Computer Networks 000 (2016) 1–19

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Framework, models and controlled experiments for 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

ARTICLE INFO

Article history:
Received 23 November 2015
Revised 22 March 2016
Accepted 1 June 2016
Available online xxx

Keywords:
Troubleshooting
Emulation
Modeling
Experiments
Root-cause analysis

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 making several interesting observation that help designing such an automated troubleshooting system.

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).

* Corresponding author.
E-mail addresses: francois.espinet@polytechnique.edu (F. Espinet), diana.joumblatt@telecom-paristech.fr (D. Joumblatt), dario.rossi@telecom-paristech.fr, dario.rossi@polytechnique.edu, dario.rossi@enst.fr (D. Rossi).

http://dx.doi.org/10.1016/j.comnet.2016.06.001
1389-1286/

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, which 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, 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. 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 [1–4] 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* [1–4,7], 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 (Section 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 (Section 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 (Section 4). The scenario of our controlled experimental evaluation is discussed next, carefully calibrating tools for delay and bandwidth emulation and measurement (Section 5). We report results of a thorough Mininet emulation campaign, investigating several important system and scenario parameters, contrasting experimental vs. modelings results (Section 6), and discuss practical aspects that a full-blown troubleshooting framework needs to take into account (Section 7). Finally, we cast our work in the context of related effort (Section 8) and summarize our main lessons learned and contributions (Section 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 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

[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.

Please cite this article as: F. Espinet et al., Framework, models and controlled experiments for network troubleshooting, Computer Networks (2016), http://dx.doi.org/10.1016/j.comnet.2016.06.001