An Internet-Wide Analysis of Traffic Policing

Tobias Flach*†, Pavlos Papageorge†, Andreas Terzis†, Luis D. Pedrosa*,
Yuchung Cheng†, Tayeb Karim†, Ethan Katz-Bassett*, and Ramesh Govindan*

* University of Southern California † Google

Abstract. Large flows like video streams consume significant bandwidth. Some ISPs actively manage these high volume flows with techniques like policing, which enforces a flow rate by dropping excess traffic. While the existence of policing is well known, our contribution is an Internet-wide study quantifying its prevalence and impact on transport-level and video-quality metrics. We developed a heuristic to identify policing from server-side traces and built a pipeline to process traces at scale collected from hundreds of Google servers worldwide. Using a dataset of 270 billion packets served to 28,400 client ASes, we find that, depending on region, up to 7% of connections are identified to be policed. Loss rates are on average 6× higher when a trace is policed, and it impacts video playback quality. We show that alternatives to policing, like pacing and shaping, can achieve traffic management goals while avoiding the deleterious effects of policing.

CCS Concepts

•**Networks** → **Network measurement;** *Network performance analysis;*

1. INTRODUCTION

Internet traffic has increased fivefold in five years [16], much of it from the explosion of streaming video. YouTube and Netflix together contribute nearly half of the traffic to North American Internet users [47,55,66]. Content providers want to maximize user quality of experience. They spend considerable effort optimizing their infrastructure to deliver data as fast as possible [11, 25, 29].

In contrast, an ISP needs to accommodate traffic from a multitude of services and users, often through different service agreements such as tiered data plans. High-volume services like streaming video and bulk downloads that require high goodput must coexist with smaller volume services like search that require low latency. To achieve coexistence and enfore plans, an ISP might enforce different rules on its traffic. For example, it might rate-limit high-volume flows to avoid network congestion, while leaving low-volume flows that have little impact on the congestion level untouched. Similarly, to enforce data plans, an ISP can throttle throughput on a per-client basis.

Policing	Enforces rate by *dropping* excess packets immediately – Can result in high loss rates + Does not require memory buffer + No RTT inflation
Shaping	Enforces rate by *queueing* excess packets + Only drops packets when buffer is full – Requires memory to buffer packets – Can inflate RTTs due to high queueing delay

Table 1: Overview of policing and shaping.

The most common mechanisms to enforce these policies are *traffic shaping* – in which traffic above a preconfigured rate is buffered – and *traffic policing* – in which traffic above the rate is dropped [15]. Table 1 compares both techniques. To enforce rate limits on large flows only, networks often configure their *shapers* and *policers* (the routers or middleboxes enforcing rates) to accommodate bursts that temporarily exceed the rate. In this paper, we focus on policing and briefly discuss shaping (§5.1.2).

The Impact of Policing. Policing is effective at enforcing a configured rate but can have negative side effects for all parties. While operators have anecdotally suggested this problem in the past [15, 62], we quantify the impact on content providers, ISPs, and clients at a global scale by analyzing client-facing traffic collected at most of Google's CDN servers, serving clients around the world. *Policing impacts content providers:* it introduces excess load on servers forced to retransmit dropped traffic. Globally, the average loss rates on policed flows are over 20%! *Policing impacts ISPs:* they transport that traffic across the Internet from the content provider to the client, only for it to be dropped. With 20% loss, a fifth of the bandwidth used by affected flows is wasted — the content provider and ISPs incur costs transmitting it, but it never reaches the client. This traffic contributes to congestion and to transit costs.

Policing impacts clients: ISP-enacted policing can interact badly with TCP-based applications, leading to degraded video quality of experience (QoE) in our measurements. Bad QoE contributes to user dissatisfaction, hurting content providers and ISPs.

Figure 1 shows the time-sequence plot of a policed flow

SIGCOMM '16 August 22-26, 2016, Florianopolis, Brazil

ACM ISBN 978-1-4503-4193-6/16/08.

DOI: http://dx.doi.org/10.1145/2934872.2934873

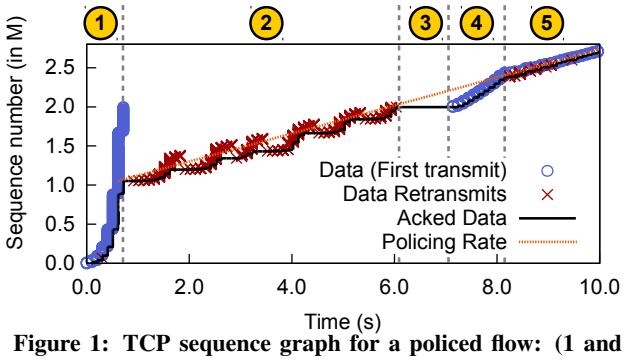


Figure 1: TCP sequence graph for a policed flow: (1 and 4) high throughput until token bucket empties, (2 and 5) multiple rounds of retransmissions to adjust to the policing rate, (3) idle period between chunks pushed by the application.

collected in a lab experiment (see §3). Because the policer is configured to not throttle short flows, the flow ramps up to over 15 Mbps without any loss (bubble 1), until the policer starts to throttle the connection to a rate of 1.5 Mbps. Since packets are transmitted at a rate that exceeds the policed rate by an order of magnitude, most of them are dropped by the policer and retransmitted over a 5-second period (2). Following the delivery of the first 2 MB, the sender remains idle until more application data becomes available (3). Since the flow does not exhaust its allotted bandwidth in this time frame, the policer briefly allows the sender to resume transmitting faster than the policing rate (4), before throttling the flow again (5). Overall, the flow suffers 30% loss.

Understanding Policing. Little is known about how traffic policing is deployed in practice. Thus, we aim to answer the following questions at a global scale: (1) How prevalent is traffic policing on the Internet? (2) How does it impact application delivery and user quality of experience? (3) How can content providers mitigate adverse effects of traffic policing, and what alternatives can ISPs deploy?

The question of user experience is especially important, yet ISPs lack mechanisms to understand the impact of traffic management configurations on their users. They lack visibility into transport-layer dynamics or application-layer behavior of the traffic passing through their networks. Further, policing means that content providers lack full control over the performance experienced by their clients, since they are subject to ISP-enacted policies that may have unintended interactions with applications or TCP.

To answer these questions, we need to overcome two hurdles. First, traffic management practices and configurations likely vary widely across ISPs, and Internet conditions vary regionally, so we need a global view to get definitive answers. Second, it is logistically difficult, if not impossible, to access policer configurations from within ISPs on a global scale, so we need to infer them by observing their impact on traffic and applications. We address these hurdles and answer these three questions by analyzing captured traffic between Google servers and its users.

Contributions. We make the following contributions:

1. We design and validate an algorithm to detect traffic policing from server-side traces at scale (§2, §3).

Region	Policed segments		Loss rate	
	(among lossy)	(overall)	(policed)	(non-pol.)
India	6.8%	1.4%	28.2%	3.9%
Africa	6.2%	1.3%	27.5%	4.1%
Asia (w/o India)	6.5%	1.2%	22.8%	2.3%
South America	4.1%	0.7%	22.8%	2.3%
Europe	5.0%	0.7%	20.4%	1.3%
Australia	2.0%	0.4%	21.0%	1.8%
North America	2.6%	0.2%	22.5%	1.0%

Table 2: % segments policed among lossy segments ($\geq$ 15 losses, the threshold to trigger the policing detector), and overall. Avg. loss rates for policed and unpoliced segments.

2. We analyze policing across the Internet based on global measurements (§4). We collected over 270 billion packets captured at Google servers over a 7-day span. This dataset gives us insight to traffic delivered to clients all over the world, spread across over 28,400 different autonomous systems (ASes).
3. We describe solutions for ISPs and content providers to mitigate adverse effects of traffic management (§5).

We find that between 2% and 7% of lossy transmissions (depending on the region) have been policed (Table 2).[1] While we detected policing in only 1% of samples overall in our dataset, connections with packet loss perform much worse than their loss-free counterparts [22, 68]. Thus, understanding and improving the performance for lossy transmissions can have a large impact on average performance [22]. We find that policing induces high packet loss overall: on average, a policed connection sees over 20% packet loss vs. at most 4.1% when no policing is involved. Finally, policing can degrade video playback quality. Our measurements reveal many cases in which policed clients spend 15% or more of their time rebuffering, much more than non-policed connections with similar goodput. With every 1% increase in rebuffering potentially reducing user engagement by over 3 minutes [18], these results would be troubling for any content provider.

While this study primarily highlights the negative side effects of policing, our point is not that all traffic management is bad. ISPs need tools to handle high traffic volumes while accommodating diverse service agreements. Our goal is to spur the development of best practices which allow ISPs to achieve management needs *and* better utilize networks, while also enabling content providers to provide a high-quality experience for *all* customers. As a starting point, we discuss and evaluate how ISPs and content providers can mitigate the adverse effects of traffic management (§5).

Stepping back, this paper presents an unprecedented view of the Internet: a week of (sampled) traffic from most of Google's CDN servers, delivering YouTube, one of the largest volume services in the world serving 12-32% of traffic worldwide [55]; a global view of aspects of TCP including loss rates seen along routes to networks hosting YouTube's huge user base; measurements of policing done by the middleboxes deployed in these networks; and statistics on client-

[1]The video traffic we examine is delivered in *segments* (or *chunks*), thus we analyze the dataset on a per-segment granularity. Many video content providers stream video in segments, permitting dynamic adaptation of delivery to network changes.

side quality of experience metrics capturing how this policing impacts users. The analysis pipeline built for this paper enabled this scale of measurement, whereas previous studies, even those by large content providers like Google, were limited to packet captures from fewer vantage points [2, 20, 22, 27, 38, 52, 68].

2. DETECTING & ANALYZING POLICING AT SCALE

In this section, we present an algorithm for detecting whether a (portion of a) flow is policed or not from a server-side trace. We added this algorithm to a collection and analysis framework for traffic at the scale of Google's CDN.

2.1 Detecting Policing

Challenges. Inferring the presence of policing from a server-side packet trace is challenging for two reasons. First, many entities can affect traffic exchanged between two endpoints, including routers, switches, middleboxes, and cross traffic. Together they can trigger a wide variety of network anomalies with different manifestations in the impacted packet captures. This complexity requires that our algorithm be able to rule out other possible root causes, including congestion at routers.[2] The second challenge is to keep the complexity of policing detection low to scale the detection algorithm to large content providers.

Definition. Traffic policing refers to the enforcement of a rate limit by dropping any packets that exceed the rate (with some allowance for bursts). Usually, traffic policing is achieved by using a token bucket of capacity N, initially filled with m tokens. Tokens are added (maximum N tokens in the bucket) at the preconfigured *policing rate* r. When a packet of length p arrives, if there are $\geq p$ tokens available, the policer forwards the packet and consumes p tokens. Otherwise it drops the packet.

Goal. The input to our algorithm is an annotated packet flow. Our analysis framework (§2.2) annotates each packet to specify, among other things: the packet acknowledgement latency, as well as packet loss and retransmission indicators.

Our goal is to detect when traffic is policed, *i.e.*, when a traffic policer drops packets that exceed the configured rate. Our approach uses loss events to detect policing (as described below).[3] If a flow requires fewer than m tokens, policing will not kick in and drop packets, and we do not attempt to detect the inactive presence of such a policer. The output of the algorithm is (a) a single bit that specifies whether the flow was policed or not, and (b) an estimate of the policing rate r.

Detection. Figure 2 outlines our *policing detector* (PD, for short). PD starts by generating the estimate for the token refresh rate r, as follows. We know that a policer drops packets when its token bucket is empty. Assuming losses are policer-induced, we know there were not enough tokens when the first loss (p_{first_loss}) and last loss (p_{last_loss}) happened within a flow. All successfully delivered packets in-between must have consumed tokens produced after the first loss. Thus, PD uses the goodput between the first and last loss to compute the token production rate (line 1).[4] Our algorithm is robust even if some losses have other root causes, such as congestion, so long as most are due to policing.

Variable: r (estimated policing rate)
Variable: p_{first_loss}, p_{last_loss} (first/last lost packet)
Variable: t_u, t_p, t_a (used/produced/available tokens)
Variable: l_{loss}, l_{pass} (lists of # tokens available when packets were lost/passed)
Variable: n_{loss}, n_{pass} (fraction of lost/passed packets allowed to not match policing constraints)

1 $r \leftarrow$ `rate`$(p_{first\_loss}, p_{last\_loss})$;
2 $t_u \leftarrow 0$;
3 **for** $p_{current} \leftarrow p_{first\_loss}$ **to** p_{last_loss} **do**
4 $t_p \leftarrow r \cdot ($`time`$(p_{current}) -$ `time`$(p_{first\_loss}))$;
5 $t_a \leftarrow t_p - t_u$;
6 **if** $p_{current}$ *is lost* **then**
7 Add t_a to l_{loss};
8 **else**
9 Add t_a to l_{pass};
10 $t_u \leftarrow t_u +$ `bytes`$(p_{current})$;
11 **if** `average`$(t_a$ *in* $l_{loss})$ $<$ `average`$(t_a$ *in* $l_{pass})$
12 **and** `median`$(t_a$ *in* $l_{loss})$ $<$ `median`$(t_a$ *in* $l_{pass})$
13 **and** $|[t_a \in l_{loss} : t_a \approx 0]| \geq (1 - n_{loss}) \cdot |l_{loss}|$
14 **and** $|[t_a \in l_{pass} : t_a \gtrsim 0]| \geq (1 - n_{pass}) \cdot |l_{pass}|$
15 **and** *RTT did not increase before* p_{first_loss} **then**
16 Add traffic policing tag to flow;

Figure 2: Policing Detector

Next, PD determines if the loss patterns are consistent with a policer enforcing rate r. To do so, it estimates the bucket fill level as each packet arrives at the policer and verifies if drops are consistent with expectation. For this estimation, it computes the following values *for each packet* between the first and the last loss (lines 3–10).

- The number of produced tokens t_p, *i.e.*, the overall (maximum) number of bytes that a policer would let through up to this point (line 4), based on the goodput estimate and the elapsed time since the first loss ($t_{elapsed} =$ `time`$(p_{current}) -$ `time`(p_{first_loss})).
- The number of used tokens t_u, *i.e.*, the number of bytes that passed through the policer already (line 10).
- The number of available tokens t_a, *i.e.*, number of bytes that a policer currently would let through based on the number of produced and already used tokens (line 5).

If the number of available tokens is roughly zero, *i.e.*, the

[2] Whether tail-drop or those using some form of active queue management, such as Random Early Drop (RED) or CoDel [24, 48].

[3] Since we rely on loss signals, we only detect policing when a flow experiences loss. To be robust against noise, we only run the algorithm on flows with 15 losses or more. We derived this threshold from a parameter sweep, which found that lower thresholds often produced false positives. On average, flows marked as policed in our production environment carried about 600 data packets out of which 100 or more were lost.

[4] If the first and/or last loss are not triggered by policing we potentially miscalculate the policing rate. To add robustness against this case, we always run the algorithm a second time where we cut off the first and last two losses and reestimate the policing rate.

token bucket is (almost) empty, we expect a packet to be dropped by the policer. Conversely, if the token count is larger than the size of the packet, *i.e.*, the token bucket accumulated tokens, we expect the packet to pass through. The exact thresholds depend on the goodput and the median RTT of the connection to account for the varying offsets between the transmission timestamp of packets that we record and the arrival times at the policer.

Based on this intuition, PD infers traffic policing if all of the following conditions hold (lines 11–15). First, the bucket should have more available tokens when packets pass through than when packets are lost. Second, we expect the token bucket to be roughly empty, *i.e.*, $t_a \approx 0$ in the case of a lost packet. This check ensures that losses do not happen when the token bucket is supposed to have sufficient tokens to let a packet pass ($t_a \gg 0$), or when the token bucket was supposed to be empty and have dropped packets earlier ($t_a < 0$). We allow a fraction of the samples (at most n_{loss}) to fail this condition for robustness against noisy measurements and sporadic losses with other root causes. A similar condition applies to the token counts observed when packets pass through, where we expect that the number of available tokens is almost always be positive. We allow fewer outliers here (at most $n_{pass} < n_{loss}$) since the policer always drops packets when the token bucket is empty. We derived the noise thresholds n_{loss} and n_{pass} from a parameter sweep in a laboratory setting (§3.1) with a preference for keeping the number of false positives low. For our analysis, we used $n_{loss} = 0.1$ and $n_{pass} = 0.03$. Finally, PD excludes cases where packet losses were preceded by RTT inflation that could not be explained by out-of-order delivery or delayed ACKs. This check is another safeguard against false positives from congestion, often indicated by increasing buffering times and RTTs before packets are dropped due to queue overflow.

By simulating the state of a policer's token bucket and having tight restrictions on the instances where we expect packets to be dropped vs. passed through, we reduce the risk of attributing losses with other root causes to interference by a policer. Other causes, like congestion, transient losses, or faulty router behavior, will, over time, demonstrate different connection behaviors than policing. For example, while a policed connection can temporarily achieve a goodput above the policing rate whenever the bucket accumulates tokens, a connection with congestion cannot do the same by temporarily maintaining a goodput above the bottleneck rate. Thus, over time the progress on connections affected by congestion will deviate from progress seen on policed connections.

2.2 Analyzing Flow Behavior At Scale

We have developed, together with other collaborators within Google, a pipeline for analyzing flows at scale. The first step of this pipeline is a *sampler* that efficiently samples a small fraction of all flows based on 5-tuple hashes, capturing all the headers and discarding the payload after the TCP header. The sampler is deployed at most of Google's CDN servers and periodically transfers collected traces to an analyzer backend in a datacenter. By running the analysis online in a datacenter, we minimize the processing overhead introduced on the CDN servers. As the traces arrive at the analyzer backend, an *annotator* analyzes each flow. We designed the annotator to be broadly applicable beyond detecting policing; for example, in §5.1.2, we use it to detect traffic shaping. For each trace, the annotator derives annotations at the individual packet level (*e.g.*, the RTT for the packet, or whether the packet was lost and/or a retransmission), and at the flow level (*e.g.*, the loss rate and average throughput experienced by the flow). It can also identify application-level frames within a flow, such as *segments* (or *chunks*) in a video flow. The annotator also captures more complex annotations, such as whether a connection experienced bufferbloat [26]. PD is just one component of the annotator: it annotates whether a segment was policed and, if so, at what rate.

Developing these annotations was challenging. The annotation algorithms had to be fast since a single trace might need several hundred annotations and we have many traces. The more complex annotations also required significant domain knowledge and frequent discussions with experienced network engineers looking at raw packet traces and identifying higher-level structures and interactions. Also complicating the effort were the complexity of the TCP specification, implementation artifacts, and application and network element behavior that led to a very large variety in observed packet traces. Our annotator is a significant step in packet analysis at scale beyond existing tools [10, 12, 45, 51, 53, 58, 61, 63]. Our analysis framework helped us explore policing in the wild and was also helpful in iterating over different designs of complex annotations. The framework can detect CDN-wide anomalies in near real-time (*e.g.*, when traffic from an ISP experiences significant loss).

3. VALIDATION

We validate our algorithm in two ways. First, we evaluate the accuracy of PD by generating a large set of packet traces in a controlled lab setting with ground truth about the underlying root causes for packet loss (§3.1). Second, we show that the policing rates in the wild are consistent within an AS, meaning the AS's traces marked as policed have goodput rates that cluster around only a few values, whereas the remaining traces see goodput rates that are dispersed (§3.2).

3.1 Lab validation

Our lab experiments are designed to stress-test our algorithm. We generated a large number of packet traces while using different settings that cover common reasons for dropped packets, focusing on the ones that could elicit traffic patterns similar to a policed connection.

- **Policing.** We use a carrier-grade network device from a major router vendor to enforce traffic policing. We configured the device in much the same way an ISP would to throttle their users, and we confirmed with the router vendor that our configurations are consistent with ISP practice. Across multiple trials, we set the policing rates to 0.5, 1.5, 3, and 10 Mbps, and burst

	Scenario	Accuracy
A	Policing (except (B) and (C) below)	93.1%
B	Policing (special cases)	48.0%
C	Policing (multiple flows)	12.3%
D	Congestion (all AQM schemes)	100.0%
E	Congestion (drop-tail, single flow, except (G))	100.0%
F	Random loss	99.7%
G	Congestion (drop-tail, single flow, min. queue)	93.2%
H	Congestion (drop-tail, multiple flows)	96.9%

Table 3: PD classification accuracy for several controlled scenarios.

sizes to 8kB, 100kB, 1MB, and 2MB.

- **Congestion.** We emulate a bottleneck link which gets congested by one or multiple flows. We evaluated drop-tail queueing and three active queue management (AQM) schemes: CoDel [48], RED [24], and PIE [50]. We varied bottleneck link rates and queue sizes across trials using the same values as for the policing scenario.
- **Random loss.** We used a network emulator to randomly drop 1% and 2% of packets to simulate the potential behavior of a faulty connection.

We simulated traffic resembling the delivery of data chunks for a video download, similar to the type of traffic we target in our analysis in §4. Overall, we analyzed 14,195 chunks and expected our algorithm to mark a chunk as policed if and only if the trace sees packet loss and was recorded in the *Policing* setting. Table 3 summarizes the results, with a detailed breakdown of all trials available online [1].

Policed traces. PD was able to detect policing 93% of the time for most policing configurations (A). The tool can miss detecting policing when it only triggers a single large burst of losses,[5] or when the token bucket is so small that it allows almost no burstiness and is therefore similar in behavior to a low-capacity bottleneck with a small queue. We aggregated these cases as special cases (B). PD is conservative in order to avoid false positives for non-policed traces (D–H). Consequently, we likely underestimate global policing levels by failing to recognize some of the special cases (B). We also analyzed the scenario where multiple flows towards the same client are policed together (C). For our in-the-wild study (§4), PD typically does not have visibility into all flows towards a single client, as the CDN servers in the study independently select which flows to capture. To emulate this setting in our validation, we also only supply PD with a single flow, and so it can only account for some of the tokens that are consumed at the policer. Therefore, its inference algorithm is unable to establish a single pattern that is consistent with policing at any given rate. Since we are interested in a macroscopic view of policing around the globe, we can tolerate a reduced detection accuracy for cases where clients occasionally receive content through multiple connections at the same time.[6] We leave improving the algorithm's accuracy for this scenario to future work which would also require the deployment of a different sampling method.

[5]Given only a single burst of losses, we cannot estimate a policing rate since all losses happened at roughly the same time.

[6]For video transfers in our setting, most of the time only one connection is actively transmitting a video chunk from the server to the client, even though multiple connections are established between them.

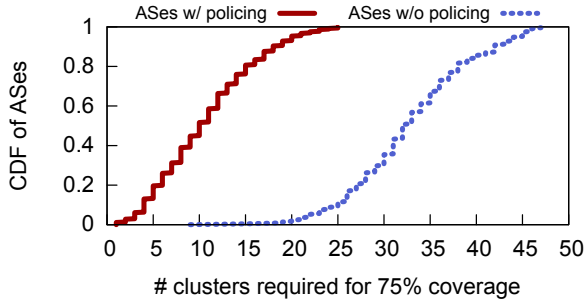


Figure 3: Number of rate clusters required to cover at least 75% of the rate samples per AS.

Non-policed traces. In our experiments, PD correctly classifies as non-policed almost all segments suffering from other common network effects, including network bottlenecks such as a congested link with packets dropped due to AQM (D) or drop-tail policy (E, G, H), and random packet loss (F). PD is able to rule out policing because it checks for consistent policing behavior across many RTTs, and other network effects rarely induce loss patterns that consistently mimic the policing signature over time. For example, when congestion overflows a queue, it drops packets similar to a policer that has exhausted tokens. However, over time congestion will not always happen at exactly the same moment as a policer enforcing the rate limit for a specific flow.

A closer look at the single-flow congestion cases shows that only trials using the minimum configurable queue size (8 kB) cause misclassifications (G). This is because a bottleneck with almost no available queue size to temporarily accommodate bursts results in the same packet loss patterns as traffic passing through a policer. However, in the wild (§4), 90% of the traces tagged as policed temporarily sustain larger bursts of 30 kB or more and therefore cannot fall in this category of false positives. In addition, a few cases of congestion from background traffic (H) induced loss patterns that were misclassified as policing. These cases have inferred bottleneck rates that vary widely, whereas we show in §3.2 that, in the wild, traces we classified as policed cluster around only a handful of goodput rates per AS. Note that a flow in the wild might experience more complex congestion dynamics, *e.g.*, when contending with hundreds of other flows at a router. However, these dynamics are unlikely to result in a per-chunk traffic pattern consistent with a policer enforcing a rate (*e.g.*, where losses always happen when exceeding a certain throughput rate), and, even if there are cases where chunks are misclassified as policed, we do not expect this to happen consistently for a large number of chunks within an AS.

Finally we validated our algorithm against traces generated by Kakhki *et al.* [32]. These traces were also generated with carrier grade equipment, configured to perform traffic shaping only. As such, none of the traces should be labeled as policed by our tool. The 1,104 traces we analyzed contained 205,652 data chunks, of which only 37 chunks were falsely marked as policed by PD. This represents an accuracy of 99.98% for this dataset.

3.2 Consistency of Policing Rates

Our case studies (discussed later in §4.5) suggest that policing rates are often tied to advertised data plan rates. Thus we conjectured that, because most ASes have few plan rates, we should observe few policing rates per AS. To validate this conjecture, we computed the number of prevalent policing rates seen *per AS*, based on traces from most of Google's CDN servers (see §4). We derived the minimum number of *rate clusters* required to cover at least 75% of the policed traces per AS. We define a rate cluster with center value v as all rates falling into the range $[0.95 \cdot v, 1.05 \cdot v]$. For example, the 1-Mbps cluster incorporates all rates that are $\geq$ 0.95 Mbps and $\leq$ 1.05 Mbps. To find a solution, we use the greedy algorithm for the partial set cover problem which produces a good approximation of the optimal solution [40].

We looked at the distribution of goodput rates for segments marked as policed in ASes with at least 3% of their traffic being policed. Rates in the majority of ASes can be accounted for by 10 clusters or less (Figure 3). By visiting ISP homepages, we observe that many offer a range of data rates, some with reduced rates for data overages. Further, many ISPs continue to support legacy rates. Thus it is not surprising that we see more than just a couple of policing rates for most ASes. In contrast, goodput rates in ASes with no policing do not display clustering around a small number of rates and see a much wider spread. Since the false positives in our lab validation see a wide spread as well, this result provides us confidence that the traces we marked as policed in our production dataset are mostly true positives.

4. POLICING IN THE WILD

In this section, we characterize the prevalence and impact of policing in the Internet.

The dataset. We analyze sampled data collected from most of Google's CDN servers during a 7-day period in September 2015. The dataset consists of over 277 billion TCP packets, carrying 270 TB of data, associated with more than 800 million HTTP queries requested by clients in over 28,400 ASes. The TCP flows carried different types of content, including video segments associated with 146 million video playbacks. The dataset is a sampled subset (based on flow ID hashing) of Google's content delivery traffic. To tie TCP performance to application performance, we analyze the data at a *flow segment* granularity. A segment consists of the packets carrying an application request and its response (including ACKs).

Overview of Results. In the following sub-sections, we present our key findings:

- Especially in Africa, a sizable amount of throttled traffic is limited to a rate of 2 Mbps or less, often inhibiting the delivery of HD quality content (§4.1).
- Policing can result in excessive loss (§4.2).
- The user quality of experience suffers with policing, as measured by more time spent rebuffering (§4.3).
- Policing can induce patterns of traffic and loss that interact poorly with TCP dynamics (§4.4).

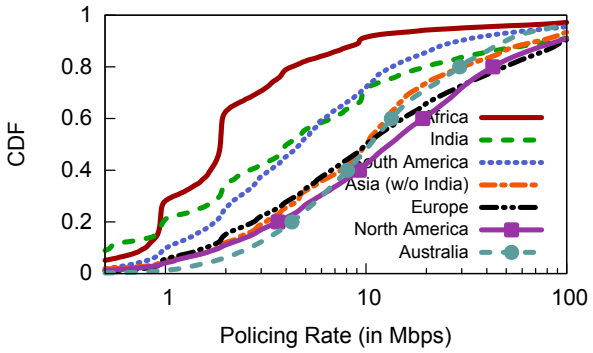


Figure 5: Observed policing rates per segment.

- Through ISP case studies, we reveal interesting policing behavior and its impact, including losses on long-distance connections. We also confirm that policing is often used to enforce data plans (§4.5).

As an aside, we conducted a supplemental study on the publicly available M-Lab NDT dataset[7] using the same detection algorithm [1, 23]. The results from the NDT dataset support this paper's findings in terms of the prevalence and impact of policing in the wild. Our technical report includes further analysis of policing rates within individual ISPs, per-country breakdowns, and longitudinal trends seen over the past seven years.

4.1 The Prevalence of Policing

A macroscopic analysis of the data (Table 2) shows that, depending on geographic region, between 2% and 6.8% of lossy segments were impacted by policing. Overall, between 0.2% and 1.4% of the segments were affected.

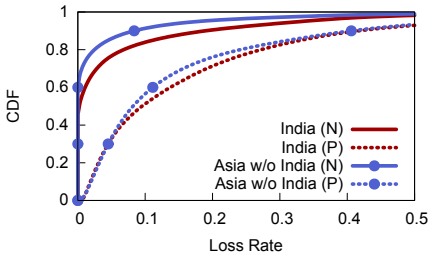
Which policing rates are prevalent across the globe? Figure 5 shows the rates enforced by policers. In Africa and India, over 30% of the policed segments are throttled to rates of 2 Mbps or less. The most frequent policing rates in these two regions are 1, 2, and 10 Mbps, as is evident from the pronounced inflections in the CDF. In §4.5 we examine some ISPs to demonstrate that this step-wise pattern of policing rates that emerge in the data reflects the available data plans within each ISP. The distributions in other regions of the world show no dominant rates, with many segments being permitted to transmit at rates exceeding 10 Mbps. This is due to aggregation effects: these regions have many ISPs with a wide variety of data plans. That said, even in these regions, at least 20% of segments stream at less than 5 Mbps.

4.2 Impact of Policing on the Network

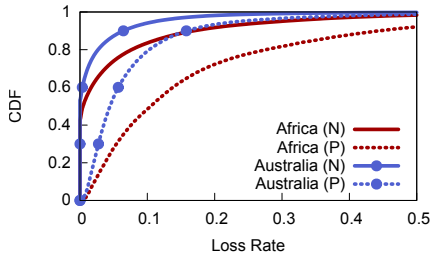
Policing noticeably increases the packet loss rate, which can in turn affect TCP performance [22, 68] and user satisfaction.[8] In our dataset, we observed an average packet loss

[7] http://measurementlab.net/tools/ndt

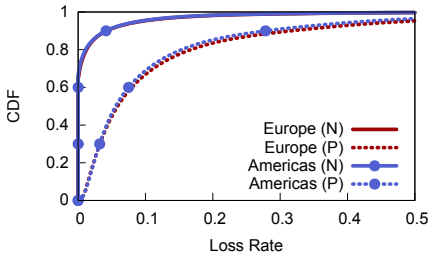
[8] To ensure that packet loss is caused by policing instead of only being correlated with it (*e.g.*, in the case where policing would be employed as a remedy to excessive congestion in a network), we compared the performance of policed and unpoliced flows within an AS (for a few dozen of the most policed ASes). We verified that policed connections observed low throughput yet high loss rates. Conversely unpoliced connections achieved high throughput at low loss rates. In addition, we did not observe any diurnal patterns – loss rates and the fraction of traffic impacted by policing are



(a) Asia



(b) Africa and Australia



(c) Europe and Americas

Figure 4: Distribution of loss rates observed on unpoliced (N) and policed (P) segments in different regions of the world.

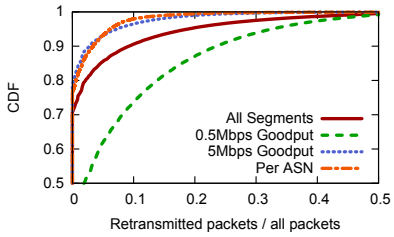


Figure 6: Loss rate CDF per segment, for segments with avg. goodput of 0.5 or 5 Mbps, and per ASN.

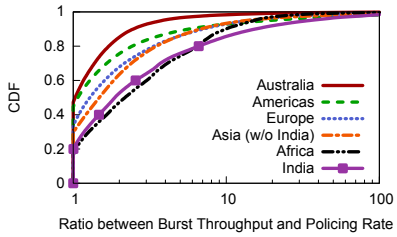


Figure 7: Ratio between the median burst throughput and the policing rate per segment.

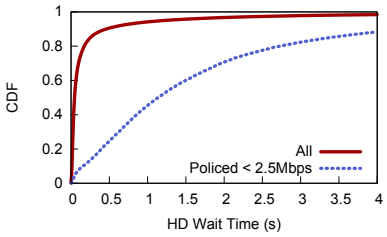


Figure 8: Wait time CDF for all HD segments (red solid line) and those policed below 2.5 Mbps (blue dotted line).

rate of 22% per segment for policed flows (Table 2).

Figure 4 plots the loss rate CDF for policed and non-policed segments observed in different regions. Policed flows in Africa and Asia see a median loss rate of at least 10%, whereas the median for unpoliced flows is 0%. Other regions witness lower loss rates, yet a sizable fraction of segments in each experiences rates of 20% or more. The 99th percentile in all regions is at least 40%, *i.e.*, almost every other packet is a retransmission. In §4.4 we analyze common traffic patterns that can trigger such excessive loss rates.

The loss rate distributions shown in Figure 6 see a wide variability with long tails: the overall loss rate distribution (*All Segments*) has a median of 0% and a 99th percentile of over 25%. The figure also shows the distribution for two segment subsets: one including the 20 million requests with an average goodput of 0.5 Mbps (±50 kbps), and the other with the 7 million requests achieving 5 Mbps (±50 kbps). Though there is some correlation between goodput and loss rates, there are many cases where high loss did not result in bad performance. For example, about 4% of the segments achieving a goodput of 5 Mbps also observe a loss rate of 10% or more. Policers are one cause for the uncommon *high loss, high goodput* behavior, as we show in §4.4.

One situation that can trigger high loss is when there is a wide gap between the rate sustained by a flow's bottleneck link and the rate enforced by the policer. We estimate the bottleneck capacity (or the burst throughput) by evaluating the interarrival time of ACKs for a burst of packets [13, 28, 35]. We found that in many cases the bottleneck capacity, and sometimes even the goodput rate achieved before the policer starts dropping packets is 1-2 orders of magnitude higher than the policing rate. Figure 7 compares the achieved burst throughput and policing rates we observed. The gap is particularly wide in Africa and India. With such large gaps, when the policer starts to drop packets, the sender may already be transmitting at several times the policing rate. Since the sender's congestion control mechanism usually only halves the transmission rate each round trip, it needs multiple round trips to sufficiently reduce the rate to prevent further policer packet drops. We investigate this and other interactions with TCP in §4.4.

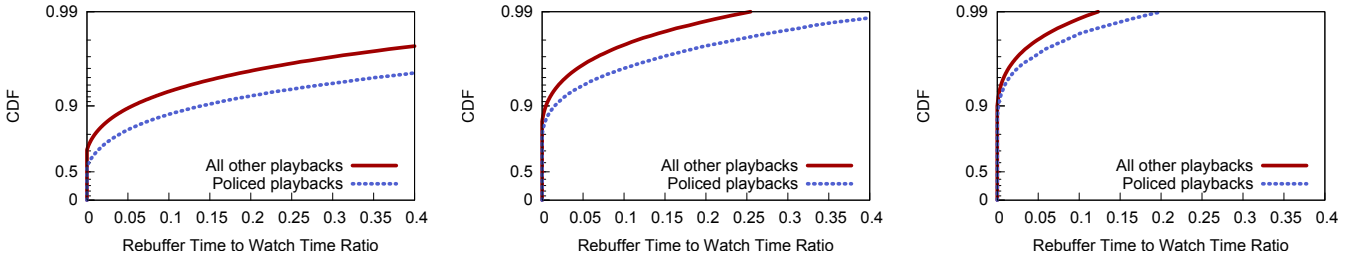
When policers drop large bursts of packets, the sender can end up retransmitting the same packets multiple times. Overshooting the policing rate by a large factor means that retransmissions as part of Fast Recovery or FACK Recovery [44] are more likely to also be lost, since the transmission rate does not decrease quickly enough. The same applies to cases where policing results in a retransmission timeout (RTO) followed by Slow Start. In this situation, the token bucket accumulated tokens before the RTO fired, leading to a few rounds of successful retransmissions before the exponential slow start growth results in overshooting the policing rate again, requiring retransmissions of retransmissions. Multiple rounds of this behavior can be seen in Figure 1.

These loss pathologies can be detrimental to both ISPs and content providers. Policing-induced drops force the content provider to transmit, and ISPs to carry, significant retransmission traffic. This motivates our exploration of more benign rate-limiting approaches in the §5.

4.3 Impact on Playback Quality

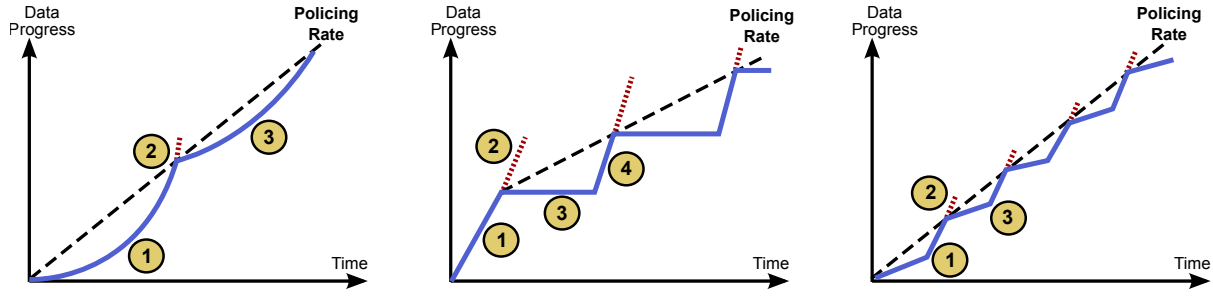
In addition to the large overheads caused by excessive packet loss, policing has a measurable impact on the user's quality of experience. Figure 9 shows, for a selection of playbacks delivered at different goodput rates, the distribution of the ratio of time spent rebuffering to time spend watching. This ratio is an established metric for playback quality

not affected by the presence of peak times. §3 provides additional evidence that policers are the root cause for losses and not the other way round.



(a) 300 kbps (b) 1.5 Mbps (c) 3 Mbps

Figure 9: Rebuffer to watch time ratios for video playbacks. Each had at least one chunk with a goodput of 300 kbps, 1.5 Mbps, or 3 Mbps ($\pm 15\%$).



(a) Congestion avoidance pattern (b) Staircase pattern (c) Doubling window pattern

Figure 10: Common traffic patterns when a traffic policer enforces throughput rates. The plots show progress over time (blue solid line) with a steeper slope representing a higher goodput, the transmitted but lost sequences (red dotted lines), and the estimated policing rate (black dashed line). Packets which would put the progress line above the policing rate line are dropped while other packets pass through successfully.

and previous studies found a high correlation between this metric and user engagement [18]. Each of the selected playbacks had at least one of their video segments delivered at a goodput rate of either 300 kbps, 1.5 Mbps, or 3 Mbps ($\pm 15\%$). 300 kbps is the minimum rate required to play videos of the lowest rendering quality, leaving little opportunity to bridge delayed transmissions by consuming already buffered data. For each selected rate, between 50% and 90% of the playbacks do not see any rebuffer events. For the rest, policed playbacks perform up to 200% worse than the unpoliced ones. For example, in the 90th percentile, playbacks policed at a rate of $\approx$ 300 kbps spend over 15% of their time rebuffering, vs. 5% when not policed. Prior work found that a 1% increase in the rebuffering ratio can reduce user engagement by 3 minutes [18]. This result substantiates our claim that policing can have a measurable negative impact on user experience.

Another way to assess playback quality is to explore the impact of observing a high-goodput short burst at the beginning of the flow, before policing starts. This can happen when the policer's token bucket starts out with a sizable amount of tokens. As such, a flow might temporarily sustain a rate that is good enough for HD video delivery, while the policing rate enforced later prevents this, *i.e.*, the rate is below the target of 2.5 Mbps. To quantify the impact of this behavior on the application, we evaluate the *wait time*. This is the delay between a segment request and the time when its playback can commence without incurring additional rebuffering events later. We can compute wait time from our traces since we can observe the complete segment behavior.

Figure 8 shows that delivering even a single HD segment over a slow connection results in larger wait times. In the median, a client has to wait over 1 second for a policed segment, whereas the median for unpoliced ones is only 10 ms.

4.4 Interaction Between Policers and TCP

Enabling traffic policing itself does not automatically result in high loss. Thus, before we can design solutions to avoid the negative side effects of policing, we need to have a better understanding about when and why configurations trigger heavy losses. We found that high loss is only observed when the policer and TCP congestion control interact poorly in specific settings. To depict these interactions, we use the diagrams in Figure 10 that show specific *patterns* of connection progress.

Congestion Avoidance Pattern. In the most benign interaction we have seen, the policer induces few losses over long time periods. The congestion window grows slowly, never overshooting the policing rate by much. This results in short loss periods, as shown in Figure 10a.

In this pattern, the sender slowly increases the congestion window while a small number of excess tokens accumulate in the bucket (1). Towards the end of this phase, the progress curve has a slightly steeper slope than the policing rate curve. Consequently, we exceed the policing rate at some point (the black dashed line) resulting in packet loss (2). The congestion window is reduced during the fast recovery, followed by another congestion avoidance phase (3).

Staircase Pattern. A particularly destructive interaction between TCP and policers is a "staircase" when flow rates be-

fore the policer drops packets are multiple times the policed rate (Figure 10b). This results in short periods of progress followed by long periods of stagnation, with the sequence graph resembling a staircase.

Initially the sender pushes data successfully at a high rate (bubble 1 in the figure). Eventually, the policer runs out of tokens and starts dropping. Since the token refresh rate is much lower than the transmission rate, (almost) all packets are lost (2). This results in a high probability of the last packet in a burst being lost, so TCP needs to fall back to timeout-based loss detection, since there are no subsequent packets to trigger duplicate ACKs. Consequently, the sender idles for a long time (3). This is problematic on low-RTT connections, since the loss detection mechanism accounts for possibly delayed ACKs, usually requiring a timeout of 200 ms or more [7], which may be much higher than the RTT. Once packets are marked as lost and retransmitted, the sender accelerates quickly (4), as the policer accumulated a large number of tokens during the idle time. In §5.1.1 and §5.2.2 we discuss how we can avoid this pattern by optimizing a policer's configuration and reducing bursty transmits.

Doubling Window Pattern. For clients near the server, the very low RTTs can enable connections to sustain high throughput rates even when the congestion window (cwnd) enables the sender to have only one packet carrying MSS bytes in flight at a time, where MSS is the maximum segment size allowed by the network. The throughput rate equals $\frac{cwnd}{RTT}$ excluding loss events. The policing rate lies between the throughputs achieved when using a congestion window of 1 MSS and a window of 2 MSS (see Figure 10c). Note that the window will grow linearly on a byte granularity, thus observing values between 1 and 2 MSS. However, Nagle's algorithm in TCP delays transmissions until the window allows the transmission of a full MSS-sized packet [46]. The pattern starts with the sender pushing data while using a congestion window of 1 MSS. In congestion avoidance mode, the window increases by 1 MSS every RTT. Thus, even though the window is supposed to grow slowly, it doubles in this extreme case (1). Next, the higher transmission rate makes the policer drop packets (2). The sender backs off, setting the congestion window back to 1 MSS. Timeout-based recovery isn't necessary since the low amount of in-flight data enables "early retransmit" upon the reception of a single duplicate ACK (3).

Even though the connection makes continuous progress without excessive loss periods, valuable bandwidth is wasted. To avoid this pattern the sender would need to send packets that carry fewer bytes than the MSS allows to match the policing rate. Since the protocol is not configured to do this, using a window of 1 MSS is the only setting enabling permanent stability. This is not supported by TCP's congestion control mechanism, since "congestion avoidance" will increase the window by 1 MSS every RTT.

4.5 Policing Pathologies

We now focus on the analysis of traces from a small set of ISPs to highlight different characteristics of policed traffic. Table 4 gives an overview of five ISPs where policing was prevalent, selected to illustrate interesting pathologies arising from policing.

ISP	ISP Region	Samples	RTT	Mobile
A	Azerbaijan	64K	Medium	
B	USA	31K	Medium	✓
C	India	137K	Very low	
D	India	17K	Low	
E	Algeria	112K	Medium	

Table 4: Overview of 5 highly policed ISPs. The RTT estimates apply only when content is fetched from the local cache. With cache misses content needs to be fetched from a data center which is potentially located much farther away, resulting in higher RTTs.

Figures 11 and 12 show the policing and loss rates seen when delivering video to clients in each ISP. We can clearly distinguish the small set of policing rates used within each ISP. The most popular choices are 1 and 2 Mbps, both of which are below the 2.5 Mbps needed for HD quality videos.

For all ISPs except ISP B, we found the advertised bandwidth of their data plans on their websites, and, in each case, the plan rates matched the observed policing rates.[9] For ISP C, we recently observed a drastic change in the rate distribution. In our earlier analysis from 2014, most traces were policed at 4 Mbps, at that point a plan offered by the ISP. Now we see 10 Mbps as the most prominent rate, which is consistent with a change of data plans advertised. We do observe two smaller bumps at roughly 3 Mbps and 4 Mbps. These rates do not correspond to a base bandwidth of any of their plans, but instead reflect the bandwidth given to customers once they exceed their monthly data cap.

Losses on long-distance connections. Traffic policing causes frequent loss, but losses can be particularly costly when the packets propagate over long distances just to be dropped close to the client. For example, for ISP A, a local cache node in Azerbaijan serves half the video requests, whereas the other half is served from more than 2,000 kilometers away. We confirmed that the policer operates regardless of content source. So the high drop rates result in a significant fraction of bandwidth wasted along the paths carrying the content. The same applies to many other ISPs (including C, D, and E) where content is sometimes fetched from servers located thousands of kilometers away from the client.

Policing in wireless environments. We observe policing in many areas across the globe, even in developed regions. ISP B provides mobile access across the United States while heavily policing some of its users to enforce a data cap. While we understand that it is necessary to regulate access by heavy users, we find that there are many cases where the bandwidth used by throttled connections is actually higher than the bandwidth used by unthrottled ones carrying HD content, since the latter do not incur costly retransmissions.

Large token buckets. ISP C sees heavy loss, with 90% of segments seeing 10% loss or more. Yet, flows achieve good-

[9] Since matching policing rates to data plans is a manual process, we only did this for the selected ISPs. However, it is unlikely that every ISP uses policing only to enforce data plans, and we leave a thorough root cause analysis to future work.

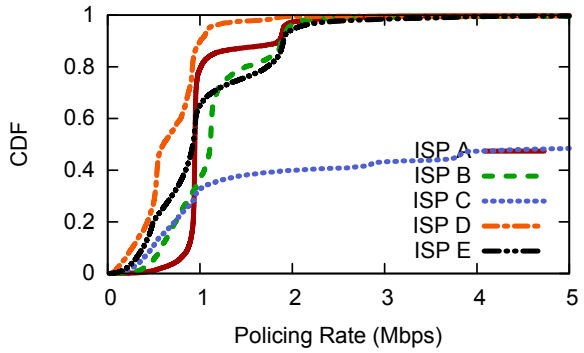


Figure 11: Policing rates in policed segments for selected ISPs.

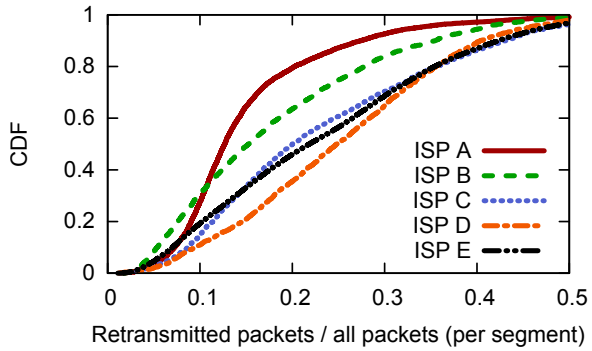


Figure 12: Loss rates in policed segments for selected ISPs.

puts that match the policing rates (10 Mbps or more in this case). There are three reasons for this. First, median bottleneck capacity is 50 Mbps on affected connections. Second, most connections see a very small RTT. Finally, the policer is configured to accommodate fairly large bursts, *i.e.*, buckets can accumulate a large number of tokens. This allows the connection to "catch up" after heavy loss periods, where progress stalls, by briefly sustaining a goodput rate exceeding the policing rate by an order of magnitude. When plotting the progress over time, this looks like a *staircase* pattern which was discussed in more detail in §4.4.

While goodputs are not adversely affected, application performance can still degrade. For example, a video player needs to maintain a large buffer of data to bridge the time period where progress is stalled, otherwise playback would pause until the "catch up" phase.

Small token buckets. ISP D is at the other end of the spectrum, accommodating no bursts by using a very small token bucket. The small bucket combined with the low RTT results in the *doubling window* pattern discussed earlier (§4.4). The small capacity also prevents a connection from "catching up." After spending considerable time recovering from a loss, the policer immediately throttles transmission rates again since there are no tokens available that could be used to briefly exceed the policing rate. As such, the overall goodput rate is highly influenced by delays introduced when recovering from packet loss.

Repressing video streaming. Finally, we note that we observed configurations where a video flow is throttled to a rate that is too small to sustain even the lowest quality. The small number of requests coming from affected users suggests that they stop watching videos altogether.

Cap. (KB)	8	16	32	64	128	256	512	1K	2K
Rebuf. (s)	3.5	2.0	1.5	1.6	1.6	1.6	2.4	3.1	3.1

Table 5: Impact of token bucket capacity on rebuffering time of the same 30-second video playback. Policing rate is set to 500 kbps.

5. MITIGATING POLICER IMPACT

We now explore several solutions to mitigate the impact of policing. Unless otherwise specified, we use the same setup as for the PD validation (see §3).

5.1 Solutions for ISPs

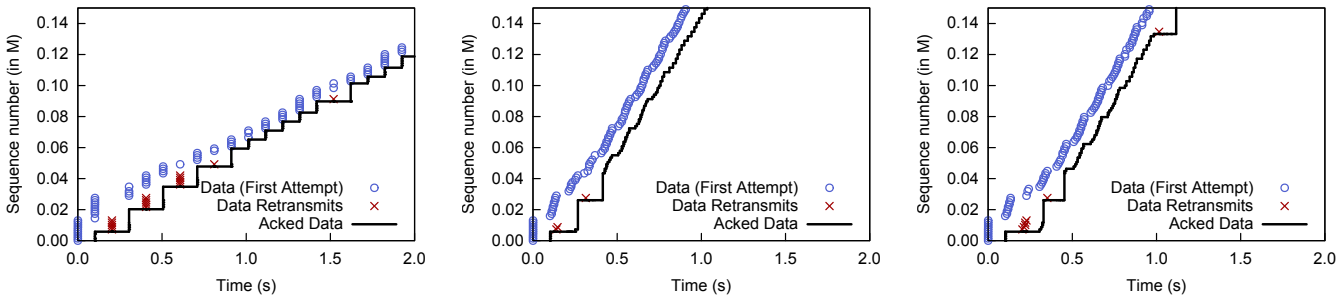
5.1.1 *Optimizing Policing Configurations*

The selection of configuration parameters for a policer can determine its impact. The policed rate usually depends on objectives such as matching the goodput to the bandwidth advertised in a user's data plan and therefore may be inflexible. However, an ISP can play with other knobs to improve compatibility between policers and the transport layer, while maintaining the same policing rate.

For example, we showed earlier that the staircase pattern can arise in the presence of large token buckets. To prevent the associated long bursty loss periods, two options come to mind. First, the enforcing ISP could configure policers with smaller burst sizes. This would prevent TCP's congestion window from growing too far beyond the policing rate. For this, we again measured the performance of video playbacks when traffic is passed through a policer. We limited the rate to 500 kbps and varied the burst size between 8 kB (the smallest configurable size) and 8 MB, using powers of two as increments. In this setting, a fairly small buffer size of 32 kB results in the lowest rebuffering delays (Table 5). Smaller buffers prevent the policer from absorbing any bursty traffic. Larger buffers allow connections to temporarily achieve throughput rates that are much larger than the policing rates, which can result in long rebuffering events if a quality level can no longer be sustained (*i.e.*, the player has to adjust to a lower bandwidth once traffic is policed) or if loss recovery is delayed (*i.e.*, we observe a staircase pattern). A more thorough sensitivity analysis is left to future work. Second, policing can be combined with shaping, as discussed below.

5.1.2 *Shaping Instead of Policing*

In contrast to a policer dropping packets, a traffic *shaper* enforces a rate r by buffering packets: if the shaper does not have enough tokens available to forward a packet immediately, it queues the packet until sufficient additional tokens accumulate. The traces of segments that pass through a shaper resemble those of segments limited by a bottleneck. Shaping can provide better performance than policing. It minimizes the loss of valuable bandwidth by buffering packets that exceed the throttling rate instead of dropping them immediately. However, buffering packets requires more memory. As with policers, shapers can be configured in different ways. A shaper can even be combined with a policer. In that case, the shaper spreads packet bursts out



(a) No modifications. (b) With sender-side rate limit. (c) With TCP pacing.

Figure 13: TCP sequence graphs for three flows passing through a policer with a token refresh rate of 1.5 Mbps and a bucket capacity of 8KB. The rate limit in (b) is set to 95% of the policing rate (*i.e.*, 1.425 Mbps).

evenly before they reach the policer, allowing tokens to generate and preventing bursty losses. One key configuration for a shaper is whether to make it *burst-tolerant* by enabling a "burst" phase. When enabled, the shaper temporarily allows a goodput exceeding the configured shaping rate, similar to Comcast's Powerboost feature [5, 6].

Burst-tolerant Shapers. We developed a detection algorithm for burst-tolerant shaping which determines whether a given segment has been subjected to this type of shaper, and estimates the shaping rate. It relies on the observation that a connection achieves a *steady* throughput rate after an initial burst phase with higher throughput. We have omitted the details of this algorithm for brevity. We found burst-tolerant shaping in 1.5% of the segments in our dataset.

Given its prevalence, we ask: can burst-tolerant shaping mitigate the adverse impact of policing? While shaping avoids the losses that policing induces, latency can increase as shapers buffer packets. To measure this effect, for each video chunk we compare the 10th percentile latency, usually observed in the burst phase, with the 90th (Figure 14). In the median, shaped segments observe a 90th percentile latency that is 4× larger than the 10th percentile. About 20% of segments see a latency bloat of at least an order of magnitude due to traffic shaping, whereas, among non-shaped segments, only 1% see such disparity. Latency-aware congestion control (*e.g.*, TCP Vegas [8]) or network scheduling algorithms (*e.g.*, CoDel [48]) can reduce this latency bloat.

Burst-tolerant shaping can also induce unnecessary rebuffering delays at the client. When shaping forces a video server to switch from the burst rate to the actual shaping rate, the content provider may reduce the quality delivered to the client based on the new bandwidth constraint. Now, the older high quality chunk takes too long to be delivered to the client, whereas the new low quality chunk does not reach before the client-side application buffer has already drained.

Shapers without burst tolerance. The alternative is shapers that enforce the shaping rate from the start. In theory, such shaping should not induce significant delays (unlike their burst-tolerant counterparts), nor drop packets like policers.

Our dataset almost certainly includes flows shaped in this way, but detecting them is hard: connections affected by shaping produce the same traffic patterns as when a TCP flow hits a bottleneck at the same rate. Significant cross traffic sharing such a bottleneck may cause throughput variance. It may be possible to identify (burst-intolerant) shapers by looking for low variance. However, since our passive measurements cannot detect when cross traffic is present, we cannot infer these shapers with any reasonable accuracy.

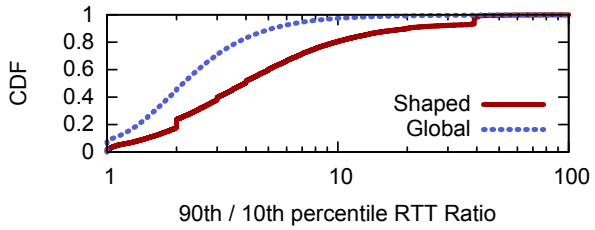


Figure 14: Per-segment ratio between 90th and 10th percentile latencies for shaped segments (red solid line) and all video segments globally (blue dashed line).

Cap.	Join Time (s)			Rebuffer Time (s)		
	(Policed)	(Shaped)	Diff.	(Policed)	(Shaped)	Diff.
8 kB	14.0	12.0	–16%	2.8	1.7	–39%
100 kB	11.1	13.3	+20%	1.6	1.3	–19%
2 MB	0.3	12.6	+4200%	4.2	1.5	–64%

Table 6: Avg. join/rebuffer times for first 30 s of a video with the downlink throttled to 0.5 Mbps by either a policer or shaper. Capacity (cap.) is the token bucket size (for policer) and the queue size (for shaper).

We evaluate the efficacy of shapers in the lab by fetching the same video playback repeatedly from YouTube and passing it through a policer or shaper (experimenting with different bandwidth limits and queue sizes) before the traffic reaches the client. Then, we calculated quality metrics using YouTube's QoE API [65]. Table 6 summarizes the impact on QoE, averaged over 50 trials per configuration.[10] Join times are generally lower when policing is used, since data can initially pass through the policer without any rate limit if enough tokens are buffered. With sufficiently large token buckets (*e.g.*, the 2 MB configuration in Table 6) a video playback can start almost immediately. However, this comes at the cost of much higher rebuffering times. The sudden enforcement of a policing rate causes the video player buffer to drain, causing high rebuffering rates. Shaping on the other hand enforces a rate at all times without allowing bursts. This reduces rebuffering by up to 64% compared to

[10] We show the results for a single throttling rate here, with other rates yielding similar trends.

the policed counterparts. Since prior work found that a low rebuffering time increases user engagement [18], reducing rebuffering time might be more beneficial than optimizing join times. Interestingly, shaping performs well even when the buffer size is kept at a minimum (here, at 8 kB) which only allows the absorption of small bursts.

Configuring shapers. While policer configurations should strive to minimize burst losses, there is no straightforward solution for shapers. Shaping comes at a higher memory cost than policing due to the buffer required to store packets. However, it also introduces queuing latency which can negatively affect latency-sensitive services [39]. Thus, ISPs that employ shaping have to trade off between minimizing loss rates through larger buffers that introduce higher memory costs, and minimizing latency through small buffers. In comparison to the cheaper policing option, a small buffer might still be affordable, and the additional hardware cost might be lower than the cost resulting from a policer that drops large amounts of traffic (*e.g.*, additional transit cost).

5.2 Solutions for Content Providers

5.2.1 Limiting the Server's Sending Rate

A sender can potentially mitigate the impact of a policer by rate-limiting its transmissions, to avoid pushing the policer into a state where it starts to drop packets. Optimally, the sender limits outgoing packets to the same rate enforced by the policer. We experimentally verified the benefits of sender-side rate limiting in a lab environment. We also confirmed the result in the wild, by temporarily configuring one of Google's CDN server to rate limit to a known carrier-enforced policing rate, then connecting to that server via the public Internet from one of our mobile devices that we know to be subject to that carrier's policing rate. In both experiments, *loss rates dropped from 8% or more to ∼0%*.

Additionally, if the policer uses a small bucket, rate limiting at the sender side can even improve goodput. We verified this by configuring a policer to a rate of 1.5 Mbps with a capacity of only 8 KB. In one trial we transmit traffic unthrottled, and in a second trial we limit outgoing packets to a rate of 1.425 Mbps (95% of the policing rate). Figures 13a and 13b show the sequence graphs for the first few seconds in both trials. The rate-limited flow clearly performs better in comparison, achieving a goodput of 1.38 Mbps compared to 452 kbps. The flow without rate limiting at the server side only gets a fraction of the goodput that the policer actually allows. The reason is that the small token bucket drops packets from larger bursts, resulting in low goodput.

Finally, we measured the benefits of rate limiting video through lab trials. We fetched videos from a YouTube Web server, with traffic passing through our lab policier. For some trials, we inserted a shaper between the server and the policer, to rate limit the transfer. Non-rate-limited playbacks observed an average rebuffering ratio of 1%, whereas the rate-limited flows did not see a single rebuffering event.

5.2.2 Avoiding Bursty Transmissions

Rate limiting in practice may be difficult, as the sender needs to estimate the throttling rate in near real-time at scale. We explored two viable alternatives to decrease loss by reducing the burstiness of transmissions, giving the policer an opportunity to generate tokens between packets.

Server	Loss (median)			Loss (95th pct.)		
	(base)	(paced)	(rec. fixed)	(base)	(paced)	(rec. fixed)
US	7.5%	6.7%	6.4%	34.8%	26.7%	32.2%
India	9.9%	7.8%	8.4%	52.1%	35.8%	34.6%

Table 7: Observed median and 95th percentile loss rates on policed connections served by two selected CDN servers.

We start by trying *TCP Pacing* [3]. Whereas a traditional TCP sender relies solely on ACK clocking to determine when to transmit new data, pacing spreads new packets across an RTT and avoids bursty traffic. Figure 13c shows the effect of pacing in the lab setup used in §5.2.1, but with a pacer in place of the shaper. Overall, the flow achieves a goodput of 1.23 Mbps which is worse than rate-limiting (1.38 Mbps) but a significant improvement over the unmodified flow (452 kbps). Packet loss is reduced from 5.2% to 1.3%.

In addition, we confirmed the benefits of pacing by turning it on/off on multiple CDN servers serving real clients. Enabling pacing consistently caused loss rates to drop by 10 – 20%. Table 7 shows the results for two of the CDN servers ("base" and "paced" columns).

Even when transmissions are not bursty, heavy losses can still occur when the sender consistently sends at a rate larger than the policing rate. In Linux, loss recovery can trigger periods of slow start [19], in which the server sends two packets for every ACKed packet. This results in sending at twice the policed rate during recovery and hence 50% of the retransmissions are dropped by the policer. To avoid this behavior, we modified the loss recovery to use packet conservation (for every ACKed packet, only one new packet is sent) initially and only use slow start if the retransmissions are delivered. Keeping slow start enables us to quickly recover from multiple losses within a window in a non-policed connection. Otherwise it will take N round trips to recover N packet losses.

As with pacing, we experimentally deployed this change which caused loss rates to drop by 10 to 20% as well ("base" and "rec. fixed" columns in Table 7). After testing, we also upstreamed the recovery patch to the Linux 4.2 kernel [14].

5.3 Summary of Recommendations

While extensive additional experimentation is necessary, we make the following initial suggestions to mitigate the adverse effects of policers:

1. ISPs can configure policers with smaller burst sizes. This prevents TCP's congestion window from growing too far beyond the policing rate when the token bucket fills up thereby resulting in fewer bursty losses.
2. ISPs can deploy shapers with small buffers instead of policers. Shaping avoids the heavy losses usually seen when employing policing, while using only small buffers prevents excessive queueing delays.
3. Content providers can rate-limit their traffic, especially when streaming large content. This can reduce the gap

between the sending rate and the policing rate resulting in fewer bursty losses.

4. Content providers can employ TCP pacing on their servers to reduce the burstiness of their traffic.

Our initial results show that these strategies can minimize or eliminate packet loss, and improve playback quality.

6. RELATED WORK

To our knowledge, no prior work has explored the prevalence and impact of policers at a global scale. Others explored policing for differentiated services [54], fair bandwidth allocation [36], or throughput guarantees [21,64]. One study explored the relationship between TCP performance and token bucket policers in a lab setting and proposed a TCP-friendly version achieving per-flow goodputs close to the policed rate regardless of the policer configuration [60]. Finally, a concurrently published study investigated the impact of traffic policing applied by T-Mobile to content delivery. This behavior was recently introduced as part of the carrier's "BingeOn" program, where traffic can be zero-rated (*i.e.*, results in no charges to customers) while it is at the same time policed to a rate of 1.5 Mbps [31].

Our work is inspired by and builds upon the large number of existing TCP trace analysis tools [10,12,45,49,51,56,61, 63]. On top of these tools, we are able to annotate higher-level properties of packets and flows that simplify analysis of packet captures at the scale of a large content provider.

A few threads of work are complementary to ours. One is the rather large body of work that has explored ways to understand and improve Web transfer performance (*e.g.*, latency, throughput) and, more generally, content delivery, especially at the tail [9, 11, 22, 25, 29, 30, 37, 41–43, 68]. None of these has considered the deleterious effects of policers.

Prior work has also explored the relationship between playback quality and user engagement [18]. Our work explores the relationship of network effects (pathological losses due to policers) and playback quality, and, using results from this prior work, we are able to establish that policing can adversely affect user satisfaction.

A line of research explores methods to detect service differentiation [4, 17, 33, 57, 67]. They all exploit differences in flow performance characteristics, like goodput or loss rate, to identify differentiated traffic classes. However, they do not attempt to understand the underlying mechanisms (policing or shaping) used to achieve traffic discrimination. Prior work has explored detecting traffic shaping using active methods [34]; in contrast, we detect burst-tolerant shaping purely passively.

Finally, some network operators were already aware of policing's disadvantages, presenting anecdotal evidence of bad performance [15, 59, 62]. We reinforce this message, quantifying the impact at scale for the first time.

7. CONCLUSION

Policing high-volume content such as video and cloud storage can be detrimental for content providers, ISPs and end-users alike. Using traces from Google, we found a non-trivial prevalence of traffic policing in almost every part of the globe: between 2% and 7% of lossy video traffic worldwide is subject to policing, often at throughput rates below what would be necessary for HD video delivery. Policers drop packets, and this results in policer-induced packet loss rates of 21% on average, 6× that of non-policed traffic. As a result of these loss rates, the playback quality of policed traffic is distributionally worse than that of non-policed traffic, a significant issue from a content provider perspective since it can affect user engagement in the content. We have identified benign traffic management alternatives that avoid adverse impacts of policing while still permitting ISPs to exercise their right to control their infrastructure: content providers can pace traffic, and ISPs can shape traffic using small buffers.

Acknowledgments

This study would not have been possible without the work and help of many software engineers that worked on the implementation and deployment of the packet capture system and the analysis pipeline, as well as helping us with deep dives into the data and reviewing our work, including: Sam Burnett, Harold Gonzales, Brendan Hickey, Van Jacobson, Devin Kennedy, Leonidas Kontothanassis, Brian Rogan, and Martijn Stevenson. Ramesh Govindan performed some of this work while employed temporarily at Google. We thank Arash Molavi Kakhki and Dave Choffnes, our shepherd Ion Stoica, and our anonymous reviewers for their valuable feedback. Finally, we thank the router vendor that kindly provided us with one of their devices for use in our lab experiments.

8. REFERENCES

[1] Policing Detection (Supplemental Material). https://usc-nsl.github.io/policing-detection/.

[2] B. Ager, N. Chatzis, A. Feldmann, N. Sarrar, S. Uhlig, and W. Willinger. Anatomy of a Large European IXP. In *Proc. of ACM Conf. of the Special Interest Group on Data Communication (SIGCOMM '12)*, 2012.

[3] A. Aggarwal, S. Savage, and T. E. Anderson. Understanding the Performance of TCP Pacing. In *Proc. of IEEE Int. Conf. on Computer Communications (INFOCOM '00)*, 2000.

[4] V. Bashko, N. Melnikov, A. Sehgal, and J. Schonwalder. BonaFide: A traffic shaping detection tool for mobile networks. In *Proc. of IFIP/IEEE Symp. on Integrated Network Management (IM '13)*, 2013.

[5] C. Bastian, T. Klieber, J. Livingood, J. Mills, and R. Woundy. *RFC 6057: Comcast's Protocol-Agnostic Congestion Management System*, 2010.

[6] S. Bauer, D. Clark, and W. Lehr. PowerBoost. In *Proc. of ACM Workshop on Home Networks (HomeNets '11)*, 2011.

[7] R. Braden. *RFC 1122: Requirements for Internet Hosts - Communication Layers*, 1989.

[8] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *Proc. of ACM Conf. of the Special Interest Group on Data Communication (SIGCOMM '94)*, 1994.

[9] B. Briscoe, A. Brunstrom, A. Petlund, D. Hayes, D. Ros, J. Tsang, S. Gjessing, G. Fairhurst, C. Griwodz, and M. Welzl. Reducing Internet Latency: a Survey of Techniques and Their Merits. *IEEE Communications Surveys & Tutorials*, 2014.

[10] K. Burns. *TCP/IP Analysis & Troubleshooting Toolkit*. John Wiley & Sons, 2003.

[11] M. Calder, X. Fan, Z. Hu, E. Katz-Bassett, J. Heidemann, and R. Govindan. Mapping the Expansion of Google's Serving Infrastructure. In *Proc. of ACM Internet Measurement Conference (IMC '13)*, 2013.

[12] CAPTCP. http://research.protocollabs.com/captcp/.

[13] R. L. Carter and M. Crovella. Measuring Bottleneck Link Speed in Packet-Switched Networks. *Performance Evaluation*, 27/28(4), 1996.

[14] Y. Cheng. tcp: reducing lost retransmits in recovery (Linux kernel patches). http://comments.gmane.org/gmane.linux.network/368957, 2015.

[15] Cisco. Comparing Traffic Policing and Traffic Shaping for Bandwidth Limiting. http://www.cisco.com/c/en/us/support/docs/quality-of-service-qos/qos-policing/19645-policevsshape.html#traffic.

[16] Cisco. The Zettabyte Era – Trends and Analysis. White Paper, 2014.

[17] M. Dischinger, M. Marcon, S. Guha, P. K. Gummadi, R. Mahajan, and S. Saroiu. Glasnost: Enabling End Users to Detect Traffic Differentiation. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI '10)*, 2010.

[18] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. A. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the Impact of Video Quality on User Engagement. In *Proc. of ACM Conf. of the Special Interest Gr. on Data Communication (SIGCOMM '11)*, 2011.

[19] N. Dukkipati, M. Mathis, Y. Cheng, and M. Ghobadi. Proportional Rate Reduction for TCP. In *Proc. of ACM Internet Measurement Conference (IMC '11)*, 2011.

[20] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin. An Argument for Increasing TCP's Initial Congestion Window. *ACM SIGCOMM Comp. Commun. Rev.*, 40, 2010.

[21] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin. Understanding and Improving TCP Performance Over Networks With Minimum Rate Guarantees. *IEEE/ACM Transactions on Networking*, 7(2), 1999.

[22] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan. Reducing Web Latency: the Virtue of Gentle Aggression. In *Proc. of ACM Conf. of the Special Interest Group on Data Communication (SIGCOMM '13)*, 2013.

[23] T. Flach, L. Pedrosa, E. Katz-Bassett, and R. Govindan. A Longitudinal Analysis of Traffic Policing Across the Web. *USC Computer Science Technical Report 15-961*, 2015.

[24] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4), 1993.

[25] A. Ganjam, F. Siddiqui, J. Zhan, X. Liu, I. Stoica, J. Jiang, V. Sekar, and H. Zhang. C3: Internet-Scale Control Plane for Video Quality Optimization. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI '15)*, 2015.

[26] J. Gettys and K. Nichols. Bufferbloat: Dark Buffers in the Internet. *Queue*, 9(11), 2011.

[27] M. Ghobadi, Y. Cheng, A. Jain, and M. Mathis. Trickle: Rate Limiting YouTube Video Streaming. In *Proc. of USENIX Annual Technical Conference (ATC '12)*. USENIX, 2012.

[28] N. Hu and P. Steenkiste. Evaluation and Characterization of Available Bandwidth Probing Techniques. *IEEE Journal on Selected Areas in Communications*, 21(6), 2003.

[29] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proc. of ACM Conf. of the Special Interest Group on Data Communication (SIGCOMM '14)*, 2014.

[30] J. Hui, K. Lau, A. Jain, A. Terzis, and J. Smith. YouTube performance is improved in T-Mobile network. http://velocityconf.com/velocity2014/public/schedule/detail/35350.

[31] A. M. Kakhki, F. Li, D. Choffnes, A. Mislove, and E. Katz-Bassett. BingeOn Under the Microscope: Understanding T-Mobile's Zero-Rating Implementation. In *Proc. of Internet-QoE Workshop*, 2016.

[32] A. M. Kakhki, A. Razaghpanah, H. Koo, A. Li, R. Golani, D. Choffnes, P. Gill, and A. Mislove. Identifying Traffic Differentiation in Mobile Networks. In *Proceedings of the 15th ACM/USENIX Internet Measurement Conference (IMC'15)*, 2015.

[33] P. Kanuparthy and C. Dovrolis. DiffProbe: Detecting ISP Service Discrimination. In *Proc. of IEEE International Conference on Computer Communications (INFOCOM '10)*, 2010.

[34] P. Kanuparthy and C. Dovrolis. ShaperProbe: End-to-end Detection of ISP Traffic Shaping Using Active Methods. In *Proc. of ACM Internet Measurement Conference (IMC '11)*, 2011.

[35] S. Keshav. A Control-theoretic Approach to Flow Control. In *Proc. of ACM Conf. of the Special Interest*

Group on Data Communication (SIGCOMM '91), 1991.

[36] J. Kidambi, D. Ghosal, and B. Mukherjee. Dynamic Token Bucket (DTB): A Fair Bandwidth Allocation Algorithm for High-Speed Networks. *Journal of High-Speed Networks*, 2001.

[37] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the Edge Network. In *Proc. of ACM Internet Measurement Conference (IMC '10)*, 2010.

[38] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian. Internet inter-domain traffic. *ACM SIGCOMM Computer Communication Review*, 41(4), Aug. 2010.

[39] G. Linden. Make Data Useful. http://sites.google.com/site/glinden/Home/StanfordDataMining.2006-11-28.ppt, 2006.

[40] L. Lovász. On the Ratio of Optimal Integral And Fractional Covers. *Discrete Mathematics*, 13(4):383–390, 1975.

[41] M. Luckie, A. Dhamdhere, D. Clark, B. Huffaker, and K. Claffy. Challenges in Inferring Internet Interdomain Congestion. In *Proc. of ACM Internet Measurement Conference (IMC '14)*, 2014.

[42] M-Lab. ISP Interconnection and its Impact on Consumer Internet Performance. http://www.measurementlab.net/static/observatory/M-Lab_Interconnection_Study_US.pdf.

[43] R. Mahajan, M. Zhang, L. Poole, and V. S. Pai. Uncovering Performance Differences Among Backbone ISPs with Netdiff. In *USENIX Symp. on Networked Systems Design & Implementation (NSDI '08)*, 2008.

[44] M. Mathis and J. Mahdavi. Forward Acknowledgement: Refining TCP Congestion Control. In *Proc. of ACM Conf. of the Special Interest Group on Data Communication (SIGCOMM '96)*, 1996.

[45] Microsoft Research TCP Analyzer. http://research.microsoft.com/en-us/projects/tcpanalyzer/.

[46] J. Nagle. *RFC 896: Congestion Control in IP/TCP Internetworks*, 1984.

[47] Netflix. Letter to Shareholders (Q4 2014). http://ir.netflix.com/results.cfm.

[48] K. Nichols and V. Jacobson. Controlling Queue Delay. *Queue*, 10(5), May 2012.

[49] J. Pahdye and S. Floyd. On Inferring TCP Behavior. In *Proc. of ACM Conf. of the Special Interest Group on Data Communication (SIGCOMM '01)*, 2001.

[50] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. PIE: A Lightweight Control Scheme to Address the Bufferbloat Problem. In *Proc. of IEEE Conf. on High Performance Switching and Routing (HPSR '13)*, 2013.

[51] V. Paxson. Automated Packet Trace Analysis of TCP Implementations. In *Proc. of ACM Conf. of the Special Interest Group on Data Communication (SIGCOMM '97)*, 1997.

[52] F. Qian, A. Gerber, Z. M. Mao, S. Sen, O. Spatscheck, and W. Willinger. TCP Revisited: A Fresh Look at TCP in the Wild. In *Proc. of Internet Measurement Conference (IMC '09)*, 2009.

[53] S. Rewaskar, J. Kaur, and F. D. Smith. A Passive State-Machine Approach for Accurate Analysis of TCP Out-of-Sequence Segments. *ACM SIGCOMM Computer Communication Review*, 36(3):51–64, 2006.

[54] S. Sahu, P. Nain, C. Diot, V. Firoiu, and D. F. Towsley. On Achievable Service Differentiation With Token Bucket Marking For TCP. In *Proc. of ACM SIGMETRICS Conf.*, 2000.

[55] Sandvine. Global Internet Phenomena Report 2H 2014. 2014.

[56] S. Savage. Sting: A TCP-based Network Measurement Tool. In *USENIX Symposium on Internet Technologies and Systems (USITS '99)*, 1999.

[57] M. B. Tariq, M. Motiwala, N. Feamster, and M. Ammar. Detecting Network Neutrality Violations with Causal Inference. In *Proc. of ACM Conf. on Emerging Networking Experiments and Technologies (CoNEXT '09)*, 2009.

[58] tcptrace. http://www.tcptrace.org.

[59] I. van Beijnum. *BGP: Building Reliable Networks with the Border Gateway Protocol*. O'Reilly Media, 2002.

[60] R. van Haalen and R. Malhotra. Improving TCP performance with bufferless token bucket policing: A TCP friendly policer. In *Proc. of IEEE Workshop on Local and Metropolitan Area Networks (LANMAN '07)*, 2007.

[61] Wireshark. http://www.wireshark.org.

[62] C. Wittbrodt. CAR Talk: Configuration Considerations for Cisco's Committed Access Rate. https://www.nanog.org/meetings/abstract?id=1290, 1998.

[63] yconalyzer. http://yconalyzer.sourceforge.net/.

[64] I. Yeom and A. L. N. Reddy. Realizing Throughput Guarantees in a Differentiated Services Network. In *Proc. of IEEE Int. Conf. on Multimedia Computing and Systems (ICMCS '99)*, 1999.

[65] YouTube JavaScript Player API Reference. https://developers.google.com/youtube/js_api_reference.

[66] YouTube Statistics. http://www.youtube.com/yt/press/statistics.html.

[67] Y. Zhang, Z. M. Mao, and M. Zhang. Detecting traffic differentiation in backbone ISPs with NetPolice. In A. Feldmann and L. Mathy, editors, *Proc. of ACM Internet Measurement Conference (IMC '09)*, 2009.

[68] J. Zhou, Q. Wu, Z. Li, S. Uhlig, P. Steenkiste, J. Chen, and G. Xie. Demystifying and Mitigating TCP Stalls at the Server Side. In *Proc. of ACM Conf. on Emerging Networking Experiments and Technologies (CoNEXT '15)*, 2015.