more of the ICMP error messages come from the destination AS, with most of them (79.5–89.3%) coming from the destination host. This appears to confirm our earlier result, that many drops occur in the destination AS, and specifically the destination host—but, given the small number of returned ICMP messages, this may also only mean that these were the locations that were more likely to produce ICMP messages instead of silently dropping packets.

The small fraction of returning ICMP messages has ramifications for an agile transport system that would “race” protocols: such a system should not be designed to heavily rely upon receiving ICMP error messages. This has analogies to the way Path MTU Discovery (PMTUD) had to be changed to no longer rely on ICMP messages (the newer version is called “Packetization Layer PMTUD” (PLPMTUD)). Generally, it seems that endpoints can no longer trust routers on paths to provide signaling via ICMP. Instead they need to actively send probes to understand what the path supports.

4.2. End system study

We received a response from the destination with at least one protocol on 289464 (53.5% of all) paths. In terms of destination ASes, 25972 (51.5%) were reachable with at least one protocol. Fig. 6 shows a destination analysis for all of these paths. As with Fig. 5 before, the shown responses constitute a lower limit for path traversal—not receiving a response does not always mean that packets of a particular protocol did not reach the host. Also note that, since our investigation focused on path traversal rather than analyzing the host software, we picked a destination host per AS, giving us a mix of servers and regular client hosts (rather than servers only, which would be a more natural choice for a destination analysis); this may further bias these results.

As one would expect, TCP won this race, with a missing response from the destination on only 5.3% of the paths. After TCP, surprisingly, not UDP but SCTP was the next best protocol when it comes to seeing a response at all (50.6% of the paths). This implies that native SCTP could sometimes be a better choice than SCTP over UDP (which provoked a response across 36.2% of the paths), provided that both end hosts are known to support it.

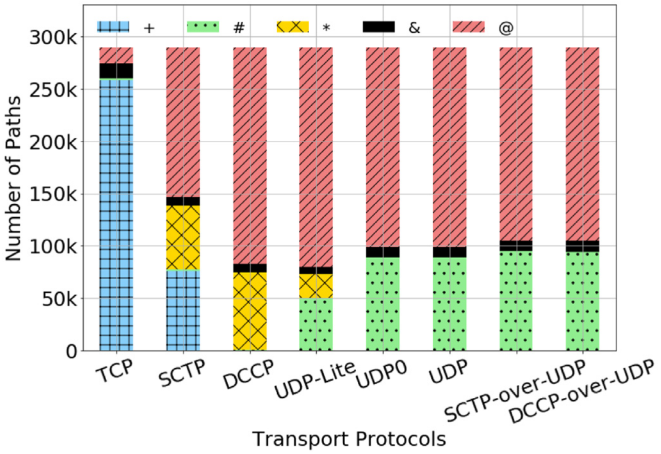


Fig. 6. Paths (source-destination IP address pairs) where the destination was reachable with at least one protocol. Responses: +: TCP: SYN/ACK or RST, SCTP: ABORT; #: ICMP port unreachable; *: ICMP protocol unreachable; &: Other ICMP error message; @: the destination was not reachable with the specific protocol, but it was reachable with another protocol. The results are a lower limit: lack of a response does not always mean that a path could not be traversed.

DCCP and UDP-Lite packets stood the smallest chance of eliciting a response from the destination (28.7% and 27.6% of the paths, respectively), but UDP-Lite looks better in this set: first, it saw a significant number of ICMP “port unreachable” messages, which may indicate support of the protocol by the destination host. Second, different from DCCP’s connection establishment handshake, the lack of a response is not a definite indication of the host not supporting UDP-Lite. DCCP over UDP appears to be a good substitute for native DCCP. UDP0 performed equal to UDP in our test, and hence was better than UDP-Lite.

Next, to derive advice for a TAPS transport system (protocol racing), we investigate correlations between protocols per path, shown in Fig. 7. The three most important results (largest cake pieces) in this figure contain one somewhat surprising result: On 21% of the paths, TCP and SCTP worked, but nothing else. To better understand whether “drop everything but TCP and SCTP” was truly a policy or whether it was simply the result of consecutive packet drops affecting different transport protocols along the path (e.g., DCCP and UDP-Lite early in the path and the UDP encapsulations later), we examined the ASes on these paths. There were 2955 ASes that appeared to directly drop everything but TCP and SCTP (we received responses for all of these protocols up to these ASes, but not beyond them). These 2955 ASes, which appeared on 53,578 (86.45%) of the 61973 paths where only TCP and SCTP worked, make up 5.23% of the total ASes in our study—a surprisingly large number of ASes where a generally restrictive policy seems to be in place that nevertheless allows SCTP, perhaps in support of telephony signaling (this it what SCTP was originally designed for, and it still is an important use case).

There were also some seemingly strange cases where several protocols worked but TCP did not elicit a response—e.g., the “all except TCP” category. All of these cases make up a small fraction of the total (TCP did not elicit a response on only 15,342 (5.3%) out of the 289,464 paths), and in 97.2% of these cases, TCP worked through the transit AS. Thus, these failures may just be due to a limiting policy at the destination AS (or a firewall at the destination itself) which does not allow incoming TCP packets on port 443.

We remind the reader that per-protocol responses only indicate lower limits for path traversal; even considering per-path protocol correlations as in Fig. 7, we do not have a comprehensive picture of “network

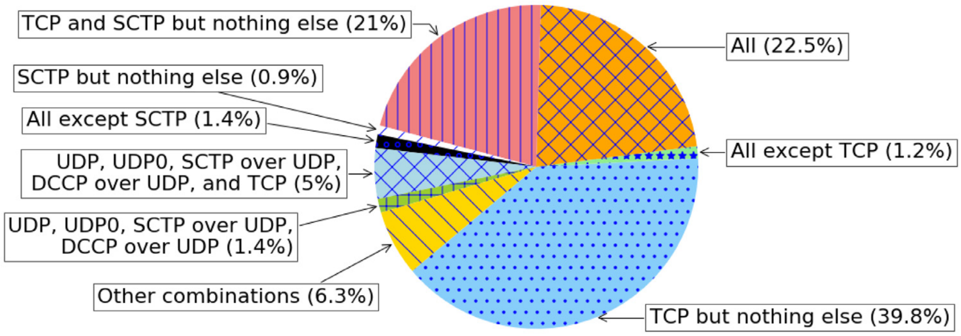


Fig. 7. Working protocol combinations per path. Results indicate path traversal and host support; for example, on 1.2% of the paths, TCP packets were dropped or (more likely) no active service was listening on TCP port 443.

support" for a certain protocol. For example, from the results that we have seen so far, it appears that native SCTP has a better chance to reach a host than UDP, which would by itself be a devastating result for QUIC. However, this would be a wrong interpretation. First, the apparent lack of UDP support can point at the usage of connection-tracking firewalls which would allow incoming UDP packets only after seeing an outgoing one from a matching source port first—indeed, we saw much wider support for UDP in the bidirectional measurement campaign in the previous section. Second, the lack of a response to a UDP packet may not always indicate that the UDP packet did not reach the host at all; it is possible that many of the tested hosts simply ignored incoming UDP packets instead of answering with an ICMP error message.

All in all, we can *not* conclude from this destination analysis that UDP works poorly, but we *can* conclude that native SCTP works surprisingly well, and we have indications that it *may* even work in some cases where UDP does not work. Also, (keeping in mind that they are a mix of clients and servers) we have **indications that hosts are better prepared to deal with UDP encapsulations than with native packets of type SCTP, UDP-Lite or DCCP (from best to worst, in order of appearance).**

5. Related work

To the best of our knowledge, Internet traversal of native UDP-Lite, DCCP or SCTP has not been measured in any prior research. Besides TCP, which was not our focus, related work has covered UDP, application-layer interference of middleboxes and NATs. NAT studies have primarily examined TCP and UDP; there is only one study which has also considered SCTP and DCCP in a local testbed, but not SCTP multi-homing, UDP-Lite or UDP with a zero checksum. In this study, Hätönen et al. [31] found that 53% (18 out of 34) of the measured NAT boxes passed SCTP packets and all the 34 NATs dropped DCCP packets. 4 NAT boxes simply forwarded the SCTP and DCCP packets without NAT'ing.

UDP: our large-scale investigation could not answer question around the cause of UDP traversal failures: were incoming packets dropped by connection-tracking firewalls? Did packets reach the host, but the host simply did not answer when no application was listening on the UDP port? From our local tests in Section 2, we know that—as one would expect—UDP works seamlessly through NATs; our smaller-scale bidirectional study in Section 3 gives us reason to believe that UDP encapsulations, including direct communication with port 443, work well when usage is initiated by the client. This is confirmed by a recent study of QUIC [34], which documents that QUIC already accounts for 7.8% of the traffic seen by a European Tier-1 ISP, and even 9.1% in the mobile network of a large European ISP (but somewhat less in two other traces). Despite being only an indirect indication of success in using the protocol, such traffic shares can probably only be produced by payload, which indicates a large success rate of negotiating QUIC. The traffic share of QUIC is also discussed in [35]: a steady growth is seen from 2015 in a traffic trace from a nation-wide ISP in Italy, culminating in a traffic share exceeding 12% in October 2017.

A more direct answer to the origins of UDP failures is given in [36]: using measurements between PlanetLab and Digital Ocean nodes, a quite low UDP blocking rate of roughly 1–5% is identified. Complementing this positive finding, a keynote at the CoNext 2018 EPIQ workshop [37] mentioned that Facebook saw a 93% use rate of QUIC with an improved racing algorithm. Most other studies on QUIC, e.g. [38] have focused on the operation of the protocol itself or server-side behavior [39] as well as observed performance [40]. Our measurements did not focus on performance, but this is an important direction for future work. For example, a decision for native or UDP-encapsulated use of a transport protocol should probably not only be based on reachability but also on the expected impact of traffic shaping (which has also been investigated for UDP in [36]).

Application layer: performance can also depend on the application layer [41], which we have considered beyond the scope of our study. However, application-layer interference by middleboxes is common, and it can take various forms, e.g. HTTP header injection [42,43]. Racing itself also has a performance impact, e.g. by potentially overloading a busy server; this is considered for TCP and SCTP, with and without TLS encryption, in [44]. Racing ("happy eyeballing") was further investigated for TCP over IPv4 and IPv6 in [45,46].

NAT: as already mentioned, many other middlebox studies exist, mostly focusing on TCP (e.g. [47,48]) or other forms of interference, e.g. with values of the DiffServ Code Point (DSCP) in the IP header [6,7,49,50]. The body of such related work is large, yet only loosely related to the present paper, which is specifically focused on non-TCP-protocols. An experimental evaluation of NAT traversal for TCP and UDP, with NAT devices from different vendors, is described in [51]. NAT traversal of TCP and UDP has been extensively studied in the context of peer-to-peer communication [31,52–55]. Some work has focused on NAT detection via ICMP messages, such as Tracebox [9] and "smart traceroute" [56].

One limitation of our work concerns the choice of destinations in our large-scale Internet study: we focused on covering a large number of ASes, and hence we did not pick the most common Internet servers, which may have drawn quite a different picture for the end host analysis (in related work, the Alexa top 1M list is often used, see for example [38]). Today, much of the Internet's traffic does not traverse a long distance, instead, Content Distribution Networks (CDN) are common, and the probability of success to reach a CDN server can be very different from the data that we have presented. Analysing CDNs is therefore definitely recommendable as future research.

6. Conclusion

The intention of this study was to arrive at some hints for a flexible transport system that tests ("races") protocols. Specifically, we focused on SCTP, DCCP and UDP-Lite, and asked: "Should they be a part of the

Table 7
Overview of NAT findings. "-" cases were not tested; X/Y means a positive result for X out of Y devices.

Protocol test	Off-the-shelf NAT tests (31 devices)	[31] (34 devices)	Linux	FreeBSD	Internet NAT tests (48 devices)
SCTP	22/31 IP layer	18/34 IP layer	Full support or IP layer only, depending on configuration	Full support or IP layer only, depending on firewall used	31/48 IP layer
SCTP multi-homing	–	–	Only when configured for IP layer NATing	Only when configured for IP layer NATing	-
DCCP	✗	✗	√	✗	5/48
UDP-Lite	✗	–	√	✗	2/48
UDP0	√	–	√	√	44/48
UDP (encapsulations)	√	√	√	√	√

set of transports that are tested, and used if they are available?" To obtain an answer, this paper has described a NAT analysis with home gateways, examined Linux and FreeBSD, performed bidirectional Internet tests (some of which involved more NATs), and did a one-sided large-scale study to better understand path traversal chances across the Internet.

Table 7 provides an overview of our key findings (together with findings from Hätönen et al. [31], the most closely related previously published work) regarding NATs. The take-away is: all UDP encapsulations work, DCCP and UDP-Lite hardly ever work, UDP with a zero checksum works quite often. Regarding SCTP, it seems common (around 2/3 of the tested devices—71 out of 113 total) that this protocol is treated as unknown, leading to IP layer NATing. This can enable communication for one SCTP source-destination pair at a time, but it is disastrous for DCCP and UDP-Lite due to their checksum's pseudo-header. In our own tests, we found second SCTP connection attempts to fail without harming the first ongoing SCTP communication. With (and only with) IP layer NATing, as we confirmed with the Linux and FreeBSD analysis, SCTP multi-homing also works.

Our analysis of protocol blocking considered a smaller-scale bidirectional study and found only minimal limitations from the client to the server, but more significant limitations from the server to the client (despite the fact that our emulated connections were client-initiated, i.e. this is not the result of a connection-tracking firewall). Blockage differed between IPv4 and IPv6, and was larger with IPv4.

The large-scale study found that the majority of protocol-specific drops occur (i) at the destination, (ii) in the destination AS, or (iii) in the penultimate AS. Our analysis of ICMP messages showed that for non-TCP protocols, the destination host itself is most likely to drop the test message and produce an ICMP error message. We also learned that roughly two-thirds of consistent protocol-specific drops are silent—this means that a protocol "racing" system should be designed to not heavily rely upon receiving ICMP error messages.

Finally, our destination host analysis—while limited in scope because the destination hosts were chosen to cover a large number of ASes rather than selecting the most commonly used servers—also indicated that SCTP works surprisingly well, and that it could even work in cases where UDP does not work (yet UDP generally seems to work better than SCTP, DCCP or UDP-Lite).

Putting all our findings together, we can now formulate some recommendations for a transport system seeking to race multiple transport protocols. Generally, we recommend trying UDP encapsulations first (all seem to work equally well). For native use of the protocols, we can conclude that for the current Internet:

SCTP: With an above-average probability of working through a NAT with IP layer NAT'ing, and given that there are (arguably rare) cases where SCTP appears to work, but UDP does not, we recommend falling-back to SCTP after trying SCTP over UDP. Since most NATs would apply IP layer NAT'ing, even multi-homing may work ("true" SCTP NAT support in Linux and FreeBSD can in fact create problems for multi-homing).

DCCP: There are cases where DCCP works through a NAT, and we even have a result where 22 of our PlanetLab clients saw all incoming UDP and UDP-Lite packets dropped but SCTP and DCCP succeeded. However, these situations where DCCP succeeds are so rare that they should probably be regarded at this time as corner cases, leading us to recommend against trying native DCCP unless the goal is to prefer a native deployment of DCCP.

UDP-Lite: Our recommendation is the same as for DCCP—both protocols have low probability of traversal. For IPv4 it is possible to emulate the functionality of UDP-Lite by using UDP with a zero checksum. This is an attractive alternative when no payload protection is needed, because traversal for UDP paths and home gateways did not seem to be hampered by having a zero checksum. Constraints remain on appropriate use of a zero UDP checksum with IPv6.

IPv6 constitutes a small part of our data, but from the data available, we saw that all tested protocols worked better across networks using IPv6. Thus, a transport system could be designed to be more "daring" when protocols operate over IPv6—and additional IPv6 measurements are encouraged to complete this picture.

This paper has strictly focused on the *traversal* of Internet paths and home gateways, not on performance—but now, equipped with some knowledge about the types of protocols and encapsulations that can work across the Internet, it would make sense to carry out a performance analysis. Such an analysis should not only cover the transport layer but also the impact of port numbers and data at the application layer. As a result of our findings, this research could be limited to native SCTP and UDP in case of IPv4, while in the case of IPv6 it probably should consider the whole set of SCTP, DCCP and UDP-Lite together with UDP.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. NAT/NAPT and their operations

The term "**NAT**" commonly refers to a middlebox function that interconnects a private/local network with the global Internet. Using a NAT is frequently necessary, since the public IPv4 address space is exhausted. A NAT usually maps many private (internal) IP addresses to public (external) IP addresses. A home router could map all internal addresses to a single public IP address. Carrier-grade NATs, deployed by Internet service providers, interconnect a larger set of internal addresses using multiple public addresses. This can easily lead to scenarios where there are multiple NATs on a single end-to-end path.

NATs only work at the IP level, i.e. a NAT would only share a set of public addresses with a limited number of *simultaneously* active internal devices. To enable multiple parallel transport connections from different internal devices, a Network Address and Port Translators (NAPT)[11] [58] translates the local IP address and port number pair into a global one. A NAPT needs to have higher-level protocol knowledge and needs to either implement a specific behavior for each supported transport protocol, or to implement a generic behavior that assumes that, e.g., port numbers always look the same.

In order for the private addresses to be reachable, the NAT box rewrites them with an external public address (or, if there is a chain of NATs, the last NAT rewrites to a public address). During the address translation, it must validate and update the IP header checksum. Similarly, incoming TCP and UDP connections to a NAPT are mapped from the external IP address and port pairs to an internal address and port number, and vice versa in the return direction. When a NAPT updates the port numbers or even only IP addresses, it needs also to update the checksum in the transport protocol's header because this checksum usually includes a "pseudo header" that protects the integrity of the IP addresses [59,60]. Since the transport layer checksum usually covers the payload, its calculation is resource-consuming. NATs therefore often use a *checksum adjustment* algorithm [15] for this update. The checksum algorithm is Internet-16 [61], which can be computed efficiently in either software or hardware.

NATs, by their nature, create problems for protocols or applications that transmit IP addresses in the IP payload. FTP is an example of an application that does this [58], prompting NAT vendors to implement special support for the FTP protocol. Modern applications should not need to transmit IP addresses. For example, Multi-Path TCP (MPTCP) [62] is a multi-homing and multi-path transport TCP extension that eliminates the need for applications to build such functions on top (which would require them to signal IP addresses), and the MPTCP implementation in the Linux kernel has been found to work flawlessly through NATs [63,64].

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.comnet.2020.107211.

CRediT authorship contribution statement

Runa Barik: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Writing - original draft, Writing - review & editing, Visualization. **Michael Welzl:** Conceptualization, Methodology, Formal analysis, Writing - original draft, Writing - review & editing, Visualization, Supervision, Resources. **Gorry Fairhurst:** Supervision, Formal analysis, Visualization, Writing - review & editing. **Ahmed Elmokashfi:** Writing - review & editing, Supervision, Resources. **Thomas Dreibholz:** Writing - review & editing, Resources. **Stein Gjessing:** Writing - review & editing, Supervision.

References

[1] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, Z. Shi, The QUIC transport protocol: design and internet-scale deployment, in: Conference of the ACM Special Interest Group on Data Communication, in: SIGCOMM '17, ACM, New York, NY, USA, 2017, pp. 183–196, doi:10.1145/3098822.3098842. ISBN 978-1-4503-4653-5

[2] S. Shalunov, G. Hazel, J. Iyengar, M. Kuehlewind, Low Extra Delay Background Transport (LEDBAT), (RFC 6817 (Experimental)), 2012, doi:10.17487/RFC6817.

[3] B. Trammell, M. Welzl, T. Enghardt, G. Fairhurst, M. Khlewind, C. Perkins, P.S. Tiesel, C.A. Wood, T. Pauly, An Abstract Application Layer Interface to Transport Services, Internet-Draft draft-ietf-taps-interface-05, Internet Engineering Task Force, 2019. Work in Progress

[4] D. Wing, A. Yourtchenko, Happy Eyeballs: Success with Dual-Stack Hosts, (RFC 6555 (Proposed Standard)), 2012, doi:10.17487/RFC6555.

[5] S. Dhesikan, D. Druta, P. Jones, C. Jennings, DSCP Packet Markings for WebRTC QoS, Internet-Draft draft-ietf-tsvwg-rtcweb-qos-18, Internet Engineering Task Force, 2017. Work in Progress

[6] A. Custura, A. Venne, G. Fairhurst, Exploring DSCP modification pathologies in mobile edge networks, in: Network Traffic Measurement and Analysis Conference (TMA), 2017, pp. 1–6, doi:10.23919/TMA.2017.8002923.

[7] R. Barik, M. Welzl, A.M. Elmokashfi, T. Dreibholz, S. Gjessing, Can WebRTC QoS work? A DSCP measurement study, 30th International Teletraffic Congress (ITC 30), 2018. Vienna, Austria

[8] N. Khademi, D. Ros, M. Welzl, Z. Bozakov, A. Brunstrom, G. Fairhurst, K. Grinnemo, D. Hayes, P. Hurtig, T. Jones, S. Mangiante, M. Tuxen, F. Weinrank, Neat: a platform- and protocol-independent internet transport api, IEEE Commun. Mag. 55 (6) (2017) 46–54, doi:10.1109/MCOM.2017.1601052.

[9] G. Detal, B. Hesmans, O. Bonaventure, Y. Vanaubel, B. Donnet, Revealing middlebox interference with tracebox, in: 13th ACM Internet Measurement Conference (IMC), Barcelona, Catalonia/Spain, 2013, doi:10.1145/2504730.2504757. ISBN 978-1-4503-1953-9

[10] R. Stewart (Ed.), Stream Control Transmission Protocol, (RFC 4960 (Proposed Standard)), 2007, doi:10.17487/RFC4960.

[11] R. Denis-Courmont, Network Address Translation (NAT) Behavioral Requirements for the Datagram Congestion Control Protocol, (RFC 5597 (Best Current Practice)), 2009, doi:10.17487/RFC5597.

[12] T. Phelan, G. Fairhurst, C. Perkins, DCCP-UDP: A Datagram Congestion Control Protocol UDP Encapsulation for NAT Traversal, (RFC 6773 (Proposed Standard)), 2012, doi:10.17487/RFC6773.

[13] R. Barik, M. Welzl, A.M. Elmokashfi, S. Gjessing, S. Islam, fling: a flexible ping for middlebox measurements, 29th International Teletraffic Congress (ITC), Genoa/Italy, 2017, doi:10.23919/ITC.2017.8064349. ISBN 978-0-9883045-3-6

[14] G. Fairhurst, M. Westerlund, Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums, (RFC 6936 (Proposed Standard)), 2013, doi:10.17487/RFC6936.

[15] P. Srisuresh, K. Egevang, Traditional IP Network Address Translator (Traditional NAT), (RFC 3022 (Informational)), 2001, doi:10.17487/RFC3022.

[16] F. Audet (Ed.), C. Jennings, Network Address Translation (NAT) Behavioral Requirements for Unicast UDP, (RFC 4787 (Best Current Practice)), 2007, doi:10.17487/RFC4787.

[17] L.-A. Larzon, M. Degermark, S. Pink, L.-E. Jonsson (Ed.), G. Fairhurst (Ed.), The Lightweight User Datagram Protocol (UDP-Lite), (RFC 3828 (Proposed Standard)), 2004, doi:10.17487/RFC3828.

[18] T. Stegel, J. Sterle, J. Bester, A. Kos, SCTP association between multi-homed endpoints over NAT using NSLP, Electrotech 5 (75) (2008) 27784. ISSN 0013-5852

[19] T. Stegel, J. Sterle, U. Sedlar, J. Bester, A. Kos, SCTP multihoming provisioning in converged IP-based multimedia environment, Comput. Commun. 33 (14) (2010) 1725–1735.

[20] L. Coene, Stream Control Transmission Protocol Applicability Statement, (RFC 3257 (Informational)), 2002, doi:10.17487/RFC3257.

[21] R.R. Stewart, M. Tuexen, I. Ruengeler, Stream Control Transmission Protocol (SCTP) Network Address Translation Support, Internet Draft, IETF, 2019.

[22] FreeBSD Documentation Project, FreeBSD Handbook, 52404th edition, 2018.

[23] D.A. Hayes, J. But, G. Armitage, Issues with Network Address Translation for SCTP, SIGCOMM Comput. Commun. Rev. 39 (1) (2008) 23–33, doi:10.1145/1496091.1496095.

[24] A. Keranen, C. Holmberg, J. Rosenberg, Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal, (RFC 8445 (Proposed Standard)), 2018, doi:10.17487/RFC8445.

[25] M. Tuexen, R. Stewart, UDP Encapsulation of Stream Control Transmission Protocol (SCTP) Packets for End-Host to End-Host Communication, (RFC 6951 (Proposed Standard)), 2013, doi:10.17487/RFC6951.

[26] J. Iyengar, M. Thomson, QUIC: A UDP-Based Multiplexed and Secure Transport, Internet Draft, IETF, 2019.

[27] M. Thornburgh, Adobe's Secure Real-Time Media Flow Protocol, (RFC 7016 (Informational)), 2013, doi:10.17487/RFC7016.

[28] M. Amend, A. Brunstrom, A. Kassler, V. Rakocevic, Lossless and overhead free DCCP - UDP header conversion (U-DCCP), Internet-Draft draft-amend-tsvwg-dccp-udp–header-conversion-01, Internet Engineering Task Force, 2019. Work in Progress

[29] S. Cheshire, J. Graessley, R. McGuire, Encapsulation of TCP and Other Transport Protocols over UDP, Internet-Draft draft-cheshire-tcp-over-udp-00, Internet Engineering Task Force, 2013. Work in Progress

[30] S. Alcock, J.-P. Möller, R. Nelson, Sneaking past the firewall: quantifying the unexpected traffic on major TCP and UDP ports, in: Internet Measurement Conference, in: IMC '16, ACM, New York, NY, USA, 2016, pp. 231–237, doi:10.1145/2987443.2987447.

[31] S. Hätönen, A. Nyrhinen, L. Eggert, S. Strowes, P. Sarolahti, M. Kojo, An experimental study of home gateway characteristics, in: 10th ACM SIGCOMM Conference on Internet Measurement, in: IMC '10, ACM, New York, NY, USA, 2010, pp. 260–266, doi:10.1145/1879141.1879174. ISBN 978-1-4503-0483-2

[32] C. Diekmann, Provably secure networks: methodology and toolset for configuration management, CoRR abs/1708.08228 (2017).

[33] B.J. Goodchild, Y.-C. Chiu, R. Hansen, H. Lua, M. Calder, M. Luckie, W. Lloyd, D. Choffnes, E. Katz-Bassett, The record route option is an option!, in: Internet Mea-

[11] RFC 3234 [57] calls NAPTs "NAT-PT", and Cisco terminology is NAT/PAT (PAT=Port Address Translation). We use the term NAPT to distinguish between IP-level NAT and NAT with port number translation.

surement Conference (IMC), in: IMC '17, ACM, New York, NY, USA, 2017, pp. 311–317, doi:10.1145/3131365.3131392. ISBN 978-1-4503-5118-8

[34] J. Rüth, I. Poese, C. Dietzel, O. Hohlfeld, A first look at QUIC in the wild, in: Passive and Active Measurement Conference (PAM), 2018. Berlin, Germany

[35] M. Trevisan, D. Giordano, I. Drago, M. Mellia, M. Munafo, Five years at the edge: watching internet from the ISP network, in: 14th International Conference on Emerging Networking EXperiments and Technologies, in: CoNEXT '18, ACM, New York, NY, USA, 2018, pp. 1–12, doi:10.1145/3281411.3281433.

[36] K. Edeline, M. Kühlewind, B. Trammell, B. Donnet, Copycat: testing differential treatment of new transport protocols in the wild, in: Proceedings of the Applied Networking Research Workshop, in: ANRW '17, ACM, New York, NY, USA, 2017, pp. 13–19, doi:10.1145/3106328.3106330.

[37] S. Iyengar, L. Niccolini, Keynote talk: moving fast at scale: Experience deploying IETF QUIC at Facebook, Workshop on the Evolution, Performance, and Interoperability of QUIC, EPIQ'18, ACM, New York, NY, USA, 2018.

[38] A.M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, A. Mislove, Taking a long look at QUIC: an approach for rigorous evaluation of rapidly evolving transport protocols, in: Internet Measurement Conference, in: IMC '17, ACM, New York, NY, USA, 2017, pp. 290–303, doi:10.1145/3131365.3131368.

[39] M. Piraux, Q. De Coninck, O. Bonaventure, Observing the evolution of QUIC implementations, in: Workshop on the Evolution, Performance, and Interoperability of QUIC, in: EPIQ'18, ACM, New York, NY, USA, 2018, pp. 8–14, doi:10.1145/3284850.3284852.

[40] L. Thomas, E. Dubois, N. Kuhn, E. Lochin, Google QUIC performance over a public SATCOM access, Int. J. Satell. Commun. Netw. (2019), doi:10.1002/sat.1301.

[41] F. Li, A.M. Kakhki, D. Choffnes, P. Gill, A. Mislove, Classifiers unclassified: an efficient approach to revealing IP traffic classification rules, in: Internet Measurement Conference, in: IMC '16, ACM, New York, NY, USA, 2016, pp. 239–245, doi:10.1145/2987443.2987464.

[42] S. Huang, F. Cuadrado, S. Uhlig, Middleboxes in the Internet: a HTTP perspective, in: Network Traffic Measurement and Analysis Conference (TMA), 2017, pp. 1–9, doi:10.23919/TMA.2017.8002906.

[43] T. Chung, D. Choffnes, A. Mislove, Tunneling for transparency: a large-scale analysis of end-to-end violations in the internet, in: Internet Measurement Conference, in: IMC '16, ACM, New York, NY, USA, 2016, pp. 199–213, doi:10.1145/2987443.2987455.

[44] G. Papastergiou, K.-J. Grinnemo, A. Brunstrom, D. Ros, M. Tüxen, N. Khademi, P. Hurtig, On the cost of using happy eyeballs for transport protocol selection, in: Proceedings of the 2016 Applied Networking Research Workshop, in: ANRW '16, ACM, New York, NY, USA, 2016, pp. 45–51, doi:10.1145/2959424.2959437.

[45] V. Bajpai, J. Schönwälder, Measuring the effects of happy eyeballs, in: Proceedings of the 2016 Applied Networking Research Workshop, in: ANRW '16, ACM, New York, NY, USA, 2016, pp. 38–44, doi:10.1145/2959424.2959429.

[46] V. Bajpai, J. Schönwälder, A longitudinal view of dual-stacked websites—failures, latency and happy eyeballs, IEEE/ACM Trans. Netw. 27 (2) (2019) 577–590, doi:10.1109/TNET.2019.2895165.

[47] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, H. Tokuda, Is it still possible to extend TCP? in: ACM SIGCOMM Internet Measurement Conference (IMC), 2011, doi:10.1145/2068816.2068834. ISBN 978-1-4503-1013-0

[48] R. Craven, R. Beverly, M. Allman, A Middlebox-cooperative TCP for a Non End-to-end Internet, in: ACM Conference on SIGCOMM, in: SIGCOMM '14, ACM, New York, NY, USA, 2014, pp. 151–162, doi:10.1145/2619239.2626321. ISBN 978-1-4503-2836-4

[49] R. Barik, M. Welzl, A. Elmokashfi, How to say that you're special: can we use bits in the IPv4 header? in: Proceedings of the 2016 Applied Networking Research Workshop, in: ANRW '16, ACM, New York, NY, USA, 2016, pp. 68–70, doi:10.1145/2959424.2959442.

[50] R. Barik, M. Welzl, A. Elmokashfi, T. Dreibholz, S. Islam, S. Gjessing, On the utility of unregulated IP DiffServ Code Point (DSCP) usage by end systems, Perform. Eval. 135 (2019) 102036, doi:10.1016/j.peva.2019.102036.

[51] C. Jennings, NAT Classification Test Results, Internet-Draft draft-jennings-behave-test-results-04, Internet Engineering Task Force, 2007. Work in Progress

[52] S. Guha, P. Francis, Characterization and measurement of TCP traversal through NATs and firewalls, in: 5th ACM SIGCOMM Conference on Internet Measurement, in: IMC '05, USENIX Association, Berkeley, CA, USA, 2005. 18–18

[53] L. Makinen, J.K. Nurminen, Measurements on the feasibility of TCP NAT traversal in cellular networks, in: Next Generation Internet Networks, 2008, pp. 261–267, doi:10.1109/NGI.2008.42.

[54] G. Halkes, J. Pouwelse, UDP NAT and firewall puncturing in the wild, in: 10th International IFIP TC 6 Conference on Networking - Volume Part II, in: NETWORKING'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 1–12.

[55] H. Kavalionak, A.H. Payberah, A. Montresor, J. Dowling, NATCloud: cloud-assisted NAT-traversal service, in: 31st Annual ACM Symposium on Applied Computing, in: SAC '16, ACM, New York, NY, USA, 2016, pp. 508–513, doi:10.1145/2851613.2851640.

[56] R. Zullo, A. Pescapé, K. Edeline, B. Donnet, Hic sunt nats: uncovering address translation with a smart traceroute, in: 2017 Network Traffic Measurement and Analysis Conference (TMA), 2017, pp. 1–6, doi:10.23919/TMA.2017.8002924.

[57] B. Carpenter, S. Brim, Middleboxes: Taxonomy and Issues, (RFC 3234 (Informational)), 2002, doi:10.17487/RFC3234.

[58] P. Srisuresh, M. Holdrege, IP Network Address Translator (NAT) Terminology and Considerations, (RFC 2663 (Informational)), 1999, doi:10.17487/RFC2663.

[59] J. Postel, Transmission Control Protocol, (RFC 793 (Internet Standard)), 1981, doi:10.17487/RFC0793.

[60] J. Postel, User Datagram Protocol, (RFC 768 (Internet Standard)), 1980, doi:10.17487/RFC0768.

[61] R. Braden, D. Borman, C. Partridge, Computing the Internet Checksum, (RFC 1071 (Informational)), 1988, doi:10.17487/RFC1071. Updated by RFC 1141

[62] A. Ford, C. Raiciu, M. Handley, O. Bonaventure, TCP Extensions for Multipath Operation with Multiple Addresses, (RFC 6824 (Experimental)), 2013, doi:10.17487/RFC6824.

[63] G. Detal, C. Paasch, O. Bonaventure, Multipath in the middle(box), CoNEXT Workshop HotMiddlebox, 2013.

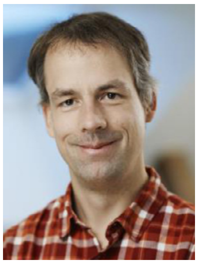
[64] O. Bonaventure, C. Paasch, G. Detal, Use Cases and Operational Experience with Multipath TCP, (RFC 8041 (Informational)). 10.17487/RFC8041

Runa Barik is a Ph.D. candidate at the Department of Informatics, University of Oslo (UiO), Norway. Mr. Barik received a Master of Science by Research from IIT Mandi, India in 2014. His previous research includes developing theoretical models, simulations and kernel implementations related to TCP and MPTCP. Mr. Barik has actively involved in several EU projects. He has submitted two IETF drafts and presented talks at the IETF meetings. His current research includes network measurements, detecting middleboxes and opportunistic use of various IPv4 and IPv6 Internet Protocols.

Michael Welzl is a full professor in the Department of Informatics of the University of Oslo since 2009. He received his Ph.D. (with distinction) and his habilitation from the University of Darmstadt/Germany in 2002 and 2007, respectively. His book "Network Congestion Control: Managing Internet Traffic", is the only introductory book on network congestion control. Michael has been active in the IETF and IRTF for many years, e.g. by chairing the Internet Congestion Control Research Group (ICCRG) leading the effort to form the Transport Services (TAPS) Working Group. He has also participated in several European research projects, including roles such as coordinator and technical manager.

Gorry Fairhurst received his first degree in applied physics and electronics from the University of Durham, UK, and a Ph.D. in communications engineering from the University of Aberdeen, UK. He is a professor at the University of Aberdeen, working in Internet Engineering where he specialises in transport (congestion-control, end-to-end Internet communication, and transport interfaces), military and broadband satellite access and large-scale Internet measurement. His research interests include protocol design, link ARQ, and Internet transport. He is an active participant in the specification and engineering of Internet standards and chairs the IETF TSVWG working group.

Ahmed Elmokashfi is a senior research scientist at Simula Metropolitan Centre for Digital Engineering and Simula Research Laboratory in Norway. He is currently heading the Centre for Resilient Networks and Applications (CRNA), which is part of the Simula Metropolitan Centre that is funded by the Norwegian government. CRNA focuses on conducting long-term research and innovation at a high international level, on robustness and reliability of communications networks and applications. Dr. Elmokashfi's research interest lies in network Measurements and Performance. In particular, he has been focusing on studying the resilience, scalability, and evolution of the Internet infrastructure; the measurement and quantification of robustness in mobile broadband networks; and the understanding of dynamical complex systems. Over the past few years, he has been leading and contributing to the development, operation and management the NorNet testbed infrastructure, which is a countrywide measurement setup for monitoring the performance of mobilebroadband networks in Norway. Dr. Elmokashfi received his PhD degree from the University of Oslo in 2011.



Thomas Dreibholz has received his Diplom (Dipl.-Inform.) degree in computer science from the University of Bonn in Bonn, Germany in 2001. Furthermore, he has received his Ph.D. degree (Dr. rer. nat.) in 2007, as well as his Habilitation (Priv.-Doz.) degree in 2012 from the University of Duisburg-Essen in Essen, Germany. Now, he works as Chief Research Engineer for the Simula Metropolitan Centre for Digital Engineering (SimulaMet) in Oslo, Norway. He has published and presented more than 70 research contributions at international conferences and in journals, on the topics of Reliable Server Pooling (RSerPool), the Stream Control Transmission Protocol (SCTP), Quality of Service (QoS), and multi-homed network infrastructures. Furthermore, he has contributed multiple Working Group and Individual Submission Drafts to the IETF standardisation processes of RSerPool and SCTP. He is also co-author of multiple RFC documents published by the IETF. In addition, he has written the RSerPool reference implementation.

Stein Gjessing is a professor of computer science in Department of Informatics, University of Oslo. He received his the Cand. Real. degree in 1975 and his Dr. Philos. degree in 1985, both form the University of Oslo. He acted as Head of the Department of Informatics for 4 years from 1987. Gjessing's original work was in the field of programming languages and programming language semantics, in particular related to object oriented concurrent programming. He has worked with computer interconnects and computer architecture for cache coherent shared memory, with DRAM organization, with ring based LANs and with IP fast reroute. His current main research interests are multipath routing and transport in IP-like networks.