Zen and the Art of Network Troubleshooting: a Hands on Experimental Study

François Espinet[1], Diana Joumblatt[2], Dario Rossi[1,2]

[1]Ecole Polytechnique - `first.last@polytechnique.edu`
[2]Telecom ParisTech - `first.last@enst.fr`

Abstract. Growing network complexity necessitates tools and methodologies to automate network troubleshooting. In this paper, we follow a crowd-sourcing trend, and argue for the need to deploy measurement probes at end-user devices and gateways, which can be under the control of the users or the ISP.

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, that we solve with an algorithm that (i) achieves perfect classification under the assumption of a strategic selection of probes (e.g., assisted by an ISP) and (ii) operates blindly with respect to the network performance metrics, of which we consider delay and bandwidth in this paper.

While previous work on network troubleshooting privileges a more theoretical vs practical approaches, our workflow balances both aspects as (i) we conduct a set of controlled experiments with a rigorous and reproducible methodology, (ii) on an emulator that we thoroughly calibrate, (iii) contrasting experimental results affected by real-world noise with expected results from a probabilistic model.

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. 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).

Currently, troubleshooting performance disruptions is complex and ad hoc due to the presence of different applications, network protocols, and administrative domains. Typically, troubleshooting 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, which possibly includes the home network of the very same user – hence, for the ISP, it would be valuable to extend its reach beyond the home gateway by instrumenting experiments directly from end-user devices. While (tech savvy) users can be assisted in their troubleshooting efforts by software tools such as [4, 6, 17, 19] which automate a number of useful measurements, these tools do not incorporate network tomography techniques [9, 21] to identify the root causes of network disruptions (e.g., faulty links). Additionally, these tools are generally ISP network-agnostic – hence, they would benefit from cooperation with the ISP.

In this paper, we propose a practical methodology to automate the identification of faulty links in the access network based on end-to-end measurements. Since the devices participating in the troubleshooting task can be either under the control of the end-user or the ISP, the knowledge of the ISP topology is not always available for the measurement probes. Consequently, we formalize the troubleshooting task as either a *clustering* or a *classification* problem – where respectively end-users are able to assess the severity of the fault, or ISPs are able to identify the faulty link.

This paper makes several contributions. While our troubleshooting *model* (Sec. 3), *algorithm* (Sec. 4) and *software implementation* (Sec. 5) are interesting per se, we believe our major contribution is the rigour of the *evaluation* methodology (Sec. 6), which overcomes state of the art limits (Sec. 2). Indeed, on one hand, previous practical troubleshooting efforts [4, 6, 16, 17, 19] are valuable in terms of domain knowledge and engineering, but lack theoretical foundations and rigorous verification. On the other hand, prior analytical efforts are cast on solid theoretic ground [9, 21], but their validation is either simplistic (e.g., simulations) or lacks ground truth (e.g., PlanetLab).

In this work, we take the best of both worlds, as we (i) propose a practical methodology for network troubleshooting with an open source implementation; (ii) provide a model of the expected fault detection probability that we contrast with experimental results; (iii) use an experimental approach where we emulate controlled network conditions with Mininet [13]; (iv) perform a calibration of the emulation setup, an often neglected albeit mandatory task; (v) in spirit with Mininet and the TMA community, we further make all our source code available for the scientific community at [1, 2].

2 Related work

Our work complements prior network troubleshooting efforts [3, 4, 6–8, 16–19, 23] that we overview in this section. Without attempting at a full-blown taxonomy, we may divide the above work as having a more *practical* [3, 4, 6, 16, 17, 19] or *theoretic* [7, 8, 18, 23] approach. While most work, including ours, uses active measurements [4, 6–8, 17–19, 23], there are exceptions that use passive measurements [16] or logs [3]. In terms of network segment, previous work focuses on home networks [17], enterprise networks [3], and backbone networks [5, 9, 18]. Some studies do not target a network segment in particular [7, 8, 23] 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 is the large body of work in network tomography which exploits the similarity of end-to-end network performance from a source to multiple receivers due to common paths to infer properties of internal network links such as network outages [18], delays [23], and packet losses [8]. However, these studies make simplifying assumptions that do not hold in real deployments [9, 15] such as the use of multicast [23]. In addition, the proposed algorithms are computationally expensive for networks of reasonable scale and their accuracy is affected by the scale and the topology of the network [9]. In this work, we instead present a practical, general framework to *identify* faulty links that we instantiate on two specific metrics: delays as in [23] and bottleneck bandwidth,

which is notoriously more difficult to measure. When full topological information is not available, our algorithm performs a *clustering* of measurement probes as in binary network tomography [21], where the inference problem is simplified by separating links (in our case probes) into good vs failed, instead of estimating the values of the link performance metrics.

Additionally, one major problem 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 [3] or user feedback [16] is extremely rare, so that the absence of ground truth is commonplace [4, 6, 17, 19]. Theoretic work builds ground truth with simulations [8], or using syslogs and SNMP data in operational networks [18]. One 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 [21]) or partially missing in testbeds [15, 18], where network events outside of the control of researchers can happen. Our setup employs controlled emulation through Mininet [13] 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. As a side effect of this choice, the NetProbes software that we release as open source [2] 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.

3   Problem statement and model

Considering an ISP network, and focusing for the sake of simplicity on its access tree, faults can occur at multiple levels in the access network hierarchy. The ability to launch measurements between arbitrary pairs of devices in the same access network would significantly enhance the diagnosis of network performance disruptions. In this work we consider two use-cases: *User-managed* probes and *ISP-managed* probes. 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). We address both use-cases with the same algorithm: clustering in the user-scenario separates measurement probes into two sets (i.e., un/affected sets), whereas an additional mapping in the ISP-scenario allows to pinpoint the root cause link.

We formalize the problem and introduce the notation used in this paper with the help of Fig. 1, which depicts a binary access network tree. The troubleshooting probe software runs in the leaf nodes of the tree. However, the ISP can strategically place probes inside the network (e.g., probe 0 in the picture attached to the root). Our algorithm runs continuously in the background to gather a baseline of network performance, and troubleshooting is triggered by the user (e.g., upon experiencing a degradation of
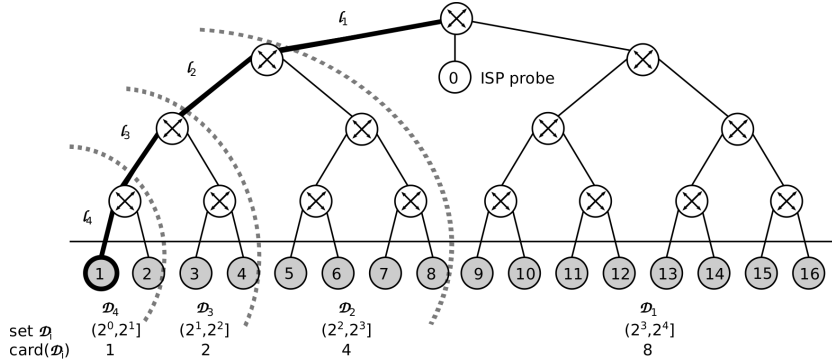
Fig. 1. Synoptic of the network scenario and model notation

network performance) or automatically by a change point detection procedure on some relevant metrics (outside the scope of this work).

For the sake of clarity, let us assume that probe 1 launches a troubleshooting task. In this context, we can safely assume that the root cause is located somewhere in the path from the user device or gateway towards the Internet (links $\ell_4, \ell_3, \ell_2, \ell_1$ in bold in Fig. 1). In order to identify which among $\ell_4, .., \ell_1$ is the root cause of the fault, probe 1 requires sending probing traffic to a number M of the overall available probes N. 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 $\mathcal{D}_i = \left(k^{D^+-i}, k^{D^+-i+1} \right]$. The set $\mathcal{D}_i$ includes probes whose shortest path from probe 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 probes from both sets $\mathcal{D}_f$ and $\mathcal{D}_{f+1}$ to infer with certainty that the fault is located at ℓ_f. For 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 [24] server). 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 naive random selection.

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 $\alpha = M/N$. 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 ($card(\mathcal{D}_f) = k^{D^+-f}$), 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 $\mathcal{D}_f$. 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., $(1-\alpha)^{card(\mathcal{D}_f)}$. Consequently, the probability to sample at least[1] one probe in $\mathcal{D}_f$ is:

$$p^-(f,\alpha) = 1 - (1-\alpha)^{k^{(D^++-f)}} \tag{1}$$

Expression (1) is a lower bound on the expected detection probability with random selection. 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 $k^d/k^{log_k(N)} = k^d/N$,

$$\mathbb{E}[p^{guess}] = \sum_{d=1}^{log_k(N)} \frac{1}{d}\frac{k^d}{N} = \frac{1}{N}\sum_{d=1}^{log_k(N)} \frac{k^d}{d} \tag{2}$$

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:

$$\mathbb{E}[p] = p^-(f,\alpha) + \big(1 - p^-(f,\alpha)\big)\mathbb{E}[p^{guess}] \tag{3}$$

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 $\mathbb{E}[p^{guess}]$). By plugging (1) and (2) into (3) we get:

$$\mathbb{E}[p] = 1 - (1-\alpha)^{k^{(D^+-f)}} + \big[1 - (1 - (1-\alpha)^{k^{(D^+-f)}})\big]\Big(\frac{1}{N}\sum_{d=1}^{log_k(N)} \frac{k^d}{d}\Big) \tag{4}$$

$$= 1 - (1-\alpha)^{k^{(D^+-f)}}\Big(1 - \frac{1}{N}\sum_{d=1}^{log_k(N)} \frac{k^d}{d}\Big) \tag{5}$$

Notice that (5) has structurally the form $1 - p_{loss}$. The term p_{loss} can be interpreted as the loss of discriminative power with respect to a perfect strategic selection that always achieves correct detection. 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.

---

[1] Note that this probability would be better expressed with the hypergeometric distribution, that models sampling without replacement; however the formulation reported here differ by less than 1% from the hypergeometric results, and further allows to express the loss of discriminative power due to random selection in a more intuitive way.

4 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. When the troubleshooting is triggered, s iteratively selects up to R batches of B of probes, so that $R \cdot B$ represents a tuneable 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 $S(p)$ can vary, and thus the next batch is selected based on the results of the previous batch.

At each step, upon doing B measurements, we compute, for each probe p, $Q(p) - Q_0(p)$ and add it to the set P: K-means clustering partitions P into P^+ and P^-. Two points are worth stressing: first, the algorithm does not associate any semantic to clusters: e.g., a node in P^+ can be affected by large delay, whereas a node in P^- can be affected by a bottleneck bandwidth. Second, in case of a single failure, it can be expected that probes in one of the two clusters exhibit $Q(p) - Q_0(p) \approx 0$, 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.

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 $p \in P^-$. As a consequence of metric-agnosis, the algorithm needs to know if links with the largest (smallest) $S(\ell)$ scores are to be pinpointed, which is done according to a link selection policy $\mathcal{S}_\ell$.

We experiment with $\mathcal{S}_p \in \{\text{random}, |IP(s) - IP(p)|, \text{balance}\}$ and combinations of the above. Random selection is useful as a baseline and to compare with the model. We additionally consider probe selection policies that are more complex to model such as the absolute distance in the IP space, as well as a policy that attempts at equating 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 (exact definition omitted due to lack of space). Moreover, we consider $\mathcal{S}_\ell \in \{\text{random}, \text{proportional}, \text{argmax}\}$. 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 (success probability $1/D^+$, much larger than the $1/2(k^{D^+} - 1) = 1/2(N - 1)$ in case of a random guess over all links). We also select links proportionally to their score (proportional policy), or only the link with the largest (smallest) score (argmax policy).

5 Calibration of the emulation environment

Before running a full-fledged measurement campaign, it is mandatory to perform a rigorous calibration phase – yet this phase is often neglected [22]. 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 techniques perform in Mininet. In order to disambiguate inconsistencies due to Mininet from measurement errors intrinsic to measurements techniques,

Algorithm 1 Detection algorithm at s

1: Get a baseline $Q_0(p)$ for metric $Q(p)$, $\forall p$ ▷ Initialization, over long timescale
2: **for** round $\in$ [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 ℓ according to a link selection strategy $\mathcal{S}_\ell$ based on scores $S(\ell)$
20: **end if**

---

we perform calibration experiments for a set of delay (expectedly easy) and bandwidth (notoriously difficult) measurement tools and assess their accuracy in Mininet. In this section, we first briefly describe Mininet and NetProbes, the diagnosis software we develop for this work (Sec. 5.1), then present the calibration results (Sec. 5.2).

5.1 Software Tools

Mininet [13] Mininet 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 [2] indicates that results from Mininet and from testbeds are in agreement.

NetProbes [2] We design NetProbes, a distributed software written in Python 3.x that runs on end-hosts and executes a set of user-defined active measurement tests. Net-Probes agents 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

---

[2] Stanford's CS224 blog: http://reproducingnetworkresearch.wordpress.com

baseline network performance. When the user experiences network performance issues, the NetProbes agent running at the user device launches a troubleshooting task to assess the severity of the performance issue and the location of the faulty link. 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, and that the software is available at [2].

5.2 Delay and bandwidth calibration

Setup We build a Mininet virtual network with the topology depicted in Fig. 1 on a server with four cores and 24 GB of RAM. We run the selected tools on probes 1 and 2. In our delay experiments, we impose five different delay values (0 ms, 20 ms, 100 ms, 200 ms, 1000 ms) 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) and we make 30 measurements of the available bandwidth between probes 1 and 7 and probes 2 and 6 (270 in total for each link capacity value). There is a plethora of measurement tools designed by the research community to estimate the available bandwidth [11]. In this work we limitedly report the calibration of three popular tools (Abing [20], ASSOLO [10], and IGI [14]) which are characterised by low intrusiveness: Abing and IGI infer the available bandwidth based on the dispersion of packet pairs measured at the receiver. ASSOLO sends a variable bit-rate stream with exponentially spaced packets and calculates the available bandwidth from the delays at the receiver side. We compare the performance of the three bandwidth estimation tools in the absence of cross traffic and under the three traffic shapers mentioned earlier.

Delay 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 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. Further results are shown in Fig. 2. All techniques exhibit a time evolution similar to ICMP `ping` whose experiment is depicted in Fig. 2(a). We report the PDF of the measurement error (i.e., the difference between the measured and the enforced RTT) in Fig. 2(b). Results for `traceroute` with various protocols are similar: we observe that, for all the delay measurement techniques, the bulk of the error distribution is less than 1 ms (with outliers not shown up to 10ms). Moreover, we note that using ICMP brings the absolute
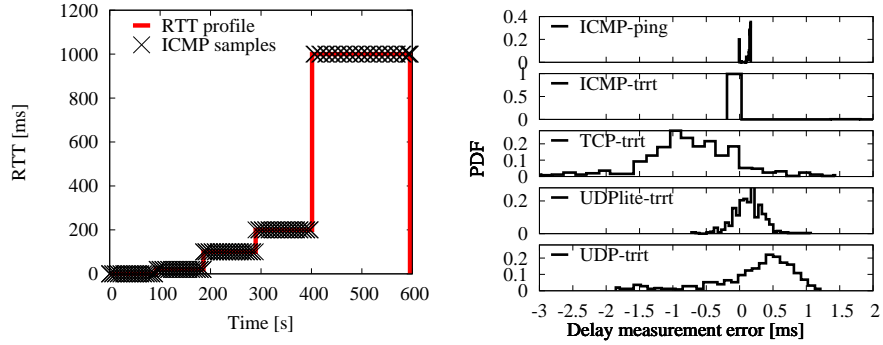
(a) Time evolution of RTTs with ICMP Ping　　　(b) PDF of measurement errors

Fig. 2. Calibration of delay measurements

error to less than 0.1 ms for both `traceroute` and `ping`. From this calibration phase, we select ICMP `ping` to measure delay: as the measurement noise is insignificant, errors in the classification outcome should be solely attributed to our troubleshooting algorithm.

Capacity Fig. 3 reports the evolution of the estimated available bandwidth as a function of three link capacity values for the cross product of {Abing, ASSOLO, IGI} $\times$ {HTB, TBF, HFSC}. We stress that while comparison of bandwidth estimation tools under the same experimental conditions has already been studied, we are not aware of any study jointly considering bandwidth estimation and bandwidth shaping, especially since many bandwidth measurement tools rely on effects of cross-trafic to estimate available bandwidth. We can see that Abing systematically fails in estimating the available bandwidth under HTB and TBF shaping, while the estimation is correct with HFSC. Similarly, AS-SOLO 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. These results and tradeoffs are interesting and require future attention. However, this is beyond the scope of this work. The most important takeaway is that measurement errors of such magnitude would invalidate all experiments, showing once more the importance of this calibration phase. We additionally gather that the IGI+HFSC combination offers the most accurate estimates of available bandwidth. As accurate input is a necessary condition for trobuleshooting success, we use this combination in the remainder of this paper.

6　Experimental results

We now evaluate the quality of our clustering and classification for various probe budgets (namely 10, 20 and 50 probes) for faults (e.g., doubling delay or halving band-
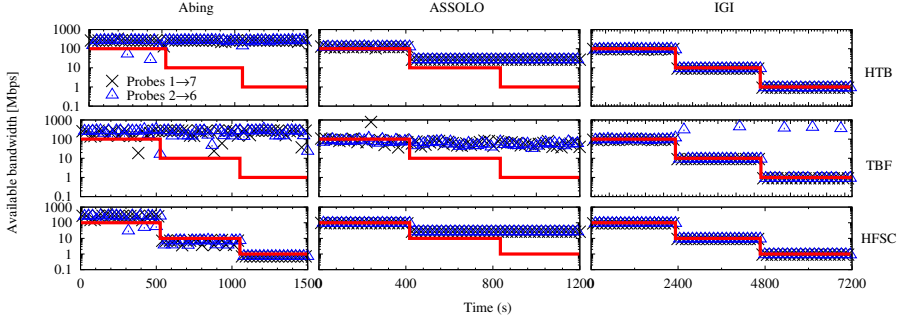
Fig. 3. Calibration of bandwidth measurements: {Abing, ASSOLO, IGI} × {HTB, TBF, HFSC}

width) at controlled depths of the tree. All the scripts to reproduce the experiments are available at [1]. We first compare experimental results in a calibrated Mininet environment (including real-world noise), with those expected by a probabilistic model (neglecting noise) (Sec. 6.1). We next perform a sensitivity analysis by varying topological properties, probe selection policies $\mathcal{S}_p$, and link selection policies $\mathcal{S}_\ell$ (Sec. 6.2).

6.1 Performance at a glance

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 would need $M/N = 9/512$ probes ($\alpha = 1.75\%$) to ensure perfect classification, but we consider larger budget $M = \{10, 20, 50\}$ in our experiments. 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 our ground truth, using the well-known rand index [12], which takes value in $[0, 1] \subset \mathbb{R}$, with 1 indicating that the data clusters are exactly the same. 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. 4-(a) 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 measurement: we point out that the loss of accuracy is not tied to our algorithm, but rather to measurements that are input to it, which was partly expected and confirms that calibration is a necessary, but unfortunately not sufficient, step.

Abstracting from limits in the measurement techniques, these result indicates that in practice our clustering methodology works well in assessing the impact of a faulty link without requiring knowledge of the network topology. Yet, root cause link identi-
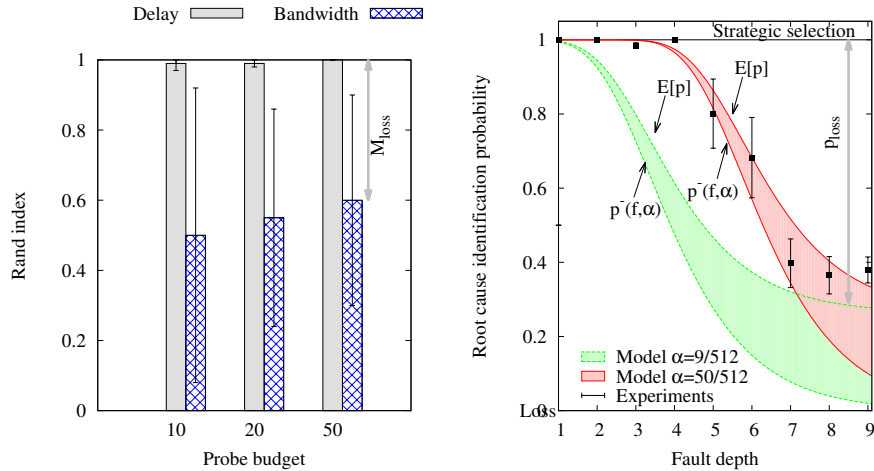
(a) Rand index of experimental output of clustering algorithm vs ground truth clusters

(b) Probability of correctly identifying the faulty link (models (1) and (3) vs experiments)

Fig. 4. Experimental results at a glance: (a) Clustering and (b) Classification

fication is a clearly more challenging and important objective, which we analyze in the following 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. We expect classification results to apply at large, as opposite to merely illustrating the algorithm performance under delay measurement (although they are not representative of bottleneck localization as per Fig. 4-(a)).

We next show that the experimental and modelling results are in agreement, with a random probe selection policy and a budget of $M = 50$ probes, which corresponds to $\alpha = 9.75\%$. For each fault depth f, we perform 10 experiments by randomizing the set of destination probes. Results, as reported in Fig. 4-(b), depict the correct classification probability of the model vs the experiments. Recall that equation (1) gives a lower bound $p^-(f, \alpha)$ to the experimental results, while (3) models the average expected detection probability $\mathbb{E}[p]$. We consider $\alpha = 9.75\%$, to directly compare with experimental results, as well as $\alpha = 1.75\%$, to assess the loss of discriminative power from a strategic selection, that could achieve perfect classification in this setting, to a random selection (denoted with p_{loss} in the figure).

6.2 Sensitivity analysis

Impact of topology We study the impact of the network topology on the classification performance. We use two trees with 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. 5 reports the correct detection probability of the faulty link as a function of
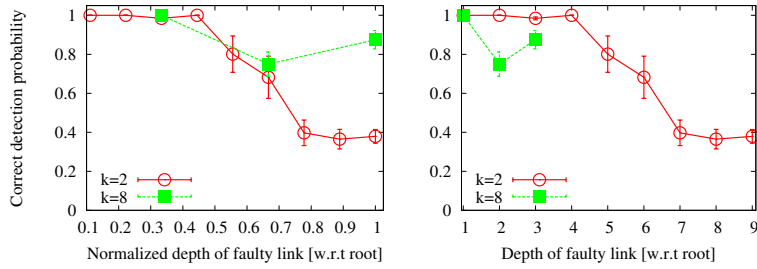
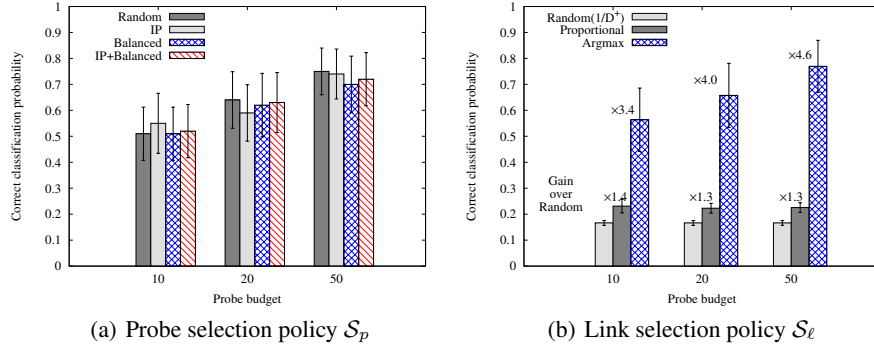**Fig. 5.** Sensitivity analysis: Impact of network topology properties

the depth of the injected fault in the tree. As expected, results indicate that the correct detection probability decreases as the fault depth increases. Thus, when the root cause link is located close to the *leaves* of the tree[3], it is harder to randomly sample another probe which is also affected by the fault: we thus need a smarter probe selection strategy to improve the link classification performance.

Impact of the probe selection policy $\mathcal{S}_p$ We consider policies based on IP-distance (IP), cluster-size (balance), and a linear combination of both. We average the results over all the depths of the binary tree and contrast them with a random selection policy. Unfortunately, our attempts are so far unsuccessful as shown in Fig. 6(a), where the discriminative power is roughly the same over all probe selection policies. This is due to the fact that the current set of metrics we consider to select probes do not encode useful information to bias the selection. The absence of a notion of net masks and hierarchy with IP-distance for example makes it hard to extract information about how topologically close/far probes are from each other. An obvious improvement would be to consider the IP-TTL field[4]. However, since Mininet uses virtual switches to construct the network, the IP-TTL field remains unchanged. As a consequence, we could not conduct experiments with this field and we leave it as future work.

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. 6. The plot is futher annotated with the gain factor over the random selection: while proportional selection brings a constant

---

[3] There is a special case worth mentioning concerning an ambiguity for link ℓ_1, i.e., close to the *root* of the tree for fanout $k = 2$: indeed, both links as all shortest path to affected probes pass through both root links, so that successful classification follows from conditioning over the set of links $\ell_{D+}, ..., \ell_1$ from the leaf to the root.

[4] We expect TTL to greatly assist the stratification in our scenario. While TTL could be learnt over time, its relevance in real operational networks may remain dubious due to the need to use raw network sockets, and its weak correlation with spatial location due the potential presence of middleboxes.

(a) Probe selection policy $\mathcal{S}_p$ (b) Link selection policy $\mathcal{S}_\ell$

Fig. 6. Sensitivity analysis: Impact of selection policies

improvement of about 40%, the argmax policy brings considerable gains (in excess of a factor 4) which grow with the probe budget.

7 Conclusions and future work

In this work, we present a troubleshooting algorithm to diagnose network performance disruptions in the home and access networks. We apply a clustering methodology to evaluate the severity of the performance issue and leverage the knowledge of the access network topology to identify the root cause link with a correct classification probability of 70% using 10% of the available probes. We follow an experimental approach and use an emulated environment based on Mininet to validate our algorithm. Our choice of Mininet is guided by our requirements to have flexibility in designing the experiments, full control over the injected faults, and realistic network settings. We contrast the experimental results with an analytical model that computes the expected correct classification probability under a random probe selection policy. We also evaluate the impact of topology, and probe and link selection policies on the algorithm.

Our proposed solution is a first step towards the goal of having reproducible network troubleshooting algorithms. We make all the code publicly available. Our future work will focus on finding meaningful probe selection policies to increase the detection probability of faulty links under probe budget constraints. Additionally, we intend to extend the algorithm to different network topologies and to diversify the set of network performance metrics.

Acknowledgement

References

1. Emulator scripts. `https://github.com/netixx/mininet-NetProbes`.
2. NetProbes. `https://github.com/netixx/NetProbes`.
3. P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *Proc. ACM SIGCOMM*, 2007.
4. Z. S. Bischof, J. S. Otto, M. A. Sánchez, J. P. Rula, D. R. Choffnes, and F. E. Bustamante. Crowdsourcing ISP characterization to the network edge. In *Proc. SIGCOMM WMUST*, 2011.
5. A. Dhamdhere, R. Teixeira, C. Dovrolis, and C. Diot. Netdiagnoser: Troubleshooting network unreachabilities using end-to-end probes and routing data. In *Proc. CoNEXT*, 2007.
6. M. Dhawan, J. Samuel, R. Teixeira, C. Kreibich, M. Allman, N. Weaver, and V. Paxson. Fathom: A browser-based network measurement platform. In *Proc. ACM IMC*, 2012.
7. N. G. Duffield, J. Horowitz, F. L. Presti, and D. F. Towsley. Multicast topology inference from measured end-to-end loss. *IEEE Transactions on Information Theory*, 2002.
8. N. G. Duffield, F. L. Presti, V. Paxson, and D. F. Towsley. Network loss tomography using striped unicast probes. *IEEE/ACM Trans. Netw.*, 2006.
9. D. Ghita, C. Karakus, K. J. Argyraki, and P. Thiran. Shifting network tomography toward a practical goal. In *Proc. CoNEXT*, 2011.
10. E. Goldoni, G. Rossi, and A. Torelli. Assolo, a new method for available bandwidth estimation. 2009.
11. E. Goldoni and M. Schivi. End-to-end available bandwidth estimation tools, an experimental comparison. In *Proc. TMA*, 2010.
12. M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2-3):107–145, 2001.
13. N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. Reproducible network experiments using container-based emulation. In *Proc. CoNEXT*, 2012.
14. N. Hu and P. Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *IEEE J. Selected Areas in Communications*, 2003.
15. Y. Huang, N. Feamster, and R. Teixeira. Practical issues with using network tomography for fault diagnosis. *ACM SIGCOMM Computer Communication Review*, 2008.
16. D. Joumblatt, R. Teixeira, J. Chandrashekar, and N. Taft. HostView: Annotating end-host performance measurements with user feedback. In *ACM HotMetrics Workshop*, 2010.
17. K. Kim, H. Nam, V. K. Singh, D. Song, and H. Schulzrinne. DYSWIS: crowdsourcing a home network diagnosis. In *ICCCN*, 2014.
18. R. Kompella, J. Yates, A. Greenberg, and A. Snoeren. Detection and localization of network black holes. In *Proc. IEEE INFOCOM*, 2007.
19. C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the edge network. In *Proc. ACM IMC*, 2010.
20. J. Navratil and R. L. Cottrell. Abwe: A practical approach to available bandwidth estimation. In *Proc. of PAM*, 2003.
21. H. X. Nguyen and P. Thiran. The boolean solution to the congested IP link location problem: Theory and practice. In *Proc. IEEE INFOCOM*, 2007.
22. V. Paxson. Keynote: Reflections on measurement research: Crooked lines, straight lines, and moneyshots. In *Proc. ACM SIGCOMM*, 2011.
23. F. L. Presti, N. G. Duffield, J. Horowitz, and D. F. Towsley. Multicast-based inference of network-internal delay distributions. *IEEE/ACM Trans. Netw.*, 2002.
24. J. Seedorf and E. Burger. Application-Layer Traffic Optimization (ALTO) Problem Statement. IETF RFC 5693, 2009.