Identifying Traffic Differentiation on Cellular Data Networks

Arash Molavi Kakhki‡, Abbas Razaghpanah⋆, Rajesh Golani⋆
David Choffnes‡ , Phillipa Gill⋆, Alan Mislove‡
‡Northeastern University, ⋆Stony Brook University

1. INTRODUCTION

The goal of this research is to detect traffic differentiation in cellular data networks. We define traffic differentiation as any attempt to change the performance of network traffic traversing an ISP's boundaries. ISPs may implement differentiation policies for a number of reasons, including load balancing, bandwidth management, or business reasons. Specifically, we focus on detecting whether certain types of network traffic receive better (or worse) performance. As an example, a wireless provider might limit the performance of third-party VoIP or video calling services (or any other competing services) by introducing delays or reducing transfer rates to encourage users to use services provided by the provider. Likewise, a provider may allocate more bandwidth to preferred applications.

Previous work [1, 3, 5] explored this problem in limited environments. Glasnost focused on BitTorrent in the desktop/laptop environment, and used port/payload randomization to avoid differentiation. NetDiff covered a wide range of passively gathered traffic from a large ISP but likewise did not support targeted, controlled experiments. We address these limitations with *Mobile Replay*.

2. MOBILE REPLAY

We assume that ISPs will differentiate traffic based on properties such as hostname, IP addresses, ports, total number of connections, payload signatures, total bandwidth and time of day. Our system currently can detect all of these forms of differentiation with the exception of server-based differentiation.

2.1 Overview

Mobile Replay identifies service differentiation using two key components. First, it tests for differentiation by replaying real network traces generated from user interactions with apps. Meddle [4] facilitates capturing this information, and we develop new strategies for replaying arbitrary app traces. Second, Mobile Replay exploits the Meddle VPN to conduct controlled experiments. By alternately replaying traffic over tunneled and untunneled connections multiple times in rapid succession, we control ISP visibility into packet contents that may used to differentiate traffic.

Figure 1: Schematic view of how our project works. Traffic is sent unencrypted and through the Meddle VPN tunnel between the mobile client and a control server.

A key challenge is how to capture and replay the salient features of application traffic such that it will be subject to differentiation from middleboxes. To this end, we design a system that captures traffic generated by users' devices (via Meddle) and replays those flows from a replay server. Another challenge is how to establish ground truth as to whether the ISP is differentiating service for replay traffic. To address this, we exploit the VPN connection that Meddle provides as follows. When the VPN is enabled, the ISP cannot inspect flow contents and thus cannot differentiate based on the above factors except total bandwidth and time-of-day. We then compare this performance to the case when we send traffic untunneled. Using multiple successive trials of tunneled and untunneled replay experiments, we can determine the noise inherent in performance metrics in each type of experiment (tunneled vs not tunneled), then identify cases where there are statistically significant differences between them, indicating differentiation.

2.2 Replay and methodology

We support both UDP and TCP flows in our replay system, which consists of a client app running on the mobile device and a replay server written in Python in a event-driven fashion using Gevent [2], which performs efficiently with many concurrent clients. The server also performs UDP hole punching to support clients behind a NAT. The client and server coordinate to replay the original flows to reproduces packet timings, sequence of bytes, ports and client IPs. Since our replay is limited to using our own replay servers, we cannot detect differentiation based on arbitrary server IPs (a topic of ongoing work). Note, however, that if our replay servers are co-located with servers contacted in our traces, we do not suffer from this limitation. For example, if

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s). Copyright is held by the author/owner(s).
SIGCOMM'14, August 17–22, 2014, Chicago, IL, USA.
ACM 978-1-4503-2836-4/14/08.
http://dx.doi.org/10.1145/2619239.2631445 .

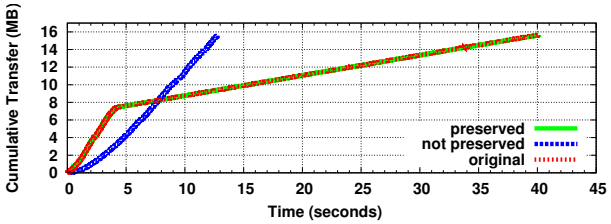

Figure 2: Cumulative byte-transfer plot of YouTube trace replays with inter-packet timing preserved vs. not preserved. The x-axis is time and y-axis is the total number of bytes transferred to that point. Observe that the original packet trace looks identical to the replay when timing is preserved.

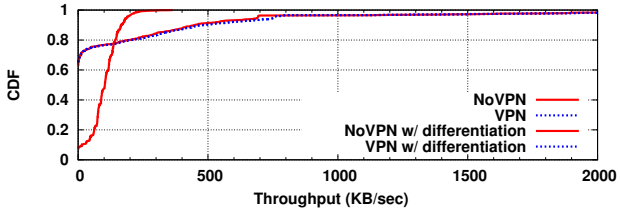

Figure 3: CDF of throughput over time, with and without extra delay and packet loss induced at the server. Our approach allows us to detect differences in throughput sample distributions when differentiation is applied.

we use EC2 servers and the tested app streams video from a server in the same data center (and/or same block of IP addresses), we may be able to trigger the same differentiation in our replays.

We detect differentiation according to the following metrics. First, we verify that the bytes sent/received at each endpoint during the replay are the same as the original trace. If not, we flag a case of content manipulation/blocking. Second, we compute summary statistics on throughput, loss, and RTT. Unlike manipulation/blocking, there are confounding factors other than differentiation that may causes changes in these statistics between the record and replay. To address this issue, we run multiple replay trials (10 total, though we are investigating dynamically adjusting the number of trials in response to observed noise), alternating between using a VPN connection (R_T) and an untunneled one (R_U). By computing statistics over multiple trials of one category (R_T or R_U) we can quantify natural variations in performance that are not a result of differentiation. Having computed the variance over R_T and R_U, we can compare the summary statistics (mean/median) of R_T and R_U and use the variance in each category to determine if the differences are statistically significant. Note that ISPs may apply differentiation to all VPN traffic, e.g., by throttling. To detect this, we group all R_T samples and compare them to all R_U samples across all applications and use the analysis described above.

2.3 Feasibility

We now demonstrate that our replay module performs as expected and can detect differentiation.

By preserving packet ordering and timing, our system produces very similar results to the original traffic. Figure 2 shows a YouTube trace replayed in both conditions side-by-side. Of course, a variety of factors can differ between record and replay, including network conditions and access technology. In particular, apps may change their behavior in response to network technology and available bandwidth. In such cases, we must ensure that we replay traffic that was originally captured over similar network conditions. In our experiments, YouTube was the only app that exhibited such behavior (due to adaptive bitrate streaming).

We next tested whether we can detect differentiation in a controlled environment. We ran a full replay test with YouTube in a test environment where we inject a simple form of differentiation by adding a 3% packet loss and 10ms of delay. Figure 3 shows CDFs of throughput during the replays, with and without differentiation. The effect of differentiation on the distribution is clear, indicating that our system and metrics allow us to distinguish between differentiation and noise. Last, we tested our approach using several application traces on Verizon in Boston. Figure 4 shows our replay results for two of those services, and we are able to confirm that there is no differentiation.

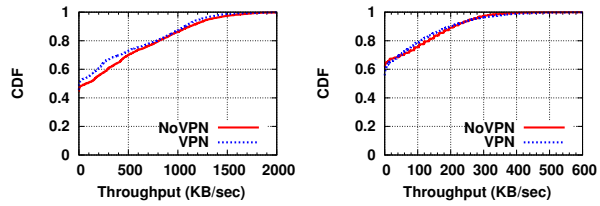

Figure 4: CDF of throughput over time for Verizon (left: Netflix, right: Spotify). Our tests show that Verizon does not differentiate traffic for these apps.

3. FUTURE WORK

To address the limitation that we cannot detect differentiation based on server ports, our future work includes leveraging source-spoofing such that the source IP in the server's packets are the same as in the original traces. Additionally, because the VPN itself may be subject to differentiation (or blocked), we are investigating using multiple differentiation detection approaches, including randomizing packet payloads and ports in the replay. We are also developing an app that allows average users to create and conduct tests from an mobile provider worldwide, and that efficiently uses scarce available data quota to run those tests. We plan to use the results to produce a Web site informing consumers of ISP policies.

4. REFERENCES

[1] M. Dischinger, M. Marcon, S. Guha, K. P. Gummadi, R. Mahajan, , and S. Saroiu. Glasnost: Enabling end users to detect traffic differentiation. In *NSDI*, 2010.

[2] `http://www.gevent.org`.

[3] M. B. Tariq, M. Motiwala, N. Feamster, and M. Ammar. Detecting network neutrality violations with causal inference. In *CoNEXT*, 2009.

[4] `http://www.meddle.mobi/papers/meddle-main.pdf`.

[5] Y. Zhang, Z. M. Mao, and M. Zhang. Detecting traffic differentiation in backbone isps with netpolice. In *SIGCOMM*, 2009.