Framework, Models and Controlled Experiments of Network Troubleshooting

Francois Espinet[a], Diana Joumblatt[b], Dario Rossi[b,a]

[a]LIX,CNRS, Ecole Polytechnique, Université Paris Saclay, Palaiseau, France
[b]LTCI, CNRS, Institut Mines-Telecom, Telecom ParisTech, Université Paris Saclay, Paris, France

Abstract

Growing network complexity mandates automated tools and methodologies for troubleshooting. In this paper, we follow a crowd-sourcing trend and argue for the need to deploy measurement probes at the edge of the network, which can be either under the control of the users (e.g., end-user devices) or the ISP (e.g., home gateways), and that raises an interesting tradeoff.

Our first contribution consists in the definition of a framework for network troubleshooting, and its implementation as open source software named NetProbes. In data mining terms, depending on the amount of information available to the probes (e.g., ISP topology), we formalize the network troubleshooting task as either a *clustering* or a *classification* problem. In networking terms, these algorithms allow respectively end-users to assess the severity of the network performance degradation, and ISPs to precisely identify the faulty link. We solve both problems with an algorithm that achieves perfect classification under the assumption of a strategic selection of probes (e.g., assisted by an ISP), and assess its performance degradation under a naive random selection. Our algorithm is *generic*, as it is agnostic to the network performance metrics; *scalable*, as it requires firing only few measurement events and simple processing; *flexible*, as clustering and classification stages are pipelined, so that the execution naturally adapts to the information available at the vantage point where the probe is deployed; and *reliable*, as it produces results that match the expectations of simple analytical models.

Our second contribution consists in a careful evaluation of the framework. Previous work on network troubleshooting has so far tackled the problem with either more theoretical or more practical approaches: inherently, evaluation methodologies lack either realism or control. In this paper we counter this problem by conducting controlled experiments with a rigorous and reproducible methodology that contrasts expectations yielded by analytical models to the experimental results gathered running our NetProbes software in the Mininet emulator. As integral part of our methodology, we perform a thorough calibration of the measurement tools employed by NetProbes to measure two example metrics of interest, namely delay and bandwidth: we show this step to be crucial, as otherwise significant biases in the measurements techniques could lead to wrong assessment of algorithmic performance. Albeit our NetProbes software is far from being a carrier-grade solution for network troubleshooting (since it does not consider neither multiple contemporary measurements, nor multiple failures, and given that we experiment with a limited number of metrics), our controlled study allows to gather several interesting observation that help designing such an automated troubleshooting system.

Keywords: Troubleshooting, Emulation, Modeling, Experiments, Root-cause analysis

1. Introduction

Nowadays, broadband Internet access is vital. Many people rely on online applications in their homes to watch TV, make VoIP calls, and interact with each other through social media and emails. Many businesses similarly offer their services over the Internet, on which the very same health of its business thus depends. Unfortunately, dynamic network conditions such as device failures and congested links can affect the network performance and cause disruptions (e.g., frozen video, poor VoIP quality, lost customers and revenue).

Currently, troubleshooting performance disruptions is complex and ad hoc due to the presence of different applications, network protocols, and administrative domains. Collection of this information in a central place is already a daunting task in reason of the volume of logs: this generally leads to terse description such as flow records, that are furthermore aggressively sampled to limit the explosion of measurement data. This tendency negatively impacts the ability to perform troubleshooting, e.g., as seeking correlation from coarse features, with some records missing[1] due to sampling is a far from ideal situation.

Yet, network troubleshooting is also complex due to the limited reach ISP have outside their network. Typically, trou-

Email addresses: `francois.espinet@polytechnique.edu` (Francois Espinet), `diana.joumblatt@telecom-paristech.fr` (Diana Joumblatt), `dario.rossi@{telecom-paristech.fr,polytechnique.edu}` (Dario Rossi)

[1]While sampling preserves information pertaining to the *same flow*, however in case of fault or anomaly, dependencies between protocols necessitates that *all flows* for the host experiencing performance disruption are observed, which per-flow sampling cannot guarantee.

bleshooting starts with a user call to the ISP help desk: however, the intervention of the ISP technician is useless if the root cause lies outside of the ISP network. The fault may be located in the Cloud offering the service, in some Autonomous System (AS) along the path, or even within the home network of its very same user. Concerning this last point, we argue that a cooperation of ISPs and user-applications can be beneficial for troubleshooting purposes. From the ISP viewpoint, it would be of course valuable to extend its reach beyond the home-gateway, i.e., by instrumenting experiments directly from end-user devices. Indeed, through home-gateway, ISPs only have a limited view of user home-network, which can be a primary source of troubles (e.g., in the case of gateways that are not directly managed by the ISPs, interference with neighbouring access points, congestion in the home network, or end-user device issues). From the complementary user-viewpoint, ISPs can provide extremely valuable information: indeed, while (tech savvy) users can leverage a number of troubleshooting tools [1, 2, 3, 4] which automate a number of useful measurements, however these tools are generally ISP-network agnostic and cannot embed tomography techniques [5, 6] to identify the root causes (e.g., faulty links) of network disruption.

In this paper, we propose a practical methodology to automate the identification of network performance disruption based on end-to-end measurements. Our proposal is to *decouple* measurement from inference: we let end-device the burden of controlling experiments and collecting results, but do not mandates the troubleshooting process to run on the same end-devices. This distributed approach implicitly alleviates control bottlenecks, while still allowing the ISPs to assist the measurement process (e.g., by biasing the set of measurement, or their spatial reach). At the same time, decoupling measurement collection from measurement analysis requires to cope with a flexible workflow, where the amount of knowledge at disposal of the inference algorithm may vary. Indeed, devices participating in the troubleshooting task can be either under the control of the ISP or the end-user: in the former case, knowledge of the ISP topology can be leveraged by IETF ALTO servers to realize a *strategic* nodes selection, whereas in the latter case absence of topology information means that only simpler *randomized* selection can be implemented. This difference further exacerbates in the troubleshooting task, that we formalize as a pipelined algorithm: a first *clustering* stage available to all users allows to just assess the severity of the fault, whereas a second *classification* stage further allows ISPs to identify the faulty link.

While our work is not the first to address the problem of network troubleshooting, we note that related effort can be roughly split in two main branches. On the one hand, there is a number of previous work with a mostly *practical focus* [1, 7, 2, 3, 4], which are very valuable in terms of domain knowledge and engineering effort, but lack otherwise theoretical foundations and rigorous verification. On the other hand, prior analytical work exists that is cast on solid *theoretic basis* [5, 6], whose validation is however either simplistic (e.g., simulations) or lacks ground truth (e.g., PlanetLab).

In this work, of which a preliminary version appeared at [8], we take the best of both worlds, and make the following main contributions:

- we propose a practical and general framework for network troubleshooting with an open source implementation;
- we provide simple yet instructive models of the expected fault detection probability, that we contrast with experimental results;
- we use an experimental approach where we emulate controlled network conditions with Mininet [9] and perform a thorough calibration of the emulation setup – an often neglected albeit mandatory task;
- sharing the same reproducibility spirit of Mininet, we further make all our source code available for the scientific community at [10, 11].

Aside our *models*, *algorithm* and its *open-source* software implementation –which are interesting per se– we believe that the rigor of our *experimental evaluation* is another crucial contribution of this paper, which is structured as follows. We first describe the problem we address from a networking viewpoint, and introduce two use-cases where our approach can be applied, that mainly differ in the amount of topological knowledge that the root-cause algorithm has at its disposal (Sec. 2). We then introduce a more formal system model, and phrase more rigorously the above problems, proposing two simple yet insightful analytical models of the expected troubleshooting performance under *randomized* selection, as well at its degradation with respect to a *strategic* selection (Sec. 3). We next describe our generic troubleshooting algorithm that, depending on the amount of available information, can be formalized as a *clustering* vs. *classification* problem (Sec. 4). The scenario of our controlled experimental evaluation is discussed next, carefully calibrating tools for delay and bandwidth emulation and measurement (Sec. 5). We report results of a thorough Mininet emulation campaign, investigating several important system and scenario parameters, contrasting experimental vs. modelings results (Sec. 6), and discuss practical aspects that a full-blown troubleshooting framework needs to take into account (Sec. 7). Finally, we cast our work in the context of related effort (Sec. 8) and summarize our main lessons learned and contributions (Sec. 9).

2. Network scenario

We describe the network scenarii we address in this work with the help of Fig.1. Specifically, the picture describes two scenarii, where we assume troubleshooting software to be deployed in the user home, and more precisely in the user terminals. In the leftmost case, users are connected as an overlay over the Internet, of which they hardly have any topological information (due to its large scale and temporal variability). In the rightmost case, users belong to the same ISP, whose topological information is smaller in scale, varying at a slower pace, and possibly available (at least to some extent, as described in

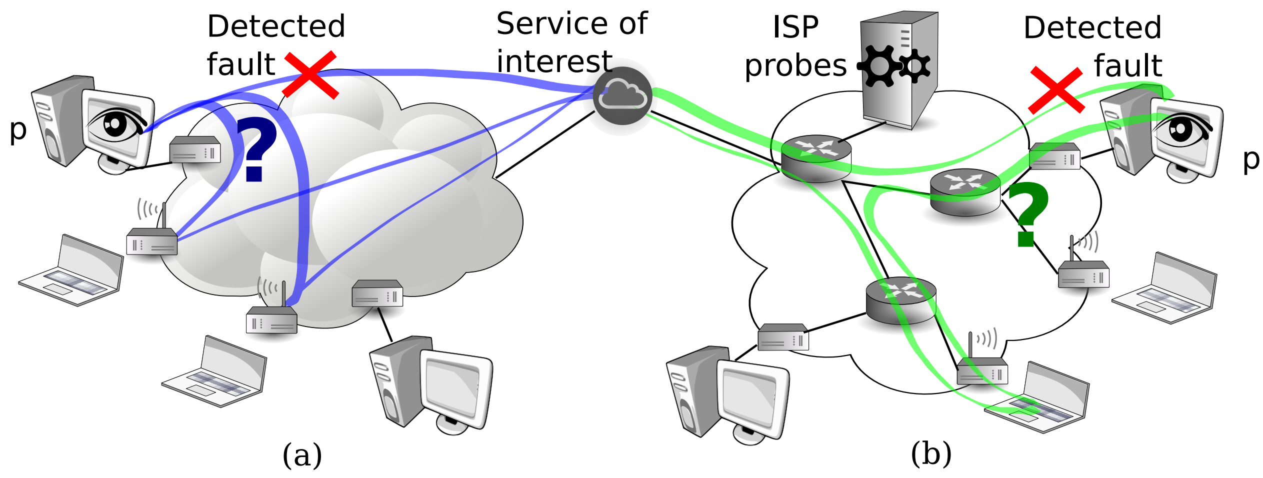


Figure 1: Network scenarii: (a) Internet case with limited information (e.g., where only randomized target selection policies are feasible) vs. (b) ISP-case with additional information (e.g., topology, for which a strategic target selection is possible) and additional means (e.g., controlled probes; ALTO server; etc.)

what follows). Additionally, in the latter case, troubleshooting software can be deployed in the home gateway, or in special network locations managed by the ISPs, complementing tools deployed in the user terminals.

2.1. *The status quo*

Currently, troubleshooting is not only complex due to the huge diversity of network apparatus, but even considering a single ISP network, by the existence of multiple "ownership" domains. While from the end-user perspective, the network is opaque and users have no means to identify where the problem happens, the same holds true for ISPs when the troubles happen inside the user's home network (e.g., in the case of gateways that are not directly managed by the ISPs, interference with neighbouring access points, congestion in the home network, or end-user device issues). This is even more true considering the increasing heterogeneity of connected equipments in user households: routers, computers, tablets, phones, smart-TVs, distributed storage, and so on, of which any is a candidate root cause of the network disruptions experienced by end-users. Objective diagnostics is hard to achieve because troubleshooting is often done remotely (most likely over the phone or the Internet). The remote operator has very little knowledge of the environment in which the problem occurs, and very little control when the problem is located in the user network. Furthermore, users have generally scarce troubleshooting knowledge while tech-savvy users have limited control over the network.

To break this status quo, we advocate as in [4] to crowdsource the troubleshooting process by using software probes at the edge of the network. With respect to [4], we further envision these probes to be either managed by either the end-users, or by the ISPs. User-managed probes run only on end-user devices and lack topology information. In contrast, ISP-managed probes can reside in home gateways, in special locations inside the ISP network, and can also be available as "apps" on user devices (e.g., smartphones and laptops). As we shall see, end-user and ISP collaboration can greatly enhance the troubleshooting process.

2.2. *Internet vs ISP cases*

Specifically as depicted in Fig.1, we envision an observer (denote with an "eye" in the picture) experiencing problems to a particular service of interest, to trigger a set of measurement to the same service from a multitude of other distributed vantage points. Results of these experiments are returned to the observer: depending on the level of cooperation with the network, the observer is either able to merely assess the severity/generality of the fault –similarly to a "Do You See What I See" (DYSWIS) [1] framework of an Internet scenario–, or possibly more precisely pinpoint the location of the fault (similarly to a tomography study [12, 13] in an ISP scenario). Notice that even in case where ISP-managed probes do precisely pinpoint the root-cause link using topology knowledge, the information passed back to users need not to be extremely detailed: the user is most likely interested in knowing whether the problem resides in his access network, shared within his ISP or general in the broader Internet. In turn, in case the issue is shared by many users in the ISP, this knowledge hopefully assists the ISP troubleshooting process, or at least assigning priorities to events.

As for the Internet scenario, note that Web-based services with an aim similar to ours do exist[2]: however, fault detection comes from analysis of third-party sources[3] as opposite to inference coming from measurements of real user data. Since analysis of third-party sources (e.g., twitter hashtags) requires active user contributions, is prone to interpretation errors, introduces a delay and makes automated troubleshooting very hard, a service such as the one we propose here is thus still valuable for the end-user. Moreover, web-based services report historical information, whereas the proposed service offers automated on-demand measurement, which are extremely

[2]See for instance `https://downdetector.com/`

[3]As for instance Twitter, see `https://downdetector.com/about-us/`

useful in the case of any new and unreported fault. The value of this kind of crowd-sourcing approaches is clearly shown by work such e.g., DYSWIS [1] and Ono [4], where users can intentionally install troubleshooting applications running in the background [1, 4], or trigger them on demand [2, 3]. Especially, the popularity enjoyed by Ono [4] and Netalyzr [3] testifies an interest from the end-users, confirming it to be a viable model. Given a good footprint of the monitoring application over the Internet, and assuming that a shared AS-level Internet map can be leveraged [14], the approach we outline in this paper could be applied in this scenario as well. Even in the worst case where Internet topology information would be absent, informations concerning the extent of the problem would still be of value – e.g., providing simple and immediate answers to questions such as "is YouTube down for everybody (again[4]) or just for my ISP?" However, given the scale of the problem and the complexity of its evaluation, we prefer to focus on an Intra-ISP case in this paper.

As for the ISP-scenario, note that even more valuable insights can be gathered via an exchange of information. On the one hand, ISPs are interested in extending the reach of the measurement infrastructure beyond the home-gateway, since limitedly performing experiments from the home-gateway could possibly miss congestion in the user home wireless network, which is a frequent cause of performance degradation [15]. Similarly, the ability to issue measurement a plethora of protocols including L4 TCP connections, L7 handshakes, running DNS queries, and measuring in-path QoS properties such as bandwidth, throughput and delay *from the end-user terminal* (as opposite as to from home gateways) could enrich the level of details available to the diagnosis (and provide a picture closer to the quality of experience perceived by user of that terminal). On the other hand, end-users can greatly benefit from ISP assistance to e.g., discriminate ambiguous cases (as we will later), issuing measurement from specific locations deeper in the operator network, make informed decisions based on topology information and possibly leverage complementary information from the home-gateway (when available and reliable [16]) or the network. Of course, confidentiality and privacy matters may arise, since ISPs may need to share information with "apps" running on end-user devices on the one hand, and given that ISPs may automatically trigger experiments from the end-user device on the other hand. In this paper, we consider the first problem and offer (i) an ALTO interface to avoid sharing of sensitive information in the measurement process as well as (ii) a decoupling of the measurement and inference processes to facilitate analysis of sensitive ISP information. Albeit we do not explicitly address the problem from the side of the end-user, we argue that it is necessary to provide users with sufficient control on (and information about) the experiments running on their devices (e.g., building on [17, 2, 4, 3]).

We address both use-cases with the same two-steps algorithm: a first clustering step (in both scenarios) separates measurement probes into two sets (i.e., un/affected sets), whereas a second classification step (available only in the ISP-scenario) requires an additional, trivial mapping to pinpoint the root cause link. Our viewpoint is thus to build a single probe capable of a rich set of tests yielding data for the above algorithm. This probe runs a diagnosis algorithm that can either rely exclusively on measurements to other distributed probes, or can leverage additional information (e.g. topology and forwarding) to refine the inference. We argue that such an approach is flexible and can cope with both use-cases in an incremental fashion, and implement this approach in our open source NetProbes [10] software (whose thorough description is out of the scope of this paper, and of which we report a terse description in Sec.4.2).

[4]`http://research.dyn.com/2008/02/pakistan-hijacks-youtube-1/`

2.3. Topology information

Tomography work usually assume that the topology can be reliably measured based on end-to-end measurements [18], with traceroute or inference techniques. Albeit simplistic, this is done to decouple the topology inference process from the identification of the faulty link on the inferred topology. This assumption is, to some extent, reasonable as topology inference happens over a longer timescale with respect to the one of the fault, where it can be assumed that topology is not evolving – with the exception of the fault itself. Along the same lines, tomography work generally focuses on trees: the justification in this case is that, while most network topologies are not trees, it is however possible to use multiple trees and merge them to obtain a single view [19, 5]. Finally, in spite of more complex topologies, to avoid loops the subset of data paths from a given observer to any other node in the network is usually a tree.

In this work we slightly depart from these assumptions in that we however do believe that fine-grained topology information is possibly very hard to get. This is due to the operational obscurity of ISP networks which limits the visibility to portion of the network links [20, 21], to measurement unreliability of traceroute [22] and even ping tools[23]. As such, rather than decoupling the problems and taking accurate topology inference for granted, we factor in the architecture a technique to avoid topology inference problem. We do so by proposing a viable cooperation path with ISP, that hold ground truth information about their network at all layers, but are of course not willing to share such sensitive information. Our key solution to design such a practically viable system that does not *directly* leverage topology information, involves the use of an IETF Application Layer Traffic Optimization (ALTO) server. Specifically, IETF ALTO [24] defines a protocol to let servers assist hosts in making informed decisions: ALTO was initially cast for peer-to-peer (P2P) network management, and its use has since then evolved to Content Delivery Networks (CDN). For instance, in P2P networks, ALTO was used to bias the overlay topology by in a cost-aware manner without explicitly disclosing such sensitive business/topology information. Specifically, ALTO servers were used as BitTorrent trackers that returned a set of candidate peers for a swarm based on sensitive ISP information, as opposite to in a random fashion as regular trackers would do.

We argue that the very same interface could be seamlessly leveraged in the troubleshooting context. Indeed, by yielding a list of prioritized alternatives to end-hosts, an ALTO server

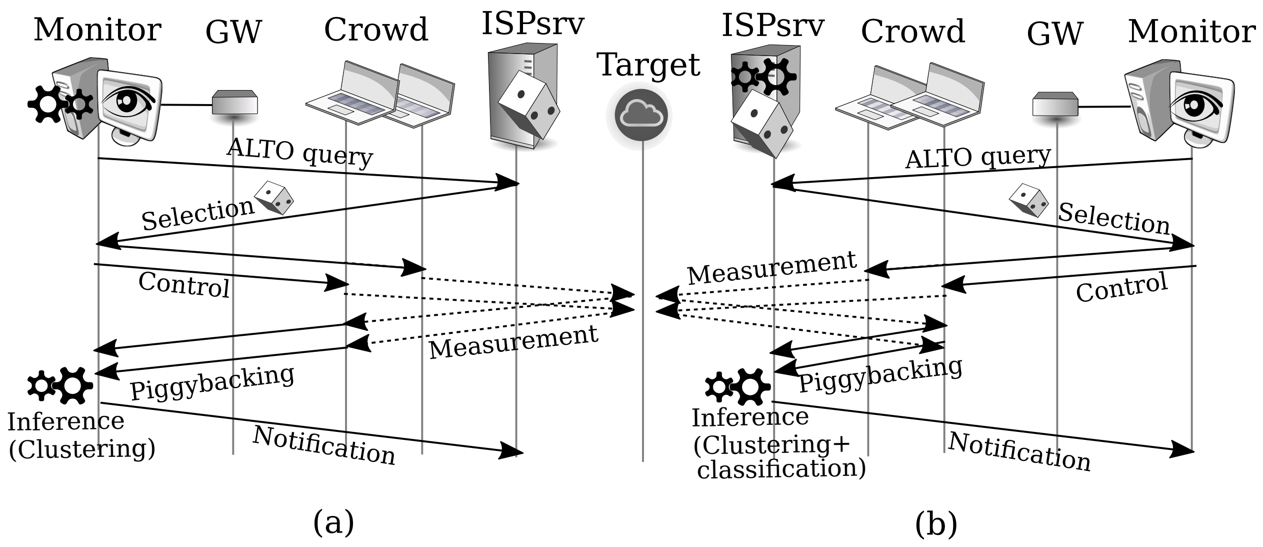


Figure 2: Synoptic of the troubleshooting and inference process in the ISP-case. Solid lines denote control traffic, dotted lines indicate data-plane measurement; results of data-plane measurement are possibly piggybacked into control messages. Two cases are considered: (a) ISP-assisted crowd selection via ALTO server, local processing with limited information vs. (b) ISP-assisted crowd selection via ALTO server, ISP-managed troubleshooting with extensive information (a subset of which is then notified to the end-user application)

.

can suggest crowd-sourcing candidates for troubleshooting purposes in a topology-aware manner, without directly disclosing topology information to the end-host. To stress the viability of an ALTO-like solution, it is worth recalling that (i) network-aware BitTorrent trackers were one of the first use of ALTO servers [25], and that (ii) tracker technology has proven significant scalability, up to million of torrents[26], in reason of this very simple interface. Hence we expect the same scalability properties in this context.

2.4. Troubleshooting and inference

An ALTO-like interface non-only allow to protect sensitive information such as topology, but further decouples the *measurement task* from the *inference process*. In turn, these measurement could be leveraged by a topology-agnostic (i.e., a end-user probe) or topology-aware (i.e., residential gateway, ISP server) troubleshooting process. In both cases, measurement where target selection have been biased by an ALTO interface could benefit from strategic sampling choices, as we shall see later. In case of ISP-processing, the end-user probe would just extend the reach of ISP troubleshooting process in the user household, accessing a wealth of additional information (e.g., SNMP, topology, etc.).

This is exemplified in Fig. 2 that consider two ISP-cases, where either (a) target probe selection is assisted via an ALTO server deployed by the ISP, but troubleshooting process is locally performed by the node with limited information vs. (b) ALTO-assisted target probe selection, and troubleshooting process is run by the ISP and has access to an extensive set of information. In both cases, a end-user monitor experiencing a problem triggers an ALTO query to obtain a set of candidates probes. By doing so, the management burden is then shifted from the ISP to the end-user application, reducing the opex cost (i.e., less powerful servers are sufficient, recall [26]). The end-user agent contacts candidates of the target set, instructing them to perform measurement toward a destination of interest (possibly, but not necessarily, the source itself) and report results back. Measurement and inference processes are decoupled, so that upon completion of the measurement, results of the experiments are piggybacked back from the probes to either (a) the original agent or (b) an ISP-dedicated server.

Upon reception of a set of measurements, the troubleshooting process can take place. Notice that in case (a) only results of clustering are available: output of local execution can be possibly notified back to the ISP to assess the severity of the damage. Conversely, in case (b) a richer set of information translate into a richer output (e.g., root cause link) at the end of the troubleshooting process. Still, the ISP can decide which subset of information can be signaled back to the user, retaining control on the amount/sensitivity of the information to be disclosed.

The flexibility of this framework allows for completely orthogonal solutions where, e.g., ISPs are willing to reduce their costs and externalize as much as possible the troubleshooting process, to solution where ISPs aims at refining their inference process and can afford investing (or leverage their existing) computing and big-data facilities to ameliorate their troubleshooting process, where furthermore the diagnosis process can leverage other ISP-sensitive sources of information.

3. System model

We formalize the problem tackled in this paper with the help of Fig. 3 and for the sake of clarity, we summarize the notation used in Tab. 1. Without loss of generality, we start by considering a regular k-ary access network tree (and depict for the sake

Table 1: Summary of the notation for the k-ary tree model

ID	source probe (1), ISP probe (0), target (> 1)
ℓ_i	link at depth i in the path from source $\to$ root
$\ell_k, .., \ell_1$	path source $\to$ root
$D^+ = log_k(N)$	depth of a k-ary tree
$\mathcal{D}_i = (k^{D^+-i}, k^{D^+-i+1}]$	set of target probes with $ID \in \mathcal{D}_i$, whose shortest path from the source probe $ID = 1$ passes through ℓ_i, but not through ℓ_{i-1}
$\text{card}(\mathcal{D}_i) = k^{D^+-i}$	size of set $\mathcal{D}_i$

M	size of target probes set
N	size of overall probes set
$\alpha = M/N$	probe budget
$f \in [1, D^+]$	fault depth
$E[p_X(f, \alpha)]$	probability of root cause identification for a failure at depth f (involving link ℓ_f)
$p_X^-(f, \alpha)$	probability to randomly sample a target in $\mathcal{D}_f$ (model X)

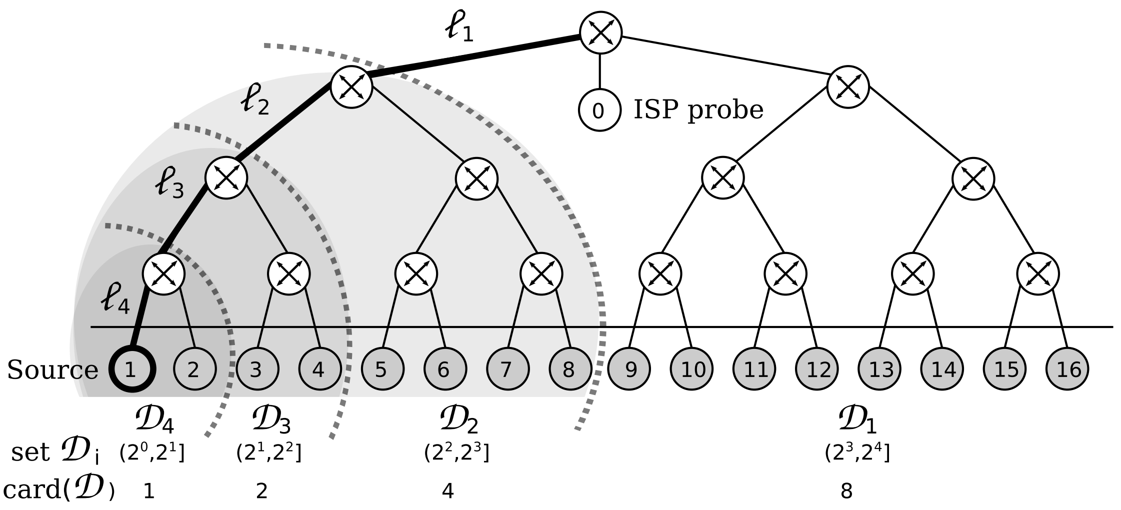


Figure 3: Synoptic of the network scenario and model notation

of simplicity a binary tree in the figure) and develop simple probabilistic models of the troubleshooting detection process.

In the user-only case, the troubleshooting software only runs in the leaf nodes of the tree and has access to very few topological information: hence, it is important to assess the performance of a *random* target selection policy. However, the ISP can strategically place probes inside the network (e.g., probe 0 in the picture attached to the root) or dispose of topological information that allows probe to perform *strategic* target selection: it is thus important to assess the gain brought by the additional knowledge and degree of freedom that ISPs have in probe placement.

Our algorithm runs continuously in the background to gather a baseline of network performance (i.e., in absence of faults), and troubleshooting is triggered by the user (e.g., upon experiencing a degradation of network performance) or automatically by a change point detection triggered by the deviation of some relevant metric from the baseline (outside the scope of this work). In the reminder of this section, we introduce a simplified view of the previous scenarii, that allow us to not only formalize the problem, but also to provide different models of the expected detection capabilities of a end-user vs ISP-managed troubleshooting system.

3.1. Notation and general framework

For the sake of simplicity, we consider the case where a single fault happens in the system. We here consider "fault" in a loose sense, i.e., more generally any detectable event in any metric of interest.

Without loss of generality, we consider that this fault is located at a depth f on a k-ary tree, and we focus on the perspective of a troubleshooting process triggered by a source probe[5] having ID= 1. We can safely assume that the root cause is located somewhere in the path from the user device or gateway towards the Internet, which in the example illustrated in Fig. 3 comprises the sequence of links $(\ell_4, \ell_3, \ell_2, \ell_1)$. In order to identify which among $\ell_4, .., \ell_1$ is the root cause of the fault, the source node $ID = 1$ sends probing traffic to a number M

[5]Notice that the following model can be developed irrespectively of which source probe triggers the measurement: i.e., the same process can independently be performed from any source. However, focusing on probe $ID = 1$ as a source allows to simplify the notation, on which case we therefore limitedly focus in what follows.

of targets probes, among the overall available N nodes in the network. Let us denote, for convenience, by $D^+ = log_k(N)$ the maximum depth (i.e., height) of a k-ary tree and by $\mathcal{D}_i$ the set of probes $\mathcal{D}_i = \left(k^{D^+-i}, k^{D^+-i+1}\right]$. By definition, the set $\mathcal{D}_i$ includes targets whose shortest path from the source probe $ID = 1$ passes through ℓ_i, but does not pass through ℓ_{i-1}. In the access tree, whenever a link ℓ_f (located at depth f in the tree) is faulty, all probes whose shortest path from the diagnostic probe (probe 1 in our example) passes through ℓ_f will also experience the problem, unlike probes that are reachable through ℓ_{f+1}: it follows that the troubleshooting algorithm requires target probes from both sets $\mathcal{D}_f$ and $\mathcal{D}_{f+1}$ to infer with certainty that the fault is located at ℓ_f.

In this section, we provide simple probabilistic models for the fault detection probability under *strategic* vs a *random* target probes selection. While the case of random selection is more involved and is developed next, in the case of strategic selection it is easy to convince that in order to identify faults at any depths, by construction, the source must select exactly one target from each set. Particularizing this to a k-ary tree, the minimum number of probes that allows to identify the faulty link irrespectively of the depth f of the fault is $M = O(log_k(N))$ – i.e., one probe in each of the $\{\mathcal{D}_i\}_{i=1}^{log_k(N)}$ strata suffices to accurately pinpoint the root cause.

Such a strategic probe selection requires either topology knowledge or the assistance of a cooperating server managed by the ISP (e.g., an IETF ALTO server as previously illustrated). However, this strategy is not feasible with user-managed probes, in which probe selection is either uniformly random or based on publicly available information such as IP addresses. It is thus important to assess the detection probability of a naïve random selection as well, to get a performance lower bound – or equivalently, the added value of an ALTO-like server.

One further point is worth elucidating. While neither the algorithm (nor the model) we present in this paper are limited to a regular tree, the use of regular trees simplify the tractation and additionally provides a reasonable test case. Indeed, the model can be trivially extended to unbalanced trees, except that the size of the strata cannot be simply expressed as a function of the depth in the tree. Similarly, access topologies are often considered to have a (logical) tree shape [27] due to loop-free forwarding (so that physically redundant links are used for protection and fault management, and only seldom for equal cost multi-path routing). Finally, while the complete ISP topology can be ultimately more complex, this work considers the case where an observer triggers a measurement: by rooting the topology on the data path from this observer to any other user in the topology, even faraway users (e.g., in other access trees beyond different DSLAMs) can be seen as belonging to different branches of the tree (that have at least the egress gateway in common). For the above reasons, unless otherwise stated we limit the tractation to regular trees in what follows.

3.2. Lower bound of the detection probability

We now focus on the case where the source probe performs a *random* selection of its target probes set. With respect to the strategic selection, the source cannot longer leverage an oracle to select exactly at least on probe per each of the $\mathcal{D}_i$ strata – which is expected to have a significant impact on troubleshooting performance. Notice indeed that, by definition, the sets $\mathcal{D}_i$ have an exponentially decreasing size: i.e., in the example Fig. 3 the set $\mathcal{D}_1$ comprises 8 nodes (i.e., half of the leaves of the whole binary tree), $\mathcal{D}_2$ has size 4 (i.e., the half of $\mathcal{D}_1$), $\mathcal{D}_3$ has size 2 and $\mathcal{D}_4$ has size 1. Clearly, without any topological knowledge, sampling targets in $\mathcal{D}_4$ is 8 times less probable than for targets in $\mathcal{D}_1$.

Consequently, the deeper is the fault location, the smaller is the number of probes available to identify the faulty link. As the size of $\mathcal{D}_f$ exponentially decreases as f increases (for the tree card$(\mathcal{D}_f) = k^{D^+-f}$), we expect the random selection strategy to easily locate faults at small depths (close to the root) and fail at large depths (close to the leaves) where a stratified selection is necessary to sample probes in the smaller set $\mathcal{D}_f$. Missing a strategic selection, a tradeoff arises on increasing the target set size vs reducing the root cause identification probability.

More formally, let us denote by $p^-(f, \alpha)$ the probability that a random selection includes a probe that is useful to locate a fault at depth $f \in [1, D^+]$, with a probe budget $\alpha = M/N$. In our previous work [8], we proposed to compute a lower bound of the detection probability by leveraging on the just described intuition: denoting this model as TMA[6] the probability that none of the M vantage points falls into $\mathcal{D}_f$ decreases exponentially fast with the size of $\mathcal{D}_f$, i.e., $P_{TMA}(X = 0) = (1 - \alpha)^{card(\mathcal{D}_f)}$. Consequently, the probability to sample at least one probe in $\mathcal{D}_f$ is approximated by:

$$p^-_{TMA}(f, \alpha) = 1 - P_{TMA}(X = 0) = 1 - (1 - \alpha)^{k^{(D^+-f)}} \tag{1}$$

The model (1), albeit admittedly simplistic, has the advantage of a simpler formulation with respect to the expression of the exact Hypergeometric (HG) model, which estimates the probability to have k successes in n draws without replacement in a finite population of N probes with K successes. In our settings, the n draws correspond to the probe budget M, and the number successes K correspond to the number of probes in the $\mathcal{D}_f$ set, so that the probability to have 1 or more probe in $\mathcal{D}_f$ can be computed as:

$$p^-_{HG}(f, \alpha) = 1 - P_{HG}(X = 0) = 1 - \frac{\binom{N-\text{card}(\mathcal{D}_f)}{M}}{\binom{N}{M}} \tag{2}$$

While the presence of a binomial coefficient makes it hard to give an intuitive explanation to equation (2), it is possible to resort to (a chain of) known approximations to provide another, more insightful and yet still accurate formulation. Indeed, it is well known that the HG model can be approximated for large population, with a binomial one: the intuition is that when N is large, chances of selecting multiple times the same probe

[6]Acronym of Traffic Monitoring and Analysis (TMA), the original venue [8] where the model was first presented

lessens, so that explicitly accounting for replacement plays a marginal role. In turn, the binomial distribution is well approximated by a Poisson distribution, so that the exact combinatorial formulation of (2) is better expressed for our purposes as:

$$p^{-}_{Poisson}(f, \alpha) = 1 - P_{Poisson}(X = 0) = 1 - e^{-\alpha k^{(D^{+}-f)}} \tag{3}$$

Interesting insights can be gathered from (1) and (3). For faults located high in the tree ($f = 0$), it appears that $p^{-}(0, \alpha) \approx 1$. Conversely, for fault close to the leafs ($f = D^{+}$), the approximation degenerates into $p^{-}_{Poisson}(D^{+}, \alpha) = 1 - e^{-\alpha}$ or even simply $p^{-}_{TMA}(D^{+}, \alpha) = \alpha$. We thus expect p^{-} to take values in the support $\in [\alpha, 1]$, with a fast varying dynamic as a function of the fault depth f.

3.3. Average detection probability

Expressions (1), (2) and (3) all represent *lower bounds* on the detection probability with random selection. However, even when a random subset of probes *does not contain* any probe in $\mathcal{D}_f$, it is still possible to correctly guess the root cause link. Here, there will be ambiguity because multiple links are equally likely to be root cause candidates. At any depth d, ambiguity will be limited to the links located between the fault and the root of the tree (i.e., $\ell_d, .., \ell_1$): since, at depth d, ambiguity involves d links, the probability of a correct guess is $1/d$. To compute the average probability of a correct guess $\mathbb{E}[p^{guess}]$, we have to account for the relative frequency of the different ambiguity cases, which for depth d happen proportionally to $k^d / k^{log_k(N)} = k^d/N$,

$$\mathbb{E}[p^{guess}] = \sum_{d=1}^{log_k(N)} \frac{1}{d}\frac{k^d}{N} = \frac{1}{N} \sum_{d=1}^{log_k(N)} \frac{k^d}{d} \tag{4}$$

For the sake of completeness, we point out that partial sum of k^d/d can be expressed as:

$$\mathbb{E}[p^{guess}] = \frac{1}{N} k^{log_k(N)+1}\big(- \Phi(k, 1, log_k(N) + 1) - log(1 - d)\big) \tag{5}$$

where $\Phi(k, 1, log_k(N) + 1$ is the Lerch trascendent,i.e.,a generalization of the Hurwitz zeta and polylogarithm functions, typically used to compactly express sums of reciprocal powers. [28]. Irrespectively of the model adopted to compute the lower bound, we can then compute the expected discriminative power of a random selection, expressed in terms of the probability to correctly identify a fault at depth f as:

$$\mathbb{E}[p] = p^{-}(f, \alpha) + \big(1 - p^{-}(f, \alpha)\big)\mathbb{E}[p^{guess}] \tag{6}$$

where the first term accounts for the proportion of random selection that is structurally equivalent to a stratified selection (so that the root cause link can be found with probability 1), and the second term accounts for the proportion of random selection able to pinpoint the faulty link by luck (thus with probability $\mathbb{E}[p^{guess}]$).

We limitedly particularize (6) to the TMA and Poisson approximations, to gather different expectations of the detection probability. Specifically, by plugging (1) and (4) into (6), after some algebra we get:

$$\mathbb{E}[p_{TMA}] = 1 - (1 - \alpha)^{k^{(D^{+}-f)}} \Big(1 - \frac{1}{N} \sum_{d=1}^{log_k(N)} \frac{k^d}{d}\Big) \tag{7}$$

whereas by plugging (3) and (4) into (6) we similarly get:

$$\mathbb{E}[p_{Poisson}] = 1 - e^{-\alpha k^{(D^{+}-f)}} \Big(1 - \frac{1}{N} \sum_{d=1}^{log_k(N)} \frac{k^d}{d}\Big) \tag{8}$$

Notice that both (7) and (8) structurally have the form $1 - p_{loss}$. The term p_{loss} can be interpreted as the loss of discriminative power of a simple randomized selection with respect to a perfect strategic selection that always achieves correct detection. Similarly, the p_{loss} term can also be interpreted as the expected performance gain following an IETF ALTO deployment for target selection. Clearly, this model is simplistic as it does not consider all combinatorial aspects which could be used to obtain finer-grained expectations at each depth of the tree. Yet, the main purpose of the model is to serve as a reality check for our experimental results.

3.4. Model comparison

For the sake of illustration, we contrast the TMA, Hypergeometric and Poisson models for variable fault depth f and different probe budgets $\alpha = M/N$. The purpose is to select a single, simple model against which to contrast experimental results in Sec. 6.

To this aim, we start by contrasting the TMA (1), hypergeometric (2), and Poisson (3) models for the *lower bound* detection probability in the left-hand side of Fig. 4. Discrepancies of the TMA (1) and Poisson (3) formulations are additionally reported in the left-hand side of Fig. 5, where discrepancies are computed as simple differences $p^{-}_{HG}(f, \alpha) - p^{-}_{X}(f, \alpha)$ for $X \in \{Poisson, TMA\}$ with respect to the Hypergeometric model of (2). Fig. 4 reports the case for $\alpha = 9/512$ (i.e., the case where a strategic selection achieves perfect classification) and $\alpha = 50/512$ (less challenging since a greater probe budget is available). To avoid cluttering the pictures, Fig. 5 instead only shows the *largest* deviations, which are obtained when $\alpha = 9/512$.

Two interesting observations can be made from the plots. First, Fig. 4 shows that both the TMA and Poisson models very closely approximate the exact Hypergeometric one: notice indeed that lines are hard to distinguish, meaning that for any depth, the Hypergeometric model and the different approximations are all in agreement, as they closely capture the average detection probability for that depth. Second, Fig. 5 allows to better exacerbate these very fine-grained difference between the Hypergeometric model and its approximations: particularly, the figure shows the TMA model to be slightly optimistic and the Poisson model to be slightly conservative (in both cases, absolute error is lower than 3% in the worst case).

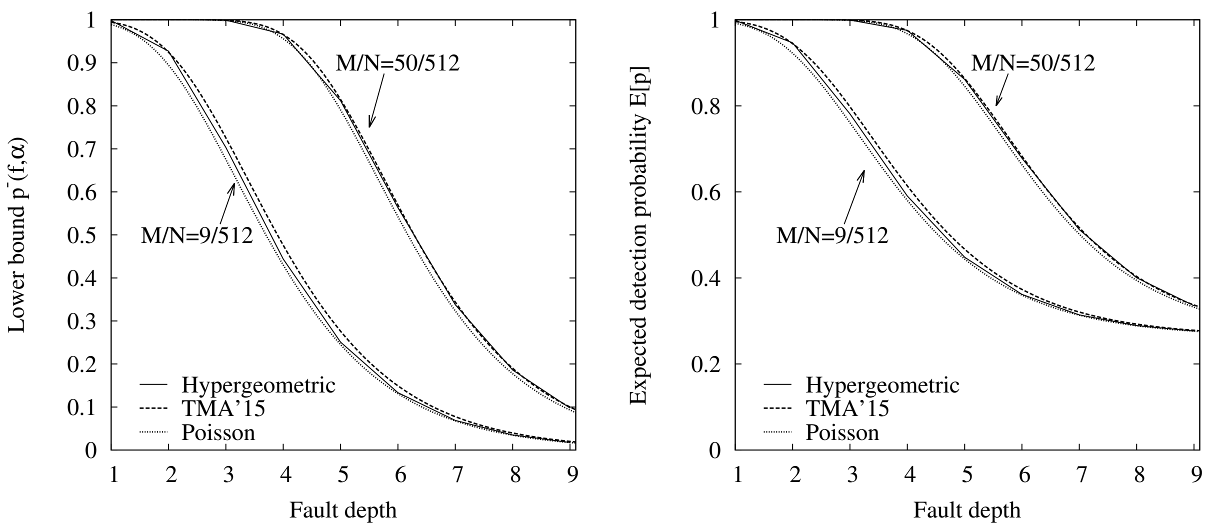


Figure 4: Model comparison: lower bound (left) and expected identification probability (right) for random probe selection and two probe budgets $\alpha = M/N \in \{9/512, 50/512\}$. Notice that lower bound and average detection probability of Hypergeometric, Poisson and TMA'15 models are quite close.

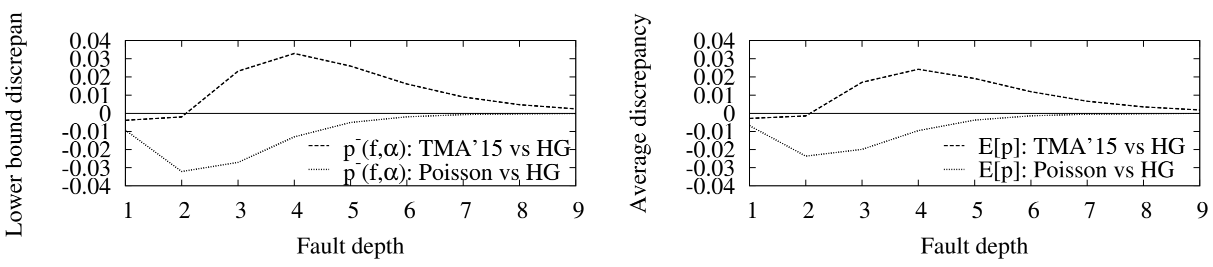


Figure 5: Model comparison: discrepancy between combinatorial Hypergeometric model (HG) vs TMA and Poisson approximations, lower bound (left) and expected identification probability (right) for random probe selection and the smallest probe budget $\alpha = M/N = 9/512$

The *average detection probabilities* for the TMA (7) and Poisson (8) models are compared to that gathered by an hypergeometric model in the right-hand side of Fig. 4 for $\alpha = 9/512$ and $\alpha = 50/512$. As before, discrepancies of the TMA (1) and Poisson (3) formulations are additionally reported in the right-hand side of Fig. 5 for $\alpha = 9/512$. Two remarks are worth making. First, discrepancies remain of the same order of magnitude: given that the Poisson model is simple and accurate (with respect to the Hypergeometric one) and additionally conservative (unlike the TMA model[8]), we select this model as a benchmark for experimental data. Second, as per our previous interpretation, the vertical gaps between the unit detection probability and the curves shown in the right plots of Fig.4 are due to the loss of discriminative power intrinsic to the random selection. This non marginal gap reinforces the soundness of our system design that attempts at exploiting topology-related information (e.g., made available through a simple interface such as IETF ALTO), to exploit benefits of simple yet powerful stratified selection.

4. Troubleshooting Methodology

In this section, we first describe the troubleshooting algorithms in abstract terms, i.e.,mapping on the system model described in the previous section. We then briefly describe NetProbes, the open source diagnosis software we develop for this work, which is available at GitHub [10].

4.1. Troubleshooting algorithm

We treat both clustering and classification problems with a single algorithm, whose pseudocode is reported in Algorithm 1. Assuming the algorithm runs at a source node s, for any performance metric Q (e.g., delay, bandwidth), s collects baseline statistics $Q_0(p)$ with low-rate active measurements towards other peers p.

Troubleshooting is either triggered automatically (e.g., on a changepoint detection of the metric of interest) or on user intervention in case of faults. Irrespectively of the trigger, the troubleshooting originator s iteratively selects up to R batches of B probes, so that $R \cdot B$ represents a tunable probing budget. Selection is made according to a selection policy $\mathcal{S}_p$, based on a probe score $S(p)$. The probe selection is iterative because $S(p)$

Algorithm 1 Detection algorithm at source probe s

B	size of the probe batch	R	number of measurement rounds
P	target probe set, card(P) = M = R $\cdot$ B	$P = P^+ \cup P^-$	results of clustering along $Q(p) - Q_0(p)$
$Q(p)$	sample of metric Q toward target probe p	$Q_0(p)$	baseline for metric Q (e.g., min$(Q(p))$ or $E[Q(p)]$
$S(p)$	score of target probe p (for selection policy)	$\mathcal{S}_p$	probe selection policy (random, IP, balanced, etc.)
$S(\ell)$	score of link ℓ (for detection policy)	$\mathcal{S}_\ell$	link selection policy (naive, argmax, proportional, etc.)

```
 1: Get a baseline $Q_0(p)$ for metric $Q(p),\ \forall p$          ▷ Initialization, over long timescale
 2: for round ∈ [1..R] do                                      ▷ When triggered upon user/ISP demand
 3:     select a batch of $B$ probes according to a probe selection policy $\mathcal{S}_p$, based on score $S(p)$
 4:     for $p \in B$ do
 5:         perform active measurements with $p$ to get $Q(p) - Q_0(p)$
 6:         add probe $p$ to probed set $P$
 7:         partition $P$ into $P^+$ and $P^-$, by K-means clustering on $Q(p) - Q_0(p)$
 8:     end for
 9:     update probe scores $S(p),\ \forall p$
10: end for
11: if topology is not available then                                          ▷ Clustering results
12:     return $P^+$ and $P^-$
13: else                                                                   ▷ Classification results
14:     for probe $p \in P$ do
15:         for link $\ell \in$ shortest path $SP(s,p)$ do
16:             $S(\ell) \leftarrow S(\ell) + \mathbb{1}(p \in P^+) - \mathbb{1}(p \in P^-)$
17:         end for
18:     end for
19:     return link $\ell$ according to a link selection strategy $\mathcal{S}_\ell$ based on scores $S(\ell)$
20: end if
```

can vary, and thus the next batch is selected based on the results of the previous batch.

4.1.1. Clustering vs Classification

At each step, upon collecting B spatial measurements samples, we compute, for each probe p, $Q(p) - Q_0(p)$ and add it to the set P: we cluster partitions P into P^+ and P^- with K-means. Clearly, the choice of partitioning in two sets is rooted in the fact that we are considering a single failure, by which probes are either affected or undisturbed.

Two points are worth stressing: first, the algorithm does not associate any semantic to clusters. For instance, a node in P^+ can be affected by a large delay (so that P^+ represents the cluster of nodes affected by an impairment), whereas in a different experiment a node in P^- can suffer from small available bandwidth (so that P^- represents in the case the cluster of nodes affected by an impairment).

Second, in case of a single failure, it can be expected that probes in the unaffected of the two clusters exhibit no noticeable difference $Q(p) - Q_0(p) \approx 0$, so P^+ and P^- should be interpreted as a syntactical difference. Once the probe budget is exhausted (or once other stop criteria, that we don't mention for the sake of simplicity, are met), the algorithm either returns P^+ and P^- (user-managed case, line 12), or continues with the mapping. When no clear partition can be established, only one set is returned.

While we do not investigate multiple independent failures in this work, K-means enables us to partition P in more that 2 sets, thus the clustering is also able to discriminate situations in which there is more than one anomaly (e.g., clusters will form according to distance from the baselines). Similarly, while we do focus on a metric-by-metric case, K-means naturally copes with cases where $\vec{Q}(p)$ represents a collection of heterogeneous metrics and $\vec{Q}_0(p)$ is the baseline vector.

To map probes in P^+ and P^- to links, the algorithm requires the knowledge of the links ℓ in the shortest path $SP(s,p)$. The score $S(\ell)$ of $\ell \in SP(s,p)$ is incremented by +1 for $p \in P^+$ and decremented by -1 for $p \in P^-$. Since we make no assumption on the semantic of the metric of interest, the algorithm needs to know if links with the largest (smallest) $S(\ell)$ scores have to be returned (line 19 of Algorithm 1), as otherwise both links with extremal score (i.e., argmax *and* argmin) are to be returned. For each new metric in the set of observable, (simple) domain-expert knowledge is thus required for correct interpretation of the results.

4.1.2. Link selection strategy

Root cause identification (line 19 of Algorithm 1) is done according to a link selection policy $\mathcal{S}_\ell \in \{\text{naïve}, \text{proportional}, \text{argmax}\}$.

The *naïve random* method makes an informed guess by selecting one of the D^+ links in the path $\ell_{D+}, \ldots, \ell_1$ to the root. We say the guess is "informed" since the naïve strategy has a success probability of $1/D^+$, which is much larger than the lowest bound $1/k^{D^+} = 1/N$ in case of a "blind" random guess over all N links: indeed, conditioning over the fact that the origin monitor launched a measurement campaign since it observed an anomaly, it follows that the anomaly lays in the path to the root, which comprises $D+$ links. A better strategy is to select links *proportionally to their score*: specifically, in this case one random link ℓ_i is returned with probability $S(\ell_i)/\sum_j S(\ell_j)$. In an attempt to bound the maximum error, the risk-adverse proportional policy however suffers from a larger average error, since it considers non-faulty links by default (albeit with a smaller probability). A final strategy is to select a single link: namely, *the link with the largest (smallest) score*: i.e., link ℓ_i with $\text{argmax}_i S(\ell_i)$, where ties are broken at random. The risk-prone argmax (argmin) instead aims at reducing the average error, at the price of a larger maximum error in case the selected link is not the faulty one.

Clearly, the suitability of argmax vs argmin depends on the metric Q semantic (e.g., larger packet loss deviation requires *argmax* detection, while bottleneck bandwidth mandates *argmin* criterion). Instead, suitability of proportional vs argmax/argmin also depend on how well the clustering algorithm separates the data, which in turns depend on how robust/noisy are the measurement results. We expect argmax/argmin to perform well where clusters are well separated along the considered metric.

4.1.3. Probe selection strategy

In terms of probe selection strategy $\mathcal{S}_p$, we consider both the ISP case where an oracle is able to perform a strategic selection as earlier illustrated, as well as the end-user case where we experiment with $\mathcal{S}_p \in \{\text{random}, |IP(s) - IP(p)|, \text{balance}\}$ and combinations of the above.

Stratified selection is performed with an ALTO-like oracle as earlier discussed. *Random selection* is useful as a baseline and to compare with the model, yet we do not expect it to be well performing. We additionally consider probe selection policies that are more complex to model, such as the *absolute distance in the IP space*: the $|IP(s) - IP(p)|$ metric (where each IP is interpreted as a 32 bits integer) however implicitly makes assumption on the way ISPs allocate IP addresses, and is thus not expected to be resilient across different setup. Finally, the *balanced selection* attempts at equalizing the size of P^+ and P^-, by selecting an IP that is close to IPs in the small cluster, and far from IPs in the large cluster. In other words, denoting $P^{small} = argmin(|P^+|, |P^+|)$ so that $P^{large} = P \backslash P^{small}$ is the largest cluster, the balanced policy will attempt at selecting a new node n such that $|IP(p) - IP(n)|$ for $p \in P^{small}$ is minimized. Clearly, balanced selection also suffer from assumptions on IP addresses allocation.

Notice that among the outlined strategies, the balanced one is the only one that directly benefits of incremental probe selection (i.e., R batches of size B each in line 2-3 of Algorithm 1). Yet, the batch size B also roughly expresses the maximum parallelism of measurement processes, to limit possible mutual interference between measurement results. We discuss this issue further in Sec.7.

4.2. Software implementation

We implement the above algorithm in NetProbes, a distributed software written in Python 3.x that runs on end-hosts and executes a set of user-defined active measurement tests. NetProbes, which is available at [10], consists of over 7000 lines of codes, scattered in about 70 files, and arranged in over 150 classes. The software suite comprises agents having a client and a server interface, and includes an orchestrator to facilitate launching small scale experiments in automated or interactive way (textual *ncurses* and graphical *Tk* user interfaces).

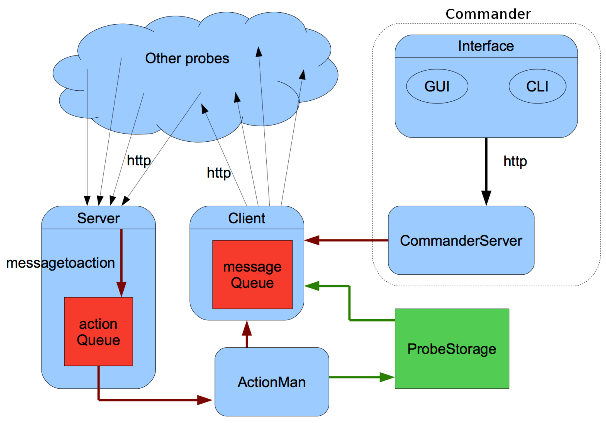


Figure 6: Schematic organization of the main NetProbes classes

It is worth pointing out that the set of measurement tasks that can be performed by NetProbes agents (e.g., HTTP or DNS requests, multicast UDP tests, etc.) is far larger than what we consider within the scope of this paper. NetProbes agent can more generally perform a set of measurement tasks (or actions): to avoid mutual interference between experiments, agents run a single action at a time, as illustrated in Fig. 6, and manage queue of measurement tasks. As this limits, but does not avoid, mutual interference, we come back on this later in Sec.7.

In the context of this work, NetProbes agents are deployed at end-user devices and gateways form an overlay. They perform a set of periodic measurements to monitor the paths in the overlay and collect a baseline network performance. When the user experiences network performance issues, the NetProbes agent running at the user device launches a troubleshooting task. To perform this task, measurement requests are sent to other agents, over HTTP. For the sake of simplicity, we avoid decoupling measurement and inference processes. Hence, upon reception of measurement results, the origin monitor assess the severity of the performance issue and the location of the faulty link by executing the clustering and classification steps illustrated in Algorithm 1.

5. Calibration experiments

Before running a fully-fledged measurement campaign, it is mandatory to perform a rigorous calibration phase – yet this phase is often neglected [29]. In this work, we follow an experimental approach using emulation in Mininet, to control the duration and the location of the faults. However, it is unclear how well state-of-the-art delay and bandwidth measurement tech-

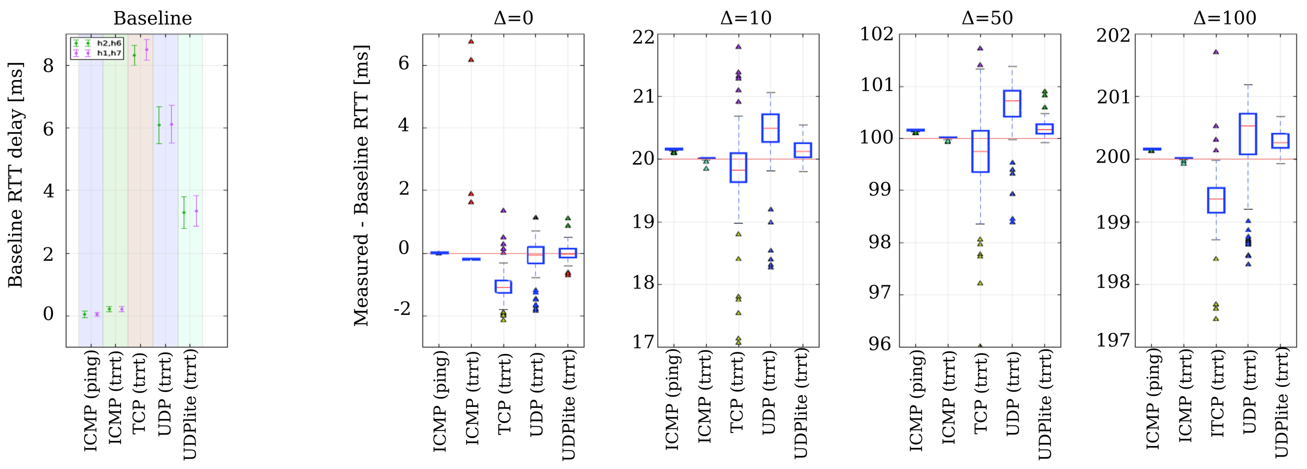


Figure 7: Calibration of delay measurements: baseline $Q_0(p)$ for two sample end-to-end paths (left) and boxplots of $Q(p) - Q_0(p)$ for several protocols and controlled delay Δ (right)

niques perform in Mininet. In order to disambiguate inconsistencies due to Mininet from measurement errors intrinsic to measurements techniques, we perform calibration experiments for a set of delay (expectedly easy) and bandwidth (notoriously difficult) measurement tools and assess their accuracy in Mininet.

5.1. *Emulation environment setup*

Mininet [9] is an open source emulator which creates a virtual network of end-hosts, links, and OpenFlow switches in a single Linux kernel and supports experiments with almost arbitrary network topologies. Mininet hosts execute code in real-time, exchange real network traffic, and behave similarly to deployed hardware. All the software developed for a virtual Mininet network can run in hardware networks and be shared with others to reproduce the experiments. Mininet provides the functional and timing realism of testbeds in addition to the flexibility and full control of simulators. Experimenters configure packet forwarding at the switches with OpenFlow and link network characteristics (e.g., delay and bandwidth) with the Linux Traffic Control (`tc`). Reproducing experiments from tier-1 conference papers indicates that results from Mininet and from testbeds are in agreement [9]. This happens provided that the emulated network size does not yield to CPU, memory or PCIx bandwidth bottlenecks, which we care to avoid.

We use Mininet to build a virtual network with the topology depicted in Fig. 3 on a Quad-Core Intel Xeon E5-1603 (2.8GHz 10MB cache) equipped with 24 GB of RAM. Given our setup, we perform preliminary stress tests to discover that this practically limits our *overlay size* to 512 nodes. The size of overall emulated network is of course larger due to all switches at intermediate depths: e.g., in a binary tree case, the network size is roughly the double of the overlay size. In practice, this overlay size allow us to consider two extreme cases corresponding to fairly deep trees with depth 9 and fanout 2, or fat trees with fanout 8 and depth 3 that both have $2^9 = 8^3 = 512$ leaves.

In the calibration phase, we run the selected measurement tools on probes 1 and 2. In our delay experiments, we impose five different delay values ($\Delta \in \{0, 10, 50, 100\}$ ms, so that the expected RTT=2Δ) on ℓ_3 located at depth $d = 3$ in the tree. At each delay level, probes 1 and 2 perform 50 measurements of round trip delays to probes 7 and 6 respectively (250 measurements in total for each pair of probes). We use Mininet processes through the Python API to issue `ping` and `traceroute` to measure RTTs (we test `traceroute` with UDP, UDP Lite, TCP, and ICMP).

Similarly, in the bandwidth experiments, we vary the link capacity of ℓ_3 (100 Mbps, 10 Mbps, 1 Mbps) under three different traffic shapers, namely the hierarchical token bucket (HTB), the token bucket filter (TBF), and the hierarchical fair service curve (HFSC). Among the set of measurement tools designed by the research community to estimate the available bandwidth [30], in this work we limitedly report the calibration of five such tools. Namely, we select Abing [31], ASSOLO [32], IGI [33], Iperf[34] and Spruce[35] and perform 30 measurements of the available bandwidth between probes 1 and 7 and probes 2 and 6 (450 experiments for each probe pair).

We use these two metrics as a non-exhaustive example of metrics that sit at the opposite spectrum in terms of complexity: delay is easy to enforce and measure, unlike available bandwidth. We note that while a full-blown system shall extend the set of metrics (e.g., from signal to noise ratio at Layer-2, to loss rate at Layer-3, to reordering at Layer-4 and to application-specific metrics at Layer-7, possibly combined in a single feature vector), however the fundamental properties of our system are well understood even with a limited set of metrics. Hopefully, in reason of their different complexity, these two metrics not only provide illustrative examples, but are also representative of the boundaries within which performance for other relevant metrics may lay.

5.2. *Delay calibration*

We expect delay measurements to be flawless. Yet we observe that the first packet sent between any two hosts exhibits a large delay variance: this is due to the fact that the corresponding entry for the flow is missing in the virtual switch and thus

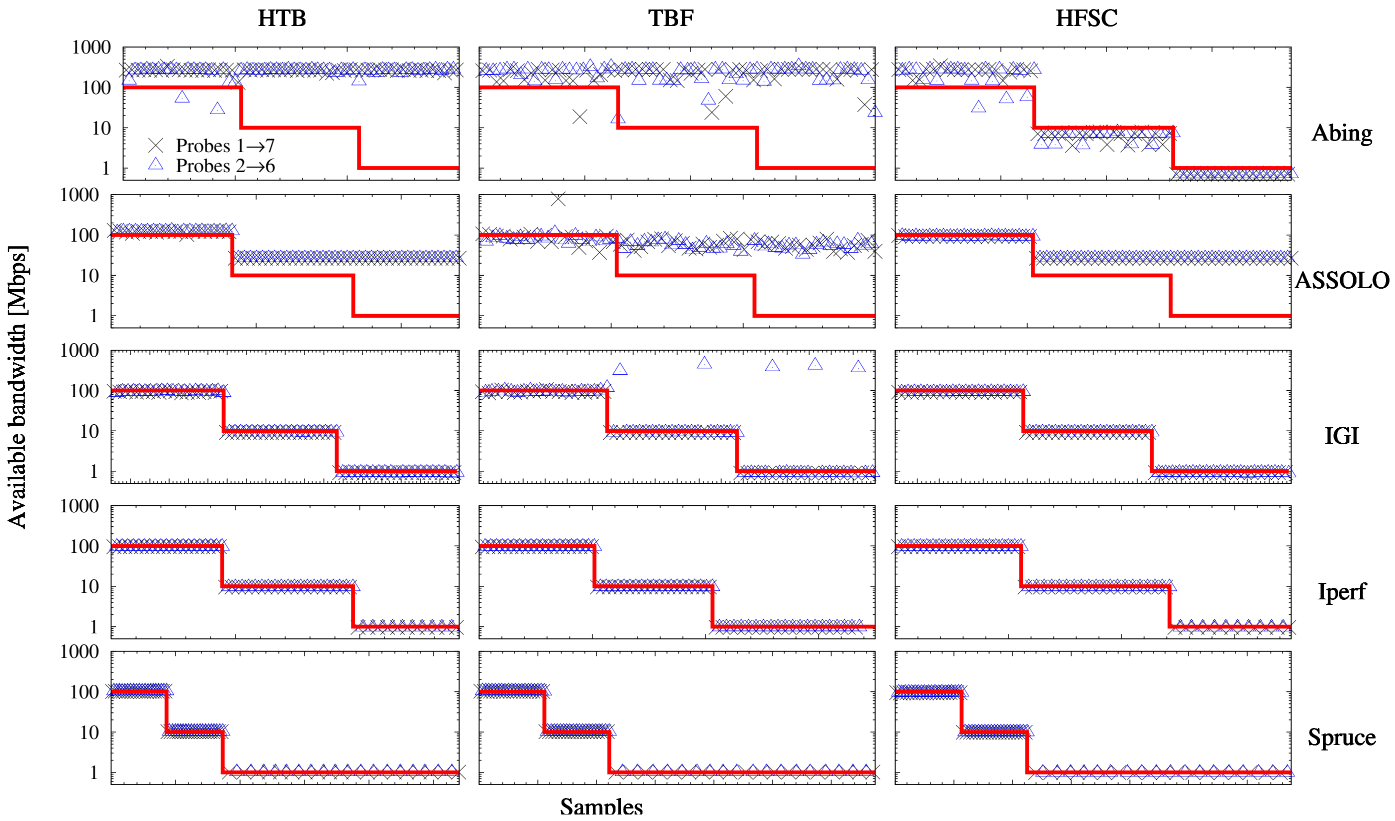


Figure 8: Calibration of bandwidth measurements: temporal evolution of {Abing, ASSOLO, IGI, Iperf, Spruce} × {HTB, TBF, HFSC}.

requires data exchange between the OpenFlow controller and the virtual switch, whereas the forwarding entry is ready for subsequent packets. We thus do the baseline $Q_0(p)$ over multiple packets (50 for delay) to mitigate this phenomenon. Doing a baseline and subtracting it from each delay measurement enables an accurate study of the effect of the imposed delay value on the accuracy of the measurement technique. The leftmost plot in Fig. 7 show baseline RTT measurement between a pair of hosts with ping (IMCP) and traceroute (UDP, UDP Lite, TCP, and ICMP). Baseline measurement already show noticeable variability, with TCP and UDP traceroute measurement exhibiting a 5ms average delay.

The four boxplots in Fig. 7 then show differences with respect to the baseline for $\Delta \in \{0, 10, 50, 100\}$ ms. The leftmost boxplot ($\Delta = 0$) shows that as expected, by computing a difference with respect to the baseline, the mean error is close to $2\Delta = 0$ (except for TCP traceroute). Also in the other boxplots, difference to the baseline approaches 2Δ as expected. Still, ICMP measurements exhibit the smallest variance among all techniques. Fig. 7 already reveals that, even simple delay measurement in a controlled emulation environment are subjects to noise. In our testbed, we however observe the extent of this noise to be low: for all the delay measurement techniques, the bulk of the distance from the baseline is less than 1 ms: it follows that changepoints can be detected with a very fine grain. Moreover, we note that using ICMP brings the absolute error to less than 0.1 ms for both `traceroute` and `ping`.

Of course, the kind of noise in a controlled environment (e.g., container based virtualization overhead) differ from the sources of noise that can be expected in an Internet environment. Internet delay measurement can be noisy due e.g., to interfering traffic and bufferbloat [36, 37, 38], or by properties of the crafted ICMP packets [23]. It follows that the calibration we perform can hardly be portable to other scenarii: hence, our contribution here is on illustrating the need for this methodological step, more than merely describing general results cast in stone. Rather, we advocate that similar methodological calibration steps need to be performed for each new environment/testbed.

Summarizing, from this calibration phase, we select ICMP `ping` to measure delay: as the measurement noise is insignificant in the Mininet testbed, errors in the classification outcome should be solely attributed to our troubleshooting algorithm.

5.3. Bandwidth calibration

We expect bandwidth measurements to be difficult. Additionally, we stress that while comparison of bandwidth estimation tools under the same experimental conditions has already been carried on (see [30] and references therein), we are not aware of any study *jointly* considering bandwidth estimation and bandwidth shaping. Yet we believe this to be an important detail, as we expect different shaping techniques to alter packet inter-arrival time in different ways, possibly affecting accuracy of the bandwidth inference technique. While a preliminary joint selection of bandwidth shaping and measurement tools is necessary, as far as calibration of our test is involved, we deem a thorough study, albeit interesting, outside the scope of the present study.

We compare the performance of five bandwidth estimation tool (Abing, ASSOLO, IGI, Iperf and Spruce) in the absence of cross traffic under three traffic shapers (HTB, TBF, HFSC). Among the bandwidth estimation tools, we point out that Iperf is particularly intrusive as it designed to *measure throughput* by sending data through backlogged transfers, whereas the other tools are designed to *infer the available bandwidth* by sending a very limited amount of traffic, so that the inferred bandwidth is available to the user application unlike in the Iperf case. As such, we use Iperf to get a reliable reference, but we do not otherwise expect Iperf to be of practical use in reason of its intrusiveness. The remaining tools, namely Abing, ASSOLO, Spruce and IGI are instead characterised by lower intrusiveness, inferring the available bandwidth based on the dispersion of packet pairs/trains measured at the receiver, and differ in the way the packets are spaced [30]. Of course, our aim is not to perform a deep and thorough study of the mutual impact of shaping and bandwidth estimation techniques, but to select a pair of techniques that avoids (or at least) minimizes unwanted artifacts in our emulation.

Fig. 8 reports the evolution of the estimated available bandwidth as a function of three link capacity values for the cross product of $\{$Abing, ASSOLO, IGI, iperf, Spruce$\} \times \{$HTB, TBF, HFSC$\}$. As expected, we see artifacts arising, with a complex dependency patterns of shaping techniques biasing results of some bandwidth measurement techniques. These results and tradeoffs are interesting per se and would deserve an attention that is however beyond the scope of this work, were we are more interested in ruling out biases, than explaining their ultimate causes.

With this aim in mind, we can see from Fig. 8 that Abing systematically fails in estimating the available bandwidth under HTB and TBF shaping, while the estimation is correct with HFSC. Similarly, ASSOLO fails in estimating 1 Mbps available bandwidth under all shapers, and additionally fails the estimation of 10Mbps under TBF. In contrast, IGI succeeds in accurately tracking changes of available bandwidth at ℓ_3, although outliers are still possible (see IGI+TBF). A downside of IGI is that the measurements last longer than measurements with Abing or ASSOLO. Finally, Iperf and Spruce are always accurate; yet Iperf is too intrusive to be used in practice, whereas Spruce makes an interesting candidate.

In addition to our measurement campaign, we can leverage comparison work such as [30] to further guide our tool selection. Specifically, [30] observes that IGI is fast but injects a larger amount of probe traffic in the network, while Spruce is lightweight but slow to converge (Spruce repeats the measurement 100 times, with measurements emulating Poisson sampling, and the result is averaged among all samples). ASSOLO and Abing sit in the middle of this tradeoff between speed and intrusiveness. Finally, [30] also show that the stability and the

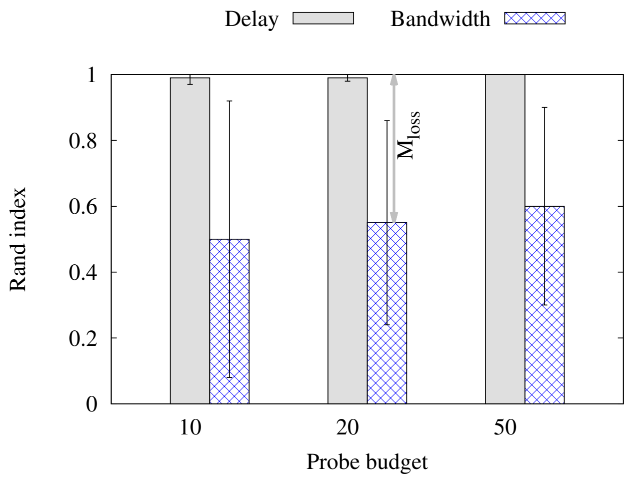


Figure 9: Clustering results at a glance: Rand index of experimental output of clustering algorithm vs ground truth clusters

accuracy of measurements obtained with IGI increase when the intensity of the cross-traffic is higher, an important and desirable property.

The most important takeaway from our measurement is that unwanted interaction of shaping and packet pair techniques generate estimation errors of significant magnitude, which could likely invalidate all experiments – which confirms once more the importance of this calibration phase. As in the previous case, it has to be pointed out that the relevance of this calibration study is limited to a Mininet testbed: indeed, the aim of this calibration was to find a combination of measurement technique and bandwidth shaper that induce the least possible distortion in our testbed. Conversely, while in the general case the choice of bandwidth measurement technique is a free parameter, the bandwidth shaping or active queue management mechanisms that are to be found in practice may differ significantly from those found in a controlled environment, and cannot likely be controlled. A different calibration, such as the one performed in [30] is then needed in this context.

Summarizing, (almost) all tested bandwidth measurement techniques appear to be accurate under the HFSC shaper, and both the IGI and Spruce bandwidth measurement techniques yield consistent results under any traffic shaper under test. Additionally, related work [30]testifies IGI to be a reasonable choice that would perform well in a more general context. As accurate input is a necessary condition for troubleshooting success, we thus select a pair in this intersection (namely, IGI and HFSC), that we expect to perform well in our Mininet emulation environment.

6. Troubleshooting experiments

Equipped with a calibrated testbed, we now assess NetProbes troubleshooting performance. Specifically, we evaluate the quality of our clustering and classification for various probe budgets for controlled faults (e.g., doubling delay or halving bandwidth) at different depths of regular tree topologies. We especially pay attention to the performance of the end-user only vs ISP-assisted cases. All the scripts to reproduce the experiments are available at [11].

In more details, we first assess clustering performance for delay and bandwidth measurement (Sec. 6.1). We next compare classification performance, which including real-world noise, to expectations of the probabilistic models, that instead neglect noise (Sec. 6.2). We finally perform a sensitivity analysis of the troubleshooting algorithm in controlled settings by varying topological properties such as tree degree (Sec. 6.3), probe selection policies $\mathcal{S}_p$ (Sec. 6.4), and link selection policies $\mathcal{S}_\ell$ (Sec. 6.5).

6.1. Clustering results

We perform experiments over a binary tree scenario ($k = 2$) with depth $D^+ = 9$ and $N = 512$ leaf nodes. In this case, a strategic probe selection needs $M/N = 9/512$ probes ($\alpha = 1.75\%$) to ensure perfect classification, but we consider a larger budget $M = \{10, 20, 50\}$ in our experiments, where without loss of generality probes are selected in $R \in \{1, 2, 5\}$ rounds of $B = 10$ batches each.

Unless otherwise stated, we use a random probe selection $\mathcal{S}_p$ and an argmax link selection $\mathcal{S}_\ell$ policies. We first evaluate the clustering methodology by comparing the two sets of affected and unaffected probes obtained from the algorithm with the clusters that can be built using our controlled ground truth. Intuitively, in case the measured clusters are equal to those obtained via the ground truth, then the performance of the algorithm are optimal. The similarity between cluster is best captured using the well-known Rand index, which takes value in $[0, 1] \subset \mathbb{R}$, with 1 indicating that the data clusters are exactly the same and 0 indicating that cluster are totally different. More information about the Rand index from an information retrieval perspective can be found in [39].

Since we have full control over the location of the fault, we build our ground truth by assigning the label "affected" to all the available probes (under a given budget constraint) for which the path to the diagnostic probe passes through the faulty link. The remaining probes constitute the unaffected set. Fig. 9 shows that, provided measurements are accurate, the clustering methodology successfully identifies the set of probes whose paths from the diagnostic software experience significant network performance disruptions (and as a consequence accurately identifies nodes in the complementary set of unaffected probes). For budgets of 10, 20 and 50 probes, the rand index shows perfect match between the ground truth and the clustering output in the case of delay measurement.

Results degrade significantly instead for bandwidth measurements (notice that the rand index is lower than 0.6): we point out that the loss of accuracy is not tied to our algorithm, but rather to the measurements that are the input to it, which was partly expected and confirms that calibration is a necessary yet insufficient step. The hidden factor in the previous calibration is the fact that we were performing a *single* capacity measurement in isolation, with capacity set by a given shaping mechanism. In this case, all $B = 10$ probes in the batch perform *concurrent experiments*, which happen to mutually interfere with each other: when multiple measurements are scheduled at the same time, since many bandwidth measurement tools rely on effects of cross-traffic to estimate available bandwidth, the probe traffic of one tool may interfere with the estimation of a host and vice-versa [40]. It thus appears that already 10 concurrent bandwidth measurement are able to generate significant noise; at the same time, it also appears that situation improves over multiple independent batches (notice a slight increase of the rand index from $R = 1$ to $R = 5$), so that a more careful scheduling of such experiments may be desirable to reduce (or avoid if possible) this interference (see Sec. 7).

Summarizing, and abstracting from limits of a specific measurement techniques, results indicate that our algorithm works well in practice without requiring knowledge of the network topology.

6.2. Classification results

Root cause link identification is a clearly more challenging and important objective, which we analyze in what follows by restricting our attention to delay experiments: as the classification step is a deterministic mapping from the clusters, as long as the measurement error remains small, the results of the classification task are not affected by the specific metric under investigation. Otherwise stated, as opposite to merely illustrating the algorithm performance under delay measurements, we expect classification results to be representative of a large set of metrics (although they are not representative of bottleneck localization, as per Fig. 9).

We now show that the experimental and modeling results are in agreement, with a random probe selection policy and a budget of $M = 50$ probes, which corresponds to $\alpha = 50/512 = 9.75\%$. For each fault depth f, we perform 10 experiments by randomizing the set of destination probes. Results, as reported in Fig. 10, depict the correct classification probability of the model vs the experiments. Recall that equation (3) gives a lower bound $p^-(f, \alpha)$ to the experimental results, while (6) models the average expected detection probability $\mathbb{E}[p]$. It can be seen that the models quite closely capture the experimental results.

We additionally report results of a strategic selection with as $\alpha = 9/512 = 1.75\%$, that is capable of perfect classification with just one fifth $M = 10$ of the probe budget of the random selection. Two arrows further annotate the loss of discriminative power with respect to a strategic selection. The right arrow loosely indicate the p_{loss} factor due to random selection; notice that the extent of the gap has a qualitative value, since the probing budget of the strategic vs random selection differ (a quantitative assessment would require using $\alpha = 9/512 = 1.75\%$ also for the random selection; both curves are reported in [8]).

The left arrow reports a special degenerated case worth mentioning, that is not captured by the model (this case that was only briefly commented in [8] due to lack of space). Namely, faults affecting link ℓ_1 close to the *root* of the tree for fanout $k = 2$ cannot be reliably discriminated by leaf measurement only: indeed, *all* shortest path to affected probes pass through

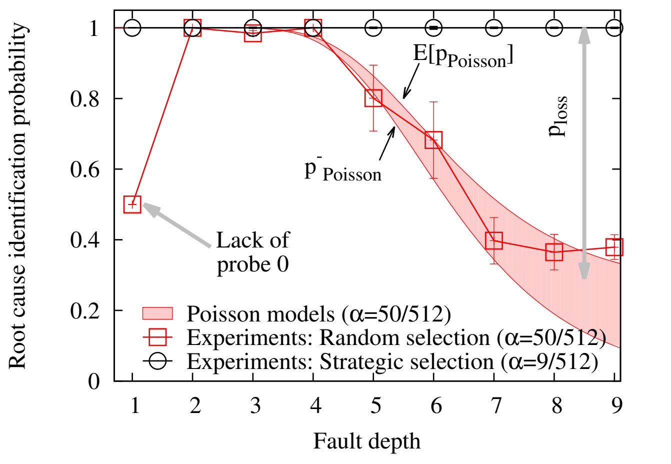


Figure 10: Classification results at a glance: Probability of correctly identifying the faulty link for the Poisson lower bound (3) and average (6) models vs experiment

both root links, that have thus the same score: breaking ties at random equal to correct classification in only 50% of the cases. Notice that this holds for root-link failures and only for $k = 2$, while trees with higher k-arity are not concerned. Thus, successful classification requires being able to condition over the set of links $\ell_{D+}, ..., \ell_1$ from the leaf to the root, which can be done with the assistance of an ISP probe 0 (recall Fig. 3): in case measurements involving Probe 0 are affected, this means that ℓ_1 is the root cause link; in case Probe 0 is not affected, this means that the other root link (symmetrical to ℓ_1) is affected (while ℓ_1 is not).

Summarizing, we see our proposed models accurately estimate classification performance. Additionally, comparing results of random vs strategic selection, we gather that knowledge of the network topology needs to be exploited, which is possible to do by simply biasing probe selections without revealing the exact topology. Finally, we see that deploying probes in specific points in the network may be required in some cases to achieve perfect classification.

6.3. Impact of topology

We study the impact of the network topology on the classification performance. We use two $d \times k$ trees with d^k =512 probes (i.e., leaves) each. The first tree has a depth $d = 3$ and a fanout $k = 8$ while the second tree has a depth $d = 9$ and a fanout $k = 2$. Fig. 11 reports the correct detection probability of the faulty link as a function of the depth of the injected fault in the tree, along with a pictographical representation of these topologies.

First, consider the deepest 9×2 tree. As expected, results indicate that the correct detection probability decreases as the fault depth increases: when the root cause link is located close to the *leaves* of the tree, it is harder to randomly sample another probe which is also affected by the fault. In this case, we thus need a smarter probe selection strategy to improve the link classification performance.

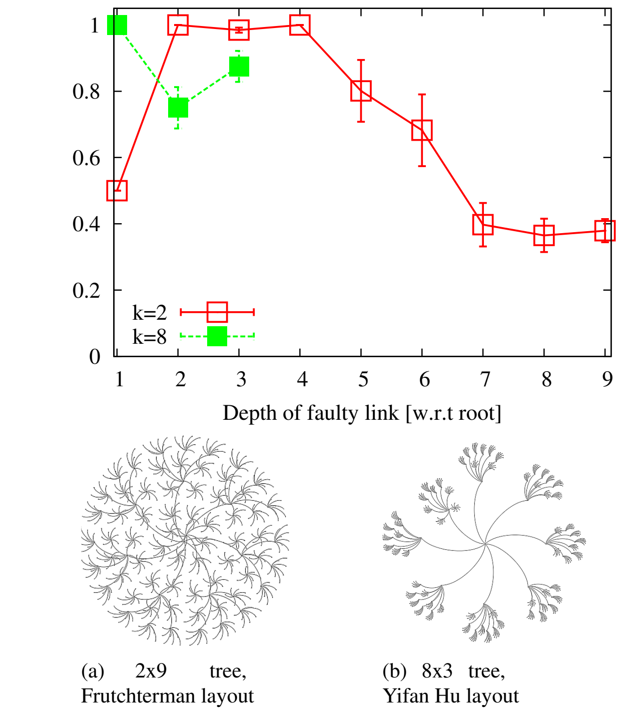


Figure 11: Sensitivity analysis: Impact of network topology on classification with random selection

Next, comparison performance of both trees show an interesting (and partly artificial) tradeoff. Notice indeed that, for faults located close to the root, detection is only ambiguous for $k = 2$, so that perfect classification is achieved for $k = 8$. As the fault depth increases, the curves cross: at depth $f = 2$, 3 and 4 the classification is perfect for the $k = 2$ tree, which is no longer the case for $k = 8$. This can be explained with the fact that whereas the absolute depth is the same, the relative depth largely differs. It follows that a fault at depth $f = 3$ in the $k = 8$ tree should be compared with a fault at maximum depth in the $k = 2$ tree: with this in mind, it is easy to see that fault detection is easier in trees with larger fanout.

Summarizing, binary trees provide a conservative estimate of classification performance.

6.4. Impact of the probe selection policy $\mathcal{S}_p$

We have early shown that a great loss of discriminative power is rooted in naive random selection of probes, arguing for the need of strategic selections implemented via biased probe suggestion through an IETF ALTO-like interface. Yet, there may be cases where either operators are not willing to (or cannot bare the cost of) deploying such strategies. It is thus worth exploring whether it would be possible to devise simple criteria to hopefully approach a stratified selection.

To this aim, we compare random selection with probe selection policies $\mathcal{S}_p$ policies based on IP-distance (IP), size of the clusters (balance), and a linear combination of both. Recall that

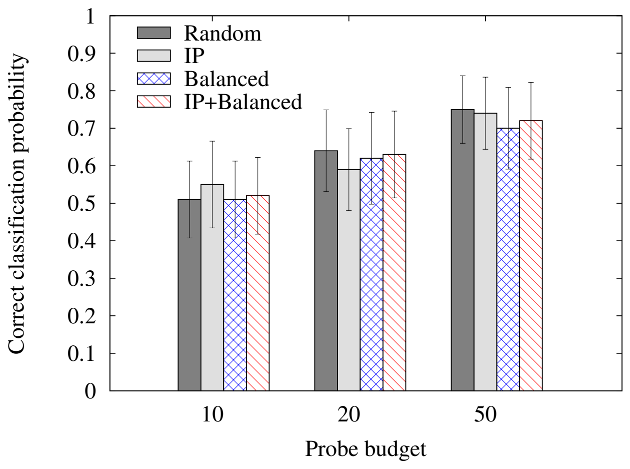


Figure 12: Sensitivity analysis: Impact of probe selection policy $\mathcal{S}_p$

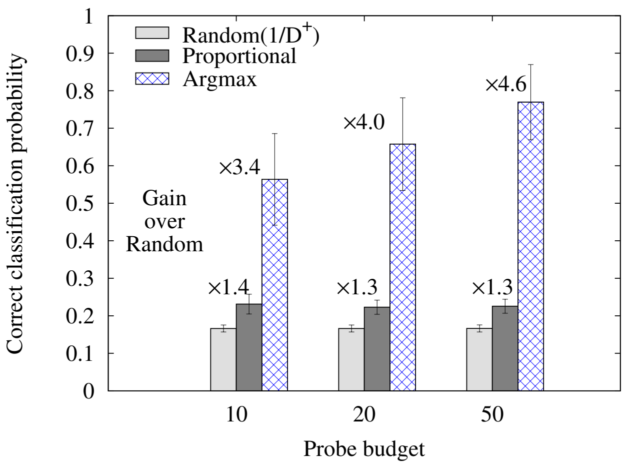


Figure 13: Sensitivity analysis: Impact of link selection policy $\mathcal{S}_\ell$

in Algorithm 1, probes were probabilistically selected according to a generic score. Notice this mechanism is quite generic: while the case of uniform random selection is represented by probes with homogeneous scores, it is possible to weight multiple scores (linearly or not) into a single score.

We average the results over all the depths of the binary tree and contrast them with a random selection policy. Unfortunately, as it can be seen from Fig. 12, the discriminative power is roughly the same over all probe selection policies. In other words, the considered set of metrics do not bring any useful information to bias the selection upon, so that at most a limited gain over uniform selection appears. For instance, the absence of any notion of netmasks and hierarchy in the IP-distance makes it hard to extract information about how topologically close/far probes are from each other. While it would be possible to design metrics that take into account tree fanout and depth, we point out that any further assumption about topological properties correspond to a significant loss of generality: in this case, which implicitly means that a good knowledge of the topology is available, an ALTO-like solution seems preferable as more flexible.

It is thus worth questioning whether these discouraging results are due to the use of a limited set of metrics. At first sight, an obvious metric that we did not explicitly took into account is represented by the IP-TTL field: in the case of trees, having an indication of the hop-count distance traveled by packets in the data plane can greatly assist in understanding to which of the different strata each probe belongs to. Yet it is easy to convince that TTL is only apparently useful. To begin with, TTL is easily observed a posteriori, so that it would be a more useful filter, but could not be used to perform probe selection. Moreover, since Mininet uses virtual switches to construct the network, no routing operation is performed on packets, so that the IP-TTL field remains unchanged and is therefore not helpful in our testbed. Additionally, while it is possible to learn TTL over time, its relevance in real operational networks remains dubious due to weak correlation of TTL with spatial location due the widespread presence of middleboxes[22]. As a consequence, TTL-based stratification cannot be considered a valid alternative.

Summarizing, the discriminative power of a stratified selection is hardly implemented in generic distributed settings, which reinforces the need for cooperation between the network and the applications as this work proposes.

6.5. Impact of the link selection policy $\mathcal{S}_\ell$

Finally, we use three different policies to select the faulty links: $\mathcal{S}_\ell \in \{\text{random}, \text{proportional}, \text{argmax}\}$. Results, averaged over all depths of the binary tree, are reported in Fig. 13. The plot is further annotated with the gain factor over the random selection: while proportional selection brings a constant improvement of about 40%, the argmax policy brings considerable gains (in excess of a factor $4\times$) which grow with the probe budget. The reason why the argmax policy is, on average, very effective, is that links scores are well separated, which is expected in reason of the good clustering properties early shown. Additionally, notice that the argmax policy is not particularly hurting even in degenerate case where the faulty links is at the root of the tree as early shown – the error here is the same as in the proportional case.

While the argmax policy is thus a simple and valid choice when no measurement error appear, one last comment is worth making. Indeed, since P^+ and P^- have no semantic explicitly associated (i.e., for low bandwidth $P^- = P^{faulty}$, while $P^+ = P^{faulty}$ for high losses) a domain expert is required to specialize the argmax vs argmin selection policy to each metric. Alternatively, metrics in the framework could be defined in such a way to be able to deterministically associate, e.g., large deviations with impairments. This is a crucial step required to make the framework of practical use, other practical considerations are discussed in Sec. 7.

Summarizing, in case no measurement bias or artifact arise,

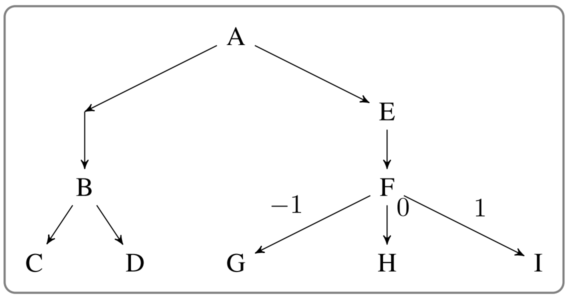


Figure 14: Example of diagnosis tree formalism

the argmax policy appears to be a simple and good enough policy for the identification of faulty links.

7. Practical considerations

In this paper we have proposed, modeled, implemented and experimentally evaluated an algorithm for network troubleshooting. Successful deployment of NetProbe in operational networks is of course conditioned to the availability (and maintenance) of the software for the most popular end-user platforms (Android, iOS, Windows, etc.), an engineering work that goes beyond the scope of this paper. Before the NetProbe software can be used in a operational network, there are also a number of scientific challenges that need to be solved, that this section discusses.

7.1. Diagnosis trees

We report considerations of practical relevance by adopting a diagnosis graph formalism, exemplified in Fig. 14 for the sake of clarity, where diagnosis actions are arranged as a tree. Depth in the tree represents a loose temporal sequence, so that items at the same depth are actions happening in parallel. (e.g., actions B and E, or any in $\{C, D, G, H, I\}$). Branches can be either conditionally or systematically followed: for instance, irrespectively of results of action A, the system launches actions E and B (the former is faster than the latter, so their execution is not necessarily happening or completed at the same time). Similarly, C and D are launched after completion of task B. Conversely, depending on the result of action F, either action G (when $F = -1$) or H (when $F = 0$) or I(when $F = 1$) is launched.

This simple formalism allows to both express (complex) sequence of measurements, as well as encode drill-down troubleshooting [41]. In the case of drill-down methodologies, chains of actions are followed depending on values of their predecessor actions, so that one single path of the tree is followed from the root to the leaves (i.e., in case all edges have conditions depending on results at the previous node) In the case of measurement sequences, the digraph representation immediately conveys a variable degree of parallelism and loose scheduling properties, that ultimately allow to gather a set of measurement features (namely, in the example of Fig. 14 the feature vector will be constituted by all the A, B, C, D, E, F measurements and one of the {G,H,I}) over which data mining or big data approaches can be applied).

With the above formalism, we now list challenges that can be faced by an NetProbe software deployed in an operational network, and illustrate guidelines for their solution.

7.2. Scheduling tradeoff

Assume for simplicity that measurements have no conditional execution[7], and represent three scheduling strategies as shown in Fig. 15. At the extreme left, all measurements are scheduled in sequence. The main drawback of this scheduling strategy is the long execution time: as the measurement conditions evolve over time, diagnosis decisions are possibly taken over measurements that represent a different behaviour of the network service to be measured, weakening the correlation between measurements. At the extreme right, all measurements are scheduled in parallel, which guarantees the shortest execution time, but possibly raises problems due to possible interference of multiple measurements happening in parallel. Notice that while the figure represents a single tree, this tree will possibly be instantiated per user. Therefore, the number of simultaneous measurements due to the fanout of the per-user tree has to be scaled up by the scale of the measurement campaign. The intermediate scenario represents a tradeoff between the long duration of sequential scheduling and the possible interference of parallel scheduling. Remapping a sequential or a parallel tree to an intermediate one is not easy in the general case; yet, scheduling implications have a possibly determinant effect on the result of the clustering/classification algorithms, which are thus worth discussing.

Clearly, the existence of many independent and concurrent instances of per-user diagnosis graphs only makes the problem harder. To cope with this, baseline measurements are easily desynchronized in fully distributed settings (e.g., by randomization and low frequency). However, in case of a troubles with a very popular Internet service, it is important to avoid a measurement flash crowd: a global view is needed in this case to avoid mutual interference across diagnosis trees (or even denial of service to the troublesome service), and an ALTO-like interface could provide some loose yet effective means of coordination.

7.3. Implications of temporal properties and guidelines

Timely detection and temporal coherence would benefit of a parallel scheduling, which may not however be feasible in practice, and which raises other potential issues. In machine learning terms, the vector of measurements (A, B, C, D) represents a scenario where all measurements are taken at possibly different times ($A(t_A), B(t_B), C(t_C), D(t_D)$), where t_A,t_B, t_C,t_D represent the measurement start time (that is $t = t_A = t_B = t_C = t_D$ only for parallel scheduling in Fig. 15).

[7]In case measurements have conditional execution, considerations developed in this section still hold, but this would unnecessarily lead to a more cumbersome notation.

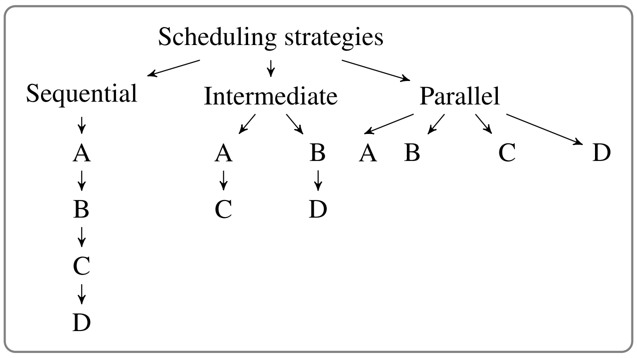


Figure 15: Illustration of the measurement scheduling tradeoff

Clearly, in the case of sequential scheduling, the network conditions can potentially change during the measurement period (or the phenomena causing performance degradations can possibly vanish, depending on the length of the chain). This also implies that correlation between any pair of measurements $X, Y \in \{A, B, C, D\}$ may weaken, making the detection problem harder: if $X(t)$ and $Y(t)$ are correlated at time t, it does not mean that they are necessarily correlated at times t_X and t_Y.

This becomes especially problematic if measurement A, B, C, D are carried from different probes, implying that a new HTTP(S) connection has to be established (and a TLS handshake performed). To reduce unnecessary delay $\|t_X - t_Y\|$ between any pair of measurements $X, Y \in \{A, B, C, D\}$, and of the whole measurement chain, it would be desirable to opportunistically establish connections in parallel at the root of the tree, and then sequentially schedule measurements over these established connections. Consequently, the time delay between a pair of consecutive measurements would be bound to the duration of the measurement itself (which is generally either known, deterministic and tunable, or can be statistically bound). This holds unless an agent is already performing a measurement task, in which case in our current action management policy the new incoming measurement request will be queued up by the agent (recall Fig.6).

7.4. Interaction of homogeneous/heterogeneous measurements

When measuring any given metric of interest, *precision* and *accuracy* are intrinsic to each tool, and can limit the usefulness of the measurement, or even possibly lead to misinformation in case of a large bias. More importantly, there may be *side effects* for the measurement tools, that are possibly *well-known* and thus avoidable or *hidden* and thus more insidious. An example of a well-known effect is represented by delay measurement under load, exploited to infer buffer size and assess the extent of bufferbloat [3, 36, 38]; an example of hidden effect is represented by mutual interference of simultaneous measurements of a bottleneck link as early noted in this work and [40].

At the same time, general guidelines are hard to precisely formalize here – notice indeed that concurrent measurements do not necessarily imply interference. For instance, consider the example in Fig. 16 where host i is scheduling parallel bandwidth measurements to hosts j and k (i.e., $Bw(i, j)$ and $Bw(i, k)$),

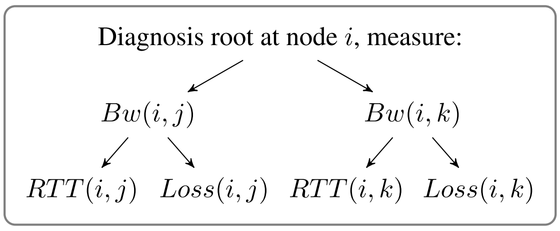


Figure 16: Interaction of homogeneous measurements: $Bw(i, j)$ and $Bw(i, k)$ may mutually interfere

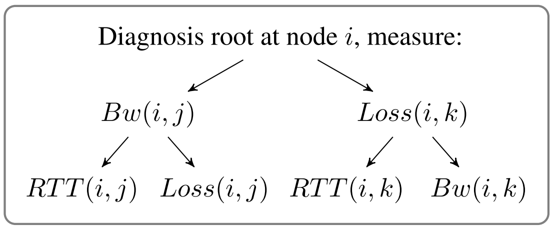


Figure 17: Interaction of heterogeneous measurements: $RTT(i, j)$ may be affected by $Bw(i, k)$

and subsequently scheduling parallel measurements of RTT and losses to the same hosts (*RTT(i,j)*, *RTT(i,k)*, *Loss(i,j)*, *Loss(i,k)*). Clearly, measurements $Bw(i, j)$ and $Bw(i, k)$ will interact if i is using the same physical interface for both measurements (in case a multi-homed host i probes j, k over unrelated interfaces such as 3G and WiFi, this would not cause interference). Additionally, the bottleneck towards $Bw(i, j)$ and $Bw(i, k)$ should be located in the path segment common to both i, j and i, k pairs (which may not be the case when per-flow load balancing techniques are used,etc.). Clearly this is even more challenging since any apparently unrelated measurement $Bw(i, j)$ and $Bw(m, n)$ may interfere when data paths i, j and m, n share a bottleneck link.

As such, while it is known that measurements may interfere, and that thus it would be good practice to reduce their potential interference by scheduling them in series, at the same time it would be perfectly legitimate to schedule them in parallel in the case of a non shared bottleneck, which is hard to assess in the general case. It follows that the problem of distributed measurement scheduling cannot be solved in the general instance in completely distributed settings. Without managing fine-grained scheduling, an ALTO-like interface could facilitate at coarse grained, keeping track of active measurements sharing common resources between peer pairs. Such a server would not of course take into account dynamic components (that will be hard to assess in real time), but it would be easy to exploit static topological information to limit the number of concurrent measurement that crosses the same path.

Morever, even heterogeneous measurement may interfere as illustrated in Fig. 17: $RTT(i, k)$ may be affected by $Bw(i, k)$, depending on the technique employed to measure $Bw(i, k)$. Additionally, in case the bottleneck is located in a segment common to both the i, j and i, k paths, then bandwidth measurement

$Bw(i,j)$ can possibly affect $Loss(i,k)$ when the probing rate is too high with respect to the available bandwidth, and provided that the buffer fills up during the measurement timescale (similar considerations hold for $Bw(i,k)$ vs $RTT(i,j)$ and $Loss(i,j)$). Exposing interference between measurements allows to arrange scheduling trees that, at least for a NetProbe agent, minimizes the interference. Additionally, as in the previous section, it is possible to select tools that are less intrusive, to reduce the interference, though this may lead to loss of accuracy for some measurements.

Assessing the combination of tools, and the selected scheduling of measurements, that will not cause interference while retaining sufficient accuracy to meet the analysis objective, requires rigorous calibration and validation — which is not different than the methodology outlined in this work, and yet requires a considerably larger experimental effort to make such a system usable in operational networks.

8. Related work

From a very high-level viewpoint, network troubleshooting efforts [1, 7, 2, 3, 4, 42, 43, 12, 13, 18] can be roughly partitioned into work having a more practical [1, 7, 2, 3, 4, 42] or theoretic [43, 12, 13, 18] flavor. While most work, including ours, uses active measurements [1, 2, 3, 4, 43, 12, 13, 18], there are exceptions that either use passive measurements [7] or have access to a multitude of logs [42]. In terms of network segment, previous work focuses on home/ access [1, 7, 4], enterprise [42] or backbone [43, 6, 44] networks. Some studies do not target a network segment in particular [12, 13, 18] and remain at a more abstract level. In this paper, we focus on home and access networks.

Our methodology is based solely on end-to-end measurements to localize the set of links that are the most likely root cause of performance degradations. Closest to our work with this regard [43, 12, 13, 1]is the large body of work in network tomography which exploits end-to-end measurement from a source to multiple receivers due to common paths to infer properties of internal network links such as network outages [43], delays [12], and packet losses [13]. However, these studies make simplifying assumptions that do not hold in real deployments [6, 45] such as the use of multicast [12]. In addition, the proposed algorithms are computationally expensive even for networks of reasonable scale and their accuracy is affected by the scale and the topology of the network [6]. Conversely, more practical work such as [1] that leverages, as we do in this work, application-layer multicast is more focused on the design of the system as opposite to its performance evaluation, which remains anecdotal at best [46, 1].

In this work, we take the best of both worlds by presenting a practical, general framework to *identify* faulty links that we study from both an analytical as well as from an experimental perspective. We instantiate such framework on two specific metrics: delays (as in [12]) and bottleneck bandwidth (which is notoriously more difficult to measure). When full topological information is available (e.g., in a ISP-managed case), our algorithm identifies the root cause (as in classic tomography). In case topology information is unavailable (e.g., in the Internet in general), our algorithm limitedly performs a clustering of measurement probes (as in binary network tomography [5]), where the inference problem is simplified by separating links (in our case probes) into good vs failed ones.

Additionally, one major limit of the related literature is the realism of ground truth data to evaluate the accuracy of the algorithms. Even in practical approaches, ground truth in the form of user tickets [42] or user feedback [7] is extremely rare, so that the absence of ground truth is commonplace [1, 2, 3, 4]. Theoretic work builds ground truth with simulations [13], or using syslogs and SNMP data in operational networks [43]. On the one hand, although simulations simplify the control over failure location and duration, they do not provide realistic settings. On the other hand, the ground truth is either completely missing in real operational networks (such as PlanetLab [5]) or partially missing in testbeds [43, 45], where network events outside of the control of researchers can happen. For instance, [45] evaluates several existing network tomography techniques on the VINI testbed [47]: the evaluation results show that techniques under study possibly detect faults that *are* not injected in the study as part of the controlled faults – which happens because the emulated links in VINI are part of paths that traverse the wide-area network, and show once more the difficulty to generate a reliable ground truth in realistic settings.

To cope with this, our setup employs controlled emulation through Mininet [9] which is (relatively) fast to implement, uses real code (including kernel stack and our software), and allows testing on fairly large scale topologies. This setup allows full control on the number, duration, and location of network problems. Additionally, by running the full network stack, Mininet keeps the real world noise in the underlying measurements, thus providing a more challenging validation environment with respect to simulation. This allows to retain control over the experiments, while adding realism tied to the use of the real protocol stack. As a side effect of this choice, the NetProbes software that we release as open source [10] has also undergone a significant amount of experimental validation. Most importantly, any peer researcher is capable of repeating our experiments in order to validate our results, compare their approach to ours, and extend this work.

With respect to our own preliminary work [8], in this paper we not only more thoroughly describe our framework and its application scope, but also present an extensive set of analytical (notably, we contrast the original model [8] with the exact Hypergeometric solution and its Poisson approximation), and experimental results (e.g., in terms of both an extended set of calibration as well as probe selection schemes). However, aside of results of punctual experiments that are discussed in this extended version, we believe that our methodological contribution, as well as the NetProbe software we release, is a more important contribution, as it empowers peer researchers with tools to not only replicate our results, but also to enlarge the boundaries of the present investigation (e.g., by comparing their approach to ours in a different set of scenarios).

9. Conclusions

This paper proposes, models, implements and experimentally evaluates an algorithm for network troubleshooting. The algorithm we implement in NetProbe is able to diagnose network performance disruptions in the home and access networks, in terms of clustering or classification. The algorithm is also flexible as it decouples the measurement from the inference process, making it suitable to cover a wide range of troubleshooting cases. We follow an experimental approach and use an emulated environment based on Mininet to validate our proposed algorithm. Our choice of Mininet is guided by our requirements to have flexibility in designing the experiments and full control over the injected faults. We perform a thorough calibration of the emulation environment, an often neglected but necessary step to achieve scientifically sound results.

Provided that measurement tools are not biases, our clustering methodology is very reliable in evaluating the severity of the performance issue. In case of ISP-assistance via simple strategic probe selection, our classification methodology further achieves perfect classification, correctly identifying the root cause links when the network topology is known. We further contrast the experimental results with simple analytical model(s) that compute the expected correct detection probability under a random probe selection strategy, with good accuracy. We also evaluate the impact of topology, probe budget and various probe selection strategies on our algorithm.

Our proposed solution is a first step towards the goal of having reproducible network troubleshooting algorithm, for which we make all the code we used in this paper publicly available to the scientific community.

Acknowledgement

This work has been carried out at LINCS `http://www.lincs.fr`. The research leading to these results has received funding from the European Union under the FP7 Grant Agreement n. 318627.

References

[1] K. Kim, H. Nam, V. K. Singh, D. Song, H. Schulzrinne, DYSWIS: crowdsourcing a home network diagnosis, in: Proc. of International Conference on Computer Communications and Networks (ICCCN), 2014.

[2] M. Dhawan, J. Samuel, R. Teixeira, C. Kreibich, M. Allman, N. Weaver, V. Paxson, Fathom: A browser-based network measurement platform, in: Proc. of ACM IMC, 2012.

[3] C. Kreibich, N. Weaver, B. Nechaev, V. Paxson, Netalyzr: Illuminating the edge network, in: Proc. of ACM IMC, 2010.

[4] Z. Bischof, J. Otto, M. Sánchez, J. Rula, D. Choffnes, F. Bustamante, Crowdsourcing ISP characterization to the network edge, in: Proc. of SIGCOMM WMUST, 2011.

[5] H. X. Nguyen, P. Thiran, The boolean solution to the congested IP link location problem: Theory and practice, in: Proc. of IEEE INFOCOM, 2007.

[6] D. Ghita, C. Karakus, K. J. Argyraki, P. Thiran, Shifting network tomography toward a practical goal, in: Proc. of ACM CoNEXT, 2011.

[7] D. Joumblatt, R. Teixeira, J. Chandrashekar, N. Taft, Hostview: Annotating end-host performance measurements with user feedback, ACM SIGMETRICS Performance Evaluation Review 38 (3) (2011) 43–48.

[8] F. Espinet, D. Joumblatt, D. Rossi, Zen and the art of network troubleshooting: a hands on experimental study, in: Proc. of Traffic Meaurement and Analysis (TMA), 2015.

[9] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, N. McKeown, Reproducible network experiments using container-based emulation, in: Proc. of ACM CoNEXT, 2012.

[10] NetProbes, `https://github.com/netixx/NetProbes`.

[11] Emulator scripts, `https://github.com/netixx/mininet-NetProbes`.

[12] F. L. Presti, N. G. Duffield, J. Horowitz, D. F. Towsley, Multicast-based inference of network-internal delay distributions, IEEE/ACM Transactions on Networking 10 (6) (2002) 761–775.

[13] N. G. Duffield, F. L. Presti, V. Paxson, D. F. Towsley, Network loss tomography using striped unicast probes, IEEE/ACM Transactions on Networking 14 (4) (2006) 697–710.

[14] [link].
URL `http://www.caida.org`

[15] S. Sundaresan, N. Feamster, R. Teixeira, Locating throughput bottlenecks in home networks, Proc. of ACM SIGCOMM, Demo Session, 2014, pp. 351–352.

[16] L. DiCioccio, R. Teixeira, M. May, C. Kreibich, Probe and pray: Using upnp for home network measurements, in: Proc. of Passive and Active Measurement (PAM), 2012, pp. 96–105.

[17] [link].
URL `http://www.apisense.com/`

[18] N. G. Duffield, J. Horowitz, F. Lo Presti, D. Towsley, Multicast topology inference from measured end-to-end loss, IEEE Transactions on Information Theory 48 (1) (2002) 26–45.

[19] T. Bu, N. Duffield, F. L. Presti, D. Towsley, Network tomography on general topologies, in: Proc. of ACM SIGMETRICS, 2002.

[20] B. Donnet, M. Luckie, P. Mérindol, J.-J. Pansiot, Revealing mpls tunnels obscured from traceroute, ACM SIGCOMM Computer Communication Review 42 (2) (2012) 87–93.

[21] N. Spring, R. Mahajan, D. Wetherall, Measuring isp topologies with rocketfuel, in: ACM SIGCOMM Computer Communication Review, Vol. 32, ACM, 2002, pp. 133–145.

[22] P. Marchetta, V. Persico, A. Pescapé, E. Katz-Bassett, Don't trust traceroute (completely), Proc. of CoNEXT Student Workhop, 2013.

[23] C. Pelsser, L. Cittadini, S. Vissicchio, R. Bush, From paris to tokyo: On the suitability of ping to measure latency, in: Proc. of ACM IMC, 2013.

[24] [link].
URL `https://datatracker.ietf.org/wg/alto/charter/`

[25] J. Seedorf, S. Kiesel, M. Stiemerling, Traffic localization for p2p-applications: the alto approach, in: IEEE Conference on Peer-to-Peer Computing (P2P), 2009.

[26] C. Zhang, P. Dhungel, D. Wu, K. W. Ross, Unraveling the bittorrent ecosystem, IEEE Transactions on Parallel and Distributed Systems, 22 (7) (2011) 1164–1177.

[27] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, S. Shenker, Less Pain, Most of the Gain: Incrementally Deployable ICN, in: Proc. of ACM SIGCOMM, 2013.

[28] I. S. Gradshteyn, I. M. Ryzhik, Table of Integrals, Series and Products, Academic Press, 2000.

[29] V. Paxson, Keynote: Reflections on measurement research: Crooked lines, straight lines, and moneyshots, in: Proc. of ACM SIGCOMM, 2011.

[30] E. Goldoni, M. Schivi, End-to-end available bandwidth estimation tools, an experimental comparison, in: Proc. of Traffic Meaurement and Analysis (TMA), 2010.

[31] J. Navratil, R. L. Cottrell, Abwe: A practical approach to available bandwidth estimation, in: Proc. of Passive and Active Measurement (PAM), 2003.

[32] E. Goldoni, G. Rossi, A. Torelli, Assolo, a new method for available bandwidth estimation, in: Proc. of International Conference on Internet Monitoring and Protection, (ICIMP), 2009, pp. 130–136.

[33] N. Hu, P. Steenkiste, Evaluation and characterization of available bandwidth probing techniques, IEEE J. Selected Areas in Communications 21 (6) (2003) 879–894.

[34] [link].
URL `https://iperf.fr/`

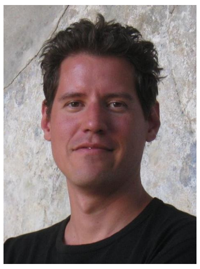
[35] J. Strauss, D. Katabi, F. Kaashoek, A measurement study of available

bandwidth estimation tools, in: Proc. of ACM IMC, 2003, pp. 39–44.
[36] H. Jiang, Y. Wang, K. Lee, I. Rhee, Tackling bufferbloat in 3G/4G networks, in: Proc. of ACM IMC, 2012, pp. 329–342.
[37] V. Cerf, V. Jacobson, N. Weaver, J. Gettys, Bufferbloat: what's wrong with the internet?, Communications of the ACM 55 (2) (2012) 40–47.
[38] C. Chirichella, D. Rossi, To the moon and back: are internet bufferbloat delays really that large, in: Proc. of Traffic Meaurement and Analysis (TMA), 2013.
[39] M. Halkidi, Y. Batistakis, M. Vazirgiannis, On clustering validation techniques, Journal of Intelligent Information Systems 17 (2-3) (2001) 107–145.
[40] D. Croce, M. Mellia, E. Leonardi, The quest for bandwidth estimation techniques for large-scale distributed systems, ACM SIGMETRICS Performance Evaluation Review 37 (3) (2010) 20–25.
[41] B. Gregg, Thinking methodically about performance, Communications of the ACM 56 (2) (2013) 45–51.
[42] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, M. Zhang, Towards highly reliable enterprise network services via inference of multi-level dependencies, in: Proc. of ACM SIGCOMM, 2007.
[43] R. Kompella, J. Yates, A. Greenberg, A. Snoeren, Detection and localization of network black holes, in: Proc. of IEEE INFOCOM, 2007.
[44] A. Dhamdhere, R. Teixeira, C. Dovrolis, C. Diot, Netdiagnoser: Troubleshooting network unreachabilities using end-to-end probes and routing data, in: Proc. of ACM CoNEXT, 2007.
[45] Y. Huang, N. Feamster, R. Teixeira, Practical issues with using network tomography for fault diagnosis, ACM SIGCOMM Computer Communication Review 38 (5) (2008) 53–58.
[46] V. K. Singh, H. Schulzrinne, K. Miao, Dyswis: An architecture for automated diagnosis of networks, in: Proc. of IEEE/IFIP Network Operations and Management Symposium (NOMS), 2008, pp. 851–854.
[47] A. Bavier, N. Feamster, M. Huang, L. Peterson, J. Rexford, In vini veritas: Realistic and controlled network experimentation, in: Proc. of ACM SIGCOMM, 2006.

Vitae

Francois Espinet received a BSc diploma for Ecole Polytechnique and is currently a MSc student at Telecom ParisTech. This work has been carried out as part of his research stage work under the supervision of Prof. Dario Rossi.

Diana Zeaiter Joumblatt received her BSc in Computer Science from the American University of Beirut (2004), and her MSc and PhD in Computer Science, both from Université Pierre et Marie Curie (UPMC), in 2008 and 2012 respecively. During 2012-2014, she held a PostDoc position at Telecom ParisTech.



Dario Rossi is a Professor at Telecom ParisTech and Ecole Polytechnique. He received his MSc and PhD degrees from Politecnico di Torino in 2001 and 2005 respectively, and his HDR degree from Université Pierre et Marie Curie (UPMC) in 2010. During 2003-2004, he held a visiting researcher position in the Computer Science division at University of California, Berkeley. He has coauthored 9 patents and over 150 papers in leading conferences and journals, that received 3 best paper awards, an IETF Applied Network Research Prize (2016) a Google Faculty Research Award (2015). He participated in the program committees of over 50 conferences including ACM ICN, ACM CoNEXT, IEEE INFOCOM of which he was also Distinguished Member (2015,2016). His current research interests include Internet traffic measurement, Information centric networks and high speed networking. He is a Senior Member of IEEE and ACM.