H2020 European Union funding
for Research & Innovation

Measurement and Architecture for a Middleboxed Internet

H2020-ICT-688421

Initial Report on Measurement Development and Deployment

Author(s):	ULg	Benoit Donnet (ed.), Korian Edeline, Raffaele Zullo
	ETH	Brian Trammell, Mirja Kühlewind
	TID	Oscar Gonzlez de Dios
	UNIABDN	Iain R. Learmonth
	SRL	Andra Lutu

Document Number:	D1.1
Internal Reviewer:	Diego R. Lopez
Due Date of Delivery:	31 December 2016
Actual Date of Delivery:	23 December 2016
Dissemination Level:	Public

Disclaimer

Contents

Executive Summary

The Measurement and Architecture for a Middleboxed Internet (MAMI) project is developing and experimentally deploying a Middlebox Cooperation Protocol (MCP), embedded in a more Flexible Transport Layer (FTL) to support stack evolution. To do so, large-scale measurements of middleboxes in the public Internet conducted on top of public available testbeds including the H2020 MONROE or other suitable FIRE+ testbeds are essential to get a good knowledge of middlebox deployment in reality and as a basis to model middlebox behavior for testing MCP's applicability.

MAMI WP1 focuses on those measurements. In particular, it is developing new measurement tools and techniques, deploying them at large scale and storing the collected measurement data to make the taken observation publicly available.

This deliverable reports MAMI early efforts in this direction, focusing on measurement tools and techniques development and deployment in the first year of the project. We discuss what has been achieved so far in terms of (i) detecting path impairment caused by middleboxes and (ii) finding these boxes in the network.

1 Introduction

In MAMI WP1 (“Large-Scale Measurements of Deployed Middleboxes”), the project aims (among other goals) at developing new, enhanced measurement techniques and methodologies to detect and subsequently analyze middlebox behavior. The collected dataset does not only provide the basis for the development of an architecture allowing endpoints and middleboxes to cooperate (i.e., WP3) and for MAMI’s further experimentation with such a protocol (i.e., WP2), it will also foster further research on this topic outside the project: MAMI makes measurement tools, methods, and data available to the network research and operations community at large (see github.com/mami-project and the MAMI Path Transparency Observatory). The collected measurement data and derived observations can be used to a) develop new and innovative measurement techniques or enhance existing, well-established ones and b) as input for transport protocol development beyond the scope of MAMI. For instance, `copycat` (described in Sec. 2.1.2) is a tool for comparing loss, latency, and throughput for TCP and UDP by generating TCP-shaped traffic with UDP headers. The objectives beyond those measurements are to assess whether one could run new transport protocols for the Internet over UDP. The interesting feature of `copycat` is that it does not only test connectivity and feature support but also measures Quality of Service (QoS) characteristics (throughput, loss, delay) by sending a sufficient amount of content data. Therefore, `copycat` can be used as generic test tool of new transport protocols and compare their performance to the current deployed state of the art.

The purpose of this deliverable is to report what has been achieved so far in MAMI WP1. In particular, we describe efforts made in developing new measurement tools as well as initial deployment and results obtained with those tools. In this deliverable, we first (Chap. 2) discuss tools for detecting path impairment caused by middleboxes for different protocols or protocol extensions: `PATHspider` performing A/B testing for TCP and IP field and options, `copycat` testing UDP vs. TCP performance and H2tool/`EYEORG` for HTTP/2 support and performance testing. Here, a path impairment is any measurable action a middlebox performs on traffic between a source and a destination (typically by modification of protocol bits or differential treatment for certain kinds of protocols/protocol extension leading to degrading performance or even complete blocking). Further in Chap. 2, we also describe tools for finding concrete instances of certain middleboxes in the network and subsequently analyze their behavior: `tracebox` as a generic tool for middlebox and proxy detection and NAT `Revelio`, especially focusing on NAT and Carrier-Grade NAT. Afterwards we report on initial deployment and results obtained from each those tools, mostly run as separate studies during the development phase of the tools (Chap. 3). Finally, Chap. 4 concludes this deliverable by summarizing its main achievements and discussing future directions of MAMI WP1 on tool development and enhancements, large-scale and automated measurement campaigns, as well as integrate of data and combined analysis of thereof in the Path Transparency Observatory.

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

2 Techniques and Implementation

This chapter describes the measurement techniques and tools developed so far by the MAMI project. The project development corresponds to two categories: (i) measuring path impairment due to the presence of middleboxes (Sec. 2.1) and, (ii), revealing the presence of middleboxes in the network (Sec. 2.2).

In a nutshell, path impairments are detected by `PATHspider`, that checks Internet path transparency to various protocol features, `copycat`, that evaluates differences in connectivity and QoS due to differential treatment for different transport protocols, the H2tool that identifies HTTP/2 impairments, and `EYEORG`, a platform for crowdsourcing web Quality of Experience (QoE) measurements. Further, middlebox presence is revealed by two tools: `tracebox`, a `traceroute` extension that reveals the presence of middleboxes along a path (in both mobile and wired network), and NAT `Revelio` that detects the presence of Carrier Grade NATs (CGN) deployed by network operators.

2.1 Measuring Path Impairment

2.1.1 PATHspider

`PATHspider` is a generalization of the earlier ECN Spider tool [26] to measure Internet path transparency to various protocol features by answers the following questions: will an attempt to use feature X fail, or will it cause connection impairment? Transparency to a certain feature can be impaired at any point along the path, either when a middlebox along the path treats those packets differently by design or accident, or when faulty server-side implementations respond poorly to unexpected traffic.

`PATHspider` performs controlled A/B experiments[1], comparing control traffic (usually a vanilla TCP connection) to the target with experimental traffic (using the feature under test). It uses integrated passive measurement of the generated test traffic to provide a network-level view of impairments as seen at the vantage point. `PATHspider` runs multiple tests concurrently to supports large-scale scanning of millions of targets within a reasonable amount of time. Further, concurrent setup of control and experimental connections minimizes the chance that a transient change at the target – e.g. rerouting or other reconfiguration – will affect the results.

By running `PATHspider` from multiple, diverse vantage points, effects close to the target can be isolated from effects due to manipulation of packets by on-path devices where an impairment is detected on one path but not another. `PATHspider` is designed to work with the MAMI Path Transparency Observatory, to be described in detail in a future deliverable D1.2, to combine observations from multiple vantage points, as well as to compare observations from diverse measurement campaigns.

`PATHspider` provides a framework for easily writing and integrating tests for new protocol features as plugins. The framework splits tests into multiple stages. The framework, as shown in Fig. 1, consists of a *configurator* for changing system settings between the control (A) and experimental (B) configurations, if necessary; an *observer* that captures and analyses traf-

[1]"A/B testing is a term for a randomized experiment with two variants, A and B, which are the control and variation in the controlled experiment" [16].

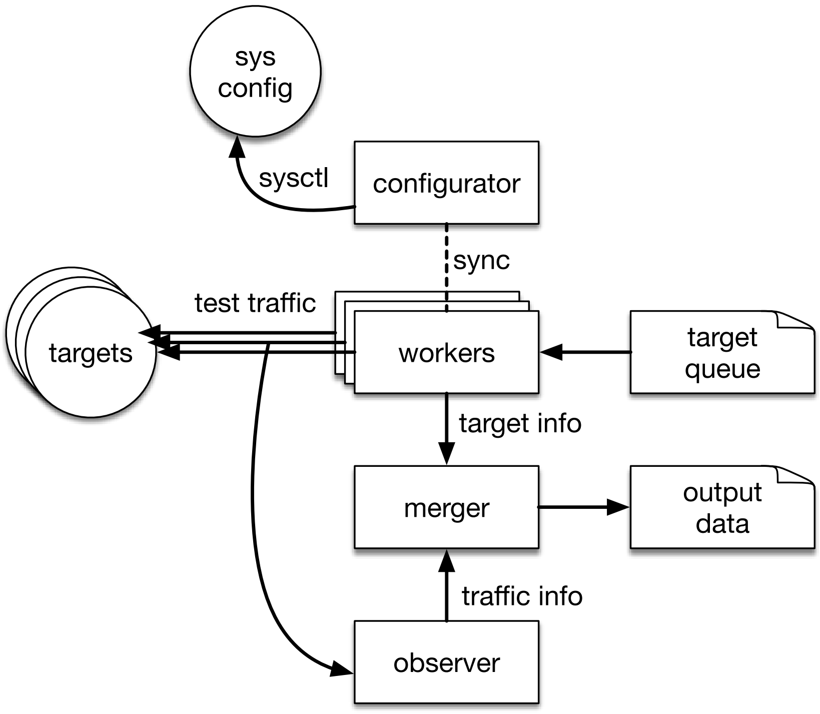


Figure 1: Block diagram illustrating control flow and flow of data between PATHspider components.

fic traces during the tests; *worker* threads that synchronize with the configurator to generate test traffic using each configuration; and a *merger* that combines API-level information from the workers with passive traffic information from the observer. PATHspider plugins are implemented by providing specific functions for the configurator, workers, observer, and merger.

PATHspider currently supports measurements of path transparency of Explicit Congestion Notification in TCP [20], TCP Fast Open [4], and Differentiated Services [18], and can probe end-host support for various web protocols using protocol negotiation extensions to Transport Layer Security (TLS)[8].

The ECN plugin measures both ECN negotiation, as well as connectivity issues related to ECN negotiation attempts. The TFO plugin does the same for the TCP Fast Open option, with the caveat that it has to use two connections for the TFO test in order to ensure a cookie has been cached and that data can be transmitted on the initial SYN of the second connection. In both cases, the host machine's kernel support for the tested feature is used, instead of crafted packet injection, so PATHspider evaluates the actual client behavior in these cases.

The DSCP plugin uses packet mangling to set an arbitrary DSCP codepoint on outgoing test traffic, and measures the effect on the codepoint on downstream traffic. Studies to find one-way DSCP rewriting require additional observation at a controlled target.

The TLS plugin measures transparency to the Application Layer Protocol Negotiation (ALPN) and Next Protocol Negotiation (NPN) extensions and as a side effect, measures support for HTTP 2 in the target webservers. This plugin is based on the same methodology as developed for the H2tool as described in section 2.1.3.

See `https://pathspider.net/` for the PATHspider codebase and documentation.

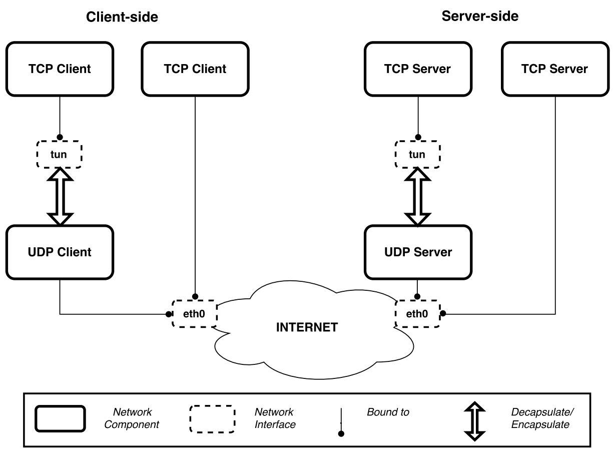


Figure 2: `copycat` measurement methodology.

2.1.2 `copycat`

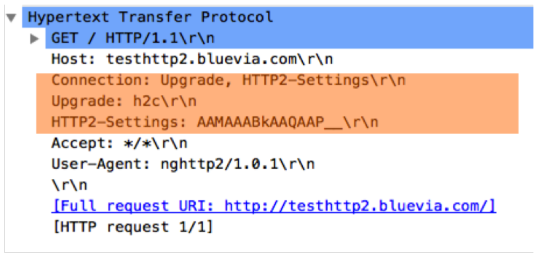
`copycat`[2] simultaneously runs pairs of flows between two endpoints, a standard TCP flow and another TCP flow using UDP as an "outer" transport, to evaluate differences in connectivity and QoS due to differential treatment of different transport protocol headers. The TCP in UPD encapsulation is used to simulate a new transport protocol that runs over UDP but provides the same traffic characteristics, e.g. by using TCP-friendly congestion control, for comparison. The two flows run in parallel with the same 4-tuples (port numbers and IP addresses), to obtain flows with the most similar possible treatment from the network, but with different transport headers. By comparing performance of these flows to each other, we are able to detect differences that can be attributed to differential treatment by the path.

As shown in Fig. 2, the UDP flow is obtained by tunneling a TCP flow over UDP. To achieve this, `copycat` first creates a `tun` virtual network interface that simulates a network layer device and operates at Layer 3. In our measurement setup, each node runs both the `copycat` client and the server. On the client side, the TCP client connects to its peer via the Internet-facing interface and receives data from it, writing it to disk. The UDP client consists of the TCP client bound to the `tun` interface, which is in turn bound by `copycat` to a UDP socket on the Internet-facing interface. `copycat` thus works as a tunnel endpoint, encapsulating TCP packets from `tun` in UDP headers, and decapsulating received UDP packets back to TCP packets to `tun`. The server-side consists of a similar arrangement, listening for connections from clients and sending data to them. The client waits for both transfers, via TCP and TCP- controlled UDP, to be completed before connecting to the next destination.

Each flow consists of a unidirectional data transfer. The smallest flow is calibrated not to exceed the TCP initial window size, which ranges from 2-4 to 10 Maximum Segment Size (MSS) depending on the kernel version used by the different measurement platforms [2, 5]. This ensures that for the smallest flow, we send all data segments at once. Then, we increase the size of the flows by arbitrary factors of 3, 30, 300, and 1500 to observe the impact of differential treatment for congestion-controlled traffic of larger flows.

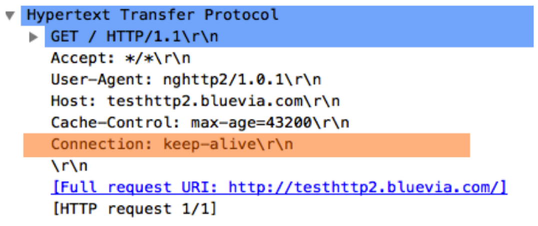
To avoid unwanted fragmentation of UDP datagrams and ICMP `message-too-long` errors, and

[2]Sources are freely available at `https://github.com/mami-project/udptun`



```
▼ Hypertext Transfer Protocol
  ▶ GET / HTTP/1.1\r\n
    Host: testhttp2.bluevia.com\r\n
    Connection: Upgrade, HTTP2-Settings\r\n
    Upgrade: h2c\r\n
    HTTP2-Settings: AAMAAABkAAQAAP__\r\n
    Accept: */*\r\n
    User-Agent: nghttp2/1.0.1\r\n
    \r\n
    [Full request URI: http://testhttp2.bluevia.com/]
    [HTTP request 1/1]
```

(a) Upgrade captured at client side.



```
▼ Hypertext Transfer Protocol
  ▶ GET / HTTP/1.1\r\n
    Accept: */*\r\n
    User-Agent: nghttp2/1.0.1\r\n
    Host: testhttp2.bluevia.com\r\n
    Cache-Control: max-age=43200\r\n
    Connection: keep-alive\r\n
    \r\n
    [Full request URI: http://testhttp2.bluevia.com/]
    [HTTP request 1/1]
```

(b) Upgrade captured at server side after middlebox.

Figure 3: H2tool experiment: UPGRADE and HTTP2-SETTINGS headers removed.

to ensure that packets from both tunneled and non-tunneled flows are equally sized, we decrease the MSS of the tunneled TCP flow by the size of the tunnel headers (IP header + UDP header = 28 Bytes).

`copycat` is coded in C to minimize the tunneling overhead. I/O multiplexing is handled using `select()`. During the measurement run network traces are captured at `eth0` with `libpcap`.

2.1.3 H2tool

The H2tool[3] tests a number of HTTP/2 related functionalities against a target URL. The HTTP protocol was recently updated from version 1.1 to HTTP/2. The rationale of the change is to overcome the limitations of a protocol that has been used for more than 15 years to carry Internet traffic. The new version of the protocol, described in detail in RFC 7540, allows to send the traffic both end-to-end encrypted (the version of the protocol is known as h2) and non-encrypted (the version of the protocol is known as h2c or HTTP/2 in the clear).

In particular, the H2tool is able to test the protocol negotiation phase where HTTP/2 is selected by using both ALPN and the older NPN mechanisms. Some sites announce HTTP/2 availability by ALPN, but later on, when trying to switch to HTTP/2, there is no such real support. The tool is able to detect if a page announces H2, and if it (as well as the path) really supports it. The tool also performs measurements in the negotiation phase, such as deriving timeout based on the ALPN duration.

In a variant of HTTP/2, called *HTTP/2 in the clear* (h2c), the traffic is sent directly over the TCP connection, without the intermediate TLS version negotiation. The proposed way to start an h2c connection is by using the UPGRADE mechanism. The sessions starts in HTTP 1.1, proposing to upgrade to h2c and, if the server agrees, there is a protocol switch to h2c. However, the Internet community has argued that the UPGRADE mechanism, which relies on specific HTTP headers, could be tampered by the behaviour of middleboxes.

Due to the above, the H2tool detects interference of middleboxes in the UPGRADE mechanism for h2c. Some middleboxes remove part of the header, and the connection falls back to HTTP 1.1. In order to better explain the behaviour, Fig. 2.3(a) shows the HTTP packet with the UPGRADE captured at the client side, before entering the network, which includes CONNECTION, UPGRADE and HTTP2-SETTINGS headers. Fig. 2.3(b) shows the HTTP packet

[3]Sources are freely available at `https://github.com/mami-project/h2-measurements`

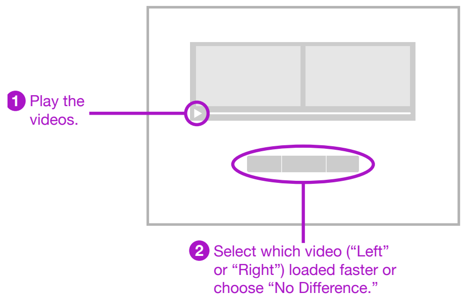

1 Play the videos.
2 Select which video ("Left" or "Right") loaded faster or choose "No Difference."

Figure 4: EYEORG's experiment: Participants watch side-by-side page load videos and indicate which load is faster.

captured at the server side after passing a proxy that modifies HTTP headers. It can be seen that the UPGRADE and HTTP2-SETTINGS headers have been removed, and thus, the connection is not able to progress to h2c. While different middleboxes react in a different way, this behaviour leads in all cases to the mentioned fallback to HTTP 1.1.

In addition to the tool, a module supporting h2c for the Apache web server was implemented and donated to Apache Project. There is a publicly available h2c test server[4], running an Apache server supporting both h2 and h2c for testing.

2.1.3.1 EYEORG

In addition to objective network measurements of HTTP/2 performance provided by the H2tool, EYEORG [29] has been developed as a platform for crowdsourcing subjective user test for web QoS measurements. EYEORG allows researchers to test the impact of changes to how a page is structured or delivered, e.g. when HTTP 1.1 or HTTP/2 is used. It uses crowdsourced participants to scale[5] and shows videos of pages loading to provide a consistent experience to all participants, regardless of their network connections and device configurations. With this approach, it is possible to maintain full control of experimental conditions and to recruit any participant with a modern web browser, without requiring special hardware or software.

As an example test and inline with the objective HTTP/2 comparison test performed by the H2tool on the network directly, we ran an experiment where participants watch two page load videos simultaneously and pick which loaded faster or "No Difference" (see Fig. 2.1.3.1). Video pairs are shown in a random order (i.e., HTTP 1.1 is not always on the left and HTTP/2 is not always on the right). As there is no guarantee that two videos in a browser stay perfectly synchronized (for instance, lost packets might momentarily stall one video while the other continues playing), we splice them into a single video file; if the playback stalls, both sides are affected equally. This kind of comparison experiment is easier to perform than e.g. indicating page load time directly for each web page load video as it is not important to choose precisely when a page is loaded completely.

[4]h2 and h2c public test URL `http://testhttp2.bluevia.com/tests.html`
[5]It is currently integrated with two popular service, Microworkers [17] and Crowdflower [6].

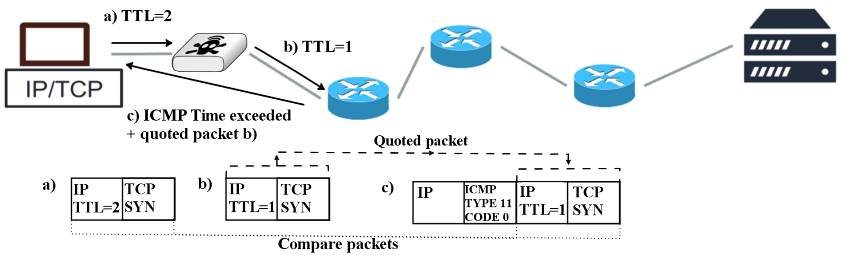


Figure 5: Middlebox detection and localization with `tracebox`.

2.2 Finding Middleboxes

2.2.1 `tracebox`

To reveal the presence of middleboxes along a path, we are working on further developing `tracebox` [7], an extension to the widely-used `traceroute` tool [28]. `tracebox`'s mechanism is illustrated in Fig. 5. It relies on RFC1812 [3] and RFC792 [19] stating that the returned ICMP `time-exceeded` message should quote the IP header of the original packet and respectively the complete payload or the first 64 bits. `tracebox` uses the same incremental approach as `traceroute`, i.e., it sends packets with increasing TTL values but that also have certain IP, UDP, or TCP fields and options set. By comparing the quoted packet to the original, one can highlight the modifications and the initial TTL value allows us to localize the two or more hops between which the change took place. In Fig. 5, packet `a` is the originally sent one. The first hop, that happens to be a middlebox, modifies its TCP Initial Sequence Number (ISN) and sends the rewritten packet `b` to the next hop. When the next hop receives the expired packet, it sends back to the client an ICMP `time-exceeded` packet `c` containing packet `b` as a payload. When the `tracebox` client receives it, it is able to compare packet `a` and the payload of packet `c` to detect any changes and the initial TTL value, i.e., 2, allows `tracebox` to bound the middlebox location.

It is worth to notice that in 80% of the cases [7], a path contains at least one router which implement RFC1812 [3], that recommends to quote the entire IP packet in the returned ICMP. This means that, in most cases, `tracebox` is able to detect any modification performed by upstream middleboxes.

`tracebox` works for IPv4, IPv6, and on Android devices [25] as described below.

2.2.1.1 TraceboxAndroid

Fig. 6 illustrates the general TraceboxAndroid architecture. As shown, it is made of three main components: the *system core* where the `tracebox` intelligence has been included (coded in C, under the front office), the *front office* (or the *application*) corresponding to the Android application (coded in Java) and the *back office* (or *server*) that is used to store data and make offline analysis. Currently, the back office is implemented in our own server. In the near future, we plan to directly communicated with the MAMI Observatory.

The front office communicates with the server using an XML API that gives the application the

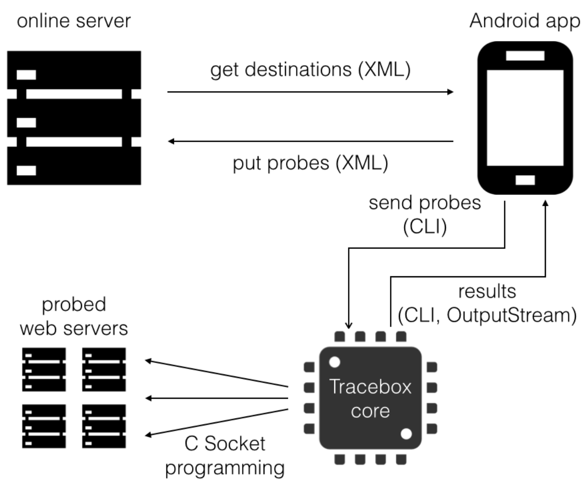


Figure 6: General overview of the TraceboxAndroid architecture

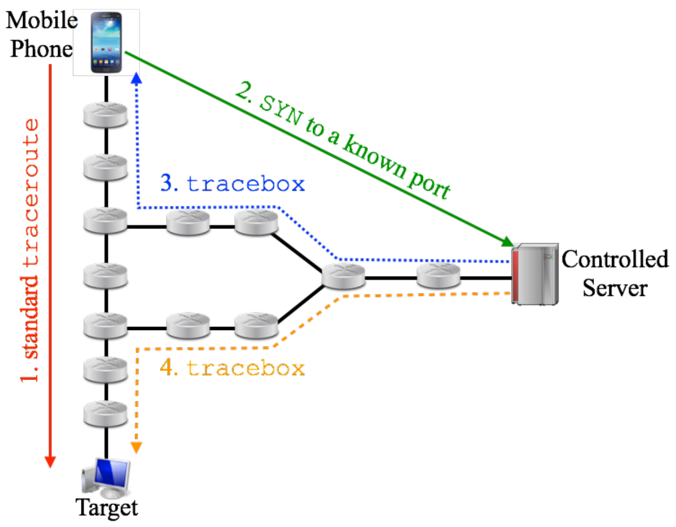


Figure 7: TraceboxAndroid on non rooted phones

destinations to be probed and allows it to send back the data collected by the system core. The core itself implements `tracebox` and sends probes to the destinations using sockets by system calls.

However, the main limitation for an Android app is that it is not possible to forge network and transport headers or read ICMP control messages in non-rooted environments. We would need raw sockets to anable this and their use is restricted to users that can grant the `CAP_NET_RAW` POSIX capability (i.e., super users). To overcome this issue, we have extended the `tracebox` methodology for TraceboxAndroid in order to be used on non-rooted phones. Fig. 7 illustrates our solution. It works in four steps and considers three entities: the non-rooted phone, the target, and a controlled server.

1. The non-rooted phone performs a standard `traceroute` towards the destination (this does

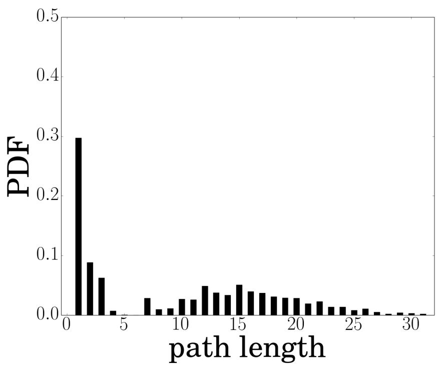


Figure 8: Path length distribution.

not require the phone to be rooted). It thus reveals the presence of routers along the path.

2. The non-rooted phone sends a `SYN` packet to a known port towards the controlled server.

3. In response, the controlled server sends `SYN+ACK` packets in `tracebox` fashion with increasing TTLs. This allows the server to discover routers between itself and the non-rooted phone but also to collect packet modifications caused by middleboxes on the path segment shared between step 1 and this step.[6] Modifications caused by middleboxes that are not in the path of interest (i.e., between the non-rooted phone and the target) are discarded.

4. Finally, the controlled server sends a regular `tracebox` to the target. This allows the server to discover routers between itself and the target, but also to collect packet modifications by middleboxes on the path segment shared between step 1 and this step. Similar as in step 3, modifications not appearing in the path segment of interest are discarded.

This mechanism, in four steps, is called *Triangle `tracebox`*.

After the Triangle `tracebox`, the phone is able to display packet modifications that happens on the first part of the path, on the last part of the path, and modifications of phone's `SYN` packet (excluding those that are only made on the path between the phone and the controlled server). Additional statistics about hops in the middle of the path could potentially be retrieved from the measurement data that other, rooted phones have previously already collected about those routers.

TraceboxAndroid as presented in this section is directly available from the Google Play Store [34].

2.2.1.2 Proxy Detection

A proxy is a special kind of middlebox that is used as an application relay. It plays the role of the server for the client, and the client for the server. Benefits of using a proxy, for an operator, are many:

- It prevents direct connections from an internal network towards the Internet;

[6]This is possible due to tree-like structure of routes [10].

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

standard `tracebox`	Triangle `tracebox`
`SYN+ACK` received at hop ≤ 5	TTL received $>$ TTL sent `SYN` not received on server side, but `SYN+ACK` received on the non-rooted phone side

Table 1: Summary of proxy detection techniques.

- It can analyze data within the application's context and filter if required (URL or DNS blacklists, keyword filtering, etc.);
- It can reformat pages (for smartphones, tablets);
- It may provide caching: the proxy can keep a local copy of content that it has fetched and, when another client asks for the same content, it can directly deliver the local copy leading to faster load times for the client.
- It supports anonymous surfing (as the user IP address is hidden).

Typical types of proxies are transparent proxies (i.e., a proxy that does not modify the HTTP request or response beyond what is required by the proxy authentication and identification [13]), FTP proxies, SMTP proxies, or DNS proxies.

Fig. 8 plots path length distribution as observed by a tracebox vantage point. The plot can be separated in two parts: below 5 hops and above 5 hops. Above 5 hops, we more or less observe the familiar bell-shaped curve typical of Internet interface distance distributions [10], with an "average" path length of 15 hops. Below 5 hops, where lies most of the distribution, we suspect the presence of a proxy replying in place of the server.

As a consequence, if a `tracebox` vantage point receives a response (`SYN+ACK` in case of TCP) for a TTL sent lower (or equal) to 5, we infer the presence of a proxy (more likely a transparent proxy).

This technique can be used in both standard `tracebox` and TraceboxAndroid. In addition, Triangle `tracebox` comes with additional features that allows to improve our proxy detection

1. TTL of received packet on the non-rooted phone. If this TTL is higher than the one sent, we advocate the TTL field has been rewritten along the path, typically by a proxy. However, in some cases, proxies delay `SYN`, so the packet cannot be received by the controlled server. In that case, the next method can be considered.
2. `SYN+ACK` received by the phone but the controlled server never sent a `SYN+ACK`. In that case, it means that the proxy answers every `SYN` (to speedup connections) before forwarding them to the actual destination. We can observe such a case as the server is under control.

Table 1 summaries proxy detection techniques with `tracebox`.

2.2.1.3 NAT Detection

Normally, Network Address Translators (NATs) are quite transparent to `traceroute`. They are supposed to modify source IP address and port on the outgoing packet but restore them on the incoming packets. They do this even in the quoted packet inside an ICMP `time-exceeded`.

However, NATs do not only have to change port number and IP address but they also have to update IP and TCP/UDP checksums to be consistent. It does happen that NATs do not update checksums in `time-exceeded` message. Thus, analyzing the checksums of the quoted packet in the received `time-exceeded` message identifies certain NATs along the path.

2.2.2 NAT `Revelio`

In terms of IP addressing, home networks generally use private IP address space where a home gateway performs Network Address Translation (home-NAT) from the private addresses within the home network to the addresses used in the ISP access network which may be public, private or shared [33]. In some cases, end-users can configure several different realms of private addresses within their home network in the context of cascaded home NATs. Independently of the home network topology, when a host within the home network communicates with a host in the rest of the Internet, the private address used by the host in the home network translates to a public address, the Globally Routable Address (GRA). For the majority of the residential Internet market, the ISP configures the GRA on the Internet-facing interface of the Costumer Premises Equipment (CPE) and the Network Address Translation (NAT) function in the CPE translates from the private addresses in the home network to the GRA [15]. An alternative, incipient, setup is one including an additional NAT function that operates in the ISP network (in addition to the NAT function in the CPE) and performs the final translation to the GRA. These configurations are usually called Carrier Grade NAT (CGN), Large Scale NAT (LSN) or NAT444. In this case, packets flowing between the home network and the Internet go through two upstream NAT-capable devices: the CPE (customer grade NAT) and the ISP NAT (Carrier-Grade NAT).

As a consequence, CGNs might represent an approach to prolonging the life of current address allocations, where ISPs share the same public IPv4 address across multiple end users. However, CGNs may introduce a number of issues for end users, service providers, and content providers. There is some evidence that CGNs can cause dropped services in peer-to-peer applications, and lead to low performance of file transfer and video streaming sessions [9, 1]. CGNs also introduce security challenges including traceability of IP addresses and anti-spoofing. Despite these challenges, CGNs offer an immediate relief to the IPv4 address scarcity problem, so it is likely that their popularity will increase or at least remain stable over time.

The goal of NAT `Revelio`[7] is to detect CGNs by distinguishing whether the NAT function translating to the GRA is located within the home network or in the ISP network. In Fig. 9 we depict the residential setup we consider to further explain the NAT `Revelio` methodology. The home network may have an arbitrary topology consisting of multiple hosts, routers and switches including multiple levels of NATs. The home network connects to the Internet through the CPE also known as “home router” or “home gateway”. The access link connects the CPE with the ISP access network. The ISP network then connects with the rest of the Internet.

In order to discern where the translation to the GRA occurs, NAT `Revelio` performs active tests from a device connected to the home network. The *probe* running NAT `Revelio` connects to the home network and may or may not be directly connected to the CPE, i.e., there may be multiple

[7]Sources are freely available at `https://github.com/mami-project/revelio`

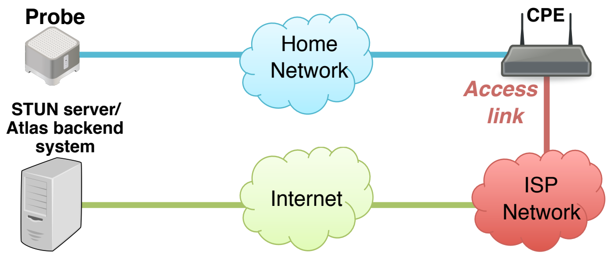


Figure 9: NAT Revelio assumed experimental setup.

hops, including ones performing NAT function(s), between the *probe* and the CPE. The target of the active tests are servers located in the Internet (Fig. 9). NAT Revelio does not require any cooperation from the ISP beyond forwarding Internet packets to and from the customer.

As mentioned earlier, the purpose of NAT Revelio is to detect whether the device performing the translation to the GRA (hereinafter GRA-NAT) resides in the home network or in the ISP network. In order to do this, NAT Revelio attempts to pinpoint the location of the GRA-NAT with respect to the access link. If the GRA-NAT lies between the *probe* and the CPE, we conclude that the user is not behind a CGN. If the GRA-NAT lies after the CPE, we conclude that the ISP deploys CGN. In order to achieve this, NAT Revelio needs to determine the location of the GRA-NAT and the location of the access link with respect to the *probe* and compare them.

2.2.2.1 Initial NAT Revelio Tests

To determine the location of the GRA-NAT, NAT Revelio performs the following steps:

1. It discovers the GRA by running STUN [22] against a public STUN Server located in the Internet (for the case of SamKnows) or by using the RIPE Atlas API[8] (for the case of Atlas);

2. It runs a traceroute to the GRA, computing the number of hops to the GRA-NAT;

The determination of the location of the access link is challenging because we aim to support arbitrary topologies in the home network and we do not have any prior information about where in the home network the *probe* connects. To determine the location of the access link we make the following assumption: the propagation delay of the access link is at least one order of magnitude higher than the propagation delay of the links in the home network. We believe this is a realistic assumption for the different access technologies and home network technologies available in the market and it is supported by existing empirical evidence [24]. In order to locate the access link, we estimate the propagation delay of different links between the *probe* and an arbitrary target server in the Internet using pathchar [11]. pathchar is a well-known technique for estimating transmission and propagation delays along the path using traceroute. Since we only need to estimate the order of magnitude of the delays, the precision of pathchar is sufficient. Using the propagation delays measured by pathchar, we determine the access link as the first one with a propagation delay at least one order of magnitude higher than the previous links.

[8]See section 3.6.1 for NAT Revelio deployment details.

By comparing the respective locations obtained for the GRA-NAT and for the access link, we can establish whether the GRA-NAT is located *before* the access link (no CGN) or *after* the access link (ISP uses CGN).

2.2.2.2 Supplementary NAT `Revelio` Tests

In addition to the main detection tests, the NAT `Revelio` client performs two other tests to gain additional insight about the presence of CGNs:

Invoke UPnP actions: If the *probe* directly connects to the CPE (i.e., the access link is one hop away), NAT `Revelio` tries to run the UPnP protocol [27] from the *probe* to retrieve the IP address of the WAN-facing interface of the CPE. If this IP address matches the GRA, we conclude that there is no CGN. Otherwise, if the UPnP query returns a private/shared address, NAT `Revelio` detects an upstream CGN.

Private/shared addresses along the path: To detect the access link location, NAT `Revelio` runs multiple traceroute measurements to a fixed target. This enables us to retrieve the IP addresses operators configure in their network. We then search for private and shared addresses after the access link. The detection of private/shared addresses after the access link alone does not imply the presence of an upstream CGN, but it serves as a hint that the ISP might be operating one. In particular, the presence of shared addresses after the access link provides a stronger indication about the presence of CGNs, because this address block is specifically reserved for CGN deployment [33].

2.2.2.3 Evolving NAT `Revelio`

After acquiring additional operational experience with NAT `Revelio` and communicating with several of the ISPs we measured, we identified some corner cases that may confuse NAT `Revelio`. To tackle this particular issue and increase the robustness of NAT `Revelio` to non-standard home network topologies, we enhanced the original NAT `Revelio` methodology by adding the following two tests to the test-suite.

Expected access technology delay. In some cases, we have detected that the propagation delay of the different links within the home network differs in one order of magnitude (e.g. one link with a delay of tens of μs and another one with delay in the hundreds of μs). In this case, the delays of both home network links are still one order of magnitude less than the propagation delay of the access link. In order to deal with this case, we define an expected range for the access link delay based on the access technology and we verify if the access link delay we measure falls within the expected range. If this is not the case, we mark the first link that falls within the expected range as the access link.

Pathchar to the GRA. We identified some home gateways that generated replies to traceroute as if they were two hops. Namely, when processing traceroute packets from the *probe*, they generate one reply from the internal interface (for packets with TTL=n) and a second reply from the external WAN-facing interface that assigns the GRA (for packets with TTL=$n+1$). We detected this behavior in several models of home routers, including SpeedPort and FritzBox. This behavior breaks the NAT `Revelio` methodology because the GRA appears to be one hop farther from the *probe* than it is. In order to detect these cases, we run `pathchar` from the *probe* to the GRA and we contrast the results with the ones we obtain running `pathchar` from the *probe* to the external server. If the CPE generates two replies to the traceroute to the GRA, the delay of the spurious link from the CPE to the GRA measured by the `pathchar` to the GRA is significantly smaller than the delay of the subsequent ”real” link measured by the `pathchar` to the external server. This is so because the spurious link is internal to the CPE, while the subsequent link is the actual access link. By comparing the two `pathchar` results, we can identify these anomalous CPEs and correctly identify the GRA at hop n.

3 Measurement Campaigns

In this chapter, we report early efforts on deploying measurement tools described in Chap. 2. These deployments were mostly done in parallel to the tool development and refinement, resulting in separate measurement studies. Integration into the Path Transparency Observatory is currently in progress.

Based on the order of discussion in the previous chapter, we first discuss path impairment as detected by `PATHspider` (Sec. 3.1) and `copycat` (Sec. 3.2). In addition to our copycat measurements, we further provide results on UDP blocking based on RIPE Atlas data (Sec. 3.3), and then discuss HTTP/2 comparison results (Sec. 3.4). Afterwards, we present our results on deteced middleboxes by `tracebox` (Sec. 3.5) and NAT `Revelio` (Sec. 3.6).

3.1 `PATHspider` Deployment

3.1.1 Measurement Setup

Our initial measurements with `PATHspider` have largely been directed at looking at path transparency on service provider, as opposed to access provider, networks. To this end, we have deployed `PATHspider` on a set of virtual machines on the network of cloud provider Digital Ocean, and focused our measurements on targets taken from the Alexa top million websites list. We have performed four separate measurement campaigns: one each measuring ECN (Sec. 3.1.2.1), DSCP (Sec. 3.1.2.2), TFO (Sec. 3.1.2.3), and HTTP/2 (Sec. 3.1.2.4).

3.1.2 Initial Results

3.1.2.1 ECN Connectivity and Negotiation

We performed measurements in June 2016 to revisit our earlier measurements in our PAM paper describing measurements taken in September 2014 [26], to determine whether ECN support in the Internet had changed.[1]

Indeed, we noted a continuation of the mostly-linear trend of ECN negotiation adoption, in that 432544 of 617873 (70.005%) of IPv4 addresses and 20262 of 24472 (82.797%) IPv6 addresses negotiated ECN as of the measurement campaign. However, we noted that the proportion of servers requiring fallback has not changed appreciably since our 2014 measurements: 0.44% of IPv4 and 0.11% of IPv6 servers. This reflects the two different forces at work: ECN support on the server side generally follows the operating system defaults, and web hosting machines generally run a recent Linux, the first operating system with server side ECN on by default. Connectivity problems, however, are often a function of faulty middleboxes, which are more slowly replaced, or firewall rules explicitly disabling ECN traffic for dubious reasons.

[1] The content of this section is taken from our MAMI project blog post at `https://mami-project.eu/index.php/2016/06/13/70-of-popular-web-sites-support-ecn/`; this work was cited at an Apple Worldwide Developer's Conference address on modern networking, available at `https://developer.apple.com/videos/play/wwdc2016/714/`

3.1.2.2 DSCP

We ran PATHspider to test DSCP codepoint rewriting from seven vantage points hosted by Digital Ocean in Amsterdam, Frankfurt, London, New York, San Francisco, Singapore and Toronto on the 30th September 2016[2]. Connections were attempted to each of the 673,230 IP addresses from each of these vantage points to give a total of 4,712,610 paths.

For the 3,523,405 paths where the baseline – outgoing DSCP value of zero – connection succeeded, we found that 3,519,626 (99.89% of those succeeding with the baseline connection) of experimental connections to IPv4 hosts succeeded with the outgoing DSCP value set to 46 (EF) and 136,469 (99.93%) of experimental connections to IPv6 hosts also succeeded with the same code point. For all hosts, we only saw code point dependent connectivity loss on 3,965 (0.11%) of paths. Additional analysis of this measurement run, also in comparison to further measurement runs planned for the beginning of 2017, are in preparation for publication.

We attempted to run this measurement from further vantage points within Microsoft's Azure cloud platform, but discovered that the DS field is bleached by Azure's network on ingress, so no meaningful results could be collected.

3.1.2.3 TCP Fast Open

We ran PATHspider to test TCP Fast Open connectivity and functionality from six vantage points hosted by Digital Ocean in Amsterdam, Frankfurt, London, San Francisco, Singapore, and Toronto on 12-13 October 2016[3]. PATHspider allows us to classify connection attempts with TFO as follows, in increasing order of brokenness:

- **TFO works**: TFO cookie received, data on SYN+ACK
- **TFO data not ACKed**: TFO cookie received, but only SYN+ACK.
- **TFO data failure**: TFO cookie received, SYN with data fails
- **TFO not negotiated**: TFO negotiation results in non-TFO connection.
- **TFO connection failure**: TFO option breaks connectivity (RST or drop).
- **Connection failure**: No connection attempt to target succeeded.

Our results are summarized in table 2 showing a total of 513 IPV4 and 50 IPv6 hosts supporting TFO of which 83% to 88% are operated by Google.

3.1.2.4 HTTP/2 Discovery via ALPN/NPN

We ran PATHspider to test seven vantage points hosted by Digital Ocean in Amsterdam, Frankfurt, London, New York, San Francisco, Singapore and Toronto on the 8th October 2016 for ALPN and on the 11th October 2016 for NPN. Connections were, again, attempted to each

[2]The content of this section is taken from a paper currently under submission to PAM 2017
[3]The content of this section is taken from a paper currently under submission to PAM 2017. Analysis and instructions for accessing raw data for this study are available at `https://github.com/mami-project/must-go-faster`

IPv4		IPv6		
hosts	pct	hosts	pct	description
15818	2.55%	2959	5.62%	Completely failed to connect
208	0.03%	3	0.01%	Failed to connect w/TFO option
4	<0.01%	0	0.00%	...consistently (non-transient)
604023	97.34%	49658	94.28%	Did not negotiate TFO
528	**0.09%**	**50**	**0.10%**	Negotiated TFO (exchanged a cookie); of which:
513	0.38%	50	100.00%	ACKed data on `SYN`
0	0.00%	0	0.00%	Failed connection with data on `SYN`
14	2.65%	2	4.00%	Returned a cookie on ACKed data
11	2.08%	0	0.00%	Responded with a 6-byte cookie
15	2.84%	0	0.00%	Responded with an experimental option
441	83.52%	44	88.00%	are in AS15169 (Google)

Table 2: TFO summary statistics, of 620560 IPv4 hosts and 52670 IPv6 hosts tested on 12-13 October 2016.

of the 673,230 IP addresses from each of these vantage points. We discovered that 381,062 (56.60%) of the distinct hosts would accept connections on TCP port 443. This is, as expected, less target measurements points than in the previous sections but still provides a total of 2,667,434 distinct paths to use for measurement.

For the 2,667,434 paths where a listening TCP port was observed for HTTPS, we saw that on 1,154,130 (43.26% compared to 12.1% in October 2015 [30]) of those paths we successfully negotiated NPN. Further we saw that on 842,245 (31.57% compared to 5.1% in October 2015 [30]) paths we successfully negotiated ALPN. Again, additional analysis of this measurement run, also in comparison to further measurement runs planned for the beginning of 2017, are in preparation for publication.

3.1.3 Future Deployment

Most of our initial measurements were limited to relatively unimpaired access networks: indeed, Digital Ocean was chosen as a provider to host `PATHspider` tests precisely because its network is essentially unimpaired (v6 support, no NAT, no blocking before the virtual host firewall), allowing us to isolate impairments on content provider networks and within the Internet core. Future planned deployment (as part of Task 1.3 starting beginning of 2017) on the MONROE wireless testbed will allow us to measure similar impairments on a variety of fixed and mobile access networks to get a better picture of the other side of path transparency impairment.

3.2 `copycat` Deployment

3.2.1 Measurement Setup

We deployed `copycat` on the PlanetLab distributed testbed on the entire pool (153) of available nodes between March 6th and April 23rd, 2016. Considering PlanetLab port binding restrictions (e.g., 80, 8000, and 53, 443 on certain nodes), we chose seven ports-53, 443, 8008, 12345,

Dataset	Throughput (kB/s)				Latency (ms)			
	< 200		> 200		< 50		> 50	
	# flows	median	# flows	median	# flows	median	# flows	median
PlanetLab	740,721	0.05	34,896	0.16	745,947	0.00	29,370	-1.65
DO v4	12,563	0.03	3,637	-0.37	9,381	-0.02	6,819	-0.44
DO v6	15,459	0.07	224	-0.16	15,656	0.00	27	3.63

Table 3: Raw number of bias measurements (throughput and initial latency) per sub dataset ("DO" stands for Digital Ocean). The 50ms cut-off roughly corresponds to inter-continental versus intra-continental latency

33435, 34567, and 54321- respectively DNS, HTTPS, HTTP alternate, a common backdoor, the RIPE Atlas UDP traceroute default (see next section), an unused and an unassigned port, to maximize routers policy diversity. For each port and pair of nodes, we generated 20 pairs of flows of 1, 3, and 30 TCP initial windows, and 10 pairs of flows of 300 and $1,500$ TCP initial windows of data to send, for a total of 4,908,650 flows.[4]

Then, we selected 93 nodes (one per subnetwork) from the entire pool to maximize path diversity. The selected nodes are located in 26 countries across North America (44), Europe (29), Asia (13), Oceania (4), and South America (3). The filtered PlanetLab dataset then consists of 1,634,518 flows.

We also deployed `copycat` on six Digital Ocean nodes, located in six countries across North America (2), Europe (3), and Asia (1). Given the less restrictive port binding policies and the more restrictive bandwidth occupation policies, we tested ports 80 and 8000 in addition of the PlanetLab ports. For each port, we generated 20 pairs of flows of 1, 3, and 30 TCP initial windows size between May 2nd and 12th, 2016. We repeated the same methodology for both IPv4 and IPv6. This dataset consists in 32,400 IPv4 and 31,366 IPv6 flows.

3.2.2 Initial Results

To evaluate the impact of transport-based differential treatment on throughput, we introduce the relative $throughput\_bias$ metric for each pairs of concurrent flows. This is computed as follows:

$$throughput\_bias = \frac{(throughput_{udp} - throughput_{tcp})}{min(throughput_{tcp}, throughput_{udp})} \times 100. \tag{3.1}$$

A positive value for $throughput\_bias$ means that UDP has a higher throughput. A null value means that both UDP and TCP flows share the same throughput.

Fig. 10 provides a global view of the $throughput\_bias$. Dataset has been split between flows $<$ 200 KB/sec and flows $>$ 200KB/sec, except for Digital Ocean IPv6, as the number of measurements is too small to be representative. Table 3 gives the size of each sub dataset and the relative median bias for throughput and latency.

For both Digital Ocean dataset, the non-null biases are mostly evenly distributed in favor and disfavor of UDP. In PlanetLab, we observe an extreme case where TCP performs better than UDP, the 4% and 2% highest $throughput\_bias$ in absolute value are respectively higher than 1% and 10%.

[4]The complete dataset is freely available at `http://queen.run.montefiore.ulg.ac.be/~edeline/copycat/`.

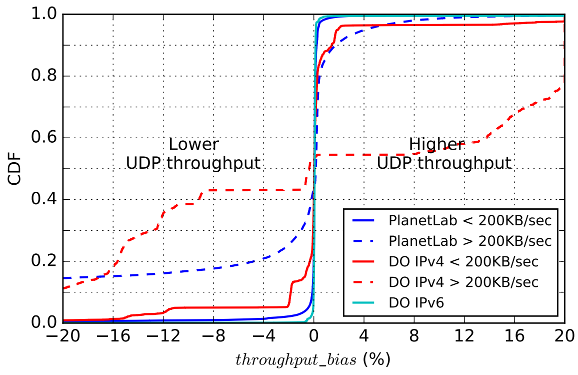


Figure 10: Relative throughput bias, as measured by `copycat` ("DO" stands for Digital Ocean). Positive values mean UDP has higher throughput. DO IPv6 has not been split in two due to lack of enough values (see Table 3).

The loss rate of congestion controlled traffic in steady state, where the link is fully utilized, is mostly determined by the congestion control algorithm itself. Therefore, there is a direct relation between throughput and loss. However, as TCP congestion control reacts only once per RTT to loss as an input signal, the actual loss rate could still be different even if similar throughput is achieved. Here, we understand *loss* as the percentage of flow payload lost, computed from sequence numbers. A value, for instance, of 10% of losses means thus that 10% of the flow payload has been lost.

Generally speaking, the loss encountered is quite low, given that small flows often are not large enough to fully utilize the measured bottleneck link. As expected based on he throughput observed, we see no significant loss difference in both PlanetLab and Digital Ocean when comparing TCP and UDP, except of 3.5% in favor of UDP for the largest flow size (6MB). However, since all `copycat` traffic is congestion controlled, throughput is also influenced by the end-to-end latency. As discussed next, we can correlate this slightly lower throughput with a slightly larger initial RTT, where we use initial RTT measured during the TCP handshake as baseline for the end-to-end latency.

In the fashion of $throughput\_bias$ (see Eqn. 3.1), we introduce the relative RTT_bias metric for each pair of concurrent flows. This is computed as follows:

$$RTT\_bias = \frac{(RTT_{tcp} - RTT_{udp})}{min(RTT_{tcp}, RTT_{udp})} \times 100. \quad (3.2)$$

A positive value for RTT_bias means that UDP has a smaller initial latency (i.e., performs better than TCP). A value of zero means that both UDP and TCP flows share the same initial latency.

The median latency bias is also listed in in Table 3 (right part). For PlanetLab, there is no latency bias for flows with an initial RTT of 50ms or less and a slight bias towards higher latency for UDP for flows with larger initial RTTs. For Digital Ocean we also observed a slight bias towards higher latency for UDP for IPv4 and no bias for IPv6 (considering 27 flows with a larger RTT than 50ms as not representative). This is confirmed by the CDF shown in Fig. 11.

The 2% and 1% most biased flow pairs have an RTT_bias respectively lower than -1% and -10%. For the Digital Ocean IPv4 campaign, 40% of the generated flows have an RTT_bias

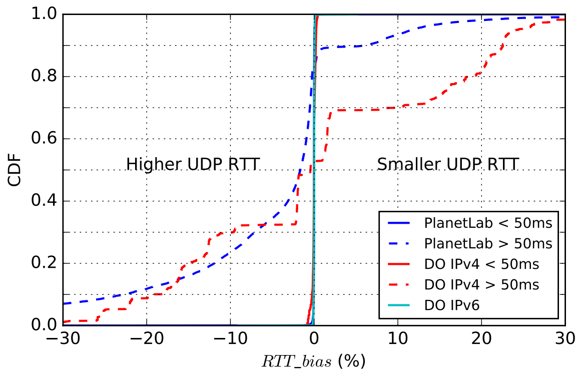


Figure 11: UDP/TCP initial RTT_bias as measured by `copycat` ("DO" stands for Digital Ocean). Positive values mean UDP is faster. DO IPv6 and Atlas have not been split in two due to lack of enough values (see Table 3).

between 1% and 30% in absolute value. The difference between IPv4 and IPv6 on Digital Ocean appears to be due to the presence of a middlebox interfering with all IPv4 traffic, both TCP and UDP.

3.3 RIPE Atlas Measurement

3.3.1 Measurement Setup

We used the RIPE Atlas [21] measurement network to provide another view (in addition to the measurements performed with `copycat` as presented in the previous section) on UDP blockage on access networks and possible MTU effects on such UDP blockage. We compared UDP `traceroute` and TCP `traceroute` measurements from a set of 115 Atlas probes in 110 networks (identified by BGP AS number) to 32 Atlas anchors (i.e., Atlas nodes having higher measurement capacities than standard Atlas probes). The measurements ran between September 23rd and 26th, 2015, with all probes testing each anchor sequentially, sending three packets in a row once every twenty minutes, for up to 17 connection attempts (51 packets) each for UDP/33435, TCP/33435, and TCP/80.

To review, TCP `traceroute` sends `SYN`s with successively increasing TTL values and observes the ICMP `time-exceeded` responses from routers along the path and the `SYN+ACK` or `RST` from the target. UDP `traceroute` sends packets to a UDP port on which presumably nobody is listening, and waits for ICMP `time-exceeded` or `destination-unreachable` responses from the path and target respectively.

For the measurements that we initiated on the RIPE Atlas platform we set the initial TTL to 199, which is sufficient to reach the destination in one run without generating any `time-exceeded` messages from the path, i.e., treating `traceroute` as a simple ping.

As further described below we found that, while "common knowledge" holds that some networks severely limit or completely block UDP traffic, this is not the case on any of the selected probes in this first measurement. To get a handle on the prevalence of such UDP-blocking networks, we also looked at 1.1 million RIPE Atlas UDP `traceroute` measurements run in 2015, including those from our first campaign. Here, we assume that probes which perform measurements against targets which are reachable by other probes using UDP traceroutes, but which never successfully complete a UDP `traceroute` themselves, are on UDP blocked access networks.

Further, we used a single campaign of about 2.5 million UDP and ICMP traceroutes from about 10,000 probes with different packet sizes in March 2016, to compare protocol-dependent path MTU to a specific RIPE Atlas anchor. We compared success rates with UDP at different packet sizes to ICMP.

3.3.2 Initial Results

Dataset	**Results**		
	# Probes		No UDP Connectivity
	total	failed	
Latency, 2015	110	0	0.00%
All UDP, 2015	2,240	82	3.66%
all MTU, March 2016	9,262	296	3.20%
72 bytes	9,111	244	2.68%
572 bytes	9,073	210	2.31%
1,454 bytes	8,952	137	1.53%

Table 4: Overview of our results on UDP connectivity. The upper part of the table shows the percentage of probes with UDP being blocked, as measured by RIPE Atlas in 2015 and 2016 (Sec. 3.3.2 for details).

Table 4 provides an overview on our main results. In summary, we show that, aside from blocking of UDP on certain ports, as well as relatively rare blocking of all UDP traffic on about one in thirty access networks, UDP is relatively unimpaired in the Internet.

Of the 2,240 RIPE Atlas probes which performed UDP `traceroute` measurements against targets which were reachable via UDP `traceroute` in 2015, 82 (3.66%) never successfully completed a UDP `traceroute`. We take this to be an indication that these probes are on UDP-blocking networks. The location of the blockage, determined by the maximum path length seen in a UDP `traceroute`, is variable, with the median probe seeing at least one response from the first five hops. These UDP-blocked probes are more likely than the population of all examined probes to be on networks in sub-saharan African and east Asian countries.

Our investigation of MTU issues showed no significant relationship between packet size and probe reachability up to 1,420 bytes per packet, as compared to ICMP. In this shorter study in March 2016 using more probes, 296 of 9,262 probes (3.20%) did not receive a response from the target from the UDP `traceroute` for any packet size. For 72, 572, and 1,454 byte packets, respectively, 2.68%, 2.31%, and 1.53% of probes received no response to a UDP `traceroute` attempt when receiving an response from an ICMP `traceroute` of the same size. These results are summarized in Table 4. Note that the relative UDP blocking numbers go down as the packet size goes up; this is because large ICMP packets are more often blocked than large UDP packets. From these results, we conclude that differential treatment between

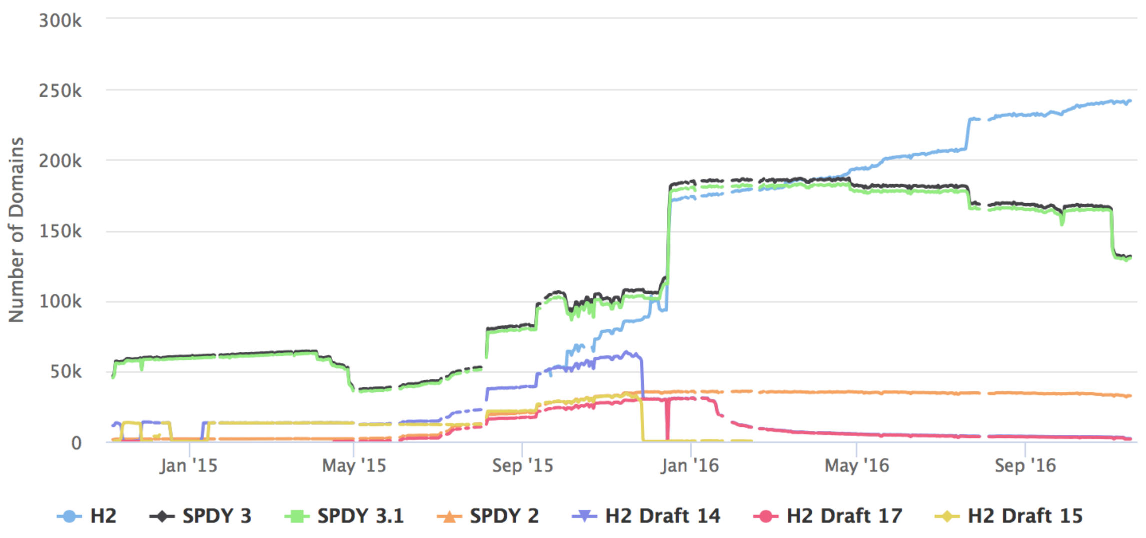


Figure 12: Plot of the data available at the HTTP/2 Dashboard

UDP and TCP should not pose a challenge to using UDP as an outer transport.

3.4 H2tool and EYEORG

3.4.1 Measurement Setup

The first measurements regarding HTTP/2 availability and h2/h2c middlebox impairments used the H2tool, according with these three phases:

- **Phase I**: Worker agents on PlanetLab probe a list of sites based on the top 1 million Alexa sites, using NPN and ALPN to determine which sites announce HTTP/2 support.
- **Phase II**: Worker agents in labs in Cleveland (USA) and Barcelona (Spain) attempt to fetch the root object for each site that claims to support HTTP/2. The result is a list of sites that partially and truly support HTTP/2..
- **Phase III**: Crawler agents in Barcelona, Cleveland, and Pittsburgh fetch each site that actually supports HTTP/2 using HTTP/1.1 and HTTP/2 and records performance and usage information like page load time and number of TCP connections used.

In addition, we captured videos of 100 top AlexaTop websites while loading over HTTP/1.1 and HTTP/2 for the EYEORG subjective QoE performance test. We repeated each load five times and kept the video with the median load time.

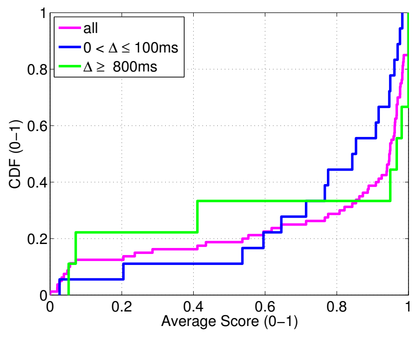


Figure 13: HTTP/1.1 vs HTTP/2.

3.4.2 Initial Results

The results of the HTTP/2 impairment measurements are currently available at the HTTP/2 Dashboard web site (`http://isthewebhttp2yet.com/`), which is the product of continuous data collection campaigns started before the beginning of the MAMI project, and enhanced during the project lifetime, as shown in Fig. 12. As previously noticed the `PATHspider` TLS plugin follows the same methodology as the used for the collection of data available at this site (see Sec. 3.1.2.4). The integration the this data into the MAMI Observatory is under currently under consideration, even with the possibilty of using `PATHspider` for inclusion into the HTTP/2 Dashboard.

In what relates to the `EYEORG` results, Fig. 13 shows the CDF of the average "score" per website; 0 means the HTTP/1.1 version was faster, 0.5 is a "split" decision, and 1 means the HTTP/2 version was faster. We plot scores for (i) all websites, (ii) websites with similar HTTP/1.1 and HTTP/2 page load times ($\Delta \leq 100$ms), and (iii) websites that loaded much faster over one protocol than the other ($\Delta \geq 800$ms). To build these subsets, we compute the page load time (PLT) using `SpeedIndex`.[5]

Fig. 13 shows that 70% of the websites have an average score of 0.8 or higher; this means that 70 out of 100 websites "feel" faster using HTTP/2 than HTTP/1.1. Conversely, 12% of the websites have an average score of 0.2 or lower and thus feel faster using HTTP/1.1. The remaining 18% of websites create some disagreement. Note that the score here does not take into account the "No Difference" responses. These websites with scores in the 0.2–0.8 range also have twice as many No Difference responses compared to the other websites. This further indicates that participants are just not sure which version was actually faster.

Next, we focus on the subset of websites with similar PLTs ($\Delta \leq 100$ms). Fig. 13 shows that participant indecision grows, with more scores in the 0.2 to 0.8 range. On the other hand, when $\Delta \geq 800$ms, participants mostly agree on which version was faster. This result indicates that, while aiming at reducing loading time of a webpage is overall beneficial, many users are not able to appreciate the difference when only few hundred milliseconds are saved.

[5]SpeedIndex defines PLT as the average time at which above-the-fold content is displayed. See `https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index`

Horizon 2020
European Union funding
for Research & Innovation

Network type	rooted	unrooted	both
Cellular	28	34	22
WiFi	38	81	40

Table 5: Subdataset considered for TraceboxAndroid analysis. It provides the raw number of device type (rooted, unrooted or both) of each kind of network connection (WiFi or Cellular).

3.5 `tracebox` Deployment

3.5.1 Measurement Setup

We deployed standard `tracebox` (see Sec. 2.2.1) on wired IPv4 networks via PlanetLab. We selected the maximum number of nodes available for each campaigns (between 108 and 129). Destinations have been selected using the top 1M Alexa website. We conducted 15 campaigns over nine different ports (80, 8080, 8000, 8800, 443, 53, 12345, 1228, 34567) with different probes (different TCP options including `MSS`, `SACKP`, and `SACK`) for a period of two months between March, 3rd and May, 8th 2016, each campaign lasting between three and sever days. The corresponding 1.3TB dataset has been uploaded to the Observatory raw data database in json format. Further analysis of these data in the Observatory are in progress.

In parallel to the measurement of standard `tracebox`, we also deployed (through crowdsourcing[6]) TraceboxAndroid (see Sec. 2.2.1.1). We were able to collect data from 214 users, scattered in 45 countries. Those users are served by 80 different carriers. We collected information on probes sent on both cellular and WiFi networks. We removed from the dataset inconsistent data (i.e., the mobile device was not able to send enough data to infer statistics). Table 5 summarizes the dataset we use in the following.

3.5.2 Initial Results

3.5.2.1 Standard `tracebox`

We first selected the nodes that remained available during all campaigns (89). During the entire measurement campaign, we observed 955,806 different responsive hops (excluding vantage points and targets addresses), parts of 2,978 different ASes. The more represented ASes are Cogent (35.7% of all addresses – Tier 1 network), CenturyLink (10.6% – Tier 1 network), Telia Carrier (6.3% – Tier 1 network), NTT (3.4% – Tier 1 network), Rackspace (1.8%), Level3 (1.6% – Tier 1 network), and Chinanet (1.5%). The corresponding addresses are geographically distributed in North America (40.4%), Europe (37.5%), Asia (18.7%), Latin America and Caribbean (2.7%), and Africa (0.7%) according to the regional Internet registries. The same addresses were registered under 189 different country codes.

We analyzed the responsive hops (addresses from which we received at least one packet) over time, on ports 80 and 443 only because firewalls blocked probes on other ports (even non standard HTTP) and avoided nodes located behind them to answer. 86.1% of all hops were

[6]Faggiani et al. [12] has already proven smartphone-based crowdsourcing can be a viable strategy for measurements by deploying a `traceroute` related Android application inside a wide academic and research network.

TCP Port	Service	Cellular	WiFi
21	FTP	3	0
25	SMTP	1	0
80	HTTP	20	7
443	HTTPS	6	0
5060	SIP	1	2
10000	-	1	0

Table 6: Number of networks in which TCP is proxied by port.

responsive during all 6 campaigns on ports 80 and 443. 2.4% of all hops were responsive during at least the first two campaigns, then disappeared. 0.75% disappeared in the same way, but then re-appeared for at least on campaign. 0.75% of all hops disappeared and re-appeared more than once. 1.9% of all nodes were not responsive during the first campaign, but appeared later and stayed responsive until the end. 2.4% only responded during one campaigns out of 6.

We analyzed the Autonomous Systems of the responsive IP addresses and found that out of the 2,978 different observed ASs, 98.2% (2,924) of them were responsive (we received at least one packet from one address parts of this AS) during all campaigns. Seven ASes disappeared after at least two consecutive campaigns and 11 appeared between campaigns, while 19 disappeared to re-appear later. Seven other ASes were only responsive during one single campaign. One AS was only responsive to probes on port 80, and one to probes on port 443.

3.5.2.2 Detecting Proxies with AndroidTracebox

Table 6 the number of networks, for both rooted and unrooted devices, in which TCP ports are proxied (see Sec. 2.2.1.2 to see how we detect the presence of a proxy). We did not test all possible ports but, rather, focus on some well known ones (FTP, STMP, HTTP) and a unreserved one (10000). We observe, unsurprisingly from Table 6 that HTTP traffic is the most proxied one.

We have observed interesting behaviors between the proxy and the packet actual destination. For instance, we were able to detect TCP coalescing between the proxy and the server by sending multiple probes with only 1 byte of payload, the server receiving a single packet with all those bytes as payload. We were also able to detect that different TCP options were set between the pair (client, proxy) and the pair (proxy, server). This is done as follows: the server sends a `SYN+ACK` with no `timestamp` option but the client receives the `SYN+ACK` with `timestamp` option. As a consequence, the client sends back payload packet with `timestamp` option enabled. But every packet received on the server side has no `timestamp` option. It is like there is a middlebox somewhere in the path adding `timestamp` option on the `SYN+ACK` packet and, then, clearing the option on every client payload packet. Another explanation would be the connection being split in two parts: the client $\rightarrow$ proxy part (with the `timestamp` option enabled) and the proxy $\rightarrow$ server part (not considering the `timestamp` option). The same behavior happened with the TCP `window-scale` option.

Weaver et al. [32] detect HTTP proxies from modifications on HTTP request / response. Although we focus on transport proxies, we were able to detect one middlebox adding HTTP

network connection	**% probes sent**		
	IP	UDP	TCP
wifi	0.0	18.09	36.14
cellular	0.0	13.62	25.81

Table 7: NAT detection feasibility

headers without being proxied (meaning it did not change the TTL).

3.5.2.3 Detecting NATs with AndroidTracebox

Table 7 shows the feasibility of our mechanism to detect NATs. TCP or UDP checksum errors are more common than IP (IP checksum is more likely to be corrected even in quoted packets).

Wang et al. [31] assume the presence of NAT by looking at the client (i.e., phone) address: if it belongs to a reserved range, it means that there is a NAT. This is underestimating NAT presence as we discovered NATs even when client address is a Public IP address. Our technique is thus complementary to Wang et al. and we plan further measurement studies to achieve more comprehensive results.

3.6 NAT `Revelio` Deployment

3.6.1 Measurement Setup

The NAT `Revelio` methodology enables us to determine from within home networks the type of upstream network address translation, namely NAT at the home gateway (customer-grade NAT) or NAT in the ISP (Carrier Grade NAT).

With the help of a large UK ISP and a large Italian ISP, we tested NAT `Revelio` on a controlled set of 30 operational DSL accesses, 2 of which connected behind a trial CGN implementation in the same ISP. NAT `Revelio` accurately detected the upstream NAT configuration of all 30 connections.

We further deployed the NAT `Revelio` test suite on two hardware-based large-scale measurement platforms in Europe (RIPE Atlas) and the United States (FCC-MBA SamKnows), which allowed us to instrument 5,121 measurement vantage points in over 60 different ISPs worldwide. We scheduled the NAT `Revelio` client to run over 20 times on each *probe* during March 2016 for the FCC-MBA platform and during May 2016 in the RIPE Atlas platform. The data we collected from the *probes* in *each* run of NAT `Revelio` are the following: the Globally Routable Address (GRA), the mapped port number, traceroute results to the GRA, traceroute results to a fixed target address (with 21 different packet sizes), UPnP query result to retrieve the IP address on the external interface of the device to which the *probes* connect (only for SamKnows - *SK probes*).

In total, we collected data from 5,121 *probes* in 64 ISPs with an average of 20 repetitions[7] per

[7] In the FCC-MBA platform, in order not to interfere with normal user Internet activity, the *probes* perform cross-traffic sensing and run the tests we schedule only when they detect no end-user traffic. Thus, the number of NAT

probe which resulted in over 2 million traceroutes. The tested ISPs include 42 DSL providers, 16 cable providers, 4 ISPs that offer fiber to the home Internet connectivity and 2 satellite providers.

3.6.2 Initial Results

As a function of the upstream NAT configuration, NAT `Revelio` classifies each *probe* into one of the following cases:

1. *inconclusive* (cases where NAT `Revelio` was unable to draw any conclusion due to incomplete or inconsistent results),
2. *no home NAT* (i.e., the *probe* where NAT `Revelio` runs is directly connected to the public Internet),
3. *simple home NAT* (the CPE performs the GRA-NAT),
4. *Carrier Grade NAT* (the GRA-NAT is outside the home network, in the ISP's network)

We present the aggregate results of the inferred upstream NAT configuration in Table 8.

Inconclusive. For 1,276 *probes* (307 *SK probes* and 969 *Atlas probes*), NAT `Revelio` gave inconclusive results either because none of the tests could run on the *probe* or because we did not obtain enough information to properly interpret the results we were able to collect. Our approach is conservative and tags as inconclusive the case of mixed responses from different tests. For example, traceroute limitations and ICMP traffic being filtered along the path to the external target server hamper our capacity to identify the access link. Without knowing the location of the access link, when the end-user deploys several levels of NAT in the home, we cannot draw conclusions regarding the presence of NAT in the ISP. These *probes* account for approximately 24% of the total, (12% of the *SK probes* and 36% of the *Atlas probes*). We discard these cases for further analysis.

No home NAT. NAT `Revelio` found that in 299 different cases (85 in *SK probes* and 214 *Atlas probes*), the NAT `Revelio` client was running on a *probe* configured with a public IP address that was also the GRA. These *probes* were operating in the public Internet, which implies that the connections were not connected behind a NAT solution. In all these cases, the *traceroute to the GRA* test also confirmed the lack of a NAT solution in the corresponding ISPs.

Simple home NAT. Out of the rest, for 3,454 *probes* (2,009 *SK probes* and 1,445 *Atlas probes*) NAT `Revelio` established the presence of simple home NAT and excluded the possibility of further NAT in the ISP. NAT `Revelio` reports the simple home NAT configuration (and, thus, the lack of NAT in the ISP for the respective connection) when at least one of the *traceroute to GRA* and *invoking UPnP actions* tests establish that the home gateway is performing the GRA-NAT. In the case of the UPnP test, for 1,300 *SK probes* the address retrieved through UPnP from the CPE matched the GRA, concluding that the CPE was the GRA-NAT. For 815 *SK probes*, the NAT `Revelio` client was unable to communicate with the CPE through UPnP, either because the CPE did not support UPnP or because the *SK probe* was not directly connected to the CPE. In the case of the `traceroute` to the GRA test, for 2,965 *probes* (1,520 *SK probes* and 1,445 *Atlas probes*) NAT `Revelio` located the GRA-NAT before the access link, concluding that the

`Revelio` repetitions differs for various measurement vantage points.

ISP Name	CC	Tech.	# of probes	Inconclusive	Simple Home NAT	Carrier Grade NAT	Confirmed
1 (Undisclosed ISP)	US	Sat	76	0	0	76	Yes
2 (Kabel Deutschland)	DE	Cable	49	27	14	8	Partially
3 (Fastweb)	IT	Fiber	26	14	8	4	Yes
4 (OTE)	GR	DSL	21	5	14	2	No Reply
5 (Liberty Global)	NL	Cable	280	133	146	1	Yes
6 (Zen)	UK	DSL	32	11	20	1	No Reply

Table 8: NAT `Revelio` Results. ISPs with at least one probe with positive NAT `Revelio` result. We report the Country Code (CC), the access technology (Tech.), the total number of probes tested (# of probes), the number of probes for which NAT `Revelio` gave inconclusive results (Inconclusive), the number of probes NAT `Revelio` tested negative (Simple Home NAT), the number of probes NAT `Revelio` tested as positive (Carrier Grade NAT) and the current status of the confirmation from the ISP with positive NAT `Revelio` results (Confirmed). For the latter, we mark this field with *Yes* if the ISP confirmed the NAT `Revelio` results, *Partially* if the ISP confirmed they use CGN but did not confirm the specific accesses tested, *No Reply* if we did not get any feedback from the ISP.

CPE was also the GRA-NAT. As a interesting data point, using *pathchar to the GRA* test NAT `Revelio` purged 165 of cases where the CPE replied as being two different hops, creating false positives. In particular, NAT `Revelio` detected this behavior in one single ISP for 78 out of 228 *probes*.

Carrier Grade NAT. For 92 *probes* in 6 ISPs (76 *SK probes* in 1 ISP and 16 *Atlas probes* in 5 ISPs) NAT `Revelio` detected the presence of CGN technology in the ISP's network. Table 8 details the number of *probes* that tested positive for CGN per ISP[8]. We identified one satellite provider in the U.S. where all *probes* tested positive for CGN. For the rest of the ISPs, we detected a mix of some *probes* that tested positive for CGN and others that did not. Overall, about 2% of the *probes* tested positive for CGN. About 10% of the ISPs we tested hosted at least one *probe* that tested positive for CGN. Of these latter ones, only one ISP had a widespread deployment of CGN, while the other ISPs presented a few scattered *probes* that tested positive, hinting at a localized deployment, e.g., possibly for trials or suggesting a specific service.

We validated both the positive and negative results at the IP level through different means, including direct contacts with the involved ISPs or, in one case, using the WHOIS database information. We managed to get several positive and negative confirmations. In particular, for ISP#1 from Table 8 – the satellite provider in the US for which all *probes* tested positive – the operator confirmed that its normal configuration includes performing the NAT function in the ISP network and that all the 76 connections that tested positive were indeed behind a CGN. ISP#3 (Fastweb) confirmed both the positive and the negative NAT `Revelio` results. For ISP#5 (Liberty Global) from Table 8, the GRA associated with the *probe* is actually tagged in the WHOIS database (in the *Organization* field) as CGNAT (the other 279 *probes* in the same ISP did not have a GRA in the subnet marked as CGN). ISP#2 (Kabel Deutschland) from Table 8

[8]We only disclose the names of the ISPs we tested using the RIPE Atlas platform. We are currently pending the approval of the FCC for disclosing the names of the ISPs we tested with the FCC-MBA testbed.

confirmed that it is using CGN in its network. However, we did not obtain explicit confirmation that the exact accesses we detected as positive are actually behind a CGN, which is why we marked it as a *partial* confirmation. As for the negative results, we obtained validation from 4 ISPs for which all *probes* tested negative for upstream CGN in the ISP. We mention that (confirmed) negative results from NAT `Revelio` testing do not preclude the existence of CGN technology in the corresponding networks.

4 Conclusion

Our initial measurements indicate some guidance for the design of the MAMI Middlebox Cooperation Protocol (MCP) to be elaborated in future deliverable D3.2. First, measurements with `copycat` indicate that there is no widespread differential treatment of UDP and TCP based solely on protocol number, so using UDP as a substrate protocol to allow userspace implementation of MCP, as well as for port numbers for NAT binding, is a fundamentally sound architectural choice. Measurements with NAT `Revelio` and `tracebox`, showing the prevalence of carrier grade NAT and standard NATs, further confirm the need for a NAT-binding shim layer. However, measurements with Atlas, showing access-network-linked blockage of all UDP traffic on around 3% of Internet access networks, indicate that the Flexible Transport Layer (FTL) will require a fallback to TCP, TLS over TCP, or even HTTPS on networks with impaired connectivity.

Measurements taken with `tracebox` (both the standard and Android version) will also feed into middlebox classification work, to appear in future deliverable D2.1.

The next steps for the measurement work package include further synthesis of these measurement results using the *Path Transparency Observatory*, to be detailed in future deliverable D1.2, as well as scaling out to more diverse paths, larger sets of vantage points, and more frequent measurement to detect trends and provide information for middlebox behavior classification efforts. Path and vantage point diversity will be provided in part by measurements taken on the MONROE testbed. Measurement frequency will be improved by increasing automation of measurement campaigns. We have already begun the automation of ECN impairment measurements to generate finer trendlines in ECN deployment and the removal of ECN impairments from the Internet.

`PATHspider` development continues, as well. Plugins to be built in the coming year include generic TCP options testing, Stream Control Transmission Protocol (SCTP) [23], and Multipath TCP [14] usability and connectivity. SCTP connectivity tests will examine a fundamental assumption behind the MCP's design: that UDP is necessary as an encapsulation because new protocols do not deploy over IP. TCP options and MPTCP tests will round out examination of impairment of current proposed evolutions of the transport layer.

`copycat` development also continues. In the following months, we plan to make `copycat` more generic so that it would be able to compare any two transport protocols (not only TCP vs. TCP-inside-UDP), not only based on the protocol framing and potential modifications thereof but also evaluating QoS characteristics.

There are also opportunities to integrate multiple tools beyond the Observatory, e.g., integration between `PATHspider` and `tracebox` to allow the latter's measurement to localize interference detected by the former.

References

[1] B. Aitken. MC/159 report on the implications of carrier grande network address translators. Final report for ofcom, Ofcom, 2013.

[2] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's initial window. RFC 3390, Internet Engineering Task Force, October 2002.

[3] F. Baker. Requirements for IP version. RFC 1812, Internet Engineering Task Force, June 1995.

[4] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain. TCP fast open. RFC 7413, Internet Engineering Task Force, December 2014.

[5] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis. Increasing TCP's initial window. RFC 6928, Internet Engineering Task Force, April 2013.

[6] Crowdflower. AI for your business, 2016. see `https://www.crowdflower.com`.

[7] G. Detal, b. Hesmans, O. Bonaventure, Y. Vanaubel, and B. Donnet. Revealing middlebox interference with tracebox. In *Proc. ACM Internet Measurement Conference (IMC)*, October 2013.

[8] T. Dierks and E. Rescorla. The transport layer security (TLS) protocol version 1.2. RFC 5246, Internet Engineering Task Force, August 2008.

[9] C. Donley, L. Howard, V. Kuarsingh, J. Berg, and J. Doshi. Assessing the impact of carrier-grade NAT on network applications. RFC 7021, Internet Engineering Task Force, September 2013.

[10] B. Donnet, P. Raoult, T. Friedman, and M. Crovella. Efficient algorithms for large-scale topology discovery. In *Proc. ACM SIGMETRICS*, June 2005.

[11] A. B. Downey. Using pathchar to estimate Internet link characteristics. In *Proc. ACM SIGCOMM*, August 1999.

[12] A. Faggiani, E. Gregori, L. Lenzini, S. Mainardi, and A. Vecchio. On the feasibility of measurement the Internet through smartphone-based crowdsourcing. In *Proc. IEEE International Symposium on Modeling and Optimization in Mobile, Ad-Hoc and Wireless Networks (WiOpt)*, May 2012.

[13] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1. RFC 2616, Internet Engineering Task Force, June 1999.

[14] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP extensions for multipath operation with multiple addresses. RFC 6824, Internet Engineering Task Force, January 2013.

[15] S. M. Garland and D. B. Smith. Communication between service providers and customer premises equipment. Patent US Patent 6,167,042, Google Patents, December 2000.

[16] R. Kohavi and R. Longbotham. *Online Controlled Experiments and A/B Testing*, pages 1–8. Springer US, 2015.

[17] MicroWorker. Work and earn or offer a micro job, 2009–2016. See `https://microworkers.com`.

[18] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers. RFC 2474, Internet Engineering Task Force, December 1998.

[19] J. Postel. Internet control message protocol. RFC 792, Internet Engineering Task Force, September 1981.

[20] K. Ramakrishnan, S. Floyd, and D. Black. The addition of explicit congestion notification (ECN to IP. RFC 3168, Internet Engineering Task Force, Septebmer 2001.

[21] RIPE NCC. Atlas, 2010. See `https://atlas.ripe.net/`.

[22] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session traversal utilities for NAT (STUN). RFC 5389, Internet Engineering Task Force, October 2008.

[23] R. Stewart. Stream control transmission protocol. RFC 4960, Internet Engineering Task Force, September 2007.

[24] S. Sundaresan, W. De Donato, N. Feamster, R. Teixeira, S. Crowford, and A. Pescapé. Broadband Internet performance: a view from the gateway. In *Proc. ACM SIGCOMM*, August 2011.

[25] V. Thirion, K. Edeline, and B. Donnet. Tracking middleboxes in the mobile world with traceboxandroid. In *Proc. 7th International Workshop on Traffic Monitoring and Analysis (TMA)*, April 2015.

[26] B. Trammell, M. Kühlewind, D. Boppart, I. Learmonth, G. Fairhurst, and R. Scheffenegger. Enabling Internet-wide deployment of explicit congestion notification. In *Proc. Passive and Active Measurement Conference (PAM)*, April 2015.

[27] UPnP Forum. UPnP specifications, 2016. `http://upnp.org/sdcps-and-certification/standards/` (accessed: 2016/06/17).

[28] V. Jacobson et al. traceroute. man page, UNIX, 1989.

[29] M. Varvello, J. Blackburn, D. Naylor, and K. Papagiannaki. EYEORG: A platform for crowd-sourcing web quality of experience measurements. In *Proc. ACM CoNEXT*, December 2016. see `https://eyeorg.net`.

[30] M. Varvello, K. Schomp, D. Naylor, J. Blackburn, A. Finamore, and K. Papagiannaki. Is the web HTTP/2 yet? In *Proc. Passive and Active Measurement Conference (PAM)*, April 2016.

[31] Z. Wang, Z. Qian, Q. Xu, Z. Mao, and M. Zhang. An untold story of middleboxes in cellular networks. In *Proc. ACM SIGCOMM*, August 2011.

[32] N. Weaver, C. Kreibich, N. Dam, and V. Paxson. Here be web proxies. In *Proc. Passive and Active Measurement Conference (PAM)*, March 2014.

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

[33] J. Weil, V. Kuarsingh, C. Donley, C. Liljenstolpe, and M. Azinger. IANA-reserved IPv4 prefix for shared address space. RFC 6598, Internet Engineering Task Force, April 2012.

[34] R. Zullo, K. Edeline, A. Pescapé, and B. Donnet. Traceboxandroid, 2016. See `https://play.google.com/store/apps/details?id=be.ac.ulg.mobiletracebox`.