# Traffic receipts for network transparency

THÈSE N$^{O}$ 8904 (2018)

PRÉSENTÉE LE 30 NOVEMBRE 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

Pavlos NIKOLOPOULOS

acceptée sur proposition du jury:

Prof. P. Thiran, président du jury
Prof. A. Argyraki, directrice de thèse
Prof. A. Perrig, rapporteur
Prof. D. Shah, rapporteur
Prof. M. Grossglauser, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2018

*If I have seen further, it is by standing*
*on the shoulders of giants.*
— Isaac Newton

To the girls of my life, Nathalie and Danae.
To my parents and my sister.

# Abstract

Today's Internet is not transparent: when packets get lost or delayed, there is typically no information about where the problem occurred, hence no information about who is responsible. This results in Internet service providers (ISPs) offering service level agreements (SLAs) that cannot be verified, and governments enacting neutrality regulations that cannot be enforced. To remedy this, we propose a "transparency system," where each participating network emits receipts for traffic it receives and delivers; an independent monitor collects these receipts and makes decisions regarding the network's performance and neutrality (or lack thereof). The main challenge we face is misbehavior: On the one hand, a network that participates in such a system has a clear incentive to game the system and influence the monitor's decisions to its advantage, by manipulating either the receipts it emits or the corresponding traffic. On the other hand, the monitor (or, more precisely, an adversary who has access to the same information as the monitor, e.g., a government that has subpoenaed the monitor's records) may have an incentive to use the receipts emitted by a network in order to infer information that is otherwise private to the network, in particular, its internal topology. We make three contributions, each one to prevent a different type of misbehavior: (1) *Incentive-compatible reporting*, which ensures that networks have no incentive to manipulate the receipts they emit in order to claim better performance or fake neutrality. The key to our solution is a trade-off that we discover between network performance and neutrality: we design our system such that the more a network tries to exaggerate its estimated performance the more likely it is to be perceived to violate neutrality (and vice versa). (2) *Unbiased reporting*, which ensures that networks cannot manipulate the traffic for which they emit receipts in order to claim better performance. The key to our solution is delayed disclosure: we design receipt generation such that, by the time a network has all the information it needs to emit a correct receipt, the network has already forwarded the traffic that this receipt concerns, hence cannot manipulate it. (3) *Topology-obfuscation reporting*, which enables networks to emit the information that is necessary for the monitor to make correct decisions without leaking any information about internal network topology. The key to our solution is the observation that topology inference exploits the diversity of pairwise similarities between the delay vectors of different network paths; hence, we design receipt generation such that any delay vectors that the monitor might compute have almost 0 pairwise similarities. We conclude that it is possible to design a transparency system that enables networks to report on their own performance such that networks have no incentive to game the system and no fear of leaking information about their private topology.

**Keywords:** Network transparency, consistent sampling, delayed disclosure, secure performance reports, mechanism design, network topology inference.

# Résumé

Aujourd'hui Internet n'est pas transparent : lorsque des paquets sont perdus ou retardés, il n'y a typiquement aucune information concernant la cause du problème et par conséquent nulle indication quand à qui en est responsable. Ceci entraine une entente entre Fournisseurs de Service Internet sur des Accords de Niveau de Services qui ne peuvent pas être vérifiés, et des gouvernements promulguant des lois dictant la neutralité qui ne peuvent pas être appliquées. Pour remédier à cela nous proposons un "système de transparence" où chaque réseau participant émet des reçus pour le trafique qu'il reçoit et fournit. Un moniteur indépendant collecte ces reçus et décide du niveau de performance et de neutralité du réseau. La difficulté principale à laquelle nous faisons face sont les comportements frauduleux : D'une part, un réseau appartenant à un tel système a clairement un intérêt à fausser les données et influencer la décision du moniteur pour améliorer le résultat en son avantage, en manipulant les reçus émis ou le trafique correspondant. D'autre part, le moniteur (ou plutôt un adversaire qui a accès aux mêmes informations que le moniteur, par exemple un gouvernement qui requière par mandat l'accès aux données du moniteur) peut avoir un intérêt à utiliser les reçus émis par un réseau afin de déduire des informations normalement privées et en particulier sa topologie. Nous réalisons trois contributions, chacune pour prévenir un type de comportement frauduleux différent : Le rapport compatible à la motivation, qui assure que les réseaux n'ont pas d'intérêt à manipuler les reçus qu'il émettent afin de justifier une meilleure performance ou une fausse neutralité. La clé de notre solution est un compromis que nous découvrons entre la performance du réseau et la neutralité : nous concevons notre système de manière à ce que plus un réseau amplifie sa performance estimée, plus il sera passible d'être reconnu comme transgressant la neutralité (et vice versa). Le rapport impartial, qui garantie que les réseaux ne peuvent pas manipuler le trafique pour lequel ils émettent des reçus afin de déclarer une meilleure performance. La clé de notre solution est la divulgation retardée : nous concevons la création de reçus de telle manière que le temps que le réseau reçoive toute l'information dont il a besoin pour émettre un bon reçu, le réseau a déjà traité le trafique correspondant à ce reçu et donc ne peut plus le fausser. Le rapport opaque de topologie, qui permet au réseau d'émettre les informations nécessaires au moniteur pour prendre les bonnes décisions sans révéler aucune information concernant la topologie interne du réseau. La clé de notre solution est le constat que la déduction de topologie utilise la diversité des similarités de toute paire de vecteurs de retard entre différents chemins de réseaux. Par conséquent, nous créons le système afin que les vecteurs de retard que le moniteur pourrait calculer présentent presque 0-similarités de paire.

**Mots-clés:** Transparence du réseau, échantillonnage consistant, divulgation différée, rapports de performance fiables, théorie des mécanismes d'incitation, inférence de topologie du réseau.

# Acknowledgments

My PhD journey required the feedback and support of many people; without their help, my thesis could not be the same. To all of them, I owe my deepest gratitude.

First and foremost, I am incredibly grateful to my wonderful adviser, Katerina Argyraki, for her guidance, unwavering support, and positive reinforcement in the past six years. Katerina is a great researcher and undoubtedly a role model for her students. She was the main reason why I got accepted to the EPFL doctoral school, and she has been there for me ever since, to help me in both my research and personal life. Katerina taught me how to think about the practical implications of my project, abstract away uninteresting details, and improve my paper writing and presentation skills. Her sharp intellect, her ability to find the right course of action in all situations, and her being supportive of one's progress in a heartfelt way, constitute an art form of mentoring that will always be an inspiration to me. I will cherish her genuine kindness, her patience, and her infectious enthusiasm. Katerina is the best adviser and hard-working collaborator I could ever ask for. It just can't get any better than this.

I am also thankful to the other members of my thesis committee—Professor Patrick Thiran, Professor Matthias Grossglauser, Professor Adrian Perrig, and Professor Devavrat Shah—for their feedback and for dedicating time to review and improve my thesis. Their excellent work has been of invaluable help and high influence during the entire period of my doctoral studies.

The main contribution of this thesis would not be possible without my co-authors: Christos Pappas, Adrian Perrig, Katerina Argyraki. I am really thankful to you and I appreciate your indispensable collaboration help.

I am fortunate to have worked with wonderful colleagues from the Network Architecture Lab. Special thanks to Iris Safaka, Dimitri Melissovas, Mihai Dobrescu, and Georgia Fragkouli for being valuable friends, for sharing their knowledge and expertise, and for having spent much of their time to give me feedback. I feel indebted to my lab fellow, Ovidiu Mara, for his precious help with producing input data for my project during the first years of my PhD. I will always remember beautiful moments with all other lab members: George Ioannidis, Jonas Fietz, Arseniy Zaostrovnykh, Zhiyong Zhang, Luis Pedrosa, Marie-Jeanne Lagarde and Isabelle Coke. I thank you all for making my everyday life at EPFL such a pleasant experience.

During this journey, the Greek "gang" from EPFL and Lausanne has been a home and a family for me. I would like to specially thank my closest friends from the "maestros" group—Manos, Christos, Alexandros, Stefanos and Natassa—for the beautiful discussions we had about

research and various other topics, and for always being in good mood, happy and supportive. I feel grateful to Matt, for being not only a very close friend, but also my technical support before paper submission deadlines. I want to thank Eleni, for her being an infinite source of laugh and optimism, and also Stella, Panayiotis, Kyveli, Chrysa, Vassiliki, Stela, Marios, Vivi, Kostas, Eleni, Sofia, Anna-Maria, Giannis, Katerina, Loukia, Giannis, Christos, Dimitris, Simos, Maya, Apostolis, Eirini, Thodoris, Giannis, Evi, Myrsini, Maria, Kyriaki, Ariadni, Kalliopi, Eleni, Odysseas, Leonidas, Iro, Polydefkis, Onur, Jean, Arash, George (all of you), and many more, for the charming discussions we had, our common birthday parties (with Iro and Giannis), our skiing activities, our trips and Sunday coffees, and all the happy moments we shared together. The Ph.D. can be a rough journey sometimes, and I was fortunate to have had all these friends around to ensure that it was a swift and smooth ride for me. Folks, I really thank you. You made Lausanne feel like home.

From the bottom of my heart, I would like to thank my family for their boundless love and support. I am grateful to my parents, George and Fouli, and my sister, Asteria, for always being there for me and encouraging me to work hard and follow my dreams. They have been the best listeners in difficult cases, and they have always managed to relieve my stress and help me make the correct decisions—thank you, I am extremely fortunate to have you in my life! Also, I would like to thank the new members of our extended family, Aliki and George, Nikos and Sabine, for their smile, kindness and encouragement.

Last but certainly not least, I would like to thank my wife, Nathalie, who has been extremely patient and supportive throughout this endeavor and has helped me the most during my PhD years (by also writing the French version of the abstract); and our daughter, Danae, for the ultimate joy and happiness she brings to our family. Thank you for your unconditional love and support, and for having endured living with a PhD candidate. This thesis is for you.

*Lausanne, 18 October 2018* P. N.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

## 1.1 The need for transparency

Today's Internet offers no transparency on the fate of forwarded traffic. When packets get lost or delayed, there is typically no information about where the problem occurred, hence no information about who is responsible.

The lack of transparency results in service level agreements (SLAs) and neutrality regulations that cannot be enforced: First, Internet service providers (ISPs) guarantee that their network will honor a minimum delivery rate (equivalently, a maximum loss rate) and a maximum latency [11, 3, 8], even though there exists no systematic way to estimate their loss rate or delay distribution. Second, governments require that ISPs should not (de)prioritize certain traffic classes, even though there exists no systematic way to detect traffic (de)prioritization; 42 governments have already adopted such neutrality regulations or laws, while 6 more are considering them [9, 50, 21, 7]. However, due to the lack of verifiable evidence, ISPs can deny any accusation of following anti-competitive practices, such as blocking or degrading the performance of competing services on their networks [5, 2, 4].

We argue that the requisition for a transparency framework is now necessary. If network users and governments care enough for SLAs and neutrality regulations to exist, then there should be a systematic way to enforce them in practice. A verifiable measurement system that provides trustworthy loss and latency information would reveal to consumers and regulators how ISPs manage traffic, and it would lead to better-informed legal and policy decisions [52, 44]. Moreover, it would offer a quantitative economic incentive to the ISPs to innovate [41].

We support the idea of a "network-layer transparency system" [12, 13, 15, 60, 14, 62, 61, 52, 45], where each participating network emits receipts for traffic it receives and delivers. The receipts are crafted such that an independent monitor can process them and estimate each network's mean loss rate and delay distribution quantiles with respect to various traffic aggregates, enabling the verification of both SLAs and neutrality regulations. Networks participate either because they are expected to by their governments, or (our preferred scenario) by their own

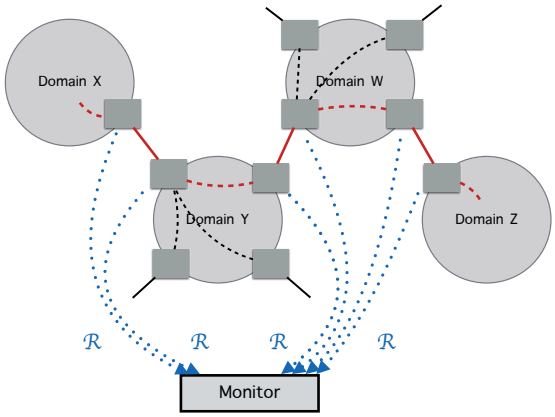



Figure 1.1 – Example network model. The black dashed lines denote the intra-domain routes between an ingress and an egress node of a domain; and the black solid lines denote the inter-domain links. The blue dotted lines show the receipts that are emitted to the monitor. The red line shows an example end-to-end packet flow from a source to a destination domain.

choice, because producing verifiable evidence of honoring SLAs and network neutrality is good for their reputation. Today, even if an ISP *wants* to produce such verifiable evidence, e.g., to defend itself against wrongful suspicion of SLA or neutrality violation, it has no way of doing it.

A critical challenge is resistance to misbehavior: a network that reports on its own performance has a clear incentive to try to game the estimation to its advantage. Transparency, however, demands a truthful performance evaluation that is resilient against any effort of unscrupulous domains to abuse it and exaggerate their performance. Any misbehavior has to be either avoided or detected and extracted from the measurement process. In this dissertation, we address this challenge without trusted hardware.

## 1.2 Definitions

A “domain” is a contiguous network area managed by a single administrative entity, e.g., an ISP, an Autonomous System (AS), an Internet eXchange Point (IXP), an enterprise or campus network, the data-center network of a content provider.

Each domain that participates in our system deploys a special “node” at each point where it exchanges traffic with another domain (Fig. 1.1). Each node runs an algorithm that takes as input a set of configuration parameters and the sequence of packets arriving at the node, and outputs a sequence of receipts that it sends to the monitor. The nodes are co-located with the border routers of the domains, and we use the terms "node" and "border router" interchangeably.

A “monitor” is a logically centralized entity that collects the receipts emitted by the participating domains and uses them to answer performance and neutrality questions about the

domains. The monitor could be owned and managed by the participating domains themselves (in which case it would be implemented as a decentralized system), or it could be owned and managed by a single authority (like those that manage the root DNS servers). In the latter case, anyone interested in the receipts (e.g., the participating domains or their customers) would gain access to them through the owning authority.

We define two kinds of traffic units: “flows” and “aggregates”:

- A flow is the set of all packets observed by a node that have the same source and destination IP prefix.
- An aggregate is a set of packets with some common observable characteristic, e.g., the set of packets from a given source to a given destination domain, or all BitTorrent packets from a given source domain; so, an aggregate may be a flow’s subset or contain one or multiple flows.

A flow/aggregate’s “path” is the sequence of all nodes that observe its traffic; a flow/aggregate’s “source node” is the first node on its path.

Nodes can classify packets per flow, but are not aware of aggregates.

The monitor, on the other hand, defines and answers questions with respect to aggregates, based on loss and/or delay “estimates”, e.g. the average loss rate or delay between two nodes, the delay variance or some important percentile of it, etc. When we talk about the "accuracy" of an estimate computed by the monitor, we mean $(\gamma,\epsilon)$-accuracy, where $\gamma$ is the lower bound of the probability (or confidence level) that the relative estimation error is $\epsilon$. This is a standard accuracy metric for loss estimates and was recently defined for delay estimates as well [55, 42]. To determine which receipts are relevant to a given aggregate (hence should be taken into account to answer questions w.r.t. that aggregate), the monitor may need help from the aggregate’s source node. For example, if the monitor defines an aggregate as all the BitTorrent packets from a given source domain, then it needs help from that source domain to determine which are the relevant receipts. By design (and for reasons we explain later), the nodes that produce the receipts are not aware of aggregates, hence cannot tag receipts with aggregate-related information.

## 1.3 Problem and Goals

We describe our problem and goals in two parts. This is because, in our context, there are two kinds of entities, the domains and the monitor(s), which play distinct roles.

**[Part I – Domains]:** Given the traffic they observe, the domains should emit receipts in a way that guarantees:

- Low resource usage.

- Topology privacy.

The first objective comes from the fact that the receipts are produced at border routers, which are already burdened with supporting line rates of tens of Gbps and increasing numbers of protocol suites.

The second objective comes from the fact that receipts may leak information about a domain that is otherwise private: it has been shown that an entity with access to network-path measurements can infer network topology through network tomography [30, 29, 56, 19, 28, 26, 27, 57]. Hence, from the point of view of a domain, the monitor (or any entity that has access to the monitor's data) may misbehave and try to use the domain's receipts to infer the domain's internal topology.

To address this part of the problem, we set two goals for our design:

G1: The algorithm running on each node should be *lightweight*: it should not increase the node's data-to-control-path bandwidth and data-path memory by more than a few percentage points.

G2: The receipts emitted by a domain should be *privacy preserving*: they should not reveal information about the domain's internal topology that is otherwise secret.

**[Part II – Monitor]:** Given the emitted receipts, the monitor should be able to answer performance and neutrality questions about the domains. We distinguish two steps in this process:

- Statistical estimation, i.e., the computation of statistics that summarize each domain's packet loss and delay distributions (e.g. mean, variance, or distribution percentiles).
- Decision making, i.e., the way in which the estimates are used to characterize domain performance and neutrality.

Consider, for example, Fig. 1.1. Based on the receipts emitted by the nodes, the monitor first computes loss and delay statistics about the domains. Then, based on these statistics, the monitor tries to answer the following questions: How does Y's loss and delay performance compare to W's with respect to X-to-Z traffic? Are Y and W treating all X-to-Z traffic the same, or are they discriminating against video traffic?

From the point of view of the monitor, nodes (and the domains that own them) may strategically misbehave: a node may try to manipulate the contents of the receipts, or the way it forwards traffic to which the receipts refer, so that the decisions made by the monitor are in favor of a domain. Such manipulation may be practiced either independently or in collusion with another node, which may or may not belong to the same domain.

To address such misbehavior, we set another two goals for our design:

G3: The monitor's statistical estimation should be *unbiased*: given an aggregate $G$, the monitor should be able to estimate a domain's loss/delay with respect to $G$ with a desired $(\gamma,\epsilon)$-accuracy in a desired measurement interval $T$; and no domain that observes $G$ should be able to bias the estimation by preferentially forwarding $G$ or any part of it.

G4: The monitor's decision making should be *incentive-compatible*: given an aggregate $G$, no domain that observes $G$ should have the incentive to strategically manipulate the contents of its receipts in order to exaggerate its reported performance or hide neutrality violations. This should be resistant both to independent manipulation and collusion.

## 1.4 Contributions

We present a transparency system that is based on sampling. Participating networks emit receipts for a small sample of the packets that they receive and deliver; based on these receipts, the monitor produces point estimates about loss and delay that enable domain-performance comparison and traffic-discrimination (or non-neutrality) detection.

We start from consistent hash-based sampling [65, 25], for reasons that we explain in Chapter 2, and we enhance it with three mechanisms to meet our goals G2–G4:

1. Incentive-compatible reporting (Chapter. 3), which incentivizes the domains not to falsify the contents of the receipts in order to claim lower loss/latency or fewer neutrality violations—instead, the domains maximize their estimated performance only if they report it truthfully.

2. Unbiased reporting (Chapter. 4), which is a lightweight sampling algorithm that incentivizes the domains not to treat sampled traffic preferentially to claim lower loss/latency—instead, the domains maximize their estimated performance only if they forward traffic legitimately.

3. Topology-obfuscation reporting (Chapter. 5), which enables the domains to emit all the information that is necessary for the monitor to make correct decisions, without leaking any information about its internal topology.

The resulting system also meets goal G1: Running our sampling algorithm requires modest functionality that can be afforded by modern networks. The data-to-control-path bandwidth is not increased by more than 1%, which is equal to the sampling probability and hence configurable. Also, the data-path memory is not increased by more than a few percentage points, e.g., it is less than 10MB for measuring traffic volumes that exceed 600Mbps with $(0.95, \pm 10\%)$-accuracy within 5 minutes, and it can be even less for larger time intervals.

## 1.5 Assumptions

We make the following assumptions:

1. Domains may drop or delay traffic, they may also misbehave in order to exaggerate their reported performance or hide possible neutrality violations, but they do not undertake other malicious actions like modifying, injecting, or replaying traffic. These types of attacks have been addressed in prior work [60, 61].

2. Domains know the true loss and delay of their own inter-domain links. For any given aggregate $G$ and any given domain $z$ on $G$'s path, the neighbor domains of $z$ know the true loss/delay performance of their inter-domain link with $z$ with respect to $G$'s packets. We believe that this is a reasonable assumption, because the domains that are adjacent to an inter-domain link can always directly debug that link and measure the loss and queuing delay it is experiencing. This assumption is used in Chapters 3 and 5.

3. The monitor employs standard statistical techniques to compute the accuracy of each loss or delay estimate; to provide confidence intervals, these techniques must assume something about the nature of the loss/delay that is being estimated, and the typical assumption is that loss/delay is either i.i.d. (independent identically distributed) across all packets, or follows the Gilbert model [36]. This assumption is used in Chapter 3 to design the neutrality detector (Section §3.4), and in Chapter 4 for the parametrization of our system (Section §4.6).

4. The packet arrivals of each flow form a stationary and ergodic process with a high enough rate (e.g. at the order of OC-12 or higher) that ergodic convergence is achieved in less than 100msec. The limitation resulting from this assumption is that we cannot reason about aggregates that consist of relatively few packets/sec. We think that this is acceptable given our motivation to enable the comparison of domain performance and the verification of neutrality regulations, which typically apply to relatively large aggregates. This assumption is used in Chapter 4 for the proofs of Lemmas 4.4.1 and 4.5.1.

# 2 Sampling-based Transparency

In this chapter we describe and justify the starting point of our design: each node emits receipts only for a small *sample* of the packets it observes. By using packet sampling, we take an initial step toward meeting our goals: we enable a lightweight implementation (goal G1); we enable the use of statistical estimation with well-established accuracy guarantees (goal G3); we enable the monitor to define aggregates of interest without having to pre-inform the domains about them (also related to goal G3). More specifically, we use pseudo-random hash-based sampling [65, 25], because of its "consistency" feature, which leads to lower estimation errors than classic random sampling and enables incentive-compatible reporting on packet-loss events (goal G4).

We first describe a very basic transparency system that is based on consistent sampling (§2.1), and then explain why we chose it as our starting design point (§2).

## 2.1 Basic Transparency System

Each node emits a receipt for a small sample of packets, drawn from all the packets it observes. Each receipt carries: a *digest* that uniquely identifies the packet with high probability; a *timestamp* that specifies when the packet was observed at the node; and a *flow ID* that specifies the packet's source and destination prefix.

Instead of using classic random sampling, which lets the nodes independently pick their samples over the observed packets, we use consistent hash-based sampling, in which all nodes along a path sample *consistently* the same set of packets modulo loss: Each packet is either sampled by all the nodes that observe it or by none of them [65, 25].

Consistent sampling relies on hashing the non-mutable contents of each observed packet and sampling the packet if the outcome falls within a predetermined range. A hash function with strong randomization properties results in almost uniform (pseudo-random) sampling, where the sampling probability is determined by the range of the hash function. Alg. 1 shows a basic hash-based sampling algorithm: When a packet $p$ arrives at a node, the node first applies a

**Algorithm 1** OnPacketArrival ($p$)

| | |
|---|---|
| $\hat{p}$ | non-mutable content of packet $p$ |
| *Receipt*() | constructs a receipt |
| *Hash*() | hash function with strong randomization properties |
| *Range* | subset of *Hash*'s range |

1: **if** $Hash(\hat{p}) \in Range$ **then**
2: $rec \leftarrow Receipt(p, currentTime)$
3: Emit receipt *rec*.
4: **end if**

hash function on a part of $p$'s non-mutable content (the non-mutable fields of the IP header and a small part of the payload). If the outcome falls within a given range (line 1), then the packet is considered as a sample and a receipt is constructed (line 2), using information from $p$ for computing the digest and the flow ID, and the current time[1] for the timestamp.

After having received all the receipts from the nodes, the monitor runs statistical-estimation and decision-making processes. It estimates each domain's performance with respect to an aggregate $G$ by identifying and comparing the sample receipts that the entry and exit nodes of the domain emit for $G$'s packets. Obviously, the monitor may also estimate performance between any two nodes that observe $G$. Assuming that the receipts correspond to a random sample of $G$'s packets, the monitor uses well-established unbiased and efficient point estimators to estimate summarizing statistics of the loss/delay distributions [42, 55].

## 2.2 Why sampling?

Sampling enables a lightweight implementation (goal G1) and provides a good starting point towards unbiasability (goal G3):

First, a node is not required to emit receipts per packet or maintain per-aggregate or per-flow state on the data-path, as in other solutions [12, 35, 15, 32, 62, 46]. Thus, it has a significantly reduced equipment cost, as required by goal G1.

Second, assuming a random and representative sample, the monitor can estimate any summarizing statistic about loss and delay distributions. This is because statistics provide us with a variety of unbiased and efficient point estimators about unknown population parameters (such as the population mean, variance, or percentiles) and a well-established theory on their accuracy (or confidence level) [55, 42].

Third, sampling enables tuning the resource cost and accuracy with only one knob, the sampling probability. The sampling probability determines the rate at which each node emits receipts, hence the fraction of the node's network bandwidth that is consumed by emitted receipts. At the same time, the rate at which a domain $z$'s nodes emit receipts for an aggregate

[1] Clock synchronization can be achieved by NTP[47] or GPS.

$G$ determines the $(\gamma,\epsilon)$-accuracy with which the monitor estimates $z$'s performance w.r.t. $G$ in a desired time interval—or, equivalently, the time interval in which the monitor estimates $z$'s performance w.r.t. $G$ with a desired $(\gamma,\epsilon)$-accuracy. In our context, the sampling probabilities that make sense are 1% or below, which are typically supported by modern routers [10].

Fourth, the monitor can define a traffic aggregate without the consent of the domains that carry the aggregate and without even pre-informing these domains what the aggregate is. This is related to unbiasability (goal G3): Each node reports on a sample of packets drawn from *all* the packets it observes, as opposed to reporting only on packets that belong to few specific aggregates of interest. If we reactively configured the network to emit receipts only for an aggregate $G$, e.g., in reaction to user suspicions that $G$ is being throttled, then a dishonest domain could change how it treats $G$ the moment it starts reporting on it. In fact, sampling is the only approach we could think of where each domain reports on a small fraction of the packets it observes, yet the monitor estimates the domain's loss and delay with respect to an aggregate without pre-informing the domain what the aggregate is. In contrast, approaches based on "aggregation" and "sketching" [13, 32, 62] require consent from the involved domains before the collection of statistics can start.

## 2.3 Why consistency?

Compared to random sampling, consistent sampling leads to lower estimation error and provides a good starting point toward incentive-compatible reporting (goal G4):

First, for a given sample size, consistent sampling leads to lower estimation error than independent random sampling (in the weak sense). This happens because random samples are not always "useful" for estimating performance; for example, suppose the extreme case where the sample sets collected by the two consecutive nodes on a flow's path are distinct, i.e. they refer to different packets, then there is no packet for which delay can be computed, and the sample set for estimating statistics is empty. Instead, all consistent samples are useful, because each sample offers information about the loss or delay that is actually experienced between the two nodes. Since the performance of the estimation is directly related to the sample size, it makes sense to use consistent sampling.

Second, independent random sampling enables a certain type of collusion: When each node/domain is allowed to sample different packets, domains have both the incentive and the opportunity to collude and trick the monitor: each domain samples, and produces receipts for, the packets it happened to treat well; this way, all domains exaggerate their performance without implicating each other. The only way to remove the incentive to collude is to expect each domain to sample a specific set of packets, which are also sampled by the domain's neighbors.

Third, consistent sampling enables incentive-compatible reporting on loss events. Consider for example a misbehaving domain, that forges its receipts to pretend that packet losses, w.r.t.

a flow $F$, do not occur in its internal network (while they actually do). If samples are consistent among all nodes along $F$'s path, then the best that the misbehaving domain can achieve with its lies, is to shift the blame for the lost packets to an inter-domain link. Since an inter-domain link is shared responsibility, such fake reporting does not exonerate the culprit; hence there is no incentive for the domain to pursue it. Moreover, the liar will be exposed to the neighbor that was implicated in its lies. In §3.4.2, we will examine closer this issue and leverage this exposure to disincentivize fake receipts of lost packets.

Despite all its advantages, consistent, hash-based sampling is not enough to address all our goals (§1.3); it needs to be equipped with mechanisms that ensure resistance to misbehavior. In the next three chapters, we will enhance consistent sampling with properties that incentivize honest reporting (Chap. 3) and unbiased sampling (Chap. 4), and enable topology obfuscation (Chap. 5).

# 3 Incentive-compatible reporting

In this chapter, we describe how to achieve goal G4: the monitor's decision making should be incentive-compatible. Each domain has an incentive to emit receipts that will lead the monitor to make the "best" decisions for the domain. Therefore, we study "bad receipt" attacks, where a misbehaving domain manipulates receipts in order to exaggerate its performance and/or hide neutrality violations. One might expect that such an attack would trivially succeed (since the monitor has no ground truth beyond the receipts emitted by the domains themselves). We show that this is not the case: by using the proper decision metrics, we ensure that the domains have an incentive to report truthfully. In particular, we design our mechanism such that, to exaggerate its performance, a domain must either falsely blame one of its neighbors and get caught; or appear to violate neutrality (more than it actually does).

We first present the attack and trust model (§3.2); then we formalize the decision-making process by introducing decision and rating functions (§3.3); and finally we describe the tools that enable truthful reporting and our approach (§3.4).

## 3.1 Notation

As stated in the introduction (§1.3), the monitor performs two kinds of processes: statistical estimation, where it estimates loss and delay statistics for which there exists an efficient point estimator (e.g., average loss rate, average delay, delay variance, some important percentile of delay); and decision making, where it uses the estimates to make a decision about a given domain's performance w.r.t. a given aggregate.

Let: $A$ denote the set of possible decisions that the monitor may make about a given domain and a given aggregate; $S_z$ denote the set of all estimates that the monitor computes based on the receipts emitted by domain $z$ for a given aggregate $G$; $S$ denote the set of all estimates that the monitor computes based on the receipts emitted by all the participating domains and for all aggregates.

There are two things to keep in mind about $S$ and $S_z$: First, if $Z$ is the set of all domains, then

$\times_{z \in Z} S_z \subset S$, because $S$ includes estimates computed based on the receipts emitted by domains other than $z$ and for aggregates other than $G$. Second, both sets may include manipulated estimates, because they are based on receipts emitted by the nodes, which may be honest or misbehaving, depending on their chosen strategies.

**Definition 3.1.1.** *A function $f_G : S \rightarrow A$ is called a decision function about aggregate $G$.*

Each decision that the monitor makes about domain $z$ and aggregate $G$ results in a *rating* of domain $z$ w.r.t. $G$, which represents the utility that $z$ gains because of the monitor's decision. In general, rating could be one-dimensional or multi-dimensional, i.e., it could be expressed by either a single number, e.g., the average loss rate or the average delay attributed by the monitor to a domain, or a vector of several numbers e.g., average loss rate, average delay, and number of neutrality violations.

**Definition 3.1.2.** *A function $u_z : S_z \times A \rightarrow \mathbb{R}^n$ is called a rating function of domain $z$.*

**Definition 3.1.3.** *A collection $(f_G, u_1, \ldots, u_Z)$ is called a decision-making mechanism about $G$.*

In addition to $s_z \in S_z$, we denote with $s_{-z} \in (S \setminus S_z)$ any statistical estimate that is not computed exclusively based on $z$'s receipts for $G$, but may also be based on receipts emitted by other domains and for other aggregates. E.g., if $s_z$ is the estimated mean loss rate of domain $z$ w.r.t $G$ (which is computed exclusively based on $z$'s receipts), then $s_{-z}$ may be: the mean loss rate of some other domain w.r.t $G$; the mean loss rate of one of $z$'s inter-domain links w.r.t $G$; $z$'s mean loss rate w.r.t. some aggregate other than $G$. As a result, vector $s \in S$ can be written as $(s_z, s_{-z})$, and the outcome selected by a decision function $f_G$ can be written as $f_G(s_z, s_{-z})$.

Also, let $s$ (resp. $s_z$) denote a statistic when it is truthfully reported, and with $s'$ (resp. $s'_z$) the same statistic when it is manipulated through a bad-receipt attack.

## 3.2 Trust model and sub-problem

In the context of this chapter, the monitor is trusted, whereas the nodes may misbehave by launching bad-receipt attacks. In such an attack, a node may: suppress a receipt, i.e., pretend that it never received a packet that it actually did receive and dropped; emit a superfluous receipt, i.e., pretend that it delivered a packet that it actually dropped; modify a receipt, i.e., pretend that it received a packet later, or delivered a packet earlier than it actually did. We distinguish two kinds of bad-receipt attacks:

- Independent attacks, where the node acts independently from the nodes of other domains, but possibly in cooperation with other nodes of the same domain.
- Collusion attacks, where two nodes that belong to neighbor domains may lie in collusion to improve both of their domains' position compared to other domains.

Given this trust model, we want to design a decision-making mechanism $(f_G, u_1, \ldots, u_Z)$ with the following properties:

1) The mechanism should be *incentive-compatible* under both independent and collusion bad-receipt attacks: the decision and rating functions should be designed such that each domain maximizes its rating by reporting truthfully.

**Definition 3.2.1** (Incentive compatibility)**.** *We call a mechanism $(f_G, u_1, \ldots, u_Z)$ incentive compatible if for every domain $z$, every statistic $s_1 \in S_1, \ldots, s_Z \in S_Z$ and every $s'_z \in S_z$, if we denote $\alpha = f_G(s_z, s_{-z})$ and $\alpha' = f_G(s'_z, s_{-z})$, then $u_z(s_z, \alpha) \geq u_z(s'_z, \alpha')$.*

We want to ensure that every estimate $s_z$ is truthful, but not necessarily every piece of information included in a receipt. This is because the monitor makes its decisions based on the estimates, not on the individual receipts used to compute the estimates. Hence, a mechanism that allows the domains to manipulate individual receipts without affecting the resulting estimates is still incentive-compatible.

2) The rating metric should enable *comparability*, i.e., one should be able to compare and at least partially order the ratings of a set of domains w.r.t. the same aggregate. Hence, we consider one common codomain set for all rating functions $u_z$ that is either one-dimensional and has a total order (in which case, in Def. 3.1.3, we have: $n = 1$ and $\mathbb{R}^n = \mathbb{R}$) or multi-dimensional and equipped with a partial order.

To compare multi-dimensional ratings, we use a Kiviat diagram [39]. When comparing two such ratings, it may happen that none of them is better than the other (e.g., one has a better average loss rate, while the other has a better average delay). However, it is still useful to determine whether a rating is non-dominated, i.e., the monitor did not compute any better rating w.r.t. the given aggregate. When comparing several multi-dimensional ratings, the non-dominated ones are the only ones of interest.

We use the following assumptions (Section §1.5):

First, for any given aggregate $G$ and any given domain $z$ on $G$'s path, each of $z$'s neighbors knows the actual loss and delay performance of the inter-domain link between itself and $z$. This is a reasonable assumption because any domain that is adjacent to an inter-domain link can always directly debug that link and measure its loss and delay performance.

Second, the monitor employs standard statistical techniques to compute the accuracy of each loss or delay estimate; to provide confidence intervals, these techniques must assume something about the nature of the loss/delay that is being estimated, and the typical assumption is that loss/delay is i.i.d. (independent identically distributed) across all packets. This is equivalent to the traffic aggregates having always a small enough size that the packet loss events and/or delays seem independent. We use this assumption to design our neutrality detector (§3.4).

## 3.3 Approach

In this section, we describe our approach. First, we explain our rationale (§3.3.1), which is based on game-theoretic mechanism design [49]. Then, we provide intuition through examples of flawed mechanisms (§3.3.2).

### 3.3.1 Rationale

We formulate the problem using decision and rating functions because of their relevance to game-theoretic mechanism design. Our rating functions are similar to the players' utility functions in game theory. Our decision functions are similar to social choice functions in game theory: a social choice function aggregates the preferences of the different participants toward a single joint decision; similarly, a decision function aggregates the information emitted by the participating domains through their receipts toward a single decision about their performance.

We build on mechanism design with payments [49, 58, 34], but instead of payments we use *rating penalization*. In our context, mechanism design with payments would enforce payments for each domain that do not depend on the information provided by the domain itself. Instead, we design the decision function such that: for every estimate $s_z \in S_z$ that influences domain $z$'s rating and is based on $z$'s receipts, the monitor causes another estimate, which is *not* based on $z$'s receipts, to also influence $z$'s rating. This is done in such a way that misbehavior always decreases a domain's rating.

This is why we defined a rating function of domain $z$ (Def. 3.1.3) to take as input not only $S_z$ (which is based on z's receipts), but also $A$ (the monitor's decision about $z$). This is crucial in our context. Suppose, for a moment, that a rating function of domain $z$ took as input only $S_z$. In that case, one domain's strategy would never affect another domain's rating. Hence, each domain could choose its strategy by solving an independent optimization problem, and the best strategy would always be to misbehave. By making the monitor's decision an input to the rating function, we make it possible to reason about how one domain's strategy affects another domain's rating.

Our challenge is to find rating penalties that make sense in our context and enable incentive compatibility. We leverage two observations related to bad-receipt attacks: First, the only way for a domain to lie about its mean loss rate or mean delay is to falsely attribute loss or delay on one of its inter-domain links, which is shared responsibility with a neighbor. Second, a domain that lies about its delay variance appears to violate neutrality. Before describing our solution, we give intuition through two examples of flawed mechanisms.

### 3.3.2 Examples

**[1] Mean performance + collusion.** Consider the following decision mechanism: The monitor estimates the mean loss rate between each pair of consecutive nodes w.r.t. each aggregate.

Then, as part of the decision function, the monitor determines each domain $z$'s performance w.r.t. an aggregate $G$ as the sum of the estimated mean loss rates of $z$'s intra-domain segment and inter-domain links w.r.t. $G$. For instance, in Fig. 3.1, the monitor would determine $z$'s performance w.r.t. $G$ as the total estimated mean loss rate between nodes $o_{z-1}$ and $i_{z+1}$. When all domains are honest, each domain's rating is equal to its performance as determined by the monitor (because this is the utility that the domain gains from the monitor's decision). When a domain misbehaves independently to harm a neighbor, it ends up with a significantly worse rating: In this mechanism, each domain determines the decision that the monitor makes about its neighbors (e.g., the monitor's decision about domain $z$ depends on the receipts emitted by nodes $o_{z-1}$ and $i_{z+1}$). If a domain lies in a way that causes the monitor to make a worse decision about a neighbor, then it enters a dispute with that neighbor, which is bad for the business of both involved domains. We capture this by assigning a rating of $\infty$ to any pair of neighboring domains that are in dispute.

With this mechanism, a domain that launches an independent bad-receipt attack does not improve its rating; moreover, if the purpose of the attack is to harm a neighbor, the misbehaving domain worsens its rating significantly. For example, consider Fig. 3.1 and suppose that node $o_z$ misbehaves, while all the other nodes are honest. In particular, every time $z$ drops a packet $p$ from aggregate $G$, node $o_z$ pretends that it observed $p$ by issuing a fake receipt for it. This does not help domain $z$: First, the monitor concludes that packet $p$ was dropped on the inter-domain link between $z$ and $z+1$; hence, the attack does not affect the decision that the monitor makes about $z$, while it causes the monitor to make a worse decision about $z+1$. Second, domain $z+1$ realizes that $z$'s exit node is misbehaving and enters a dispute with $z$, causing both domains' ratings to drop significantly.

On the other hand, with this mechanism, a pair of neighboring domains have an incentive to collude and improve both domains' ratings. Consider again Fig 3.1 and suppose that all nodes are honest except for nodes $o_z$ and $i_{z+1}$ that misbehave in collusion in the following way: Node $o_z$ manipulates its receipts to *increase* the estimated mean loss rate of $z$'s intra-domain segment by $\Delta_z$, while node $i_{z+1}$ manipulates its receipts to *also increase* the estimated mean loss rate of $z+1$'s intra-domain segment by $\Delta_{z+1}$. Even though this increases the estimated loss rate of each domain's intra-domain segment, it decreases the estimated loss rate of their inter-domain link by $(\Delta_z + \Delta_{z+1})$. As a result, not only do both domains increase their ratings, but they also do it without affecting any other neighbor, hence without entering any dispute.

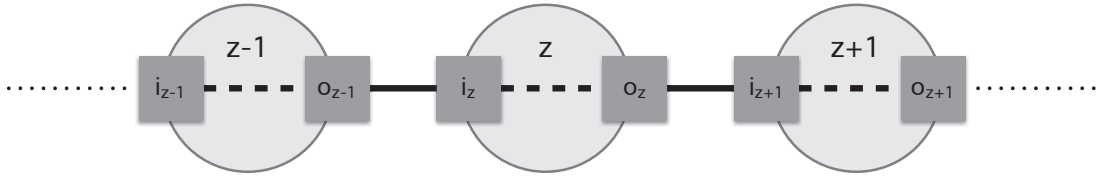



Figure 3.1 – A part of an aggregate's path.

The problem is that summing up the estimated loss rates of overlapping segments creates the opportunity for neighboring domains to increase their ratings by transferring utility between them.

**[2] Performance variance + independent attacks.** Consider a decision mechanism where the monitor estimates the mean delay and delay variance between each pair of consecutive nodes. Then, as part of its decision process, the monitor determines domain $z$'s performance w.r.t. aggregate $G$ as a six-dimensional vector that contains the mean delay and the delay variance of $z$'s intra-domain segment and inter-domain links w.r.t. $G$. The rating function is similar to the one from the previous example: when a domain is not in dispute with any other domain, its rating is equal to its performance (the six-dimensional vector) as computed by the monitor.

With this mechanism, a pair of neighboring domains cannot improve their ratings through collusion. This is achieved simply by replacing the one-dimensional rating of the previous example with a multi-dimensional one, which prevents neighboring domains from transferring utility between them.

On the other hand, with this mechanism, each domain has an incentive to launch an independent bad-receipt attack and claim lower delay variance for its intra-domain segment. For example, consider Fig. 3.1 and suppose that node $o_z$ misbehaves, while all the other nodes are honest. Let $D$ be the true delay of $z$'s intra-domain segment w.r.t. $G$; let $X$ be the true delay of the inter-domain link between $o_z$ and $i_{z+1}$ w.r.t. $G$; and let $D'$ and $X'$ be the corresponding estimated delays (which are affected by $o_z$'s misbehavior). Since all the nodes except for $o_z$ are honest, $D + X = D' + X'$. Node $o_z$ changes the timestamps of its receipts such that: it does not affect any of the estimated mean delays, while $var(D') < var(D)$, $var(X') < var(X)$ and $var(D + X) = var(D' + X')$. This causes the monitor to make better decisions for both $z$ and $z+1$, which means that $z+1$ has no incentive to enter a dispute with $z$, and both domains improve their ratings.

The problem is that, unlike changes in mean performance, changes in performance variance are not always externalizable. As a result, a domain that lies about the performance variance of an intra-domain segment may end up increasing both its own and a neighbor's rating (as in the example above).

## 3.4 Analysis and Solution

In this section, we present our solution. First, we state necessary and sufficient conditions that make a decision mechanism incentive-compatible (§3.4.1). Then, we describe two fundamental trade-offs involved in bad-receipt attacks (§3.4.2, §3.4.3). Finally, we specify a mechanism that leverages these trade-offs to impose penalties that meet the conditions for incentive-compatibility (§3.4.4).

### 3.4.1 Rating penalties

As stated in §3.3.1, we capture the fact that domain ratings depend on the monitor's decisions by making each monitor decision an input to a rating function. To make this dependence explicit, we update our initial notation for the rating functions: instead of the generic notation $u_z\big(s_z, f_G(s_z, s_{-z})\big)$, where $f_G(s_z, s_{-z})$ is the monitor's decision, we will use $u_z(s_z, \pi_z)$, where $\pi_z \in S$ is the penalty that the monitor's decision inflicts on $z$'s rating.

The following theorem adapts mechanism design with payments to mechanism design with rating penalization:

**Theorem 3.4.1** (Proper penalization)**.** *A mechanism $\big(f_G, u_1, \ldots, u_Z\big)$ is incentive-compatible iff:*

*i. The rating function of each domain $z$ includes a penalty that does not depend on $s_z$, but on the decision made by the mechanism $f_G(s_z, s_{-z})$. I.e. for every $s_{-z}$, there exist penalties $\pi_\alpha \in S$, for every $\alpha \in A$, such that for all $s_z$ with $f_G(s_z, s_{-z}) = \alpha$, we have that the rating of $z$ is $u_z(s_z, \pi_\alpha)$.*

*ii. The mechanism optimizes for each domain $z$ when it tells the truth. I.e. for every domain $z$, we have $f_G(s_z, s_{-z}) \in argmax_\alpha(u(s_z, \pi_\alpha))$, where the maximization is over all alternative decisions in the range of $f_G(\cdot, s_{-z})$.*

*Proof.* See section §3.5. □

Theorem 3.4.1 formalizes the simple intuition already discussed in Section 3.3.1: each domain's penalty $\pi_z$ should not depend exclusively on the estimates $s_z \in S_z$ computed from $z$'s receipts (which $z$ could manipulate), but also on other estimates $s_{-z} \in (S \setminus S_z)$.

### 3.4.2 Disputes and externalization

Certain statistics have the "externalization" property: if a domain $z$ launches an independent bad-receipt attack to exaggerate its intra-domain performance w.r.t. such a statistic, then at least one of its inter-domain links will appear worse w.r.t. the same statistic. Mean loss rate is such a statistic, and its externalization was illustrated in Example [I] in Section §3.3.2.

We can leverage externalization to provide a rating with proper penalization (in the sense of Thm 3.4.1): If the monitor determines, as part of a domain $z$'s performance w.r.t. aggregate $G$, not only $z$'s intra-domain performance, but also that of its inter-domain links, then a misbehaving domain's rating is penalized in two ways:

- *Weak* penalization: A bad-receipt attack essentially shifts the blame for some loss or delay from an intra-domain segment to at least one inter-domain link. For instance, if domain $z$ uses bad receipts to reduce its perceived intra-domain mean loss rate w.r.t. $G$

by $\Delta$, it will cause the perceived aggregate mean loss rate of its inter-domain links w.r.t. $G$ to increase by $\Delta$.

Weak penalization makes a misbehaving domain internalize the externality caused by the attack, which means that the domain has no incentive to launch the attack.

- *Strong* penalization: By shifting the blame for loss/delay to an inter-domain link, a domain essentially shifts part of the blame to the neighbor connected through that link (because an inter-domain link is shared responsibility between two domains). Moreover, the falsely blamed neighbor detects the attack and identifies the misbehaving domain (because of the assumption we have made in Section 3.2 that a domain knows the true performance of its inter-domain links). Hence, a domain that launches a bad-receipt attack independently from the neighbor(s) affected by the attack enters a dispute with them. This is strong penalization in the sense that such a dispute with a neighbor has a significant negative impact on a domain's business.

  Strong penalization provides an incentive to domains to not launch bad-receipt attacks.

Let $x_z \in (S \setminus S_z)$ denote the estimate of the same type as $s_z$ that the monitor computes for the inter-domain link that follows domain $z$ on a given aggregate's path. E.g., if $s_z$ is domain $z$'s intra-domain mean loss rate w.r.t. an aggregate $G$, then $x_z$ is the mean loss rate of the inter-domain link between $z$ and $z+1$. Similarly, $x_{z-1} \in (S \setminus S_{z-1})$ denotes the estimate of the same type as $s_{z-1}$ (and $s_z$) that the monitor computes for the inter-domain link that follows domain $z-1$.

**Definition 3.4.1.** *We say that an estimate $s_z$ is "externalizable" when the following holds: if $s'_z \lessgtr s_z$, then $x'_z \geqslant x_z$ or $x'_{z-1} \geqslant x_{z-1}$.*

**Lemma 3.4.2.** *Mean estimates are externalizable; variances and percentiles are not.*

*Proof.* The proof is in §3.5, but a short version is the following: the expectation of the aggregated loss/delay distribution of all three segments that a domain is responsible for (intra- and inter-domain) is a linear operator over the estimates of these segments, whereas variance and percentiles are not. □

**Lemma 3.4.3** (Externalization is sufficient under independent attacks)**.** *There exists a decision mechanism that is based on externalizable statistics and is incentive-compatible under independent bad-receipt attacks.*

*Proof.* See section §3.5. □

**Lemma 3.4.4** (Multi-dimensional rating with partial order is sufficient under collusion)**.** *There exists a decision mechanism that is based on externalizable statistics, uses multi-dimensional rating with a partial order, and is incentive-compatible under colluding bad-receipt attacks.*

*Proof.* See section §3.5. □

The last lemma is related to Example [2] in Section 3.3.2, where two misbehaving domains exploited one-dimensional rating and colluded to improve their ratings. The lemma says that a multi-dimensional rating prevents this type of collusion. The gist of the proof is that having separate rating values for different (intra-domain and inter-domain) segments prevents neighbors from "exchanging" utility.

To summarize: we can design a decision mechanism, where the monitor estimates externalizable statistics (mean loss rate and mean delay) and does so separately about each intra- and inter-domain segment; such that each participating domain has an incentive to emit truthful receipts.

### 3.4.3 Discrimination and internalization

Unfortunately, delay variance is not externalizable, yet it does need to be part of our decision mechanism. First, network users care about it: for a flow packet rate of hundreds of packets per second or less, collecting the sample size necessary to achieve reasonable accuracy takes tens of minutes or more; mean delay over such a long period of time does not provide enough information on its own to assess network performance. Second, knowing the delay variance is necessary for accurately estimating mean delay: recall that we express accuracy through $\gamma$-confidence intervals, which are functions of variance; hence, without knowing the delay variance, the monitor could not compute the accuracy of its mean-delay estimates. In summary, we face the following challenge: mean delays are externalizable, so we could incentivize the domains to report them truthfully; however, to assess the accuracy of each mean-delay estimate, we need to know delay variance, which is not externalizable, hence domains could lie about it.

Our solution relies on the relationship between variance and neutrality. If a network segment does not experience performance variance *and* is neutral, then it has the same mean performance w.r.t. all flows that traverse it. Conversely, a network segment's performance variance and degree of discrimination determine how different its mean performance is w.r.t. the different flows that traverse it. As we will see, because of this relationship, if a domain manipulates its receipts to report lower than its actual performance variance, this is equivalent to reporting more than its actual discrimination, and vice versa. Hence, we can design an incentive-compatible decision mechanism by incorporating both variance and discrimination in the decision and rating functions.

#### Discrimination detection

A straightforward way to detect whether domain $z$ discriminates aggregate $G$ is to compare $z$'s performance w.r.t. two aggregates: $G$ and the aggregate that consists of all traffic that enters

and exits $z$ at the same nodes as $G$; we denote the latter by $Y$. If $z$'s mean loss rates or mean delays w.r.t. $G$ and $Y$ are significantly different, then $z$ discriminates either against or in favor of $G$.

Our system does not enable this kind of detection, because it exposes only the domains' sample mean performance (based on their receipts), which can differ from the true performance due to sampling error. Comparing the sample means without taking into account their accuracy does not make sense.

Let $\mu_Y$ ($\mu_G$) denote domain $z$'s true performance mean w.r.t. aggregate $Y$ ($G$), and let $\sigma_Y^2$ ($\sigma_G^2$) denote $z$'s true variance. Also, let $\hat{\mu}_Y$ ($\hat{\mu}_G$) denote $z$'s sample performance mean w.r.t. aggregate $Y$ ($G$), and let $\hat{\sigma}_Y^2$ ($\hat{\sigma}_G^2$) denote $z$'s sample variance. The sample means and variances are computed based on z's receipts (i.e. a random sample of the relevant traffic aggregate).

We can approximate $\mu_Y$ and $\sigma_Y^2$ with $\hat{\mu}_Y$ and $\hat{\sigma}_Y^2$. This approximation is acceptable because $Y$ consists of all the flows that traverse the corresponding path segment, hence typically has a significantly bigger sample size than the rest of the aggregates.

According to the central limit theorem (CLT), $\hat{\mu}_G$ is distributed around $\mu_G$ according to the normal distribution with variance $\sigma_G^2/N_G$, where $N_G$ is $G$'s sample size[1]. We denote by $f_{\mathcal{N}}$ the density function of the normal distribution $\mathcal{N}\left(\hat{\mu}_Y, \frac{\hat{\sigma}_Y^2}{N_G}\right)$, and by $F_{\mathcal{N}}^{-1}$ its inverse cumulative distribution function.

**Definition 3.4.2** (Neutrality likelihood)**.** *We define the likelihood $L_G$ that $z$ is neutral w.r.t. $G$ as $L_G \equiv f_{\mathcal{N}}(\hat{\mu}_G)$.*

*$L_G$ is simply the likelihood that $\mu_G = \mu_Y$, where $\mu_Y$ is approximated by $\hat{\mu}_Y$.*

**Definition 3.4.3** (Discrimination detector)**.** *We determine, with confidence level $\gamma$, whether domain $z$ discriminates w.r.t. aggregate $G$ based on the following function:*

$$detect_G(\hat{\mu}_G, \hat{\mu}_Y, \hat{\sigma}_Y^2, \gamma, N_G) = \begin{cases} G \text{ was treated worse}, & \text{if } \hat{\mu}_G > F_{\mathcal{N}}^{-1}\left(1 - \frac{1-\gamma}{2}\right) \\ G \text{ was treated better}, & \text{if } \hat{\mu}_G < F_{\mathcal{N}}^{-1}\left(\frac{1-\gamma}{2}\right) \\ G \text{ was treated neutrally}, & \text{otherwise} \end{cases} \tag{3.1}$$

Our detector (Def. 3.4.3 and Fig. 3.2) simply determines whether domain $z$'s sample mean performance w.r.t. aggregate $G$ is "far away" from $z$'s sample mean performance w.r.t. aggregate $Y$ (the rest of the traffic), with confidence level $\gamma$. Conversely, one can view $1-\gamma$ as the significance level of the null hypothesis that $G$ is not neutrally treated.

[1] The use of CLT is subject to our second assumption of Section §3.2: $G$ has a small enough size that the packet loss events and/or delays seem independent; hence, a random sample of $G$ always yields a set of i.i.d. loss/delay values. If the size of $G$ is larger so that the sample loss/delay values can be assumed to be weakly dependent, then the CLT for dependent random variables [37] must be used instead.

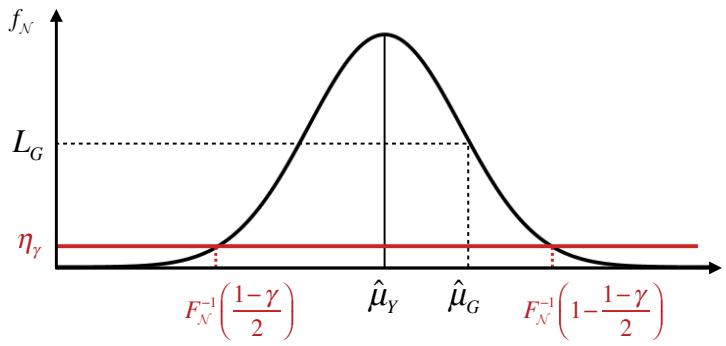



Figure 3.2 – Traffic-discrimination detection.

**Internalization of misbehavior:**

We can leverage the relationship between variance and discrimination to provide a rating with proper penalization (in the sense of Thm 3.4.1): Suppose the monitor includes in its decision about domain $z$ and aggregate $G$, not only z's mean performance w.r.t. $G$, but also: $z$'s sample mean performance and sample variance w.r.t. $Y$ and $z$'s likelihood of discrimination w.r.t. $G$. Then, a misbehaving domain's rating is penalized in two ways:

- *Weak* penalization: Suppose domain $z$ lies about its variance, i.e., manipulates its receipts to artificially decrease its sample variance w.r.t. $Y$; this automatically increases the likelihood that $z$ discriminates w.r.t. $G$. Conversely, suppose $z$ lies about its neutrality, i.e., manipulates its receipts to artificially decrease the likelihood that it discriminates w.r.t. $G$; this automatically increases $z$'s sample variance w.r.t. $G$.

  By including both metrics (sample variance w.r.t. $Y$ and likelihood of discrimination w.r.t. $G$), we prevent a domain from improving its rating by lying about either metric, which means that the domain has no incentive to do so.

- *Strong* penalization: By definition of our discrimination detector, when domain $z$ is honest, it is perceived as discriminating w.r.t. a fraction $\phi_z$ of its aggregates that is always equal to $\gamma$. Conversely, when $z$ lies about its variance, $\phi_z \neq \gamma$. Hence, one can tell that $z$ is lying about its variance simply by comparing $\phi_z$ and $\gamma$. This is strong penalization in the sense that being exposed as lying about variance, especially to hide discrimination, can have a significant impact on a domain's business.

We call the trade-off between variance and discrimination an "internalization," because lying about either one directly affects the lying domain's rating through the other—as opposed to an "externalization," where lying first affects another domain's rating, then indirectly the lying domain's rating through the resulting dispute.

To summarize, we can design a decision mechanism, where the monitor estimates not only externalizable statistics, but also variances; such that each participating domain has an incentive to emit truthful receipts.

### 3.4.4 The mechanism

In Def. 3.4.4, we state our decision-making mechanism $\left(f_G, u_1, \ldots, u_Z\right)$ that uses the externalization and internalization trade-offs to provide incentive-compatibility:

**Definition 3.4.4** (Mechanism)**.** *Given an aggregate G, the monitor attributes a performance to each domain z that observed G, by using all the emitted receipts and the following decision function:*

$$f_G(S) = \begin{cases} z\text{'s intra-domain average performance is } \hat{\mu}_G^{\{0\}} \\ z\text{'s inter-domain average performance is } \left(\hat{\mu}_G^{\{+1\}}, \hat{\mu}_G^{\{-1\}}\right) \\ \sigma_G^{\{0\}} = \hat{\sigma}_Y^{\{0\}} \end{cases} \tag{3.2}$$

*where the superscripts denote z's intra- and inter-domain segment on G's path; i.e.:*

- $\hat{\mu}_G^{\{0\}}$ *is the sample mean loss (resp. delay) w.r.t. G inside z,*
- $\hat{\mu}_G^{\{+1\}}$ *is the sample mean loss (resp. delay) w.r.t. G of the inter-domain link after z,*
- $\hat{\mu}_G^{\{-1\}}$ *is the sample mean loss (resp. delay) w.r.t. G of the inter-domain link before z,*
- $\hat{\mu}_Y^{\{0\}}$ *is the sample mean loss (resp. delay) w.r.t. Y inside z,*
- $\hat{\sigma}_Y^{\{0\}}$ *is the standard deviation of the loss rate (resp. delay) w.r.t. Y inside z.*

*Given the monitor's decisions about all domains and aggregates, z's rating*[2] *becomes:*

$$u_z = \begin{cases} (\infty, \infty, \infty, \infty, \infty, -\infty) & \text{, if } z \text{ is in dispute or } \phi_z \neq \gamma \text{ (strong penalization)} \\ \left(\hat{\mu}_G^{\{0\}}, \hat{\mu}_G^{\{+1\}}, \hat{\mu}_G^{\{-1\}}, \hat{\mu}_Y^{\{0\}}, \hat{\sigma}_Y^{\{0\}}, L_G^{\{0\}}\right) & \text{, otherwise (weak penalization)} \end{cases} \tag{3.3}$$

The mechanism of Def. 3.4.4 is incentive-compatible, because it fulfills the two necessary conditions of Thm. 3.4.1: The rating function of each domain $z$ includes penalties $\pi_z = \left(\hat{\mu}_G^{\{+1\}}, \hat{\mu}_G^{\{-1\}}, \hat{\mu}_Y^{\{0\}}, \hat{\sigma}_Y^{\{0\}}, L_G^{\{0\}}\right)$ that do not depend on $z$'s receipts for $G$, but on the monitor's way of deciding, i.e. the attribution of the average performance of the inter-domain links, and the use of the detection method described in §3.4.3 and the fact that the monitor approximates the variance of $z$'s performance about $G$ with the variance of the performance about the entire traffic, i.e., $\sigma_G^2 = \hat{\sigma}_Y^2$. Note that $\hat{\mu}_G^{\{+1\}}$, $\hat{\mu}_G^{\{-1\}}$, $\hat{\mu}_Y^{\{0\}}$, $\hat{\sigma}_Y^{\{0\}}$ and $L_G^{\{0\}}$ are just special types of $s_{-z}$ that do not depend only on $z$'s receipts about $G$, as opposed to $\hat{\mu}_G^{\{0\}}$. Also, because of strong penalization, the mechanism maximizes each domain's rating when the domain tells the truth about the input statistics.

[2] For the strong penalization of a misbehaving domain, each vector element of its multi-dimensional rating is selected to be either $\infty$ or $-\infty$ according to its nature: e.g., if the vector element denotes a loss rate or delay estimate, then we use $\infty$, because it is worse than any real value; else, if the vector element denotes the neutrality likelihood, then we use $-\infty$, because it is worse than any likelihood value.

Our mechanism addresses our goal G4: it enables domain-performance comparison and traffic-discrimination detection based on truthful statistics. For neutrality checks, the monitor uses the detector in Def. 3.4.3, while for performance comparison of any two domains $z_1, z_2 \in Z$ that are not in dispute and do not violate neutrality (i.e. $\phi_z = \gamma$), it uses a partial order and Kiviat diagrams [39] to compare the rating vectors $u_{z_1}$ and $u_{z_2}$.

## 3.5 Proofs

### Theorem 3.4.1

*Proof.* ["if" part:] Let $\alpha = f_G(s_z, s_{-z})$, $\alpha' = f_G(s'_z, s_{-z})$, and consider the corresponding penalties: $\pi_\alpha$ and $\pi'_\alpha$. The rating of $z$ when telling the truth about $s_z$ is therefore $u_z(s_z, \pi_\alpha)$, while when misbehaving it is $u_z(s'_z, \pi'_\alpha)$. Then, since the mechanism optimizes for $z$, i.e. $\alpha = f_G(s_z, s_{-z}) \in argmax_\alpha(u(s_z, \pi_\alpha))$, we have that $u_z(s_z, \pi_\alpha) \geq u_z(s'_z, \pi'_\alpha)$.

The "only if" part is proved through contradiction:

["only if" part – item (i):] Assume that for some $s_z$ and $s'_z$, we have $f_G(s_z, s_{-z}) = f_G(s'_z, s_{-z}) = \alpha$, but $u_z(s_z, \pi_\alpha) \leq u_z(s'_z, \pi_\alpha)$, then $z$ could increase its rating by declaring $s'_z$ instead of $s_z$ (untruthful reporting).

["only if" part – item (ii):] Assume that $f_G(s_z, s_{-z}) \notin argmax_\alpha(u(s_z, \pi_\alpha))$ and let some other decision $\alpha'$ maximize $z$'s rating in the range of $f_G(\cdot, s_{-z})$: i.e. $\alpha' \in argmax_\alpha(u(s_z, \pi_\alpha))$. Thus, for some $s'_z$, $\alpha' = f_G(s'_z, s_{-z})$, which means that $z$ could increase its rating by declaring $s'_z$ instead of $s_z$ (untruthful reporting). □

### Lemma 3.4.3

*Proof.* Lemma is proved if we find a rating function with the conditions of Thm. 3.4.1. A mechanism $(f_G, u_z)$, where the monitor computes each domain's performance as a vector containing the mean (loss rate or and/or delay) performance in its intra- and inter-domain segments, considers all three segments for which $z$ is responsible by construction. Denote with $s_z$ the statistic of interest. The rating of each domain can be the following vector: $u_z = (s'_z, x'_z, x'_{z-1})$. The rating includes statistics that do not depend only on $z$'s receipts: $x'_z, x'_{z-1} \in S \setminus S_z$. Also, without loss of generality suppose that $z$ reports $s'_z < s_z$, which implies that at least one of the two holds, $x'_z > x_z$ or $x'_{z-1} > x_{z-1}$. Then, $(s_z, x_z, x_{z-1})$ is not dominated by $(s'_z, x'_z, x'_{z-1})$, which means that $z$ does not increase its rating by lying. Moreover, $z$ is exposed to at least one neighbor, which results in strong penalization. In this case, the monitor can assign to $z$ a very bad rating, e.g. $u_z = (\infty, \infty, \infty)$, which makes the mechanism optimize for $z$ only when $z$ reports truthfully, because $(s_z, x_z, x_{z-1}) < (\infty, \infty, \infty)$. □

### Lemma 3.4.2

*Proof.* For the proof of this lemma, we will abuse notation and let $s_z$, $x_z$ and $x_{z-1}$ denote not the statistics of the intra-domain and inter-domain segments, but the loss/delay distributions of the corresponding segments:

[Mean:] In Fig. 3.1, let domain $z$ lie about the mean loss/delay of its internal network by manipulating the receipts of its exit node, i.e. it reports $\mathbb{E}\left[s'_z\right] \neq \mathbb{E}\left[s_z\right]$; while the entry node of $z+1$ reports truthfully[3]. Then, since $\mathbb{E}\left[s_z + x_z\right] = \mathbb{E}\left[s'_z + x'_z\right] \Leftrightarrow \mathbb{E}\left[s_z\right] + \mathbb{E}\left[x_z\right] = \mathbb{E}\left[s'_z\right] + \mathbb{E}\left[x'_z\right]$, we have: $\mathbb{E}\left[s'_z\right] < \mathbb{E}\left[s_z\right] \Rightarrow \mathbb{E}\left[x'_z\right] > \mathbb{E}\left[x_z\right]$. Similarly, $\mathbb{E}\left[s'_z\right] > \mathbb{E}\left[s_z\right] \Rightarrow \mathbb{E}\left[x'_z\right] < \mathbb{E}\left[x_z\right]$.

[Variance:] As previously, let $z$ manipulate the loss/delay variance through the receipts of its exit node, i.e. it reports $var\left[s'_z\right] \neq var\left[s_z\right]$, while the entry node of $z+1$ reports truthfully. Then, since $var\left[s_z + x_z\right] = var\left[s'_z + x'_z\right] \Leftrightarrow var\left[s_z\right] + var\left[x_z\right] + 2cov\left[s_z, x_z\right] = var\left[s'_z\right] + var\left[x'_z\right] + 2cov\left[s'_z, x'_z\right]$, externality is not always possible: for a given $cov\left[s_z, x_z\right]$, that is unknown to the monitor, $z$ can manipulate $cov\left[s'_z, x'_z\right]$, so that $var\left[s'_z\right] < var\left[s_z\right]$, and at the same time $var\left[x'_z\right] < var\left[x_z\right]$.

[Percentiles:] A domain can report a lower percentile, by only decreasing its intra-domain variance and without changing the reported intra-domain mean. Since variance is not externalizable, percentiles are not either. □

### Lemma 3.4.4

*Proof.* Let $s_z$ be the mean intra-domain loss/delay estimate and $s_z, x_z, x_{z-1}$ be the mean inter-domain loss/delay estimates. Consider a mechanism where the monitor computes each domain's performance as a vector containing the mean (loss rate or delay) performance in its intra- and inter-domain segments; hence by construction takes into account all three segments for which $z$ is responsible. Let $z$ and $z+1$ collude and report $s'_z = s_z + \Delta_z$ and $s'_{z+1} = s_{z+1} + \Delta_{z+1}$, so that $x'_z = x_z - \Delta_z - \Delta_{z+1}$, where $\Delta_z, \Delta_{z+1} \geq 0$. Then, because of the partial order, $z$'s rating $u_z(s'_z, x'_z, x'_{z-1}) = u_z(s_z + \Delta_z, x_z - \Delta_z - \Delta_{z+1}, x'_{z-1})$ does not dominate the rating that it would have if it reported truthfully, i.e. $u_z(s_z, x_z - \Delta_{z+1}, x'_{z-1})$. □

[3] This is without loss of generality, because manipulating the receipts of $z$'s entry node is just the dual problem towards its other neighbor $z-1$.

# 4 Unbiased reporting

In this chapter, we describe how we achieve goal G3: the monitor's estimation should be unbiasable. When a domain's performance is estimated based on how it treats a small sample of forwarded packets, the domain has an incentive to bias the sample—treat the sampled packets better than the rest—resulting in arbitrarily inaccurate estimation of its performance. Therefore, we study "prioritization attacks", where the domains emit truthful receipts, yet prioritize sampled packets to claim lower loss or delay. In the face of such attacks, basic consistent sampling (§2.1) is not enough: the nodes can easily determine whether a packet is sampled upon its arrival and treat it accordingly. Instead, we propose *retro-active* packet sampling, where the sampling function is keyed on subsequent traffic, making the samples unpredictable.

We first present our trust model (§4.1). Then, we describe our algorithm (§4.2) and sketch one possible hardware implementation (§4.3). Finally, we analyze the attack resistance (§4.4), accuracy (§4.5), and resource requirements (§4.6) of our algorithm and confirm them with an experimental evaluation (§4.7).

## 4.1 Trust model and sub-problem

In the context of this chapter, the monitor is trusted, while a node (and the domain that owns the node) may launch a "prioritization attack": it emits truthful receipts, but treat some or all of the sampled packets preferentially (e.g., by assigning them to higher-priority queues or routing them through a better intra-domain path), thereby introduce sampling bias to exaggerate the domain's performance.

Given this trust model, we want to design a sampling algorithm that: (a) yields an unbiased, representative sample based on which the monitor can compute estimates with a desired $(\gamma,\epsilon)$-accuracy in a desired time interval $T$, while (b) it does not require an increase in the node's data-to-control-path bandwidth or data-path memory by more than a few percentage points.

| Symbols | |
|---|---|
| $p$ | A packet |
| $d$ | A disclosure packet |
| $F$ | A flow |
| **Workload characteristics** | |
| $r$ | Packet arrival rate of a flow at a node |
| $R$ | Total packet arrival rate at a node |
| **Algorithm parameters** | |
| $\beta$ | The size of the receipt buffer |
| $\kappa$ | The duration of the quiet period |
| $\delta_r$ | Disclosure rate: prob. that a packet from a flow with packet rate $r$ is picked as a disclosure packet |
| $\sigma$ | Selection rate: prob. that a non-excluded packet is sampled |

Table 4.1 – Parameters and symbols

We restate the following assumption (§1.5):

The packet arrivals of each flow form a stationary and ergodic process with a high enough rate (e.g. at the order of OC-12 or higher) that ergodic convergence is achieved in less than 100msec. The limitation resulting from this assumption is that we cannot reason about aggregates that consist of relatively few packets/sec, e.g., an aggregate that consists of a typical TCP flow. We think that this is acceptable given our motivation to enable the comparison of domain performance and the verification of neutrality regulations, which typically apply to relatively large aggregates. This assumption is used for the proofs of Lemmas 4.4.1 and 4.5.1.

## 4.2 Solution

We now describe our algorithm, which we call "retroactive sampling": first the parts that are adapted from prior work (§4.2.1), then the novel parts (§4.2.3) and the rationale behind them (§4.2.4).

Table 4.1 states all the relevant symbols.

### 4.2.1 Basic Delayed Disclosure

The algorithm (Alg.2) takes as input the sequence of packets arriving at a node and outputs a sequence of receipts to be exported to the monitor. Each node maintains a circular "receipt buffer" of size $\beta$, where it adds a receipt for every new packet (lines 1,2). Moreover, from each observed flow $F$, the node picks some packets that act as "disclosure packets"; these special packets determine the sampling fate of the previously observed packets from $F$. For example, in Fig. 4.1, disclosure packet $d_1$ determines that, among previously observed packets $p_0 \dots p_9$, only $p_1$ and $p_4$ will be sampled.

**Algorithm 2** RetroActiveSampling ($p$)

| | |
|---|---|
| $\hat{p}$ | non-mutable content of packet $p$ |
| *Receipt*() | constructs a receipt |
| *PacketRate*() | computes packet rate |
| *DiscHash*() | hash function |
| *DiscRange*() | subset of *DiscHash*'s range |
| *Hash*() | hash function |
| *Range* | subset of *Hash*'s range |

```
1:  rec' ← Receipt(p, currentTime)
2:  Add rec' to receipt buffer.
3:  r ← PacketRate(rec'.flowID)
4:  if DiscHash(p̂) ∈ DiscRange(r) then
5:      Emit receipt rec'.
6:      if LateDisclosure(flowID) then
7:          Emit warning.
8:      end if
9:      for all receipts rec in receipt buffer with
10:         rec.flowID = rec'.flowID do
11:         Remove rec from receipt buffer.
12:         if (currentTime − rec.timestamp) ≤ κ then
13:             continue
14:         end if
15:         if Hash(rec.digest, rec'.digest) ∈ Range then
16:             Emit receipt rec.
17:         end if
18:     end for
19: end if
```

More specifically: When a new packet $d$ arrives, the node picks $d$ as a disclosure packet with some probability, by applying a hash function with strong randomization properties on $d$'s non-mutable content (line 4). If $d$ is picked as a disclosure packet, the node samples $d$ (line 5), removes from the receipt buffer all receipts with the same *flowID* as $d$ (lines 9–11), and picks some of them for sampling (lines 15, 16).

One key point is that, whether a packet $p$ is sampled or not depends on the next disclosure packet $d$ from the same flow as $p$ that arrives at the node; we say that "disclosure for $p$ happens" when $d$ arrives at the node, and that $d$ is "$p$'s disclosure packet."

Another key point is that disclosure packets are normal packets that happen to be picked by the nodes to play a particular role in the sampling algorithm; disclosure packets are *not* explicitly labeled by anyone, nor explicitly introduced into the traffic stream by the monitor, the nodes, or the traffic sources. If two nodes observe the same flow, they pick the same packets as disclosure packets (modulo loss), by virtue of using the same function on line 4.

### 4.2.2 Late Disclosure

We say that "packet $p$ suffers late disclosure" at a node, when $p$'s disclosure packet arrives at the node after $p$'s receipt has been overwritten in the circular receipt buffer (in which case lines 9 to 18 are never executed for $p$). For example, suppose that, in Fig. 4.1, by the time disclosure packet $d_1$ arrives at the node, packet $p_1$'s receipt has been overwritten; in this case, $p_1$ suffers late disclosure.

When late disclosure occurs at a node, the node warns the monitor: When a node picks packet $d_1$ from flow $F$ as a disclosure packet, it checks whether the receipt for the previous disclosure packet, $d_0$, from $F$ is still in the buffer (line 6); if yes, it is certain that none of $F$'s packets that arrived at the node between $d_0$ and $d_1$ suffered late disclosure. if no, the node emits a "late disclosure" warning (line 7), which references the earliest of $F$'s packets whose receipt is still in the buffer. Back to our last example: When $d_1$ arrives at the node, the node detects that $d_0$'s receipt is not in the buffer, and that the earliest of $F$'s packets whose receipt is still in the buffer is $p_2$. Hence, the node emits a late-disclosure warning that references $p_2$, which indicates that $F$'s packets that arrived at the node between $d_0$ and $p_2$ may have suffered late disclosure.

### 4.2.3 Quiet Periods and Adaptive Disclosure

We want disclosure to happen too late to be useful to a misbehaving node/domain; the challenge lies in achieving this goal for the millions of flows observed by each node, without violating the simplicity of basic delayed disclosure.

First, we impose a "quiet period" of duration $\kappa$ before each disclosure packet, during which no packets from the same flow can be sampled. More specifically, if $p$'s disclosure packet arrives less than $\kappa$ time units after $p$, the node explicitly excludes $p$ from sampling (lines 12,13). For example, in Fig. 4.1, the node excludes packets $p_6$ to $p_9$ from sampling, because they arrive during the quiet period that precedes their disclosure packet $d_1$.

Second, we adapt the disclosure process to each flow's packet rate: When a new packet $p$ arrives, the node roughly estimates $F$'s packet rate $r$ (line 3) and picks $p$ as a disclosure packet with a probability $\delta_r$ that depends on $r$. The nodes track flow packet rates without maintaining explicit per-flow state. A separate lookup table that maintains per-flow packet rates is out of the question, as it would introduce precisely the kind of expensive state we have been trying to avoid with our design. In the Appendix, Section 4.3, we describe a hardware-friendly implementation that does not keep explicit per-flow state.

When disclosure for packet $p$ happens neither early nor late, the node picks $p$ for sampling with a fixed probability $\sigma$, by applying a hash function with strong randomization properties on $p$'s and its disclosure packet's non-mutable contents (lines 15, 16). For example, in Fig. 4.1, after processing disclosure packet $d_1$, the node samples previously observed packets $p_1$ and $p_4$.

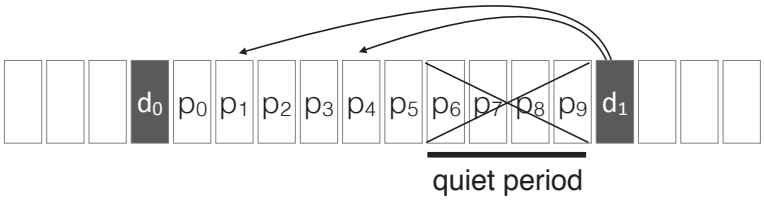



Figure 4.1 – Delayed disclosure and exclusion.

### 4.2.4 Rationale

Our algorithm is the result of combining delayed disclosure with consistent sampling: The former requires that each packet $p$'s sampling fate be determined by a subsequent disclosure packet; the latter requires that all nodes that observe $p$ make the same sampling decision about it. Combining the two, $p$ and its disclosure packet should ideally traverse the same nodes, i.e., the same inter-domain path. In the current Internet architecture, there is no way to guarantee this; the best indication a node has that two arriving packets follow the same inter-domain path is that they share the same flow—the same source and destination IP prefix—which is why nodes pick $p$'s disclosure packet from the same flow.

Given that our algorithm's decisions are affected by local traffic, the nodes may, occasionally, sample inconsistently (e.g., due to late disclosure); when this happens, the monitor identifies and discards the inconsistent samples. So, inconsistent sampling affects efficiency (it causes the monitor to collect fewer useful samples), but not correctness (it does not cause the monitor to make incorrect estimates).

Quiet periods make prioritization attacks more expensive by ensuring that the decision to sample a packet $p$ is disclosed within time $\kappa$ from $p$'s observation only for a limited number of packets that is configurable through the arrival probability of the disclosure packets or differentially the disclosure rate $\delta$. As we will see, by tuning $\kappa$ and $\delta$, we can control how much a misbehaving domain worsens its perceived delay performance.

The adaptation of the disclosure process to each flow's packet rate regulates early and late disclosure and ensures that each node produces enough receipts from each flow to enable accurate statistics. Both early and late disclosure exclude packets from sampling; hence, if we want a node to sample packets from a flow $F$ at some minimum rate, we have to ensure that enough packets from $F$ escape early and late disclosure. The probability of early disclosure depends on the interplay between $F$'s packet rate $r$, the disclosure rate $\delta$, and the quiet-period duration $\kappa$; this interplay determines how many of $F$'s packets fall into a quiet period. The probability of late disclosure depends on the interplay between $r$, $\delta$, the rest of the traffic arriving at the node, and the size of the receipt buffer $\beta$; this interplay determines how quickly the receipt buffer fills up and how many of $F$'s receipts are overwritten prematurely. So, to keep the probabilities of early and late disclosure at a desired value, we have to quantify the above interactions and continuously adapt the disclosure process to $F$'s packet rate—this is

why nodes track flow packet rates and why $\delta$ is a function of $r$.

## 4.3 Sketch of a Hardware Design

Our proposed hardware implementation consists of two parallel threads: a "producer" thread that processes incoming packets, computes receipts, and adds them to the local state; and a "consumer" thread that removes receipts from the local state and determines whether to emit or discard each receipt. The local state, however, is organized differently: there is a receipt buffer for "normal" packets, stored in SRAM and accessible only as a queue; and a receipt buffer for disclosure packets, stored in content-addressable memory (CAM) that supports $O(1)$ parallel lookups on the *flowID* field.

Organizing state in separate receipt buffers for normal and disclosure packets (as opposed to a common receipt buffer) does not introduce any extra or any per-flow state; we keep exactly as many receipts for normal and disclosure packets as Alg. 2, we just store them in two different kinds of memory, because receipts for disclosure packets need to be looked up by *flowID* (hence require CAM), whereas receipts for normal packets, which constitute the vast majority, do not (hence can be stored in cheaper SRAM).

The consumer thread performs the actual sampling: It iterates over the receipt buffer for normal packets, removes receipts, and performs the following operations for each receipt *rec* that corresponds to a normal packet $p$: (a) It retrieves the set of all receipts $\mathscr{R}$ in the disclosure buffer with the same *flowID* as $p$. (b) It finds in $\mathscr{R}$ the receipt $rec'$ that corresponds to $p$'s disclosure packet: it is the one with the earliest timestamp that satisfies $rec'.timestamp > rec.timestamp$. It increments $rec'.packetCtr$. It determines whether to exclude or select $p$ according to lines 10 and 12 of Alg. 2. (c) It checks whether disclosure may have occurred late for $p$, i.e., if there is no receipt in $\mathscr{R}$ with $timestamp < rec.timestamp$. If so, it emits a warning.

The producer thread creates the local state: It processes each incoming packet $p$, computes a receipt $rec'$ for it, extracts the *flowID*, retrieves $F$'s packet rate, checks whether to pick $p$ as a disclosure packet according to line 4 of Alg. 2, and adds $rec'$ to the appropriate receipt buffer.

Moreover, the producer thread tracks flow packet rates based on information that is piggybacked by the consumer thread on the disclosure receipts. We previously said that receipts consist of three fields (§2.1); receipts for disclosure packets have two additional fields for packet-rate tracking, *packetRate* and *packetCtr*. Consider a flow $F$ and suppose the first disclosure packet $d_1$ arrives at time $t_1$; when the second disclosure packet $d_2$ arrives at time $t_2$, the producer thread retrieves $d_1$'s receipt *rec*, estimates $F$'s packet rate (as *rec.packetCtr* divided by $t_2 - t_1$), and stores the estimate in $d_2$'s receipt (in the *packetRate* field); when the third disclosure packet $d_3$ arrives, the node repeats the process, computes a new packet-rate estimate, and stores it in $d_3$'s receipt; and so on. The node may compute a brand new estimate every time a disclosure packet arrives, or it may update the previous estimate, akin to keeping a moving-average filter [38]; we choose the latter, because it ensures a smooth variation of

estimate without risking sharp increases in rare cases where disclosure packets arrive very close to each other.

We do not need perfect packet-rate tracking: Suppose a flow's real packet rate is $r$, while the estimated one is $\hat{r}$. If $r$ and $\hat{r}$ are in the same zone (map to the same disclosure rate), the disclosure process works as intended; if they map to different disclosure rates, then, of course, it is possible that the chosen disclosure rate is too high or too low for the real packet rate, and we miss the desired accuracy until the estimate catches up with the real value. The fewer disclosure-rate values we use, the less likely we are to hit this scenario, and this is why we designed our disclosure process to operate with smallest number of disclosure-rate values.

With this design, each node performs per packet: two hash computations, a couple of timestamp comparisons, a read and a write access to SRAM, and a parallel lookup and an update in the CAM.

## 4.4 Misbehavior Analysis

In this section, we analyze our algorithm's behavior in the face of prioritization attacks: we present our attack model (§4.4.1), then prove that it is possible to parametrize our algorithm such that any prioritization attack is ineffective (§4.4.2).

### 4.4.1 Attack Model

In a prioritization attack, the misbehaving node runs the algorithm, but also buffers each arriving packet $p$ for a maximum "buffering period" $t \geq 0$ before forwarding it, with the hope that it will learn some information about $p$'s sampling fate. The attack is successful when the benefit gained from learning this information outweighs the cost of buffering—hence delaying—packets.

More concretely, consider domain $x$ in Fig. 4.2 with an entry node $i$ and an exit node $o$, and two routes between $i$ and $o$: a "good" route with mean loss and delay $\{l_g, d_g\}$, and a "bad" route with mean loss and delay $\{l_b \geq l_g, d_b \geq d_g\}$. We consider one flow $F$ that enters $x$ at node $i$ with packet rate $r$ and exits at node $o$. Of all the packets arriving at $i$, the maximum fraction that can be forwarded over the good route is $g$. Node $i$ forwards a fraction $g \in [0,1)$ of $F$'s packets over the good route and the rest over the bad route. Upon receiving a packet $p$, node $i$ runs Alg. 2, then Alg. 3: It buffers $p$ for a maximum period $t \geq 0$ (line 1). If disclosure occurs while $p$ is buffered (lines 2,3), and it occurs early or determines that $p$ will not be sampled (line 4), then $i$ forwards $p$ over the bad route (line 5), unless it has already exhausted the bad route's capacity (line 6). If disclosure occurs while $p$ is buffered, does not occur early, and determines that $p$ will be sampled (line 7), then $i$ forwards $p$ over the good route (line 8), unless it has already exhausted the good route's capacity (line 9). Finally, if the buffering period runs out before disclosure occurs, then $i$ forwards $p$ over the good or the bad route (line 13) based on

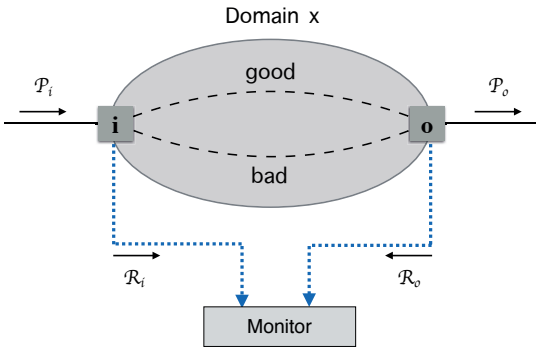



Figure 4.2 – Example.
$\mathscr{P}_i$ and $\mathscr{P}_o$ are the packet streams observed by nodes $i$ and $o$,
$\mathscr{R}_i$ and $\mathscr{R}_o$ are the sampled packet receipts of nodes $i$ and $o$.

| **Attack parameters (unknown to us)** | |
|---|---|
| $\{l_g, d_g\}$ | Loss/delay of good route |
| $\{l_b, d_b\}$ | Loss/delay of bad route |
| $g$ | Fraction of traffic sent over good route |
| $t$ | Buffering period |
| **Symbols used in analysis** | |
| $\{\hat{l}_{hon}, \hat{d}_{hon}\}$ | Loss/delay estimates if node $i$ is honest |
| $\{\hat{l}_{mis}, \hat{d}_{mis}\}$ | Loss/delay estimates if node $i$ misbehaves |
| $\hat{l}_{hon} - \hat{l}_{mis}$ | Loss benefit |
| $\hat{d}_{hon} - \hat{d}_{mis}$ | Delay benefit |

Table 4.2 – Attack parameters and symbols.

an arbitrary forwarding strategy (as long as, in the end, $i$ forwards a fraction $g$ of $F$'s packets over the good route and the rest over the bad route).

By varying the parameters $t$ and $g$, we capture all rational behaviors of node $i$ that involve no more than two priority levels: (a) Honest behavior w/o prioritization: $t = 0$ and $g = 0$ ($i$ does not buffer and forwards all packets on the same route). (b) Honest behavior w/ prioritization: $t = 0$ and $g > 0$ ($i$ does not buffer and uses both routes). (c) Misbehavior: $t > 0$ and $g > 0$ ($i$ buffers and uses both routes). For any behavior where node $i$ uses more than two priority levels (more than two routes), it is trivial to show that there exists a behavior where $i$ uses only two priority levels (only the best and worst of its routes) and domain $x$ achieves at least the same perceived performance. Hence, if we prove that any prioritization attack captured by the above model is unsuccessful, then we have also proved that any prioritization attack that uses more priority levels is unsuccessful.

The monitor estimates $x$'s loss and delay with respect to $F$. When node $i$ is honest, the expected values of the mean loss and mean delay estimates are:

$$\hat{l}_{hon} = g \cdot l_g + (1-g) \cdot l_b, \quad \hat{d}_{hon} = g \cdot d_g + (1-g) \cdot d_b,$$

**Algorithm 3** PrioritizationAttack ($p$)

```
1: buffer p, start timer tm
2: while tm < t do
3:     if disclosure for p then
4:         if tm < κ then
5:             forward p over bad route
6:         else if p sampled then
7:             forward p over good route
8:         else
9:             forward p over bad route
10:        end if
11:    end if
12: end while
13: forward p randomly over good or bad route
```

since a fraction $g$ of the traffic is forwarded over the good route, and the rest over the bad route. When node $i$ misbehaves, we denote the estimated mean loss and mean delay by $\hat{l}_{mis}$ and $\hat{d}_{mis}$, respectively.

We define the attack's "loss benefit" as $\hat{l}_{hon} - \hat{l}_{mis}$, and its "delay benefit" as $\hat{d}_{hon} - \hat{d}_{mis}$. These numbers quantify how much $x$ exaggerates its perceived loss and delay performance by misbehaving, taking into account that misbehavior involves buffering, which necessarily delays packets.

It is trivial to show that the attack's loss benefit is always positive: The longer node $i$ buffers packets before forwarding them, the bigger the fraction of packets whose sampling fate is disclosed to $i$ before forwarding. Ultimately, if $i$ does not mind introducing extra delay, it can buffer every single packet long enough to learn its sampling fate and forward all sampled packets over the good route, and all non-sampled packets over the bad route. This would result in estimated mean loss rate $\hat{l}_{mis} = l_g$ and loss benefit $\hat{l}_{hon} - \hat{l}_{mis} = (1-g)(l_b - l_g)$.

The interesting question is what happens with the attack's delay benefit: under what conditions does the cost of buffering packets outweigh the benefit of sending some of the sampled packets over the better route?

A clarification: Our attack model involves one good and one bad route, but the exact same results can be achieved by considering multiple routes, of which the "best" has the same features as our good route, and the "worst" the same features as our bad route. The gist is that the misbehaving node $i$ has a cheating opportunity because it can forward sampled packets over a better route than non-sampled packets.

### 4.4.2 Conditions for Resistance

**Lemma 4.4.1.** *Let packets arrive as a stationary and ergodic process, disclosure arrivals form a Bernoulli random process on the top of it, and quiet periods be long enough for the ergodicity theorem to hold (i.e., assumptions stated in §1.5 hold). If node $i$ attempts to predict the probability that each new packet from flow $F$ will be sampled, then its best prediction (given a perfect estimate of $F$'s packet rate $r$) will be the same for all new packets from flow $F$.*

*Proof.* See section §4.7.5. □

Lemma 4.4.1 says that, when packet $p$ arrives at node $i$, the probability of $p$ being sampled is the same as for any other newly arrived packet. Hence, node $i$ can have no probabilistic benefit from treating $p$ preferentially, unless it buffers $p$ for $t > 0$ and gains information on $p$'s sampling fate from subsequent packet arrivals.

**Lemma 4.4.2.** *If both of the following conditions hold:*

$$\kappa > d_b - d_g; \tag{4.1}$$

$$(1-\delta_r)^{r\kappa} \geq \frac{1}{e}; \tag{4.2}$$

*then:*

$$\hat{d}_{hon} - \hat{d}_{mis} < 0; \tag{4.3}$$

$$\frac{d\{\hat{d}_{hon} - \hat{d}_{mis}\}}{dt} \leq \frac{d_b - d_g}{\kappa} - 1. \tag{4.4}$$

*Proof.* See section §4.7.6. □

Lemma 4.4.2 says that, if we set $\kappa$ and $\delta_r$ such that Inequalities 4.1 and 4.2 hold, then: (a) The attack's delay benefit is always negative (Inequality 4.3), which means that the misbehaving domain $x$ always worsens its perceived delay performance. (b) The attack's delay benefit always decreases as the buffering period $t$ increases (Inequality 4.4), which means that the longer node $i$ buffers packets to gain information about their sampling fate, the worse $x$'s perceived delay performance becomes. In particular, if $\kappa \gg d_b - d_g$, then $\frac{d\{\hat{d}_{hon} - \hat{d}_{mis}\}}{dt} \to -1$, which means that: for every msec that node $i$ buffers a packet, it worsens $x$'s perceived delay performance by almost one msec. In our opinion, given the importance of delay for modern applications, no domain would choose to penalize its perceived delay performance this way in order to exaggerate its loss performance.

We now provide intuition behind the two lemmas:

The intuition behind Lemma 4.4.1 is that the sampling fate of a new packet $p$ depends only on future events that are unpredictable at the moment of $p$'s arrival. In particular, packet

$p$ is sampled when two events happen: (1) the if-statement on line 15 of Alg. 2 is true and (2) disclosure for $p$ happens neither early nor late. Event (1) always happens with a fixed probability $\sigma$. Event (2) depends on when (after how many other packets) $p$'s disclosure packet will arrive; this "distance-to-disclosure" follows a geometric distribution with parameter $\delta_r$[1], hence is memoryless, i.e., does not depend on $p$'s time of arrival or any prior packet arrivals. The best node $i$ can do is track $F$'s packet arrivals, predict its average packet rate $r$ until the next disclosure packet, and compute an expectation of when Event (2) will occur. The result, however, is the same for all new packets from $F$, hence node $i$ cannot tell whether one new packet is more likely to be sampled than another.

The first condition of Lemma 4.4.2 is straightforward: the quiet period should be longer than the delay difference between the bad and good route within the misbehaving domain. The longer the quiet period, the longer node $i$ needs to buffer $p$ before learning its sampling fate. By making the quiet period longer than the good-bad route delay difference, we ensure that $i$ cannot compensate for the buffering delay by sending $p$ over the good route.

The second condition, however, is more subtle and something we did not expect—it emerged from the math: the average fraction of packets that do *not* suffer early disclosure should be at least $1/e$. If disclosure for a packet $p$ happens early (i.e., during a quiet period), node $i$ learns that $p$ will not be sampled and cheats by forwarding $p$ over the bad route. Increasing the quiet-period duration does not help control this event—quite the contrary: longer quiet periods lead to more early-disclosure events and more opportunities for this kind of cheating. Ultimately, the amount of packets that suffer early disclosure could become large enough to saturate the bad route; in this case, by forwarding all packets that suffer early disclosure over the bad route, node $i$ ends up forwarding all packets that do *not* suffer early disclosure—which include all the sampled packets—over the good route. The second condition prevents this scenario. This being a sufficient (not necessary) condition, we do not have an intuitive explanation for why the $1/e$ bound in particular works; it is the tightest bound we could find that—together with the first condition—enabled us to prove that the delay benefit is always negative, but a tighter bound may exist.

## 4.5 Accuracy Analysis

To estimate a domain's performance w.r.t. aggregate $G$, the monitor first picks a target $(\gamma,\epsilon)$-accuracy, then computes the sample size $N$ (number of $G$'s receipts) necessary for achieving it. For this estimation, the monitor uses well-established statistical methods that assume either i.i.d. loss of minimum rate $loss_{min}$ or Gilbert loss of minimum rate $loss_{min}$ and maximum burst size $burst_{max}$ [36]. The parameters of the loss model depend on the scenario: When estimating the loss of a domain that promises maximum loss $\ell$, $loss_{min}$ should be set to

[1] If flow $F$ has packet rate $r$, each packet from $F$ is chosen as a disclosure packet with the same probability $\delta_r$; hence, the arrivals of $F$'s disclosure packets form a standard Bernoulli stochastic process that is renewed at each packet arrival from $F$.

$\leq \ell$; the typical loss rate mentioned in today's SLAs is 0.1% [11, 3, 8], so this is the default value we use in our examples. When assuming Gilbert loss, we use $burst_{max} = 2$, because we experimentally found that this value is conservative even for highly congested environments.

The monitor can achieve any target accuracy as long as it waits long enough to collect the necessary number of receipts $N$. At the same time, we want to offer some notion of timeliness: for an aggregate of a given packet rate, collecting the necessary number of receipts—hence achieving a target accuracy—should take a predictable, reasonable amount of time.

**Lemma 4.5.1.** *Given a flow $F$ arriving at a node with packet rate $r$, the expected number of receipts that the node emits for $F$'s packets per time period $T$ is at least equal to $N$, if:*

$$N \leq r \cdot T \cdot \left[ (1-\delta_r)^{r\kappa} - (1-\delta_r)^{\frac{r}{R}\beta} \right] \cdot \sigma. \tag{4.5}$$

*Proof.* See section §4.7.7. □

Informally, Lemma 4.5.1 says that it is possible to parametrize our algorithm such that the monitor collects the necessary receipts to achieve a target accuracy in a timely manner. More precisely, if we set a node's $\beta$, $\kappa$, and $\delta_r$ according to Inequality 4.5, then the node emits at least $N$ receipts per time period $T$ for any flow with packet rate $r$. If an aggregate consists of one or more such flows, then the monitor achieves the target accuracy for it (collects $N$ receipts) within $T$. If an aggregate makes up $\frac{1}{n}$th of such a flow, then the monitor achieves the target accuracy for it within approximately $n \cdot T$.

We now provide intuition behind the lemma: The term on the right-hand side of Inequality 4.5 is the expected number of receipts emitted by the node per time period $T$ for any flow $F$ with packet rate $r$. We explain each term: $r \cdot T$ is the expected number of packets from $F$ that arrive per $T$; $\sigma$ is the probability that a packet $p$ from $F$ is sampled given that its disclosure is neither early nor late; and the term in brackets approximates the probability that $p$'s disclosure is neither early nor late. More specifically, $(1-\delta_r)^{r\kappa}$ approximates the probability that $p$'s disclosure is not early, i.e., no packet that arrives within $r\kappa$ after $p$ is a disclosure packet; $(1-\delta_r)^{\frac{r}{R}\beta}$ approximates the probability that $p$'s disclosure is late, i.e., none of the $F$ packets that are in the buffer when $p$'s receipt is overwritten are disclosure packets. These are approximations, not exact probabilities; the exact probabilities depend on quantities that we cannot predict (the number of packets that arrive after $p$ within a $\kappa$-time period, and the number of packets that arrive after $p$ until $p$'s receipt is overwritten in the buffer). However, due to the stationarity assumption, we can compute their expected values ($r \cdot \kappa$ and $\frac{r}{R} \cdot \beta$, resp.). Moreover, due to the ergodicity assumption and the high rates $r$ considered in this paper (higher than OC-3c or OC-12 transmission rates), the actual values of the distances converge to their expected values at a sub-second granularity. Because of this convergence, we can use the expected values of the distances to approximately compute the probabilities of early and late disclosure as summarized above.

## 4.6 Parametrization and Resource Analysis

We now describe how to set the four parameters of our algorithm (Table 4.1) based on the analysis of the last two sections (§4.6.3), then the resulting resource requirements (§4.6.4).

First, we set the selection rate $\sigma$—the probability with which a non-excluded packet is sampled, hence the maximum sampling rate that our algorithm may apply to a flow. In the current Internet, we would set $\sigma$ to 1% or so, which is typically supported by modern network devices [10].

Second, we set the duration of the quiet period $\kappa$ based on Lemma 4.4.2: Ideally, we want $\kappa \gg d_b - d_g$, such that the delay benefit of any prioritization attack not only is negative, but drops as the buffering period increases with a rate of $-1$ (Inequality 4.4). Given that we cannot know the value of $d_b - d_g$ for all prioritization attacks, we need to set $\kappa$ conservatively, such that it is significantly larger than any realistic intra-domain route difference. In the current Internet, we would set $\kappa$ to 100msec or so.

Third, we compute each node's buffer size $\beta$ based on Lemma 4.5.1: (a) We pick the loss model, i.e., $loss_{min}$ and—if we assume Gilbert loss—$burst_{max}$, as well as the target accuracy that the monitor should achieve given this model. From these, we compute the number of receipts $N$ that the monitor needs to collect per flow in order to compute an estimate (computation in Section §4.6.1). (b) We pick the time interval $T$ and the minimum flow packet rate $r_{min}$ for which the monitor should achieve the target accuracy. Given (a), (b), and each node's maximum total packet rate $R$, we analytically compute, from Inequality 4.5, the minimum buffer size $\beta$ that the node needs in order to emit $N$ receipts per $T$ time units from each flow with packet rate $r \geq r_{min}$. (computation in Section §4.6.2). There are two important things to node: (1) For each node, we compute $\beta$ such that it is sufficient for *all* packet rates above $r_{min}$. (2) Across nodes, $\beta$ can differ significantly, because it depends on the maximum observed total packet rate $R$.

Last, we pick the maximum flow packet rate $r_{max}$ for which the monitor should achieve the target accuracy, and we define the disclosure function $r \rightarrow \delta_r$ based on Lemmas 4.4.2 and 4.5.1: Recall that $\delta_r$ is the probability with which a packet from a flow of packet rate $r$ is chosen as a disclosure packet. For given values of $\sigma$, $\kappa$, and $\beta$, Lemmas 4.4.2 and 4.5.1 define the "operational regime" of our disclosure process: the set of all possible disclosure functions that both make prioritization attacks ineffective and result in the node producing at least the target number of receipts $N$. We design our disclosure function such that any tuple $\{\delta_r, r\}$ falls in this operational regime and such that we use the smallest number of different disclosure rates possible.

### 4.6.1 Optimal Sample Size

We compute the minimum sample size $N$, that is required to achieve $(\gamma,\epsilon)$-accuracy, using statistics. In general, $N$ is a function of the target accuracy and some distribution moment of the quantity we want to estimate.

In this paper, we focus on packet-loss estimates, and since we do not know the distribution of the actual loss in advance, we consider two practical options:

(a) Packet loss is a Bernoulli process: given the sequence of packets from flow $F$ that arrive at node $i$, any packet is lost before reaching node $j$ with probability $l \geq loss_{min}$, and packet-loss events are independent. In this case, the standard central limit theorem (CLT) yields a closed formula for $N$:

$$N = \left(\frac{\eta}{\epsilon}\right)^2 \cdot \frac{1 - loss_{min}}{loss_{min}},$$

where $\eta$ is the $\frac{1+\gamma}{2}$-th quantile of the standard normal distribution.

(b) Packet loss is a bursty process governed by the Gilbert model [36]: which is based on a Markov chain with two states: the process is either in a "low-loss" (good) or in a "high-loss" (bad) state, and it transitions between the two with given probabilities. In this case, we can approximate $N$ empirically through Monte-Carlo simulations.

We can choose one of the two above options based on the nature of the measured traffic (how bursty we expect it to be). If we want to be conservative, we must choose option (b), which yields a sufficient sample size for both bursty and non-bursty traffic; the disadvantage is that, in the non-bursty case, the resulting sample size will be larger than necessary, i.e., we will use more resources than necessary to meet our target accuracy.

Instead of focusing on packet-loss, we could have focused on estimates of average delay or delay quantiles. In this case, optimal $N$ for the mean delay estimation is again given by the central limit theorem:

$$N = \left(\frac{\eta}{\epsilon}\right)^2 \cdot (\mathrm{CoV}_d)^2,$$

where $\eta$ is the $\frac{1+\gamma}{2}$-th quantile of the standard normal distribution and $\mathrm{CoV}_d$ is the coefficient of variation of the delay distribution. For computing $CoV_d$, one must use a model describing the delay distribution, which can be obtained by model fitting methods on actual historical data [18][31][51]. On the other hand, optimal $N$ for a specific $\chi$-quantile is given from the theorem of the "$\gamma$-confidence interval for the sample median and other quantiles" [55] [42][2]:

$$N = \left(\frac{\eta}{\epsilon}\right)^2 \cdot \frac{1-\chi}{\chi},$$

[2]The length of the confidence interval for quantiles cannot be known in advance, because it is taken by two specific $x$ and $y$ order statistics around the $\chi$-th quantile of the sample, the actual values of which are be known in advance. So, the relative error $\epsilon$ of our $(\gamma,\epsilon)$-target accuracy is defined here as the index-difference $(y-x)$ of the confidence interval bounds (instead of the interval between their values).

where $\eta$ is the $\frac{1+\gamma}{2}$-th quantile of the standard normal distribution and $0 < \chi < 1$. Comparing the above to the closed formula for the loss estimation, one may verify that for quantiles $\chi \geq 0.5$, the required sample size for measuring delay is always lower than the one needed for loss, because typically $loss_{min} << 0.5$. Hence, in practice, a system configuration for loss estimation is already sufficient for accurately measuring delay quantiles.

### 4.6.2 Minimization of buffer size $\beta$

In this section, we analytically derive the smallest buffer size necessary for an operational regime to exist. In summary, we make condition from Lemma 4.5.1 an equality and solve for $\beta$ to obtain $\beta(\delta_r, r)$. This is a convex function of $\delta_r$, and we minimize it over the range of $\delta_r$ imposed by Lemma 4.4.2 to obtain $\beta^*(r) = \min_{\delta_r}\{\beta(\delta_r, r)\}$. This is the minimum buffer size needed to produce $N$ receipts for a flow with packet rate $r$, and it is a monotonically decreasing function of $r$. Hence, by setting a node's buffer size to $\beta^*(r_{min})$, we ensure that the node produces receipts at the target receipt rate for any flow with packet rate $r \geq r_{min}$.

Consider a flow $F$, of packet rate $r$, which enters a domain at node $i$ and exits at node $j$; we have derived conditions which guarantee that: the sample produced by nodes $i$ and $j$ is sufficiently large for $(\gamma, \epsilon)$-accuracy (Lemma 4.5.1) and representative even when node $i$ is launching a biasing attack (Lemma 4.4.2).

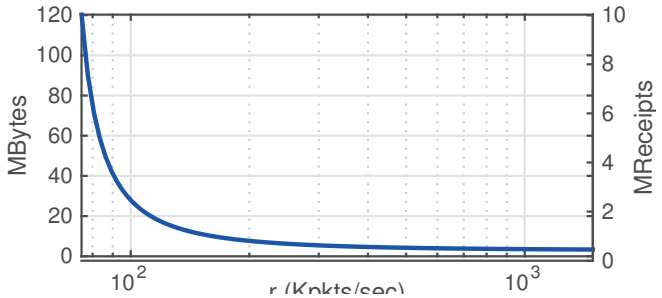
The minimum buffer size that satisfies Lemma 4.5.1 depends on input and workload parameters (which are given), as well as algorithm parameters $\sigma$, $\kappa$, and $\delta_r$. The values of $\sigma$ and $\kappa$ are dictated, respectively, by practicality (§4.2) and attack-resistance (§4.4.2) concerns, which are independent of the buffer size. The disclosure rate $\delta_r$ is restricted by Lemma 4.4.2. Hence, we minimize the buffer size over the disclosure rate $\delta_r$:

$$\beta^* = \min_{\delta_r \in \left(0,\, 1-e^{-\frac{1}{r\kappa}}\right)} \frac{\ln\left((1-\delta_r)^{r\kappa} - \frac{N}{r \cdot T \cdot \sigma}\right)}{\ln\left(1-\delta_r\right)^{\frac{r}{R}}}. \tag{4.6}$$

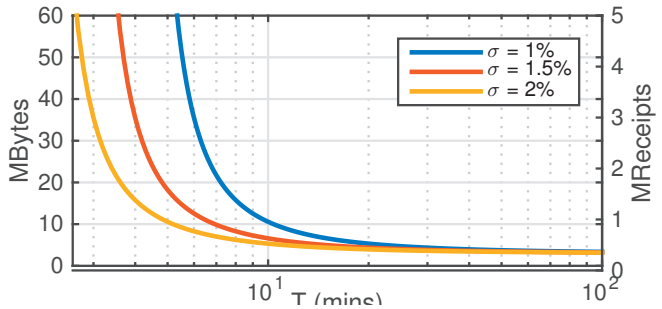
The objective function comes directly from Ineq. 4.5 in Lemma 4.5.1, and the minimization range of $\delta_r$ comes directly from Ineq. 4.2 in Lemma 4.4.2. This is a convex function of $\delta_r$, and we find numerically the disclosure rate $\delta_r^*$ that minimizes it.

The minimum buffer size $\beta^*$ is a decreasing function of the flow packet rate $r$, i.e., the lower a flow's packet rate, the more memory we need to measure the loss rate experienced by the flow. We show this in Fig. 4.3a, where we plot $\beta^*$ as a function of $r$. Intuitively: To achieve our target accuracy, we must lower-bound the probability of late disclosure to some value $\pi$ (because late disclosure reduces the sample size). For any given disclosure rate $\delta_r$, there are two ways to decrease the late-disclosure probability: increase the buffer size or increase the packet rate. Hence, as the packet rate increases, the minimum buffer size necessary to reduce the late-disclosure probability to $\pi$ decreases.

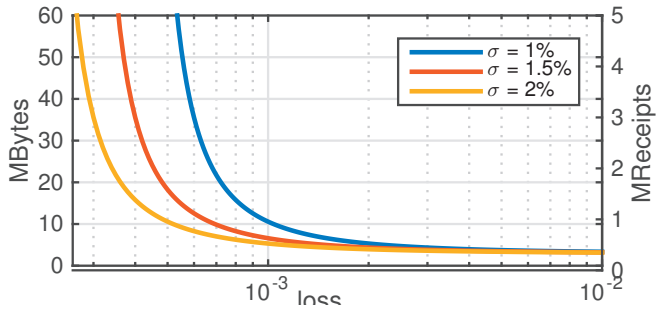
Now consider the same domain and the same entry and exit point, but instead of a single




(a) $\beta^*$ as a function of $r$.
$loss_{min} = 0.1\%$, $\sigma = 1\%$, $T = 10min$.




(b) $\beta^*$ as a function of $T$.
$loss_{min} = 0.1\%$, $r_{min} = 155Kpkts/sec$ (OC-12).




(c) $\beta^*$ as a function of $loss_{min}$.
$T = 10$min, $r_{min} = 155Kpkts/sec$ (OC-12).

Figure 4.3 – Minimum buffer size as a function of various parameters. $\kappa = 100$msec, $\epsilon = 10\%$, $\gamma = 95\%$, $R = 2.5M$ packets/sec (which corresponds to a saturated OC-192 link assuming an average packet size = 500 bytes).
The $y$-axis numbers assume a receipt size of 12 bytes.

flow of constant packet rate, consider a set of flows, each with a potentially different, variable packet rate; our only assumption is that the packet rate of any of these flows always falls within a range $[r_{min}, r_{max}]$, which is given as problem input.

We pick the buffer size $\beta$ based on the smallest supported packet rate $r_{min}$:

$$\beta = \beta^*(r_{min}),$$

where $\beta^*$ is given by Eq. 4.6. Since the minimum buffer size is a decreasing function of the flow packet rate, the minimum buffer size that corresponds to the smallest supported packet rate is sufficient for all the other packet rates.

To provide concrete numbers, Figs. 4.3b and 4.3c show the resulting buffer size in various realistic settings. For instance, for a minimum supported rate equal to $r_{min} = 155K$ packets/sec (which corresponds to a saturated OC-12 link assuming an average packet size = 500 bytes), we see that a few MB of memory are typically sufficient for measuring a minimum loss rate of $loss_{min} = 0.1\%$ in $T = 10$ minutes. We also see that the buffer size drops rapidly as the measurement period $T$ and/or the minimum measurable loss rate $loss_{min}$ increase. Intuitively, the more time we can afford to reach a target accuracy, and the higher the loss rate that we want to be able to measure accurately, the lower the necessary sampling rate, and the fewer the resources necessary to provide it.

Clarification: We see that, for any selection rate, there always exists a minimum $T$ and a minimum $loss_{min}$ at which $\beta^*$ becomes infinite, i.e., at these points, we could not achieve the target accuracy even with an infinite buffer size. This is because, at these points, the number of packets produced by the slowest flows in our supported range falls below the minimum necessary sample size.

### 4.6.3 Operational Regime

Fig. 4.4 shows an example operational regime. The particular scenario for which it was computed does not matter for this discussion, but we provide it for completeness: target accuracy ($\gamma = 95\%$, $\epsilon = 10\%$), target time interval $T = 10$min, minimum packet rate $r_{min} =$ 155Kpps (saturated OC-12 interface, assuming average packet size 500B), maximum packet rate $r_{max} = 2.5$Mpps (saturated OC-192 interface, assuming average packet size 500B), and algorithm parameters $\sigma = 1\%$, $\kappa = 100$msec, and $\beta = 10.5$MB. The operational regime consists of the surface enclosed between the lower bound obtained from Ineq. 4.5 (blue curve) and the smallest of the two upper bounds obtained, respectively, from Ineq 4.2 (yellow curve) and Ineq 4.5 (red curve). In this particular example, the smallest upper bound is the former (the yellow curve).

In more detail: (a) We analytically compute all the tuples $\{\delta_r, r\}$ that satisfy Inequality 4.2 of Lemma 4.4.2: solving the inequality for $r$ yields an upper bound for $r$ as a function of $\delta_r$ (the middle, yellow curve in Fig. 4.4). (b) We numerically[3] compute all the tuples $\{\delta_r, r\}$ that satisfy the inequality of Lemma 4.5.1: solving the inequality for $r$ yields an upper and a lower bound for $r$ as a function of $\delta_r$ (top/red curve and bottom/blue curve in Fig. 4.4). (c) We divide $[r_{min}, r_{max}]$ into non-overlapping packet-rate "zones" and map each zone to one disclosure rate that falls between the lower bound and the smaller of the two upper bounds, using as few zones as possible. In Fig. 4.4, we use two zones, shown on the figure as black

[3]We can only do this numerically. Solving Ineq. 4.5 for $r$ analytically would require solving a polynomial of degree $\frac{\beta}{R} - \kappa$.

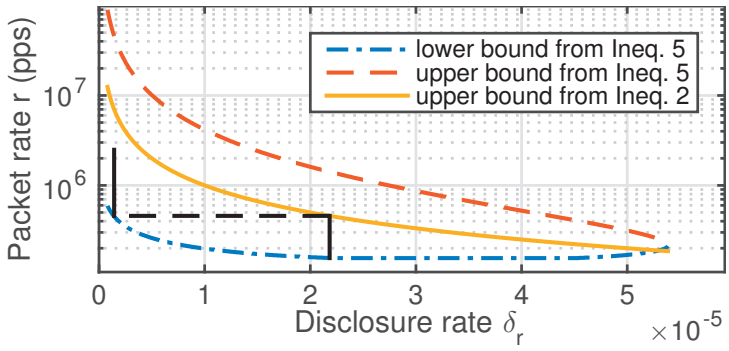



Figure 4.4 – Example operational regime.

vertical lines: packet rates 155–437Kpps map to disclosure rate $2.18 \cdot 10^{-5}$, while packet rates 155Kpps–2.5Mpps map to disclosure rate $0.137 \cdot 10^{-5}$.

The disclosure function affects the implementation of our algorithm in the following sense: The algorithm tracks each flow's packet rate $r$ in order to choose disclosure packets from that flow with the right probability $\delta_r$ (lines 3,4 in Alg. 2). The fewer packet-rate zones we use, the less precise our packet-rate tracking needs to be, which simplifies our algorithm's implementation. For example, if we use two zones (as in the example of Fig. 4.4), then our packet-rate tracking needs to be only precise enough to determine whether a flow's packet rate falls in one zone or the other. We found 2 or 3 zones to be enough in all the scenarios we considered.

Do we know that the operational regime is always computable—that, for any $\sigma$, $\kappa$, and $\beta$, Lemmas 4.4.2 and 4.5.1 yield an upper and lower bound for $r$ as a function of $\delta$, and that, for any fixed $\delta$, we can identify a closed range of packet rates $r$ that honor the two bounds? This was true in all the scenarios we considered, but we cannot prove it in the general case, because step (b) of the computation (solving Ineq. 4.5 for $r$) is numerical, not analytical. We can only prove it in one particular, interesting case: when $\beta$ becomes so large that it never overflows (the probability of late disclosure becomes negligible for all flows). In this particular case, Ineq. 4.5 *can* be solved for $r$ analytically and yields the following lower and upper bound:

$$\frac{W_0\left(\frac{N\kappa}{T\sigma}\ln(1-\delta_r)\right)}{\kappa\ln(1-\delta_r)} \leq r \leq \frac{W_{-1}\left(\frac{N\kappa}{T\sigma}\ln(1-\delta_r)\right)}{\kappa\ln(1-\delta_r)}$$

where $W_0$ and $W_{-1}$ are the two branches of the Lambert function. The exact formulas do not matter—we provide them for completeness; the point is that, in this one case where we can analytically compute the operational regime, it has the shape that we expect based on our numerical solution.

### 4.6.4 Resource Requirements

Having established how to set the parameters of our algorithm, we considered the values we would use in various realistic scenarios and computed the resulting resource requirements.

**Memory overhead**: We found that our algorithm requires a buffer size $\beta$ that increases data-path memory only by a few percentage points, which is at least an order of magnitude less than alternative algorithms.

We present a concrete example: Consider a node collocated with a 10GigE interface, observing a total packet rate $R$ = 2.5Mpps with an average packet size 500B. Assume 12B per receipt. We set $\sigma$ = 1% and $\kappa$ = 100msec. We set target accuracy ($\gamma$ = 95%, $\epsilon$ = 10%), within a time interval of $T$ = 10min, assuming i.i.d. loss of minimum rate $loss_{min}$ = 0.1% (e.g., we are verifying SLAs that promise loss rate below 0.1%). We set $r_{max}$ = 2.5Mpps, which is the maximum packet rate at which a flow could arrive at this node. Fig. 4.5 shows the amount of data-path memory required by our algorithm, as well as various alternatives, as a function of $r_{min}$ (the minimum supported flow packet rate).

Our algorithm (red line, abbrev. "retro") requires a few MB of data-path memory, whereas basic delayed disclosure (green circles, abbrev. "basic DD") requires at least an order of magnitude more. To put this in perspective, a 10GigE interface needs about 125MB of packet buffers (using the "one round-trip worth of traffic" rule and assuming a typical Internet round-trip of 100msec), i.e., our algorithm increases data-path memory only by a few percentage points.

There are three reasons why our algorithm requires less data-path memory: (a) Basic DD samples more packets than necessary to achieve the target accuracy in the target time interval, because it tries to achieve a target sampling probability, not the target accuracy in the target time interval. (b) Basic DD uses a fixed disclosure rate $\delta$, yet no single $\delta$ works well for all flows: faster flows need a lower $\delta$ to prevent early disclosure from happening too often, while slower flows need a higher $\delta$ to prevent late disclosure from happening too often. Basic DD picks a $\delta$ that is low enough to accommodate the fastest flows (those with packet rate $r_{max}$) and, as a result, requires too much memory to protect the slowest flows (those with packet rate $r_{min}$) from late disclosure. (c) Basic DD avoids late disclosure more than necessary to achieve the target accuracy in the target time interval, because it was designed to avoid late disclosure with a fixed, high probability.

To quantify the impact of each issue, Fig. 4.5 shows the data-path memory required by upgraded versions of basic DD: (a) Basic DD+optimal $N$ is like basic DD, except it tries to achieve the target accuracy in the target time interval. (b) Basic DD+optimal $N$+zones moreover uses an adaptive disclosure rate, though it still computes the disclosure function so as to avoid late disclosure with a fixed, high probability. (c) Retro—our algorithm—uses an adaptive disclosure rate and computes the disclosure function so as to achieve the target accuracy in the target time interval. We see that each issue plays a non-trivial role, with the last one being the most impactful: avoiding late disclosure improves accuracy but requires a bigger receipt

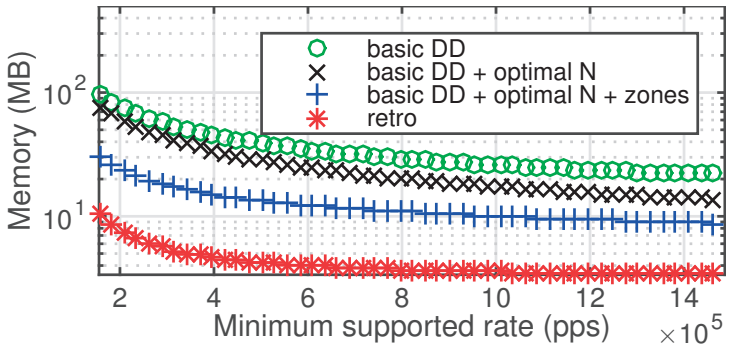



Figure 4.5 – Data-path memory consumed by receipt buffer as a function of the minimum supported packet rate.

buffer; hence, it is important to avoid it just enough to achieve the target accuracy in the target time interval, but not with an arbitrarily high probability.

**Processing overhead**: Our algorithm requires a small number of hash computations, timestamp comparisons, and accesses to data-path memory per packet, which is similar to basic delayed disclosure (we claim no improvement on processing overhead). The exact numbers depend on the implementation; the hardware design sketched in the Appendix, Section 4.3, requires per packet: two hash computations, a couple of timestamp comparisons, one read and one write access to data-path SRAM, and a parallel lookup and an update to data-path CAM.

## 4.7 Experimental Evaluation

After describing our methodology (§4.7.1), we demonstrate that our algorithm is useful (§4.7.2), confirm that it works as expected (§4.7.3) and is resistant to prioritization attacks (§4.7.4).

### 4.7.1 Methodology

In each experiment, we emulate some number of flows, crossing one or more domains. We use 1-hour backbone traces made available by CAIDA in 2016 (chicago-equinix, direction A). Each flow observed at an entry node consists of one entire trace, while the total traffic observed at an entry node consists of multiple traces merged into one. Why this particular emulation: The question we are most frequently asked is how well our system would work for busy backbone routers located at the Internet core. The traffic rate of a single CAIDA trace ranges from a few hundred Mbps to a few Gbps; this is too low to represent the total traffic arriving at a busy backbone-router interface, but could represent, e.g., the traffic between a source and destination prefix connected to the Internet through OC-48 or lightly loaded 10GigE links. By merging multiple traces—and shifting packet timestamps such that all traces start at the same time—we created higher-rate ingress streams.

In each experiment, we emulate either i.i.d. or bursty loss (of various rates). For the latter, we obtained the loss pattern from an actual congested link: we created in our lab a simple topology where 16 pairs of end-points communicated over a bottleneck GigE link, and we had each pair exchange back-to-back TCP flows; this resulted in the bottleneck link experiencing packet loss of average rate 4.8% and burstiness 1.52 packets.

Our configuration is purposefully not realistic in all scenarios, because we want our algorithm to operate at its limits. For instance, in a real deployment, we would set $loss_{min} = 0.1\%$ or so. However, if the actual loss rate is $\gg loss_{min}$, our algorithm will use significantly more memory than necessary to estimate this loss rate, and our results will be obviously good. Hence, we set $loss_{min}$ to the actual loss rate, which results in our algorithm using the absolute minimum memory needed to estimate this loss rate. So:

(a) In all experiments, we set the selection rate to $\sigma = 1\%$, the quiet-period duration to $\kappa =$ 100msec, the target accuracy to ($\gamma = 95\%$, $\epsilon = 10\%$), and the target time interval to $T = 10$min.

(b) In all experiments, we set $loss_{min}$ to the actual loss rate and—when the actual loss is bursty—$burst_{max}$ to the actual loss burstiness. This way, we test how accurately the monitor estimates loss that is exactly at the limit of what the nodes were configured to handle.

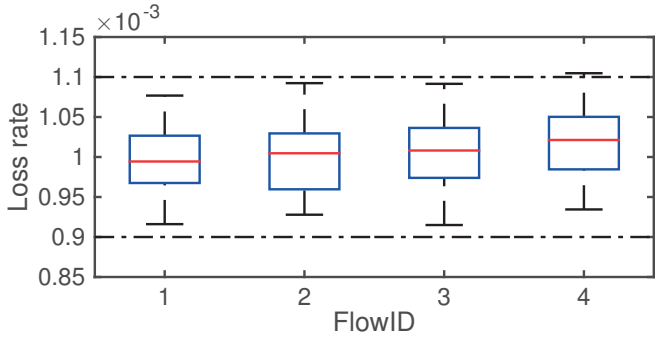
(c) In §4.7.2, we set $r_{min}$ to the average packet rate of the target flow $F$. This way, we test how accurately the monitor estimates loss experienced by a flow whose packet rate is exactly at the limit of what the nodes were configured to handle. This way, we test how accurately the monitor estimates loss experienced by a flow whose packet rate is exactly at the limit of what the nodes were configured to handle. For the same reason, in §4.7.3, we set $r_{min}$ to the average packet rate of the slowest flow involved in the experiment.

(d) In §4.7.4, where node $i$ launches prioritization attacks, we configure the nodes such that the average packet rate of the target flow $F$ falls exactly on the upper bound of the operational regime (the middle, yellow curve in Fig 4.4). This is the most challenging setting we could think of: if node $i$ operates close to the upper bound of the operational regime, it is possible that $F$'s instant packet rate fluctuates so fast that it temporarily pushes node $i$ out of the operational regime, where it could potentially cheat.

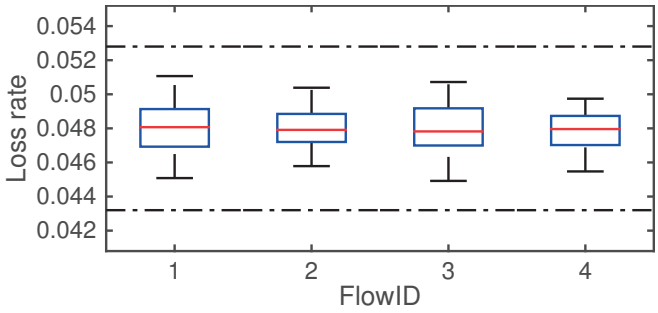
### 4.7.2 Use Cases

First, we demonstrate that our algorithm enables the monitor to draw useful conclusions about network behavior.

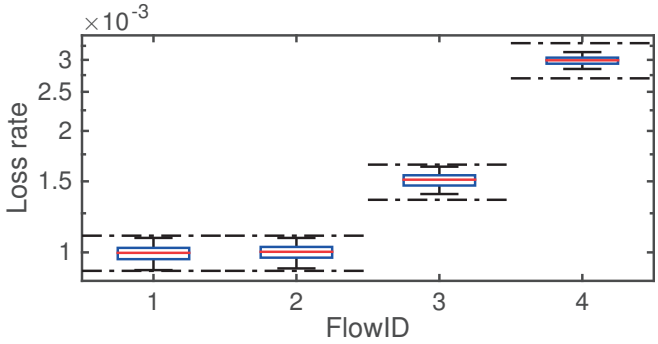
In each experiment, we emulate two to four flows that enter an ISP $x$ at node $i$ and exit at node $o$. Both nodes are collocated with highly loaded 10GigE interfaces. We use four CAIDA traces. The total traffic observed at node $i$ consists of the four traces merged together and has a rate of 8Gbps.




(a) I.i.d. loss 0.1%.




(b) Bursty loss 4.8%.




(c) Loss rates per flow ID: 0.1%, 0.1%, 0.15% and 0.3%.

Figure 4.6 – Loss estimates after $T = 10$min.

**SLA verification.** ISP $x$ has signed an SLA with a customer network, promising loss below 0.1%; the customer's traffic suffers exactly 0.1% loss within $x$. On the customer's request, the monitor estimates $x$'s loss with respect to traffic from the customer's prefix to four popular destination prefixes (so, we have four aggregates, each corresponding to a flow). Fig. 4.6a shows the monitor's loss estimates with respect to each aggregate/flow after the target 5min time interval; in each boxplot, the red line shows the median, while the limits of the boxplot indicate the 95% confidence interval. We see that all estimates achieve the target accuracy ($\gamma = 95\%, \epsilon = 10\%$). Hence, $y$ correctly determines that $x$ is borderline violating its SLA.

**Cheap SLA verification.** Same as above, but this is a cheaper SLA, promising loss below 5%; the customer's traffic suffers 4.8% loss with burstiness 1.5 due to a congested bottleneck link

at $x$'s core. Fig. 4.6b shows that the monitor's loss estimates achieve the target accuracy after the target 5min time interval. Hence, $y$ correctly determines that $x$ is honoring its SLA, despite the fact that it is introducing loss just below the promised maximum.

**Subtle prefix discrimination.** Transit ISP $x$ is dissatisfied with networks $y$ and $z$: the former is a popular video provider; the latter is an eyeball ISP whose customers freely participate in peer-to-peer networks; $x$ wants to renegotiate its peering agreement with each of these networks, but they are resisting. In response, $x$ subtly discriminates against them: while most transit traffic experiences loss 0.1%, $y$'s and $z$'s traffic experience, respectively, 0.15% and 0.3% loss. After user complaints, the monitor estimates $x$'s loss with respect to two random flows, one flow originating from $y$, and one flow originating from $z$ (so, again, we have four aggregates, each corresponding to a flow). Fig. 4.6c shows that the monitor catches the discrimination after the target 5min time interval.

**Policing of SYN packets.** ISP $x$ has signed an SLA with a customer network, promising loss below 0.1%; it honors this SLA for all but TCP SYN packets, which are policed such that they experience 3–30 times higher loss (we conducted 240 experiments with various policing rates). The customer observes end-to-end that something is wrong with connection setup and asks the monitor whether $x$ is discriminating against its SYN packets. In response, the monitor defines two aggregates: SYN packets from the customer's prefix to some popular destination prefix; all other packets with the same source and destination prefix. So, in this case, we have two aggregates that are subsets of the same flow.

The challenge is that the SYN aggregate is relatively small, and the monitor would need to collect receipts for hours in order to estimate $x$'s loss with respect to the SYN aggregate with the target accuracy of ($\gamma = 95\%$,$\epsilon = 10\%$). However, the goal here is not to estimate $x$'s performance, but to determine whether it treated the two aggregates differently. This can be done much faster, with a simple Maximum Likelihood differentiation detector: the monitor estimates $x$'s loss rate for each aggregate based on the receipts it collects within some period of time (minutes, not hours); and computes the corresponding confidence intervals (CI); if the lower limit of the CI for the SYN aggregate estimate is greater than the upper limit of the CI for the no-SYN aggregate, then the monitor concludes that $x$ discriminates against the SYN aggregate.

Our results, after the target 5min time interval: When $x$'s loss with respect to SYN packets is 5 or more times higher, the monitor detects differentiation with probability 100% (in all experiment runs); when SYN loss is 3–5 times higher, detection rate is $\geq 94\%$; for subtler differentiation, detection rate drops sharply. For completeness, we also ran 480 experiments where $x$ does not differentiate, and the monitor correctly detects no differentiation.

### 4.7.3 Basic Operation

Next, we confirm that our algorithm works as it should, i.e., the nodes sample consistently at the necessary rates.

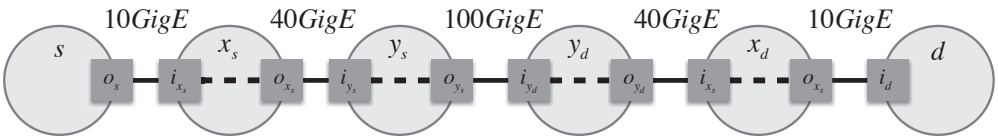



Figure 4.7 – Topology emulated in §4.7.3.

We emulate the topology in Fig. 4.7: there are 4 flows between edge networks $s$ and $d$; 6 additional flows enter and exit the topology, respectively, at regional ISPs $x_s$ and $x_d$; and 22 additional flows enter and exit, respectively, at Tier-1 ISPs $y_s$ and $y_d$. Hence, we use a total of 32 CAIDA traces. The total packet rates observed at nodes $o_s$, $o_{x_s}$, and $o_{y_s}$ are, respectively, 1.92, 4.81 and 15.47Mpps (equiv. 7.67, 19.2 and 61.8Gbps). There is i.i.d. loss of rate 0.2% inside domain $x_s$ and 0.1% inside domain $y_s$.

Fig. 4.8 shows, for each flow, the consistent sample size (the number of consistently sampled packets) produced within the target time interval by all the nodes that observed the flow. For instance, Fig. 4.8a shows the consistent sample size produced by each of the 4 flows observed by all 10 nodes; Fig. 4.8c shows the consistent sample size produced by each of the 32 flows observed by nodes $o_{y_s}$ and $i_{y_d}$. In all plots, the horizontal red line shows the minimum sample size $N$ needed to estimate i.i.d. loss of minimum rate $loss_{min} = 0.1\%$ with the target accuracy.

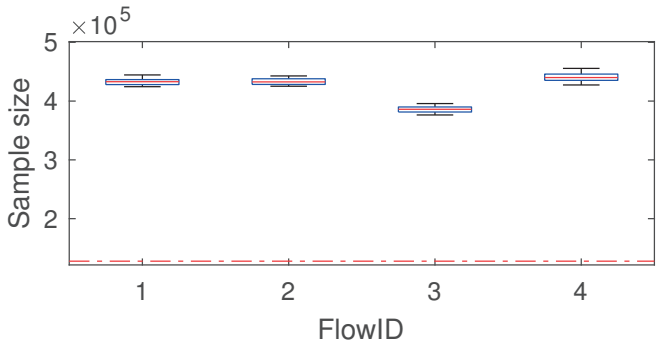
We see that, for all flows, the produced consistent sample size exceeds the minimum necessary. In particular, Fig. 4.8a shows that all 10 nodes sample consistently at the necessary rate, despite the fact that they observe total packet rates that differ by an order of magnitude; this is because each node's receipt buffer is large enough to accommodate the maximum total packet rate that the node may observe. Moreover, Fig. 4.8c shows that nodes $o_{y_s}$ and $i_{y_d}$ sample consistently at the necessary rate from all 32 flows that they observe in common; this is because each node is configured to produce at least the minimum sample size for all flows with packet rate $\geq r_{min}$.

### 4.7.4 Resistance to Prioritization

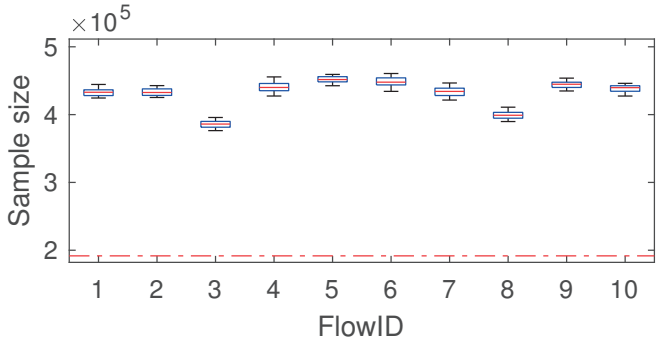
Finally, we confirm that our algorithm makes prioritization attacks ineffective, whereas basic delayed disclosure allows a misbehaving network to significantly exaggerate its performance.

ISP $x$ launches a prioritization attack (§4.4.1) against one target flow. The delay of the two routes is $d_b = 50$msec and $d_g = 10$msec (the good route is 5 times faster than the bad one). We vary the attack parameter $g$ (the fraction of traffic that fits in the good route) from 0 to 100%, and the attack parameter $t$ (the buffering period) from 0 to 120msec. We measure the attack's relative delay benefit: by how much the misbehaving domain exaggerates its perceived delay performance relative to the delay performance it would achieve without a prioritization attack.

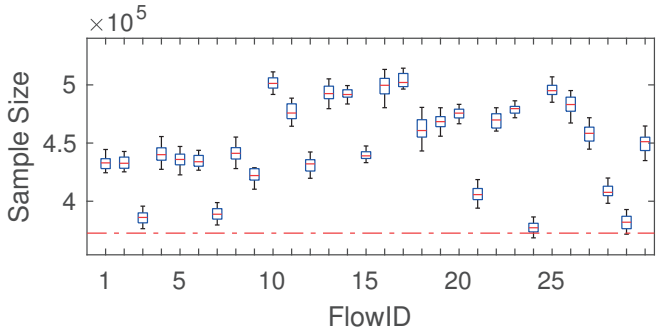
Fig. 4.9a shows the results. The $y$-axis measures the attack's relative delay benefit (in percentage points), while the two $x$-axes measure, respectively, the attack parameters $t$ (in msec) and $g$ (in percentage points). *We do not obtain the plotted data points from our formulas, but*




(a) Flows between $s$ and $d$.




(b) Flows between $x_s$ and $x_d$.

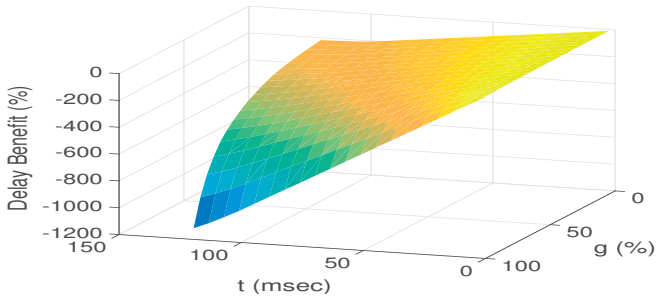



(c) Flows between $y_s$ and $y_d$.

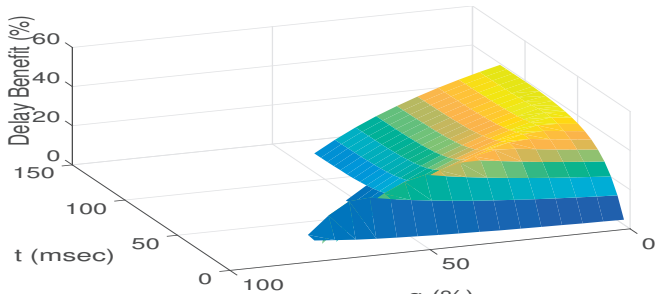
Figure 4.8 – Consistent sample size after $T = 10$min.

*experimentally, as the monitor would compute them from the receipts produced by x's entry and exit nodes.*

We see that the attack's relative delay benefit is always negative and as low as $-1122\%$ (by misbehaving, $x$ worsens its perceived delay performance by this much). The highest delay benefit is close to 0, and it occurs when $t \to 0$, at which point $x$ almost does not buffer/cheat. The lowest delay benefit occurs at $t = 120$msec and $g \to 1$. At this point, $x$ forwards most packets over the 10msec good route, while it buffers sampled packets for 120msec. As a result, relative to an honest behavior (where it would not buffer and its performance, both true and perceived, would be almost 10msec), $x$ worsens its relative delay benefit by almost $-1200\%$. This is consistent with the math in the proof of Lemma 4.4.2, which shows that $x$'s misbehaving




(a) Our algorithm.




(b) Basic delayed disclosure.

Figure 4.9 – Relative delay benefit of a prioritization attack as a function of attack parameters $t$ and $g$.

function is convex w.r.t. $t$ and concave w.r.t. $g$, hence the lowest delay benefit occurs when either $g \to 0$ or $g \to 1$.

One might expect that basic delayed disclosure would also handle prioritization attacks well, without the added complexity of quiet periods and an adaptive disclosure rate. To show that this is not the case, we repeat the experiment, but have node $i$ run basic delayed disclosure instead of our algorithm. To cheat, node $i$ buffers each packet $p$ for $t$ or until disclosure occurs, whichever comes first; if disclosure comes first and $p$ is sampled, node $i$ forwards $p$ over the good route, otherwise, it forwards $p$ such that, overall, a fraction $g$ of all packets are forwarded over the good route. We configure basic delayed disclosure with $\beta = 4.2$MB (the same amount used by our algorithm) and $\delta = 9.9 * 10^{-5}$, $\sigma = 1\%$ (as advised in [14]).

We see, in Fig. 4.9b, that there exist multiple prioritization strategies—multiple $\{g, t\}$ combinations—where the attack's delay benefit is both positive and significant, as high as +41% (by misbehaving, $x$ improves its perceived delayed performance by this much). The intuition is that $x$ does not need to know the sampling fate of *all* the packets it forwards in order to exaggerate its perceived delay performance; knowing the sampling fate of a small fraction of the forwarded packets is enough, and this knowledge can be gained with a surprisingly short $t$ that does not introduce significant delay overhead. As a result, without counter-measures in place, prioritization attacks defeat the purpose of delayed disclosure.

### 4.7.5 Proof of lemma 4.4.1

Let $F$ (resp. $P$) denote the stationary and ergodic point process with intensity $r$ (resp. $R$) that corresponds to the packet arrivals from flow $F$ (resp. to the packet arrivals of the total traffic observed by node $i$). $P$ and $F$ are two point processes of the same stationary continuous-time arrival process. Their intensities ($R$ and $r$) are just two different event "clocks" of that process: $P$-clock ticks every time that a packet arrives and $F$-clock ticks only if the arriving packet belongs to flow $F$.

Consider a packet $p$ from flow $F$ arriving at node $i$ at random time $t = 0$; that is, an $F$-point occurs at an arbitrary point in time[4]. $X_p$ denotes the number of $F$-points until $p$'s disclosure packet arrives (which is also an $F$-point occurrence); $n_F[0,\kappa]$ denotes the number of $F$-points that occur a time interval $\kappa$ after $t = 0$; $n_F\left[P_0, P_\beta\right]$ denotes the number of $F$-points that occur in the upcoming $\beta$ $P$-points after $p$.

Packet $p$ is sampled when the following three events occur: (a) the disclosure of $p$ is not early, i.e., $X_p > n_F[0,\kappa]$; (b) neither is it late, i.e., $X_p \leq n_F\left[P_0, P_\beta\right]$; and (c), $p$ is selected by line 15 of Alg. 2.

Therefore, $p$'s sampling probability $\mathbb{P}^0(a,b,c)$, as computed at time $t = 0$ is given by the chain rule of probability:

$$\mathbb{P}^0(a,b,c) = \mathbb{P}(a) \cdot \mathbb{P}(b|a) \cdot \mathbb{P}(c|a,b) \tag{4.7}$$

The last part $\mathbb{P}(c|a,b)$ of Eq. 4.7 is equal to the selection rate $\sigma$. The first two parts are computed based on the following observation: In Alg. 2, *DiscHash* has strong randomization properties, which means that the arrivals of the disclosure packets occur independently over the $F$-arrivals and with equal probability $\delta_r$. I.e., the disclosure arrivals form a Bernoulli stochastic process on the top of $F$-arrivals, which is renewed at each $F$-point. Therefore, for any arbitrary packet $p$ of flow $F$, the "distance" until the disclosure packet $X_p$, counted in $F$-points, follows the geometric distribution:

$$\begin{aligned}
&\mathbb{P}(a) \cdot \mathbb{P}(b|a) = \\
&= \mathbb{P}\left(X_p > n_F[0,\kappa]\right) \cdot \mathbb{P}\left(X_p \leq n_F\left[P_0, P_\beta\right] \mid X_p > n_F[0,\kappa]\right) \\
&= \mathbb{P}\left(X_p > n_F[0,\kappa]\right) \cdot \left(1 - \mathbb{P}\left(X_p > n_F\left[P_0, P_\beta\right] \mid X_p > n_F[0,\kappa]\right)\right) \\
&= \mathbb{P}\left(X_p > n_F[0,\kappa]\right) \cdot \left(1 - \mathbb{P}\left(X_p > n_F\left[P_0, P_\beta\right] - n_F[0,\kappa]\right)\right) \\
&= (1-\delta_r)^{n_F[0,\kappa]} \cdot \left(1 - (1-\delta_r)^{n_F[P_0,P_\beta] - n_F[0,\kappa]}\right) \\
&= (1-\delta_r)^{n_F[0,\kappa]} - (1-\delta_r)^{n_F[P_0,P_\beta]},
\end{aligned} \tag{4.8}$$

[4]For this proof, we use the same convention as in Palm Calculus [42]: $t = 0$. This convention is the one used to give a meaning to an arbitrary point in time and differs from the beginning of the process; it is the time of arbitrary point in a process that has a stationary regime and has run long enough to be in steady state.

where the third derivation step is because of the memoryless property of the geometric distribution.

Given Eq. 4.7 and 4.8, the sampling probability of packet $p$ is:

$$\mathbb{P}^0(a,b,c) = \left((1-\delta_r)^{n_F[0,\kappa]} - (1-\delta_r)^{n_F[P_0,P_\beta]}\right) \cdot \sigma \tag{4.9}$$

**Main proof**

To prove lemma 4.4.1, it is sufficient to show that the sampling probability $\mathbb{P}^0(a,b,c)$ that node $i$ can predict upon $p$'s arrival does not depend on $p$'s arrival time ($t = 0$), and it is equal for all packets from flow $F$. We will use the same assumption of Section §1.5, which we restate here for completeness: $P$ and $F$ are stationary, ergodic, and both intensities $r$ and $R$ are high enough that the ergodic convergence occurs in less than 100msec, i.e., in time less than the typical values for $\kappa$.

Stationarity of point processes implies that: distributions of the number of points in a fixed interval $(t_1', t_1'']$ are invariant under translation, i.e., is the same for $(t_1' + h, t_1'' + h]$ for all $h$. An immediate consequence is that the distribution of the number of points $n_F$ in an interval depends only on the length of the interval and not its time origin [22].

Therefore, the distributions of $n_F[0,\kappa]$ and $n_F\left[P_0,P_\beta\right]$ depend only on $\kappa$ and $\beta$ and not on the time that $p$ is observed. They are the same at any arbitrary point in time.

As for any random variable, the best prediction for $n_F\left[P_0,P_\beta\right]$ and $n_F[0,\kappa]$ is obtained using their expected values. This is true because if one wants to predict a random variable from its distribution, then the mean-square-error (MSE) predictor and the best one-number guess is just its expected value [1].

We compute the expected values of $n_F[0,\kappa]$ and $n_F\left[P_0,P_\beta\right]$ using Palm Calculus [42]:

$$\mathbb{E}\left[n_F[0,\kappa]\right] = r \cdot \kappa \tag{4.10}$$

$$\mathbb{E}\left[n_F\left[P_0,P_\beta\right]\right] = \frac{r}{R} \cdot \beta \tag{4.11}$$

Eq. 4.10 is an immediate consequence of the definition of the intensity of a stationary point process, which is equal to the expected number of points per time unit. Eq. 4.11 requires the auxiliary lemma 4.7.1 (provided at the end of this section), and linearity of expectation.

By applying Eqs. 4.10 and 4.11 to Eq. 4.9, we see that the best prediction of the sampling probability of an arbitrary packet $p$, as computed at the time of arrival, does not depend on $p$'s arrival time or any other previously observed packet arrival. It only depends on the the intensities $R$ and $r$. Even if node $i$ manages to estimate perfectly $R$ and $r$, when it attempts to predict the sampling probability of each new packet $p$ that it observes, it computes the *same* sampling probability for all new packets.

**A longer version:**

In the rest of this section, we provide a longer version of this proof starting from the ergodicity assumption, which is implied in the above results based on Palm Calculus.

The ergodic theorem [33] for weakly[5] stationary processes implies that: for each weakly stationary process, the time average of the process converges to a random variable that has the same expected value. Let $F(t)$ denote the continuous-time arrival process of $F$-point occurrences and $F_P[j]$ denote the arrival process of the same $F$-point occurrences according to the $P$-event clock. I.e., $F(t)$ (resp. $F_P[j]$) is 1, if an $F$-point occurs at time $t$ (resp. at $P$-point $j$) and 0 otherwise. From the definition of processes $F$ and $P$, $F$-events are a subset of $P$-events; hence, process $F$ is stationary in both continuous time and $P$-event clock. Let also the random variables to which the time averages converge be $Y$ and $Y_P$. Then, given that convergence rates are fast due to our assumption of high packet arrival rates, the ergodic theorem suggests that:

$$\frac{1}{\kappa}\int_0^{\kappa} F(t)\,dt \xrightarrow{m.s.} Y$$

$$\frac{1}{\beta}\sum_{j=1}^{\beta} F_P[j] \xrightarrow{m.s.} Y_P$$

Because of the definition of $n_F[0,\kappa]$ and $n_F\left[P_0,P_\beta\right]$, we have the following: $n_F[0,\kappa] = \int_0^{\kappa} F(t)\,dt$ and $n_F\left[P_0,P_\beta\right] = \sum_{j=1}^{\beta} F_P[j]$. So, $n_F[0,\kappa]$ and $n_F\left[P_0,P_\beta\right]$ converge to $\kappa \cdot Y$ and $\beta \cdot Y_P$, respectively.

We obtain the MSE (best) predictor for $n_F\left[P_0,P_\beta\right]$ and $n_F[0,\kappa]$, by computing the best predictors for $Y$ and $Y_P$; i.e., $\mathbb{E}[Y]$ and $\mathbb{E}[Y_P]$. From ergodicity, we get: $\mathbb{E}[Y] = \mathbb{E}[F(0)]$ and $\mathbb{E}[Y_P] = \mathbb{E}[F_P[1]]$ (same mean condition above); and from the definition of the intensity $r$ of process $F$ and lemma 4.7.1, we get: $\mathbb{E}[F(0)] = r$ and $\mathbb{E}[F_P[1]] = \pi_F = \frac{r}{R}$.

Therefore, the best predictions for $n_F\left[P_0,P_\beta\right]$ and $n_F[0,\kappa]$ are:

$$\widehat{n_F[0,\kappa]} = r \cdot \kappa \tag{4.12}$$

$$\widehat{n_F\left[P_0,P_\beta\right]} = \frac{r}{R} \cdot \beta \tag{4.13}$$

By applying the best predictions to Eq. 4.9, we see that the sampling probability is independent of $p$'s arrival time and is the same for all packets from flow $F$.

Note 1: The geometric distribution requires only an integer exponent of $(1-\delta_r)$. This is because in the PMF of a geometrically distributed random variable the exponent denotes the number of trials until the success, which must be an integer. But, in our case, $r\kappa$ and $\frac{r}{R}\beta$ are

[5] Weak stationarity is enough for the proof of our theoretic results. There is no need to assume strong stationarity, which is already impractical to verify in real traffic traces.

not necessarily integer values. Another way for computing the probabilities $\mathbb{P}(a)$ and $\mathbb{P}(b|a)$ would be to regard the arrivals of the disclosure packets as a Poisson arrival process, which is the continuous-time equivalent of the Bernoulli. In this case, $X$ would be exponentially distributed with rate $\delta_r$. However, if the exponents of the geometric distribution are generally large and the probability $\delta_r$ is relatively small (which holds in our case as shown in Fig. 4.4), then the probabilities obtained using the geometric distribution approximate very well the ones obtained by the exponential distribution. In this work, we have used the geometric distribution, because we believe it provides more intuitive results. A similar analysis holds when considering a Poisson arrival process for the disclosure packets.

Note 2: Lemmas 4.4.1 and 4.7.1 (below) do not imply any assumption about independence of $F$-arrivals, nor Poisson packet arrivals. The proofs are based on Palm Calculus and linearity of expectation, that do not require independence. The only assumptions here are stationarity and ergodicity. The "steady-rate"or the Poisson assumption make the computation of Eq. 4.11 easier and provide a good insight of the proof, but do not correspond to real traffic arrival patterns.

**Lemma 4.7.1.** *Given two point processes $P$ and $F$ of the same stationary process, which have rates $R$ and $r$, respectively, the probability of an arbitrary $P$-arrival to be an $F$-arrival (i.e the arrived packet belongs to flow $F$), is:*

$$\pi_F = \frac{r}{R}$$

*Proof.* We base our proof on the concepts of Palm Calculus [42].

Let $\lambda_F(P)$ denote the intensity of the $P$ point process measured with the event clock $F$. Also, let $n_F[P_0, P_1]$ denote the number of points of process $F$ that fall between the two random subsequent $P$ points $P_0$ and $P_1$.

Apply Neveu's Exchange Theorem [42] to obtain:

$$\lambda_F(P) = \frac{R}{r} \quad \text{and} \quad \mathbb{E}(n_F[P_0, P_1]) = \frac{1}{\lambda_F(P)}.$$

Note that $n_F[P_0, P_1]$ is 1, if the packet at $P_0$ is an $F$ packet, i.e. if the $P$ point at $P_0$ is also an $F$ point, and 0 otherwise. Hence, $\mathbb{E}(n_F[P_0, P_1])$ is the probability $\pi_F$ that an arbitrary packet is a packet from flow $F$. Thus:

$$\pi_F = \frac{\lambda(F)}{\lambda(P)} = \frac{r}{R}$$

□

### 4.7.6 Proof of lemma 4.4.2

The main part of the proof consists of an analysis of the expected misbehavior benefit w.r.t. $t$ and $g$. A node launches a prioritization attack only if its expected misbehavior benefit is non-negative. We work with expectations because the actual (final) misbehavior benefit is not known to the node (or anyone else) at the time that it observes traffic; it can be computed only when the measurement ends. We consider all possible cases for $t > 0$ and $g \in (0,1)$ and show that Ineq. 4.1, 4.2 are sufficient to make the expected misbehavior benefit negative:

$$MB_d < 0 \tag{4.14}$$

**Case a:** $t \geq \kappa$

In this case the exact computation of the misbehavior benefit is not easy, but it is enough to commute an upper bound and show that it is negative.

Since $t \geq \kappa$, the monitor's delay estimate cannot be less than the delay of the good path (achieved when all samples happen to follow the good path) plus $\kappa$:

$$\hat{d}_{mis} \geq d_g + \kappa \tag{4.15}$$

Given now the definition of the misbehavior benefit, we get the following upper bound:

$$\begin{aligned} MB_d &:= \hat{d}_{hon} - \hat{d}_{mis} \\ &= g \cdot d_g + (1-g) \cdot d_b - \hat{d}_{mis} \\ &\leq \left(1-g\right)\left(d_b - d_g\right) - \kappa \\ &\overset{(4.1)}{<} -g\Delta_d \leq 0 \end{aligned}$$

Thus, Ineq. 4.1 implies Ineq. 4.14, i.e., the condition about $\kappa$ given by Ineq. 4.1 is sufficient to guarantee that a prioritization attack of Case (a) is unsuccessful.

**Case b:** $t < \kappa$

For this case, we divide the packets forwarded by a misbehaving node $i$ that performs a prioritization attack into two categories:

- Category I: The packets for which disclosure happens within $t$. Node $i$ knows that these are excluded from the sampling process, because they fall into the quiet period $\kappa$, and it forwards them over the bad path.

- Category II: The packets for which disclosure does not happen within $t$ (hence, each of them is buffered for $t$ before being forwarded). Node $i$ knows that a subset of these will not be excluded, and a subset of the non-excluded ones will be sampled, but it does not know which subsets these are. Hence, it forwards as many as it can over the good path and the rest over the bad path.

**Misbehavior Benefit Function**

Let $\Delta_d = d_b - d_g$ be the delay difference of the two paths and $m(t)$ denote the expected fraction of packets for which disclosure happens within $t$ (packets of Category I described above). Let also $t^\star$, denote the buffering time at which the bad path is saturated with packets whose disclosure happens during their buffering period and therefore their exclusion for the sampling process is verified by the misbehaving node. I.e.:

$$m(t^\star) = 1 - g \tag{4.16}$$

Given the attack model that is described in §4.4.1, for any $g \in (0,1)$, the expected misbehavior benefit takes the following form w.r.t. $t$:

$$MB_d = \begin{cases} \frac{gm(t)}{1-m(t)}\Delta_d - t & \text{if } 0 < t \leq t^\star \\ (1-g)\Delta_d - t & \text{if } t \geq t^\star \end{cases} \tag{4.17}$$

**Computation of the upper branch of Eq. 4.17:**

Due to its definition, $m(t)$ is an increasing function of $t$. Hence:

$$\begin{aligned} & m(t) \leq m(t^\star) \\ \overset{(4.20)}{\Longleftrightarrow} & m(t) \leq 1 - g \\ \Leftrightarrow & \frac{g}{1-m(t)} \leq 1 \end{aligned} \tag{4.18}$$

Ineq. 4.18 shows that the packets of Category II exceed or exactly match the capacity of the good path. In this case, the node forwards as many of these packets as it can (i.e. $\frac{g}{1-m(t)}$) over the good path and the rest (i.e. $1 - \frac{g}{1-m(t)}$) over the bad path. It does so, because this is the only way it never exceeds the capacities of both paths and it maximizes the number of packets of Category II (from which the sample will be selected) that follow the good path.

Note that the exact algorithm, with which the node manages to perform the above policy at line rates, is not required for this proof. I.e. we consider here the best possible misbehaving scenario that is offered to the node because of the domain's network conditions. This makes our analysis a *worst-case* approach.

The monitor always produces an accurate delay estimate, based on a sample of packets of

Category II. Hence, the expected[6] value of estimated mean delay of the domain is:

$$\hat{d}_{mis} = \frac{g}{1-m(t)}\left(d_g + t\right) + \left(1 - \frac{g}{1-m(t)}\right)(d_b + t) \tag{4.19}$$

Based on Eq. 4.19, we obtain the upper branch of Eq. 4.17:

$$\begin{aligned} MB_d =& g \cdot d_g + (1-g)\cdot d_b - \hat{d}_{mis} \\ \overset{(4.19)}{=}& -\frac{g - gm(t) - g}{1-m(t)} d_g - t \\ &+ \frac{1 - g - m(t) + gm(t) - 1 + g + m(t)}{1-m(t)} d_b \\ =& \frac{gm(t)}{1-m(t)} \Delta_d - t, \ \ \forall t \in \left[0, t^\star\right]. \end{aligned}$$

**Computation of the lower branch of Eq. 4.17:**

At the buffering time $t = t^\star$, the capacity of the bad path is saturated with packets of Category I, and, because of Eq. 4.20, the misbehavior benefit becomes $MB_d(t^\star) = \left(1-g\right)\Delta_d - t^\star$. I.e. all sampled packets follow the good path.

If the node buffers for time $t > t^\star$, then the packets will be delayed for more time, while all non-excluded packets will continue to follow the good path. So, $\hat{d}_{mis} = d_g + t$ and thus $MB_d = \left(1-g\right)\Delta_d - t$.

**Upper bound for Eq. 4.17:**

We can compute an upper bound for the expected misbehavior benefit using an upper bound for the expected fraction of packets of Category I, i.e., $m(t)$.

Given a disclosure probability $\delta_r$ and a buffering period $t$, for any packet from flow $F$ that arrives at time $t_0$, the probability that its disclosure packet arrives within period $t$ follows the geometric distribution with parameter $\delta_r$, and it is equal to: $1-(1-\delta_r)^{n_F[t_0, t_0+t]}$, where $n_F[t_0, t_0+t]$ is the number of packets from flow $F$ that fall inside the time interval $(t_0, t_0+t]$. Therefore, since $n_F[t_0, t_0+t]$ is random variable, the actual fraction of packets of Category I is also a random variable.

An upper bound for the expected fraction of packets of Category I $m(t)$ can be found using our stationarity assumption (§1.5) and Jensen's Inequality: Due to that flow-$F$ packet arrivals

[6]The exact estimate cannot be known in advance because it is produced from the collected sample, but the estimate will be very close to the expected value. This is because the number of packets of Category II is smaller than the total flow traffic. This, in turn, means that the collected sample (as computed in §4.6.1) is larger than necessary to estimate the average delay of those packets with adequate accuracy. By fixing $\gamma$ and $\epsilon$ to reasonable values (e.g. $\gamma = 95\%$ and $\epsilon = 10\%$), the accuracy of $\hat{d}_{mis}$ is, almost surely, adequately high.

form a stationary and ergodic point process with intensity $r$, $n_F[t_0, t_0+t]$ is a random variable whose distribution is invariant under time shifts, i.e., it is the same for any $t_0$.

Also, from the definition of the intensity $r$ (which is the expected number of points per time unit [42]), the expected value of $n_F[t_0, t_0+t]$ is: $\mathbb{E}[n_F[t_0, t_0+t]] = r\,t$, for any $t_0 \geq 0$.

Last, since the function $1-(1-\delta_r)^{n_F[t_0,t_0+t]}$ is concave w.r.t. $n_F[t_0, t_0+t]$, from Jensen's Inequality we get:

$$m(t) \leq 1-(1-\delta_r)^{rt} \tag{4.20}$$

Given Ineq. 4.20, we now obtain an upper bound for the expected misbehavior benefit $MB_d$:

$$MB_d \leq \begin{cases} \frac{g\left(1-(1-\delta)^{rt}\right)}{(1-\delta)^{rt}}\Delta_d - t & \text{if } 0 < t \leq t^\star \\ \left(1-g\right)\Delta_d - t & \text{if } t \geq t^\star \end{cases} \tag{4.21}$$

We denote the right hand side of the above with $\max MB_d$. I.e.:

$$\max MB_d = \begin{cases} \frac{g\left(1-(1-\delta)^{rt}\right)}{(1-\delta)^{rt}}\Delta_d - t & \text{if } 0 < t \leq t^\star \\ \left(1-g\right)\Delta_d - t & \text{if } t \geq t^\star \end{cases} \tag{4.22}$$

**Analysis of Eq. 4.22:**

The upper branch of Eq. 4.22 is a convex function of $t$, while the lower branch is a decreasing function of $t$. To see why, we provide hereunder the first and second derivatives of $\max MB_d$. One can verify that the second derivative (Eq. 4.24) of the upper branch is non-negative and the first derivative (Eq. /4.23) of the lower branch is negative.

$$\frac{d(\max MB_d)}{dt} = \begin{cases} g\Delta_d\left(-r\,(1-\delta_r)^{-rt}\ln(1-\delta_r)\right) - 1 & , 0 < t \leq t^\star \\ -1 & , t \geq t^\star \end{cases} \tag{4.23}$$

$$\frac{d^2(\max MB_d)}{dt^2} = \\ = \begin{cases} g\Delta_d\left(-r\ln(1-\delta_r)\right)^2(1-\delta_r)^{-rt} & , 0 \leq t \leq t^\star \\ 0 & , t \geq t^\star \end{cases} \tag{4.24}$$

Since the maximum misbehavior benefit $\max MB_d$ is a continuous function of $t$, that is convex in $\left[0, t^\star\right]$ and decreasing for $t \geq t^\star$, then, for any $g \in [0,1]$, the function takes its maximum value at one of the edges of the support of its upper branch, i.e. $t = 0$ or $t = \min\left\{\kappa, t^\star\right\}$ (because

of the general assumption of Case b: $t < \kappa$). At $t = 0$, $MB_d(0) = 0$, while at $t = t^\star$, the sign of $MB_d$ depends on the actual values of $g$, $\delta_r$ and $\kappa$.

Moreover, the maximum misbehavior benefit $\max MB_d$ is decreasing with $t^\star$. We obtain a lower bound for $t^\star$ directly from Ineq. 4.20 and Eq. 4.16:

$$t^\star \geq \frac{\ln g}{r \ln(1-\delta_r)} \tag{4.25}$$

Thus, to show that the conditions of lemma 4.4.2 about $\kappa$ and $\delta_r$ (i.e. Ineqs. 4.1 and 4.2) imply Ineq. 4.14, it is sufficient to show that, when they hold, then:

$$\begin{aligned} &\max MB_d\left(\min\left\{\kappa, \min\left\{t^\star\right\}\right\}, g\right) \\ =&\max MB_d\left(\min\left\{\kappa, \frac{\ln g}{r\ln(1-\delta_r)}\right\}, g\right) < 0, \ \ \forall g \in [0,1] \end{aligned} \tag{4.26}$$

In this case, the expected misbehavior benefit $MB_d$ is always non-positive, for any possible $t$ and $g$. We will show this, by considering two sub-cases w.r.t. $g$:

**Case b1:** $(1-\delta_r)^{r\kappa} \geq g$

Using this assumption, we get:

$$\frac{\ln g}{r\ln(1-\delta_r)} \geq \kappa \Rightarrow \min\left\{\kappa, \frac{\ln g}{r\ln(1-\delta_r)}\right\} = \kappa. \tag{4.27}$$

Therefore,

$$\begin{aligned} (4.26) \overset{(4.22),(4.27)}{\Longleftrightarrow} MB_d(\kappa, g) &< \left(1-g\right)\Delta_d - \kappa \\ &\overset{(4.1)}{<} -g\Delta_d \leq 0. \end{aligned}$$

Thus, Ineq. 4.1 implies Ineq. 4.26, i.e, the condition about $\kappa$ given by Ineq. 4.1 is sufficient to guarantee that a prioritization attack of Case (b1) is unsuccessful.

**Case b2:** $(1-\delta_r)^{r\kappa} < g$

Using this assumption, we get:

$$\frac{\ln g}{r\ln(1-\delta_r)} < \kappa \tag{4.28}$$

Since $\min\left\{t^\star\right\} = \frac{\ln g}{r\ln(1-\delta_r)} < \kappa$, no conditions about $\kappa$ exist that provide an upper bound for the maximum possible benefit at $t = \min\left\{t^\star\right\}$. At that time, the node manages to forward over the bad path only excluded packets, which equivalently ensures that all the remaining packets, among which the non-excluded ones, are forwarded over the good path. Moreover, at that

time, the maximum benefit depends on the values of $g$ and $\delta_r$ and it cannot be bounded only through the choice of $\kappa$. It is necessary to properly set $\delta_r$, too.

We consider again Ineq. 4.26 and show why 4.1 and 4.2 are sufficient to guarantee the requirement.

$$\begin{aligned}(4.26) &\overset{(4.22),(4.28)}{\Longleftrightarrow} (1-g)\Delta_d - \frac{\ln g}{r\ln(1-\delta_r)} < 0 \\ &\Leftrightarrow \frac{-1}{r\Delta_d \ln(1-\delta_r)} > \frac{g-1}{\ln g} \end{aligned} \tag{4.29}$$

First, we need to prove the following auxiliary lemma:

**Lemma 4.7.2.** $\ln g \leq g-1$

*Proof.* Take the Taylor expansion of $\ln g$ around $g_0 = 1$:

$$\begin{aligned}\ln g &= \ln g_0 + (g-g_0)\left.\frac{\mathrm{d}\ln g}{\mathrm{d}g}\right|_{g=g_0} \\ &\quad + \frac{1}{2}(g-g_0)^2 \left.\frac{\mathrm{d}\ln g}{\mathrm{d}g^2}\right|_{g=\xi}, \text{ for some } \xi \in [g_0, g] \\ &= \ln(1) + (g-1)\frac{1}{1} + \frac{1}{2}(g-1)^2\left(-\frac{1}{\xi^2}\right)^1\end{aligned}$$

The last term of the above is always non-positive. Thus,

$$\ln g \leq 0 + (g-1)$$

□

Because of lemma 4.7.2, we get that $\frac{g-1}{\ln g} \leq 1$.

Now, we go back to Ineq. 4.29. To show that the inequality is satisfied, it is enough to show that the left hand side (LHS) of the inequality satisfies the following: $LHS(Ineq.\ 4.29) > 1$, for some $\delta_r \in (0,1)$. But,

- if $(1-\delta_r)^{r\kappa} \geq \frac{1}{e}$ (i.e. Ineq. 4.2 holds), then:

$$\begin{aligned}&(1-\delta_r)^{r\kappa} \geq \frac{1}{e} \\ \Leftrightarrow\ &\ln\left((1-\delta_r)^{r\kappa}\right) \geq -1 \\ \Leftrightarrow\ &r\kappa\ln(1-\delta_r) \geq -1 \\ \Leftrightarrow\ &\frac{-1}{r\kappa\ln(1-\delta_r)} \geq 1.\end{aligned} \tag{4.30}$$

- if additionally $\kappa > \Delta_d$ (i.e. Ineq. 4.1 holds), then:

$$\begin{aligned}
& \frac{1}{\Delta_d} > \frac{1}{\kappa} \\
\Leftrightarrow \quad & \frac{-1}{r\Delta_d \ln(1-\delta_r)} > \frac{-1}{r\kappa \ln(1-\delta_r)} \\
\overset{(4.30)}{\Longrightarrow} \quad & \frac{-1}{r\Delta_d \ln(1-\delta_r)} > 1 \\
\overset{(lem.4.7.2)}{\Longrightarrow} \quad & \frac{-1}{r\Delta_d \ln(1-\delta_r)} > \frac{g-1}{\ln g} \rightarrow (4.29, 4.26) \text{ hold}
\end{aligned}$$

Hence, if an attack is considered unsuccessful according to Ineq. 4.14, then Ineq. 4.1 and 4.2 are sufficient to guarantee that an attack of Case (b2) is unsuccessful.

In conclusion, by considering both cases (a) and (b), we have showed that for any $t > 0$ and any $g \in (0,1)$, Ineq. 4.1 and 4.2 are sufficient to make a prioritization attack unsuccessful.

**Proof of Inequality 4.4**

The inequality can be derived directly from Eq. 4.23 and 4.2:

$$\frac{d(MB_d)}{dt} \leq \begin{cases} \frac{g}{(1-\delta_r)^{-rt}} \Delta_d (-r \ln(1-\delta_r)) - 1 & , t \leq t^\star \\ -1 & , t \geq t^\star \end{cases}$$

$$\overset{(4.2)}{\Longrightarrow} \frac{d(MB_d)}{dt} \leq \begin{cases} \frac{g}{(1-\delta_r)^{-rt}} \Delta_d \frac{1}{\kappa} - 1 & , t \leq t^\star \\ -1 & , t \geq t^\star \end{cases}$$

And because for $t \leq t^\star \overset{(4.25)}{\Longleftarrow} g < (1-\delta_r)^{-rt}$, the above results in:

$$\begin{aligned}
& \frac{d(MB_d)}{dt} \leq \begin{cases} \frac{\Delta_d}{\kappa} - 1 & , t \leq t^\star \\ -1 & , t \geq t^\star \end{cases} \\
\overset{\frac{\Delta_d}{\kappa} > 0}{\Longrightarrow} \quad & \frac{d(MB_d)}{dt} \leq \frac{\Delta_d}{\kappa} - 1
\end{aligned}$$

### 4.7.7 Proof of lemma 4.5.1

To prove this lemma it is enough to prove that the expected sample size after time $T$ w.r.t a flow with rate $r$ is:

$$r \cdot T \cdot \left[ (1-\delta_r)^{r\kappa} - (1-\delta_r)^{\frac{r}{R}\beta} \right] \cdot \sigma$$

Let the number of packets that are observed at a node in a measurement period $T$ be $M$. Then

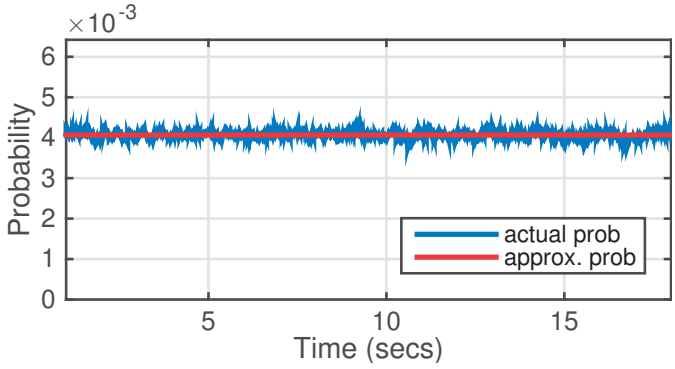



Figure 4.10 – Actual sampling probability.

the corresponding sample size $n_M$ in time $T$ will be have the following expected value:

$$\mathbb{E}[n_M] = \sum_{j=1}^{M} 1_{\{\text{j-th pkt is sampled}\}} \cdot p_j \tag{4.31}$$

where $p_j$ is the sampling probability of $j$-th packet of the flow to be sampled.

For this proof, we approximate the actual number of packet arrivals with the expected values. The sampling probability $p_j$ of each packet $j$ depends only on the exact number of flow $F$'s packet arrivals in a period of $\kappa$ time following $j$, and the exact number of packet arrivals before $j$ is dropped from the circular buffer. Because of stationarity (§1.5), the expected values of these numbers are $r \cdot \kappa$ and $\frac{r}{R} \cdot \beta$, respectively. Also, due to ergodicity and the very high packet rates that are considered in this paper ($R$'s that correspond to rates higher than OC-12), the actual number of packet arrivals converges to its expected value at a sub-second granularity (less than 100msec, which are the typical values for $\kappa$). So, the exact number of packets that arrive in time $\kappa$ can be approximated by $r \cdot \kappa$. Therefore, we can approximate $p_j$ with:

$$p_j \approx \left[ (1-\delta_r)^{r\kappa} - (1-\delta_r)^{\frac{r}{R}\beta} \right] \cdot \sigma,$$

where the term $(1-\delta_r)$ comes from the Bernoulli arrival process of the disclosure packets over the $F$ packet arrivals.

To verify the latter approximation, we have computed the "actual" sampling probability of all packets in real flow traces. We used the same flow traces as the ones in our evaluation section, and for every packet $j$ in the trace we have computed $p_j$ without making the above mentioned approximations of the number of packets in time $\kappa$ or in the buffer $\beta$. In all cases, we found that the approximation works very well in practice. Fig. 4.10 depicts an example 20-sec trace case (Equinix-chicago April 2016). The actual and the approximated probabilities for all packets are indeed very close packets.

To complete the proof, we may also take $M \approx r \cdot T$. Due to the larger time $T$, this approximation is even better.

Using the above approximations into Eq. 4.31, we finally get:

$$\begin{aligned}\mathbb{E}[n_M] &= M \cdot \left[ (1-\delta_r)^{r\kappa} - (1-\delta_r)^{\frac{r}{R}\beta} \right] \cdot \sigma \\ &= r \cdot T \cdot \left[ (1-\delta_r)^{r\kappa} - (1-\delta_r)^{\frac{r}{R}\beta} \right] \cdot \sigma\end{aligned}$$

which implies the lemma's statement.

Instead of the approximation for the sampling probability, we could use a similar approach to the one that was followed for the derivation of Ineq. 4.20 in Section §4.7.6, to find an lower bound for $\mathbb{E}[n_M]$ using Jensen's inequality: $\mathbb{E}[n_M] \geq \left[ (1-\delta_r)^{r\kappa} - (1-\delta_r)^{\frac{r}{R}\beta} \right]$ However, as showed in our experiments, due to the fast ergodic convergence this bound is tight enough that an equality can be used instead of inequality.

# 5 Topology-obfuscation reporting

In this chapter, we describe how to achieve goal G2: the traffic receipts that a domain emits should not disclose information about the domain's internal topology. Internet service providers typically consider their internal topology to be private information. Therefore, we study "topology-discovery" attacks, where the monitor (or any adversary who has access to the same information as the monitor, e.g., a government that subpoenas the receipts collected by the monitor) tries to use the receipts emitted by a domain in order to reverse-engineer the domain's internal topology. Starting from the observation that internal topology can be revealed through the diversity of intra-domain path delays, we design receipts that obfuscate this information. A key challenge we face is that topology obfuscation requires a special kind of receipt manipulation, which is something that our incentive-compatible reporting (Chapter 3) was designed to penalize. We show that it is possible to obfuscate topology and at the same time emit receipts that enable the monitor to make correct decisions and yield correct domain ratings.

First, we state our trust model (§5.1). Then we describe our general approach (§5.2) and two specific algorithms that implement it (§5.3). We close with a brief experimental evaluation (§5.4).

## 5.1 Trust model and problem statement

The receipts that a domain emits can leak information that is otherwise private: It has been shown that an entity with access to measurements of a network's paths can use network tomography to infer the network's topology [30, 29, 19, 27, 57]. Today, obtaining such measurements is typically hard, as it requires coordination and trust between large numbers of end-users. Our transparency system, however, changes this, as it requires the participating domains themselves to publish measurements of their network paths.

In the context of this chapter, the monitor (or any adversary who has access to the same information as the monitor) may misbehave by launching "topology-discovery" attacks: consider

the receipts that a domain emits and use network tomography in an effort to infer the domain's internal topology. We assume that the monitor uses a perfect tomographic algorithm: if a domain emits correct receipts as described in the last two chapters, the monitor correctly reconstructs the domain's internal topology without error. We make this assumption because we want to base our solution on fundamental weaknesses of network tomography that pose hard limits on how well an adversary can infer network topology, as opposed to imperfections of particular tomographic algorithms.

Given this trust model, we want to design an algorithm for producing receipts that enables the monitor to make correct decisions (as specified in Chapter 3), but prevent the monitor from inferring internal domain topology.

We restate the assumption from Section §1.5 that is relevant to this chapter: Domains know the true loss and delay of their own inter-domain links. For any given aggregate $G$ and any given domain $z$ on $G$'s path, the neighbor domains of $z$ know the true loss/delay performance of their inter-domain link with $z$ with respect to $G$'s packets.

## 5.2 Approach

In this section, we describe our approach. First, we describe how topology inference works (§5.2.1) and how we can prevent it in principle (§5.2.2). Then, we explain the rationale behind our algorithms (§5.2.3).

### 5.2.1 Topology inference

Topology inference exploits the diversity of the pairwise similarities (either covariances or correlations) of network-path delay vectors. An (intra-domain network) path is defined by the pair of its entry and exit nodes. The delay vector of a path is the vector of the delays experienced by the sequence of sampled packets that entered and exited, respectively, at these nodes. Consider the topology in Fig. 5.1. Packets entering the domain at node $a$ and exiting at either

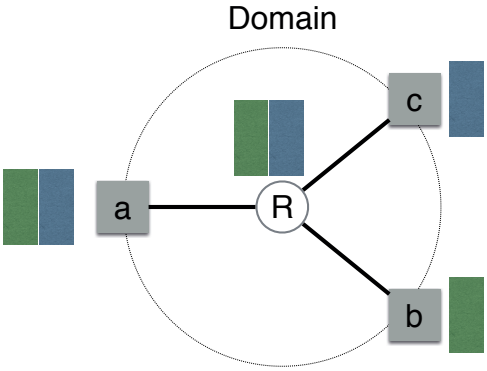



Figure 5.1 – Example of topology inference on a simple logical topology.

node $b$ or node $c$ travel along the same path until router $R$. Consider two packets that enter back-to-back at $a$ and exit, respectively, at $b$ and $c$. Topology inference assumes that these two packets experience highly correlated delays before router R (because they encounter the same network conditions) and uncorrelated delays after router R (because they follow different paths after R). Given these assumptions, the covariance of the path delays of back-to-back packets that follow paths a,b and a,c leaks information about the amount of shared physical topology between the two paths. Covariance, however, is sensitive to the delay variances of the involved paths. Hence, topology-inference algorithms estimate the "pairwise similarity" (the amount of shared physical topology between two paths) based on the correlation coefficient of the delay vectors of the two paths: $corr(\{a,b\},\{a,c\}) = \frac{cov(\{a,b\},\{a,c\})}{\sqrt{var(\{a,b\})var(\{a,c\})}}$.

Given the pairwise similarities of a domain's paths, a topology-inference algorithm proceeds in two steps: First, for each set of paths with a common entry node, it reconstructs the likeliest tree topology based either on agglomerative clustering [27, 57, 16, 30, 29] or maximum likelihood [19]. Agglomerative clustering resolves the hierarchical clustering of a set of objects with pairwise similarity values by finding the maximum similarity element and merging the rows/columns of the similarity matrix corresponding to those two end-hosts, then finding the next maximum element and merging those rows/columns of the similarity matrix to the new maximum element. In contrast to clustering strategies, maximum likelihood techniques use a global maximum-penalized likelihood criterion for topology identification. Second, the algorithm merges all the reconstructed trees to reconstruct the entire topology [20].

### 5.2.2 Topology Obfuscation

Our approach is to alter the timestamps of (some of) the domain's emitted receipts, so that any pairwise similarities that the monitor might compute are negligible. By doing so, we target the fundamental reason that enables topology inference, which is the existence of different pairwise similarities between network path delays. Alternatively, we could have exploited the sources of error of specific topology-inference algorithms. However, a domain cannot know in advance which algorithm will be used to infer its topology. By reporting pairwise similarities (close to) 0, it ensures that any such algorithm will fail.

We face two challenges:
(1) Our solution must scale with the number of the domain's intra-domain paths. This precludes a straightforward solution that considers all path pairs and all possible timestamp changes (and tries to find one that makes all pairwise similarities 0). Instead, we design an obfuscation algorithm that can be applied individually to each path or flow.

(2) Our solution must not penalize domains for obfuscating their topologies. In Ch. 3, we designed decision and rating functions precisely to penalize domains that manipulate their receipts. Hence, we need a solution that somehow enables domains to change the timestamps on some of their receipts without affecting the decision and rating functions, hence without getting penalized.

**Algorithm 4** ChangeTimestamps (*flowID*)

| | |
|---|---|
| $\mathscr{R}_i$ | all receipts for *flowID* that initially computed by entry node $i$ |
| $\mathscr{R}_o$ | all receipts for *flowID* emitted by exit node $o$ |
| $\mathscr{R}'_i$ | all new receipts for *flowID* emitted by entry node $i$ |
| *Obfuscate*() | algorithm that creates topology-obfuscation (altered) delay vector |
| $j$ | index for the delay vector |

1: $delaySamples \leftarrow \{\}$
2: **for** each $r_o \in \mathscr{R}_o$ **do**
3: Find receipt $r_i$ from node $i$ with $r_i.digest = r_o.digest$.
4: $delaySamples(j) \leftarrow r_o.timestamp - r_i.timestamp$
5: $j \leftarrow j+1$
6: **end for**
7: $delaySamples' \leftarrow Obfuscate(delaySamples)$
8: **for** each $r_o \in \mathscr{R}_o$ **do**
9: Find receipt $r_i$ from node $i$ with $r_i.digest = r_o.digest$.
10: $r_i.timestamp \leftarrow r_o.timestamp - delaySamples'(j)$
11: $j \leftarrow j+1$
12: **end for**
13: Output $\mathscr{R}'_i$ and $\mathscr{R}_o$.

Alg. 4 presents our "altered-timestamps" approach: It takes as input the sequences of the true receipts $\mathscr{R}_i$ and $\mathscr{R}_o$ that a domain's entry node $i$ and exit node $o$ would emit w.r.t. flow $F$, and it outputs the sequences of receipts that the two nodes should emit in order to obfuscate the domain's internal topology. For each receipt that the two nodes compute for the same sampled packet, Alg. 4 computes the intra-domain delay for that packet and adds it to the delay vector *delaySamples* (lines 1-6). Then, an *Obfuscate*() algorithm alters the elements of the delay vector so that they leak no information about the domain's internal topology, i.e., the covariance between this delay vector and the delay vector of any other flow that exited the domain from another exit node is close to 0 (line 7). Finally, Alg. 4 alters the timestamps of the entry node's receipts, such that the sequence of sampled packets appear to have experienced the obfuscating delays $delaySamples'$ (lines 8-13).

### 5.2.3 Rationale

Given a flow $F$, Alg. 4 changes the timestamps emitted by the flow's entry node. We made this choice for two reasons:

First, by hiding the true entry timestamps from the monitor, we prevent it from identifying the packets that would be most helpful for topology inference. Even through the monitor has access to all the receipts from all the flows observed by a domain, if it does not know the true entry timestamps, it cannot identify packet pairs like the one in Fig. 5.1 (packets that enter the domain back-to-back at the same entry point and exit at different points).

Second, changing only the entry timestamps is simpler, in that each change affects the performance of a single inter-domain link in a predictable way. The alternative would be to change both the entry and exit timestamps, which would affect the performance of both of the domain's inter-domain links, hence the ratings of both of the domain's neighbors. Controlling the impact of the change on the ratings of all the involved domains would require complex coordination among the domains.

We discuss the *Obfuscate*() algorithm in the next section, but state here the conditions that it needs to satisfy: The last and most important point is how the algorithm works.

1. Applying *Obfuscate*() to each flow observed by a domain should make the pairwise covariances of all flow delay vectors be (close to) 0. This is equivalent to the following condition: once normalized such that their means are zero, the altered delay vectors of all flows should form an (almost) orthogonal basis.

2. *Obfuscate*() should not significantly impact the monitor's decisions and the domains' ratings. This precludes straightforward solutions, e.g., based on Gram-Schmidt orthogonalization.

## 5.3 Solution

We present two algorithms for topology obfuscation that are built on scrambling: they take as input a vector of intra-domain delays experienced by a sequence of sampled packets; and they produce as output a permutation of these delays. Hence, by construction, a domain that uses our algorithm alters its receipt timestamps in a way that does not affect the intra-domain delay distribution that can be estimated from the domain's receipts. This approach has two advantages related to our condition (2): First, it does not affect the monitor's delay estimates (mean, variance, percentile) of any intra-domain path. Second, it does not affect the monitor's mean delay estimate of any inter-domain link. However, depending on how the scrambling occurs, it may affect the monitor's delay variance and percentile estimates of inter-domain links.

We avoid random scrambling because it does not scale. A randomly chosen permutation of the input delays will most likely not meet our two conditions: it will not make all the pairwise similarities close to 0, and it will affect the monitor's delay variance estimates of inter-domain links. Hence, to meet our two conditions, we would need to consider all pairs of flows and search over all possible combinations of permutations. If the length of the input delay vectors is $N$ (equal to the sample size) and the number of flows is $N_F$, then the complexity of this search is $O\left(N^2 \cdot \binom{N_F}{2}\right)$.

To reduce complexity, each algorithm scrambles its input delay vector such that the outcome is as close as possible to another, specially crafted vector. Let $D$ be the input delay vector (which is computed from the true receipts of the entry and exit nodes), and $Q$ be another, specially

crafted vector of equal size. Also, let $C : D \times Q \mapsto \mathbb{R}$ be a cost function for the assignment.

**Definition 5.3.1** (Assignment problem[1])**.** *Given $D$, $Q$ and $C$, find a bijection $b : D \mapsto Q$ that minimizes the aggregated cost: $\sum_{d \in D} C(d, q)$.*

The solution of the above problem assigns every element of $D$ to an element of $Q$. Let $D[j]$, $j = 1, \ldots, N$, be the elements of $D$ and $Q[k]$, $k = 1, \ldots, N$, be the elements of $Q$. The minimizing bijection of the solution is just a mapping of the indices $j$ to the indices $k$. As a result, we obtain a new order of the elements of $D$ that "follows" the indices $k$ of $Q$. This permuted version of $D$, which we denote with $D'$, is the output of the *Obfuscate*() algorithm, i.e. *Obfuscate*$(D) = D'$. We say that *Obfuscate*$(D)$ "maps" its input vector $D$ to vector $Q$ with minimum cost, by reordering its elements.

In our first algorithm, *Obfuscate*1(), vectors $D$ are mapped to vectors $Q$ that are chosen from an orthonormal basis (i.e. meet our condition (1) by construction) and we design the cost function $C$ such that $D'$ is as close as possible to a chosen $Q$. In our second algorithm, *Obfuscate*2(), each vector $D$ is mapped to a vector $Q$ that contains the delays between entry node $i$'s predecessor node (the exit node of the previous domain) and exit node $o$ of the path which is related to $D$. And we design the cost function $C$ such that the output vector comes as close as possible to meeting our conditions (1) and (2). We can minimize the cost function in polynomial time using the standard solution to the assignment problem: the Hungarian algorithm [48], which has complexity $O(N^3)$.

### 5.3.1 *Obfuscate*1(): Assignment to Fourier orthogonal basis

This algorithm maps the input delay vector to a vector that belongs to the Fourier orthogonal basis. So, for each input delay vector $D$, $Q$ is a real-time cosine signal of frequency $f$, where $f = 0, \ldots, N-1$. The frequencies are chosen randomly but differently for each $D$. The cost function $C$ is the L2-norm of the vector $(D - Q)$. Hence, the assignment algorithm minimizes the mean square error of reporting $D' =$ *Obfuscate*$(D)$ instead of $Q$: since $Q$ is a cosine of a single frequency, $D'$ is a noised version of $Q$, where the noise is minimized.

The intuition behind this algorithm is the observation that the covariance of two delay vectors is almost equivalent to their dot product (they differ only by a fixed factor $1/N$). Given that each scrambled vector is "close" to a vector that belongs to the basis (a cosine signal with a given frequency), then the covariance of any two scrambled vectors will be "close" to 0.

Any orthogonal basis could work; we chose the Fourier basis because of its flexibility and its well-established theory: Using Discrete Fourier Transforms (DFT), we can design the basis to contain only a small set of frequencies that are far apart from one another such that any noise

[1] Another way to formulate the assignment problem is by considering a complete bipartite graph with $N$ vertices corresponding to the elements $d$ of the delay vector $D$, $N$ vertices corresponding to the elements $q$ of special vector $Q$, and where each edge has a non-negative cost $C(d, q)$. Then, an assignment is a perfect matching with minimum cost.

effect is amortized. Also, given any two delay vectors, $D'_1$ and $D'_2$, we can reason about their dot product (and their covariance) in the time domain $\langle D'_1, D'_2 \rangle$ by looking at the product of their DFT spectra (Parceval Theorem).

### 5.3.2 *Obfuscate2*()**: Assignment to extended-path delays**

In this algorithm, if the input vector $D$ contains the delays w.r.t. flow $F$ between node $i$ and node $o$, the vector $Q$ contains the delays w.r.t. $F$ between node $i$'s predecessor and node $o$. We specify the cost function as a matrix where element $[k, j]$ indicates the cost of assigning element $D[j]$ to element $Q[k]$: $C(D,Q)[k,j] = \Big(\big(Q[k] - D[j]\big)^2 - var(Q-D)\Big)^2 + D[j] \cdot D[k]$.

The intuition behind this algorithm is the following: with *Obfuscate2*() we want to achieve obfuscation without affecting the inter-domain link delay variance $var\left(X'\right)$, according to our condition (2). However, we cannot add directly $var\left(X\right) - var\left(X'\right)$ to the minimization objective of the assignment problem, because $X'$ depends on $D'$, which is the assignment solution; we need a vector that remains unchanged by obfuscation. The only vector that has this property and is related to both $X$ and $X'$ is: $Q = (D + X) = \left(D' + X'\right)$.

The first term of the cost function $\Big(\big(Q[k] - D[j]\big)^2 - var(Q-D)\Big)^2$ helps minimize the change in the estimated variance of the inter-domain link between node $i$'s predecessor and node $i$ that is due to obfuscation; this minimizes the impact of obfuscation on the decision and rating functions, according to our condition (2). The second term of the cost function ($d[j] \cdot d[k]$) helps minimize the dot product between true and scrambled delay vectors, hence make the two as less correlated as possible; this targets to achieve close-to-zero pairwise similarities, according to our condition (1).

## 5.4 Experimental evaluation

We now confirm that our two obfuscation algorithms indeed obfuscate the pairwise similarities between delay vectors, i.e., the information that enables topology inference.

We use synthetic tree topologies of different sizes and shapes: complete balanced binary trees, complete unbalanced binary trees and ternary trees of depth 2. In each topology, we assign to each link a fixed propagation delay and a random queueing delay per packet. Queuing delays are chosen from a Weibull distribution with with scale parameter 0.5 and a shape parameter that varies uniformly in the interval $[0.6, 0.82]$ for each link, according to the analysis of single-hop queuing delays in [51].

In each experiment, we emulate the scenario where a sequence of packets from multiple flows enter domain $z$ at the same entry node $i$ (the root of one of our tree topologies). Each flow follows a different path through $z$ and exits at a different node (a leaf of the topology). For each flow $F$, domain $z$ runs Alg. 4, which alters the timestamps of $F$'s receipts emitted by node $i$.

This affects the monitor's delay estimates for: (a) The inter-domain link that $F$ traverses before entering $z$. We denote by $X_F$ and $X'_F$, respectively, the original and scrambled vectors of this inter-domain link w.r.t. $F$. (b) The intra-domain segment that $F$ traverses within $z$. We denote by $D_F$ and $D'_F$, respectively, $z$'s original and scrambled intra-domain delay vectors w.r.t. $F$.

At the end of each experiment, we compute how useful our obfuscation algorithms were in preventing topology inference. In particular, we compute the covariance of each pair of original delay vectors $D_{F_i}$ and $D_{F_j}$ and compare it to the covariance of the corresponding scrambled delay vectors $D'_{F_i}$ and $D'_{F_j}$. Every time the original vectors have significant covariance while the scrambled vectors do not, our obfuscation algorithms were useful. Also, to measure the impact on the inter-domain link we compute and compare the variances of each $X_F$ and $X'_F$.

We consider two types of scenarios:

[I] $X_F$ is independent from $D_F$. In this case, the inter-domain delays are chosen similarly to the intra-domain delays, based on a Weibull distribution.

[II] $X_F$ is correlated to $D_F$ with correlation coefficient $\rho \in (-1,1)$.

The purpose of the type [II] scenarios is to explore the limits of Obfuscate2(). Recall that our second obfuscation algorithm maps each flow's original intra-domain delay vector $D_F$ to vector $(X_F + D_F)$. Even though the algorithm tries to find a mapping such that $D'_F$ is not correlated to $D_F$ (through the second term of its cost function), this becomes increasingly hard as the correlation between $D_F$ and $X_F$ increases. Hence, we would like to assess the algorithm's performance as a function of the correlation coefficient $\rho$. In practice, we expect $\rho < |0.2|$, i.e., $D_F$ and $X_F$ to be loosely, if at all, correlated, because of the high rate and diversity of the traffic that crosses inter-domain links.

All our topologies yielded similar results. Hence, we present results from only one topology: a complete balanced tree of depth 5, with 31 links, 16 paths, and one flow traversing each path. The mean intra-domain propagation delay was set to 50msec. The mean and variance of each inter-domain delay vector $X_F$ were set to 10%, respectively, of the mean and variance of the corresponding intra-domain delay vector $D_F$.

Figs. 5.2 and 5.3 depict our results in scenario [I]. In the upper two figures, we plot the covariances and correlation coefficients of 8 flow pairs that share the longest path inside the tree topology, i.e., they have 4 common links and only the last link before their exit is different for each flow. These flow pairs are the hardest to obfuscate, because their initial delay vectors are the most highly correlated. In the third figure, we plot the inter-domain delay variances $X_F$ and $X'_F$ for each flow. We see that *Obfuscate*1() performs better with respect to our condition (1), i.e., obfuscating internal domain topology: the correlation between the scrambled intra-domain delay vectors is always close to 0. This is not the case for *Obfuscate*2(), although it still reduces correlation significantly—the correlation coefficient of the scrambled intra-domain delay vectors is below 0.2 in 50% of the cases. On the other hand, *Obfuscate*2()

performs better with respect to our condition (2), i.e., preserving the variance of inter-domain links: the variance of each scrambled inter-domain delay vector $X'_F$ follows closely that of the original delay vector $X_F$. This is not the case for *Obfuscate*1(), which increases the estimated inter-domain link variance by 3-4 times.

Figs. 5.4 and 5.5 depict our results in scenario [II] with respect to a particular flow pair $[F1, F2]$. Flows $F1$ and $F2$ share one of the longest common intra-domain paths and exit the tree topology from two nodes that are siblings. In the upper two figures, we plot the covariance and correlation coefficient of pair $[F1, F2]$ as a function of the intra- and inter-domain correlation coefficient $\rho$. In the third figure, we plot the delay variances of flow $F1$: $X_{F1}$ and $X'_{F1}$. We see that *Obfuscate*1() obfuscates well independently from inter/intra-domain correlation, whereas *Obfuscate*2() obfuscates increasingly worse when $\rho > 0.4$.

In the end, there exists a fundamental trade-off between truthfully reporting a domain's performance and obfuscating its internal topology: To obfuscate internal domain topology, we need to hide certain information about intra-domain delay. We can do this without affecting the monitor's mean delay estimates, but we need to affect *some* of the monitor's delay-variance estimates. The question is which ones. We chose to not affect the estimated delay variance of intra-domain segments (in order to preserve the incentives of our decision mechanism). But this means that we need to affect the estimated delay variance of inter-domain links. Our two obfuscation algorithms make different choices: *Obfuscate*1() obfuscates as well as it can without worrying about inter-domain delay variance. *Obfuscate*2() tries to preserve inter-domain delay variance as much as possible, but obfuscates less well. The two of them together illustrate the best that can be done while considering only the principle behind topology inference (the pairwise similarities of delay vectors). We do think that better obfuscation algorithms can be designed, but that would require considering specific topology-inference algorithms and exploiting their limitations. We leave this to future work.

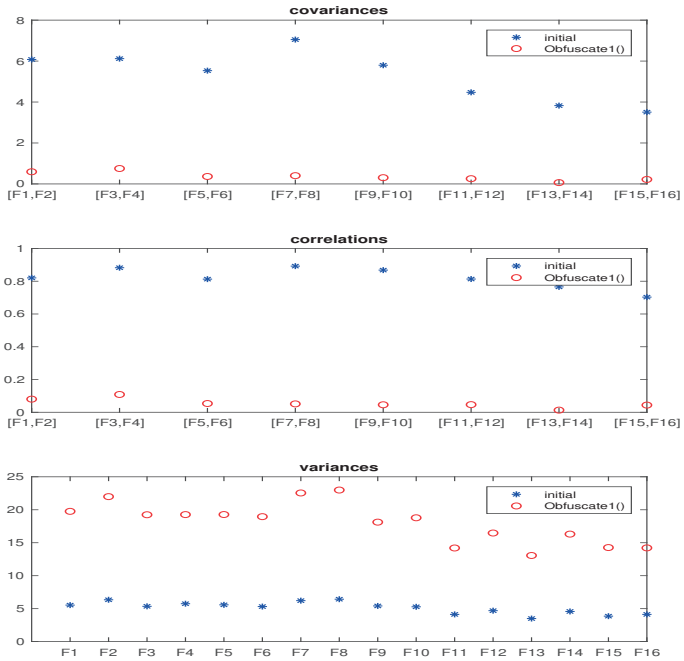



Figure 5.2 – Results of *Obfuscate*1() in scenario I.

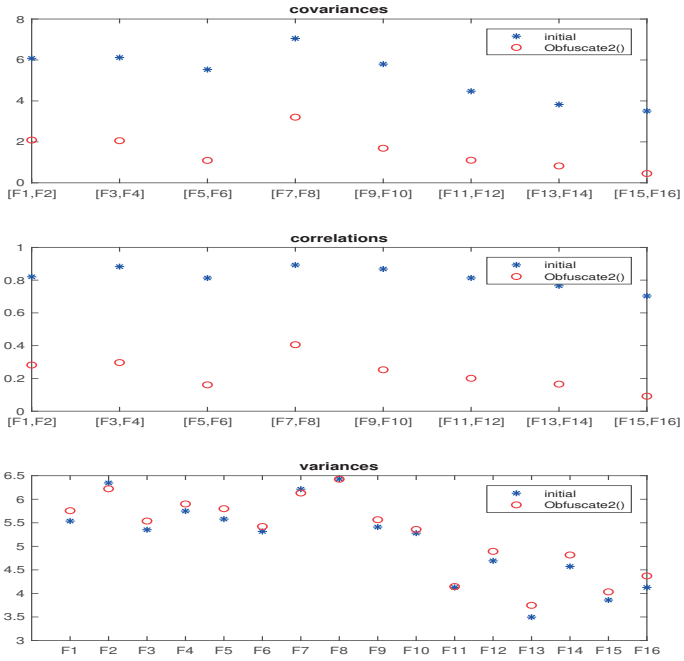



Figure 5.3 – Results of *Obfuscate*2() in scenario I.

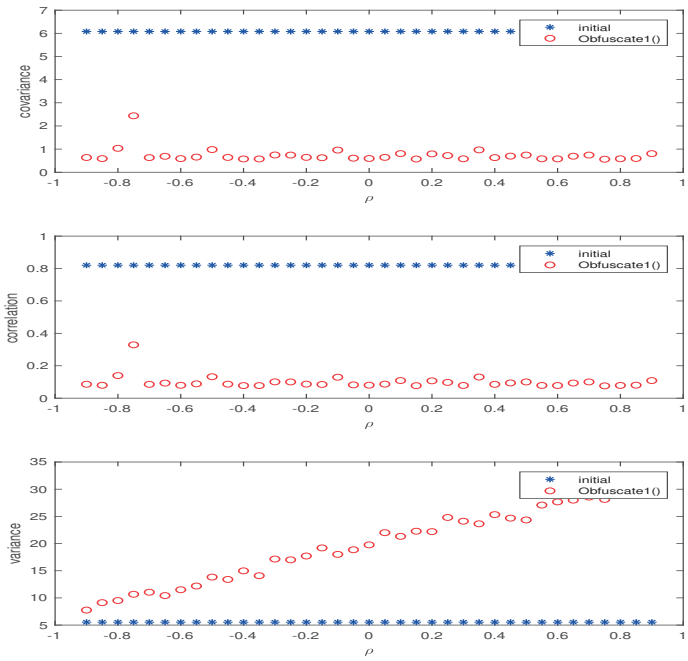



Figure 5.4 – Results of *Obfuscate*1() in scenario II.

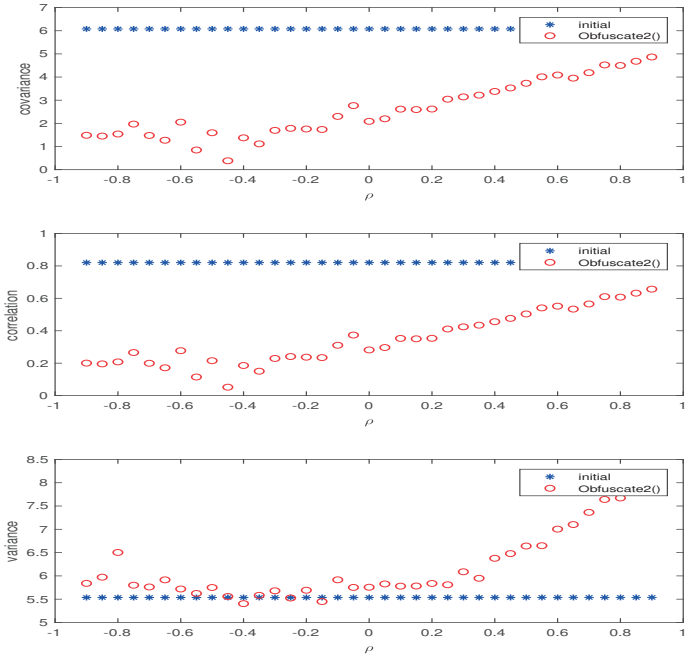



Figure 5.5 – Results of *Obfuscate*2() in scenario II.

# 6 Related Work

**Network confessional (same problem, similar approach):**

The closest work to ours is Network Confessional [14], which shares the same goal about incentive-compatible and unbiased reporting, but offers no privacy-preserving guarantees for the reporting domains. Network Confessional also involves consistent packet sampling and relies on a basic idea of delayed disclosure to avoid sampling bias: when a network node observes a packet $p$, it cannot immediately determine whether it should sample $p$ or not, because that information is disclosed by subsequent traffic; by the time disclosure happens, the node has normally forwarded $p$, hence cannot treat it according to its sampling fate.

Basic Delayed Disclosure (which is how we call the sampling algorithm proposed in [14], to differentiate it from ours) is promising, however, an analysis of the algorithm reveals flaws: First, vulnerability to subtle prioritization attacks (§4.7.4), where a misbehaving network buffers packets long enough to learn their sampling fate with a non-trivial probability, yet short enough not to introduce significant buffering delays; we show that such an attack enables a misbehaving network to claim significantly less loss and delay (as much as 41% in our experiments) than it actually introduces. Second, in many realistic scenarios, achieving good accuracy in a timely manner requires many tens of MBs of data-path memory per 10Gbps of forwarding capacity (§4.6); at this cost, we might as well use a completely different approach from sampling, e.g., maintain explicit per-flow loss and delay information on the data-path, which is not vulnerable to prioritization attacks in the first place.

Our contribution is a sampling algorithm that builds on basic delayed disclosure, but corrects these flaws: First, it is provably robust to prioritization attacks (§4.4). Second, it provably uses the minimum amount of data-path memory necessary for achieving a desired accuracy in a desired time interval (§4.5). As a result, it achieves good accuracy in a timely manner, while requiring a modest amount of resources, affordable by modern networks; for instance, in the same scenario where basic delayed disclosure requires many tens of MBs of data-path memory per 10Gbps of forwarding capacity, our algorithm requires only a couple of MB—an order of magnitude less (§4.6). To achieve these properties, we enhance delayed disclosure

with carefully regulated "quiet periods," during which no sampling may occur, and with a new disclosure process, which is continuously adapted to the observed traffic. These two techniques allow us to control the pace of disclosure, such that we emit no more receipts than necessary and make prioritization attacks ineffective: by the time a misbehaving network has learned the sampling fate of a packet, it has buffered—hence delayed—the packet for so long that it cannot benefit from prioritization any more. Our experimental evaluation confirms our analysis using real traffic traces (§4.7).

The Network Confessional paper assumed that prioritization attacks would be too expensive to implement in practice, but we disagree; in fact, without proper defense, even buffering for a few msec can yield a significant benefit to a misbehaving domain. The paper did not offer any analytical security or efficiency argument about how much such an attack can affect the monitor's estimates. This assumption has deep implications for the system's design: If a node never buffers packets, then there is no need for quiet periods; as a result, the nodes need a much smaller receipt buffer, which in turn removes the need for adaptive disclosure rates. Moreover, the bias/accuracy analysis becomes trivial (one just needs to make the receipt buffer large enough to avoid frequent overflows). In short, if we ignore prioritization attacks, the problem becomes significantly simpler. The Network Confessional paper acknowledges this issue and references a technical report that discusses it; however, even that report does not include a bias/accuracy analysis nor offer a complete, practical solution (for instance, it sketches a solution that requires 200MB of fast memory for a 10GigE link, whereas we require less than 5MB for a 40GigE link). The core of our contribution lies in our analysis of why/how our disclosure process with properly regulated quiet periods and adaptive disclosures penalizes prioritization attacks and achieves a target accuracy with modest resources.

**Secure reports for fault localization (same problem, different approaches):**

Apart from network confessional, delayed disclosure has been mentioned twice in the context of packet sampling for fault localization [60, 61], but none of these proposals offers in-band disclosure process: sampling nodes are explicitly told which packets to sample, either by end-hosts or by a central controller.

Zhang et al. were the first to mention delayed disclosure, as part of their PAAI-1 proposal [60]. However, they did not explore the idea in depth or include any bias/accuracy analysis; they only proposed that an *end-host* that sends out a packet and wants this packet sampled by the network should send an *explicit sampling request* some time after the packet: upon receiving a packet, a node computes a traffic receipt, stores it in a receipt buffer, and starts a timer; if the node receives an explicit request to sample this packet before the timer expires, it emits the traffic receipt and stops the timer; when a timer expires, the node discards the corresponding traffic receipt. Traffic sources can then locate where packets got lost or corrupted by examining traffic reports from the nodes. This is conceptually simple, but impractical to implement in our context: a node located at the Internet core may observe millions of packets per second,

and we are not aware of any network hardware with the capability to start and stop timers with that frequency. Also, involving the end-hosts would require upgrading the end-hosts, which would make deployment significantly harder.

Subsequently, Zhang et al. proposed a packet sampling technique where routers store packet fingerprints and later a central controller sends explicit sampling requests to all the sampling nodes by distributing a key that determines a sampling function [61]. The sampled packet fingerprints are then sent to the central server. Their approach was designed for a local network and would face scalability concerns if deployed in an inter-domain setting. We cannot imagine a secure, scalable system where backbone routers handle controller disclosure messages for individual packets. Also, the paper did not include any analysis that combines misbehavior, accuracy and resources.

Beyond delayed disclosure, there exists extensive related work on fault localization [12, 32, 13, 62, 15]:

Packet Obituaries [12] is a proposed transparency framework where networks produce and store a receipt for *every* single packet they observe. A modest extension of that protocol would be conceptually straightforward— no sampling nor any packet processing beyond receipt computation and management: each node could compute a receipt for every observed packet. Then the monitor could be able to perform all its decisions w.r.t. aggregate $G$, with better accuracy than our sampling algorithm, because it is able to compute the summarized statistics on the entire packet population and not only a sample of it. Bad-receipt attacks can be exposed on a per-packet basis: e.g. if node $i$ computed a receipt for the packet, and node $o$ did not; then the monitor detects an inconsistency, it notifies both involved nodes (hence, if the inconsistency is the result of one node being detectably faulty, that node is exposed to its peering node). Neutrality violations are detected with probability 1, because the monitor computes the means accurately and the means cannot be forged because of the exposure to the neighbors. Although the extended version of Packet Obituaries may be appealing, it makes sense only in low-rate environments and cannot be used in our context. It requires storing and disseminating per-packet receipts for the *entire* traffic, which yields a forbidding resource overhead in the high-rate environments that we consider (it requires an extravagant amount of fast on-path memory–approx. 400 MB of SRAM for a packet rate of 14.88 Mpps) and leaves no room to a participating domain to tune, according to network conditions the amount of resources it devotes to reporting its performance. Even if we configured the network to emit per-packet receipts only for one aggregate $G$, the resulting overhead could still be forbiddingly high; for instance, if $G$ is "all traffic from content provider $X$ that transits through ISP $Y$" or "all BitTorrent traffic that transits through ISP $Y$," having each of $Y$'s nodes emit a receipt for every single such packet would be an overkill.

Another family of proposals is based on *aggregated* performance reports for entire aggregates, as seen in the "Secure Sketch" [32], AudIt [13], and ShortMAC [62] papers. Such reports provide approximate information about the traffic aggregate, whereas a sample provides exact

information about a specific packet. In the "Secure Sketch" protocol [32], a node aggregates all the observed traffic into a space-efficient data structure and reports it to the source. In AudIt [13], ISPs export performance information – aggregated at the granularity of TCP flows – back to the source. In ShortMAC [62], the routers maintain per-flow counters to record the number of packets originated from a given source and at the end of each measurement epoch, the source retrieves the counter reports from all routers and the destination, via a secure channel, to perform fault localization. These techniques were not designed with our goals and constraints (no pre-defined aggregates or per-aggregate state) in mind. It is unclear: (a) how networks would agree on which aggregates to produce receipts for, and (b) what the resulting memory requirements would be, given that all these techniques require per-aggregate or per-flow state. Furthermore, their coarser estimation granularity enables a node to hide preferential treatment behind the aggregated performance metrics (e.g., average delay). On the contrary, random sampling does not require a global consent of the networks on which traffic aggregates to report on, it inherently covers a wide variety of them. The monitor can therefore define the aggregates as necessary, potentially with help from the source nodes, but without affecting the operation of the rest of the network. Also, if properly enhanced with retro-active mechanism (as we showed in Chapter 4), sampling can resist to preferential treatment.

Subsequent work brought formal rigor and security guarantees to the idea of networks reporting on their own performance [15], even when intermediate nodes on the path misbehave. This proposal shows that any fault can be localized to a link between two subsequent reporting nodes—even when there is no centralized monitor, and networks may tamper both with packet contents and with the receipts produced by other networks. However, it focuses on fault localization for specific "paths" (in our context, a path would correspond to a flow) and require nodes to keep per-path state. The authors recognize that their results require active cooperation (i.e. maintaining keys and agreeing on, and performing, cryptographic protocols) from all of the intermediate nodes along the path. This may be problematic in the Internet, where links operate at extremely high speeds, and intermediate nodes are owned by competing business entities with little incentive to cooperate.

In addition to the differences described above, our work is different from all prior work on fault localization [12, 13, 14, 15, 32, 60, 61, 62] in the way we approached bad-receipt attacks:

All proposals argue that the best a domain can achieve with a bad receipt attack is to shift the blame for a lost or delayed packet to an inter-domain link; since an inter-domain link is shared responsibility, the attack does not exonerate the culprit, hence there is no incentive for a domain to launch it. For example, suppose domain $y$ delivers packet $p$ to domain $x$, which drops it; both $y$ and $x$ sample $p$, but $x$ suppresses the receipt produced for $p$ at its entry point, i.e., pretends that it never received $p$; since $y$ produces a receipt for $p$ but $x$ does not, the monitor concludes that $p$ was dropped on the inter-domain link between $y$ and $x$. Whenever the monitor concludes that an inter-domain link between $x$ and $y$ introduces certain loss/delay in aggregate $G$, the monitor attributes this loss to *both* $x$ and $y$.

We partially disagree with the "shifting-responsibility' argument: while it holds when the monitor cares only about the domains' loss rate, things become more complicated when reasoning about each and every packet with a Bernoulli random variable is not possible and the distribution being estimated is not known in advance; which is what happens when the monitor cares also about the domains' delay. The problem is that the accuracy of a delay estimate that the monitor computes depends on the variance of the delay distribution, and as we proved in Chapter 3, a bad-receipt attack w.r.t. the variance is not always externalized to the inter-domain links. Moreover, collusion attacks or other sophisticated attacks that manipulate the summarizing delay estimates without manipulating each and every receipt, make this problem more complex.

Instead of adopting the shifting-responsibility argument as is, we analyzed bad-receipt attacks in a structured way that was based on the ideas of incentive-compatible mechanism design [49]: we identified the decisions about loss and delay in which the monitor is interested, and designed the decision mechanism in such a way that each domain's reporting w.r.t. the estimates is always truthful. In this way, externalization (which we term that we use for the shifting-responsibility idea) was proved to be only the one part of the solution.

**Coordinated measurements (different problem, related approaches):**

Our work shares common ground with proposals for coordinated traffic measurements inside one domain [25, 40, 43], which fulfill different objectives: In [25], Duffield *et al* propose Trajectory Sampling an implicitly coordinated method among routers for consistent sampling and direct inference of packet trajectories. The sampling decisions are based on deterministic hash functions over packets' invariant part (i.e. those bits that do not change from bit to bit). To reduce collection overhead, the authors use a second hash (instead of compression) to label the huge variety of trajectories with a few bits and parameterize the hash functions in a way that maximizes the expected fraction of unambiguous labels. In [40], Kompella *et al* explore the problem of measuring fine-grained latency and loss between any sender $A$ and a receiver $B$ (segmented measurement). They propose the Lossy Difference Aggregator (LDA), a low-overhead mechanism which aggregates timestamps of the sampled packets to overcome the linear relationship between sample size and communication overhead. They make their system resilient to packet loss using uniform consistent sampling similar to [25] and different sampling probabilities in parallel. In [43], Lee *et al* propose Consistent NetFlow that utilizes existing NetFlow architecture, which already reports the first and last timestamps per-flow, and consistent sampling to ensure that two adjacent routers record the same flows. They introduce a Multiflow estimator that approximates the intermediate delay samples from other background flows to improve the per-flow latency estimates compared to the naive estimator that only uses actual flow samples.

We build on the same idea of consistent sampling that all these proposals use and that was introduced (to the best of our knowledge) in [65]: a packet is either sampled by all or none of the nodes inside a network; sampling is performed by computing a digest over the immutable

packet content. Also, the estimation processes that we use for the monitor are quite similar to theirs. However, that work was designed for single-domain management. Its target was to aid traffic engineering by also addressing the accuracy/overhead trade-off: reduce the data of traffic reports to meet processing, storage and bandwidth constraints, on the one hand, and supply sufficiently accurate measurements for applications, on the other. None of those proposals considers untrusted sampling nodes and therefore cannot be used in adversarial environments, especially where prioritization attacks are possible.

**Other relayed work on network measurements (common vision):**

We share common vision with another interesting domain called network provenance [63, 64], where a set of network nodes maintain a distributed provenance graph that records interesting network events—potentially including packet arrivals and departures—and can be used for diagnosis and forensics. However, this approach is mostly suitable for the control plane due to its high overhead. We believe that our work could help the system scale: recording every single packet arrival and departure is expensive; instead, the provenance graph could record only arrivals and departures of a few representative packets—and our retro-active sampling algorithm would ensure that these are indeed representative.

Last, there two more measurement systems that allow network diagnosis: Handigol et al. [35] propose NetSight, an extensible platform that captures histories of every packet's journey through the intra-domain network and enables applications to concisely and flexibly retrieve packet histories of interest; but, they do not consider malicious components. Liu et al. propose UnivMon as a universal packet monitoring function [46], however, they assume a non-adversarial environment.

**Network tomography for topology inference:**

For our work on topology obfuscation, we needed to identify the common ground of all the state-of-the-art approaches in the area of network tomography for topology inference [30, 29, 56, 19, 28, 26, 27, 57]. All proposals start from the same fundamental idea: the level of covariance between the delays experienced from a series of back-to-back packets that followed two different intra-domain paths, gives information about the amount of shared logical topology between those paths. And then, they provide ways to turn this information into logical-topology reconstruction in two steps: first they construct a tree topology between any given node and the other nodes of the domain; second, the entire domain topology is reconstructed with the help of tree-merging algorithms [20]. The first step of this process is the one that the approaches differ, as some of them use an agglomerative clustering algorithm [27, 57, 16, 30, 29], and some others use maximum likelihood techniques [19]. Agglomerative clustering resolves the hierarchical clustering of a set of objects with pairwise similarity values by finding the maximum similarity element and merging the rows/columns of the similarity matrix corresponding to those two end-hosts, then finding the next maximum

element and merging those rows/columns of the covariance matrix to the new maximum element. Maximum likelihood techniques use a global maximum-penalized likelihood criterion for topology identification, which is a global optimality criterion as opposed to the suboptimal, pair-clustering strategies.

Our topology-obfuscating algorithms targeted the covariances instead of the topology-reconstruction techniques: We craft the transparency receipts such that their delay covariance is always close to 0 for all the domain's pairs of intra-domain paths, and thus leak no information about the domain's topology.

**Mechanism Design (our basis for incentive compatibility):**

We build our incentive-compatible reporting (Chap. 3) on the ideas of game-theoretic mechanism design [49]. Mechanism design is a subfield of economics that is concerned with the question of how to incentivize agents (for us agents are the domains) to truthfully report their private information, also known as their type. Given potentially non-truthful reports from the agents, a mechanism (which in our case is owned by the monitor) determines a single joint solution, and possibly additional monetary transfers to and from the agents. A mechanism is said to be incentive compatible if it is always in the agents' best interest to report their true types. Our monitor's decision function (§3.1) are similar to social choice functions: as a social choice function aggregates the preferences of the different participants toward a single joint decision, in the same way, our decision function aggregates the emitted information of the participating domains toward a single decision about their performance. However, instead of payments [34], the mechanism adds penalties to the domain's rating/utility functions, so that the domains maximize their utility by telling the truth.

More distantly related to our work from a game-theoretic perspective, is Regression Learning studies that consider strategic agents [23, 53, 59, 17]: an analyst wants to construct a real-valued function based on a training set of examples, where each example consists of an input to the function and its corresponding output, but each agent holds as private information an individual distribution over the input space and values for the points in the support of this distribution, and measures the quality of a regression function with respect to this data. That work offers more rigorous results than ours, but it is not applicable in our context. The goal of the work is to do well with respect to a statistical estimate (e.g. the average) of the individual points of view–not all of them–and it is often assumed that the precisions with which the agents perturb their reported values are known to the analyst.

# 7 Conclusions

We proposed a network transparency system, where domains produce receipts for the traffic they observe, and independent monitors collect each domain's receipts and use them to estimate the domain's loss/delay performance and detect neutrality violations with respect to various traffic aggregates. Our design was studied in an adversarial context, where both domains and monitors may misbehave for different reasons: the domains may launch bad-receipt or prioritization attacks, while the monitor may try to infer the domains' topologies using network tomography.

We proposed solutions that make the system robust to misbehavior:

First we proposed a performance and neutrality decision-making mechanism that is provably robust to bad-receipt attacks, where domains manipulate their receipts to cause the monitors make decisions that are to their advantage. Our mechanism was structured on the principles of game-theoretic mechanism design, which enables truthfulness using payments. We designed the monitors' decisions such that the each domain's performance rating includes loss/delay estimates that the domain cannot control through its receipts and are worsen when the domain launches a bad-receipt attack.

Second, we proposed a packet sampling algorithm that prevents the network node that performs the sampling from treating the sampled packets preferentially. Our algorithm builds on delayed disclosure—where the sampling function is disclosed to the sampling node with a delay—and enhances it with quiet periods, during which sampling is disallowed, and a disclosure process that adapts to each flow's packet rate. These two techniques together ensure that a misbehaving domain that tries to bias the sample to exaggerate its performance instead provably worsens its perceived delay performance. Our algorithm can be configured to provably emit enough receipts to achieve a desired accuracy within a desired time interval, while using the minimum amount of data-path memory necessary—which ends up being a few MB of data-path memory per 10Gbps of forwarding capacity.

Third, we proposed two receipt-generating algorithms that prevent tomographic approaches

from inferring a domain's topology. Our algorithms build on the idea of scrambling the reported packet delays of each intra-domain path, so that all pairwise covariances of the path delays are (close to) 0. This causes any tomographic approach to construct only direct logical links among the domain's nodes, which gives no information about the internal topology. Also, since the sample delay distribution is reported truthfully, the quality of the monitors' statistical estimation is not penalized.

## Limitations and Reality Check

We state the limitations of our system in the form of critical questions:

*Can we compute statistics for arbitrary aggregates?* No. We can compute statistics for any aggregate of significant volume, but not aggregates that consist of a few packets, e.g., a single short TCP or UDP flow. To reason about the statistical significance of a loss estimate that is based on sampling, we have to model the estimated loss as either i.i.d. or Gilbert, which is is reasonable only for large aggregates.

*Can we catch SLA violations against end-users?* Yes, but that requires deploying nodes at the user end, for instance, collocated with users' DSL or cable modems. Involving the end-users is unavoidable if we want to reason about their network experience. Even lightweight techniques that do not require any in-network infrastructure [24, 54] do require active user participation. In our examples and evaluation, we consider the scenario where the users of our system are edge networks like eyeball ISPs or enterprise/campus networks.

*Why would networks agree to produce receipts?* Any entity—e.g., government or regulatory body—that wants network innovation has an incentive to push for transparency [41]. If such an entity has the authority to enact, e.g., neutrality regulations (and apparently many do [9]), it also has the authority to require infrastructure that leads to transparency. All this aside, it is possible that, once a low-cost transparency system is shown to be feasible, some ISPs will want to participate in order to showcase their qualities, i.e., the market will drive deployment—but we have not shown this.

*How can the nodes be implemented?* Either as a bump-in-the-wire appliance or integrated in router linecards, potentially as a NetFlow or sflow extension. Our algorithm has standard hardware requirements, e.g., hashing of packet headers and timestamp computation. We describe a hardware-friendly implementation in our technical report.

*How can the monitor be implemented?* As a set of controllers, akin to a scalable SDN control plane [6]: each controller collects receipts from specific nodes/domains, and each node knows how to reach its controller and communicate with it securely. Such an implementation is a non-trivial engineering problem, but there is nothing fundamentally new in implementing a scalable logically centralized controller that collects reports from network nodes.

# A Appendix

## A.1 Basic Delayed Disclosure Algorithm

**Algorithm 5** BasicDelayedDisclosure ($p$)

| | |
|---|---|
| $\hat{p}$ | non-mutable content of packet $p$ |
| *Receipt*() | constructs a receipt |
| *DiscHash*() | hash function |
| *DiscRange* | subset of *DiscHash*'s range |
| *Hash*() | hash function |
| *Range* | subset of *Hash*'s range |

1: $rec' \leftarrow Receipt(\hat{p}, currentTime)$
2: Add $rec'$ to receipt buffer.
3: **if** $DiscHash(\hat{p}) \in DiscRange$ **then**
4: Emit receipt $rec'$.
5: **for** all receipts *rec* in receipt buffer with
6: $rec.flowID = rec'.flowID$ **do**
7: Remove *rec* from receipt buffer.
8: **if** $Hash(rec.digest, rec'.digest) \in Range$ **then**
9: Emit receipt *rec*.
10: **end if**
11: **end for**
12: **end if**

See Algorithm 5.

## A.2 Monitor Algorithm

See Algorithm 6.

**Algorithm 6** PerformanceEstimation (*flowID*)

$\mathscr{R}_i$ all receipts for *flowID* emitted by node $i$
$\mathscr{R}_o$ all receipts for *flowID* emitted by node $o$

1: $intervalStart \leftarrow 0$.
2: $numPktsIn \leftarrow 0$. $numPktsOut \leftarrow 0$.
3: $delaySamples \leftarrow \{\}$.
4: **for** each disclosure receipt $r_i{}^*$ in $\mathscr{R}_i$ **do**
5: $intervalEnd \leftarrow r_i{}^*.time$.
6: Output "Time interval:
7: *intervalStart* to *intervalEnd*".
8: Remove $r_i{}^*$ from $\mathscr{R}_i$.
9: Find disclosure receipt $r_o{}^*$ in $\mathscr{R}_o$
10: with $r_o{}^*.digest = r_i{}^*.digest$.
11: **if** (none found) **then**
12: Output "No estimate: disclosure packet lost".
13: **continue**
14: **end if**
15: Remove $r_o{}^*$ from $\mathscr{R}_o$.
16: **if** $(r_o{}^*.late)$ **then**
17: Output "No estimate: late disclosure".
18: **continue**
19: **end if**
20: **for** each receipt $r_i$ in $\mathscr{R}_i$
21: with $r_i.timestamp < r_i{}^*.timestamp$ **do**
22: Remove $r_i$ from $\mathscr{R}_i$.
23: **if** $r_i{}^*.timestamp - r_i.timestamp > \kappa + \mu$ **then**
24: ++ *numPktsIn*.
25: **end if**
26: **end for**
27: **for** each receipt $r_o$ in $\mathscr{R}_o$
28: with $r_o.timestamp < r_o{}^*.timestamp$ **do**
29: Remove $r_o$ from $\mathscr{R}_o$.
30: Find receipt $r_i$ from node $i$
31: with $r_i.digest = r_o.digest$.
32: **if** (none found AND $r_o{}^*.timestamp - r_o.timestamp > \kappa + \mu$) **then**
33: Output "Potentially inaccurate estimate:
34: reordering or jitter."
35: **else**
36: ++ *numPktsOut*.
37: Add to *delaySamples* $r_o.timestamp - r_i.timestamp$.
38: **end if**
39: **end for**
40: Output loss estimate $\frac{numPktsIn - numPktsOut}{numPktsIn}$.
41: Output delay samples *delaySamples*.
42: $intervalStart \leftarrow intervalEnd$.
43: $numPktsIn \leftarrow 0$. $numPktsOut \leftarrow 0$.
44: $delaySamples \leftarrow \{\}$.
45: **end for**

# Bibliography


[1] Optimal Prediction (with Refreshers). (????). http://www.stat.cmu.edu/~cshalizi/mreg/15/lectures/01/lecture-01.pdf

[2] 2005. FCC Fines Telecom that Blocked Vonage VoIP Calls. (March 2005). http://bit.ly/1MokIA4

[3] 2009. Comcast SLA for Wholesale Dedicated Internet. (January 2009). https://www.comcastwholesale.com/sites/default/files/service_level_agreement_for_wholesale_dedicated_internet_sla07292014.pdf

[4] 2012. AT&T Faces Formal FCC Complaint for Blocking Cellular FaceTime Use. (Sept. 2012). http://bit.ly/1JYNxpt

[5] 2014. Netflix Performance on Verizon and Comcast Has Been Dropping for Months. (Feb. 2014). http://bit.ly/1URc8zR

[6] 2015. A Distributed and Robust SDN Control Plane for Transactional Network Updates. In *Proc. of IEEE INFOCOM*.

[7] 2015. Regulation of the European Parliament and of the Council. http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32015R2120&from=en. (2015).

[8] 2016. Cogent Network Services SLA Global. (September 2016). https://cogentco.com/files/docs/network/performance/global_sla.pdf

[9] 2016. Status of Net Neutrality Around the World. (2016). https://www.thisisnetneutrality.org/

[10] 2017. Cisco IOS NetFlow. (2017). http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html

[11] 2017. Verizon Global Latency and Packet Delivery SLA. (2017). http://www.verizonenterprise.com/terms/global_latency_sla.xml

[12] Katerina Argyraki, Petros Maniatis, David Cheriton, and Scott Shenker. 2004. Providing Packet Obituaries. In *Proc. of ACM HotNets*.

[13] Katerina Argyraki, Petros Maniatis, Olga Irzak, Subramanian Ashish, and Scott Shenker. 2007. Loss and Delay Accountability for the Internet. In *Proc. of IEEE ICNP*. 194–205. https://doi.org/10.1109/ICNP.2007.4375850

[14] Katerina Argyraki, Petros Maniatis, and Ankit Singla. 2010. Verifiable Network-Performance Measurements. In *Proc. of ACM CoNEXT*.

[15] Boaz Barak, Sharon Goldberg, and David Xiao. 2008. Protocols and Lower Bounds for Failure Localization in the Internet. In *Proc. of EUROCRYPT*.

[16] R.M. Castro, M.J. Coates, and R.D. Nowak. 2004. Likelihood Based Hierarchical Clustering. *Trans. Sig. Proc.* 52, 8 (Aug. 2004), 2308–2321. https://doi.org/10.1109/TSP.2004.831124

[17] M. Chessa, J. Grossklags, and P. Loiseau. 2015. A Game-Theoretic Study on Non-monetary Incentives in Data Analytics Projects with Privacy Implications. In *2015 IEEE 28th Computer Security Foundations Symposium*. 90–104. https://doi.org/10.1109/CSF.2015.14

[18] Baek-Young Choi, Sue Moon, Zhi-Li Zhang, K. Papagiannaki, and C. Diot. 2004. Analysis of point-to-point packet delay in an operational network. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, Vol. 3. 1797–1807 vol.3. https://doi.org/10.1109/INFCOM.2004.1354590

[19] Mark Coates, Rui Castro, Robert Nowak, Manik Gadhiok, Ryan King, and Yolanda Tsang. 2002. Maximum Likelihood Network Topology Identification from Edge-based Unicast Measurements. In *Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '02)*. ACM, New York, NY, USA, 11–20. https://doi.org/10.1145/511334.511337

[20] Mark Coates, Michael Rabbat, and Robert Nowak. 2003. Merging Logical Topologies Using End-to-end Measurements. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement (IMC '03)*. ACM, New York, NY, USA, 192–203. https://doi.org/10.1145/948205.948230

[21] European Commission. 2013. On-line public consultation on "specific aspects of transparency, traffic management and switching in an Open Internet". (2013). https://ec.europa.eu/digital-single-market/en/news/answers-public-consultation-specific-aspects-transparency-traffic-management-and-switching-open

[22] David Cox and P A. W. Lewis. 1966. *The statistical analysis of series of events / by D. R. Cox and P. A. W. Lewis.* 59–60 pages.

[23] Ofer Dekel, Felix Fischer, and Ariel D. Procaccia. 2010. Incentive Compatible Regression Learning. *J. Comput. Syst. Sci.* 76, 8 (Dec. 2010), 759–777. https://doi.org/10.1016/j.jcss.2010.03.003

[24] Marcel Dischinger, 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 7th USENIX Conference on Networked Systems Design and Implementation (NSDI'10)*. USENIX Association, Berkeley, CA, USA, 27–27. http://dl.acm.org/citation.cfm?id=1855711.1855738

[25] Nick G. Duffield and Matthias Grossglauser. 2001. Trajectory Sampling for Direct Traffic Observation. *IEEE/ACM Trans. Netw.* 9, 3 (2001).

[26] N. G. Duffield, J. Horowitz, F. Lo Presti, and D. Towsley. 2002. Multicast topology inference from measured end-to-end loss. *IEEE Transactions on Information Theory* 48, 1 (Jan 2002), 26–45. https://doi.org/10.1109/18.971737

[27] N. G. Duffield and Francesco Lo Presti. 2004. Network Tomography from Measured End-to-end Delay Covariance. *IEEE/ACM Trans. Netw.* 12, 6 (Dec. 2004), 978–992. https://doi.org/10.1109/TNET.2004.838612

[28] N. G. Duffield1, J. Horowitz, F. Lo Presti, and D. Towsley. 2001. Network Delay Tomography from End-to-End Unicast Measurements. In *Evolutionary Trends of the Internet*, Sergio Palazzo (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 576–595.

[29] B. Eriksson, G. Dasarathy, P. Barford, and R. Nowak. 2010. Toward the Practical Use of Network Tomography for Internet Topology Discovery. In *2010 Proceedings IEEE INFOCOM*. 1–9. https://doi.org/10.1109/INFCOM.2010.5461970

[30] Brian Eriksson, Gautam Dasarathy, Paul Barford, and Robert Nowak. 2012. Efficient Network Tomography for Internet Topology Discovery. *IEEE/ACM Trans. Netw.* 20, 3 (June 2012), 931–943. https://doi.org/10.1109/TNET.2011.2175747

[31] C. Fraleigh, F. Tobagi, and C. Diot. 2003. Provisioning IP backbone networks to support latency sensitive traffic. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, Vol. 1. 375–385 vol.1. https://doi.org/10.1109/INFCOM.2003.1208689

[32] Sharon Goldberg, David Xiao, Eran Tromer, Boaz Barak, and Jennifer Rexford. 2008. Path-Quality Monitoring in the Presence of Adversaries. In *Proc. of ACM SIGMETRICS*.

[33] G.R. Grimmett and D.R. Stirzaker. 2001. *Probability and random processes*. Vol. 80. Oxford university press. 393–394 pages. http://scholar.google.com/scholar.bib?q=info:xzStZXK20NkJ:scholar.google.com/&output=citation&hl=en&as_sdt=0,5&ct=citation&cd=0

[34] Theodore Groves. 1973. Incentives in Teams. *Econometrica* 41, 4 (1973), 617–31. https://EconPapers.repec.org/RePEc:ecm:emetrp:v:41:y:1973:i:4:p:617-31

[35] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. 2014. I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks. In *NSDI*.

[36] Gerhard Hasslinger and Oliver Hohlfeld. 2008. The Gilbert-Elliott Model for Packet Loss in Real Time Services on the Internet. In *Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB), 2008 14th GI/ITG Conference* -. 1–15.

[37] Wassily Hoeffding and Herbert Robbins. 1994. *The Central Limit Theorem for Dependent Random Variables*. Springer New York, New York, NY, 205–213. https://doi.org/10.1007/978-1-4612-0865-5_9

[38] Manish Joshi and Theyazn Hassn Hadi. 2015. A Review of Network Traffic Analysis and Prediction Techniques. *CoRR* abs/1507.05722 (2015). http://arxiv.org/abs/1507.05722

[39] Kenneth W. Kolence. 1973. The Software Empiricist. *SIGMETRICS Perform. Eval. Rev.* 2, 2 (June 1973), 31–36. https://doi.org/10.1145/1113644.1113647

[40] Ramana Rao Kompella, Kirill Levchenko, Alex C. Snoeren, and George Varghese. 2009. Every Microsecond Counts: Tracking Fine-grain Latencies with a Lossy Difference Aggregator. In *Proc. of ACM SIGCOMM*. New York, NY, USA.

[41] P Laskowski and J Chuang. 2006. Network Monitors and Contracting Systems: Competition and Innovation. In *Proc. of ACM SIGCOMM*.

[42] Jean-Yves Le Boudec. 2010. *Performance Evaluation of Computer and Communication Systems*. EPFL Press, Lausanne, Switzerland.

[43] Myungjin Lee, Nick G. Duffield, and Ramana Rao Kompella. 2010. Two Samples are Enough: Opportunistic Flow-level Latency Estimation using NetFlow. In *Proc. of IEEE INFOCOM*.

[44] William Lehr, Erin Kenneally, and Steven Bauer. 2015. The Road to an Open Internet is Paved with Pragmatic Disclosure & Transparency Policies. (2015). http://ssrn.com/abstract=2587718

[45] W. Lehr, E. Kenneally, and S. Bauer. 2015. The Road to an Open Internet is Paved with Pragmatic Disclosure and Transparency Policies. In *Telecommunications Policy Research Conference (TPRC)*.

[46] Zaoxing Liu, Antonis Manousis, Greg Vorsanger, Vyas Sekar, and Vladimir Braverman. 2016. One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon. In *Proc. of ACM SIGCOMM*.

[47] D. Mills, U. Delaware, J. Martin, Ed.ISC, J. Burbank, and W. Kasch. 2010. Network Time Protocol (RFC 5905). (2010). https://tools.ietf.org/html/draft-ietf-ntp-ntpv4-proto-06

[48] James Munkres. 1957. ALGORITHMS FOR THE ASSIGNMENT AND TRANSPORTATION PROBLEMS. (1957).

[49] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. 2007. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA.

[50] Body of European Regulators for Electronic Communications (BEREC). 2016. BoR (16) 127: Guidelines on the Implementation by National Regulators of European Net Neutrality Rules. (2016). http://berec.europa.eu/eng/document_register/subject_matter/berec/download/0/6160-berec-guidelines-on-the-implementation-b_0.pdf

[51] K. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, and C. Diot. 2006. Measurement and Analysis of Single-hop Delay on an IP Backbone Network. *IEEE J.Sel. A. Commun.* 21, 6 (Sept. 2006), 908–921. https://doi.org/10.1109/JSAC.2003.814410

[52] Christos Pappas, Katerina Argyraki, Stefan Bechtold, and Adrian Perrig. 2015. Transparency Instead of Neutrality. In *Proc. of ACM HotNets (HotNets-XIV)*. ACM, New York, NY, USA, Article 22, 7 pages. https://doi.org/10.1145/2834050.2834082

[53] Javier Perote and Juan Perote-Peña. *Strategy-Proof Estimators for Simple Regression*. EcoMod2003 330700120. EcoMod. https://EconPapers.repec.org/RePEc:ekd:003307:330700120

[54] Mario A. Sánchez, John S. Otto, Zachary S. Bischof, and Fabián E. Bustamante. 2011. Dasu - ISP Characterization from the Edge: A BitTorrent Implementation. In *Proceedings of the ACM SIGCOMM 2011 Conference (SIGCOMM '11)*. ACM, New York, NY, USA, 454–455. https://doi.org/10.1145/2018436.2018517

[55] Joel Sommers, Paul Barford, Nick Duffield, and Amos Ron. 2007. Accurate and Efficient SLA Compliance Monitoring. In *SIGCOMM*.

[56] Han Hee Song, Lili Qiu, and Yin Zhang. 2006. NetQuest: A Flexible Framework for Large-scale Network Measurement. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '06/Performance '06)*. ACM, New York, NY, USA, 121–132. https://doi.org/10.1145/1140277.1140293

[57] Yolanda Tsang, Mehmet Yildiz, Paul Barford, and Robert Nowak. 2004. Network Radar: Tomography from Round Trip Time Measurements. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement (IMC '04)*. ACM, New York, NY, USA, 175–180. https://doi.org/10.1145/1028788.1028809

[58] William Vickrey. 1961. COUNTERSPECULATION, AUCTIONS, AND COMPETITIVE SEALED TENDERS. *Journal of Finance* 16, 1 (1961), 8–37. https://EconPapers.repec.org/RePEc:bla:jfinan:v:16:y:1961:i:1:p:8-37

[59] C. Papadimitriou Y. Cai, C. Daskalakis. 2014. Optimum statistical estimation with strategic data sources. (2014). preprint,availableasarXiv:1408.2539.

[60] Xin Zhang, Abhishek Jain, and Adrian Perrig. 2008. Packet-dropping Adversary Identification for Data Plane Security. In *Proc. of ACM CoNEXT*.

[61] Xin Zhang, Chang Lan, and Adrian Perrig. 2012. Secure and Scalable Network Fault Localization under Dynamic Traffic Patterns. In *Proc. of IEEE Security & Privacy*.

[62] Xin Zhang, Zongwei Zhou, Hsu-Chun Hsiao, Adrian Perrig, and Patrick Tague. 2012. ShortMAC: Efficient Data Plane Fault Localization. In *Proc. of NDSS*.

[63] Wenchao Zhou, Qiong Fei, Arjun Narayan, Andreas Haeberlen, Boon Thau Loo, and Micah Sherr. 2011. Secure Network Provenance. In *Proc. of ACM Symposium on Operating Systems Principles (SOSP)*.

[64] Wenchao Zhou, Suyog Mapara, Yiqing Ren, Yang Li, Andreas Haeberlen, Zachary Ives, Boon Thau Loo, and Micah Sherr. 2013. Distributed Time-Aware Provenance. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*.

[65] Tanja Zseby, S Zander, and G Carle. 2001. Evaluation of Building Blocks for Passive One-Way-Delay Measurements. (2001).

# Pavlos Nikolopoulos – Résumé


| | | | |
|---|---|---|---|
| **Address** | Rue du Centre 6A, 1023 Crissier | **Mobile Phone** | +41 78 667 41 91 |
| **Date of Birth** | $14^{th}$ April 1983 | **Email** | pavlos.nikolo@gmail.com |
| **Nationality** | Greek (Permit C) | | |


## Education

**2012-now** PhD in Computer Science/ Network Systems - École Polytechnique Fédérale de Lausanne

*Thesis*: "Traffic receipts for Network Transparency".
A network-layer system for verifiable and robust SLA/Neutrality monitoring.
Areas: Network systems, security, statistics.

**2008-2012 (part time)** MSc in Radioelectrology and Electronics - National Kapodistrian University of Athens (UoA)
Co-organized by Departments of Informatics&Telecommunications and Physics (120 ECTS).

*GPA:* 8.07/10, *GPA in major–communications:* 8.78/10.
*MSc thesis:* "On socio-aware routing for wireless opportunistic networks".
Grade: 10/10, published in school's book as one of the 4 best 2012 MSc theses.

**2001-2005** BSc in Telecommunications and Electronics - Hellenic Air Force Academy (HAFA)
Department that requires one of the highest entrance grades out of all Greek universities.

*GPA:* 92.44/100, class valedictorian and ranked top in all years.
*Dissertation:* "Direction-of-arrival estimation algorithms for smart antennas".
Grade: 100/100, highest class grade.

**2001** Higher Secondary education - General Lyceum (Direction Science), Glyfada, Athens, Greece
*GPA:* 19.6/20, Class Valedictorian.

## Work Experience

**Sep 2010 - Sep 2012** Hellenic Air Force - Telecommunications and Electronics Means Plant, Athens, Greece
*R&D Engineer and Support Team Leader*

– Programmer of Electronic Countermeasure (ECM) systems for F-16 and F-4E fighter aircrafts, developer of jamming and flare-dispensing time sequences.
– Greek representative engineer in the NATO electronic-warfare trial "EMBOW XIII" on the aircraft protection methods against infrared missile threats, Biscarrosse, France. [Sep-Oct 2011] – Team leader of the F-16 and F-4E ECM third-level support teams (6 members).

**Areas:** Signal processing, Radar theory, Communications, Electronics, Databases (minor).
**Major achievement:** Upgraded the programming system of the countermeasures dispenser AN/ALE-47 of naval helicopters S70B and enabled it to be programmed with routines used for F-16 fighter aircrafts. [2012 Air Force Project, market value = 2M€].

**Jan 2006 - Sep 2010** Hellenic Air Force - 2nd Area Control Center, Parnis, Athens, Greece
*Chief Engineer of support team and later Team Leader—also a parallel R&D role*

– Chief engineer of a small second-level support team (6 members) for the Radar and the Communication/Information Systems (CIS). [2006-2008]
– Leader of the entire technical second-level support team (23 members) for all systems (radar, CIS, communications) of the largest NATO air-control campus in Greece. [2008-2010]

**Areas:** Electronics, Networks (military and commercial standards).
**Major achievement:** Member of a small R&D group (8 members) that developed "Radios over IP", a network that enabled remote access and control (data&voice) of all radio stations in Greece, using Cisco routers. [2007-2011 Air Force Project, market value = 10M€]

## Teaching Experience

**Jan 2013-Jan 2018** École polytechnique fédérale de Lausanne, Lausanne, Switzerland
*Teacher Assistant*

Assisted in teaching the following courses: Computer Networks, Signal Processing for Communications, Probabilities and Statistics, Programming languages C and C++.

**Sep 2009, Sep 2010** Hellenic Air Force - 128 Telecommunincation & Electronics Training Group, Athens, Greece
*Seminar Lecturer*

Lecturer of the course "NATO standardization and tactical data links", part of a 6-month seminar program about avionics, that is hosted on an annual basis by the Hellenic Air Training Command (HATC) for the graduates of the Hellenic Air Force Academy.

## Publications

- **Conferences and Journals**
  – P. Nikolopoulos, C. Pappas, K. Argyraki, A.Perrig *"Retroactive Packet Sampling"*, to appear in ACM Sigmetrics/ Performance, 2019.
  – P. Nikolopoulos, C. Pappas, K. Argyraki, A.Perrig *"Robust statistics for network transparency"*, POMACS.

- **Workshops**
  – P. Nikolopoulos, T. Papadimitriou, P. Pantazopoulos, M. Karaliopoulos, I. Stavrakakis, *"How much off-center Are Centrality Metrics for Routing in Opportunistic Networks?"* ACM MobiCom 2011 - Workshop on Challenged Networks (CHANTS'11), September 23, 2011, Las Vegas, USA.

## Skills

- **System analysis and design**
  – Formal analysis and Statistical performance evaluation
  – Secure network architectures
- **Programming languages**
  – [primary:] MATLAB
  – [secondary:] C, C++, Python (with TensorFlow)
- **Languages**
  – English (excellent)
  – French (excellent for every-day communication, very good for technical affairs)
  – Greek (mother tongue)
- **Team management experience (as a 1st and 2nd Lieutenant)**
  – 4 years as Team Leader
  – 2 years as Chief Engineer

## Other Interests

- Sports: swimming (former athlete), skiing, cycling
- Traditional folk dancing: participation in international festivals (Hungary 2001, Czech Rep. 2007)

## Referees

| | | | |
|---|---|---|---|
| **Name** | Katerina Argyraki | **Name** | Vasileios Katsampas |
| **Company** | EPFL | **Company** | NATO (previously with Hellenic Air Force) |
| **Position** | Professor | **Position** | Senior Engineer |
| **Contact** | katerina.argyraki@epfl.ch | **Contact** | katsampas.vasileios@hq.nato.int |