Testing for traffic differentiation with ChkDiff: the downstream case

Riccardo Ravaioli
Université Nice Sophia Antipolis
Laboratoire I3S/CNRS UMR 7271
Sophia Antipolis, France
ravaioli@i3s.unice.fr

Guillaume Urvoy-Keller
Université Nice Sophia Antipolis
Laboratoire I3S/CNRS UMR 7271
Sophia Antipolis, France
urvoy@i3s.unice.fr

Chadi Barakat
Inria
Sophia Antipolis, France
chadi.barakat@inria.fr

***Abstract*—**
In the past decade it has been found that some Internet operators offer degraded service to selected user traffic by applying various differentiation techniques. If from a legal point of view many countries have discussed and approved laws in favor of Internet neutrality, confirmation with measuring tools for even an experienced user remains hard in practice. In this paper we extend and complete our tool ChkDiff, previously presented for the upstream case, by checking for shaping also on the user's downstream traffic. After attempting to localize shapers at the access ISP on upstream traffic, we replay downstream traffic from a measurement server and analyze per-flow one-way delays and losses, while taking into account the possibility of multiple paths between the two endpoints. As opposed to other proposals in the literature, our methodology does not depend on any specific Internet application a user might want to test and it is robust to evolving differentiation techniques that alter delays or induce losses. We provide here a detailed description of the downstream tool and a validation in the wild for wired, WiFi and 3G connections.

I. Introduction

The increasing popularity of bandwidth-hungry applications, like peer-to-peer and video streaming, has induced some Internet Service Providers (ISPs) in the last decade to deploy some traffic management techniques that offer degraded performance instead of best-effort service to specific traffic flows. Reported cases abound: from blocking of BitTorrent traffic when a user is actively sharing files [1], to reduced performance of NetFlix by a few US operators [2], [3] and throttling of YouTube during peak hours in the evening [4], [5]. Also competing services such as VoIP have been the target of ISPs, for instance when all Vonage calls were systematically blocked by a regional mobile operator [6] and when Apple's FaceTime was disabled for mobile customers who did not opt for a more expensive data plan [7].

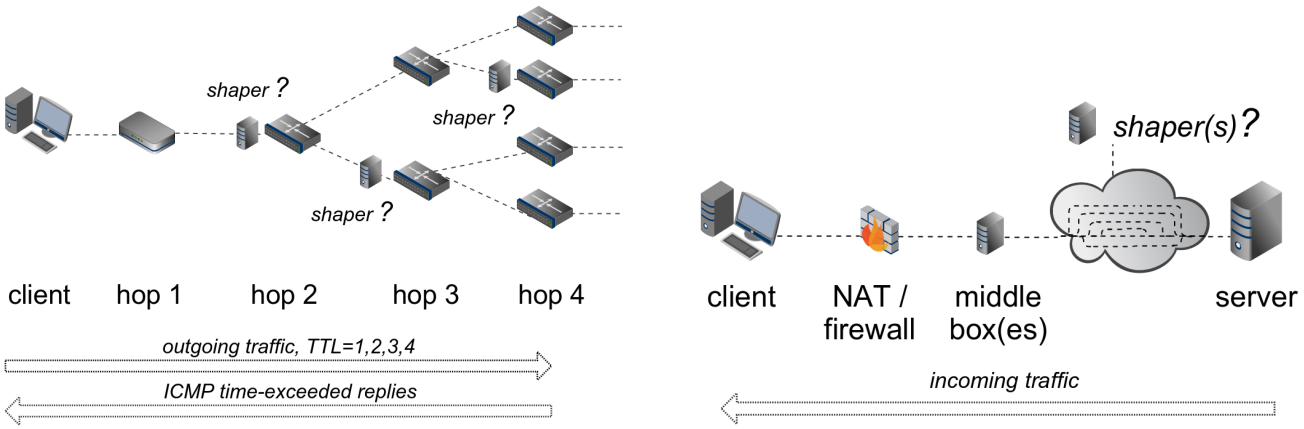
All these examples constitute clear violations of Internet neutrality, a principle according to which a network should treat all its incoming traffic equally, without deliberately offering worse or better performance to any traffic of its choice. There has been discussion [8] about whether the Internet has been conceived and implemented as a strictly level-playing field, but from a broader point of view it is generally agreed upon that all attempts that selectively deteriorate certain types of Internet traffic over others are, to say the least, controversial. Because of this, legislative efforts aiming at prohibiting cases like the above ones have appeared in a number of countries, the first of which to approve such laws were Chile [9] in 2010 and the Netherlands [10] in 2012.

In the literature, several tools [11]–[16] have been described in recent years to try to establish whether an ISP is applying traffic differentiation to specific applications (e.g., BitTorrent, YouTube, Skype, etc.) and with the use of specific differentiation techniques (e.g., port blocking, token-bucket shaper, etc.).

The solution that we propose in this paper, ChkDiff, directly addresses the problems of scalability to different user applications and of applicability to different shaping techniques affecting delays and losses. We achieve this by performing active measurements with the real user traffic, comparing the performance of a flow against the performance of the rest of the replayed traffic, and by analyzing for each flow its delays and losses, which reflect any alteration introduced by a shaper inside the network.

This complements our previous work [17], in which we focused on the user's upstream traffic and replayed it with low TTL values in a traceroute-like manner against the routers at the first few hops away from the user in order to detect differentiation and localize shapers. We extend this with a new experiment in the downstream direction, where we replay the user's incoming traffic from a server, measure one-way delays and losses, and check for differentiation on a per-flow basis. We describe in details the measures taken by ChkDiff in order to successfully deliver the replayed trace and validate the tool in two differentiation scenarios, with the server located in three different data centers, and over wired, WiFi and 3G connections.

The paper is organized as follows: in Section II we provide a detailed description of the methodology we used in ChkDiff; in Section III we validate the tool; we discuss our method in Section IV and assess it with respect to related work in Section V; we give closing remarks in Section VI.



(a) Upstream experiment

(b) Downstream experiment

Fig. 1: The two experiments in ChkDiff.

II. Methodology

The design of a new tool for the detection of traffic differentiation has to necessarily consider two weak points of existing methods: the difficulty to scale to different applications and the limitation to specific differentiation techniques. We overcome this in three steps: *a*) we use the real traffic of a user and not a synthetic trace; *b*) we minimize the modifications to the trace needed for the experiment to work and *c*) we analyze the performance of a flow in terms of delays and losses with respect to the rest of the trace in order to infer neutrality violations: these two metrics alone are able to capture the effect of shapers at the IP layer.

A complete run of ChkDiff consists of two experiments, one that replays the user's outgoing traffic (upstream direction) to the routers at the first few hops away from the user and one that replays the user's incoming traffic (downstream direction) from a measurement server to the user. We report in Algorithm 1 an outline of a full execution of ChkDiff.

At first the tool dumps client traffic for a time window of typically 3-5 minutes, while the user is asked to run the applications and services of an Internet session she wishes to test. Next, packets are grouped into 5-tuple flows (source and destination IP addresses, transport protocol, source and destination port numbers), which are further arranged into an outgoing trace, $trace_{out}$, and an incoming one, $trace_{in}$. We now briefly describe the upstream experiment, as proposed in an earlier work [17], and then go on to illustrate in detail the methodology for the downstream case.

A. Upstream experiment, in a nutshell

Non-trivial outgoing traffic that an access ISP might want to differentiate includes media uploading, P2P file sharing and VoIP; a run of ChkDiff in the upstream direction should ideally test at least one type of such traffic. Before conducting the actual experiment, we shuffle $trace_{out}$ in such a way that the position of the packets of each flow inside the trace follows a Poisson process, so that according to the PASTA property (Poisson Arrivals See Time Averages) [18], each flow will see the same network conditions when the trace is replayed. The order of packets within each flow is preserved. As in the first half of Algorithm 1, we consider the first few hops away from the user, at or in proximity of her access ISP network (up to hop 3 or 4, as in Figure 1a), and for each hop h we replay $trace_{out}$ at a constant sending rate higher than the original one and with a modified IP TTL set to h, so that the trace will expire on the router(s) at hop h and generate ICMP time-exceeded messages. We showed the validity of this ICMP feedback in a previous work [19], along with its robustness to ICMP rate-limitation. Each flow having its own set of Round-Trip Times (RTTs) and losses, we can now compare its performance up to a given hop to that of the rest of the flows along the same path. We use Kolmogorov-Smirnov test to analyze delays and a binomial-inspired test to analyze losses. By aggregating the results of each flow across consecutive hops we are able to infer the presence of a shaper and localize it in terms of number of hops from the client.

B. Downstream experiment

In this paper, we present the downstream version of ChkDiff and validate it experimentally. In brief, the experiment in the downstream direction consists in taking all necessary measures to replay the original incoming flows, having the server replay the trace to the client and finally analyzing the results in a way that takes into account the possibility of having multiple paths to the client (Figure 1b).

The second half of Algorithm 1 outlines the main steps of the downstream experiment, which we describe in details in the remainder of this section. First, we need to shuffle $trace_{in}$ in the same way we did in the upstream experiment. This allows us to eliminate cross traffic noise from the effects of possible neutrality violations. Then, for a replayed flow to be able to successfully reach the client, we need to deal with possible Network Address Translation (NAT) and firewall devices a user might be behind and also other possible middleboxes that might be deployed along the path from the

server. After all connections are initiated from the client side, the server replays the shuffled $trace_{in}$ to the client at a rate higher than the original one. We compute One-Way Delays (OWD's) for each flow and note the number of losses, if any. In order to infer differentiation, we run a clustering analysis on delays so that we can distinguish when different delay distributions are due to shaping and when they are due to a variety of paths. Lastly, a test on flow losses completes the analysis. We elaborate now on each of the above actions.

1) Getting ready for replaying. As opposed to the upstream experiment [17], we do not pad packets in $trace_{in}$ to a fixed size (the maximum packet size in the trace). In the upstream experiment the low variability of the total delay along a short wired path is in the same order of magnitude as the variability of the transmission delay, which is proportional to the packet size. That makes it impractical to replay packets exactly as captured, since flows with large packets over a wired connection experience larger delays solely because of their packet size. For the downstream case, we examine a much longer path in terms of hops and delays, and such source of error is canceled out by the inherent delay variability along a larger path. Therefore we are able to replay the packets with their original payloads, as seen by the client upon receiving them.

Replaying incoming traffic from a single source (i.e. our server) means that we cannot keep the original source IP addresses of the user trace, for two main reasons. Firstly, most access networks today are configured to drop outgoing packets with a source address that does not belong to the address space of the access network itself. In other words, they do not allow IP address spoofing [20]. Secondly, as we will see shortly, if a NAT device is present, we can replay a flow only if we find the mapping applied by the NAT to that flow for external endpoints. Since we are not in control of other endpoints (that is to say, all network applications or services run by the user) than our measurement server, we need to overwrite the IP source address of each packet in $trace_{in}$ with the IP address of the server; original port numbers are retained. Conversely, in the upstream experiment the original source and destination IP addresses are preserved. However, even if in the downstream case we lose the ability to reveal shapers based on the source IP address, we can combine upstream and downstream experiments to overcome the limitations of both. Furthermore, the commercial shapers studied in a recent work [21] do not use this piece of information to classify flows.

Algorithm 1 ChkDiff complete execution

```
Capture user traffic, store into trace_out and trace_in
▷ Upstream experiment
for each hop h ∈ {1, 2...k} do
    for each run r ∈ {1, 2, 3} do
        shuffle trace_out
        replay trace_out with TTL ← h
        collect ICMP time-exceeded replies
    end for
    detect shaped flows at hop h
end for
aggregate results and locate shaper(s), if any
▷ Downstream experiment
for each run r ∈ {1, 2, 3} do
    shuffle trace_in
    for each flow f in trace_in do
        find NAT mapping for f
        initiate connection from client
    end for
    replay trace_in from server to client
    compute one-way delays and losses
end for
detect shaped flows
```

After being shuffled, $trace_{in}$ is sent via FTP to the server. While we deploy a server with a public IP address, listening on a known port, a client is likely less easy to reach: a few network elements need to be considered before we can replay the trace to the client.

- **NAT's.** In today's networks, where IPv4 addresses are running out, deploying a NAT device has become a widespread practice. For our purposes, this means that the client's view of a flow might not be the same as the server's. Since the trace to replay is originally as seen by the client, we might need to modify the destination IP address and port number in order to reach the client from an external endpoint (Figure 1b). NAT devices are usually defined according to how they map the same source pair $X\!:x$ of IP address X and port number x of an internal network when different external destination IP addresses and port numbers are reached (for instance $Y_1\!:y_1$ and $Y_2\!:y_2$) [22], [23]. We distinguish four cases: *i*) in the simplest scenario the mapping is independent of the external destination endpoint and will not change for the same source pair $X\!:x$; *ii*) some NAT's generate the same mapping only with same external destination IP address ($Y_1 \equiv Y_2$) or *iii*) with the same external destination IP address *and* port number ($Y_1 \equiv Y_2$ and $y_1 \equiv y_2$)) and *iv*) the mapping might be connection-dependent and vary each time a new connection to the same external destination pair is initiated.
- **Firewall.** We assume that any user is protected by a firewall from the outer network. Consequently, all flows need to be initiated from the user side (the so-called hole punching) before the server can replay the trace. For TCP flows, we reproduce the whole 3-way handshake without the interaction of the kernel on both endpoints. We assign an initial sequence number for each side of a TCP flow and modify it accordingly in the data packets. For UDP flows, we only send one probe from the client.
- **Initial sequence numbers.** It has been observed [24], [25] that some middleboxes overwrite the initial sequence number of TCP flows. Since firewalls can easily keep track of the sequence numbers and reject inconsistent packets, it is important to intercept the overwritten

sequence number from the SYN packet of the TCP handshake and modify all subsequent sequence and acknowledgement numbers individually for each TCP flow.

- **Timeouts.** In NetFilter [26], the standard firewall provided in Linux, the TCP timeout for established connections is 5 days, while for UDP it is only 30 seconds whether packets have been seen in one or both directions. This means that we need to make sure that the time elapsing between two consecutive packets of each UDP flow, initial probes included, is less than 30 seconds. NAT mappings expire too in order to remove stale entries from the NAT table. Given the large amount of commercial NAT devices, we refer in this case to the requirements and guidelines found in the RFC's. For UDP flows [22] the timeout should not be less than 2 minutes, while 5 minutes is recommended; for TCP [23] if the connection is not yet in the established state, the timeout should not be less than 4 minutes, and if it is already in the established state it should be no less than 2 hours and 4 minutes. These values are all large enough for the purpose of our experiment and do not interfere with ChkDiff. Finally, to avoid unnecessary synchronization between client and server, we do not actually send any acknowledgments from the client side for the TCP flows we replay. In the Linux kernel, the socket parameter TCP_USER_TIMEOUT sets the maximum amount of time that data can be transmitted without being acknowledged. Its default value is 20 minutes [27], which is roughly one order of magnitude larger than a single run of ChkDiff.

Given that our goal is not to classify all possible devices along a path but to rapidly deliver the trace to the client, we take a conservative approach and assume that the most stringent restrictions among the above ones are in place. We expect the NAT to be connection-dependent and perform a per-flow mapping discovery already during the hole punching initiated by the client against her firewall. For each flow, we encrypt the client's view of its source IP address and port number and add it to the payload of its SYN packet or UDP probe, as NAT devices are expected to overwrite every occurrence of the client's own IP address in a packet. We set the client-side initial sequence number of TCP flows in accordance to the acknowledgement numbers used in the trace (since no packets are sent from the client during the replaying phase, the acknowledgement number of a TCP flow sent by the server is constant across packets of the same flow). From the server side, for TCP flows, we keep track of incoming SYN packets along with the observed sequence numbers and client's source pair, and mimic the TCP handshake; for UDP flows we just keep track of client's and server's views of the client-side source pair. When these two views differ, we modify the client-side IP address and port number of the corresponding flows in $trace_{in}$ with the pair as seen by the server. The server also overwrites the acknowledgement number of a TCP flow, when the number in the trace does not match the sequence number seen in the

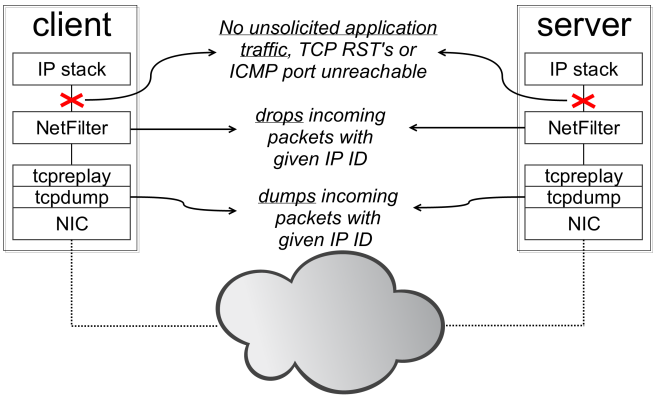


Fig. 2: Configuration of client and server.

received SYN packet. Additionally, in order to overcome the relatively short timeout on UDP flows, we enforce a maximum interval of 30 seconds between any two UDP packets of the same flow when shuffling $trace_{in}$ and start a timeout on the client side to make sure that we do not exceed 30 seconds between the initial UDP probe of a flow and its first occurrence in the trace. We discard the current run of the experiment and start a new one if ever this timeout expires. In any case, during an ordinary execution of ChkDiff only a few seconds elapse between hole punching and the start of the replaying phase.

We avoid the overhead of opening a socket for each flow and replaying the trace from the application layer by injecting packets with *tcpreplay* [28] directly between the IP layer and the Network Interface Card. Since we are emulating TCP and UDP flows below the IP layer, we need to prevent our packets from reaching their respective client applications, which could cause unsolicited traffic or unexpected application behaviour. Also, our hole-punching probes target ports on the server on which no process should be listening and will trigger TCP reset packets for TCP SYN probes, ICMP port-unreachable messages for UDP probes and real application packets if ever a process is indeed listening. As error messages cross a firewall on their way to the user, the corresponding newly-created connection entries are removed. Therefore, we need a way to distinguish between experiment packets and regular traffic, so that we can drop the former right before they reach the IP stack and we can allow the latter to pass through. We achieve this by assigning a unique number to each user session and overwrite with this value the 2-Byte IP ID field of each experiment packet between a given user and the server. Through a combined use of *tcpdump* and *iptables*, as shown in Figure 2, we dump the experiment packets right at the Network Interface Card and drop them before they reach the kernel.

2) Replay. We are now ready to replay the trace from the server at a constant packet rate higher than the original one (by default, we replay it at twice the original rate).

We dump the replayed trace on both the server and the client and then for each flow we measure One-Way Delays and note the number of lost packets. For simplicity, we avoid any clock synchronization between user and server: any effect

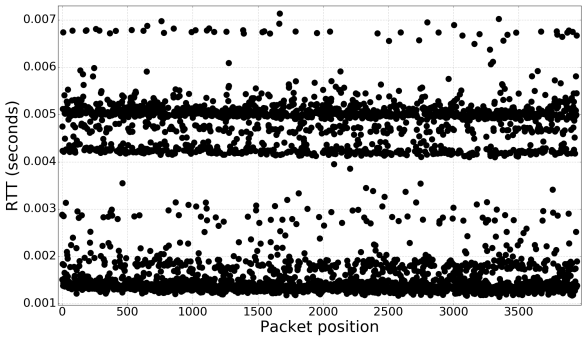


Fig. 3: Timeseries of an experiment with packets following multiple paths. Each packet is represented by a black dot.

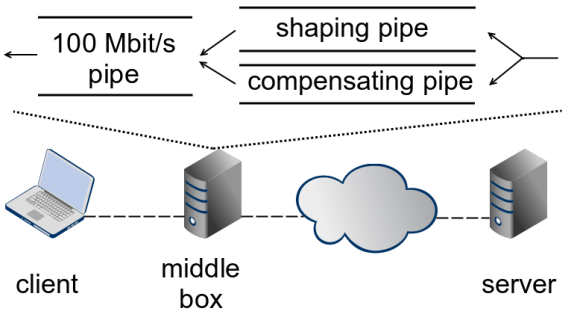


Fig. 4: Setup used in the validation of ChkDiff, along with the shaper configuration.

due to clock skewing is not expected to disrupt the measured delays for the short time window of one experiment (a few tens of seconds) and in any case will affect all flows equally, as they are evenly spread across the trace.

Once the server has completed the replaying phase, the client closes the emulated TCP connections by sending an RST packet for each TCP flow in $trace_{in}$. This has the added benefit of clearing space in the open connection table of the firewall, if ever a per-user restriction is active.

3) Results analysis. The study of delays between two endpoints across a path of several hops has to necessarily take into account the possibility of multiple paths. Discovering the paths taken by each flow would be cumbersome: first of all, we do not know the exact hash function applied in a load balancing decision and we observed that some data centers, where the measurement server could be deployed, make a massive use of load balancers; second of all, it would take some extra time, as we would need to probe at low rates (i.e. 1 packet per second) to bypass ICMP rate limitation [19] and have a complete view of each path. An example is provided in the timeseries of Figure 3, where we can visually identify at least five different paths; a direct comparison between any two flows becomes harder in this case. In the absence of the ground truth, we can rely on the fact that non-differentiated flows following the same path will have similar delay distributions, which a clustering algorithm can group together. A differentiated flow going on any of the available paths will show a distribution significantly different from that of all other flows and should not belong to any of the discovered groups. We combine this with a loss analysis in order to capture the behaviour of shapers.

- **Delays.** Our choice of clustering algorithm for delays is dbscan [29], which groups together points that are in the same high-density area and labels as outliers those that do not belong to any found cluster. As a representative point for each flow we take its 25th percentile: it is close enough to the real path delay, it discards possible queueing delays and it is robust to delay variations due for example to WiFi. The algorithm then takes two parameters: the minimum number n of core samples to form a cluster and the maximum distance ϵ between any two samples for them to be included as core points in a cluster. Since we expect shaped flows to stand out from non-shaped flows, we set n to 2. As for ϵ, we need a value that reflects the delay variations of a path: we take the core values (2nd quartile range, i.e. the 25th-50th percentile range) of the delay distribution of each flow, we aggregate them and then pick for ϵ a large value in this set, the 75th percentile. The output of dbscan will be a set of clusters of flows and a set of outliers, which we label as having failed the delay analysis.
- **Losses.** We compare the losses of a flow to the loss rate of the rest of the trace as a whole. The reasoning is the following: if a flow i with s_i packets has not been differentiated, its number of lost packets can be modeled as a binomial random variable of parameters $B(s_i, p)$, where p is the loss rate of the rest of the trace. If we approximate this binomial to a normal random variable of parameters $N(s_i p,\ s_i p(1-p))$, we can verify whether the number of lost packets l_i of flow i lies within α standard deviations of the normal mean, where α is approximated to 2.58 for our chosen significance level of 99%. Since we are interested to know whether a flow experienced *more* losses than it should have with the global loss rate p, we check that $l_i < p s_i + \alpha\sqrt{p(1-p)s_i}$. If the condition does not hold, the flow is rejected by our loss analysis.

Since a shaper affects the delays or the losses of a flow, or both, we reject a flow if it fails either analysis.

We repeat the whole experiment three times in order to remove transient errors and claim that a flow was differentiated if in *all* three runs it failed the combined analysis of delays and losses.

III. Validation

We validate the downstream experiment of ChkDiff in wired, WiFi and 3G setups (Figure 4) with a client located in France and the server located in three different Amazon data centers: Germany, Ireland and Oregon (USA). The client is directly connected to a middlebox, where the shaper is deployed and which serves also as the client's gateway. In the WiFi setup, the client is connected to the gateway through a dedicated WiFi network operating on the same channel as

the local University WiFi network to cause more link-level collisions. In the 3G setup, the client is connected to the middlebox via a wired connection and the middlebox is connected via WiFi to a mobile phone functioning as hotspot. We test ChkDiff in two differentiation scenarios: given a set of flows we want to differentiate, in Scenario 1 we throttle their bandwidth and in Scenario 2 we apply a uniform packet drop rate to them. We configure dummynet [30] on the middlebox to shape incoming traffic, as shown in the upper part of Figure 4. Flows to shape are forwarded to the upper pipe, which applies the desired differentiation technique according to the scenario we test; flows that we do not intend to differentiate go through the lower pipe, which only adds a constant delay equal to the transmission delay of the upper pipe in Scenario 1 and has no effect in Scenario 2. This way, in Scenario 1 the difference in delays between shaped and non-shaped flows is due only to the queueing delay at the upper pipe. A final pipe, where all flows eventually go, emulates a 100 Mbit/s link. In Scenario 2 this last pipe causes uniform drops on the whole trace.

In all experiments we replay a trace of approximately 9000 packets captured during an Internet session of 3 minutes that included watching a short streaming video, browsing a news website and making a call on Skype. In dummynet, our pipes have a buffer length of 100 packets and use droptail buffer management policy.

A. Shaping pipe (Scenario 1).

In this scenario, we compute in $trace_{in}$ the overall sending rate of the flows to shape and set the shaping pipe to a fraction $k_{bw} < 1$ of this value. The second parameter we vary is the fraction fr of packets we shape. When picking which flows to differentiate, we choose iteratively the flow closest to the target fr, until the desired size is reached. All flows we differentiate go through the same shaping pipe. By combining delay and loss analysis, we show that we can effectively identify all shaped flows as long as the fraction fr does not constitute most of $trace_{in}$, which is what we expect when we take the whole trace as baseline for comparison.

We present in Figure 5 the results in terms of recall[1] for each of the three server locations over wired, WiFi and 3G connections. For constraints of space, we do not include the results in terms of precision, commonly defined as the number of true positives over the number of positives, since we found it for all experiments to be the optimal one, at 100%; in short, we never flagged a non-differentiated flow erroneously. For each pair of fr and k_{bw} we show a pie where the color of the upper-left quarter represents the result of the delay analysis, the color of the upper-right quarter represents the result of the loss analysis, and the color of the lower half is the outcome of the combined analysis. If for $fr \leq 40\%$ in almost all cases the delay analysis suffices to detect all shaped flows, for $fr = 60\%$ we see that combining the delay and loss analysis is essential for a correct output. When $k_{bw} = 1.0$ the shaping pipe is configured with the average bit rate of incoming packets, so the flows that are supposedly being differentiated might not be shaped at all, hence the uncertain outcome on the top row of each graph.

In this scenario, ChkDiff appears to cope just as well with a wired connection as it does with WiFi and 3G. Over the two wireless connections, there seems to be more noise that is in any case neutralized for the most part when combining the two analysis.

B. Uniform drops (Scenario 2).

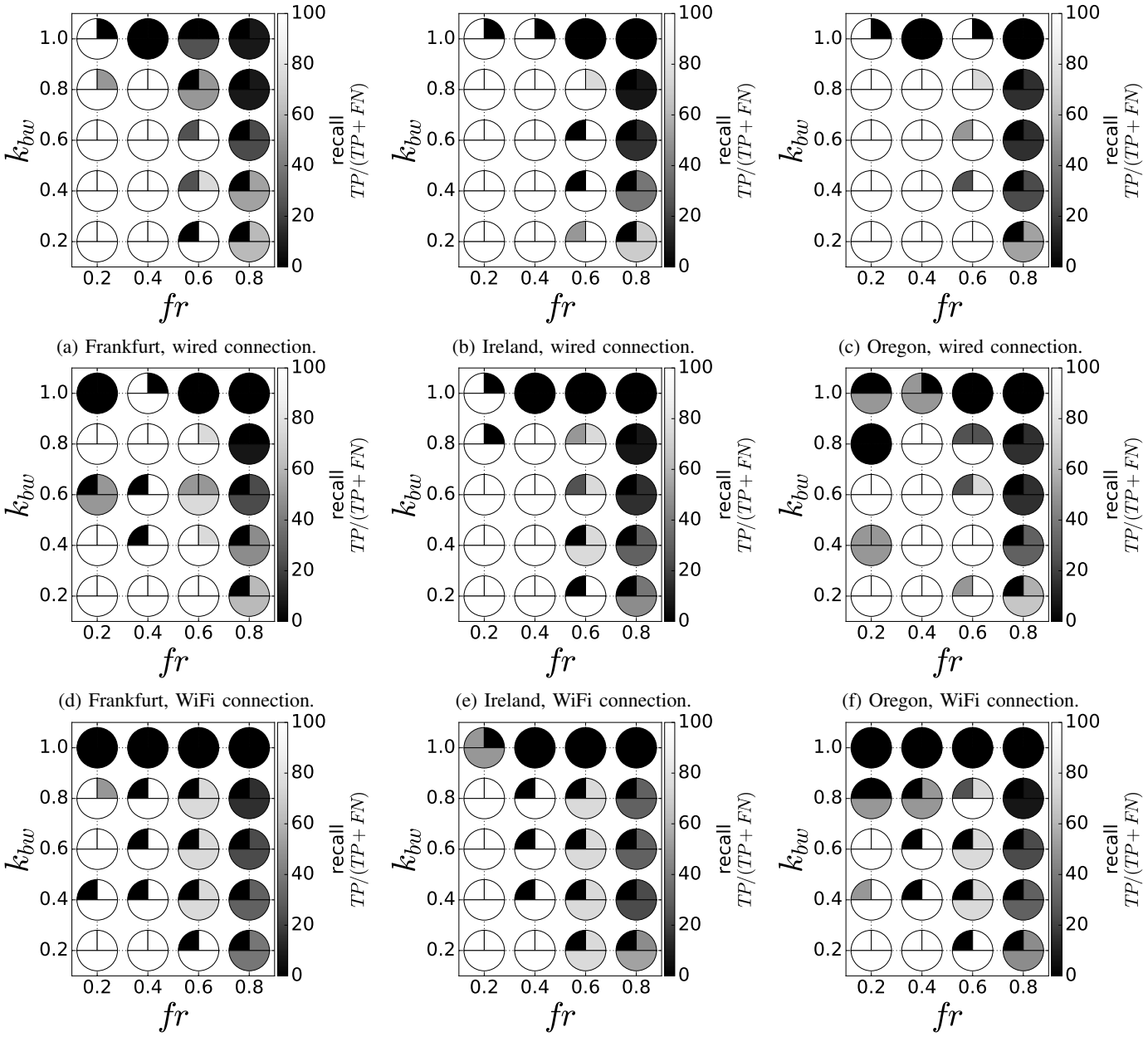
We consider now a shaper that uniformly drops packets of selected flows at a loss rate lr (upper-right pipe in Figure 4) and of the whole trace at a loss rate lr_{all} (left pipe in the same figure). As opposed to the previous scenario, we deploy here a dedicated shaping pipe for each flow we want to differentiate. We vary lr and lr_{all}, as well as the fraction fr of traffic impacted by lr. Shaped flows will thus have an overall loss rate lr_{shaped} equal to $1-(1-lr)(1-lr_{all})$. For reasons of space we only show the results for the server located in the data center in Germany, over the three types of connection considered previously (Figure 6). Nevertheless, results across the three server locations are qualitatively similar. Since in this second scenario the differentiation we apply does not affect delays, we focus only on the outcome of the loss analysis. For completeness, results are shown also for high values of lr_{all}, even if in practice a global loss rate of 20% is already able to disrupt TCP connections. We observe for both wired and wireless setups that when fr is less than half of the trace, ChkDiff is able to detect all differentiated flows except for the cases on the bottom diagonal, where the difference in loss rate between differentiated and non-differentiated flows is the lowest (5%, 10% and sometimes 20%). Experiments over WiFi and 3G seem to be only slightly worse than over wired for the lowest values of lr_{all}.

IV. Discussion

A full run of ChkDiff in upstream and downstream directions is able to detect differentiation when, regardless of its implementation, it directly worsens the throughput, packet delay and losses of user applications. This is the typical effect introduced by a shaper. Even though in this paper we used the terms shaping and differentiation interchangeably, the former is a subset of the latter and shaping is what we aim at detecting with our current tool. As we saw in the validation section, we cannot precisely reveal shaping when most of the user traffic is affected, as our baseline in the analysis would mainly be made of differentiated flows. To counteract this, a user should be running different applications during the capturing phase, so as to have a variety of flows to check against. In the extreme case, if really an ISP throttles the bandwidth of all traffic for a given user, it would not be possible to discern it from severe network congestion from the view point of this particular user.

In this work, we assume that ISPs select flows to differentiate based on packet fields (IP addresses, ports and

[1]The definition of recall we use is the classic one: the number of true positives over the sum of true positives and false negatives; in our specific case, it is the number of differentiated flows correctly detected as differentiated by the combined analysis over the number of flows that we know were shaped, whether or not we have detected them. A recall of 100% indicates that we have correctly identified all shaped flows, while a recall of 0% indicates that we have missed them all.



(a) Frankfurt, wired connection.

(b) Ireland, wired connection.

(c) Oregon, wired connection.

(d) Frankfurt, WiFi connection.

(e) Ireland, WiFi connection.

(f) Oregon, WiFi connection.

(g) Frankfurt, 3G connection.

(h) Ireland, 3G connection.

(i) Oregon, 3G connection.

Fig. 5: Recall of combined analysis (delay and losses) for Scenario 1 over wired, WiFi and 3G connections.

payload). We do not directly address classification based on flow bandwidth, but we make sure that we replay the trace at a higher global rate than the original one.

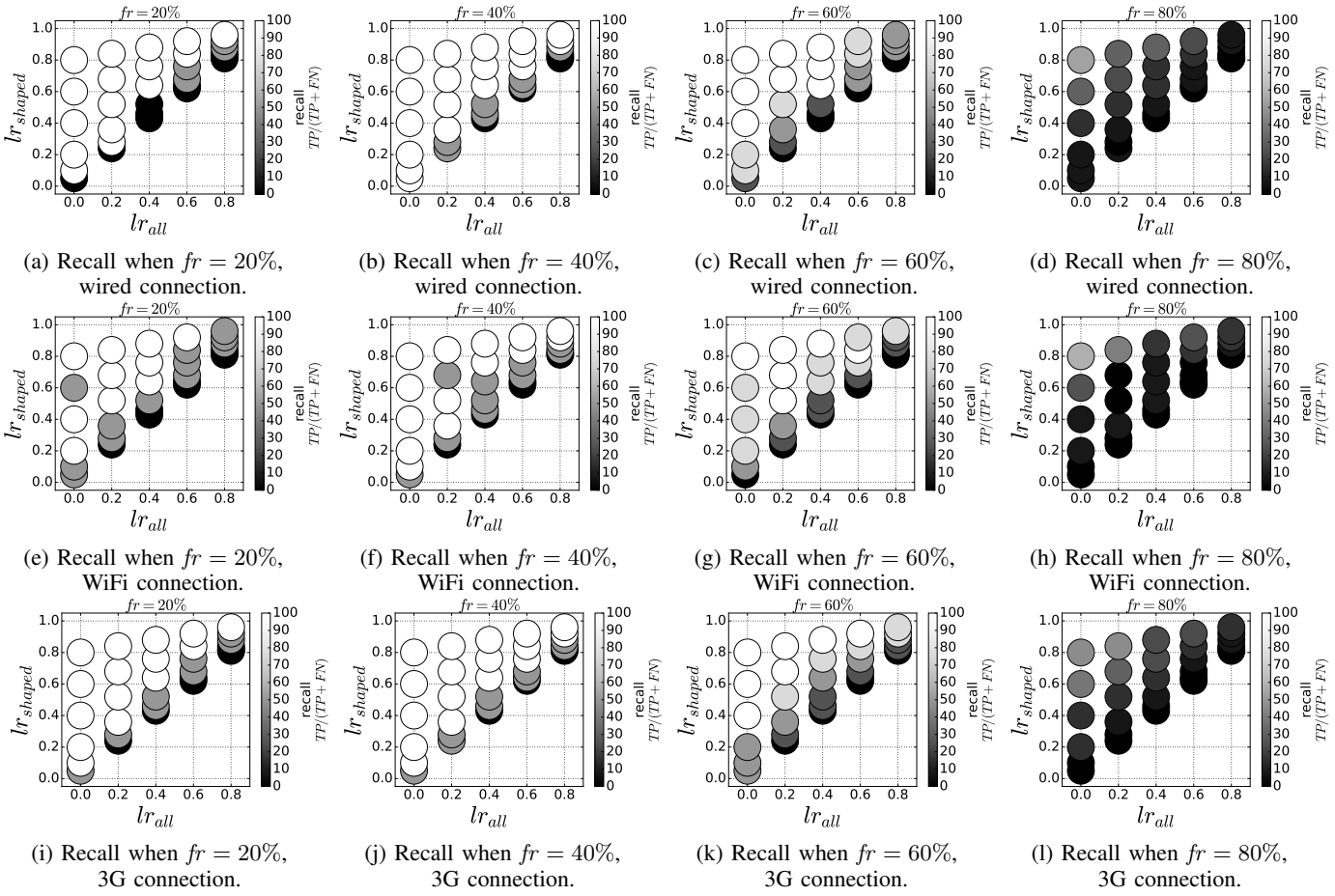
Also, we do not detect differentiation when it aims at providing better treatment to selected traffic. Such behaviour could be the result of an agreement between a content provider and an ISP and it does not necessarily imply any worse conditions for the rest of the traffic than in normal network conditions. It would be interesting to redefine the delay analysis in both upstream and downstream experiments to account for this, as it could reveal what services and applications are favored in a given network. We plan to address this in future work.

The tool is available for Linux machines on the web page of the project[2]. We currently provide a server located in our lab, where no traffic differentiation is taking place.

V. Related Work

Many tools for the detection of traffic differentiation have appeared in the literature in recent years. Among the first ones, BT-test [12] checks for injected RST packets while emulating a BitTorrent packet exchange between a user and a server. Other tools [13]–[16] compare the performance of a synthetic

[2]http://chkdiff.gforge.inria.fr/



(a) Recall when $fr = 20\%$, wired connection.
(b) Recall when $fr = 40\%$, wired connection.
(c) Recall when $fr = 60\%$, wired connection.
(d) Recall when $fr = 80\%$, wired connection.
(e) Recall when $fr = 20\%$, WiFi connection.
(f) Recall when $fr = 40\%$, WiFi connection.
(g) Recall when $fr = 60\%$, WiFi connection.
(h) Recall when $fr = 80\%$, WiFi connection.
(i) Recall when $fr = 20\%$, 3G connection.
(j) Recall when $fr = 40\%$, 3G connection.
(k) Recall when $fr = 60\%$, 3G connection.
(l) Recall when $fr = 80\%$, 3G connection.

Fig. 6: Recall of loss analysis as we vary fr in Scenario 2. Results are from the server located in Germany over wired, WiFi and 3G connections.

application flow to the performance of a similar flow with some modified or randomized packet fields (port numbers or payloads), so that a shaper targeting such application would affect the former and not the latter. Glasnost [13] looks for differences in throughput between these two flows, while DiffProbe [14] attempts to create congestion in the ISP network by scaling up the replaying rate of application and control flows, and then analyzes their delay and loss distributions. ShaperProbe [15] expands on DiffProbe by considering the case of a shaper implemented as a token bucket and tries to infer its parameters as the received rate at the destination shows a level shift. Packsen [16] also tries to identify the shaper type and its parameters, but claims to use a more efficient statistical analysis. As opposed to ChkDiff, these tools are limited to the set of application traces made available by their authors (e.g., Skype, BitTorrent, YouTube, etc), which would make it hard to maintain them in the long run.

A recent work, Differentiator Detector [21], aims at solving this by replaying between user and server a captured user trace and reproducing the same original application behaviour (ports, payloads, inter-packet times) *at* the application layer, first through a direct path between the two endpoints (application flow) and then through VPN tunnel to a middlebox (control flow). It measures throughput, RTT distribution and losses in order to detect shaping. Even though our tool replays separately upstream and downstream traffic, in the upstream direction it tests the real hops where the original packets went instead of testing the path between client and server, and in the downstream direction it deals in a more robust and scalable way with NAT devices. Moreover, differentiation of VPN's is not unheard of [31] and the released version of Differentiation Detector only provides a set of pre-recorded traces to replay.

NetPolice [11] makes use of TTL-limited probes from numerous vantage points to ingress and egress routers of backbone ISPs to find differentiation in backbone networks. It replays a few synthetic application flows along with an HTTP flow, which serves as control flow, and looks for differences in loss rates on the same path segments.

Zhang et al. [32] propose a theoretical framework where they conduct measurements on target links from a large number of vantage points and try to derive differentiation from link properties typical of non-differentiated networks, in a way inspired by network performance tomography. This method, if deployed, would need a substantial and diverse user base in order to have enough vantage points and would require a central server to process measurement results.

Nano [33] differs from existing solutions in that it carries out passive measurements on user traffic and compares it against a data set of other users in the same geographical area, with comparable machine setups, at the same time of the day, but connected to a different ISP. While this method is undeniably independent of user applications and differentiation techniques, its main disadvantage is that it needs a fairly large number of users for it to be operational.

VI. Conclusion

We extended with a downstream experiment ChkDiff, a tool which enables users to detect differentiation on their own traffic. After first checking for degraded traffic performance on upstream traffic, the tool replays user incoming flows from a measurement server to the user and analyzes delays and losses to verify whether each flow experienced the same network conditions as the rest of the trace. While in the upstream direction our tool proved to be robust to rate limitation in the ICMP feedback generated by routers, in the downstream case we successfully cope with NAT's and middleboxes in front of the client and with end-to-end measurements possibly representing a diversity of paths between server and client. We validated ChkDiff in the wild, with two differentiation scenarios over three types of connections: wired, WiFi and 3G. We showed that it correctly identifies shaped flows when up to half of a user trace is affected.

In future work, we envisage to include in the tool tests that check for differentiation techniques that do not necessarily alter delays and losses, as for instance TCP RST injection. We also intend to run a study with volunteers in a variety of wired and mobile setups in order to have a mapping of the current practices of ISPs.

Acknowledgments

This work was funded by the French Government (National Research Agency, ANR) through the "Investments for the Future" Program reference #ANR-11-LABX-0031-01.

References

[1] "Dslreports: comcast is using sandvine to manage p2p connections." [Online]. Available: http://goo.gl/bpFpru

[2] "Cogent now admits they slowed down netflix's traffic, creating a fast lane & slow lane." [Online]. Available: http://goo.gl/WDiLom

[3] "Netflix performance on Verizon and Comcast has been dropping for months," 2013. [Online]. Available: http://goo.gl/fTSbCe

[4] "Respect my net." [Online]. Available: http://respectmynet.eu/view/196

[5] "Respect my net." [Online]. Available: http://respectmynet.eu/view/205

[6] "Vonage says broadband provider blocks its calls." [Online]. Available: http://www.cnet.com/news/vonage-says-broadband-provider-blocks-its-calls/

[7] "AT&T blocking iPhone's FaceTime app would harm consumers and break net neutrality rules." [Online]. Available: http://www.freepress.net/press-release/99480/att-blocking-iphones-facetime-app-would-harm-consumers-and-break-net-neutrality

[8] J. Crowcroft, "Net neutrality: the technical side of the debate: a white paper," *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 49–56, January 2007.

[9] "Chile, primer país en incorporar la neutralidad en la red." [Online]. Available: http://www.elmundo.es/elmundo/2010/07/16/navegante/1279272468.html

[10] "Net neutrality enshrined in dutch law." [Online]. Available: http://www.theguardian.com/technology/2011/jun/23/netherlands-enshrines-net-neutrality-law

[11] Y. Zhang, Z. M. Mao, and M. Zhang, "Detecting traffic differentiation in backbone isps with netpolice," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '09. ACM, 2009, pp. 103–115.

[12] M. Dischinger, A. Mislove, A. Haeberlen, and K. P. Gummadi, "Detecting BitTorrent Blocking," in *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement (IMC'08)*, Vouliagmeni, Greece, October 2008.

[13] M. Dischinger, M. Marcon, S. Guha, K. Gummadi, R. Mahajan, and S. Saroiu, "Glasnost: Enabling End Users to Detect Traffic Differentiation," in *Proceedings of the 7th Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, Apr 2010.

[14] P. Kanuparthy and C. Dovrolis, "Diffprobe: detecting ISP service discrimination," in *Proceedings of the 29th conference on Information communications*, ser. INFOCOM'10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 1649–1657.

[15] ——, "Shaperprobe: End-to-end detection of isp traffic shaping using active methods," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11. ACM, 2011, pp. 473–482.

[16] U. Weinsberg, A. Soule, and L. Massoulié, "Inferring traffic shaping and policy parameters using end host measurements," in *INFOCOM*, 2011, pp. 151–155.

[17] R. Ravaioli, G. Urvoy-Keller, and C. Barakat, "Towards a general solution for detecting traffic differentiation at the internet access," in *Teletraffic Congress (ITC 27), 2015 27th International*. IEEE, 2015, pp. 1–9.

[18] R. W. Wolff, "Poisson arrivals see time averages," *Operations Research*, vol. 30, no. 2, pp. 223–231, 1982.

[19] R. Ravaioli, G. Urvoy-Keller, and C. Barakat, "Characterizing ICMP Rate Limitation on Routers," in *IEEE International Conference on Communications (ICC)*, 2015.

[20] "BCP 38 - Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing." [Online]. Available: https://tools.ietf.org/html/bcp38

[21] A. Molavi Kakhki, A. Razaghpanah, A. Li, H. Koo, R. Golani, D. Choffnes, P. Gill, and A. Mislove, "Identifying traffic differentiation in mobile networks," in *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*. ACM, 2015, pp. 239–251.

[22] "RFC 4787 - Network Address Translation (NAT) Behavioral Requirements for Unicast UDP." [Online]. Available: http://tools.ietf.org/html/rfc4787

[23] "RFC 5382 - NAT Behavioral Requirements for TCP." [Online]. Available: http://tools.ietf.org/html/rfc5382

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

[25] G. Detal, B. Hesmans, O. Bonaventure, Y. Vanaubel, and B. Donnet, "Revealing middlebox interference with tracebox," in *Proceedings of the 2013 Conference on Internet Measurement Conference*, ser. IMC '13. New York, NY, USA: ACM, 2013, pp. 1–8.

[26] "NetFilter: Firewalling, NAT and packet mangling for Linux." [Online]. Available: www.netfilter.org

[27] "Linux Programmer's Manual - TCP protocol ." [Online]. Available: http://man7.org/linux/man-pages/man7/tcp.7.html

[28] "Tcpreplay." [Online]. Available: http://tcpreplay.appneta.com/

[29] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[30] M. Carbone and L. Rizzo, "Dummynet revisited," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 2, pp. 12–20, Apr. 2010.

[31] "I just doubled my PIA VPN throughput that I am getting on my router by switching from UDP:1194 to TCP:443." [Online]. Available: http://goo.gl/pxXIWf

[32] Z. Zhang, O. Mara, and K. Argyraki, "Network neutrality inference," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 63–74.

[33] M. B. Tariq, M. Motiwala, N. Feamster, and M. Ammar, "Detecting network neutrality violations with causal inference," *ACM SIGCOMM CoNext*, p. 289, 2009.