Network Neutrality Inference using Network Tomography

THÈSE N^{O} 8076 (2018)

PRÉSENTÉE LE 21 MARS 2018

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

LABORATOIRE D'ARCHITECTURE DES RÉSEAUX

PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Ovidiu Sebastian MARA

acceptée sur proposition du jury:

Prof. M. Grossglauser, président du jury
Prof. A. Argyraki, directrice de thèse
Prof. C. Dovrolis, rapporteur
Prof. K. Gummadi, rapporteur
Prof. P. Thiran, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2018

Everything that happens happens as it should, and if you observe carefully, you will find it to be so. (Marcus Aurelius)

This document has been written in Madoko.

Abstract

Our work studies network neutrality, a property of communication networks which means that they treat all traffic the same, regardless of application, content provider or communication protocol. This is an important problem, because sometimes users suspect their ISPs of violating network neutrality, but apply inaccurate methods to check their suspicions, reaching incorrect conclusions.

Prior non-neutrality detection methods either provide only detection, but lack localization capabilities; or also perform localization, but assume perfect measurements. For the latter, we show that, in practice, the measurement process can severely impact the results.

We present a method that performs both non-neutrality detection and localization, using only end-to-end measurements, and assuming an imperfect measurement process. We identify the sources of measurement error that may affect our method, we address them, and evaluate the method extensively with simulations, emulations and experiments on the Internet. We also use our method in two studies, investigating suspicions that a set of ISPs prioritize speed-test traffic, or differentiate against BitTorrent traffic; despite circumstancial evidence that they do, we obtain reliable evidence to the contrary. Finally, we present the network emulator that we built to evaluate our method, hoping that it will be a useful tool in future research.

We conclude that it is feasible to detect and localize network neutrality violations based solely on end-to-end measurements, without assuming a perfect measurement process; and that it is important that reasoning about network neutrality is based on reliable evidence of network behavior.

Keywords

network neutrality, network tomography, congestion

Résumé

Notre recherche étudie la neutralité du réseau, une propriété des réseaux de communication ce qui signifie qu'ils traitent tout le trafic de la même manière, indépendamment de l'application, du fournisseur de contenu ou du protocole de communication. C'est un problème important, car parfois les utilisateurs soupçonnent leurs FAI de violer la neutralité du réseau, mais appliquent des méthodes inexactes pour vérifier leurs soupçons, qui peut conduire à des conclusions erronées.

Les méthodes de détection antérieures de la non-neutralité soit fournissent seulement la détection, mais manquent de fonctionnalités de localisation; soit effectuent également la localisation, mais suppose que les mesures sont parfaites. Pour ce dernier, nous montrons que, dans la pratique, le processus de mesure peut avoir une incidence sévère sur la fidélité des résultats.

Nous présentons une méthode qui effectue la détection et la localisation de la non-neutralité, analysant des mesures prises aux points d'extrémité d'une réseau informatique, en supposant un processus de mesure imparfait. Nous identifions les sources d'erreur de mesure qui peuvent affecter notre méthode, nous les adressons, et nous évaluons la méthode avec des simulations, des émulations et des expériences sur l'Internet. Nous utilisons notre méthode dans deux études, pour examiner les soupçons selon lesquels un ensemble de FAI privilégient le trafic de test de vitesse, ou ralentissent le trafic BitTorrent; malgré les preuves circonstancielles qui suggèrent qu'ils le font, nous obtenons des preuves fiables au contraire. Enfin, nous présentons l'émulateur de réseau que nous avons construit pour évaluer notre méthode, en espérant qu'il sera un instrument outil dans des recherches futures.

Nous concluons que c'est possible de détecter et localiser les violations de la neutralité du réseau en se basant exclusivement sur des mesures prises aux points d'extrémité du réseau, sans assumer un processus de mesure parfait; et que c'est important que le raisonnement sur la neutralité du réseau est basé sur des preuves fiables du comportement du réseau.

Mots-clés

neutralité du réseaux, tomographie du réseaux, congestion

Acknowledgements

I would like to express my gratitude to my advisor, Katerina Argyraki, for guiding me on this path, showing me how to express my ideas with elegance, and sharing with me your enthusiasm of performing research.

I also offer my thanks to the members of my defense committee: Constantine Dovrolis, Krishna Gummadi, Patrick Thiran and Matthias Grossglauser. I appreciate the effort you are putting in to evaluate my work.

To my colleagues, thanks for all the interesting discussions we had together.

To my family, thanks for your patience and support.

Contents

1. Introduction

1.1. Network neutrality

Network neutrality has been debated widely throughout the world over the last decade. But what is exactly network neutrality? The most commonly given definition of a neutral network is that it is a network that treats all traffic the same; for example, it does not favor an application, a content provider or a subset of users over others. This definition is not very precise, since there are several exceptions to this rule: for example, it is perfectly reasonable for a network to throttle or block denial-of-service traffic, malicious traffic, or traffic from unintentionally misbehaving systems that may have a negative impact over the communication of others—none of these actions make a network less neutral. Perhaps a more accurate definition, that remains intuitive without delving into legal terms, is the following:

> A neutral network is a network where any two endpoints (i.e. users and content providers) that decide to exchange traffic with each other are able to do so without the network blocking, impairing or improving the performance experienced by their communication based on any criteria other than data rates and data volume limits agreed upon between the network and the respective endpoints.

We think that this definition summarizes well recent net neutrality guidelines provided by European legislators [Council of the European Union, 2015], and that it also covers the major scenarios that caused widespread debate over the last decade: First there were discussions about Internet service providers (ISPs) throttling BitTorrent traffic crossing their network boundaries [M Dischinger et al., 2008]. Then attention switched to ISPs throttling [Vukas, 2014] or not providing sufficient bandwidth for [The Verge, 2014] transit traffic from popular content providers. More recently, there has been recurring user suspicion and circumstantial evidence that some ISPs prioritize speed-test traffic to create an exaggerated impression of the bandwidth they provide to their users [Byrne, 2016].

Currently, end-users apply improvised, inaccurate methods to determine whether their ISPs are differentiating against some of their traffic. A simple web search for “is my ISP throttling BitTorrent?” or “is my ISP prioritizing speed-test traffic?” finds a large number of reports from users suspecting their ISP of non-neutrality. But at closer inspection, we found that most of them reach this conclusion using methods that are too rudimentary to eliminate common types of biases that cause performance differences in neutral networks. For example, a BitTorrent download may be slower than a web download simply because the number of peers in the swarm is small, or because they are found in distant locations. Similarly, speed-tests may observe faster throughput than regular web downloads only because the testing server is located in the same ISP as the user, whereas the other data is transferred from a more distant network. These are just two examples, but in practice there is a variety of reasons why simple throughput measurements may yield conflicting results [Sundaresan et al., 2011].

We need a method that accurately assesses the neutrality of a given link or link sequence, going beyond circumstantial evidence. We are not arguing for rigid neutrality rules, just for transparency: if an ISP differentiates against specific end-hosts, protocols, or applications, that should be visible to the affected parties: End-users should be able to prove it with a well-defined level of confidence.

There are already several methods that can be used to detect non-neutrality. Some of them, like DiffProbe [Kanuparthy and Dovrolis, 2010] and Glasnost [Marcel Dischinger et al., 2010], rely on throughput differences: they consider one source-destination pair at a time, make it exchange traffic of type x (e.g., HTTP traffic) then of type y (e.g., BitTorrent traffic), and compare the achieved throughput; a significant difference indicates non-neutrality. Others, such as [Kanuparthy and Dovrolis, 2011] and [Flach et al., 2016], analyze packet captures from each individual flow to determine if its performance was limited by a traffic shaper or traffic policer.

We present a different kind of method, which takes as input end-to-end measurements from a small set of paths and determines whether a given link sequence, traversed by these paths, is neutral, i.e. treated all measurement traffic the same. Our method differs from existing ones in two ways:

1. It can determine the neutrality of relatively short link sequences, not only end-to-end paths; hence, we can not only *detect* non-neutrality, but also *localize* it to specific parts of the network.
2. It is not affected by differentiation criteria: if the links treat some part of the measurement traffic worse than the rest, our method will detect it, even if the differentiation is based on network-layer criteria, such as throttling *all* traffic between specific source-destination pairs.

Neither of these goals can be achieved by methods that consider a single source-destination pair at a time.

1.2. Network tomography

Our work builds upon network performance tomography: a set of techniques that infer performance properties of links (such as loss rate, latency, congestion status, or congestion probability) based on external observations, i.e. without monitoring these links directly, only taking measurements at the edge of the network. A tomographic technique typically forms a system of equations

$$\vec{y} = \mathbf{A} \cdot \vec{x}, \tag{1}$$

where $\vec{y}$ is a given vector of external observations (end-to-end path measurements), $\mathbf{A}$ is a given matrix that specifies the relationships between links and paths, and $\vec{x}$ is the vector of link properties that we are trying to infer. It then estimates $\vec{x}$, either by solving this system of equations when it has a unique solution [Caceres et al., 1999; Coates and Nowak, 2000; Bu et al., 2002; Nguyen and Thiran, 2007a, 2007b; Ghita et al., 2010] or by choosing a solution that has some desirable property: for example, assumes the smallest number of problematic links [Padmanabhan et al., 2003; Duffield, 2006; Song et al., 2006; Dhamdhere et al., 2007] or occurs with the highest probability [Nguyen and Thiran, 2007b].

Network performance tomography fundamentally relies on the assumption that the network is neutral (each link treats traffic from all paths equally), otherwise it would be impossible to express path measurements as a function of link properties and form a solvable system of equations.

Recent work [Z. Zhang, Mara, and Argyraki, 2014] turned this assumption on its head in order to detect and localize non-neutral links. It relies on the following observation: if we perform perfect end-to-end measurements from different vantage points, and find them to be inconsistent with each other, this necessarily means that the network covered by the measurements is non-neutral. Hence, unlike network performance tomography, which typically tries to form solvable systems of equations that connect link and path properties, this work tries to form *unsolvable* systems of equations that connect link and path properties, because such systems constitute evidence of non-neutrality.

The main problem with applying this work in practice is that it assumes perfect measurements. There are three aspects that are problematic:

Firstly, the method assumes that different paths that share a congested link will experience identical congestion states, the latter being computed from the measurements by comparing a performance metric (loss rate) with a fixed threshold. This is problematic because different paths may observe slightly different values of the performance metric. When the threshold is close to these values (for example, equal to their average), congestion states of different paths will not be identical, but poorly correlated.

Secondly, the method does not take into account uncertainty in estimating the frequencies with which paths are congested. The uncertainty is negligible only when the number of congestion state samples is very large, but this would require experiment durations of tens to hundreds of hours. These are impractical, because on such time scales, network conditions or even the topology itself are likely to change, rendering the measurements unusable. Ignoring the uncertainty and applying the algorithm anyways leads to unreliable results, which we demonstrate through simulations.

Thirdly, the method uses hand-picked thresholds to reason about congestion and neutrality. In the presence of imperfect measurements, such an approach is not reliable: without estimating the uncertainties in the measurements and analyzing how they affect the inference, it cannot be guaranteed that the thresholds are high enough to avoid false positives; or small enough to allow detection of non-neutrality.

1.3. Contributions

This dissertation contributes a tomography-based method that detects and localizes non-neutrality, using only end-to-end measurements, and assuming an imperfect measurement process. In particular:

- We identify three aspects of Zhang et al.'s theory that do not hold when faced with an imperfect measurement process, and we propose three techniques that, in combination, enable reliable neutrality inference:
    - Identifying and filtering out inaccurate path congestion measurements:

    This is an important problem, due to the way Zhang et al.'s method correlates congestion states of different paths. We propose a method that estimates the accuracy of the measured states in the absence of any ground-truth knowledge, and eliminates those considered unreliable;
  - Modeling and computing the uncertainty in the inferred link performance: We show that this uncertainty is significant and unavoidable in practical scenarios, due to the limited duration over which paths and network conditions are stable in typical networks; the latter imposes a constraint over the number of samples we can collect, and thus over the accuracy of the inference. We design a method to estimate this uncertainty from the measurements;
  - Redesigning non-neutrality inference as a method that takes into account uncertainties caused by imperfect measurements and no longer uses any arbitrary hand-picked thresholds: instead, it is configured by a desired significance level of the non-neutrality verdict.

  We evaluate the redesigned method empirically with simulations, emulations and Internet experiments in controlled setups.

- We present two studies where we apply our method to investigate neutrality violations against two types of traffic: BitTorrent traffic (which is known to have been throttled in the past) and Internet "speed test" traffic (which is suspected of being prioritized).

  - We find that no such violations occur in the networks we investigate.
  - We show that our method is more reliable than a naïve, throughput-based alternative, which would have yielded false positives in our context.
  - We conduct what-if analyses and show that our method would have correctly detected and localized throttling or prioritization, had they occurred in the investigated networks.

- We present a network emulator that we built to evaluate our method. We show that, for networks consisting of up to a few hundred links, our emulator offers accurate packet processing at a granularity of 10 microseconds on average, with a worst case of about 100 microseconds. This makes it a useful tool for evaluating not only tomography algorithms, but also methods for detecting and localizing many different network policies, such as queuing, traffic shaping or traffic policing.

We conclude that it is feasible to detect and localize network neutrality violations based solely on end-to-end measurements, without assuming a perfect measurement process.

1.4. Outline

The contents of the thesis are organized as follows: Section 2 presents a quick review of prior work in the field of network performance tomography and neutrality inference. Section 3 presents our setup, including definitions, notations and assumptions;

it also describes our problem statement, and the algorithm from the prior work our method is based upon. Section 4 identifies and addresses the issues that occur when applying the algorithm in practice; it then presents the redesigned method and its evaluation through simulations, emulations and Internet experiments in controlled setups. Section 5 presents two studies where we apply our method to investigate neutrality violations on the Internet. Section 6 describes our network emulation setup, and the results of a performance and accuracy evaluation. Finally, the conclusions are presented in Section 7, followed by the Appendix.

2. Related work

2.1. Network Performance Tomography

2.1.1. Overview

Network performance tomography is a set of techniques that infer performance properties of links in the network based on external observations, i.e. without monitoring these links directly, only taking measurements at the edge of the network. Typically, these techniques operate in three stages:

1. Topology discovery: the topology of the network connecting the measurement endpoints is obtained, often by using tools like *traceroute*, although other sources, such as BGP route dumps, may be used;
2. Measurement collection: the performance of the traffic exchanged between endpoints is recorded. Traffic may consist either of regular traffic exchanged naturally between endpoints, in which case we call the measurement process to be *passive*; or probing traffic, such as ICMP pings[1], UDP pings, constant-rate or Poisson UDP traffic etc., in which case we call the measurement process to be *active*;
3. Performance inference: the topology is used to determine the relationships between the performance metrics of links and paths, creating a system of equations where the link performance metrics are unknown, and the path performance metrics are given (measured). This system is then solved to determine the link performance metrics.

What varies among these techniques is:

- The type of traffic that is measured—there are a lot of different methods, particularly in the case of active probing measurements;
- The performance metric that is used, such as loss rate, latency, congestion status, or congestion probability;
- The assumptions made about links and traffic, for example assuming temporal and spatial independence of link performance is very common;
- How the path overlay is chosen; some techniques work only on trees, while others can handle general graphs;
- How the system of equations is formed; the most common choices are systems of linear equations, systems of boolean (binary) equations and ad-hoc, graph-based models;
- The method used to solve the system of equations, in particular how a solution is chosen when it is not unique.

[1]Only the end-to-end outcome of ping probing is used, for example whether the probe or its reply has been received or not, or with what delay; network tomography techniques do not typically require nor make use of cooperation from network devices on the forwarding path; for example, they do not probe each hop from the path, nor use traceroute results as performance measurements.

2.1.2. Literature Selection

We now provide a selection of works published in the field of network performance tomography that are related to our method. This selection is by no means complete nor exhaustive. We focus on prior techniques, findings and insights that offered key elements our work could be based upon.

The first well-known network performance tomography technique was the work of [Caceres et al., 1999]: it inferred link loss rate from end-to-end multicast probing measurements taken from a tree topology. While not practical on the Internet due to the use of multicast traffic, several of its key elements have been used extensively in following tomography work, such as: packet loss events modeled as Bernoulli processes; the assumption of temporal and spatial independence of link performance; and choosing a single "best" solution from multiple possible solutions, in this case via Maximum Likelihood Estimation.

The work of [Coates and Nowak, 2000] provided a similar, but more practical technique that used unicast probing. It introduced the assumption that packet loss events are correlated on short time scales. The probing method was designed to take advantage of this assumption: a combination of single and two-packet back-to-back probes sent through the network. The network topology was still limited to a tree.

[Bu et al., 2002] took tomography in a different direction: they still used multicast probing, but created an algorithm that could work on general graphs, consisting of multiple, overlapping trees. They were among the first to show that tomography could be adapted to other performance metrics, in particular queuing delay.

Further work to scale tomography to large graphs was done, among others, by [Song et al., 2006]. They studied measurement path selection, and showed that results having good accuracy can be obtained by taking measurements from as little as 2% of the paths in the network.

[Tsang et al., 2000] applied network tomography to passive (unicast) measurements of TCP traffic flows. While the topology was very simple—a tree, consisting of a server at the root, sending traffic to clients located at the leaves—and the inference still used packet pairs (sampled from the traffic), the work provided useful results and insights related to packet loss correlation on short time scales, and how to avoid it by sampling packets with a minimum time spacing. We applied a similar technique in our method.

Another work that used passive measurements was [Padmanabhan et al., 2003]. For us, the most important aspect of the work were the extensive, real measurements they collected from a busy Microsoft server, which allowed them to provide the following insight: on most paths, loss rate tends to be stable over a period of several minutes. This is useful, since we will show that we divide our experiments in time intervals of 1 minute each, and we assume that loss rate is stationary in each interval. This prior result supports our assumption. As a side note, this is also the earliest article we found to describe performance metrics assigned to sequences of links—linear sections of a network path that contain no branches—, which we also use in our method.

The work of [Duffield, 2003, 2006] introduced the notion of binary network tomography: instead of inferring link loss rates, he changed the target to simply infer which links were affected by congestion or failures. We found interesting the detailed discussion of the assumptions made about the network: (i) that links are fair for the flows that traverse them; (ii) that failure events are rare; and (iii) that failures are independent and equally likely to occur on any link in the network. The first assumption hints at network neutrality; the paper follows by stating:

> Even if we do not perform correlated end-to-end measurements at the individual packet level, it is still reasonable to expect that two distinct packet streams that pass through a given link over the same period of time would exhibit some correlation in performance at a statistical level.

This assumption relaxes the previous stringent measurement requirements, by eliminating the need of packet-level correlation (such as the one obtained through back-to-back probing) in binary tomography.

The paper also introduced the notion of separability—the connection between congestion on the links and on the paths in the network— and made the distinction between strong separability (a path is congested if and only if at least one of its constituent links is congested) and weak separability (a path being congested implies at least one of its constituent links is congested). An insight we found useful in our work was that they showed that the congestion threshold—used to detect congestion by comparing loss rate against it—can be always chosen such that weak separability holds.

NetDiagnoser, presented in [Dhamdhere et al., 2007] relaxed two assumptions from Duffield's binary tomography method: (i) they provided an algorithm that works on arbitrary graphs, not just trees; and (ii) they assume that inter-AS links might not treat traffic fairly (which may happen due to BGP misconfigurations or by intentional filtering at the domain edges). They proposed a method that, given sufficient path coverage, is able to infer link states for each class of traffic. Their idea, to split each potentially non-neutral inter-AS link into multiple parallel links, one for each class of traffic, is something we also use in our method.

The work of [Nguyen and Thiran, 2007b] improved binary tomography in a different way: They showed that extra equations can be formed by combining data from multiple paths, which allows the system of equations to become fully determined, thus a unique solution can always be found. Our previous work from [Z. Zhang et al., 2014], and also our current work, are based upon this specific technique.

Also notable is the work of [Ghita et al., 2010], which improved binary tomography further by relaxing the spatial link independence assumption: they showed that under certain conditions, binary tomography can be used effectively even when some of the links are correlated with each other.

2.2. Neutrality Inference

The prior research related to neutrality inference falls into the following main categories:

Comparing the performance of different types of traffic over the same path: most notable are the following projects: [Marcel Dischinger et al., 2010] starts transfers between a user and a testing server, and compares the throughput achieved by different types of traffic; this project has been used extensively to detect differentiation against, and blocking of BitTorrent traffic; [Bashko et al., 2013] applies a similar method, but targets specifically mobile network users; [Lu et al., 2007] analyzes end-to-end measurements by performing rank classification of loss rates for different types of traffic; [Y. Zhang et al., 2007] attempts to identify significant loss rate differences between different types of traffic, with path segments traversing certain ISPs; [Kanuparthy and Dovrolis, 2010] compares the delays and packet losses experienced by a regular flow and a probing flow; and [Molavi Kakhki et al., 2015] detects if a path is differentiating against a type of traffic by recording and replaying the traffic in encrypted form and with different port numbers, then comparing the observed performance. The common theme of these methods is to have two end-hosts exchange different traffic flows over the same network path; if the performance results are significantly different, then the network path must be non-neutral. Traffic classes are defined by transport-layer headers or payload. Unlike our algorithms, these methods were not designed to detect traffic differentiation that affects all flows of an end-host, nor localize it to specific links. Still, the Glasnost project made steps towards localization, since its wide-scale deployment and large userbase allowed for compiling non-neutrality statistics per groups of paths, for example per ISP and per country [Marcel Dischinger and Gummadi, 2011]; similar efforts were undertaken by [Y. Zhang et al., 2007].

Detecting specific methods of throttling by analyzing packet timings: ShaperProbe [Kanuparthy and Dovrolis, 2011], Packsen [Weinsberg et al., 2011] and [Flach et al., 2016] detect whether a network path is shaping or policing a user's traffic and also determine the parameters of the shaper/policer. These methods are complementary to our work: while they detect whether any single flow is subjected to rate limiting and identifies the parameters, we detect whether different traffic flows are subjected to different treatment (of any kind) and localize this differentiation to specific links.

Traceroute-like probing: NetPolice [Y. Zhang et al., 2007] uses traceroute-like probes to measure the loss rate inflicted by an ISP on traffic associated with different neighboring ASes. By targeting specific path segments, it can perform both detection and localization of differentiation. This is complementary to our work, since we focus on the scenario where we cannot rely on traceroute-like probes to directly measure the loss rates of links, for example when such probing is not possible, or probes are not treated the same as other traffic.

Statistical comparison of wide-scale measurements: Nano [Tariq et al., 2008] collects a large set of measurements and then applies statistical methods to detect whether different ISPs inflict different performance on the same kind of traffic. Such methods are also complementary to our work: we detect whether any particular link (or link sequence) inflicts different performance on different traffic.

3. Setup

We state our network model in Section 3.1, we describe the link and path performance metrics in Section 3.2 and we provide a formal definition of network neutrality in Section 3.3. We present our formal problem statement in Section 3.4, our notation in Section 3.5, and the assumptions on which the models and algorithms are based upon in Section 3.6. Finally, we describe the strawman algorithm from [Z. Zhang et al., 2014], on which our work is based upon, in Section 3.7.

3.1. Topology

We model the network as a graph where each vertex corresponds to a node and each edge corresponds to a link connecting two nodes. Edges are directed, since we are interested in link performance, which may be different in opposite directions.

Nodes are divided into *end-hosts* and *relays*. The former source and/or sink traffic, but never relay traffic between other nodes. The latter do, for example they may be routers or switches.

A *path* is a loop-free sequence of consecutive links starting and ending at end-hosts. While in reality relays may also send or receive their own traffic, in our work we rely on only end-to-end measurements, thus we are not interested in paths that start or end at a relay.

The basic notation we use are l to refer to a link, $L = \langle l_j, ... l_k \rangle$ to refer to a loop-free sequence of links, p to refer to a path, and $P = \{p_j, p_k\}$ to refer to a path pair.

We organize our measurement traffic into *traffic classes*, i.e. criteria the network might use to introduce traffic differentiation. For example, to model a network that deprioritizes BitTorrent traffic relative to other traffic, we could define traffic class 1 as non-BitTorrent traffic and class 2 as BitTorrent traffic[2]. For simplicity, we assume that each path carries traffic from a single class, but that is by no means a requirement of our method.

For example, Figure 1 shows a network with five links and four paths. Traffic class 1 (in blue) consists of all traffic carried by path pair $\{p_1, p_2\}$, while traffic class 2 (in red) consists of all traffic carried by path pair $\{p_3, p_4\}$.

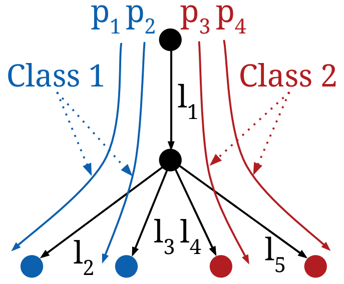


Figure 1. Example network.

[2]For consistency, we usually denote with a lower number the higher priority traffic.

3.2. Performance metrics

In boolean network tomography, we use end-to-end measurements to classify the observed performance of a path during a given time interval as either *congested* or *good*; where *congested* means that the path experiences poor performance according to some specific criterion. In order to define these two possible states precisely, we choose a *performance metric* that can be measured from end-to-end data, such as loss rate, throughput or latency, and we compare it with a *congestion threshold.* For example, to determine whether a path carrying video traffic has good performance, we can use throughput as a performance metric and set the congestion threshold to the minimum throughput value required to support some specific video quality, say 5 Mbps. If we observe only 4 Mbps in a measurement, the path is congested; if we observe 7 Mbps, the path is good.

In addition to the performance metric, we are also interested in the *performance number*, which is the probability that a path has good performance during an arbitrary time interval. While the performance metric shows the performance of the path during one measurement interval, the performance number shows how the path performs in general. The latter is important because performance metrics may vary from measurement to measurement, thus a single measured value is not sufficient to draw a reliable conclusion regarding the general performance of a path. Even well-provisioned paths may suffer occasional congestion, thus some samples of the performance metric might be poor, yet the performance number is close to 1, because most samples are good. By contrast, a poorly provisioned path suffers congestion often, so its performance number is lower. Performance numbers allow us to discriminate between paths that suffer different amounts of congestion, and are used extensively in our work.

3.2.1. Loss rate

Paths are characterized by *loss rates* and *performance numbers*, defined as follows:

During a given time interval, the *loss rate of path p*, denoted by Λ_p, is the probability with which a packet from p is dropped by any of the links that compose p. Λ_p is a continuous random variable that can take values from 0 to 1.

During a given time interval, *the performance number of path p*, denoted by y_p, is the probability that p's loss rate does not exceed a *congestion threshold t_c*:

$$y_p \equiv Pr[\ \Lambda_p \leq t_c\ ]. \tag{2}$$

During a given time interval, the *performance number of path pair $P = \{p_j, p_k\}$*, denoted by y_P, is the probability that none of the two paths' loss rates exceed the congestion threshold:

$$y_P \equiv Pr[\ \Lambda_{p_j} \leq t_c\ \wedge\ \Lambda_{p_k} \leq t_c\ ]. \tag{3}$$

Informally, when we say that a path is "congestion-free," we mean that traffic along this path suffers relatively little loss; a path's performance is the frequency with which the path is congestion-free.

Link sequences are characterized by similar metrics, with the difference that a link sequence may behave differently toward each traffic class, because it may be non-neutral:

During a given time interval, the *class-c loss rate of link sequence L*, denoted by $\Lambda_L(c)$, is the probability with which L drops a packet from class c. $\Lambda_L(c)$ is a continuous random variable that may take values from 0 to 1.

During a given time interval, the *class-c performance number of link sequence L*, denoted by $x_L(c)$, is the probability that L's loss rate for class c does not exceed the congestion threshold t_c:

$$x_L(c) \equiv Pr[\ \Lambda_L(c) \leq t_c\ ]. \tag{4}$$

Informally, when we say that a link sequence is "congestion-free for traffic class c," we mean that it drops relatively few packets from this class; the class-c performance of a link sequence is the frequency with which the link sequence is congestion-free for class c.

3.2.2. Other Metrics

We have decided to use loss rate as a performance metric, but we could have also worked with throughput or latency.

Existing neutrality methods, like Glasnost [Marcel Dischinger et al., 2010] and DiffProbe [Kanuparthy and Dovrolis, 2010], rely on throughput differences: compare the throughput achieved along the same path p for traffic type x and traffic type y.

We choose not to rely on throughput differences, firstly because they do not always provide the hard evidence that we want. The fact that traffic type x achieves significantly higher throughput than traffic type y along the same path p could be due to non-neutrality on any link along p (not the specific link sequence we are interested in), or the source pacing itself differently for the two traffic types. The latter is possible, for example, if the source is a content server that paces itself differently when sending video versus other traffic. Our method should never mistaken self-pacing at the source with non-neutrality inside the network. Unlike Glasnost, in our experiments we often do not have control over the software used on both endpoints of a path, thus we cannot assume that such issues do not occur.

Secondly, a method that relies on throughput differences can be manipulated by the network it is checking. For instance, a network that throttles BitTorrent traffic may throttle *all* traffic exchanged between nodes running BitTorrent clients; or, a network that prioritizes speed-test traffic may prioritize *all* traffic sent by nodes running speed-test servers. In both cases, it would be impossible to catch this behavior by comparing throughput values achieved along the same path. We have no evidence that networks perform such manipulation today, and they have no reason to. However, our method would give them a reason, if successful, hence we want to make it robust to such manipulation.

Regarding latency, it is a useful metric to detect buffering, which may be caused by congestion or traffic shaping. The problem is that it is not influenced by traffic policing, which causes packet drops without introducing any buffering, therefore latency is a less useful metric than packet loss as it cannot be used to detect policing.

We chose loss over latency because there is strong, recent evidence that ISPs may employ policing as their throttling mechanism [Flach et al., 2016].

3.3. Non-neutrality

We focus on neutrality violations where traffic from different paths experiences different performance on the same link. This happens, for instance, when a link throttles traffic of a certain type, i.e. it upper-bounds its throughput to a fraction of the link's capacity; this can result in higher packet-loss rate and/or latency than the one experienced by unthrottled traffic. Hence, in our model, a traffic type is represented by a set of paths. For example, suppose a network link throttles traffic coming from a specific content provider; we model this by saying that the network has two "traffic classes," one class comprising all the paths that start at the content provider, and the other class comprising all the other paths. Similarly, suppose a network link throttles a specific kind of P2P traffic; again, we model this by saying that the network has two traffic classes, one class comprising all the paths that carry this form of P2P traffic, and the other class comprising all the other paths.

There are scenarios where neutrality violations occur where different types of traffic do not share the same links. For instance, an ISP may identify a specific type of traffic and decide to route it differently than regular traffic, on a path that might be longer or offer less capacity. While this would be a valid neutrality violation, its mechanism is very different than the one where a shared link throttles some of the traffic. The effects on the traffic and on the end-to-end performance metrics measurable by the user, such as packet loss rate or delay, are different; thus the two scenarios might require different detection methods. Our method does not address this kind of violations. We focus only on neutrality violations that occur on shared links, and we narrow the definition of "neutrality violation" to the situation where traffic from two different network paths experiences different performance when traversing the same network link.

We do not target specific mechanisms the non-neutral links may use to throttle traffic. These may include traffic policing, which drops traffic that exceeds a configured average rate; traffic shaping, which buffers in a separate queue traffic that exceeds a configured average rate; or more sophisticated packet queue management algorithms, such as weighted fair queuing (WFQ), weighted random early detection (WRED) etc.

As long as the throttling method introduces a non-negligible difference in throughput and loss rate for the different types of traffic, our technique should be able to detect it. This offers an advantage compared to ad-hoc methods tailored for a specific kind of throttling, since our method does not have to be adjusted every time a new mechanism is invented, or an existing one is changed.

3.4. Problem statement

Our input is a link sequence L and a set of paths that traverse L, such that their pairwise intersection is L, whose loss rates and performance numbers we can measure actively or passively (we want our method to work with both kinds of measurements).

Our goal is to assess whether L's performance differs significantly across traffic classes. For example, in Figure 1, the target link sequence is link l_1, and we want to assess whether l_1's class-1 performance differs significantly from l_1's class-2 performance.

Our primary use case is that our method is used to check whether a given network differentiates against or in favor of a given traffic type, for example, against BitTorrent, against Netflix video, or in favor of speed-test traffic. The user of the method defines two traffic classes: one that consists of the traffic type in question, and one that consists of other, "standard" traffic, such as HTTP requests and responses. The method determines whether the given network (more precisely, a link sequence in that network) has significantly different performance across the two traffic classes.

The main deployment challenge is finding the right paths to measure. As we will see, we need paths that can be grouped in pairs, such that each pair intersects at the same link sequence (and this is the link sequence whose neutrality we can assess). When the goal is to check a given ISP for differentiating against BitTorrent traffic, or prioritizing speed-test traffic, finding such paths is relatively easy for anyone with access to PlanetLab or cloud-hosted clients (we describe how we do it for our case study in Section 5.1). But when the goal is to check a given ISP for differentiating against traffic from a given content provider, then the only entity that is guaranteed to find the necessary paths is the content provider itself—but we, as third parties, may not be able to find and measure path pairs that carry traffic from the given content provider *and* intersect at the same link sequence within the given ISP.

Another use case—that we do not explore in this work—is that the method is used to check whether a given network employs differentiation in general. In this case, the user of the method generates measurement traffic that involves many different IP addresses and/or port numbers and randomly assigns measurement flows to traffic classes; if the method determines that the given network performs differently across classes, then the user can analyze the classes to determine the network's differentiation criteria. We did not try this, because, in order for it to be useful, one needs access to a large variety of measurement vantage points, which is beyond our reach. However, a popular content provider or distribution network should be able to use the method in this way, as long as they were willing to passively measure the loss rates and performance numbers of the end-to-end paths accessed by their users (which, to the best of our understanding, some of them already do).

3.5. Notation

Table 1 summarizes our notation. We observe the following rules:

Network topology elements are denoted by latin letters, specifically l denotes a link, L denotes a link sequence; p denotes a path, and P denotes a path pair.

We use $L_{p_j \cap p_k}$ to denote the longest common link sequence of paths p_j and p_k, and $L_{p_j \setminus p_k}$ to denote the longest link sequence traversed by path p_j but not path p_k. For example, in Figure 1, $L_{p_1 \setminus p_2} = \langle l_2 \rangle$, $L_{p_2 \setminus p_3} = \langle l_3 \rangle$, and $L_{p_1 \cap p_2} = \langle l_1 \rangle$.

Regarding performance metrics and measurements, capital greek letters are used for random variables, while small greek letters denote true values that random variables take at specific instances; small greek letters with a hat denote measurement-based estimates of true values. For example, in a particular time interval i, Λ_p takes value $\lambda_{p,i}$; if we estimate this value by observing some traffic from path p, the result is denoted by $\hat{\lambda}_{p,i}$.

3.5.1. Paths

We define the *congestion state* of a path p as Σ_p, a boolean random variable with a value of 0 indicating congestion, and 1 indicating good performance:

$$\Sigma_p = \Lambda_p \leq t_c \tag{5}$$

The true value Σ_p takes in a given interval i is denoted by $\sigma_{p,i}$, defined as:

$$\sigma_{p,i} = \lambda_{p,i} \leq t_c \tag{6}$$

The value of the congestion state we estimate from end-to-end measurements in a given interval i is denoted by $\hat{\sigma}_{p,i}$, which is computed from the loss rate estimate $\hat{\lambda}_{p,i}$:

$$\hat{\sigma}_{p,i} = \hat{\lambda}_{p,i} \leq t_c \tag{7}$$

Finally, the true performance number of a path p is denoted by y_p, while its estimate derived from end-to-end measurements is denoted by $\hat{y}_p$:

$$\hat{y}_p = \frac{1}{\|I\|} \sum_{i \in I} \hat{\sigma}_{p,i}, \tag{8}$$

where I is the set of all measurement intervals.

3.5.2. Path pairs

We use similar notation for path pairs, specifically Σ_P is a random variable that represents the congestion state of path pair $P = (p_j, p_k)$:

$$\Sigma_P = \Sigma_{p_j} \wedge \Sigma_{p_k} \tag{9}$$

The true congestion state of the path pair in a given interval is denoted by $\sigma_{P,i}$:

$$\sigma_{P,i} = \sigma_{p_j,i} \wedge \sigma_{p_k,i} \tag{10}$$

The estimated congestion state of the path pair in a given interval is denoted by $\hat{\sigma}_{P,i}$, defined as:

$$\hat{\sigma}_{P,i} = \hat{\sigma}_{p_j,i} \wedge \hat{\sigma}_{p_k,i} \tag{11}$$

Finally, the true performance number of the path pair P is denoted by y_P, while its estimate derived from end-to-end measurements is denoted by $\hat{y}_P$:

$$\hat{y}_P = \frac{1}{\|I\|} \sum_{i \in I} \hat{\sigma}_{P,i}, \tag{12}$$

3.5.3. Link sequences

Similar notation is used for link sequences, except that we must take into account that link sequences may exhibit different performance for each traffic class when they are non-neutral.

The congestion state of a link sequence L for traffic class c is denoted by the random variable:

$$\Sigma_L(c) = \Lambda_L(c) \leq t_c(L) \tag{13}$$

where $t_c(L)$ is the congestion threshold used for the link sequence[3].

The true congestion state of the link sequence in a given interval i is denoted by:

$$\sigma_{L,i}(c) = \lambda_{L,i}(c) \leq t_c(L) \tag{14}$$

Finally, the true performance number of the link sequence L for traffic class c is denoted by $x_L(c)$, as defined in Equation (4). Its estimate derived from end-to-end measurements by an inference algorithm is denoted by $\hat{x}_L(c)$; when the estimate has been derived from measurements taken for a single path pair P, we may denote it by $\hat{x}_L(P)$.

For simplicity, when a link sequence is known to be neutral, we may ommit the traffic class from the notation of the performance number, i.e. we write x_L instead of $x_L(c)$.

3.6. Assumptions

This section presents the assumptions that allow us to form the equation systems used in boolean network tomography.

3.6.1. Link independence

The congestion states of different links are independent. Formally, for any two different links l_j, l_k, we have that $\Sigma_{l_j} \perp \Sigma_{l_k}$.

This is a well understood assumption in network tomography; we will not discuss it in detail.

3.6.2. Performance correlation

Different paths from the same traffic class that share a common link sequence experience the same congestion states (at the same time) on that link sequence.

Formally, for any two different paths p_j, p_k from the same traffic class c, that share a link sequence L, we have that $\sigma_{L,i}(p_j) = \sigma_{L,i}(p_k)$, $\forall i \in I$, where $\sigma_{L,i}(p_j)$ is the congestion state of the link sequence L in interval i when taking into account only packets from path p_j when computing the loss rate.

This assumption is important, since it allows us to perform localization of congestion, by comparing congestion states of every two paths that intersect, pairwise,

[3]Traditionally, network tomography uses different congestion thresholds for link sequences, depending on the expected number of simultaneously congested links. This is related to the *Separability* assumption discussed in Section 3.6. However we will see that the choice of this parameter is not important in our case, as we do not use it in the inference algorithm.

Identifier	Formula	Description
i	–	A measurement time interval
I	–	The set of all measurement intervals during an experiment
l, L	–	A link, a link sequence
p, P	–	A path, a path pair
$L_{p_j \cap p_k}$	–	The longest common link sequence traversed by paths p_j and p_k
$L_{p_j \setminus p_k}$	–	The longest link sequence traversed by path p_j but not path p_k
Λ_p	–	The loss rate of path p (random variable)
$\lambda_{p,i}$	–	The true value of Λ_p during time interval i
$\hat{\lambda}_{p,i}$	–	The estimated value of $\lambda_{p,i}$ during time interval i
t_c	–	The congestion threshold
Σ_p	$\Lambda_p \leq t_c$	The congestion state of path p (random variable)
$\sigma_{p,i}$	$\lambda_{p,i} \leq t_c$	The true value of the congestion state of path p in interval i
$\hat{\sigma}_{p,i}$	$\hat{\lambda}_{p,i} \leq t_c$	The estimated value of the congestion state of path p in interval i
y_p	$Pr[\, \Sigma_p = 1 \,]$	The performance number of path p
$\hat{y}_p$	$\frac{1}{\lVert I \rVert} \sum_{i \in I} \hat{\sigma}_{p,i}$	The estimate of y_p
Σ_P	$\Sigma_{p_j} \wedge \Sigma_{p_k}$	The congestion state of path pair $P = (p_j, p_k)$ (random variable)
$\sigma_{P,i}$	$\sigma_{p_j,i} \wedge \sigma_{p_k,i}$	The true congestion state of path pair P in interval i
$\hat{\sigma}_{P,i}$	$\hat{\sigma}_{p_j,i} \wedge \hat{\sigma}_{p_k,i}$	The estimated congestion state of path pair P in interval i
y_P	$Pr[\, \Sigma_P = 1 \,]$	The performance number of path pair $P = (p_j, p_k)$
$\hat{y}_P$	$\frac{1}{\lVert I \rVert} \sum_{i \in I} \hat{\sigma}_{P,i}$	The estimate of y_P
$\Lambda_L(c)$	–	The loss rate of link sequence L for class c (random variable)
$\Sigma_L(c)$	$\Lambda_L(c) \leq t_c$	The congestion state of link sequence L for class c
$\sigma_{L,i}(c)$	$\lambda_{L,i}(c) \leq t_c$	The true congestion state of link sequence L in interval i
$x_L(c)$	$Pr[\, \Sigma_L(c) = 1 \,]$	The performance number of link sequence L for class c
$\hat{x}_L(P)$	–	The estimate of $x_L(c)$ derived from measurements for path pair P

Table 1. Notation for topology elements, congestion states and performance metrics.

as it will be seen in the Section 3.7. The intuition behind it is that when two paths traverse a link sequence that is neutral, or one that is non-neutral but the paths belong to the same traffic class, the link sequence will treat the traffic neutrally, thus it should experience similar performance. However, there are two possible scenarios in which this assumption is violated in practice:

Firstly, the loss rate of a shared, congested link may be slightly below the congestion threshold when taking into account packets from all paths that traverse the link, but each of the paths might observe a slightly different loss rate; for some paths, the loss rate might exceed the congestion threshold, while for others it will be below. Therefore some paths observe congestion, while others do not. This effect is possible, since the packet sets used to compute the two loss rate values are different. The violation becomes less likely to occur as the difference between loss rate and congestion threshold increases. We take into account this measurement problem, relaxing this assumption, in Section 4.6.

Secondly, the shared link may use fair queuing (FQ) [Nagle, 1985, 1986] instead of FIFO as queue management algorithm, which may lead to placing packets of the two paths into separate queues. While FQ attempts to ensure that the two paths achieve similar egress throughput, it does not guarantee that the loss rates are similar. On the contrary, if one of the paths has higher ingress throughput than the other, it will probably experience a higher loss rate. However, if the two paths have similar ingress throughputs, the loss rates should be similar: either their ingress throughput is lower than the fair share enforced by the link, in which case both paths do not saturate their queues; or their ingress throughput is higher than the fair share, in which case both paths saturate their queues and experience congestion. It is therefore important to only consider path pairs that send traffic at similar throughput through a link sequence that we suspect might perform fair queuing. If this is not possible, we should at least use a large number[4] of path pairs to infer the congestion state (or the neutrality) of the link sequence; if the link sequence is neutral but uses fair queuing, the inference results from multiple pairs will likely be inconsistent, therefore we will not misdetect the link sequence as non-neutral.

3.6.3. Separability

A path is congested if and only if at least one of its links (or link sequences) is congested. Formally, if a path p from class c consists of the disjoint link sequences $\mathcal{L} = \{L_1, L_2, ..., L_k\}$, we have that:

$$\Sigma_p = \wedge_{L \in \mathcal{L}} \Sigma_L(c) \tag{15}$$

This assumption links the path states to the link states, allowing us to form our systems of equations, and eventually to reason about the link congestion state while only observing end-to-end measurements.

A possible scenario in which the assumption is violated occurs when a path traverses multiple bottleneck links, the loss rate of each being below the congestion threshold, but the overall loss rate of the path exceeds the congestion threshold. Paths that traverse only one of the bottlenecks do not observe congestion, while paths that traverse multiple bottlenecks do. The violation becomes less likely to occur as the difference between loss rate and congestion threshold increases. We take into account this measurement problem, relaxing this assumption, in Section 4.6.

[4]In our experiments, we encountered networks that appeared to employ fair queuing; with as little as 6-8 paths, we could choose a sufficient number of path pairs to observe inconsistent results, thus avoid false positives.

3.6.4. Canonicity

The performance number of any link sequence or path is non-zero.

A performance number equal to zero represents complete network failure, or traffic blocking (such as blackhole routing). We are not interested in diagnosing such problems. While this assumption is not crucial for our algorithms to work, it helps simplify the mathematical equations.

3.7. Neutrality Inference using Maximum Likelihood Estimation

We now describe an algorithm that performs non-neutrality detection and localization, given that all the assumptions from section 3.6 are satisfied and that measurements are perfect. This is essentially the method proposed in [Z. Zhang, Mara, and Argyraki, 2014]. We name it *Straw*, as we will use it as a reference to compare our work against in Section 4.

Consider the topology from Figure 2, which shows a network with five links and four paths. There are two traffic classes, class 1 (in blue), consisting of all traffic carried by path pair $\{p_1, p_2\}$, and traffic class 2 (in red), consisting of all traffic carried by path pair $\{p_3, p_4\}$. Link l_1 (in red) is non-neutral: it is never congested for class 1, whereas it *is* congested for class 2 half of the time. Also, suppose that the other links are never congested (for either traffic class).

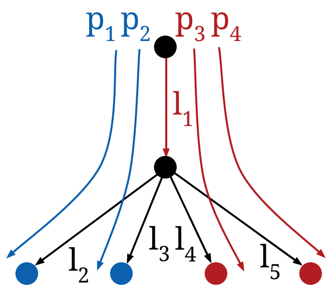


Figure 2. Example network.

In this scenario, the performance numbers of the paths are $y_{p_1} = y_{p_2} = y_{P_1} = 1$ (paths 1 and 2 are congestion-free with probability 1), while $y_{p_3} = y_{p_4} = y_{P_2} = 0.5$ (paths 3 and 4 are congestion-free with probability 0.5, and they are congestion-free at the same time). The performance numbers of the links are $x_{l_1}(1) = 1$, $x_{l_1}(2) = 0.5$ and $x_{l_k} = 1, \forall k = 2..5$. Suppose that we only know the path and path pair performance numbers, and we would like to compute the link performance numbers.

Consider path pair $P_1 = \{p_1, p_2\}$. We can write:

$$\begin{aligned} y_{p_1} &= x_{l_1}(1) \cdot x_{l_2} \\ y_{p_2} &= x_{l_1}(1) \cdot x_{l_3} \\ y_{P_1} &= x_{l_1}(1) \cdot x_{l_2} \cdot x_{l_3}. \end{aligned} \tag{16}$$

As shown in [Nguyen and Thiran, 2007b], we can solve this system and obtain $x_{l_1}(1), x_{l_2}, x_{l_3}$ as a function of $y_{p_1}, y_{p_2}, y_{P_1}$. Specifically, for link l_1 and traffic class 1:

$$x_{l_1}(1) = \frac{y_{p_1} y_{p_2}}{y_{P_1}} \tag{17}$$

Consider path pair $P_2 = \{p_3, p_4\}$. We can write:

$$\begin{aligned} y_{p_3} &= x_{l_1}(2) \cdot x_{l_4} \\ y_{p_4} &= x_{l_1}(2) \cdot x_{l_5} \\ y_{P_2} &= x_{l_1}(2) \cdot x_{l_4} \cdot x_{l_5}. \end{aligned} \tag{18}$$

We can solve this system using MLE and obtain $x_{l_1}(2), x_{l_4}, x_{l_5}$ as a function of $y_{p_3}, y_{p_4}, y_{P_2}$. For link l_1 and traffic class 2, we obtain:

$$x_{l_1}(2) = \frac{y_{p_3} y_{p_4}}{y_{P_2}} \tag{19}$$

The two systems yield different values for link l_1's performance: System (16) yields $x_{l_1}(1) = 1$, whereas System (18) yields $x_{l_1}(2) = 0.5$. Note that the path p_3's and path p_4's concurrent congestion is attributed to link l_1, as opposed to concurrent congestion in links l_4 and l_5, because we assume link independence—hence we cannot have $x_{l_4} = x_{l_5} = x_{l_4} \cdot x_{l_5} = 0.5$.

Because of link l_1's non-neutrality, *Straw*'s observations of P_1 are inconsistent with its observations of P_2: the former indicate that link l_1 is congestion-free, whereas the latter indicate that l_1 is congested half of the time; since these cannot both be true in a neutral network, the network is non-neutral; and since the only link that is shared between the four paths is l_1, this link must be the source of non-neutrality.

3.7.1. Algorithm

The *Straw* algorithm takes as input a link sequence L and a set of paths P^* that can be grouped in pairs, such that each pair intersects exactly at L. Additionally, it needs several parameters to be set by the user, as listed in Table 2. Then *Straw* performs the following three steps:

Parameter	**Description**
n_{int}	Number of time intervals in experiment
n_{pkt}	Number of measurement packets per time interval
t_c	Congestion threshold
t_n	Neutrality threshold

Table 2. The parameters taken by the *Straw* algorithm.

Step 1: End-to-end measurements

Straw divides time in n_{int} fixed-size intervals and measures, in each interval, the performance of each path and path pair in P^*. For instance, consider a path p. In interval i, *Straw* considers n_{pkt} measurement packets (packets that are sent on path p), and estimates the value of p's loss rate $\lambda_{p,i}$ as the fraction $\hat{\lambda}_{p,i}$ of measurement packets that were dropped. At the end, *Straw* estimates p's performance y_p as $\hat{y}_p$, the fraction of intervals where $\hat{\lambda}_{p,i} \leq t_c$, i.e. p's estimated loss rate was below the congestion threshold.

Step 2: Inference of link performance.

For each defined traffic class c, *Straw* infers L's class-c performance $x_L(c)$ by creating and solving a tomographic system of equations based on a different path pair. For instance, consider a path pair $P_c = \{p_j, p_k\}$ carrying traffic from class c. *Straw* creates the following system of equations:

$$\begin{aligned} \hat{y}_{p_j} &= x_L(c) \cdot x_{L_{p_j \setminus p_k}} \\ \hat{y}_{p_k} &= x_L(c) \cdot x_{L_{p_k \setminus p_j}} \\ \hat{y}_P = x_L(c) \cdot x_{L_{p_j \setminus p_k}} &\cdot x_{L_{p_k \setminus p_j}} \end{aligned} \tag{20}$$

On the left side of each equation we have the performance number of a path or path pair, as measured in Step 1; on the right side, we have a product of the performance numbers of different link sequences, which are unknown. Solving this system yields a maximum likelihood estimate (MLE) of $x_L(c)$.

Step 3: Neutrality assessment.

Straw labels L as non-neutral if its inferred performance numbers are significantly different from each other. Specifically, *Straw* defines L's *neutrality bias* as the maximum difference between any two of its performance numbers:

$$\text{bias} \equiv \max_{c,m} |x_L(c) - x_L(m)| \tag{21}$$

Then, it labels L as non-neutral if bias $> t_n$, where t_n is a configurable *neutrality threshold.*

4. Reliable Neutrality Inference using Bayesian Statistics

The theory behind the *Straw* algorithm (presented in Section 3.7) assumes perfect measurements. As a result, in practice, *Straw* suffers from errors that we discuss in 4.1. While we cannot eliminate these errors, we redesign the measurement process such that we can reason about them and avoid magic thresholds; we describe how in Section 4.1. We evaluate each the various components of our new measurement process when we introduce them. At the end of the chapter, in Section 4.7, we also evaluate the measurement process as a whole, in a realistic setting that is outside our control.

4.1. Sources of Error

Straw is subject to four sources of error, each related to an algorithm parameter:

#1. Inaccurate path-congestion measurements (during one interval of n_{pkt} packets)

Step 1 estimates whether, in each interval, a path's loss rate exceeds the congestion threshold t_c. The accuracy of each such estimate depends on the number of measurement packets n_{pkt}. Moreover, the closer the path's actual loss rate is to t_c, the harder it is to accurately determine whether one exceeds the other, hence the larger the n_{pkt} we need. The problem is that, without ground truth—without some a-priori knowledge of the network conditions—we do not know how to reason about the reliability of these estimates.

#2. Inaccurate path-performance measurements (over multiple n_{int} intervals)

Step 1 also estimates a path's overall performance, by monitoring the path's congestion over n_{int} measurement intervals. The accuracy of each such estimate depends on n_{int}. Moreover, the error of each such estimate can be significantly amplified in Step 2, where the path-performance estimates serve as input to linear systems of equations. Again, without some a-priori knowledge about the stability of network conditions, we do not know how to reason about the reliability of these estimates and the link performance numbers that we infer from them.

#3. Inappropriate neutrality definition (the choice of t_n, the safety margin)

Step 3 determines the neutrality of a link sequence by comparing its maximum performance gap to a neutrality threshold t_n. Picking t_n involves the usual balancing act between false positives (favored by a low threshold) and false negatives (favored by a high threshold). Moreover, we should ideally pick t_n as a function of the number of intervals n_{int}: the fewer the intervals, the less reliable the path performance estimates in Step 1 and the link performance inference in Step 2, hence the more conservative we should be in Step 3 (the higher we should set t_n) to avoid false negatives. However, because we have no way to quantify the reliability of the estimation and inference in Steps 1 and 2, we also have no basis for setting t_n.

#4. Inappropriate congestion definition (the choice of t_c)

Step 1 compares each path's estimated loss rate to a congestion threshold t_c; not surprisingly, the value of t_c has a dramatic impact on the results. An inappropriate threshold can lead to false negatives: if it is too low, paths appear to be congested most of the time; if it is too high, paths appear to be congestion-free most of the time; either way, paths appear to experience congestion with similar probabilities, which may incorrectly indicate a neutral network. Moreover, an inappropriate threshold can lead to false positives; for instance, if $t_c = 0.01$, and path p_1 experiences loss rate 0.009 (just below the threshold), while path p_2 experiences loss rate 0.011 (just above), then the two paths appear to experience congestion with very different probabilities, which may incorrectly indicate a non-neutral network. In short, *Straw* may miss real neutrality violations or detect non-existent ones because of an unfortunate combination of network conditions and the chosen congestion threshold.

4.2. Example Topology

In a small network like the one in Figure 2, *Straw* works well; to motivate our changes, we need a more realistic example, where the sources of error we described in Section 4.1 manifest clearly; we use the network in Figure 3, which has a similar topology to RedIris, the Spanish academic network [”RedIRIS Network Map,” n.d.], obtained from the Internet Topology Zoo project [Knight et al., 2011].

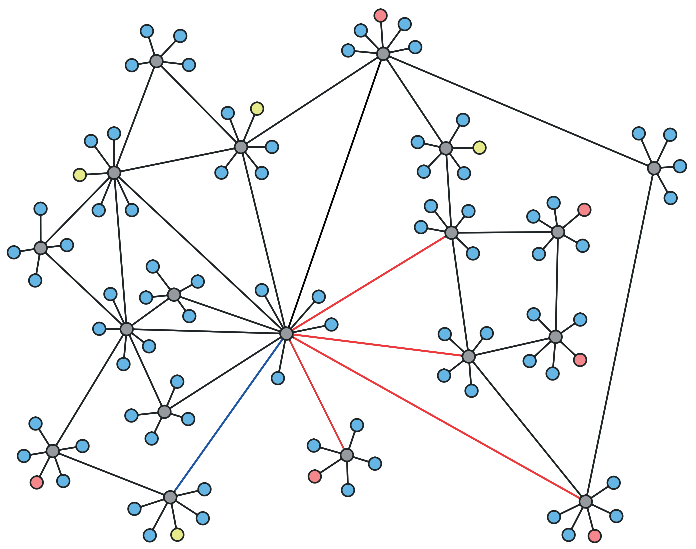


Figure 3. Example network with 4 non-neutral shared links (in red), 17 neutral shared links (in black and in blue), and 86 neutral edge links (in black). The blue (red) nodes exchange class-1 (class-2) TCP traffic. The yellow nodes exchange UDP traffic. The non-neutral links police class 2.

Our example network carries a mix of TCP and UDP flows of various sizes and includes 4 shared non-neutral links (in red) and several shared neutral links (in black). The non-neutral links police the traffic exchanged by a particular set of nodes (the red ones), so this traffic constitutes our class-2, while the rest of the traffic constitutes class 1. The policing and generated traffic are such that the non-neutral links introduce significant loss to class 2, while some of the neutral links introduce significant loss to both classes. Our goal is to assess the neutrality of all link sequences that are traversed by at least 2 path pairs, one from each class (as link l_1 in Figure 2). A key challenge is to distinguish non-neutrality from congestion, i.e., neither allow congestion to mask non-neutrality (and miss the fact that certain congested link sequences are non-neutral), nor mistaken congestion for non-neutrality (and miss the fact that certain congested link sequences are neutral).

We emulated this network in LINE [Mara, 2017], our network emulator built on top of the PF_RING [Deri and others, 2004] packet processing framework. The transmission rate of the shared links is 150Mbps, while that of the non-shared links is 20–40Mbps; link latency is 3–10ms. The experiment lasts 40 minutes, with 40 measurement intervals of 1 minute each.

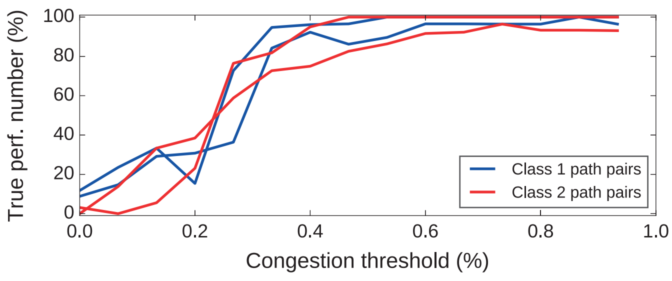
In each experiment, the end-hosts generate TCP and UDP traffic. Each pair of communicating end-hosts starts a number of TCP flows with the transfer size following a Pareto distribution; when a TCP flow ends, a new one starts after an idle time that is governed by an exponential distribution. We chose this model because there is evidence that it captures well the communication between pairs of Internet end-hosts [Crovella and Bestavros, 1997], but it is not crucial to our results—it is just one way of generating dynamic traffic patterns. To avoid symmetries, we vary the number of flows adjacent to end-hosts, as well as the parameters of the Pareto and exponential distributions (that govern flow size and inter-flow idle time, respectively). These parameters are configured as follows:

Each node exchanges TCP traffic only with other nodes of the same color. Blue and red pairs exchange TCP traffic in the following configuration: 35 blue and 35 red pairs exchange long TCP flows (infinite transfer size); 12 blue and 12 red pairs exchange medium TCP flows (average transfer size 25MB); and 110 red pairs exchange medium-long flows (average transfer size 50MB). Finally, there are 4 red–yellow pairs exchanging Poisson UDP traffic at a rate of 30Mbps. All the TCP flows serve as measurement traffic, while UDP serves as background traffic (it affects network conditions, but does not participate in the measurements).

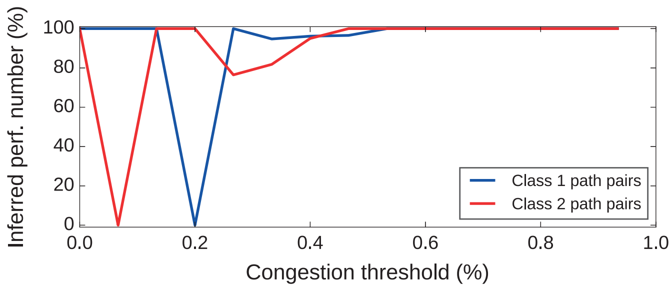
All traffic exchanged by the blue nodes belongs to class 1, and by the red nodes to class 2. Three of the non-neutral links police class 2 at 30% of their capacity, while the fourth one polices class 2 at 40% of its capacity.

4.3. Bayesian Performance Inference

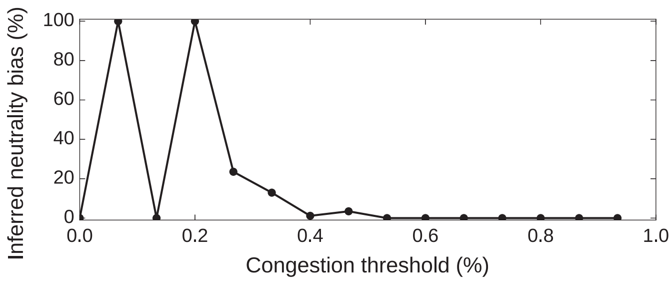
We start from error source #2: inaccurate path-performance measurements. *Straw* estimates path performance in Step 1 and uses the outcome to infer link performance in Step 2; given that the estimates may be inaccurate, we must determine whether the resulting inference is reliable.



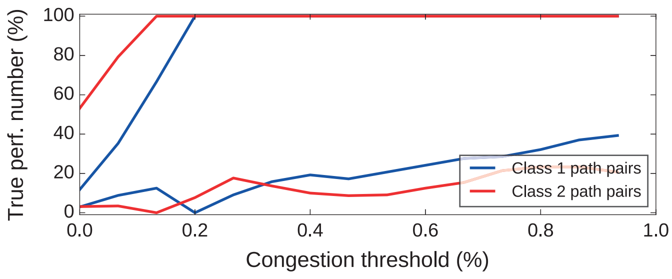
(4a) Ground truth for the common neutral link.



(4b) *Straw*'s inferred curves.



(4c) *Straw*'s inferred neutrality bias.



(4d) Ground truth for the 4 outer links.

Figure 4. Class-1 (blue) and class-2 (red) performance for two path pairs intersecting over a neutral link sequence, as a function of the congestion threshold. The topology is equivalent to the one in Figure 2. Top: true performance on the common link sequence. Upper mid: performance as inferred by *Straw*. Lower mid: performance bias as inferred by *Straw*. Bottom: true performance on the outer link sequences.

4.3.1. Illustration of the Problem

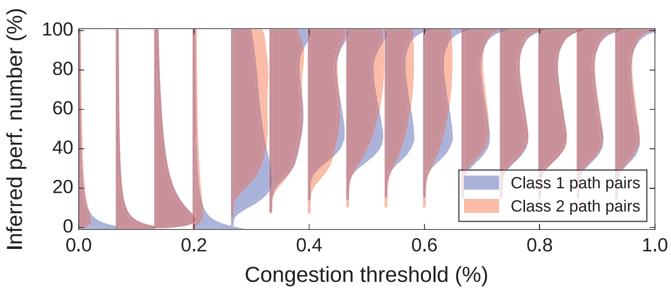
Figure 4 illustrates how *Straw* mischaracterizes a *neutral* shared link (the blue link in Figure 3). Figure 4a shows the ground truth, i.e., how frequently the link is actually congestion-free with regard to class 1 (blue) and class 2 (red), as a function of the congestion threshold t_c. Figure 4b shows *Straw*'s inference, i.e., how frequently the link appears to be congestion-free with regard to each class based on two path pairs, one carrying class-1 traffic, the other class-2: for $t_c = 0.07$, the link appears to be congestion-free 100% with regard to class 1 and 0% with regard to class 2, while exactly the opposite happens for $t_c = 0.2$. Figure 4c shows *Straw*'s conclusion, i.e., the neutrality bias it computes for the link: the maximum gap between class-1 and class-2 performance is 100%, hence *Straw* incorrectly concludes that the link is non-neutral.
The problem is that *Straw*'s math is very sensitive to inaccurate input when the input values are small: System (20) has a unique solution as long as it is not the case that both paths of a path pair are never congestion-free. If this assumption is violated, then multiple solutions fit the end-to-end measurements: congestion could be due to the shared part of the path pair, or due to the non-shared parts, or both. If the assumption holds but one or both paths are rarely congestion-free, then many solutions "almost fit" the end-to-end measurements, and a minor error in estimating path performance might cause *Straw* to pick the wrong solution.

We can confirm that this is what went wrong in the inference by analyzing the ground truth for the other links in the network. Figure 4d shows the true performance numbers for the four links that act as bottlenecks for the four paths used in *Straw*'s inference. For small values of the congestion threshold t_c, two of the four paths are rarely congestion-free. The number of measurement intervals is not sufficient for accurately estimating such small numbers. The estimation error of each number is small, but it is amplified by System (20) and causes *Straw*'s inference (Figure 4b) to differ significantly from the ground truth (Figure 4a).

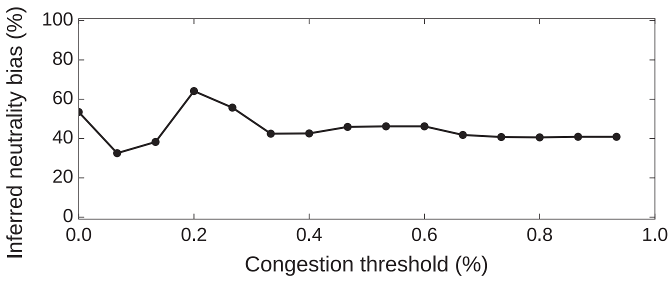
4.3.2. Our Approach

Our solution is to replace *Straw*'s Step 2 with Bayesian inference: given a target link sequence L, instead of inferring its performance number $x_L(c)$ by solving a tomographic system of equations, we infer $x_L(c)$'s posterior distribution $\Pr[x_L(c)|E]$ given the observed measurements E. The inferred posterior distribution does not represent a unique solution, but the probability of each possible solution $x_L(c)$ that could match the observed measurements. This intrinsically encodes the reliability with which we can solve the equations: when the posterior distribution is narrow, i.e. it has a tall peak at the likeliest value, the measurements strongly support a single solution or a small set of solutions close to it; by contrast, the more uniform the posterior distribution, the less the likeliest value "stands out," and the less reliable our inference.

Figure 5a illustrates how this helps: For each value of the congestion threshold t_c, we infer two posterior distributions, one for the link's performance with regard to class 1 (blue distribution) and one with regard to class 2 (red distribution). The fact that none of our inferred distributions has a clear peak tells us that we cannot



(5a) Inferred posterior distribution.



(5b) Inferred posterior neutrality bias.

Figure 5. Performance computed with Bayesian inference.

reliably infer the link's performance number with regard to any class, hence we cannot reason about the link's neutrality.

Our method captures the fact that not all inferences are equally reliable. *Straw* assumes perfect measurements, which ensures that the tomographic system of equations it solves has a unique solution. The moment we admit imperfect measurements, we also admit multiple possible solutions. If, assuming imperfect measurements, there exist multiple solutions that fit our measurements, and they all occur with similar probabilities, then we cannot reliably infer the true one. This is precisely the information provided by the posterior distribution.

4.3.3. Our Approach in Detail

We now show how the posterior distribution can be computed from end-to-end measurements. Consider a simpler topology consisting of a single path pair, as shown in Figure 6, since the computation needs to be performed not for four paths but one path pair at a time, essentially replacing Equation System (20). Assume that the two paths p_1 and p_2 belong to the same traffic class c.
Our input consists of the end-to-end measurements for the paths $\hat{y}_{p_1}$, $\hat{y}_{p_2}$ and the path pair $\hat{y}_P$, and of the number of measurement intervals n_{int}; we denote them as $E = (\hat{y}_{p_1}, \hat{y}_{p_2}, \hat{y}_P, n_{int})$. The goal is to compute the posterior distribution $\Pr[x_{l_1}(c)|E]$.

4.3.4. Numerical Computation using Monte Carlo Simulation

We propose a method to approximate the posterior distribution using Monte Carlo (MC) simulation. We restate our goal: compute the posterior distribution

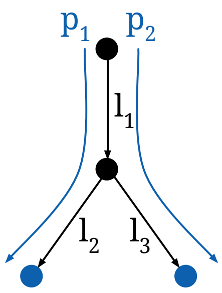


Figure 6. Network topology.

$\Pr[x_{l_1}(c)|E]$ given the end-to-end measurements $E = (\hat{y}_{p1}, \hat{y}_{p2}, \hat{y}_P, n_{int})$, for a topology consisting of two paths and three links, as shown in Figure 6.

We should clarify that our method has nothing to do with network (e.g., NS3) simulation as typically used to evaluate network proposals—we do not simulate traffic generation, network conditions, etc. We simply use MC simulation as a numerical method to solve a mathematical problem for which we could not find a tractable analytical solution.

Our MC simulation creates a large number of possible "ground truths," i.e., combinations of performance numbers for the three links in the network; this can be done by iterating over the range of all real numbers between 0 and 1 with a small fixed step ϵ. For each ground truth, and for each link, the simulation creates n_{int} random samples of the link sequence's congestion status that are compatible with this ground truth; from these samples, it computes the congestion statuses of the two paths during each interval, and the estimated performance numbers of the two paths and the path pair. If the three computed performance numbers are the same as the ones measured in the experiment (i.e., we created a ground truth that fits the end-to-end measurements), then we record the corresponding samples; otherwise we discard them. After the simulation has created enough ground truths for the recorded sample set to reach $100,000$ values or so, we use the samples to estimate the posterior distribution of l_1's performance number $x_{l_1}(c)$. This method is implemented by Algorithm 1.

Algorithm 1: Posterior Computation with Monte-Carlo simulation

```
# Computes the posterior distribution $Pr[x_{l_1}(c)|E]$
# Input: E as $\hat{y}_{p1}, \hat{y}_{p2}, \hat{y}_P, n_{int}$
# Output: D = mapping $x_{l_1}(c) \in [0,1] \rightarrow [0,1]$
# Parameters:
# * min_samples = 100,000
# * $\epsilon$ = 0.01
function ComputePosterior($\hat{y}_{p1}, \hat{y}_{p2}, \hat{y}_P, n_{int}$):
  D = {}
  n_samples = 0
  while n_samples < min_samples:
    for x_1 in range(0.0, 1.0, step=$\epsilon$):
      for x_2 in range(0.0, 1.0, step=$\epsilon$):
        for x_3 in range(0.0, 1.0, step=$\epsilon$):
```

```
        y_1, y_2, y_12 = SimulateExperiment(x_1, x_2, x_3, $n_{int}$)
        # Check if the simulation result matches the measurements
        if y_1 == $\hat{y}_{p_1}$ and y_2 == $\hat{y}_{p_2}$ and y_12 == $\hat{y}_P$:
          n_samples += 1
          D[x_1] += 1
  # Normalize D
  for x, value in D:
    D[x] /= n_samples
  return D

# Simulates an experiment, computing a set of end-to-end measurements
# from the ground truth.
# Input: ground truth for the 3 links and the number of intervals
# Output: one set of end-to-end measurements
function SimulateExperiment(x_1, x_2, x_3, $n_{int}$):
  # End-to-end estimates
  y_1 = y_2 = y_12 = 0.0
  # Simulate $n_{int}$ intervals
  for i in range(0, $n_{int}$):
    link_state_1 = Bernoulli(x_1)
    link_state_2 = Bernoulli(x_2)
    link_state_3 = Bernoulli(x_3)
    # Compute path states from link states
    path_state_1 = link_state_1 and link_state_2
    path_state_2 = link_state_1 and link_state_3
    path_pair_state = path_state_1 and path_state_2
    # Update end-to-end estimates
    if path_state_1:
      y_1 += 1
    if path_state_2:
      y_2 += 1
    if path_pair_state:
      y_12 += 1
  # Normalize estimates to [0, 1]
  y_1 /= $n_{int}$
  y_2 /= $n_{int}$
  y_12 /= $n_{int}$
  return (y_1, y_2, y_12)

# Runs a Bernoulli trial.
# Input: success probability of Bernoulli trial
# Output: result of Bernoulli trial (boolean)
function Bernoulli(prob_success):
  return rand_uniform(0, 1) <= prob_success
```

While Algorithm 1 is quite simple and parallelizable, it is too inefficient to be practical. The problem is in the step that checks if the simulation result matches the measurements from function `ComputePosterior`. Given a fixed ground truth, the number of possible end-to-end outcomes is in $O(n_{int}^3)$; although some are much less likely than others, and some are impossible. This means that the probability that a simulation result matches the measurement can be as small as n_{int}^{-3} or even a few orders of magnitude smaller; which means that for an experiment with 100 intervals, we would have to run 1 million simulations to get just one sample of D, and we need $100,000$ of them; assuming that one random number generation requires 10 arithmetic operations, the computation would require up to $100,000$ samples $\times$ $1,000,000$ simulations $\times$ 100 intervals $\times$ 10 operations $= 10^{14}$ operations, possibly more. This means that running the inference algorithm for just one path pair might take from several hours to several days. Clearly we need a method that offers better performance.

What is striking about Algorithm 1 is that it does not compute simply $Pr[x_{l_1}(c)|E]$; in fact, it performs all the simulations needed to compute $Pr[x_{l_1}(c)|E]$ for any possible value of E (with a fixed n_{int}). This means that for a given n_{int}, we could run the simulations in advance, and instead of filtering by E, we could store all the results in a mapping $E \rightarrow x_{l_1}(c) \rightarrow [0,1]$, which we could use as a lookup table. Then, when we need to run the inference for a concrete value of E, we simply search the lookup table to find the posterior distribution. While creating the lookup table is straightforward, using it poses two challenges.

Firstly, the lookup table requires a non-negligible amount of storage, due to the large number of posterior distributions we have to store. In practice it is difficult to fix the number of intervals in advance, for which the reason is shown in Section 4.6. Therefore we have to compute the mappings for many possible values, on the order of a few hundreds. For each one, the number of distributions depends on the number of possible values of E, which in turn depends on the granularity of the measured path performance numbers, which is $\max(\frac{1}{n_{int}}, \epsilon)$; therefore we may have up to $\min(n_{int}, \frac{1}{\epsilon})^3$ values of E for each n_{int}. Each distribution requires storing $\frac{1}{\epsilon}$ pairs of floating point values, thus for values of n_{int} up to $1,000$, the required storage may add up to hundreds of GB unless we use compression. However, we can take advantage of the fact that the posterior distributions do not change significantly for small variations of n_{int}; for example, for $n_{int} = 100$ and $n_{int} = 110$ they are very similar. Therefore we can use a smaller granularity for n_{int}. We used a step of 10 for values under 100; a step of 50 for values between 100 and 500; and a step of 100 for values between 500 and 1200. This has reduced storage requirements by a factor of 40. We reduced space further by using fixed-point arithmetic (for $\epsilon = 0.01$, we can store real numbers using only 8 bits instead of 32), and compressing the table further with DEFLATE [Deutsch, 1996] as implemented by the zlib library [Gailly and Adler, 1998], which reduced the size by a factor of approx. 10. The final storage size is 117 MB, smaller than a naive data format by a factor of 1600.

Secondly, the lookup is non-trivial: due to potentially mismatched values of n_{int}, it might not be possible to find an exact match in the table for a measured E. Therefore support for approximate lookups is necessary. For simplicity, we opted to first choose the subtable computed for the closest value of n_{int}; then look up the value from the subtable that is closest to the given $\hat{y}_{p_1}, \hat{y}_{p_2}, \hat{y}_P$, using Euclidean distance as a metric. To implement this search efficiently, we indexed the values with ball trees.

The final method for precomputing and finding the posterior distribution is sketched in Algorithm 2. The precomputation step was performed by a parallel implementation on a machine with 48 CPU cores; we have sped up the simulations further by using AVX-2 SIMD instructions to compute 4 values at a time in parallel, and we used the xorshift128+ pseudo-random number generator (PRNG), currently among the fastest non-cryptographic PRNG, while still providing good quality random numbers [Marsaglia and others, 2003; Oya et al., 2011; Vigna, 2017]. Overall, the optimized precomputation step ran for about 2 weeks.

Algorithm 2: Posterior Precomputation with Monte-Carlo simulation

```
# Precomputes the posterior distribution lookup table
# Input: $n_{int}$
# Output: D = mapping $E \in [0,1]^3 \to x_{l_1}(c) \in [0,1] \to [0,1]$
# Parameters:
# * repeats = 48 * 4096
# * $\epsilon$ = 0.01
function PrecomputeTable($n_{int}$):
  D = {}
  for x_1 in range(0.0, 1.0, step=$\epsilon$):
    for x_2 in range(0.0, 1.0, step=$\epsilon$):
      for x_3 in range(0.0, 1.0, step=$\epsilon$):
        for r in range(0, repeats):
          y_1, y_2, y_12 = SimulateExperiment(x_1, x_2, x_3, $n_{int}$)
          D[y_1][y_2][y_12][x_1] += 1
  # Normalize each D[y_1][y_2][y_12]
  for y_1 in D:
    for y_2 in D[y_1]:
      for y_12 in D[y_1][y_2]:
        total = sum([value for x, value in D[y_1][y_2][y_12]])
        for x, value in D[y_1][y_2][y_12]:
          D[x] /= total
  return D

# Precomputes all lookup tables
# Parameters:
# * n_values = [10, 20, .., 100, 150, .., 500, 600, .., 1200]
function PrecomputeAllTables():
  for n in n_values:
    T = BallTree(PrecomputeTable(n))
    Store(T)

# Computes the posterior distribution $Pr[x_{l_1}(c)|E]$
# Input: E as $\hat{y}_{p_1}, \hat{y}_{p_2}, \hat{y}_P, n_{int}$
# Output: D = mapping $x_{l_1}(c) \in [0,1] \to [0,1]$
# Parameters:
# * n_values = [10, 20, .., 100, 150, .., 500, 600, .., 1200]
function ComputePosterior($\hat{y}_{p_1}, \hat{y}_{p_2}, \hat{y}_P, n_{int}$):
  n = FindClosest(n_values, $n_{int}$)
  T = Load(n)
  y_1, y_2, y_12 = FindClosest(T, $\hat{y}_{p_1}, \hat{y}_{p_2}, \hat{y}_P$)
  return T[y_1][y_2][y_12]
```

4.3.5. Alternative Method: Analytical Derivation

As an alternative to the Monte Carlo method, we also provide an analytical method for computing the posterior distribution. Unfortunately, we show that the computation is intractable, thus this method is not practical. We still present our attempt of solving the problem analytically for the sake of completeness, and also to justify the necessity of the less elegant MC solution.

Under the assumptions from Section 3.6, the state of the network in each interval i is fully characterized by the congestion states of the three links, i.e. the values taken by the random variables $\Sigma_{l_1}(c)$, $\Sigma_{l_2}(c)$ and $\Sigma_{l_3}(c)$. The path congestion states are random variables that depend on the link congestion states:

$$\begin{aligned} \Sigma_{p_1} &= \Sigma_{l_1}(c) \wedge \Sigma_{l_2}(c) \\ \Sigma_{p_2} &= \Sigma_{l_1}(c) \wedge \Sigma_{l_3}(c) \\ \Sigma_{P} &= \Sigma_{l_1}(c) \wedge \Sigma_{l_2}(c) \wedge \Sigma_{l_3}(c) \end{aligned} \tag{22}$$

All the possible states the network can be in during one interval, and their probabilities, are shown in Table 3. We use the following notation:

- u_1 is the value taken by $x_{l_1}(c) = Pr[\ \Sigma_{l_1}(c) = 1\ ]$;
- u_2 is the value taken by $x_{l_2}(c) = Pr[\ \Sigma_{l_2}(c) = 1\ ]$;
- u_3 is the value taken by $x_{l_3}(c) = Pr[\ \Sigma_{l_3}(c) = 1\ ]$.

We can group the states the network can be in during one interval into four possible end-to-end outcomes, with the corresponding probabilities:

$$\begin{aligned} Pr[\ \Sigma_{p_1}(c) = 0 \wedge \Sigma_{p_2}(c) = 0\ ] = &(1-u_1)(1-u_2)(1-u_3) + (1-u_1)(1-u_2)u_3 + \\ &+ (1-u_1)u_2(1-u_3) + (1-u_1)u_2u_3 + \\ &+ u_1(1-u_2)(1-u_3) \ = \\ =& u_1u_2u_3 - u_1u_2 - u_1u_3 + 1 \end{aligned} \tag{23}$$

$$\begin{aligned} Pr[\ \Sigma_{p_1}(c) = 0 \wedge \Sigma_{p_2}(c) = 1\ ] &= u_1(1-u_2)u_3 \ = \\ &= u_1u_3 - u_1u_2u_3 \end{aligned} \tag{24}$$

$$\begin{aligned} Pr[\ \Sigma_{p_1}(c) = 1 \wedge \Sigma_{p_2}(c) = 0\ ] &= u_1u_2(1-u_3) \ = \\ &= u_1u_2 - u_1u_2u_3 \end{aligned} \tag{25}$$

$$Pr[\ \Sigma_{p_1}(c) = 1 \wedge \Sigma_{p_2}(c) = 1\ ] = u_1u_2u_3 \tag{26}$$

We now consider multiple intervals. We define the following variables:

Congestion states						Probability
l_1	l_2	l_3	p_1	p_2	P	–
0	0	0	0	0	0	$(1-u_1)(1-u_2)(1-u_3)$
0	0	1	0	0	0	$(1-u_1)(1-u_2)u_3$
0	1	0	0	0	0	$(1-u_1)u_2(1-u_3)$
0	1	1	0	0	0	$(1-u_1)u_2u_3$
1	0	0	0	0	0	$u_1(1-u_2)(1-u_3)$
1	0	1	0	1	0	$u_1(1-u_2)u_3$
1	1	0	1	0	0	$u_1u_2(1-u_3)$
1	1	1	1	1	1	$u_1u_2u_3$

Table 3. All the possible states of the network during one interval.

- $k_{1,2}$ is the number of intervals with the outcome $(\Sigma_{p_1} = 0 \wedge \Sigma_{p_2} = 0)$.
- $k_1 - k_{1,2}$ is the number of intervals with the outcome $(\Sigma_{p_1} = 0 \wedge \Sigma_{p_2} = 1)$.
- $k_2 - k_{1,2}$ is the number of intervals with the outcome $(\Sigma_{p_1} = 1 \wedge \Sigma_{p_2} = 0)$.
- $n_{int} - k_1 - k_2 + k_{1,2}$ is the number of intervals with the outcome $(\Sigma_{p_1} = 1 \wedge \Sigma_{p_2} = 1)$.

Note that $(k_1, k_2, k_{1,2}, n_{int})$ is equivalent to the end-to-end measurements $E = (\hat{y}_{p_1}, \hat{y}_{p_2}, \hat{y}_P, n_{int})$, since we can write $\hat{y}_{p_1} = \frac{n_{int}-k_1}{n_{int}}$, $\hat{y}_{p_2} = \frac{n_{int}-k_2}{n_{int}}$ and $\hat{y}_P = \frac{n_{int}-k_1-k_2+k_{1,2}}{n_{int}}$. Then the probability to observe a given value of the end-to-end measurements for the entire experiment can be written as:

$$\begin{aligned}
&Pr[\ E|x_{l_1}(c) = u_1 \wedge x_{l_2}(c) = u_2 \wedge x_{l_3}(c) = u_3\ ] = \\
&Pr(\ \hat{y}_{p_1} = \frac{n_{int}-k_1}{n_{int}} \wedge \hat{y}_{p_2} = \frac{n_{int}-k_2}{n_{int}} \wedge \hat{y}_P = \frac{n_{int}-k_1-k_2+k_{1,2}}{n_{int}}| \\
&|x_{l_1}(c) = u_1 \wedge x_{l_2}(c) = u_2 \wedge x_{l_3}(c) = u_3\ ] = \\
&= \binom{n_{int}}{k_{1,2}}\binom{n_{int}-k_{1,2}}{k_1-k_{1,2}}\binom{n_{int}-k_1}{k_2-k_{1,2}} \times \\
&\times Pr[\ \Sigma_{p_1}(c) = 0 \wedge \Sigma_{p_2}(c) = 0|x_{l_1}(c) = u_1 \wedge x_{l_2}(c) = u_2 \wedge x_{l_3}(c) = u_3\ ]^{k_{1,2}} \times \\
&\times Pr[\ \Sigma_{p_1}(c) = 0 \wedge \Sigma_{p_2}(c) = 1|x_{l_1}(c) = u_1 \wedge x_{l_2}(c) = u_2 \wedge x_{l_3}(c) = u_3\ ]^{k_1-k_{1,2}} \times \\
&\times Pr[\ \Sigma_{p_1}(c) = 1 \wedge \Sigma_{p_2}(c) = 0|x_{l_1}(c) = u_1 \wedge x_{l_2}(c) = u_2 \wedge x_{l_3}(c) = u_3\ ]^{k_2-k_{1,2}} \times \\
&\times Pr[\ \Sigma_{p_1}(c) = 1 \wedge \Sigma_{p_2}(c) = 1|x_{l_1}(c) = u_1 \wedge x_{l_2}(c) = u_2 \wedge x_{l_3}(c) = u_3\ ]^{n_{int}-k_1-k_2+k_{1,2}}
\end{aligned} \tag{27}$$

Which expands to:

$$
\begin{aligned}
Pr[\ E|x_{l_1}(c) = u_1 \wedge x_{l_2}(c) = u_2 \wedge x_{l_3}(c) = u_3\ ] = \\
= \binom{n_{int}}{k_{1,2}}\binom{n_{int}-k_{1,2}}{k_1-k_{1,2}}\binom{n_{int}-k_1}{k_2-k_{1,2}}\times \\
\times(u_1u_2u_3 - u_1u_2 - u_1u_3 + 1)^{k_{1,2}}\times \\
\times(u_1u_3 - u_1u_2u_3)^{k_1-k_{1,2}}\times \\
\times(u_1u_2 - u_1u_2u_3)^{k_2-k_{1,2}}\times \\
\times(u_1u_2u_3)^{n_{int}-k_1-k_2+k_{1,2}}
\end{aligned}
\tag{28}
$$

We can integrate it over u_2 and u_3 to compute:

$$
\begin{aligned}
Pr[\ E|x_{l_1}(c) = u_1\ ] = \\
= \int_0^1\int_0^1 Pr[\ E|x_{l_1}(c) = u_1 \wedge x_{l_2}(c) = u_2 \wedge x_{l_3}(c) = u_3\ ]du_2du_3
\end{aligned}
\tag{29}
$$

To finally obtain the posterior distribution, we apply Bayes' theorem and assume that all possible link performance numbers are equally likely ($Pr[x_{l_1}(c) = u_1] = 1$):

$$
\begin{aligned}
Pr[\ x_{l_1}(c) = u_1|E\ ] &= \frac{Pr[\ E|x_{l_1}(c) = u_1\ ]Pr[\ x_{l_1}(c) = u_1\ ]}{Pr[\ E\ ]} = \\
&= \frac{Pr[\ E|x_{l_1}(c) = u_1\ ]}{\int_0^1 Pr[\ E|x_{l_1}(c) = u_1\ ]du_1}
\end{aligned}
\tag{30}
$$

Unfortunately, the double integral from Equation (29) is difficult to solve for symbolic values of $n_{int}, k_1, k_2, k_{1,2}$, the powers of the polynoms from Equation (28); and still difficult for specific measured values due to the large numeric values of n_{int} and some, or all of k_1, k_2, and $k_{1,2}$. Therefore this derivation cannot be used in practice.

4.3.6. Evaluation Through Simulation

The simulations used to evaluate the Bayesian performance inference algorithm were performed as follows: Consider a topology consisting of three links and two paths, as in Figure 2. Starting from a known ground truth, i.e. the performance numbers of the three links x_{l_1}, x_{l_2} and x_{l_3}, we use three corresponding Bernoulli processes to generate random congestion states for the links in each of the n_{int} intervals. From the link congestion states, we compute the path congestion states and the end-to-end performance number estimates for the path and the path pair. From the end-to-end data, we infer the performance number of the common link with two methods: (i) the MLE, as computed by *Straw*; (ii) the posterior distribution, as computed by our Bayesian algorithm.

The parameters used in the simulations are the following:

- n_{int} is set to 60, 300, 600 or 1200 (with 60-minute intervals, this corresponds to 1 hour, 5 hour, 10 hour and 20 hour long experiments);
- The link performance numbers may be in one of three configurations:

  - *good* = 1.0, i.e. the link is never congested;
  - *light* = 0.8, i.e. the link is congested 20% of the time;
  - *heavy* = 0.2, i.e. the link is congested 80% of the time.

- There are four scenarios:
  - Common link lightly congested, side links good;
  - Common link good, side links lightly congested;
  - Common link heavily congested, side links good;
  - Common link good, side links heavily congested.

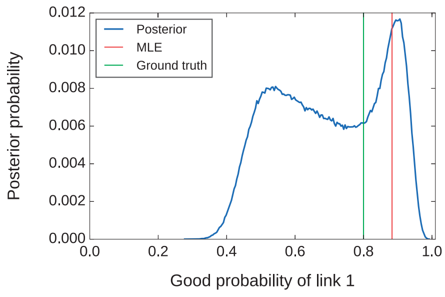
Figures 7 and 8 show the scenarios in which the common link is either good or lightly congested. Due to the low amount of congestion in the network, these should be easy scenarios for both algorithms. Indeed, we can see that: (i) the MLE is close to the ground truth; (ii) the posterior distribution is also in concordance with the ground truth, which falls within regions of high likelihood; (iii) the posterior is very wide for lower numbers of intervals, showing high uncertainty, and only converges visibly towards the ground truth starting from 600 intervals.

Interestingly, the posterior distributions from Figures 7a and 8a appear similar. In the first scenario, the estimated good probabilities of the paths and the path pair are all equal to 0.88, since the path congestion states are perfectly correlated, pointing correctly to congestion on the common link. In the second scenario, they are different but only slightly: 0.71, 0.81 and 0.61, because we observe "false" correlation since in some intervals the two side links happen to be congested simultaneously. For small number of intervals, it is not possible to discern between true and "false" correlation, thus the posterior distribution is wide.

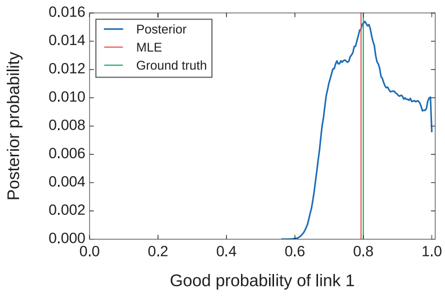
The scenarios showing heavy congestion are shown in Figures 9 and 10. When the common link is heavily congested, all algorithms perform well: the peak of the posterior distribution is close to the MLE and to the ground truth; and even for a small number of intervals, the posterior distribution has a tall, relatively narrow peak, showing high certainty.

The difficult scenario is when the side links are heavily congested and the common link is good. Again, in this case we observe false correlation (since in most intervals both paths are congested), and the posterior distribution is wide, showing high uncertainty, as expected. What is remarkable is that the MLE is highly inaccurate, having an error of about 0.3 as seen in Figures 10a and 10b. Consider what would happen if this result were used as by *Straw* in neutrality inference: when comparing this path pair with another that also traversed link 1, but had the side links only lightly congested (as in Figure 8), *Straw* would compute a large neutrality bias and incorrectly classify link 1 as non-neutral. By contrast, the Bayesian method would compare two wide posterior distributions, and correctly decide that the link cannot be classified as non-neutral with sufficient certainty: the bias is 62%, respectively 65% for the data for 60 and 300 intervals, well below the typically 90% or 95% significance level required to conclude that the link is non-neutral.

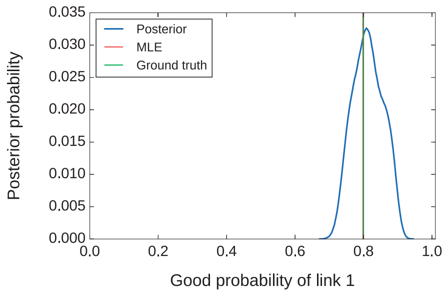
We conclude that the Bayesian algorithm is more reliable at detecting congestion than the MLE, which experiences very large errors under certain situations, leading to potential false positives in non-neutrality detection. The disadvantage of using the Bayesian is that it may not be able to return a conclusion with high certainty when the number of intervals is small; but this is a deficiency of the measurement data, not of the algorithm per se.



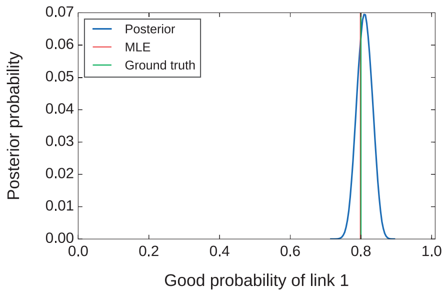
(7a) 60 intervals.



(7b) 300 intervals.

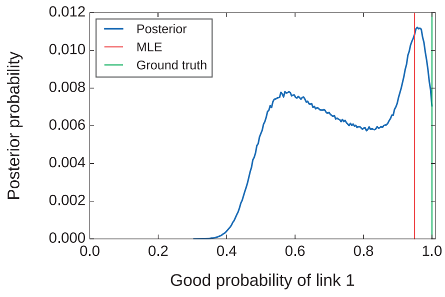


(7c) 600 intervals.

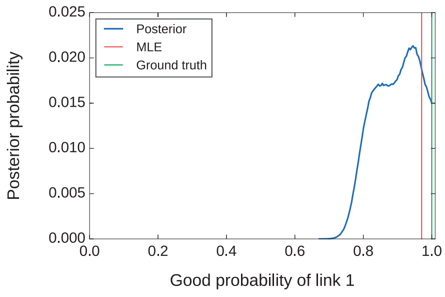


(7d) 1200 intervals.

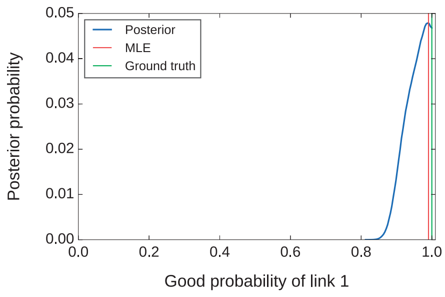
Figure 7. Common link lightly congested, side links good.



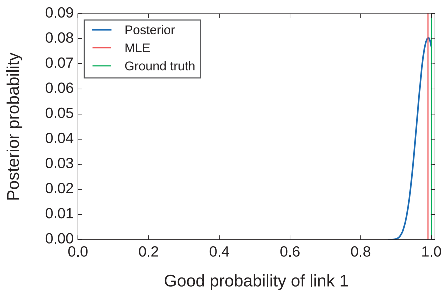
(8a) 60 intervals.



(8b) 300 intervals.

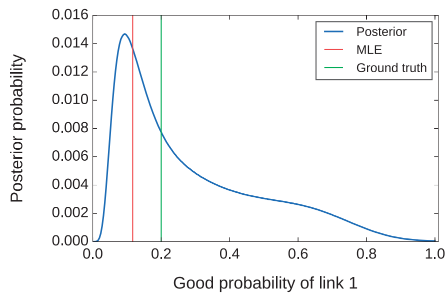


(8c) 600 intervals.

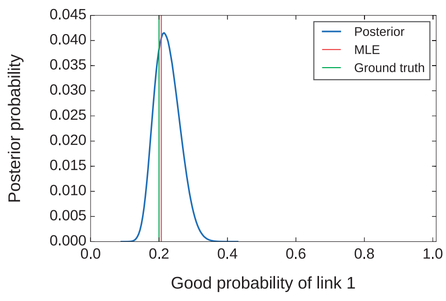


(8d) 1200 intervals.

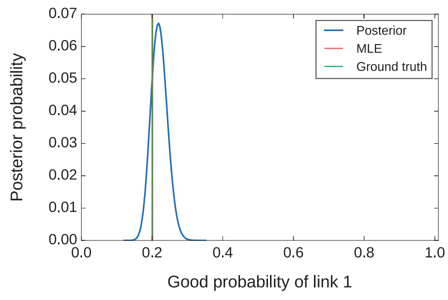
Figure 8. Common link good, side links lightly congested.



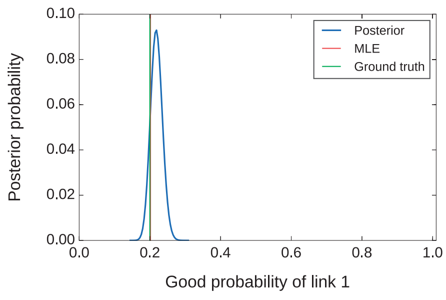
(9a) 60 intervals.



(9b) 300 intervals.

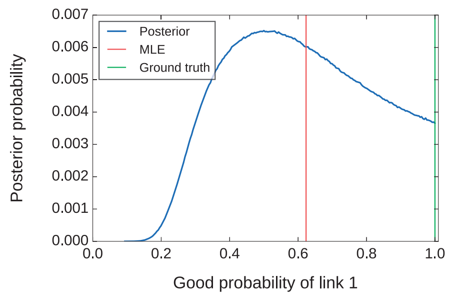


(9c) 600 intervals.

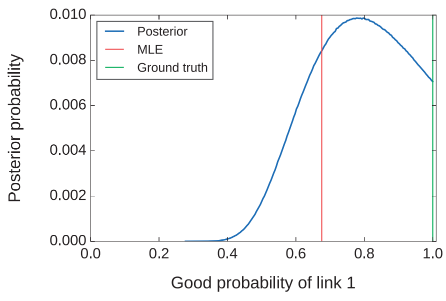


(9d) 1200 intervals.

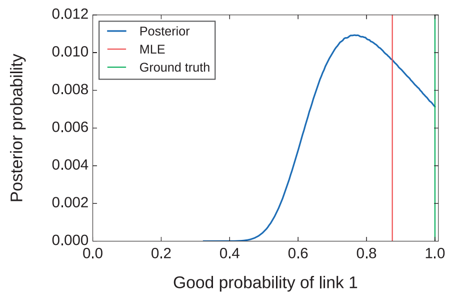
Figure 9. Common link heavily congested, side links good.



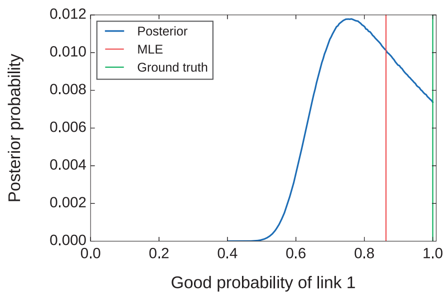
(10a) 60 intervals.



(10b) 300 intervals.



(10c) 600 intervals.



(10d) 1200 intervals.

Figure 10. Common link good, side links heavily congested.

4.4. Non-neutrality Inference with Confidence

We now address *Straw*'s error source #3: how to set an appropriate neutrality threshold.

Bayesian inference allows us to redefine link neutrality in a more meaningful way: not based on how much the link's performance numbers differ, but on our confidence that they do. In particular, we can redefine L's bias as:

$$\text{bias} \equiv \max_{n,m} \Pr[x_L(m) > x_L(n) + \epsilon | E)] \tag{31}$$

where ϵ is a small value to account for rounding errors in the MC simulation, which we set to $\max(1\%, 1/n_{int})$ unless otherwise noted. In other words, a link's bias is the probability that its performance for any two different traffic classes differs by more than ϵ. We label L as non-neutral when bias $> \alpha$, where α is our desired level of confidence.

Figure 5b shows the (redefined) neutrality bias of the target link in our running example. For all the values of the congestion threshold t_c, the neutrality bias (our confidence that that the performance of the link differs for the two traffic classes) ranges from 0 to 65%, reflecting the fact that there exist many "ground truths" that fit the end-to-end measurements, hence we cannot reason about the link's neutrality.

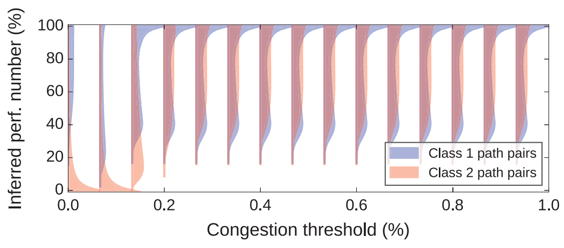
Why replace one pre-defined configuration parameter (t_n) with two (ϵ and α)? Because, in our opinion, the latter better captures what network users want to know: whether a link differentiates against certain traffic and how confident we are that it does. *Straw* labels a link as non-neutral when its performance discrepancy exceeds threshold t_n, not because network users would not care about smaller discrepancies, but because it is, in general, harder to reliably detect smaller discrepancies. Instead, we separate the issue of how large a performance discrepancy we want to detect (that's ϵ) from the issue of how confident we are that the discrepancy exists (that's α).

4.5. Threshold-Free Inference

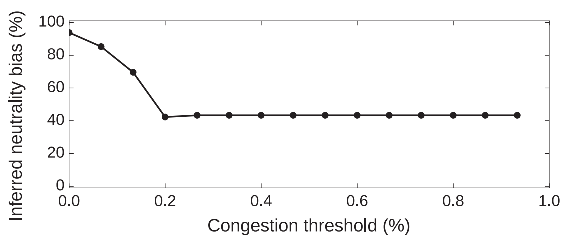
We now address *Straw*'s error source #4: how to set an appropriate congestion threshold. We found this to be the most obvious yet most frustrating source of error: obviously, the congestion threshold t_c affects our ability to detect whether different traffic classes experience congestion with different frequencies; but how can we choose a value of t_c without assuming knowledge of differentiation mechanisms and configurations?

Our solution is to remove the need for choosing a congestion threshold: *Straw* considers one congestion threshold and infers, for each path pair, one performance *number*; instead, we consider the entire range of possible congestion thresholds and infer, for each path pair, a performance *curve*, as well as a posterior distribution for each point in this curve.

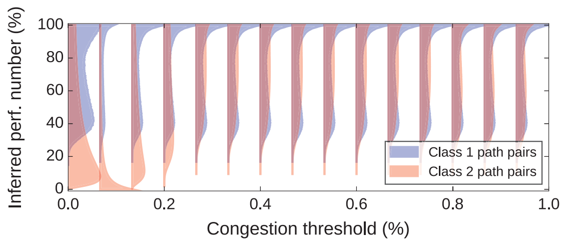
The insight is simple but, in our opinion, deeper than it may seem at first: Choosing the congestion threshold is a hard, open problem in traditional network tomography, where the ultimate goal is to infer link performance, and an unfortunate threshold simply yields wrong results. Instead, our goal is to infer significant



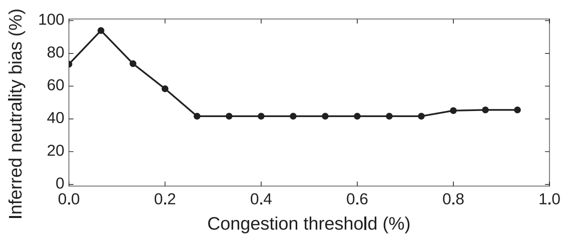
(11a)



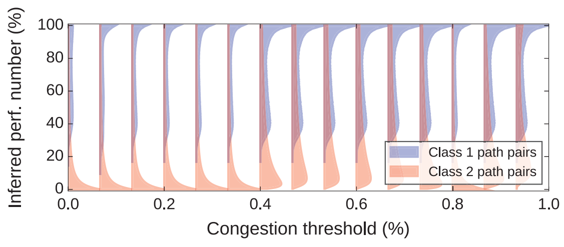
(11b)



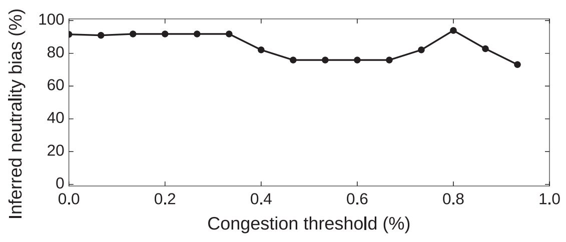
(11c)



(11d)



(11e)



(11f)

Figure 11. Neutrality assessment of three non-neutral shared links from Figure 3.

differences in link performance from the point of view of different traffic aggregates; if such a difference exists, then *some* subset of all the possible congestion thresholds will reveal it, i.e., some portion of the inferred performance curves will be different; if it does not exist, then all inferred performance curves will be similar.

In more detail, *Straw* considers one congestion threshold t_c and infers, for each path pair P_c, one performance number $x_L(c) \equiv Pr[\Sigma_L(c) = 0] \equiv Pr[\Lambda_L(c) \leq t_c]$, which is, essentially, a point in the cumulative distribution function (CDF) of the random variable $\Lambda_L(c)$ (L's loss rate for class c). Instead, we consider a set of congestion thresholds, $t_c = 0..1$, that covers the entire range of loss-rate values from 0 to 1, and we repeat our measurement and inference steps for each one. As a result, we infer, for each path pair, a set of performance numbers $\{x_L(c, t_c) \equiv Pr[\Lambda_L(c) \leq t_c] \mid t_c = 0..1\}$. By connecting these points, we obtain a performance curve, which is essentially an approximation of the CDF of the random variable $\Lambda_L(c)$.

With this, we can amend our neutrality detection test to avoid having to choose the congestion threshold t_c. We redefine L's bias as:

$$\text{bias} \equiv \max_{n,m} \max_{t_c} \Pr[x_L(m, t_c) > x_L(n, t_c) + \epsilon | E)] \tag{32}$$

In words, a link's bias is defined as the probability that there exists a congestion threshold such that the link's performance for any two different traffic classes differs by more than ϵ. We label L as non-neutral when bias $> \alpha$, where α is our desired level of confidence. This is akin to the two-sample Kolmogorov-Smirnov test, a standard non-parametric test used to determine whether two samples belong to the same underlying probability distribution [Andrey, 1933; Nikolai, 1948].

Figure 11 illustrates how this helps: Each column concerns a shared *non-neutral* link from Figure 3: the top graph shows the inferred posterior distribution of the link's performance with regard to the two traffic classes, while the bottom graph shows the corresponding neutrality bias. All links have maximum neutrality bias 94% (we are that confident that they differentiate) but for different congestion thresholds. The most interesting case is the first link (left-most graphs), where both classes suffer different amounts of loss during different intervals due to congestion; moreover, class 2 always suffers at least some minimal amount of loss due to policing. As a result, we detect the link's behavior most confidently when $t_c = 0$: in Figure 11a, the red distribution has a clear peak at 0 for $t_c = 0$, and, in Figure 11b, the neutrality bias peaks at 94% for $t_c = 0$. However, as the congestion threshold increases beyond 0.2%, the effect we catch is the one due to congestion, not policing: in Figure 11a, when $t_c > 0.2\%$, it becomes hard to distinguish the blue from the red distributions, and our confidence that the link differentiates goes to only 40%.

4.6. Measurement Filtering

Lastly, we address *Straw*'s error source #1. Even though we do not choose a single congestion threshold, we still perform threshold-dependent measurements in Step 1: like *Straw*, during each measurement interval, we estimate whether a path's loss rate exceeds a threshold t_c given n_{pkt} measurement packets; unlike *Straw*, we repeat this for multiple values of t_c. We must ensure that each of these estimates is reliable.

There are three slightly different ways in which the measurements may be erroneous, which share the same underlying cause.

Firstly, the congestion state of the path might not be measured correctly. Due to the fact that the number of packets per interval is limited, the estimated loss rate may be different than the true theoretical value of the loss rate as used in our model. As the number of packets increases, the two should become closer in value, but will never be perfectly identical as long as the number of packets is finite. While the difference may be small relative to the values, the problem is that when the congestion threshold t_c is similar in value to the loss rate, the true value may be slightly below while the estimated one is slightly above the threshold, or vice versa. Thus, the estimated congestion state of the path will not match the true one. The consequence is that our equations that link path congestion states to link congestion states no longer model the network correctly, thus the inference algorithm may return incorrect results. While we could increase the number of packets per interval, and thus increase the sampling rate for estimating the loss rate more accurately, this comes with a trade-off: we either reduce the number of intervals, which may reduce the non-neutrality detection rate; or we increase the duration of the experiment, which may not be practical (for example, due to topology changes or changes in network conditions). Therefore none of these solutions are desirable.

Secondly, the *performance correlation* assumption (shown in Section 3.6) may be violated. Consider the case when the loss rate of a congested link shared by multiple paths is close in value to the congestion threshold. Due to the fact that the traffic of each of the paths that traverse the link consists of disjoint packet sets, each path might observe a slightly different loss rate on that link. For some paths, the loss rate might exceed the congestion threshold, while for others it might be below. Therefore some paths might observe congestion, while others do not. Again, this affects our system of equations and may cause errors in the inference.

Thirdly, the *separabiliy* assumption (shown in Section 3.6) may be violated. When a path traverses multiple bottleneck links, the loss rate of each being below the congestion threshold, the overall loss rate of the path might exceed the congestion threshold. Paths that traverse only one of the bottlenecks do not observe congestion, while paths that traverse multiple bottlenecks do; our systems of equations are yet again affected. For two bottleneck links, this problem occurs when the loss rate of the links falls within in the interval $[t_c/2, t_c]$. This is a more rare problem than the others. Due to the very specific conditions that have to be met so that a path has multiple bottleneck links (the maximum achievable throughput over the links must be almost identical and vary slightly over time so that queuing occurs on every link), we do not think it is very likely to occur with two bottlenecks, and with more than two it is higly unlikely. But if it does occur, we note that when the loss rate is not similar in value to the congetion threshold, there is no ill effect over the equations and the inference.

All three problems share the same cause: the loss rate (true or estimated) is too close to the congestion threshold, leading to incorrect congestion state estimates. Our solution is to filter the measurements, avoiding the intervals in which we suspect incorrect estimates. For this purpose, we introduce a reliability metric for the measurements. Without a-priori knowledge of network conditions, we may not always choose a good n_{pkt} and t_c combination, but we can always use our reliability metric to filter out unreliable measurements. Our metric is the probability of a correct estimation given the estimated path loss rate $\hat{\lambda}_{p,i}$ and the measurement parameters n_{pkt} and t_c:

$$a = Pr[\ (\Lambda_p \le t_c) = (\hat{\lambda}_{p,i} \le t_c) | \hat{\lambda}_{p,i}, n_{pkt}, t_c\ ] \tag{33}$$

We set a reliability threshold a and filter out measurements that do not meet it. We compute this metric entirely from end-to-end measurements, without knowing any ground truth about the underlying network.

4.6.1. Computation of the Measurement Reliability Metric

We now show how the reliability metric from Equation (33) can be computed.

We make the following assumptions:

- The loss rate is stable during the measurement interval. This is reasonable in practice for intervals that are not too large.
- The transmission states (dropped or forwarded) of the probing packets are independent. In practice, this is not true for packets that arrive at a link in temporal proximity to each other, i.e. packet losses are correlated. Previous work has shown that correlation disappears for packets spaced by 10-20

ms or more [Tsang et al., 2000]. We have confirmed empirically using auto-correlation analysis of the end-to-end data that with a spacing of at around 50 ms between packets, almost all packet losses are independent. This imposes an upper bound on the number of packets we can use per interval, if the interval size T is given in seconds:

$$n_{pkt} \leq \frac{T}{0.050} = 20T \tag{34}$$

For example, with 100 second intervals, we can consider up to $2,000$ packets.

When these two assumptions hold, the number of lost packets during one interval N_{lost} is a random variable that follows a binomial distribution with parameters[5] n_{pkt} and λ:

$$Pr[\ N_{lost} = m\ ] = \binom{n_{pkt}}{m} \lambda^m (1-\lambda)^{n_{pkt}-m},\ m = 0, 1..n_{pkt} \tag{35}$$

The distribution of the estimated loss rate $\hat{\Lambda} = \frac{N_{lost}}{n_{pkt}}$ is:

$$Pr[\ \hat{\Lambda} = \hat{\lambda}\ ] = \begin{cases} \binom{n_{pkt}}{n_{pkt}\hat{\lambda}} \lambda^{n_{pkt}\hat{\lambda}} (1-\lambda)^{n_{pkt}(1-\hat{\lambda})}, & \text{if } \hat{\lambda} = 0, \frac{1}{n_{pkt}}..1 \\ 0, & \text{otherwise.} \end{cases} \tag{36}$$

Equation (36) is the basis for quantifying the measurement errors. We use the notation $f_L(n_{pkt}, \lambda, \hat{\lambda})$ for the probability mass function of the estimated loss rate:

$$f_L(n_{pkt}, \lambda, \hat{\lambda}) = \binom{n_{pkt}}{n_{pkt}\hat{\lambda}} \lambda^{n_{pkt}\hat{\lambda}} (1-\lambda)^{n_{pkt}(1-\hat{\lambda})} \tag{37}$$

In practice, the number of packets n_{pkt}, and the loss rate estimate $\hat{\lambda}$ are known, but the true value of the loss rate λ is not known. There are two possible situations:

1. When the observed congestion state of the path is *good*, i.e. $\hat{\lambda} \leq t_c$:

$$\begin{aligned} a =& Pr[\ \Lambda \leq t_c | \hat{\Lambda} = \hat{\lambda}\ ] = \frac{Pr[\ \Lambda \leq t_c \wedge \hat{\Lambda} = \hat{\lambda}\ ]}{Pr[\ \hat{\Lambda} = \hat{\lambda}\ ]} \\ =& \frac{\int_0^{t_c} Pr[\ \Lambda = \lambda \wedge \hat{\Lambda} = \hat{\lambda}\ ] d\lambda}{\int_0^1 Pr[\ \hat{\Lambda} = \hat{\lambda} | \Lambda = \lambda\ ] Pr[\ \Lambda = \lambda\ ] d\lambda} = \frac{\int_0^{t_c} Pr[\ \hat{\Lambda} = \hat{\lambda} | \Lambda = \lambda\ ] Pr[\ \Lambda = \lambda\ ] d\lambda}{\int_0^1 Pr[\ \hat{\Lambda} = \hat{\lambda} | \Lambda = \lambda\ ] Pr[\ \Lambda = \lambda\ ] d\lambda} \\ =& \frac{\int_0^{t_c} f_L(n_{pkt}, \lambda, \hat{\lambda}) Pr[\ \Lambda = \lambda\ ] d\lambda}{\int_0^1 f_L(n_{pkt}, \lambda, \hat{\lambda}) Pr[\ \Lambda = \lambda\ ] d\lambda} \end{aligned} \tag{38}$$

2. When the observed congestion state of the path is *congested*, i.e. $\hat{\lambda} > t_c$:

$$\begin{aligned} a =& Pr[\ \Lambda > t_c | \hat{\Lambda} = \hat{\lambda}\ ] = \frac{Pr[\ \hat{\Lambda} = \hat{\lambda} \wedge \Lambda > t_c\ ]}{Pr[\ \hat{\Lambda} = \hat{\lambda}\ ]} \\ =& \frac{\int_{t_c}^1 f_L(n_{pkt}, \lambda, \hat{\lambda}) Pr[\ \Lambda = \lambda\ ] d\lambda}{\int_0^1 f_L(n_{pkt}, \lambda, \hat{\lambda}) Pr[\ \Lambda = \lambda\ ] d\lambda} \end{aligned} \tag{39}$$

[5]To simplify the notation, we drop the path and the interval from the loss rate, i.e. we write λ instead of $\lambda_{p,i}$. The theory from this subsection refers to a single path and a single measurement interval.

Thus the final formula for computing the measurement reliability metric is:

$$a = \begin{cases} \dfrac{\int_0^{t_c} f_L(n_{pkt}, \lambda, \hat{\lambda}) Pr[\ \Lambda = \lambda\ ] d\lambda}{\int_0^1 f_L(n_{pkt}, \lambda, \hat{\lambda}) Pr[\ \Lambda = \lambda\ ] d\lambda}, & \text{if } \hat{\lambda} \leq t_c \\ \dfrac{\int_{t_c}^{1} f_L(n_{pkt}, \lambda, \hat{\lambda}) Pr[\ \Lambda = \lambda\ ] d\lambda}{\int_0^1 f_L(n_{pkt}, \lambda, \hat{\lambda}) Pr[\ \Lambda = \lambda\ ] d\lambda}, & \text{otherwise.} \end{cases} \tag{40}$$

Using this formula in practice poses two problems.

Firstly, in order to compute the accuracy metric, we need to know the prior distribution of the ground truth Λ to compute $Pr[\ \Lambda = \lambda\ ]$. This is obviously unknown. Without any prior information, we could assume that Λ is uniformly distributed from 0 to 1, i.e. $P(\Lambda = \lambda) = 1$; but this may not be realistic, since loss rates of more than a few percent are very unlikely in practice. An alternative is to estimate the distribution of Λ from the sampling distribution of the measurements $\hat{\lambda}_{p,i}$ from all intervals. We evaluate both methods in Section 4.6.3.

Secondly, integrals of the form $\int_a^b f_L(n_{pkt}, \lambda, \hat{\lambda}) d\lambda$ are difficult to compute analytically due to the large exponents in f_L. We approximate them numerically by dividing the interval $[0, 1]$ taken by λ into a discrete set of numbers $S = \{0, 1/K, 2/K, \cdots, 1\}$, intersecting this set with $[a, b]$ and approximating:

$$\int_a^b f_L(n_{pkt}, \lambda, \hat{\lambda}) d\lambda \approx \sum_{k \in S \cap [a,b]} f_L(n_{pkt}, \frac{k}{K}, \hat{\lambda}) \tag{41}$$

For each number from S, we have to compute a binomial coefficient, which may be costly for large values of n_{pkt}, thus we want to reduce K as much as possible while still providing a reasonably accurate result. We choose K such that the number of values from S that are below the threshold t_c or the measured loss rate $\hat{\lambda}$, is at least 100; we found empirically that this offers a sufficient performance-accuracy trade-off for our data. Additionally, we take advantage of the fact that f_L is convex; we start the integration from the peak, and we iterate over S in both directions, stopping when the value is 200 times smaller than the peak. Overall, the optimized implementation generally incurs an error of less than 1% for our data, but is more than two orders of magnitude faster than integrating numerically at a fixed step. We found this optimization important: for the topology from Figure 3, having a few tens of paths and over 100 intervals, without optimizations, the accuracy computation alone for a single experiment would take several hours; optimized, it runs in under 1 minute.

4.6.2. Example

We provide intuition on how this technique helps with an example: Consider a link that subjects all traffic from a given class to packet loss 2.5%. Any two paths carrying traffic from this class experience highly correlated congestion, and capturing this correlation is key to reasoning about the link's neutrality. The blue (lowest) curve in Figure 12 shows the correlation of the performance numbers of two such paths, as inferred by *Straw*, as a function of the congestion threshold t_c: as t_c

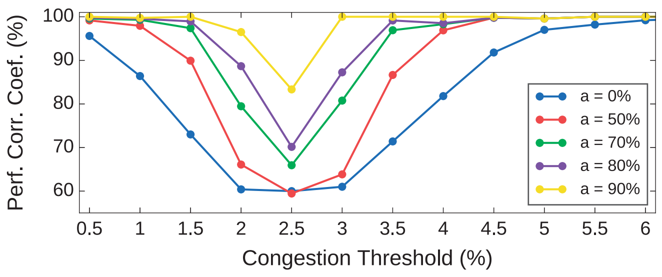


Figure 12. Same-class performance correlation.

approaches the true loss rate of the paths, correlation decreases (which is bad for neutrality inference), because the estimated loss rates of the two paths increasingly fall on opposite sides of t_c. The other curves show the same information, but after we have performed measurement filtering, each curve corresponding to a different reliability threshold a. If we choose $a = 90\%$, our filtered measurements perfectly capture the performance correlation of the two paths, except in the unlucky case where the congestion threshold falls exactly on the actual path loss rate of 2.5%.

In short, with measurement filtering we avoid the measurements for which the congestion states might be estimated incorrectly, which ensures that the systems of equations that link the path congestion states to the link congestion states are correct, thus we can apply the inference algorithm.

4.6.3. Evaluation Through Simulation

To evaluate the measurement filtering algorithm, we performed simulations in the following way: Consider a topology that consists of a single path, which in turn consists of a single link[6]. During an experiment, suppose we send on the path a fixed number of packets n_{pkt} in each of the n_{int} measurement intervals. Suppose the loss rate of the link is governed by the random variable Λ: in each interval, we sample this variable to obtain the ground truth of the loss rate λ_i. We simulate packet losses using a Bernoulli process with loss probability λ_i, whose outcomes we use to compute the loss rate estimate $\hat{\lambda}_i$. The estimated congestion state of the path is then computed for several values of the congestion threshold. At the same time, we use the true value of the loss rate to compute the true congestion state of the path as a reference: we compare the two states to determine the fraction of the intervals for which they are consistent. Our primary goal is to confirm that measurement filtering raises this fraction closer to 1. A secondary goal is to determine whether estimating the prior distribution of the loss rate from the data is sufficiently accurate; as we have seen from Equation (40), some way of estimating the prior distribution (which includes even assuming a uniform distribution) is necessary.

We use the following parameters:

- $n_{pkt} = 1,000$: this corresponds to an interval size of just under 1 minute and a packet sampling rate of 1 every 50 ms;

[6]While not realistic, this is the simplest possible topology that demonstrates the measurement filtering problem.

- $n_{int} = 60$: i.e. 1-hour experiments;
- Λ follows one of the distributions:
    - $Constant(0.5\%)$: lightly congested link;
    - $Constant(2\%)$: heavily congested link;
    - $Pareto(mean = 0.5\%, scale = 2, shifted\text{-}to\text{-}zero)$: lightly congested link, long-tailed distribution;
    - $Pareto(mean = 2\%, scale = 2, shifted\text{-}to\text{-}zero)$: heavily congested link, long-tailed distribution.
- The measurement filtering algorithm is one of the following:
    - Not performed;
    - Performed assuming a uniform prior distribution of Λ;
    - Performed estimating the prior distribution of Λ from the samples $\lambda_i, i = 1..n_{int}$;
    - Performed using the true prior distribution of Λ (impossible in practice, but included for comparison).
- The minimum accepted value of the reliability metric, used to filter the measurements, is set to 0.9.

For each scenario (i.e. choice of Λ), we plot the fraction of intervals for which the measured and the true congestion states are consistent as curves: each curve corresponds to a filtering method, with the congestion threshold t_c varying on the X-axis. We also plot the estimated loss rate distribution (obtained from the n_{int} measured loss rate samples), the true distribution of the measured loss rate (which we obtain by simulating $1,000,000$ intervals) and the likelihood function of the measured loss rate (i.e. f_L from Equation (37)).

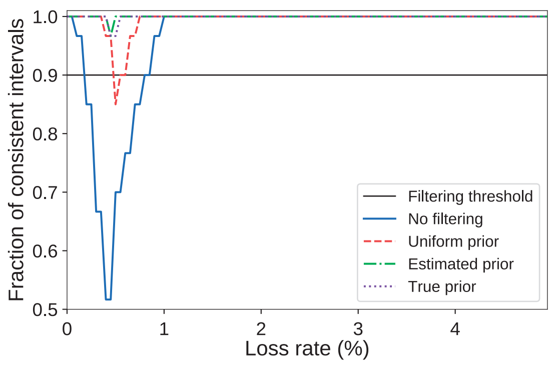
Figures 13 and 14 show the results for the scenarios using constant loss rate. We can see that without filtering, the fraction of consistent intervals drops to half when the loss rate threshold coincides with the peak of the measured loss rate distribution. Filtering with an uniform prior produces results that are almost good enough to satisfy our minimum reliability threshold; while filtering with the estimated prior produces excellent results, which are almost identical to filtering with the true prior.

Figures 15 and 16 show the results for the scenarios using long-tailed loss rate. In this case, the fraction of consistent intervals is quite high even without filtering; filtering improves it slightly, and the choice of loss rate prior does not seem to matter. The reason why the latter happens is the following: when computing the reliability metric, the integrand from Equation (40) is weighted by the likelihood function f_L; when the measured loss rate p.d.f. and f_L have different shapes[7], the most common values of the measured loss rate are assigned only a small weight; the values that matter most in the integration are the ones from the tail of the measured loss rate, where it has an almost uniform distribution—thus an uniform prior works

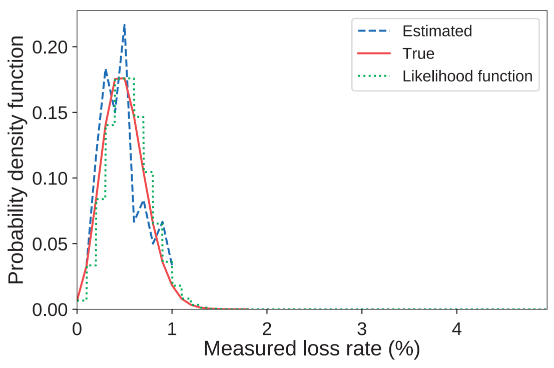
[7] f_L has the shape of the binomial distribution. When Λ has a small variance, the true loss rate in every interval is about the same, so the measured loss rate will also follow a distribution similar to the binomial. But when Λ has high variance, or in this case, a tail, the measured loss rate p.d.f. will have a higher "spread" than f_L, and possibly a different peak.

about as well as the estimated one. But when the two curves match, as we have seen in Figures 13 and 14, the prior becomes important. In practice, we cannot know whether the curves are similar or not, therefore it is safer to always use the estimated prior.

From these simulations, we conclude that the measurement filtering, using a loss rate prior estimated from the measured data, works as expected and produces acceptable results.

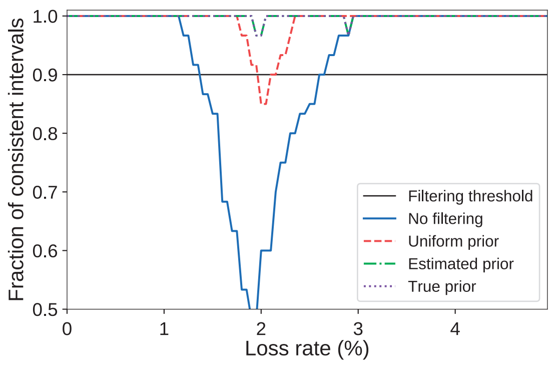


(13a) Fraction of consistent intervals.

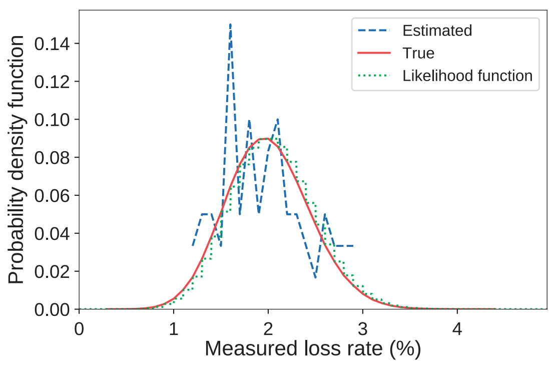


(13b) Measured loss rate.

Figure 13. $\Lambda \sim \mathit{Constant}(0.5\%)$.



(14a) Fraction of consistent intervals.



(14b) Measured loss rate.

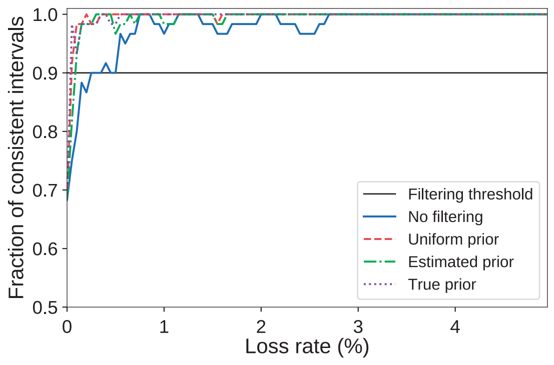
Figure 14. $\Lambda \sim \mathit{Constant}(2\%)$.

4.7. Evaluation

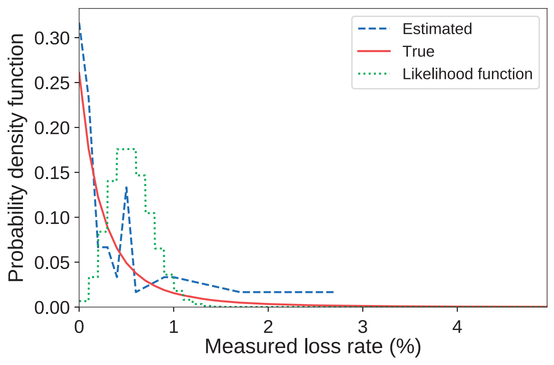
4.7.1. Emulations

We now present the final results from the emulation experiment performed on the RedIris topology, for which we discussed a few examples in Section 4.2.

Table 4 shows the results obtained when running the unmodified *Straw* algorithm. While its detection rate is good, the false positive rate is very high: even for

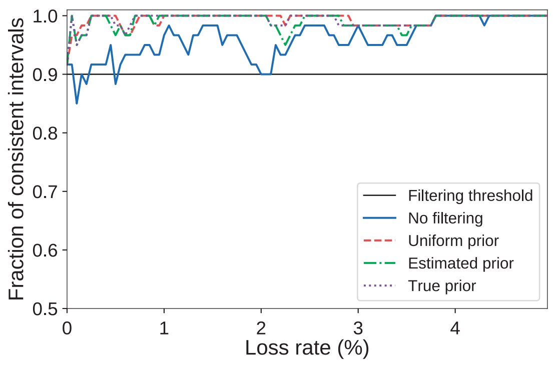


(15a) Fraction of consistent intervals.

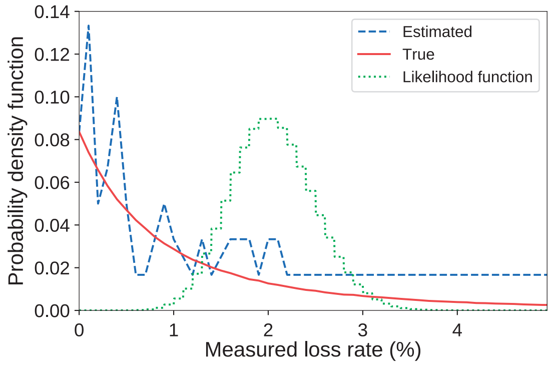


(15b) Measured loss rate.

Figure 15. $\Lambda \sim Pareto(mean = 0.5\%, scale = 2, shifted\text{-}to\text{-}zero)$.



(16a) Fraction of consistent intervals.



(16b) Measured loss rate.

Figure 16. $\Lambda \sim Pareto(mean = 2\%, scale = 2, shifted\text{-}to\text{-}zero)$.

a large value of the neutrality threshold, one third of the non-neutral links it detects are actually neutral.

Table 5 shows the results obtained when running a modified version of the *Straw* algorithm, which filters out measurements that do not satisfy the reliability metric. While this helps slightly for neutrality threshold 0.2, the rest of the results are not much different from *Straw*'s.

Finally, Table 6 shows the results obtained for the Bayesian algorithm (measurement filtering included). For a significance level of 0.9, the true positive rate is 77%, i.e. as good as *Straw*'s, while the false positive rate is only 16%, slightly above our acceptable margin of error of 10%—a result that fulfills expectations.

When increasing the significance level to 0.95, non-neutrality is no longer detectable due to the too-small number of intervals. While this seems a poor result, it is actually normal: from the given data, it is impossible to detect non-neutrality for very high values of the significance level. In fact, an algorithm that always detects non-neutrality for arbitrary values of the significance level is probably incorrect and unreliable.

The burden of choosing the desired significance level rests on the users: only they can decide the appropriate trade-off between acceptable false positive rate and detection rate. In this experiment, a threshold of 90% offers good detection rate, but the users should be aware that around a tenth of the links may be misdetected as non-neutral. The advantage over *Straw* is twofold: (i) the neutrality threshold has a clear meaning, with simple and well-understood effects on the detection metrics; and (ii) false positive rates are much lower.

Neutrality threshold (t_n)	True positives	False positives
0.2	96 %	77 %
0.3	85 %	54 %
0.4	78 %	42 %

Table 4. Inference results for algorithm *Straw*.

Neutrality threshold (t_n)	True positives	False positives
0.2	85 %	54 %
0.3	77 %	53 %
0.4	77 %	46 %

Table 5. Inference results for algorithm *Straw + measurement filtering.*

Neutrality threshold (t_n)	True positives	False positives
0.9	77 %	16 %
0.95	2 %	4 %

Table 6. Inference results for the Bayesian algorithm.

4.7.2. Detecting Amazon SMTP Throttling

To test whether our method functions correctly on a real Internet setup where the ground truth is known, we leverage the fact that cloud providers like Amazon throttle outgoing SMTP connections, in an attempt to reduce the amount of spam e-mail sent from compromised clients. This is an excellent opportunity for evaluating our method in the wild, because the fact that certain traffic is throttled is documented, although the throttling happens outside our control.

Clarification: Unlike other kinds of throttling, SMTP connection throttling is neither controverisal nor denied. The point of this section is not to reveal that cloud providers engage in this behavior (which is already known), but to assess we can detect and localize realistic throttling correctly.

Ground truth: Amazon throttles outgoing SMTP connections [Amazon.com, 2016]. To the best of our knowledge, neither the rate of the throttling nor its physical location within the Amazon network are publicly disclosed.

Topology: Figure 17 shows a small piece of the network topology covered by our experiments: one node hosting two sources, located in the Amazon cloud, and two PlanetLab [Chun et al., 2003] nodes, each one hosting two destinations; all traffic from the Amazon node to the same PlanetLab (PL) node follows the same path and traverses 3–4 Autonomous Systems (ASes). Our topology consists of many interconnected such pieces: the same Amazon node sends traffic to multile PL node pairs, and the same PL node receives traffic from multiple Amazon nodes. In total, we use 10 Amazon nodes, located in datacenters in Frankfurt, London, or Dublin, and 74 PL nodes, chosen because they are supposed not to be installed behind firewalls or intrusion detection systems [The PlanetLab Consortium, n.d.], which could potentially block or throttle our traffic.

Traffic classes: We define two classes of measurement traffic: class 1 consists of traffic that we have no reason to believe is throttled (details soon to follow); class 2 consists of SMTP connection setup traffic (TCP SYN packets with destination port 25), which we know is throttled. Initially, we defined class 1 as HTTP connection setup traffic (TCP SYN packets with destination port 80). However, we noticed that HTTP and SMTP traffic sent by the same source took different paths through the cloud network—not surprising, given that Amazon is known to perform load-balancing by 5-tuple (which includes the transport protocol). This prevented us from creating a topology like the one in Figure 2, i.e., with a shared link where we could localize the neutrality violation. Thus, we redefined class 1 as TCP non-SYN packets with destination port 25, which led to our class-1 and class-2 traffic having the same 5-tuple, hence follow the same path through the cloud network.

Experiments: Each experiment lasts 2 hours, divided into 15-second intervals and involves 10 sources. Each source sends traffic from one class to 2 destinations, at a rate of 4 requests per minute per destination; all this serves as measurement traffic. Moreover, each source sends background traffic to the same destinations, which is like class-1 and class-2 traffic but does not contribute to the measurements. Throughout the experiment, each source also performs periodic traceroutes to the same destinations, to ensure that the path to each destination is stable.

Inference example: Figure 18a shows an example of inferred distributions for two traffic classes of which one is known to be throttled; the computed non-neutrality bias is 95.7%, thus the link sequence is correctly detected as non-neutral. Figure 18b shows another example in which both traffic classes are expected not to be throttled. In this case the non-neutrality bias is only 61.8%, and the link sequence is correctly detected as neutral with respect to these two traffic classes. In both cases, the number of valid intervals (during which routes were stable) was $n_{int} = 42$, i.e. a duration of 21 minutes.

Throttling link sequences: After discarding unstable paths, we are left with 282 instances of the scenario in Figure 17, where a link sequence located inside the Amazon cloud is traversed by 2 path pairs, each one carrying traffic from a different class. In 129 out of these 282 instances, we correctly conclude that the target link sequence differentiates against class 2 with confidence 95%; in the remaining instances, our measurements are not accurate enough to draw this conclusion with the same confidence. The fact that not all instances lead to a non-neutrality conclusion is not surprising: to avoid blacklisting, our sources send traffic at a rate of

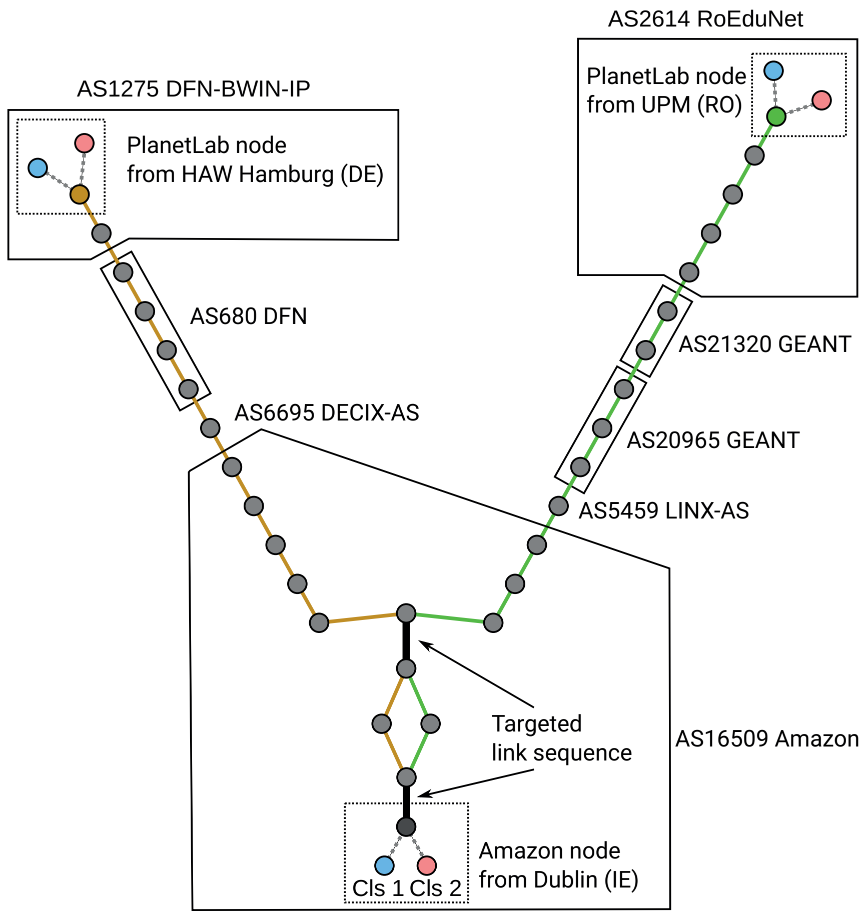


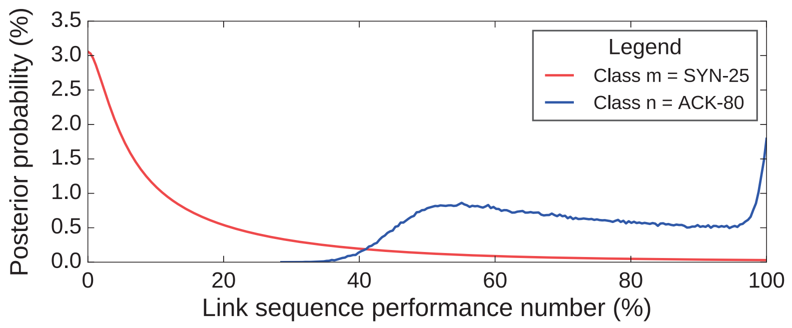
Figure 17. A small piece of our topology.

4 requests per minute per destination; this low rate combined with small numbers of time intervals for which routes are stable often causes measurement data that is not sufficient to support conclusive inference.

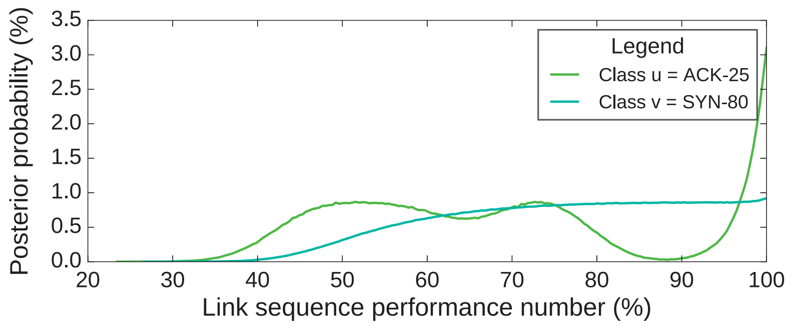
In Figure 19 we summarize the results from 10 different experiments. If a link sequence detected as non-neutral contains links from the same AS number as our provider known to throttle, it is counted as a true positive; otherwise it is a false positive. We observed that for link sequences covered by stable paths in up to 200 intervals (100 minutes), we detect non-neutrality in about a third of them; overall the detection has a $129/282 = 45.7\%$ true positive rate.

Non-throttling link sequences: After discarding unstable paths, we are left with 538 instances where a link sequence located *outside* the Amazon cloud is traversed by 2 path pairs, each one carrying traffic from a different class. We have no ground truth about these link sequences but expected most of them to not differentiate, as networks hosting PL nodes are not supposed to perform any throttling. Indeed, in only 3 out of the 538 instances we conclude (with confidence 95%) that the target link sequence differentiates against class 2.

Our algorithm has also detected 7 other anomalies, that we initially thought were false positives. We hand-checked each of these, and concluded that they were caused by a small subset of PlanetLab nodes located in networks that block TCP SYN packets sent to port 25. In fact, these were true positives.



(18a) Non-neutral link sequence.



(18b) Neutral link sequence.

Figure 18. Inferred posterior distribution of the performance number for two link sequences.

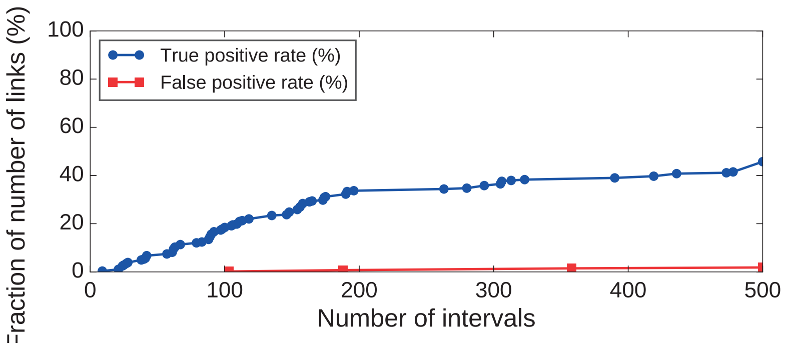


Figure 19. The true positive and false positive rate as a function of the number of intervals used.

Granularity of localization: All the Amazon link sequences that we confidently characterized as throttling consisted of a few links connecting the source to the cloud network, i.e., we found that Amazon throttles SMTP connections very close to their source. The 3 non-Amazon link sequences that we confidently characterized as throttling spanned from 1 to 3 ASes. We should clarify that existing neutrality methods that rely solely on throughput differences could have detected that a given end-to-end path throttles SMTP connections, but not localized the throttling to specific link sequences.

5. Studies

5.1. Speed Test Prioritization

We used our method to investigate whether ISPs perform speed-test[8] prioritization, which has been confirmed to happen on at least one occasion [Byrne, 2016].

5.1.1. Background

On one confirmed occasion, an ISP exploited the fact that a popular speed-test protocol uses TCP port 8080[9] to exchange test traffic, whereas standard web traffic typically uses port 80. The ISP configured its network to prioritize all traffic with source port 8080, i.e., all traffic going from a speed-test server to an ISP client testing its download throughput. We have also found a guide for system administrators to perform exactly this kind of prioritization [Jahanzaib, 2015], which suggests that such behavior may be common.

Unfortunately, it is not known exactly why the ISP had decided to implement this kind of traffic management, but we can discuss the situation hypothetically. At a first look, prioritizing speed-test traffic between clients and a test server located in their proximity seems strange: the very reason many ISPs choose to host speed-test servers is that this causes the measurement results of their clients to reach excellent values, i.e. high throughput and low latency. This is because most speed-testing applications select the closest server to the client; thus if the server is located in the same ISP, the round-trip time is usually low, allowing TCP to ramp up the transmission rate quickly; also, the short paths benefit from high capacity. In fact, this is why speed-test results are often better than the actual experience of the user, since regular traffic exchanged with the Internet usually crosses multiple ISPs over longer paths, thus incurs higher latency and lower throughput[10]. Yet this ISP has decided to prioritize speed-test traffic, enhancing the results. This suggests that without prioritization, the results might have been unsatisfactory; most likely, the ISP was suffering from capacity issues, thus users could not achieve the expected throughput. Prioritizing speed-test traffic may thus reduce the amount of complaints received from the clients, and also avoid bad publicity, since test results are often shared between users; both are incentives for ISPs to implement it.

5.1.2. Stage 1: Suspect Selection

We identified approximately 6000 speed-test nodes, deployed in various ISPs around the world, with the following property: each one hosts both a speed-test web server

[8]While the correct term is “throughput test”, most of the tools use the name “speed-test”.

[9]This is due to a technicality: when the same machine runs both a standard web server and a speed-test web server, it is practical to isolate the two services through different port numbers.

[10]While searching for documented cases of speed-test prioritization, we found many reports from users who did not understand this inherent bias, thus were falsely accusing their ISP of cheating the tests. Confirming a case of prioritization currently requires either the manual intervention of an expert who can interpret the data correctly, or using specialized tools to collect and analyze the measurements automatically. We think that untrained users are unlikely to reach a correct verdict when looking only at speed-test data and trying to interpet it themselves.

listening on port 8080, and a speed-test web server running an older protocol on port 80. The nodes are evenly distributed geographically across all continents, with the exception of Africa and Oceania, as shown in Table 7.

Continent	Node count	Fraction (%)
Europe	2070	33.5
North America	1271	20.5
South America	1263	20.4
Asia	1191	19.3
Africa	255	4.1
Oceania	135	2.2

Table 7. Geographical distribution of the speed-test nodes we used.

Since our measurement pipeline requires experiments that last for about 30 minutes or more in order to detect non-neutrality, testing this many speed-test nodes is not practical: the entire probing would take several months; it would consume a large amount of bandwidth; and it may put pressure on the speed-test infrastructure, possibly affecting other user's results.

Therefore we have decided to probe the nodes in two stages: In the first stage, we only conducted short throughput measurement experiments, to identify quickly "suspect cases," where traffic with source port 8080 achieves significantly higher throughput than traffic with source port 80. Then in the second stage, we ran the longer experiments of our pipeline. We now focus on the first stage.

We used a single probing node to measure throughput to and from each of the 6000 speed-test nodes. The probing node was located on our university campus and was connected to the Internet via a 1 Gbps link, fast enough to ensure that our campus network was not the bottleneck—necessary since our goal is to detect throttling in other ISPs. The speed-test nodes were probed sequentially; in addition, each of the two protocols supported by the node were probed in sequence, with throughput measured during a 60-second transfer. For each node and protocol, the measurement was repeated 10 times, with the different protocol measurements alternated; this was done in order to improve measurement accuracy in case of network conditions variable on short-term, such as a third party starting a test with the same speed-test node we were probing. This design is similar to the method used by the Glasnost project to measure throughput reliably when two hosts communicate over different protocols [Marcel Dischinger et al., 2010].

As an optimization that shortened the overall probing time, and also avoided stressing the speed-test nodes, we stopped probing a node if port 80 traffic achieved over 100 Mbps during the first 10 seconds of the transfer at the earliest. We chose this specific threshold because it is higher than the download throughput achieved by 95% of residential Internet users, statistic we extracted from the dataset published by M-Lab's NDT project [Measurement Lab, 2017] using a method presented in Appendix 8.1. About 80% of the nodes achieved a high throughput and were eliminated. For the 1328 remaining slower nodes, we present the histogram of the throughput per transfer grouped by port in Figure 20. Both distributions appear bimodal, with a common tallest peak in the range 0-5 Mbps; the second tallest peaks are different, showing a bias towards higher throughput for port 8080 traffic. It is useful to also compare the throughput values achieved on the two port numbers for each node, which is shown in Figure 21: each point in the scatter plot represents a node, with the X and Y coordinates showing the port 80 and 8080 throughput, respectively. We draw bands that show the magnitude of the relative difference, for example for points that fall inside the green band, of relative difference up to 10%, we have

$$\frac{\max(tput_{80}, tput_{8080})}{\min(tput_{80}, tput_{8080})} < \frac{110}{100} \tag{42}$$

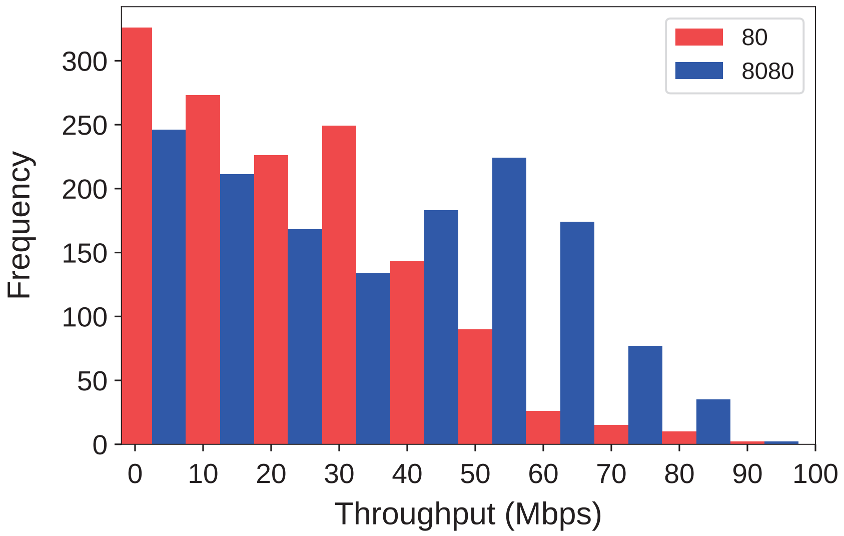


Figure 20. Histogram of the average throughput per transfer, grouped by port number.

From the scatter plot, we can conclude the following:

- Most of the points are located above the main diagonal, meaning that throughput on port 8080 is generally higher than on port 80, which confirms the bias we have seen in the histograms;
- There are many nodes for which the throughput ratio is above 1.5;
- There are a few nodes with very high throughput ratios, of 8:1 and more;
- For nodes with small throughput on both ports, say under 15 Mbps, the relative difference is less meaningful, as most are scattered outside the bands both above and below, because the bands are too narrow. It is not clear if there is indeed a bias for these or it is just noise.

The fact that a fraction of nodes do not show a bias; some show a bias, but not very strong; and there are some nodes that show a very strong bias, suggests that there may be multiple mechanisms at play that lead to different amounts of bias. But not all are of interest to us. To understand why, consider four examples of nodes that on

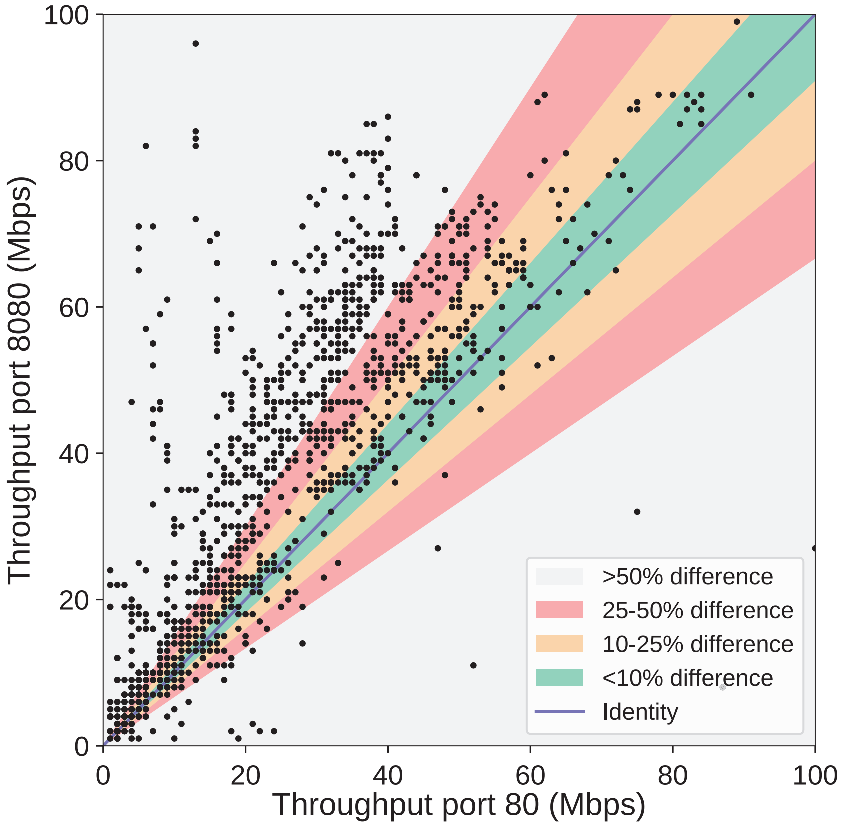
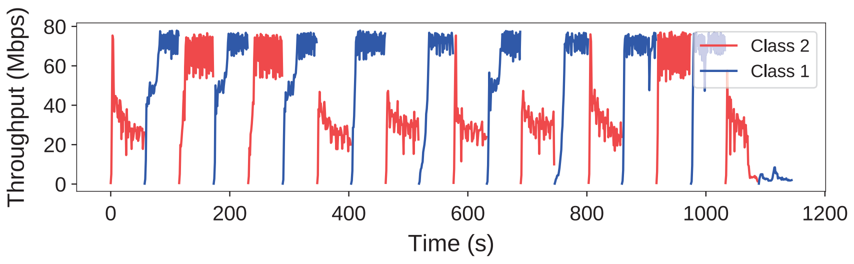


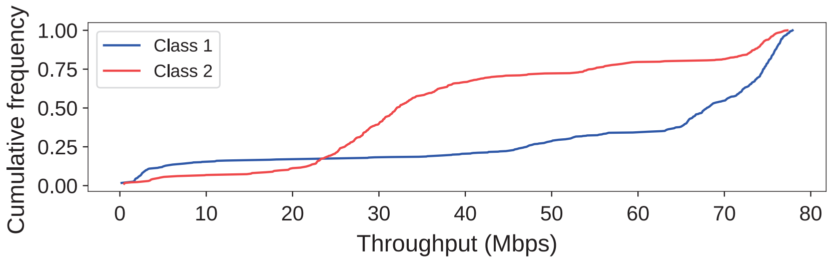
Figure 21. Scatter plot of the median throughput per server and port number.

a first look show a bias: Figures 22, 23, 24 and 25 present the measured throughput over time and the corresponding empirical CDFs (for throughput sampled once per second during the transfer). While the CDFs suggest different distributions, interpreting the timeline plots reveals a different story: Figure 22a shows that during 3 out of 10 transfers, port 80 (class 2) traffic has achieved the same throughput as port 8080 (class 1), thus the difference may be caused by changing network conditions. Figure 23a shows a similar situation, where the change in network conditions is higher in magnitude and more consistent over time. Figure 24a shows that both types of traffic achieve the same maximum throughput, but throughput for port 80 traffic has a much higher variance, which causes the difference in shape of the CDFs. Figure 25a shows another case of inconsistent differences, which may be caused by a combination of network conditions and higher variance of the throughput for both types of traffic.

The two mechanisms we have seen that lead to a bias, but are not caused by non-neutrality in the network are: (i) changing network conditions between measurements, and (ii) different variance in throughput measurements. The former is clearly not under our control, thus the best we can do is identify it and filter it out. The latter is more difficult to reason about. The difference in variance appears to be caused by different traffic patterns for the two protocols, which in turn originate in the way the two protocols have been designed: the protocol sending traffic on port 8080 sends pseudo-random data from a small circular memory buffer; the other

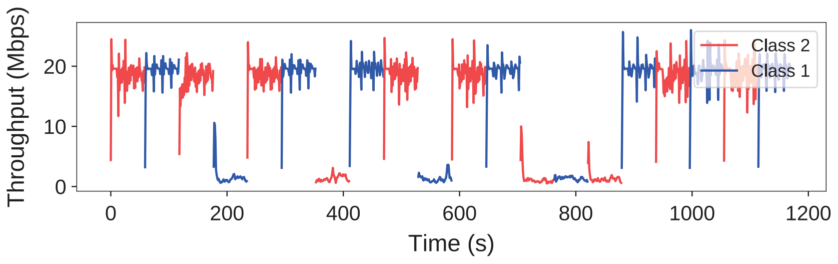


(22a) Timeline.

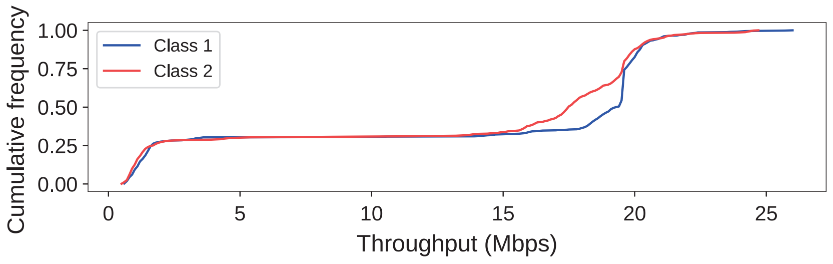


(22b) Empirical cumulative distribution function.

Figure 22. A node for which the differences in throughput are inconsistent. We can see in the timeline that class 2 traffic sometimes achieves the same throughput as class 1.



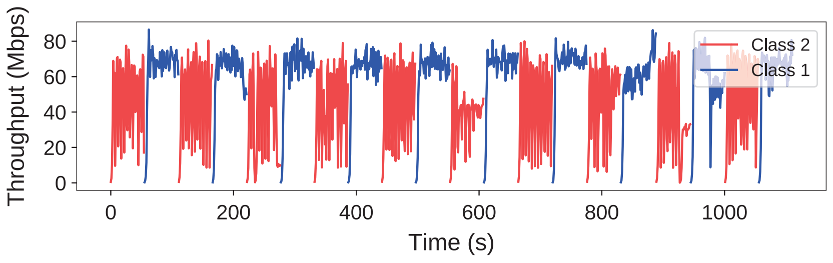
(23a) Timeline.



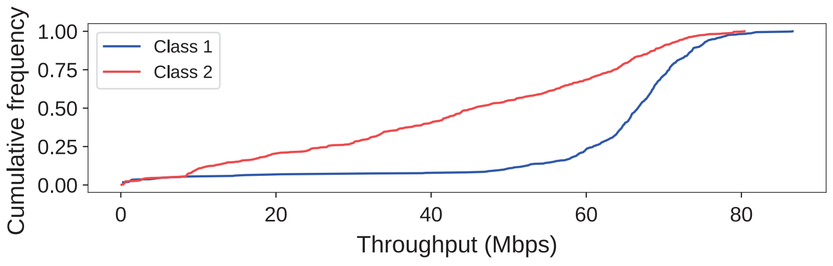
(23b) Empirical cumulative distribution function.

Figure 23. A node for which the apparent differences in throughput in the CDFs are likely caused by changing network conditions.

protocol transfers a static file repeatedly over HTTP. The messaging pattern is the same: the measurements use a single TCP connection with requests sent repeatedly back-to-back. The only difference that remains is that one application has to per-

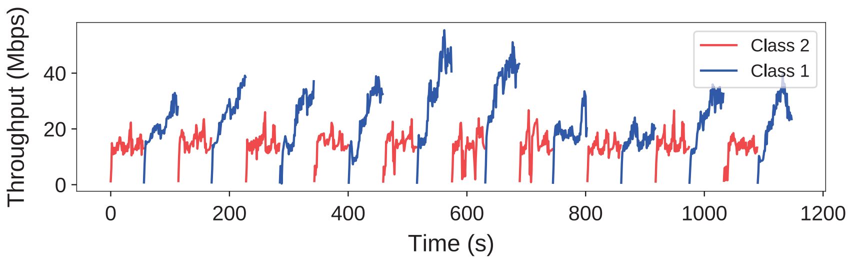


(24a) Timeline.

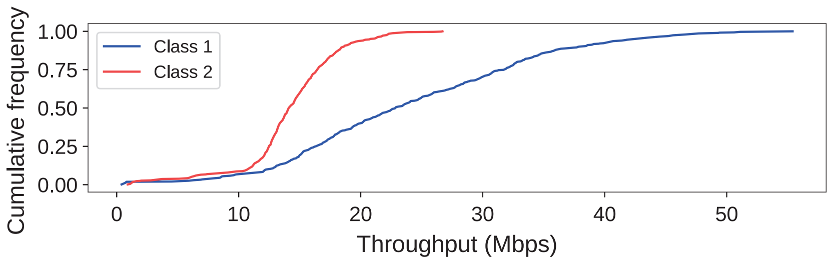


(24b) Empirical cumulative distribution function.

Figure 24. A node for which the apparent differences in throughput in the CDFs are caused by high variance experienced by one of the protocols.



(25a) Timeline.



(25b) Empirical cumulative distribution function.

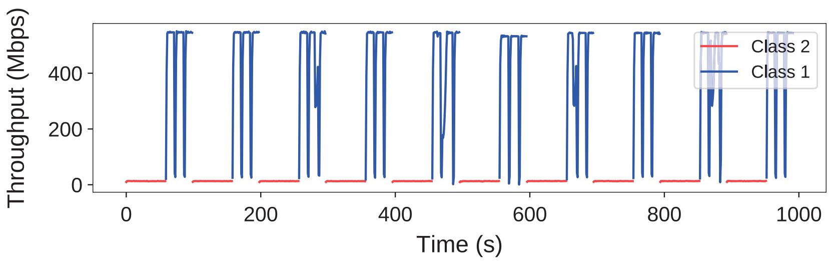
Figure 25. Another node for which the differences in throughput are inconsistent.

form a couple of disk seeks and some extra processing every few seconds to serve the file, while the other one incurs no such overheads as it works only in memory. This may be sufficient to lead to different traffic patterns and the higher variance and slightly lower average throughput measured on port 80 that we have seen in many experiments, even in a neutral network. Unfortunately this difference between applications takes place server-side, on nodes that we do not control, so the best we can do is filter out these effects.

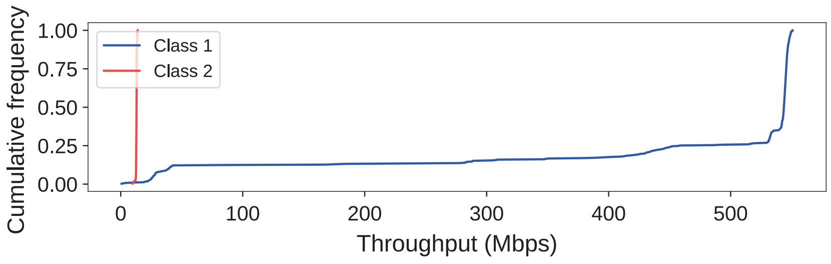
The algorithm we used to identify suspect nodes is the following:

1. For each node, we pair together the transfers that occured in sequence on different protocols into 10 pairs. Each pair contains 60 throughput samples for each of the two protocols. We compare the samples pairwise using the Kolmogorov-Smirnov statistical test to determine if they belong to the same distribution. The result is a set of 10 boolean values, one for each pair.
2. If all 10 results are identical, we add the node to the list of suspects. This step rules out primarily measurements affected by changing network conditions.
3. If the difference in average throughput is higher than 3:1, we label the node as suspect with very high bias. This is a very likely candidate for testing with our Bayesian method.
4. If the difference in average throughput is higher than 3:2, we label the node as suspect with high bias. After a few iterations with different parameters, we have found empirically that this ratio is high enough to exclude most biases caused by differences in throughput variance.
5. Otherwise, we simply label the node as suspect with possible bias and we inspect the result manually.

Out of 1400 nodes, only 4 were labeled with very high bias. The most dramatic result is the one shown in Figure 26: port 80 traffic achieves a mere 12 Mbps, whereas port 8080 reaches 500 Mbps, i.e. the average throughput ratio is about 40:1. The second highest bias is shown Figure 27, with 70 vs. 8 Mbps.

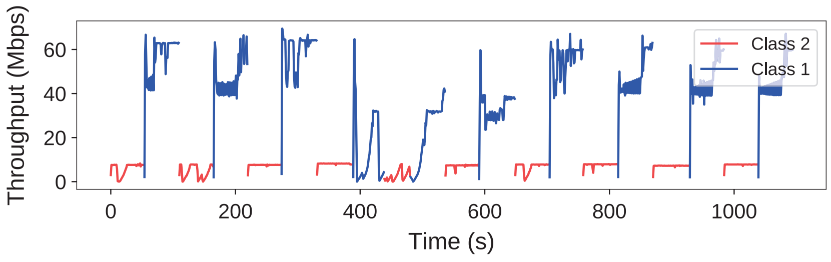


(26a) Timeline.

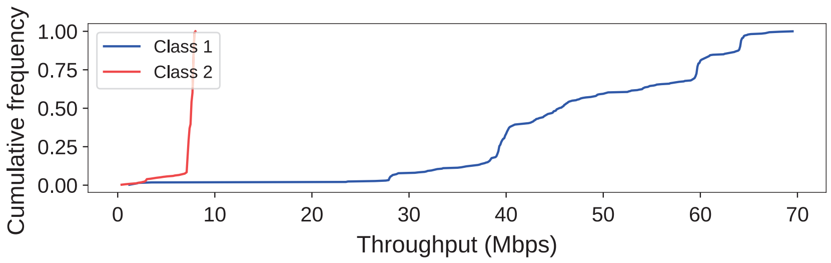


(26b) Empirical cumulative distribution function.

Figure 26. A node showing very high differences in throughput between the two protocols.



(27a) Timeline.



(27b) Empirical cumulative distribution function.

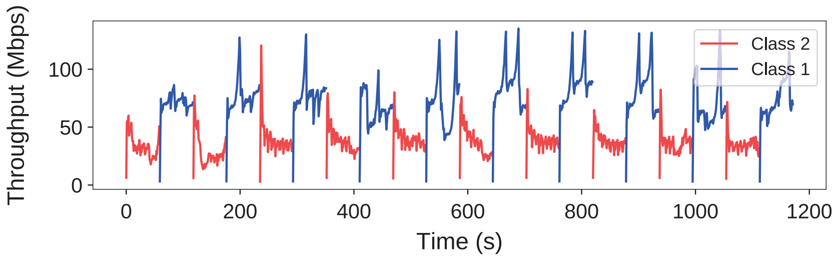
Figure 27. Another node showing very high differences in throughput between the two protocols.

We labeled another 15 nodes with high bias, such as the one shown in Figure 28, and 12 nodes with possible bias. From the latter, after manual inspection we have chosen only 2 nodes, for a total number of suspects of 21. In other words, less than 0.4% out of all 6000 existing nodes appeared to show a bias, according to our criteria. The suspect nodes do not appear clustered in any specific country or network: almost all belong to different ISPs and are located in regions with a variety of network conditions, such as U.S.A., Brazil, Austria, Russia etc.

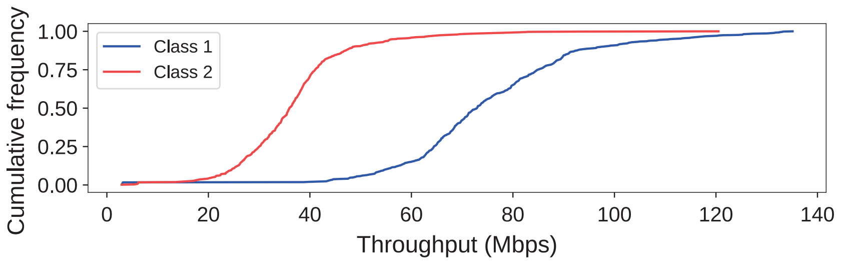
5.1.3. Stage 2: Long Experiments

In the second stage, we ran the longer experiments of our pipeline, targeting 42 speed-test nodes: the 21 suspect cases identified in the first stage (which we call "suspect"), plus another 21, chosen at random as control (which we call "control"). For each of them, we created 6 topologies like the one in Figure 29: the speed-test node (always hosting two web servers, listening at ports 8080 and 80) exchanges traffic with two probing nodes, each one hosting two clients. The speed-test clients run on 9 probing nodes under our control, rented from several cloud providers and each one located in a different datacenter from Europe or North America.
We defined two classes of measurement traffic: class 1 consists of traffic with source port 8080, which is suspected to be prioritized, while class 2 consists of traffic with source port 80. In all figures, plot elements corresponding to class 1 traffic are shown in dark blue or light blue, while those of class 2 traffic are shown in red or orange.

We conducted experiments lasting 1 hour, divided into 60 1-minute intervals, targeting either one suspect or one control speed-test node. In each experiment, each client initiates subsequent 25-second speed-tests with the server. The measure-



(28a) Timeline.



(28b) Empirical cumulative distribution function.

Figure 28. A node showing high differences in throughput between the two protocols.

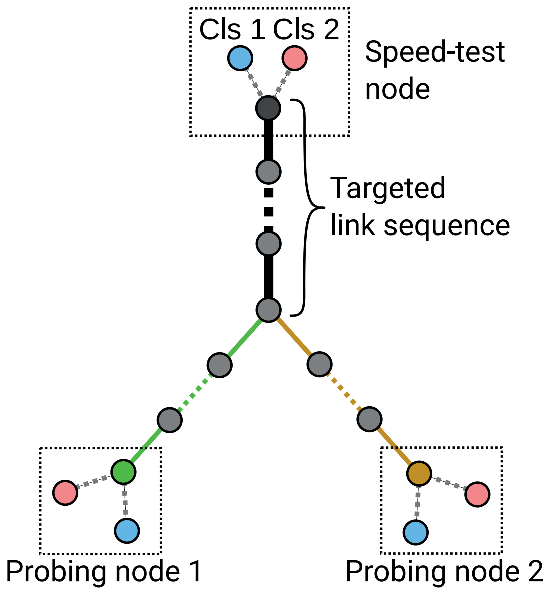


Figure 29. One of our topologies.

ment traffic consists of the traffic sent from the servers to the clients. Unlike the suspect selection experiments, traffic is now exchanged simultaneously on the two port numbers, instead of alternately.

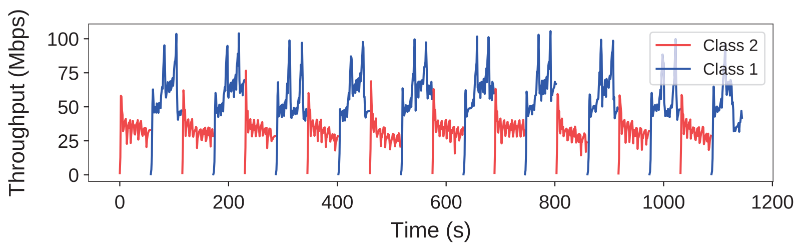
5.1.4. Results

In all 21 experiments concerning control nodes, as well as in 18 out of 21 experiments concerning suspect nodes, we found no evidence of differentiation: the inferred posterior distributions for the two traffic classes overlapped significantly, and the corresponding neutrality biases were small. In the 3 remaining experiments, there was inconsistent evidence of differentiation, i.e. the Bayesian method reported non-neutrality for some combinations of paths, but not all. Further manual analysis showed that although loss rate and throughput appeared to have different distributions for the two traffic classes, the most likely explanation was not differentiation in the network, but at the source: the nodes were probably performing self-throttling at the application layer. So, the method drew the right conclusion—technically, the target link sequence was non-neutral—but it was not an interesting conclusion, because it did not indicate controversial ISP behavior.

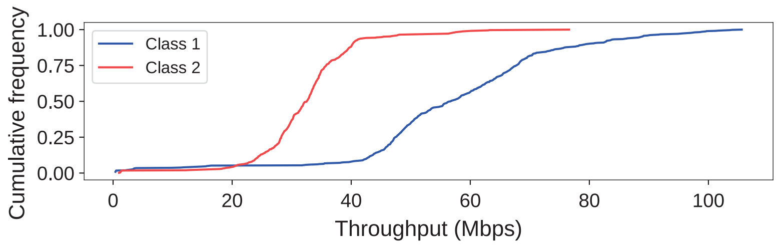
We now choose two experiments, one belonging to each category, to discuss in more detail.

5.1.4.1. Experiment Showing Verdict: Neutral

We provide a detailed analysis of a node labeled as having high bias in the suspect selection stage, yet shows no evidence of non-neutrality when investigated with our pipeline in stage 2. The result of the first stage is shown in Figure 30: there is a visible difference in throughput, with class 1 traffic achieving about twice more than class 2 on average.



(30a) Timeline.

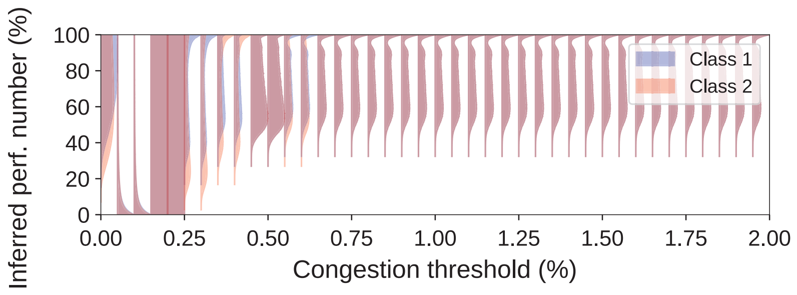


(30b) Empirical cumulative distribution function.

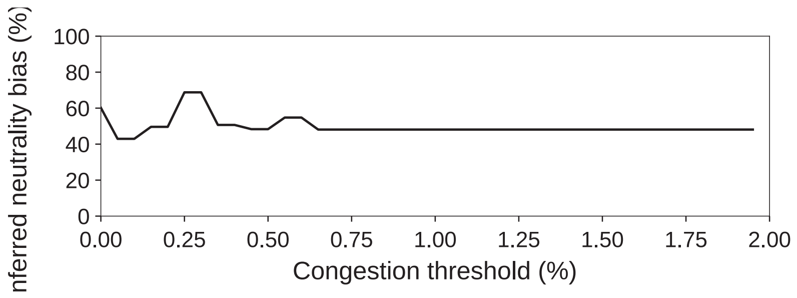
Figure 30. Result of the suspect selection stage.

The result of the inference performed in the second stage is shown in Figure 31. For the entire range of values of the congestion threshold, the posterior distributions overlap almost completely, thus the inferred neutrality bias is always under 70%, much lower than the significance level of 95% we require to reach a non-neutrality verdict.

Unfortunately, the ground truth of the shared link sequence is not known, thus we cannot verify precisely whether this result is correct or not. However, we can



(31a) Posterior distributions.



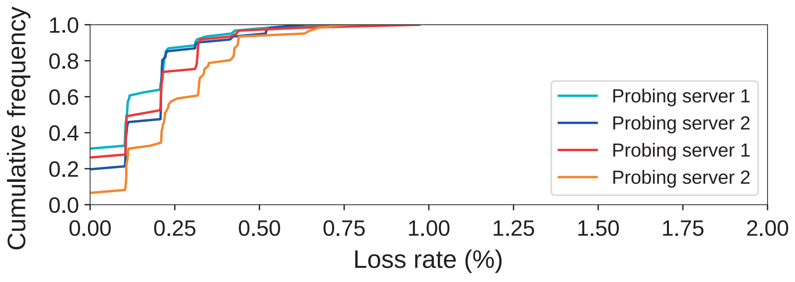
(31b) Inferred neutrality bias.

Figure 31. Result of the Bayesian inference performed in the second stage.

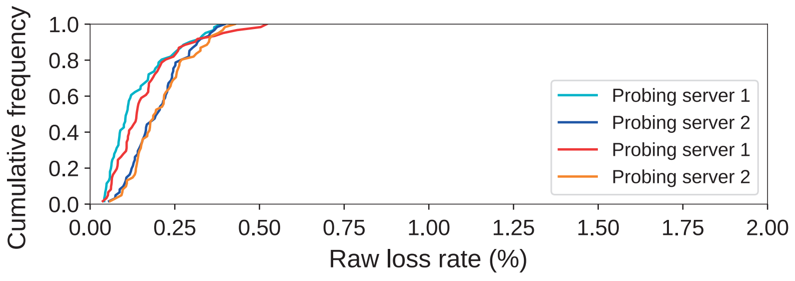
inspect the end-to-end data further, taking advantage of the fact that the two traffic types exchanged by each speed-test and probing node pair share the same path; thus we can compare the end-to-end loss rate and throughput. The data is shown in Figures 32 and 33. As we can see, the curves correspnding to class 2 data are very similar to the ones for class 1, for both loss rate and throughput, which means that there is no evidence of non-neutrality, confirming our result.

A question that arises is why the node was flagged during the suspect selection stage, when there is a visible difference in throughput between the two traffic classes. A clue to the answer is given by Figure 33a, which shows that the variance of the throughput in the long experiment is large: for example for the dark blue curve, the 10-th percentile is 10 Mbps, while the 90-th percentile is 35 Mbps, a 3.5x difference. This is most likely caused by changing network conditions: there is probably a common bottleneck on both paths (since the empirical distributions of the loss rate are very similar) which in addition to our measurement traffic, carries other background traffic we cannot observe. We can verify this by plotting the throughput over time for the four types of traffic in a stacked plot, shown in Figure 33b: the aggregate throughput varies over time, instead of reaching a constant limit (as it would if our traffic alone was saturating a bottleneck link). Due to the varying background traffic, the share of the bottleneck capacity taken by measurement traffic varies randomly over time. The data shows that when using a single TCP connection at a time (as in the suspect selection stage), class 1 traffic is able to get a higher share of the capacity than class 2; but when we use multiple parallel connections (as in the second stage), this effect is diminished and both types of traffic achieve similar throughput.

Without more information about what background traffic was exchanged on the

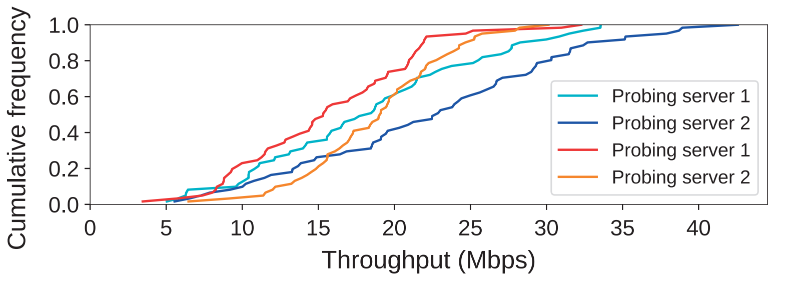


(32a) Empirical CDF of the loss rate.

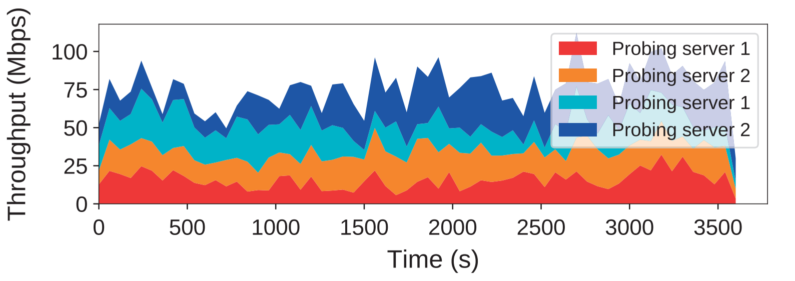


(32b) Empirical CDF of the raw loss rate (without packet sampling).

Figure 32. End-to-end loss rate measurements in the second stage.



(33a) Empirical CDF of the throughput.



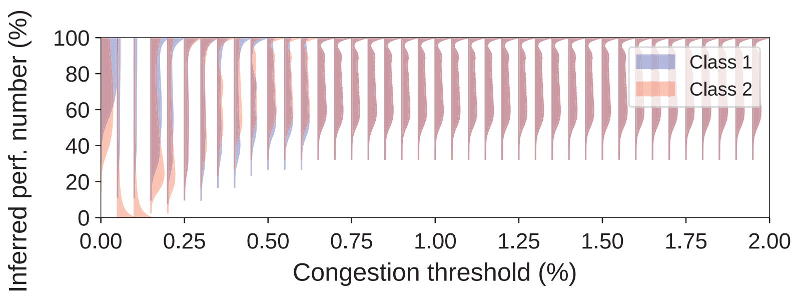
(33b) Timeline of the throughput, stacked plot.

Figure 33. End-to-end throughput measurements in the second stage.

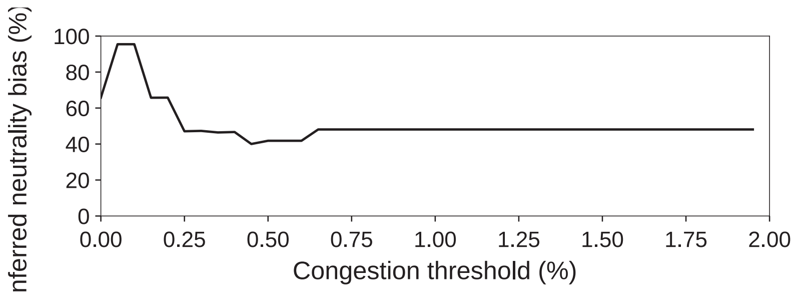
bottleneck link, and how the speed-test server is configured, we cannot give a good explanation for why this happens. A plausible explanation is that port-80 traffic achieves lower throughput because it is paced at the source per flow, not because it encounters more network congestion. As the number of connections increases, the aggregate throughput increases, causing congestion on the bottleneck, which lowers the throughput per flow below the limitation, its effect disappearing. A speed-test node may pace its outgoing port-80 traffic if all services running on the node are rate-limited except for the speed-test server listening at port 8080, such that other traffic does not interfere with speed-testing. We do not have sufficient evidence to conclude that this is indeed the case, but it is a possible explanation.

5.1.4.2. Experiment Showing Verdict: Non-Neutral

Among the 3 experiments for which we found evidence of differentiation, one is the experiment with the highest throughput bias we have seen earlier in the suspect selection stage in Figure 26. Figure 34 shows the results of Bayesian inference, Figure 35 shows the end-to-end loss rate measurements and Figure 36 shows the end-to-end throughput measurements. We can see that for congestion threshold 0.05% and 0.10%, the inference finds a significant bias of 95.4%, just above the 95% threhsold, thus the verdict is that the common link sequence is non-neutral. However, the threshold was exceeded only for some of the path pair combinations, others (not shown here) reaching a neutral verdict; thus we decided that the experiment required further investigation. We can see from the end-to-end loss rate data shown in Figure 35a that there indeed appears to be a small difference between the empirical CDFs for the two traffic classes of each node (i.e. light blue vs. orange, and dark blue vs. red). The difference is also present in Figure 35b, showing the raw loss rate (i.e. the actual loss rate experienced by the TCP flow, before sampling packets). Figures 36a and 36b too show a very clear difference in throughput.

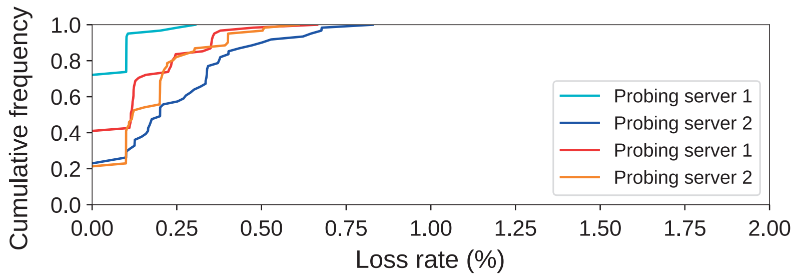


(34a) Posterior distributions.

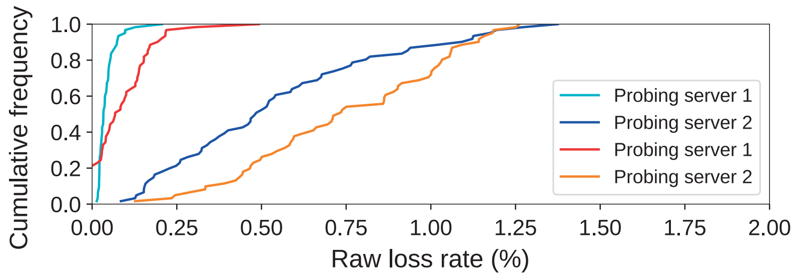


(34b) Inferred neutrality bias.

Figure 34. Result of the Bayesian inference for a topology detected as non-neutral.

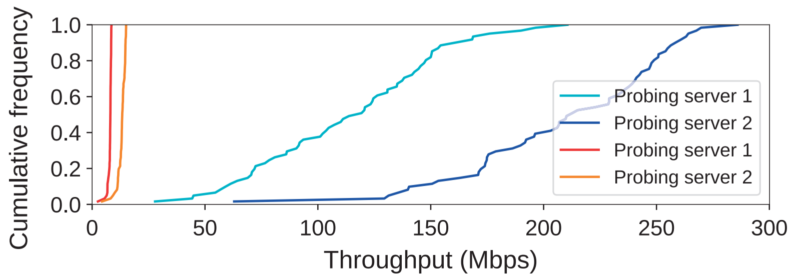


(35a) Empirical CDF of the loss rate.

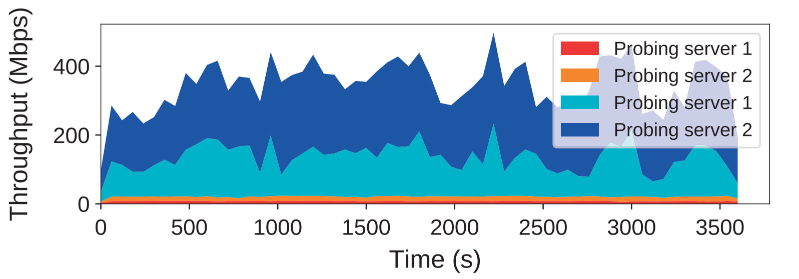


(35b) Empirical CDF of the raw loss rate (without packet sampling).

Figure 35. End-to-end loss rate measurements.



(36a) Empirical CDF of the throughput.



(36b) Timeline of the throughput, stacked plot.

Figure 36. End-to-end throughput measurements.

At a first look, all data supports the conclusion that there is indeed non-neutrality. However, we found three details suspicious: (i) the throughput timeline plot of class 2 traffic is very smooth; this would exclude traffic policing as throttling method (since in our experience, it usually causes a higher variance in throughput), which leaves only shaping and application-layer pacing as possible differentiation mechanisms; (ii) the inferred bias is high only for a small range of congestion thresholds; whereas in our lab experiments that introduce differentiation, there is usually a wider range of thresholds for which the bias is high (as it can be seen in Figure 11, for example) since the loss rate of congested flows tends to have high variance, and thus a “wider” CDF; (iii) differentiation was visible only for about half of the path pair combinations. In the light of these observations, we decided that the result is inconclusive, requiring further investigation to confirm the result and potentially to discover the throttling mechanism.

We performed 10-minutes, single-flow transfers on each port number from a single probing node and we captured the traffic. We inspected the packet capture and we found that when we exchanged traffic over port 80 with a single flow, there was no packet loss for long periods of time, while TCP throughput was almost perfectly constant (i.e. there was no sawtooth behavior). By contrast, port 8080 traffic was normal-looking TCP traffic, with variations in throughput over time and periodic retransmissions. This shows that throughput is likely limited at the application layer (source paced) by the web server running on port 80; while port 8080 traffic is limited by the network. This explains what we have seen in the experiment involving multiple paths: the non-throttled flows were filling all the available capacity of the bottleneck link, introducing a small amount of packet loss that was experienced by both types of flows. Due to the fact that packets were paced differently (port 8080 flows sending packets more than an order of magnitude more often than the others), there were small differences in measured packet loss between the two types of traffic; packet sampling reduced this effect, but did not eliminate it completely, which lead occasionally to a non-neutral verdict in the Bayesian inference. This suggests that the bottleneck link shared by the flows is in fact, neutral; but the link sequence we targeted—which includes the source node—is non-neutral, since the source is self-throttling port-80 traffic[11]. Similar results were observed for the other 2 speed-test nodes for which our tool returned a non-neutral verdict for some path pair combinations.

Although this result is not sensational, we still think it is interesting: if we consider only the throughput achieved by the two traffic classes, it is tempting to hypothesize that the network prioritizes port-8080 traffic. Our tool avoids this false-positive most of the time, because it infers the congestion behavior of the shared link sequence where prioritization would occur.

5.1.5. What-if Analysis

We still wanted to test whether our tool would catch actual speed-test prioritization, if it happened, so we implemented it ourselves: We modified all our topologies, such that both probing nodes were located in our campus. In each topology, we connected the probing nodes to a switch with QoS capabilities, itself connected to a router performing traffic classification, as shown in Figure 37. We configured the

[11]A speed-test node may be configured to self-throttle port-80 traffic when all services running on the node are rate-limited except for the speed-test server listening at port 8080, such that other traffic does not interfere with speed-testing.

router to set the DSCP field in the IP header of each forwarded packet to a value encoding whether the packet belongs to class 1 or class 2; we configured the switch to accept this label and police class 2 traffic at a maximum rate of 20Mbps, while allowing class 1 traffic to consume as much as the entire available capacity of the link (1Gbps). The exact configuration is shown in Appendix 8.2. Then we repeated the same 42 experiments as above.

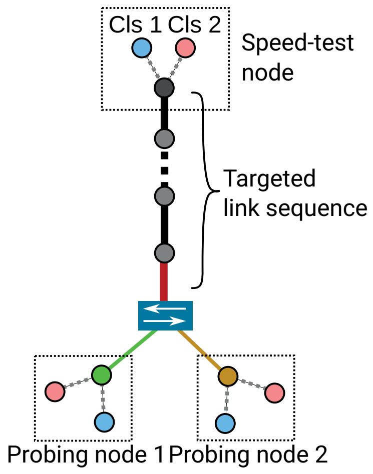
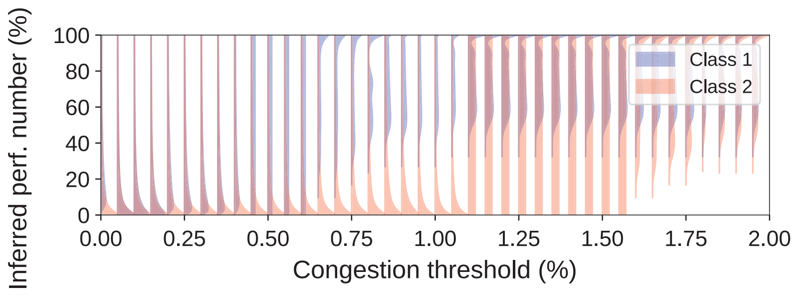


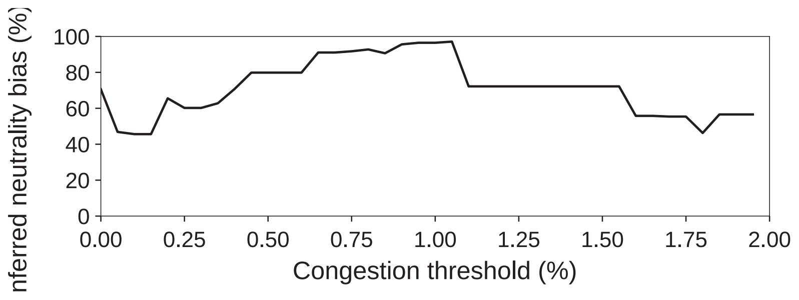
Figure 37. The topology where we introduce differentiation ourselves. The non-neutral link is drawn in red.

Our tool correctly concluded that there was differentiation in about one third of the experiments. In the remaining experiments, there was actually no differentiation, despite the presence of the policer: either the policed link was not bottlenecked, so policing did not kick in at all; or policing did kick in, but both traffic classes experienced significant congestion on other links, which reduced the impact of the policer.

As example, we present an experiment where the non-neutrality we introduced was detected. The result of the inference is shown in Figure 38: the posterior distributions are well separated for a range of values of the congestion threshold between 0.6% and 1.1%; the gap reaches a maximum of 97.1%, leading to a non-neutral verdict, as expected. Figures 39 and 40 show the end-to-end measurements: there is a clear difference in the distribution of both loss rate and throughput measurements between the two traffic classes. This confirms that our method can detect non-neutrality in experiments like the ones performed in this study.

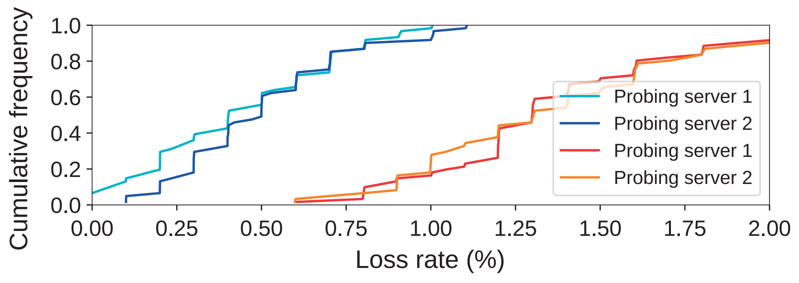


(38a) Posterior distributions.

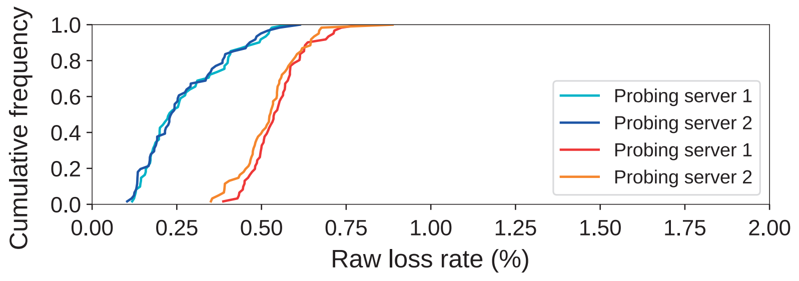


(38b) Inferred neutrality bias.

Figure 38. Result of the Bayesian inference for a non-neutral topology.

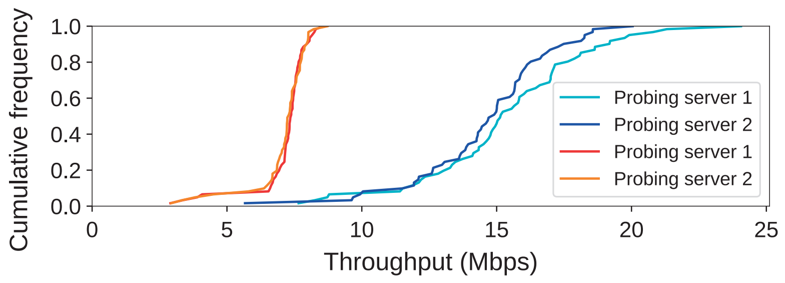


(39a) Empirical CDF of the loss rate.

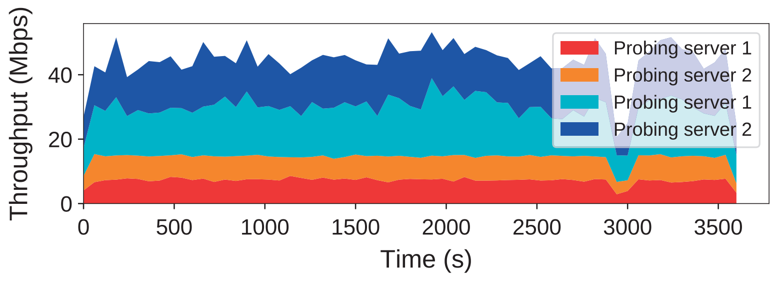


(39b) Empirical CDF of the raw loss rate (without packet sampling).

Figure 39. End-to-end loss rate measurements.



(40a) Empirical CDF of the throughput.



(40b) Timeline of the throughput, stacked plot.

Figure 40. End-to-end throughput measurements.

5.2. BitTorrent Throttling

We have also used our tool to investigate whether ISPs perform BitTorrent [Cohen, 2001] prioritization, which has been ivestigated extensively during the last 10 years, for example by the Glasnost project [Marcel Dischinger et al., 2010].

5.2.1. Background

After its appearance in 2001, BitTorrent rose quickly in popularity, becoming in the following years the dominating peer-to-peer file-sharing protocol, estimated to be responsible for between 10 to 30% of the global Internet traffic[12] [TorrentFreak, 2010; Sandvine, 2016]. Unlike client-server file transfer protocols, such as HTTP or FTP, BitTorrent nodes exchange data with many other nodes simultaneously. Some popular implementations, at their default settings, open up to 50 parallel connections per transfer and up to 250 connections overall[13]. While this helps maximize the overall throughput achieved by the node, it may impair performance for other TCP flows that share the same bottleneck: other applications and/or endpoints might transfer data over a single TCP flow, and TCP achieves only per-flow fairness, not per-application or per-endpoint fairness. Thus if the bottleneck is shared by multiple users, as it might happen in a metropolitan area network that is not sufficiently well provisioned to handle utilization from many users at the same time, performance may suffer for all active users.

The immediate reaction from many ISPs, to this surge in peer-to-peer traffic, was to throttle it. The Glasnost project provided an invaluable tool to discover such practices, identifying dozens of ISPs that showed strong evidence of differentiation [Marcel Dischinger and Gummadi, 2011]. Shortly after, users from the BitTorrent community started a joint project to document whether their own ISPs appeared to throttle BitTorrent traffic, which lead to the creation of a list of "Bad ISPs" in a wiki page hosted by the then-popular Azureus client [”Bad ISPs,” n.d.]. In 2009, the European Union initiated a three-year investigation that concluded that approximately 22% of ISPs in Europe throttled peer-to-peer traffic on some occasions between 2009-2012 [BEREC, 2012].

From 2015 onwards, legislation passed in many countries and regions, such as the European Union [”All You Need to Know about Net Neutrality Rules in the EU,” n.d.], in combination with the bad publicity caused by application-specific traffic management practices, have led to many ISPs from Europe and North America to reduce or eliminate such measures. In place, many providers that experience congestion often in metropolitan networks use "capping", i.e. limiting traffic from heavy users, regardless of the application and/or protocols used; either by setting a maximum short-term throughput limit, or by limitting the amount of traffic that can be exchanged for a longer period of time, such as per day or per month. This method remains legal, and is generally viewed as more fair by users, since it does not dictate what kind of traffic users can send or receive, while still allowing a fair

[12]This trend has diminished in recent years, with video streaming and cloud-based file storage now dominating Internet traffic.

[13]Not necessarily all connections are transferring data, due to the "choking" mechanism in the protocol, but many might be.

distribution of available network capacity between active users.

We wanted to find out whether our tool can be used to detect and localize BitTorrent throttling. Although performing such a study in 2017 is not very likely to find many occurences, we hoped that we might still identify throttling in ISPs from countries that have not yet passed net neutrality legislation, or in ISPs that have not yet updated their configurations.

5.2.2. Experiments

To generate BitTorrent traffic, we used our own BitTorrent client based on the libtorrent library [Norberg, n.d.]. What makes it different from regular BitTorrent clients is that:

- Once a torrent reaches 90% completion, it clears all downloaded data and restarts it from scratch. This allows us to run long-lasting experiments in which our nodes are active downloaders, not just uploaders (in BitTorrent jargon, "seeds");
- It bans fast peers, that is peers with which we achieve a throughput exceeding 100 Mbps. Similar to the node filtering we performed in the speedtest study, we want to exclude very fast nodes, since these are unlikely to be throttled, and will just congest our access links;
- It uses only TCP as a transport protocol, with two modes of communication: either regular BitTorrent traffic exchanged on the standard port numbers (6881-6889), or obfuscated BitTorrent traffic exchanged on a non-standard port (8080 and 443);
- It keeps downloaded data only in memory, to avoid any overheads related to disk writes, which may otherwise become a bottleneck and affect performance (since the network links we use are fast, at 1 Gbps, and we already write to the disk captured traffic at a non-negligible rate).

We deployed the client to 7 nodes rented from two cloud service providers, most of them located in different datacenters in Europe, and one in Canada. We configured the client to join the BitTorrent swarms sharing the installation disc images of 10 popular Linux distributions, with a total file size of about 25 GBs. On each node, we run an instance of the client using regular traffic, and another using obfuscated traffic. We took packet captures of all BitTorrent transfers, and we also performed whois and geolocation lookups and collected traceroutes for all observed peers. The experiment ran continuously for approximately 1 week.

The two classes of traffic we define consist of regular BitTorrent traffic sent on the standard port numbers (class 2) and obfuscated BitTorrent traffic sent on HTTP port numbers (class 1). We expect that the most common type of throttling targets class 2 traffic, while class 1 often evades classification and throttling. While it is known that BitTorrent traffic obfuscation may still be detected using deep packet inspection and packet size and timing analysis [Carvalho et al., 2009], we hope that it is not a common practice, since it requires more expensive equipment, and that exchanging such traffic from HTTP port numbers (443 and 8080) may help avoid detection.

We found it difficult to target specific networks, or to create specific topologies, such as two probing nodes exchanging traffic with the same peer simultaneously. This is because BitTorrent nodes choose the peers they exchange traffic with based on a complicated, randomized algorithm; thus the probability that one would choose two of our nodes out of hundreds, or thousands of peers in the swarm is very low. Indeed, we tried this approach by configuring one of the probing nodes as a “master”, and all the other probing nodes in “follower” mode, only allowing connections to/from peers the master has been exchanging traffic over the last 15 minutes; most of the time, the followers failed to connect to any peers, or if they did, they would exchange very little traffic with them, insufficient for our inference algorithm. Thus we confirmed that such an approach performs poorly. Instead, we opted for a brute force approach: we joined a large number of swarms at the same time from all probing nodes and we allowed the protocol to choose the peers at will, hoping that some of the probing nodes would connect to the same peers. When this occurs, the respective paths intersect, forming sub-topologies of the entire overlay network that are useful for our inference.

The topologies we are interested in are the ones where two or more probing nodes connect to the same peer and exchange both classes of traffic, as shown in Figure 41. These topologies allow us to reason about the common link sequence, located most likely in the ISP of the peer—but may also include some transit ISPs. We think transit ISPs are less likely to throttle due to inter-ISP agreements, thus the result would be relevant to the peer's ISP. The mirror topologies, where a single probing node connects to two or more peers, can also be used to run our inference algorithm, however in this case the common link sequence is located in our cloud providers (and possibly in a few transit ISPs), which is not useful in our case since we know that our ISPs are not throttling BitTorrent traffic. Thus we do not consider them.

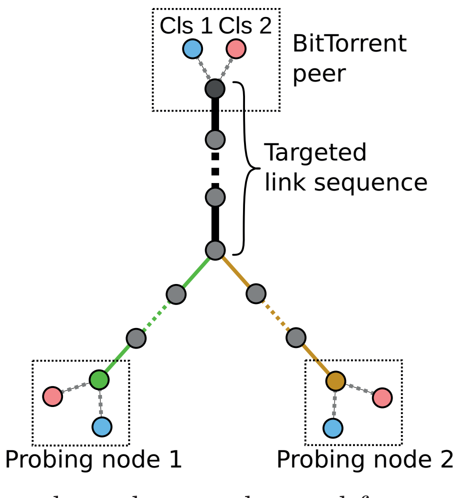


Figure 41. A topology that can be used for neutrality inference.

During the entire experiment, we exchanged over 100 TB of traffic with 50,307 peers from almost 5,000 ISPs (more precisely, autonomous systems) from 189 countries. The top 20 ISPs by peer count are shown in Table 8, while the top 20 countries are shown in Table 9. From this overlay netowrk, we were able to extract approximately 7,000 topologies of the form shown in Figure 41, for 556 ISPs in 76 countries.

ASN	Peer count	Topology count	Name	Country
AS28573	996	34	Embratel	Brazil
AS8151	841	47	UNINET	Mexico
AS3223	811	1	voxility.net	Brazil
AS7738	743	1	Oi’s route object	Brazil
AS3320	600	187	Deutsche Telekom	Germany
AS4755	576	0	BSNL-VSNL	India
AS8708	554	38	RDSNET	Romania
AS8167	527	2	Oi’s route object	Brazil
AS3269	524	10	Interbusiness	Italy
AS43350	476	67	NFOrce Entertainment	Netherlands
AS20845	454	127	DIGI-10	Hungary
AS5483	390	15	Magyar Telekom	Hungary
AS8048	384	34	CANTV-NET	Venezuela
AS12322	367	140	Free SAS	France
AS28513	354	1	Claro	Dominican Republic
AS8402	344	37	Corbina	Russia
AS7922	328	75	Comcast	United States
AS8436	312	45	UPC Magyarorszag	Hungary
AS55836	310	0	R4G	India
AS3215	289	42	France Telecom	France
AS3352	274	34	Red IP Multi Acceso	Spain
AS8452	269	2	Telecom-Egypt-Data	Egypt
AS6830	264	45	UPC Technology	Austria
AS12389	260	10	ROSTELECOM NETS	Russia
AS16276	246	0	OVH	Poland
AS6697	244	0	Beltelecom	Belarus
AS41440	235	22	Sibirtelecom	Russia
AS1267	227	3	Infostrada	Italy
AS17974	227	0	PT Telekomunikasi	Indonesia
AS6057	223	0	ANTEL	Uruguay
AS6849	216	1	UKRTELECOM	Ukraine

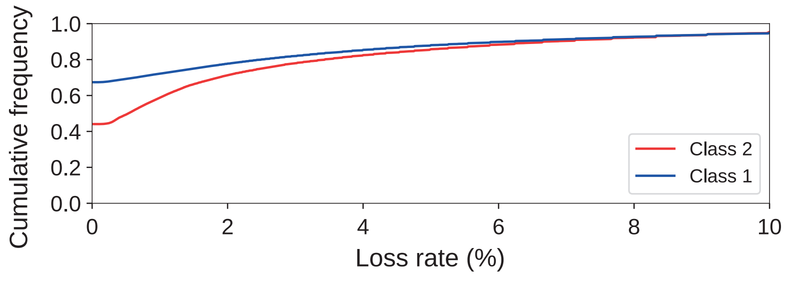
Table 8. Top ISPs we exchanged traffic with.

Code	Peer count	Topology count	Name
BR	6707	9	Brazil
RU	5874	3	Russia
US	4364	7	United States
IN	3160	1	India
UA	2029	1	Ukraine
HU	1634	4	Hungary
FR	1513	16	France
DE	1355	22	Germany
IT	1328	19	Italy
MX	1136	4	Mexico
GB	1031	67	United Kingdom
ES	1027	5	Spain
NL	1017	170	Netherlands
CN	955	2	China
RO	906	1	Romania
AR	904	18	Argentina
CA	800	14	Canada
PL	719	2	Poland
AU	519	1	Australia
ID	502	0	Indonesia
CL	457	10	Chile
VE	440	7	Venezuela
CO	429	1	Colombia
SE	390	4	Sweden
PK	355	3	Pakistan
DZ	354	2	Algeria
VN	343	1	Vietnam
EG	337	2	Egypt
BY	329	1	Belarus
TR	327	5	Turkey
TH	326	1	Thailand

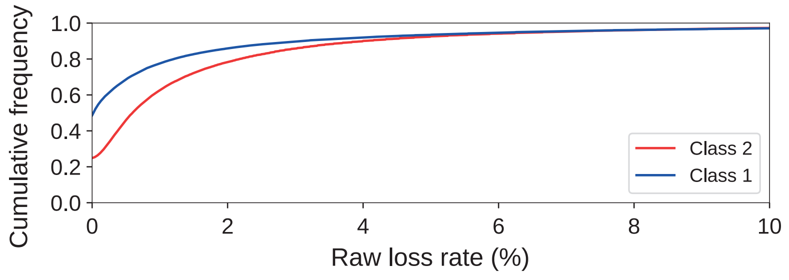
Table 9. Top countries we exchanged traffic with.

5.2.3. Results

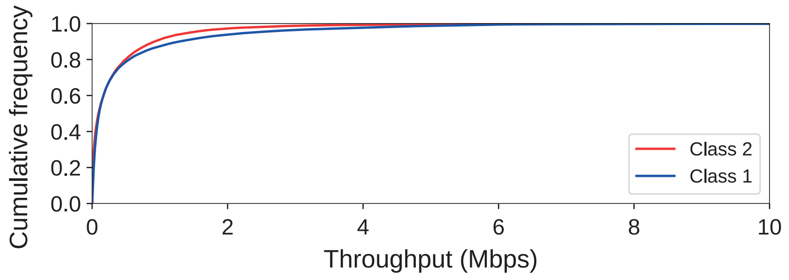
Figure 42 shows the end-to-end measurement statistics for all paths in the network. We can see that there is a slight difference between the two classes of traffic for loss rate zero: class 1 achieves zero loss rate in just over 40% of the intervals, whereas class 2 in only 20%; then the curves become similar at around 3% packet loss. In the plot of the loss rate without packet sampling, the difference persists. On the other hand, the throughput distributions show almost no performance difference between the two classes. These plots suggests that there might be a small performance difference between the two classes of traffic, but it is not clear if the cause is the network being non-neutral, or if there is another kind of bias, such as nodes using the standard port number being connected to the Internet via better-provisioned links that experience less packet loss, because they are special-purpose nodes used to seed these torrents.



(42a) Empirical CDF of the loss rate (with packet sampling).



(42b) Empirical CDF of the raw loss rate (without packet sampling).



(42c) Empirical CDF of the throughput.

Figure 42. End-to-end metric statistics for all paths in the experiment.

We applied the inference algorithm on the 7,000 topologies; only 3 were found to be non-neutral with sufficient confidence. They are listed in Table 10.

ASN	Topology count	Non-neutral	Name	Country
AS29562	20	1	KabelBW	Germany
AS11351	52	2	Time Warner	United States

Table 10. The ISPs for which we had positive results.

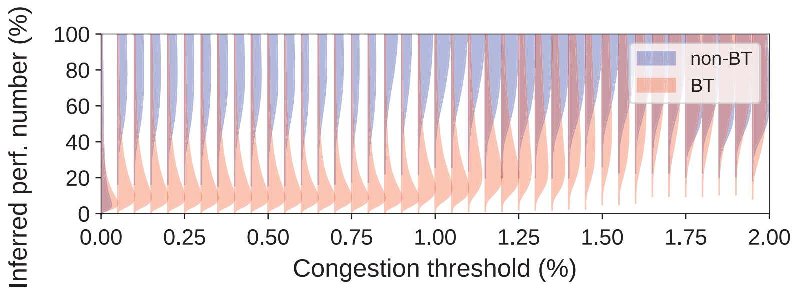
5.2.4. Case Study

We now present the results obtained for AS11351, for which 2 topologies were identified as non-neutral. The two topologies show similar results, so we present them together. The inference results are shown in Figures 43 and 45; and the end-to-end measurements are shown in Figures 44 and 46, respectively. The two topologies share the same remote peer; just the probing nodes are different.

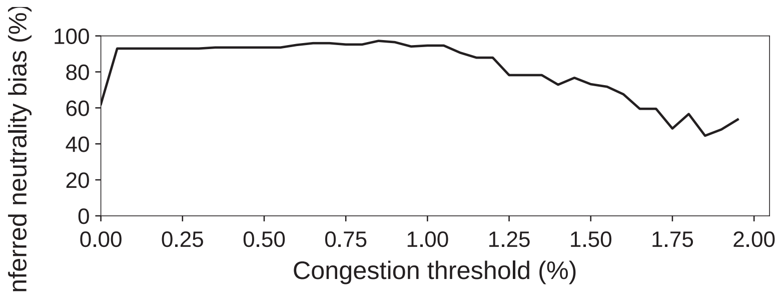
We can see that the inference detects a significant gap consistently for a wide range of congestion thresholds. The difference is also visible in the end-to-end measurements. The data suggests that the network is treating the two classes of traffic differently. What is striking is the large loss rate (90-th percentile around 10%), and the small throughput (90-th percentile at around 0.3 Mbps). In general, small values of throughput can be explained by the fact that the interval size is of 1 minute, while BitTorrent peers may exchange only a few blocks at a time, thus traffic tends to be sent in short bursts that may last only a few seconds; this can cause throughput measurements of even an order of magnitude lower than expected. However, this does not explain why we observe such a large loss rate in these experiments for both classes of traffic (although with different distributions). It is possible that these are users connecting over wireless links; these may introduce high packet loss.

For the same ISP, we have also analyzed the topologies for which the inference verdict is neutral. For about half of these nodes, we observed a much smaller amount of loss rate, and slightly higher throughput; as an example, the data from one topology is shown in Figures 47 and 48. But for other nodes, the measurements are similar to the non-neutral case: very large values of the loss rate, and slightly different distributions of the end-to-end loss rate and throughput, for example as shown in Figures 49 and 50. In these cases the verdict was neutral because the number of intervals was much smaller, thus the inferred posterior distributions were wide and the neutrality inference algorithm could not reach a positive verdict with strong confidence.

We conclude that for this ISP, there are several nodes for which the network appears to treat BitTorrent traffic differently. Unfortunately in each case the common link sequence includes the last mile to the users, thus it is not possible to tell if it was the ISP that introduced this performance difference, or a problem in the user's local network (such as a wireless link suffering from poor signal and/or high interference).

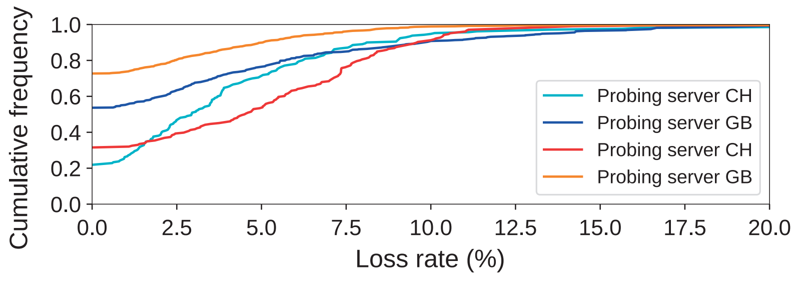


(43a) Posterior distributions.

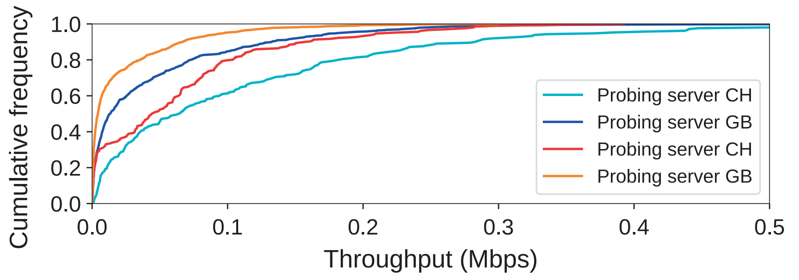


(43b) Inferred neutrality bias.

Figure 43. Result of the Bayesian inference for a topology detected as non-neutral.

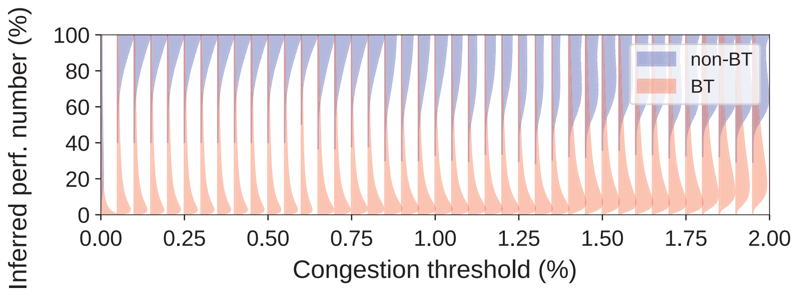


(44a) Empirical CDF of the loss rate.

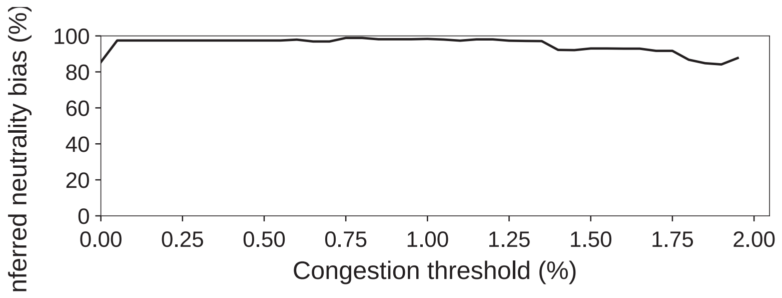


(44b) Empirical CDF of the throughput.

Figure 44. End-to-end loss rate measurements.

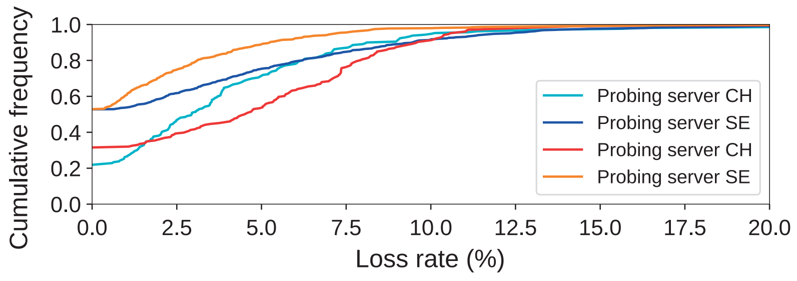


(45a) Posterior distributions.

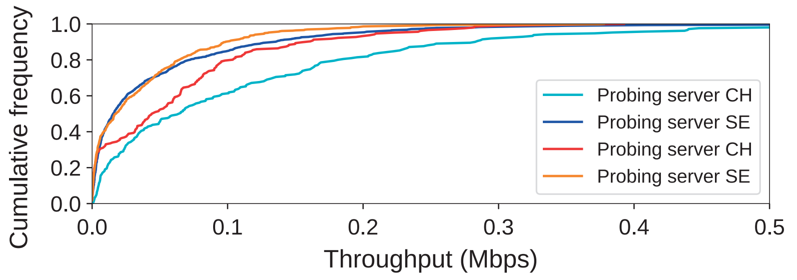


(45b) Inferred neutrality bias.

Figure 45. Result of the Bayesian inference for a topology detected as non-neutral.

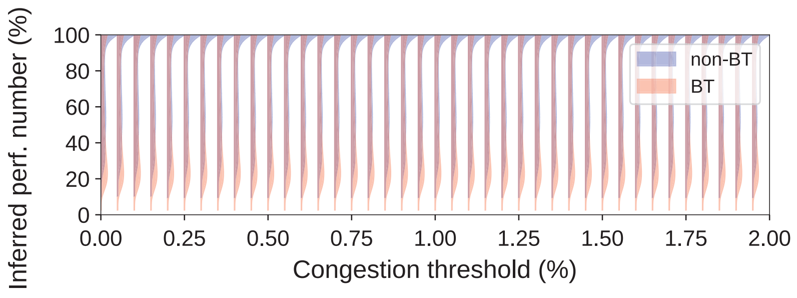


(46a) Empirical CDF of the loss rate.

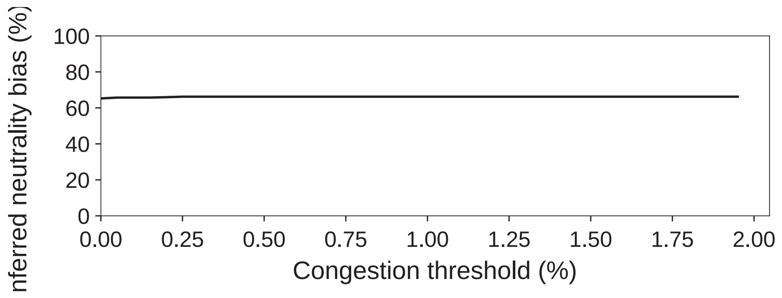


(46b) Empirical CDF of the throughput.

Figure 46. End-to-end loss rate measurements.

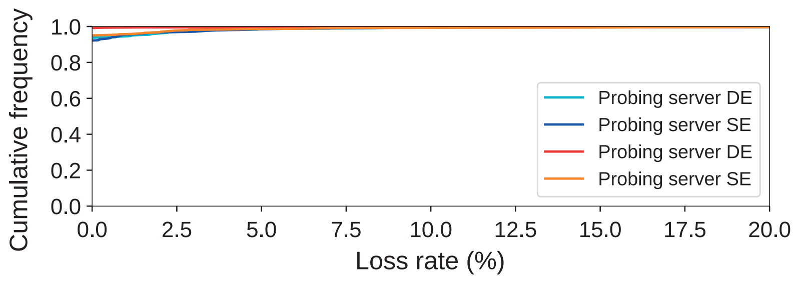


(47a) Posterior distributions.

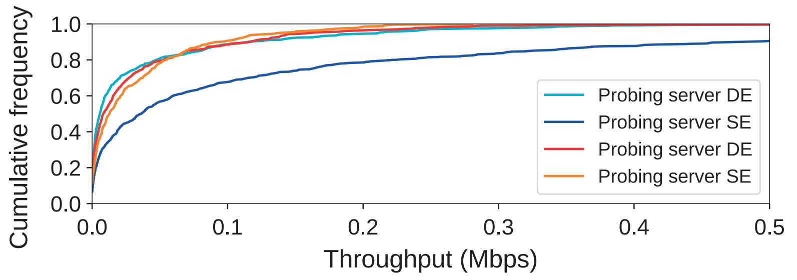


(47b) Inferred neutrality bias.

Figure 47. Result of the Bayesian inference for a topology detected as neutral.

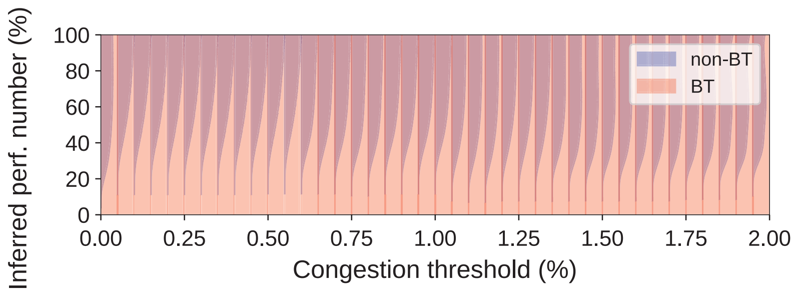


(48a) Empirical CDF of the loss rate.

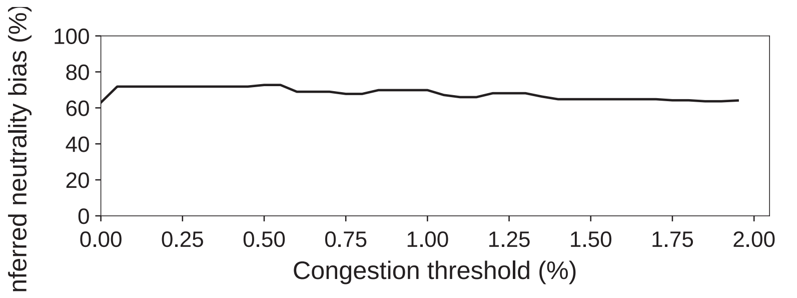


(48b) Empirical CDF of the throughput.

Figure 48. End-to-end loss rate measurements.

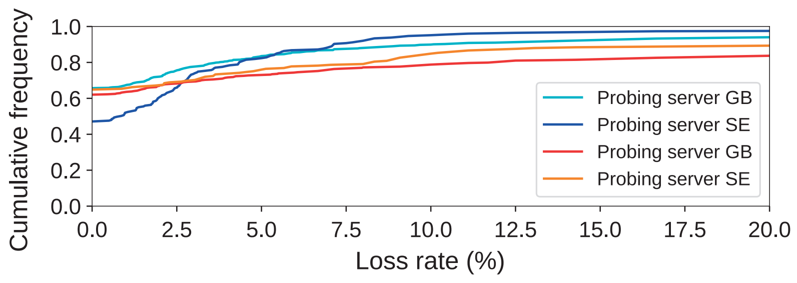


(49a) Posterior distributions.

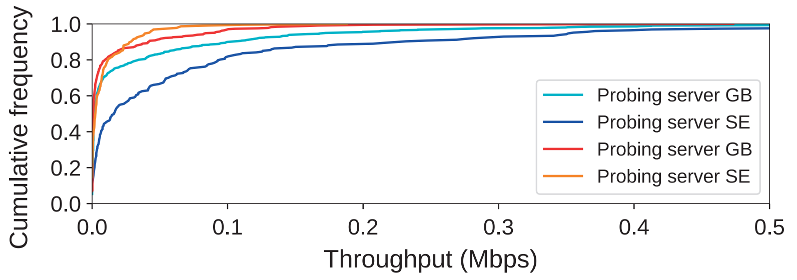


(49b) Inferred neutrality bias.

Figure 49. Result of the Bayesian inference for a topology detected as neutral.



(50a) Empirical CDF of the loss rate.



(50b) Empirical CDF of the throughput.

Figure 50. End-to-end loss rate measurements.

5.2.5. What-if Analysis

We wanted to test whether our tool would catch actual BitTorrent throttling, if it happened, so we implemented it ourselves: We ran a new experiment, in which we throttled traffic exchanged on the standard BitTorrent port range (6881-6889) at a maximum rate of 20 Mbps; any other traffic, including obfuscated BitTorrent traffic, was free to consume the entire capacity of the link, i.e. 1 Gbps. Traffic throttling was performed in both directions: ingress traffic was policed, while egress traffic was shaped; both methods were implemented using the Linux traffic control (`tc`) tool. The exact configuration we used is presented in Appendix 8.3.

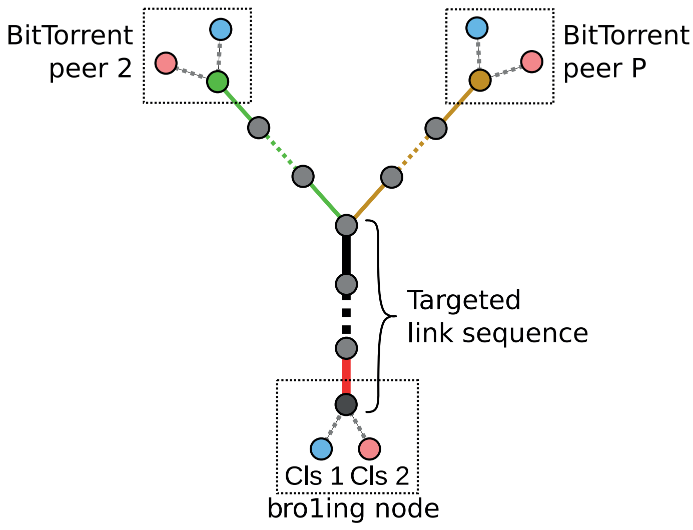


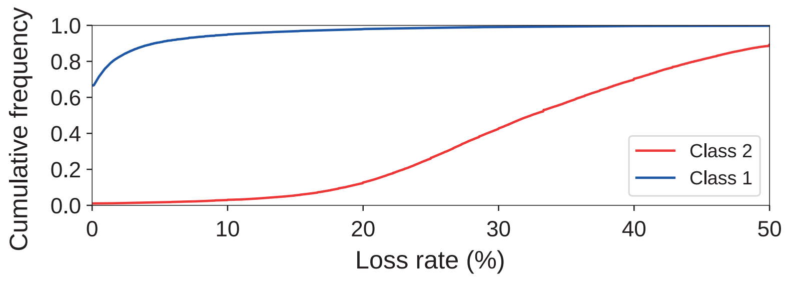
Figure 51. A topology that can be used for neutrality inference.

When analyzing the data, we extracted the topologies formed by a single probing node that is exchanging traffic with two peers, as shown in Figure 51. The throttled link is shown in red: it is the link connecting the probing node to the Internet. After running our experiment for 6 hours, we communicated with about 10,000 peers from 150 countries and 2,231 ISPs; from this overlay, we extracted 2,812 topologies like the one in Figure 51.

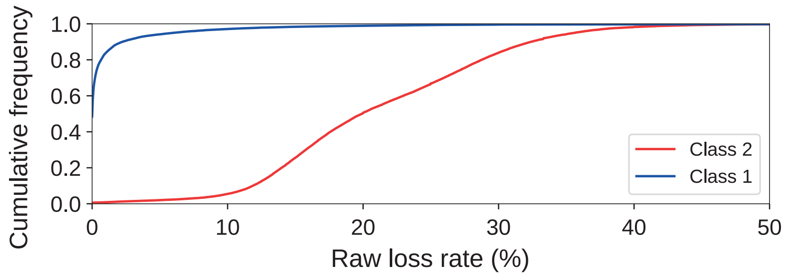
Figure 52 shows the end-to-end measurement statistics for all paths in the network. We can see that there is a visible difference in loss rate between the two classes of traffic: class 1 achieves zero loss rate in just over 60% of the intervals, whereas class 2 almost never does; the curves do not converge not even in the 90-th percentile (10% loss rate for class 1 vs. over 50% for class 2). The difference persists in the plot of the loss rate without packet sampling, slightly smaller in magnitude, but still significant. The throughput distributions also show differences between the two classes: a small difference at the median, and a large difference at the 90-th percentile, where class 2 achieves only 0.5 Mbps, whereas class 1 achieves almost 10 Mbps[15]. These plots suggest that there is likely a performance difference between

[15]There is a large difference between the 90-th percentile for throughput in what-if experiments versus the regular experiments. This is because for what-if experiments, although the number of peers is smaller than in regular experiments, we can form a very large number of inference topologies (millions). We could not process so much data, thus we filtered many topologies, preferring those

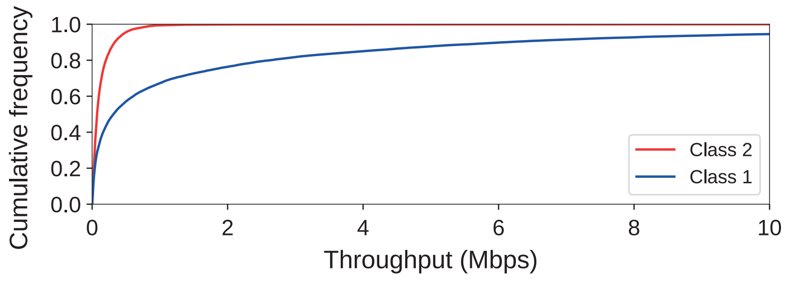
the two classes of traffic, which is much more pronounced compared to the plots we have seen for non-throttled (at least not by us) traffic from the previous section (i.e. Figure 42).



(52a) Empirical CDF of the loss rate (with packet sampling).



(52b) Empirical CDF of the raw loss rate (without packet sampling).



(52c) Empirical CDF of the throughput.

Figure 52. End-to-end metric statistics for all paths in the experiment.

When applying the inference algorithm, we reached a non-neutral verdict for 188 out of 2,812 topologies, i.e. about 10%. The detection rate is quite low, because for many paths, the throttling had little to no effect in many or even all intervals, due to the small throughput, i.e. the throttled link was not the bottleneck. Even so, the detection rate is much higher than in the case of the non-throttled experiment. This shows that our method can be used to detect BitTorrent throttling implemented with traffic policing and/or traffic shaping.

that had traffic in at least 30 intervals for both traffic classes (both, to avoid introducing a bias). It is likely that considering all data, the overall throughput would decrease; but we think a bias would remain.

6. Network Emulation Setup

6.1. Motivation

We now present the network emulation setup we developed to help us evaluate our neutrality inference method. We used this setup extensively in our work throughout all the iterations it took to design and improve our method; it has also helped immensely by making it easy to take measurements from any point in the network, allowing us to observe and understand how traffic reacts to congestion and differentiation. The motivation behind using emulation instead of simulations or instead of replicating a network topology in the lab was the following:

Topology size: we wanted to run experiments in non-trivial topologies, with more than a few links and paths. For example, one of the effects we wanted to observe is how congestion manifests in a link that carries traffic from many users, such as the link serving a small neighborhood that is insufficiently provisioned. In this case, we may want to have many paths that intersect over the link, possibly carrying traffic from flows that have different RTTs or different numbers of hops; some may also experience congestion in a remote network. This is a scenario difficult to replicate in the lab, as it requires configuring a large number of switches and/or routers. It also requires introducing propagation delays on the links, otherwise all RTTs would likely be under 1 ms and certain effects affecting TCP flows would be different[16]. This is not trivial to achieve in the lab: the only practical option is to introduce additional machines on each link, that add a constant delay using, for example, the `netem` component of Linux traffic control, but this greatly increases the number of machines we must dedicate for the experiments, adding cost, installation and maintenance burden to the disadvantages.

Flexibility: as we made gradual changes to our algorithms or the traffic configuration, we sometimes wanted to rerun older experiments. Using a real topology in the lab would make this difficult, as we would have had to either reconfigure the topology often, or to have multiple topologies constantly present.

Realism: network emulation reflects real network behavior better than simulation, since it carries real traffic, exchanged by real software and hardware network stacks. Network simulators usually reimplement the entire stack up to the application layer, and the resulting behavior may be slightly different from how a real machine performs. Emulation eliminates this disadvantage.

Control: when using a real topology, the experimenter is usually limited to using the traffic management capabilities offered either by proprietary equipment, or by software such as the Linux kernel. Both of them have various quirks, and are not flexible enough to cover *all* the possible behaviors of equipment performing differentiation. For example, certain devices that offer such features allow setting only the policing or shaping rate, but not always also the token buffer size; Linux offers more flexible mechanisms, but they are more quirky: for example, it is not recommended to use both throttling and adding propagation delay on the same machine. With emulation, this problem is easier to solve, since if it does not support

[16]For example, we have seen that the performance of TCP flows that are policed while having a small RTT is poor; however it improves greatly as the RTT increases to more than 10 ms.

precisely what we need, we can modify it ourselves more easily than changing the Linux kernel, or proprietary equipment.

6.2. Design

We modeled our emulator after ModelNet [Vahdat et al., 2002], a network emulator written as a FreeBSD kernel module. We found that making changes to an in-kernel emulator is time consuming and error prone; so we rewrote it from scratch in userspace as a Linux application. The first working implementation, which we will call *version 1*, was completed as my Master's thesis project [Mara, 2011]; however it took much more development effort until it was capable of emulating a large non-neutral topology such as the one in Figure 3; we call the current setup *version 2*.

The emulator consists of two components: the routing core and the traffic generator, as shown in Figure 53. The traffic generator consists of one or more real machines that run network applications, sending all their traffic towards the routing core. The routing core consists of a single machine emulating all the routers and links in the network: it receives traffic, emulates all the link behavior (routing, queuing, propagation delays, packet drops, traffic shaping etc.) and finally sends some of the traffic back to the traffic generator.

The architecture of the routing core is depicted in Figure 54. There are three subcomponents, each running in real-time[17] on a dedicated CPU core:

- The packet receiver captures traffic efficiently from the network card via the PF_RING [Deri and others, 2004] kernel module, with packets bypassing the Linux network stack. Each incoming packet is stored in a data structure, using memory from a pre-allocated memory pool to avoid memory allocation latencies. Some initial processing is performed, for example analyzing the IP and TCP headers to determine how the packet should be classified by emulated links, and to compute the path the packet will take through the emulated network. Then a pointer to this structure is sent to the event scheduler.
- The event scheduler performs the bulk of the work: queuing incoming packets on the emulated links, forwarding packets between links, managing all the policers and shapers, dropping packets as needed, and finally, sending packets that exit the network towards the packet sender.
- The packet sender simply receives packets from the scheduler, adjusts the network and link-layer headers as needed, and sends them via PF_RING outside the routing core.

The traffic generator is more simple: it consist of one or more machines that run network applications. Both sides of the communication (i.e. the client and the server) are handled by these machines. To ensure that traffic is routed through the

[17]While Linux is not a real-time OS, we obtain real-time behavior by pinning the three threads to specific CPU cores, and disabling the process scheduler and interrupt handling on these cores. Thus the threads can run indefinitely without interruption. Sharing of data between the three threads is performed with lock-free lists instead of mutexes, to reduce synchronization delays to a minimum.

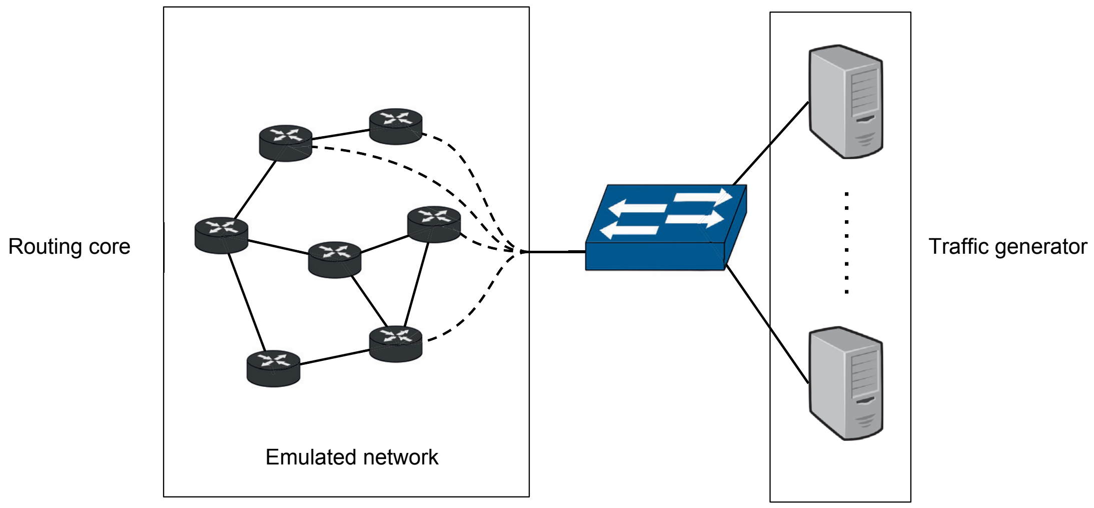


Figure 53. Diagram of the emulator.

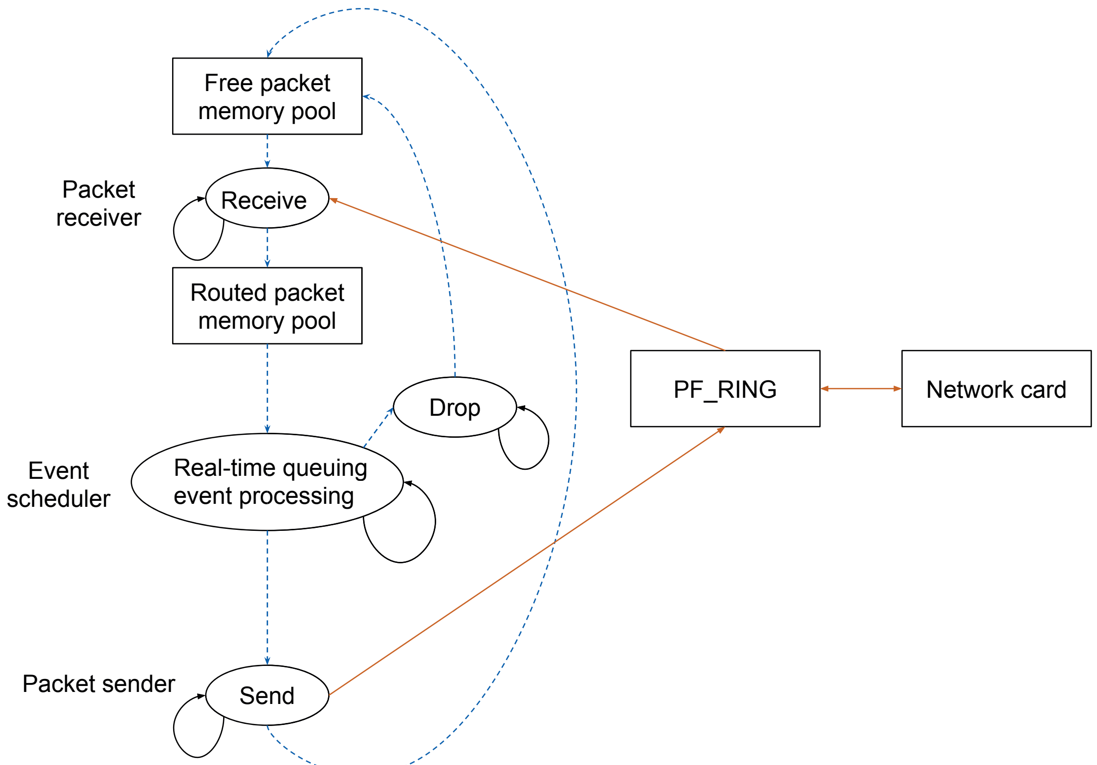


Figure 54. Diagram of the routing core. Blue links show how data travels internally between userspace components. Orange links show how data travels between userspace and the kernel, or the kernel and the network card.

emulator and not shunted directly between them, we use static NAT applied twice on the emulator side (on packet reception and on sending). This means that any application that opens normal network sockets can be used on the traffic generator. In case that many hosts are needed in the network, multiple virtual network interfaces can be configured on the same machine, this functionality being provided by the Linux kernel; in this case, the different applications must bind their sockets to their corresponding IP address.

We found that running one application per IP address does not scale well in emulated networks when the ratio between processes and the number of CPU cores in the traffic generator exceeds a few hundred: in some of our experiments, CPU load was 100%, but the system was spending more than a third of the time in kernel mode. We were not sure if this affected the generated traffic patterns or not—but we wanted to eliminate the risk that the application, instead of the network, became the bottleneck. For this reason, we changed our traffic generation approach to using a single process per core; each process manages a fraction of the network connections, opening and closing sockets as needed, and exchanging data using non-blocking, event-based I/O. This has improved performance, reducing the time spent in kernel mode. However it also limited us in terms of the traffic we were sending: we could no longer run any application we desired, at least not without modifying its code to merge its event loop with ours.

The traffic types we have implemented in our traffic generator are the following:

- On-off traffic: TCP or UDP flows that start or stop at fixed or random times;
- Poisson UDP traffic: UDP flows that send traffic at a configured average rate (measured in packets per second);
- Poisson TCP flows: TCP transfers that initiate at a configured average rate (measured in flows per second);
- Random Pareto-distributed transfer sizes for TCP and UDP flows;
- Per-flow TCP congestion control algorithm configuration (all the algorithms supported by the Linux kernel);
- Media streaming emulation: rate-limited TCP flows that adapt their sending rate based on their recent performance.

6.3. Contributions

Version 1 of the emulator was functional, but suffered from a few deficiencies:

Accuracy: Version 1 did not offer satisfactory accuracy in experiments longer than a few minutes. We measure accuracy the same way as the original ModelNet paper did: in terms of packet processing delay in the routing core, compared to the theoretical delay that should be experienced by the packets. For example, if the emulated links have a propagation delay of 10 ms, a packet processing delay of 1 ms would cause a 10% error. The acceptable error threshold we used throughout our experiments was 10%, and we used links with at least 10 ms propagation delay, thus the maximum acceptable processing delay was 1 ms.

While version 1 obtained a packet processing delay of hundreds of microseconds on average, which was sufficiently small, there were occasional latency spikes that caused a maximum processing delay of up to a few tens of milliseconds. Such spikes caused trains of packets of up to 1-2 seconds to experience unacceptable extra latency. They usually occured about once or twice per minute.

We made two changes that eliminated latency spikes:

- We removed all memory allocations during the experiment, since these may cause latency spikes due to the locking in the allocator code. Instead, we pre-allocated all data structures, or used memory pools of pre-allocated objects. To make this manageable, we wrote a small library preloaded in the routing core, that intercepts all memory allocation calls made to the standard C library, and records a histogram of the amount of time they took to execute, optionally also dumping a stacktrace and stopping the program. We used this library to eliminate one by one all allocations from the latency-sensitive code paths, such as the event scheduler loop.

- In addition to pinning the three threads of the routing core to separate CPU cores, we also instructed the Linux kernel to isolate those cores, such that no other applications are scheduled to execute there; we disabled handling of all interrupts on those cores (and stop the IRQ balancer daemon, which may undo this); and we compiled a custom kernel with the tickless option set, which allowed us to completely disable the process scheduler on those cores. Latencies introduced by context switches caused by the scheduler may be in excess of 1 ms; these measures eliminated them.

With these changes, version 2 achieved a maximum processing delay of the same order of magnitude with the average processing delay, for all the topologies we used in our experiments, such as the one seen earlier in Figure 3.

Performance: Version 1 could not emulate topologies having a total capacity—i.e. the maximum rate at which traffic can arrive at the routing core, equal to the sum of the capacities of all the emulated access links—of more than a few hundreds of Mbps. There were two potential bottlenecks in our design: the receiving and sending of traffic from/to the network card; and the queuing event scheduler loop.

The former was not under our control, but given that we used a fast packet capture and injection framework, we expected the bottleneck to be higher than 1 Gbps, thus it was likely not the culprit. We confirmed this by bypassing the queuing event scheduler, i.e. the routing core would just receive packets, perform NAT, and send them back into the network. With this setup, we could reach over 2 Gbps easily.

The bottleneck was the scheduler loop. We found that some of the changes we made to eliminate latency spikes also helped in increasing its throughput. Here is why: the scheduler loop must process events at specific moments in time; at each iteration, the loop reads the system clock, checks which events have expired so must be processed, processes them—which may schedule new events for later—and then moves on to the next iteration. Any latency spikes in a previous iteration may cause the scheduler to process much more expired events than usual in the current iteration, causing it to take longer than usual, thus triggering another latency spike in an avalanche effect. If this lasts for at least a few hundreds of milliseconds, the hosts that are injecting traffic into the routing core will slow down due to normal TCP behavior (since the hosts observe an increase of RTT), which causes throughput to drop. If the drop is large enough, the routing core may recover, but then TCP flows increase their sending rate gradually until the core is overloaded again and another latency spike occurs. Eliminating random latency spikes prevents this effect and makes the emulator more predictible: given the total capacity of the network, we can tell whether the emulator can run the experiment with satisfactory accuracy consistently, regardless of the topology[19], based on previous benchmarks.

Another major improvement we made that helped improve throughput was to optimize the scheduler loop. Initially, we were storing all events in a priority queue implemented with a map from event expiration time to a packet structure. We bechmarked three different implementations for this map: the C++ STL map, which uses internally a red-black tree; the Qt QMap, which uses skip lists; and our own implementation, using a pairing heap. We found that the average scheduler loop time was twice as small in the STL version than in the other two implementations, i.e. the STL version was the fastest. However in the end we decided to eliminate the global event queue completely, taking advantage of the fact that events are intrinsically ordered correctly in the packet queues of the emulated links. In each iteration of the loop, the scheduler can poll all queues to see if any events need processing; in case they do, the respective packets are removed from the queues and possibly pushed in the queues of other links[18]. Implementing the queues with circular buffers, we can process each event in $O(1)$ operations instead of $O(\log num\_packets)$, as was the case with a red-black tree (where $num\_packets$ is the total number of packets in flight in the network). This reduced the average loop iteration time by a factor of 5; for example, on a topology for which the loop using a priority queue achieved 140 us, version 2 of the emulator achieves around 30 us. Another advantage of this approach is that the processing delay does not depend strongly on the number of packets in flight in the network; whereas for the priority queue it does, with all packets suffering extra processing delay during periods of high congestion, for example if there is queuing on many links in the network.

Traffic management: Version 1 did not support any traffic management methods, only a single tail-drop, head-drop or random-drop queue per link. In version 2, we implemented traffic shaping, allowing the use of multiple queues per link, with the possibility to assign different rates and buffer sizes to each queue; and traffic policing, allowing the use of one or more token buffer filters that either allow or drop packets before queuing packets on the link. Packets are assigned to the corresponding shaper or policer based on the Differentiated Services field from the IP header; we support up to 8 traffic classes, but we could extend this to 256 by using all the bits in the field. To improve performance, we do not perform traffic classification in the routing core; instead, the applications from the traffic generator set the correct value of the field through a socket option for every connection.

Monitoring: Version 1 took measurements only through packet counters on the links in the network. We sometimes wanted to inspect what happened on a link in more detail; for this purpose, we implemented in version 2: (i) metric recording, which records periodically various parameters of the links or token buckets, similar to SNMP performance monitoring; (ii) packet captures, i.e. all ingress or egress traffic for an emulated link could be captured in a PCAP file, which could then be manually inspected. Both of them store data in pre-allocated buffers in memory

[19]Within reasonable limits, since increasing the number of links over a certain limit will eventually reduce throughput. However, in most of our topologies, consisting of up to a few hundred links, this was never a bottleneck. Thus we did not study this limitation.

[18]We do not store packet contents in the queues as that would be inefficient. We just store pointers to packet structures.

until the end of the experiment, to avoid the performance impact that may be caused by disk accesses.

6.4. Performance Evaluation

Our goal is to emulate networks with as much total capacity as possible, while still achieving acceptable accuracy. Thus we evaluate the emulator performance by running benchmarking experiments on topologies with the same shape but different total capacities. For each one, we measure the average and maximum packet processing delay. Generally, we use links with 10 ms of propagation delay, so we define acceptable processing delays as maximum delays under 1 ms.

The topology we use in the benchmarking experiments consists of 5 parallel paths, each one traversing 3 links. The links have 10 ms propagation delay and the buffer size set to 1 RTT worth of traffic. On each path, we start 20 long TCP flows, i.e. the total number of flows in the network is 100. We run the experiment for 1 minute.

6.4.1. Performance of Version 1

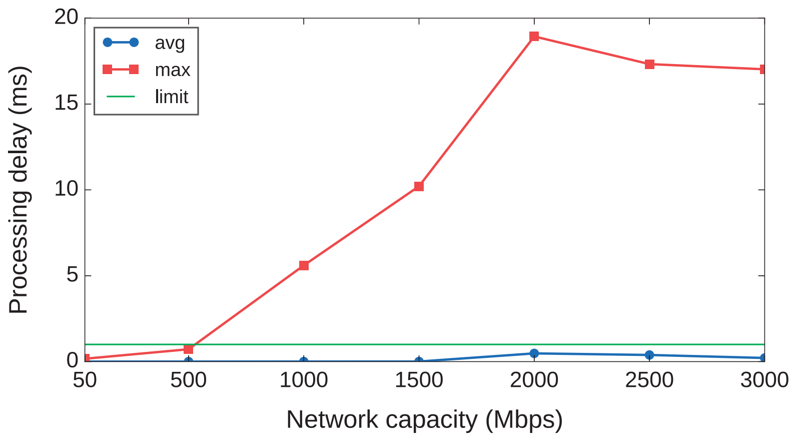


Figure 55. Benchmark results for emulator version 1.

The results of the benchmark for version 1 are presented in Figure 55. The processing delay is excellent for a network capacity of 50 Mbps, but almost reaches the 1 ms limit for a capacity of 500 Mbps. As the network capacity increases, the average processing delay remains below the limit, but the maximum delay increases to tens of milliseconds, well beyond the acceptable limit. After the network capacity reaches 2 Gbps, we see an apparent drop in processing delay; but this just an artefact that appears because the emulator starts dropping packets during packet capture, since the code is unable to read them quickly enough from the PF_RING buffers. The drop fraction is small, just above 0.1% of packets; but packets are dropped in trains during high latency periods. As a result, TCP flows observe larger packet loss and back off, thus the amount of traffic that enters the emulator is reduced, leading to the drop in processing delay that can be seen in the plot.

The data shows that version 1 of the emulator can emulate accurately only topologies with a total capacity below 0.5 Gbps.

6.4.2. Performance of Version 2

The results of the same benchmark, this time ran on version 2 of the emulator, are shown in Figure 56. We can see that the processing delay is very small (under 100 us) for emulated network capacities of up to 3.5 Gbps, and only exceeds the acceptable limit at 4 Gbps. For this entire range of capacities, no packet drops were experienced by the receiver, unlike we have seen for version 1.

Similar results are obtained for more complex topologies or longer experiments. For example, the emulated topology we have seen earlier in Figure 3 had a total capacity of just over 1 Gbps, and an experiment duration of 33 minutes. The average processing delay was recorded as 20 us, and the maximum 329 us. The maximum relative error in the delay experienced by packets over the paths they traversed, relative to the theoretical one, was of only 1%.

Based on this data, we can state with good confidence that the emulation setup we used to evaluate our method offered sufficient performance and accuracy to ensure that the measurements we collected were reliable and realistic.

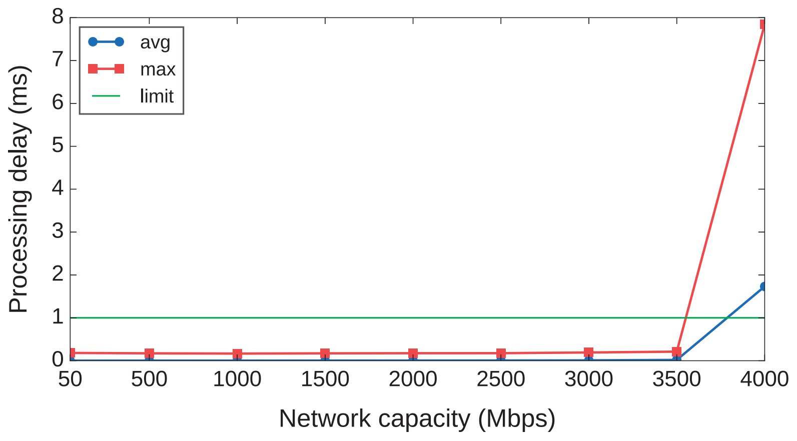


Figure 56. Benchmark results for emulator version 2.

7. Conclusion

We studied the problem of assessing the neutrality of a target link sequence based on end-to-end measurements. For this purpose, we based our work on the method presented by [Z. Zhang et al., 2014], which assumes perfect measurements. In practice, that method suffers from several sources of errors. We identified and analyzed each one, and redesigned the measurement and inference process to be reliable, by not making unrealistic assumptions or using magic thresholds. We made three major modifications:

Firstly, we modeled the uncertainty in the inferred link performance, and proposed a method to estimate it. We showed that this uncertainty is significant and unavoidable in practical scenarios, due to the limited duration over which paths are stable in typical networks (due to routing changes); which in turn imposes a constraint over the number of samples we can collect. We designed a method to estimate this uncertainty from the measurements using Bayesian inference.

Secondly, we proposed a method to identify and filter out inaccurate path congestion measurements. We found that without this filtering, up to 40% of the path pair congestion states may be incorrect in some experiments, due to the way Zhang et al.'s method correlates congestion states of different paths. We showed that our method to estimate the accuracy of the measured congestion states in the absence of any ground-truth knowledge, in order to filter out the ones likely to suffer from poor correlation, is effective: it could reduce the fraction of incorrect states to the target value we desired.

Thirdly, we redesigned the non-neutrality inference step to take into account the uncertainties caused by imperfect measurements, and to eliminate any arbitrary hand-picked thresholds. The method we proposed is configured by a single, well-understood parameter: the desired significance level of the non-neutrality verdict.

We evaluated the method with simulations, emulations and on real networks. We used simulations to test each component of our method in isolation, and showed that they work as expected. We used emulations to evaluate the full method on real traffic, in a controlled environment; we found this setup particularly useful, so we published it, in the hope that it may aid future research. Finally, we evaluated the method on the Internet, leveraging documented throttling of SMTP connections in the Amazon cloud; we demonstrated that we can correctly detect and localize realistic throttling outside our control.

We used our method in two studies, investigating suspicions that a set of ISPs prioritize speed-test traffic, or differentiate against BitTorrent traffic. We obtained reliable evidence that they do not. We also conducted what-if analyses to show that our method would have correctly detected and localized throttling or prioritization, had they occurred in the investigated networks.

In a world where the edge has moved towards playing an active role in shaping traffic behavior, such as through source pacing or custom congestion control, there are many reasons why two traffic classes may experience different end-to-end performance. If we care to reason about network neutrality, we suggest that we do it based on reliable evidence of network behavior.

We conclude that it is feasible to detect and localize network neutrality violations based solely on end-to-end measurements, without assuming a perfect measurement process. We hope that this work is a small step toward making the Internet more transparent.

8. Appendix

8.1. M-Lab Download Throughput Statistics Extraction

Algorithm 3: BigQuery script that computes throughput histogram.

```
SELECT
  ROUND((web100_log_entry.snap.HCThruOctetsAcked * 8) /
        (web100_log_entry.snap.SndLimTimeRwin +
         web100_log_entry.snap.SndLimTimeCwnd +
         web100_log_entry.snap.SndLimTimeSnd) / 10) * 10
         AS Throughput,
  COUNT(*) AS Count
  FROM
    plx.google:m_lab.ndt.all
  WHERE
    -- NDT
    project = 0 AND
    -- Server to client (download)
    IS_EXPLICITLY_DEFINED(connection_spec.data_direction) AND
    connection_spec.data_direction = 1 AND
    -- Last row in table contains the experiment result
    IS_EXPLICITLY_DEFINED(web100_log_entry.is_last_entry) AND
    web100_log_entry.is_last_entry = True AND
    -- The TCP connection was established normally
    (web100_log_entry.snap.State == 1 OR
     (web100_log_entry.snap.State >= 5 AND
      web100_log_entry.snap.State <= 11)) AND
    -- At least 8kB were sent
    IS_EXPLICITLY_DEFINED(web100_log_entry.snap.HCThruOctetsAcked) AND
    web100_log_entry.snap.HCThruOctetsAcked >= 8192 AND
    -- Test did not fail (ran for >= 9 seconds)
    (web100_log_entry.snap.SndLimTimeRwin +
     web100_log_entry.snap.SndLimTimeCwnd +
     web100_log_entry.snap.SndLimTimeSnd) >= 9 * POW(10, 6) AND
    -- Test stopped normally (ran for <= 1 hour)
    (web100_log_entry.snap.SndLimTimeRwin +
     web100_log_entry.snap.SndLimTimeCwnd +
     web100_log_entry.snap.SndLimTimeSnd) < 3600 * POW(10, 6) AND
    -- Data is sane
    IS_EXPLICITLY_DEFINED(web100_log_entry.snap.SegsRetrans) AND
    IS_EXPLICITLY_DEFINED(web100_log_entry.snap.DataSegsOut) AND
    web100_log_entry.snap.DataSegsOut > 0  AND
    web100_log_entry.snap.SegsRetrans <
     web100_log_entry.snap.DataSegsOut
    -- Date filter
    AND YEAR(UTC_USEC_TO_YEAR(web100_log_entry.log_time
                              * 1000000)) >= 2016
  GROUP BY
    Throughput
  ORDER BY
    Throughput ASC;
```

Results are shown in Table 11. The data was summarized from 117,671,648 download throughput tests. 95% of tests achieved less than 110 Mbps, and 99% of tests achieved less than 200 Mbps.

Throughput (Mbps)	Count	Fraction (%)	Cumulative (%)
0-10	47987831	40.8	40.8
10-20	23558505	20	60.8
20-30	12149660	10.3	71.1
30-40	8035525	6.8	78.0
40-50	4884304	4.2	82.1
50-60	4115825	3.5	85.6
60-70	3818792	3.2	88.8
70-80	2068640	1.8	**90.6**
80-90	2531530	2.2	92.8
90-100	1605010	1.4	94.1
100-110	1149506	1.0	**95.1**
110-200	4632580	3.9	**99**
200-380	1164915	0.9	**99.9**

Table 11. Query results.

8.2. Speed-test Throttling Configuration

We used a Cisco C3850 switch, configured to police traffic tagged with DSCP value 34 (0x22), at 20 Mbps:

```
configure terminal

class-map match-any SLOW
 match ip dscp 34

policy-map SLOW
 class SLOW
  police cir 20m bc 249810 conform-action transmit exceed-action drop

interface GigabitEthernet1/0/3
 switchport mode access
 speed 1000

interface GigabitEthernet1/0/4
 switchport mode access
 speed 1000

interface GigabitEthernet1/0/24
 switchport mode access
 speed 1000
 service-policy input SLOW
 service-policy output SLOW

end
```

Port 80 traffic was marked with the DSCP value on a Linux router using iptables:

```
$ iptables -t mangle -L
Chain PREROUTING (policy ACCEPT)
target prot opt source   destination

Chain INPUT (policy ACCEPT)
target prot opt source   destination

Chain FORWARD (policy ACCEPT)
target prot opt source   destination
DSCP   tcp  --  anywhere anywhere    tcp spt:http DSCP set 0x22
DSCP   tcp  --  anywhere anywhere    tcp dpt:http DSCP set 0x22
DSCP   tcp  --  anywhere anywhere    tcp spt:5001 DSCP set 0x22
DSCP   tcp  --  anywhere anywhere    tcp dpt:5001 DSCP set 0x22

Chain OUTPUT (policy ACCEPT)
target prot opt source   destination
```

```
Chain POSTROUTING (policy ACCEPT)
target prot opt source   destination
```

Note: we also throttled port-5001 traffic (iperf [”iPerf - The TCP, UDP and SCTP Network Bandwidth Measurement Tool,” n.d.]), which we used for testing that the policer works correctly.

8.3. BitTorrent Throttling Configuration

We used egress traffic shaping and ingress traffic policing via the Linux `tc` mechanism [”Linux Advanced Routing and Traffic Control,” n.d.]:

```
#!/bin/bash

export PATH=$PATH:/sbin

# Find WAN interface
wan=$(ip route | grep default | grep -oE 'dev [a-z0-9]*' | cut -d ' ' -f 2)

# Shape egress BitTorrent traffic at 20 Mbps
tc qdisc del dev $wan root
tc qdisc add dev $wan root handle 1: htb default 99
tc class add dev $wan parent 1: classid 1:66 htb rate 20mbit ceil 20mbit
tc qdisc add dev $wan parent 1:66 handle 66: sfq perturb 10

# Classification for egress BitTorrent traffic (and iperf)
for port in 6881 6882 6883 6884 6885 6886 6887 6888 6889 5001
do
  tc filter add dev $wan protocol ip parent 1: prio 1 \
    u32 match ip dport $port 0xffff flowid 1:66
  tc filter add dev $wan protocol ip parent 1: prio 1 \
    u32 match ip sport $port 0xffff flowid 1:66
done

# Police ingress BitTorrent (and iperf) traffic at 20 Mbps
tc qdisc del dev $wan ingress
tc qdisc add dev $wan handle ffff: ingress
for port in 6881 6882 6883 6884 6885 6886 6887 6888 6889 5001
do
  tc filter add dev $wan parent ffff: protocol ip prio 1 \
    u32 match ip dport $port 0xffff \
    police rate 20mbit burst 100k mtu 65535 drop flowid :1
  tc filter add dev $wan parent ffff: protocol ip prio 1 \
    u32 match ip sport $port 0xffff \
    police rate 20mbit burst 100k mtu 65535 drop flowid :1
done
```

Note: we also throttled port-5001 traffic (iperf), which we used for testing that the policer and the shaper work correctly.

Figures

Tables

Algorithms

References

"All You Need to Know about Net Neutrality Rules in the EU." n.d. http://berec.europa.eu/eng/netneutrality/.

Amazon.com. 2016. "EC2 Port 25 Throttle." https://aws.amazon.com/premium-support/knowledge-center/ec2-port-25-throttle/.

Andrey, Kolmogorov. 1933. "Sulla Determinazione Empirica Di Una Legge Di Distribuzione." *G. Ist. Ital. Attuari* 4: 83–91.

"Bad ISPs." n.d. https://wiki.vuze.com/w/Bad_ISPs.

Bashko, Vitali, Nikolay Melnikov, Anuj Sehgal, and Jürgen Schönwälder. 2013. "Bonafide: A Traffic Shaping Detection Tool for Mobile Networks." In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, 328–35. IEEE.

BEREC. 2012. "BEREC Findings on Traffic Management Practices in Europe." http://berec.europa.eu/eng/document_register/subject_matter/berec/download/0/45-berec-findings-on-traffic-management-pra_0.pdf.

Bu, T., N. Duffield, F. Lo Presti, and D. Towsley. 2002. "Network Tomography on General Topologies." In *Proceedings of the ACM SIGMETRICS Conference*.

Byrne, Seán. 2016. "Ookla Confirms Myce ISP Speed Test Manipulation Article." http://www.myce.com/news/ookla-contacts-myce-regarding-inflated-isp-speed-test-results-article-78473/.

Caceres, R., N. G. Duffield, J. Horowitz, and D. Towsley. 1999. "Multicast-Based Inference of Network-Internal Loss Characteristics." *IEEE Transactions on Information Theory* 45: 2462–80.

Carvalho, David A, Manuela Pereira, and Mário M Freire. 2009. "Towards the Detection of Encrypted BitTorrent Traffic through Deep Packet Inspection." *Security Technology*. Springer, 265–72.

Chun, Brent, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. 2003. "Planetlab: An Overlay Testbed for Broad-Coverage Services." *ACM SIGCOMM Computer Communication Review* 33 (3). ACM: 3–12.

Coates, M., and R. Nowak. 2000. "Network Loss Inference Using Unicast End-to-End Measurement." In *Proceedings of the ITC Specialist Seminar on IP Traffic Measurement, Modeling and Management*.

Cohen, Bram. 2001. "BitTorrent." .

Council of the European Union. 2015. "Guidelines on the Implementation by National Regulators of European Net Neutrality Rules." http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32015R2120.

Crovella, Mark E, and Azer Bestavros. 1997. “Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes.” *IEEE/ACM Transactions on Networking* 5 (6). IEEE: 835 – 846.

Deri, Luca, and others. 2004. “Improving Passive Packet Capture: Beyond Device Polling.” In *Proceedings of SANE*, 2004:85–93. Amsterdam, Netherlands.

Deutsch, L Peter. 1996. “DEFLATE Compressed Data Format Specification Version 1.3.”

Dhamdhere, Amogh, Renata Teixeira, Constantine Drovolis, and Christophe Diot. 2007. “NetDiagnoser: Troubleshooting Network Unreachabilities Using End-to-End Probes and Routing Data.” In *Proceedings of the ACM CoNEXT Conference*.

Dischinger, Marcel, and Krishna P. Gummadi. 2011. “Glasnost: Results from Tests for BitTorrent Traffic Shaping.” http://broadband.mpi-sws.org/transparency/results/.

Dischinger, Marcel, Massimiliano Marcon, Saikat Guha, Krishna P Gummadi, Ratul Mahajan, and Stefan Saroiu. 2010. “Glasnost: Enabling End Users to Detect Traffic Differentiation.” In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

Dischinger, M, A Mislove, A Haeberlen, and K P Gummadi. 2008. “Detecting BitTorrent Blocking.” In *Proceedings of the ACM Internet Measurement Conference (IMC)*.

Duffield, Nick. 2003. “Simple Network Performance Tomography.” In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, 210–15. ACM.

———. 2006. “Network Tomography of Binary Network Performance Characteristics.” *IEEE Transactions on Information Theory* 52 (12): 5373–88.

Flach, Tobias, Pavlos Papageorge, Andreas Terzis, Luis Pedrosa, Yuchung Cheng, Tayeb Karim, Ethan Katz-Bassett, and Ramesh Govindan. 2016. “An Internet-Wide Analysis of Traffic Policing .” In *Proceedings of the ACM Conference of the Special Interest Group on Data Communication (SIGCOMM ’16)* . Florianópolis, Brazil .

Gailly, JL, and M Adler. 1998. “The Zlib Home Page.” https://zlib.net/.

Ghita, Denisa, Katerina Argyraki, and Patrick Thiran. 2010. “Network Tomography on Correlated Links.” In *Proceedings of the ACM Internet Measurement Conference (IMC)*.

“iPerf - The TCP, UDP and SCTP Network Bandwidth Measurement Tool.” n.d. https://iperf.fr/.

Jahanzaib, Syed. 2015. “Prioritize SpeedTest.Net Results via Mikrotik Queue.” https://aacable.wordpress.com/2015/11/12/prioritize-speedtest-net-results-via-mikrotik-queue/.

Kanuparthy, Partha, and Constantine Dovrolis. 2010. "DiffProbe: Detecting ISP Service Discrimination." In *Proceedings of the IEEE INFOCOM Conference*.

———. 2011. "ShaperProbe: End-to-End Detection of ISP Traffic Shaping Using Active Methods." In *Proceedings of the ACM Internet Measurement Conference (IMC)*.

Knight, Simon, Hung X Nguyen, Nick Falkner, Rhys Bowden, and Matthew Roughan. 2011. "The Internet Topology Zoo." *IEEE Journal on Selected Areas in Communications* 29 (9). IEEE: 1765–75.

"Linux Advanced Routing and Traffic Control." n.d. http://lartc.org/.

Lu, Guohan, Yan Chen, Stefan Birrer, Fabian E Bustamante, Chi Yin Cheung, and Xing Li. 2007. "End-to-End Inference of Router Packet Forwarding Priority." In *Proceedings of the IEEE INFOCOM Conference*.

Mara, Ovidiu. 2011. "A Testbed for Evaluating the Practicality of Network Tomography." Phdthesis, Ecole Polytechnique Federale de Lausanne.

———. 2017. "LINE Network Emulator." http://wiki.epfl.ch/line/.

Marsaglia, George, and others. 2003. "Xorshift Rngs." *Journal of Statistical Software* 8 (14): 1–6.

Measurement Lab. 2017. "The M-Lab NDT Data Set 2016-01-01–2017-07-27." https://measurementlab.net/tools/ndt.

Molavi Kakhki, Arash, Abbas Razaghpanah, Anke Li, Hyungjoon Koo, Rajesh Golani, David Choffnes, Phillipa Gill, and Alan Mislove. 2015. "Identifying Traffic Differentiation in Mobile Networks." In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, 239–51. ACM.

Nagle, John. 1985. "On Packet Switches with Infinite Storage."

———. 1986. "Congestion in the Internet–Doing Something About It." In *Proceedings of the 16-17 January 1986 DARPA Gateway Algorithms and Data Structures Task Force*. IETF.

Nguyen, Hung X., and P. Thiran. 2007a. "The Boolean Solution to the Congested IP Link Location Problem: Theory and Practice." In *Proceedings of the IEEE INFOCOM Conference*.

Nguyen, Hung X., and Patrick Thiran. 2007b. "Network Loss Inference with Second Order Statistics of End-to-End Flows." In *Proceedings of the IEEE Internet Measurement Conference (IMC)*.

Nikolai, Smirnov. 1948. "Table for Estimating the Goodness of Fit of Empirical Distributions." *Annals of Mathematical Statistics* 19: 279–81.

Norberg, Arvid. 2009. "BEP 29: uTorrent Transport Protocol." http://www.bittorrent.org/beps/bep_0029.html.

———. n.d. “Libtorrent.” http://http://www.libtorrent.org.

Oya, Kentaro, Takanori Kitada, and Shinichi Tanaka. 2011. “Study on Random Number Generator in Monte Carlo Code.” *Transactions of the Atomic Energy Society of Japan* 10 (4). Atomic Energy Society of Japan: 301–9.

Padmanabhan, V. N., L. Qiu, and H. J. Wang. 2003. “Server-Based Inference of Internet Performance.” In *Proceedings of the IEEE INFOCOM Conference*.

“RedIRIS Network Map.” n.d. http://www.rediris.es/conectividad/weathermap/.

Sandvine. 2016. “Global Internet Phenomena.” https://www.sandvine.com/downloads/general/global-internet-phenomena/2016/global-internet-phenomena-report-latin-america-and-north-america.pdf.

Song, Han Hee, Lili Qiu, and Yin Zhang. 2006. “NetQuest: A Flexible Framework for Large-Scale Network Measurement.” In *Proceedings of the ACM SIGMETRICS Conference*.

Sundaresan, Srikanth, Walter De Donato, Nick Feamster, Renata Teixeira, Sam Crawford, and Antonio Pescapè 2011. “Broadband Internet Performance: A View from the Gateway.” In *ACM SIGCOMM Computer Communication Review*, 41:134–45. 4. ACM.

Tariq, Mukarram Bin, Murtaza Motiwala, Nick Feamster, and Mostafa Ammar. 2008. “Detecting Network Neutrality Violations with Causal Inference.” In *Proceedings of the ACM CoNEXT Conference*.

The PlanetLab Consortium. n.d. “PlanetLab - Hosting Requirements.” https://planet-lab.org/hosting.

The Verge. 2014. “Major ISPs Accused of Deliberately Throttling Traffic.” https://www.theverge.com/2014/5/6/5686780/major-isps-accused-of-deliberately-throttling-traffic.

TorrentFreak. 2010. “BitTorrent Still Dominates Global Internet Traffic.” https://torrentfreak.com/bittorrent-still-dominates-global-internet-traffic-101026/.

Tsang, Yolanda, A Tsang, Mark Coates, and Robert Nowak. 2000. “Passive Unicast Network Tomography Based on TCP Monitoring.” In *Rice University, ECE Department*. Citeseer.

Vahdat, Amin, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostic, Jeff Chase, and David Becker. 2002. “Scalability and Accuracy in a Large-Scale Network Emulator.” In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

Vigna, Sebastiano. 2017. “Further Scramblings of Marsaglia’s Xorshift Generators.” *Journal of Computational and Applied Mathematics* 315. Elsevier: 175–81.

Vukas, Matt. 2014. “Comcast Is Definitely Throttling Netflix, and It’s Infuriating.” http://mattvukas.com/2014/02/10/comcast-definitely-throttling-netflix-infuriating/.

Weinsberg, Udi, Augustin Soule, and Laurent Massoulie. 2011. “Inferring Traffic Shaping and Policy Parameters Using End Host Measurements.” In *Proceedings of the IEEE INFOCOM Mini-Conference*.

Zhang, Ying, Zhuoqing Morley Mao, and Ming Zhang. 2007. “Detecting Traffic Differentiation in Backbone ISPs with NetPolice.” In *Proceedings of the ACM Internet Measurement Conference (IMC)*.

Zhang, Zhiyong, Ovidiu Mara, and Katerina Argyraki. 2014. “Network Neutrality Inference.” In *ACM SIGCOMM Computer Communication Review*, 44:63–74. 4. ACM.

Curriculum Vitae

Education

- Present: PhD in Computer Science, EPFL, Switzerland
- 2011: MSc in Computer Science, EPFL, Switzerland
- 2009: BSc in Computer Science, PUB, Romania

Work experience

- EPFL, Network Architecture Laboratory
    - PhD in Computer Science
- Google, Infrastructure
    - Software Eng. Intern
- EPFL, Operating Systems Laboratory
    - Research Intern

Publications

- *Network Neutrality Inference*, Zhang, Zhiyong, Ovidiu Mara, and Katerina Argyraki; ACM SIGCOMM 2014;
- *Neutrality Inference without Perfect Measurements*, Ovidiu Mara, Zhiyong Zhang, Katerina Argyraki; under review;
- *Limits of Network Tomography*, Ovidiu Mara, Zhiyong Zhang, Katerina Argyraki; under review.