Pythia: Detection, Localization, and Diagnosis of Performance Problems

Partha Kanuparthy, Yahoo Labs
Danny H. Lee, Warren Matthews, and Constantine Dovrolis, Georgia Institute of Technology
Sajjad Zarifzadeh, Yazd University

ABSTRACT

Performance problem diagnosis is a critical part of network operations in ISPs. Service providers use a combination of approaches to troubleshoot performance of their networks, such as active monitoring infrastructure and data collection (SNMP, Netflow, router logs, table dumps, etc.) along with customer trouble tickets. Some of these approaches, however, do not scale to wide area inter-domain networks due to unavailability of such data; moreover, troubleshooting is either *reactive* (e.g., driven by customer complaints) or (typically) automated using static thresholds. In this article, we describe the design and implementation of a system for root cause analysis and localization of performance problems in ISP networks. Our approach works with legacy monitoring infrastructure (e.g., perfSONAR deployments) and does not need specialized active probing tools or network data. Our system provides a language for network operators to define performance problem signatures, and provides near-real-time performance diagnosis and localization. We describe our deployment of Pythia in perfSONAR monitors in production networks in Georgia, covering over 250 inter-domain paths.

INTRODUCTION

Research networks such as the U.S. Department of Energy ESnet and the GÉANT in Europe, as well as commercial networks, are rapidly scaling up their network capacity in response to network-intensive scientific applications (e.g., remote computation and visualization of distributed big data) and geographically distributed cloud infrastructure. Increase in available bandwidth and network capacity, however, may not necessarily translate to improved network performance. These applications and infrastructure use end-to-end paths that span multiple administrative domains and use diverse link and network technologies, thus making it hard to monitor and troubleshoot network performance. Performance problems may not manifest themselves as outright failures, but sporadic issues that go undetected, in turn reducing the maximum possible performance.

In order to understand problems, network operators typically use end-to-end monitoring infrastructure that comprises a number of commodity nodes located at vantage points. Measurement federations between networks enable operators and users to monitor and troubleshoot end-to-end performance. There are several key challenges in building a measurement federation for performance problem localization and diagnosis. First, the system has to scale to a large number of monitoring nodes. Second, the system should be able to diagnose sporadic (short-term) performance problems, which are not uncommon in high-capacity networks, and diagnose them in near real time to be useful. Third, the diagnosis system should be configurable with performance problem knowledge from operators of different networks in the federation.

Nodes of a federation perform active measurements between each other. For example, monitoring infrastructure based on perfSONAR [1] is used in several academic, research, and commercial networks. A typical monitoring deployment using perfSONAR consists of commodity nodes that perform full or partial mesh one-way delay, loss, reordering, and throughput measurements between each other; perfSONAR uses active measurement tools such as OWAMP [2] and BWCTL to do these measurements. In addition, perfSONAR nodes can periodically record the IP-layer network topology using the traceroute tool.

Network operators typically also deploy passive monitoring infrastructure such as Simple Network Management Protocol (SNMP) traps, Netflow, and syslogs from network devices. While active and passive methods are useful, they have limitations. Passive monitoring gives detailed performance data, which can be used to perform detailed diagnosis and localization (e.g., AT&T's G-RCA [3], NICE [4], and Giza [5];

This work was done when Partha Kanuparthy and Sajjad Zarifzadeh were at Georgia Tech.

0163-6804/13/$25.00

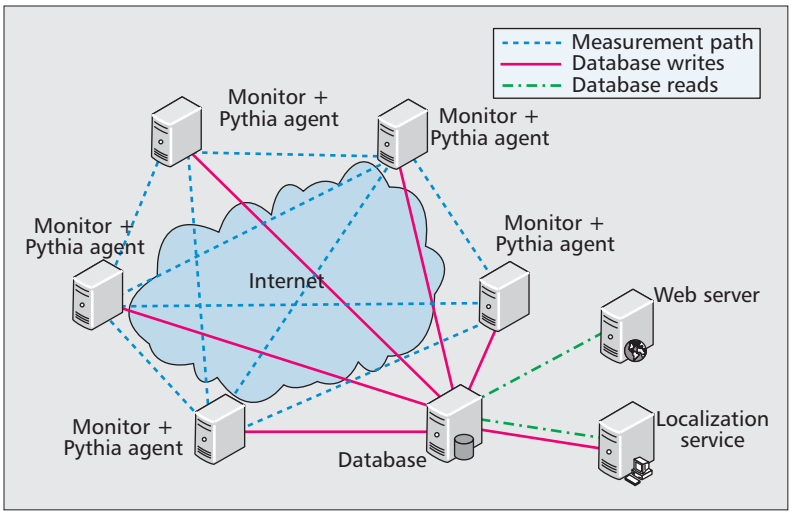

Figure 1. *Architecture of a Pythia deployment over a legacy monitoring infrastructure. We deploy lightweight agents on the monitors, and delegate storage, indexing, and compute-intensive tasks to nodes outside the monitoring infrastructure. The web server renders results as a browser-based front-end.*

SyslogDigest [6] and Sherlock [7]). Active monitoring infrastructure, on the other hand, scales to large networks and works when paths in the ISP traverse other administrative domains such as transit networks. In this article, we focus on active monitoring infrastructure.

Pythia is a distributed system that aims to fill a gap in existing network monitoring approaches by incorporating novel techniques to identify and explain problems in inter-domain paths, attributing probable causes to the router interfaces responsible. Pythia identifies problems that include standard ones like link failures and route changes, but also less obvious problems like router misconfiguration, intermittent congestion, and underprovisioned buffers. Specifically, Pythia works on top of legacy monitoring infrastructure deployments to solve three objectives:

- Detect performance problems
- Diagnose root cause(s) of detected problems
- Localize problems to network interfaces

Detection refers to the question of *whether* there is a performance problem in a network path at any point of time, while diagnosis and localization refer to the *why* (i.e., root cause) and *where* of the detected performance problem, respectively.

We design Pythia to augment existing troubleshooting procedures that network operators use. Pythia provides operators with near-real-time performance problem diagnosis and localization. Pythia focuses on short-term performance problems (shorter than five minutes). This is an important goal, since short-term problems may not be evident to the operator, unlike problems that last for hours; moreover, short-term problems may indicate a potential failure. Pythia provides operators with a visualization of problem summaries across the monitored network. Operators can extend the diagnosis functionality of Pythia by adding new definitions for performance problems based on experience or knowledge of the network.

We deploy Pythia in conjunction with perfSONAR in several K–12 school district production networks in Georgia (United States) and in Georgia Tech. Our deployment monitors over 250 inter-domain Internet paths. We configure Pythia with a default set of 11 performance pathology definitions, spanning congestion, buffer provisioning, loss, reordering, and routing-based pathologies. Our system has scaled to over 100,000 pathologies a day (Pythia provides a sensitivity knob to tune reporting). Operators can access the system front-end at http://pythia.cc.gatech.edu.

In this article, we describe the design and implementation of Pythia. This article is organized as follows. First, we cover the system architecture. We next give an overview of the detection, diagnosis, and localization methods. Then, we describe our deployment and results.

System Architecture

Pythia works with legacy monitoring infrastructure. We design Pythia to scale to hundreds (potentially thousands) of monitoring nodes without affecting the accuracy and timing of measurement processes on the monitoring nodes. There are design constraints when we work with legacy monitoring infrastructure. In our design, we decentralize computation (detection, diagnosis, and localization) as much as possible, and at the same time, we implement efficient algorithms as agents at monitoring nodes. We implement computation that requires a view of measurements across multiple monitors using a central *bulletin board* instead of communications between monitors.

At a high level, the Pythia system consists of the following components (Fig. 1). We run an agent process at each monitoring node. The agents report useful summaries in real time to a central bulletin board — a relational database engine. A web server interfaces with the database to generate an interactive dashboard showing statistics of performance problems in the monitored network.

Agent

We introduce a *lightweight* agent process on each monitoring node. Recall that each monitoring node records measurements as it receives probing packets generated by other monitoring nodes. The agent interfaces with these measurements in real time to:

- Detect if there is a performance problem at any point of time on each monitored path
- Diagnose root cause(s) of detected performance problems
- Write real time diagnosis and summaries of path performance to the central database

We design efficient algorithms and optimize CPU-bound logic in the agent process. On an average, the agent process takes about 60 μs to process each measurement; this allows the agent to scale to a large number of monitored paths.[1]

Database

The agents write problem diagnosis and time series summaries to a central relational database. The database is used as a bulletin board for agents

[1] *The typical path sampling rate in perfSONAR deployments is 10 Hz.*

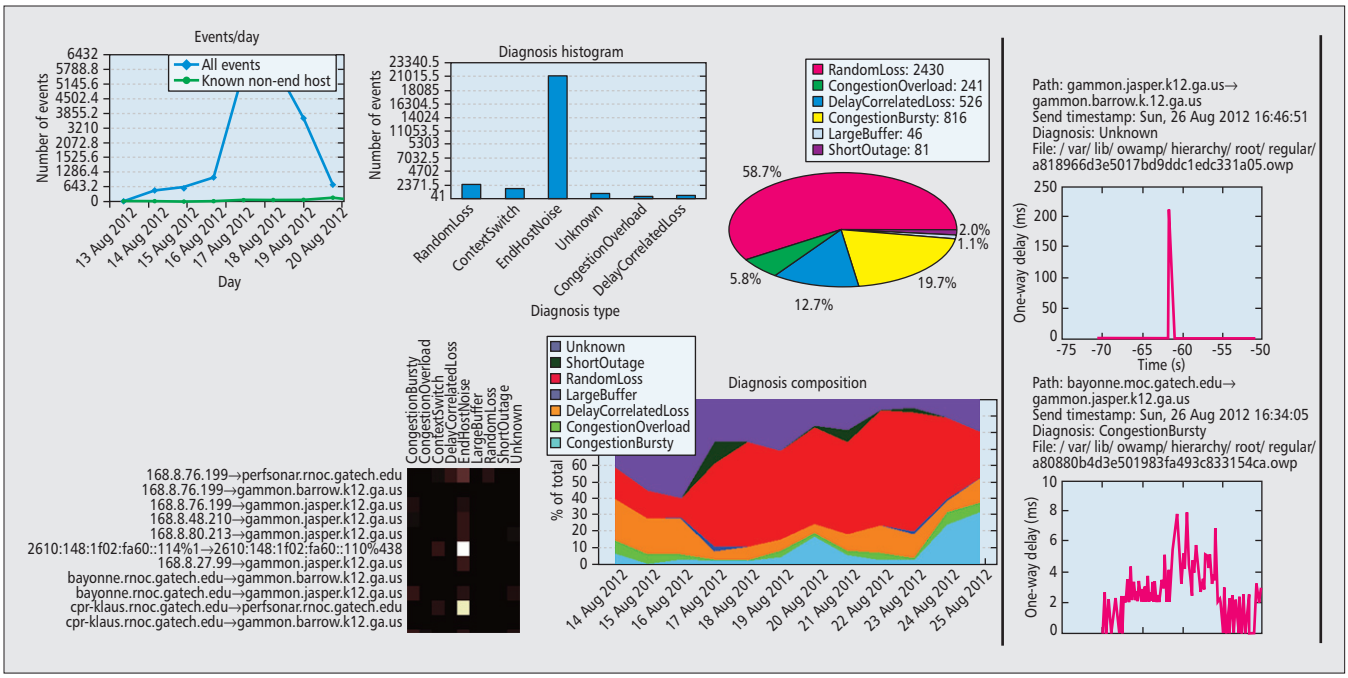

Figure 2. *A rendering of the Pythia front-end showing the different visualizations. The live dashboard can be accessed at http://pythia.cc.gatech.edu.*

to interface diagnosis information from other monitoring nodes, for localization, and for generating the front-end. We require data from other agents to diagnose certain classes of pathologies. The time series summaries are used for localization of performance problems. In our initial implementation, we have used a MySQL database; we are looking at methods to horizontally scale the data and to optimize the read paths.

Localization Service

We delegate localization to a central process, since our localization algorithms require measurement data from the complete monitoring mesh (this simplifies the design by eliminating inter-agent communication). The localization process interfaces with time series summaries for each monitored path,[2] and with topology data (collected from periodic traceroutes between monitors). The localization service periodically queries monitors for the most recent topology data and caches it before running localization.

Front-End

Pythia uses a browser-based front-end to show summary statistics and visualizations of performance problems. We implement this functionality using a PHP engine on a web server; the PHP code interfaces with the database. The dashboard shows an overview of the health of the network, and allows operators to examine specific results or network paths in more detail using filters. The dashboard includes charts showing diagnoses aggregated by type and number of occurrences, and heatmaps visualizing frequency of diagnoses. The front-end shows localization data using a network map, with a heatmap overlay for problem frequency. Operators can select paths or IP interfaces to view problem summaries. Figure 2 shows example charts from the front-end.

The architecture of Pythia enables easy deployment in federated networks, since it integrates with legacy monitoring infrastructure. There are some limitations, however, of the architecture. Deploying agents in legacy infrastructure may be hard in some networks. In federated networks, it may be required to have access policies on performance problem data from different networks. In this work, we do not address these limitations.

Detecting Performance Problems

Detection is the following problem: *Given a time series of measurements (delays, losses, reordering) of a path in some time window, is there a performance problem?* Detection is the first step toward performance diagnosis and localization. It is executed by the agent at a monitor in real time, as the agent interfaces with measurement streams recorded at that monitor. The agent running on the monitor divides incoming measurements for a path into back-to-back windows of 5 s duration, and the agent schedules detection on windows across paths in least recently used order.

Based on the problem statement for detection, we define three types of detection: delay, loss, and reordering problem detection. Our goal is to implement lightweight detection algorithms that are simple: specifically, we do not expect the network operator to define and constantly fine-tune static thresholds for path delay, loss, and reordering. Instead, we define a performance problem using first principles.

We approach the detection problem as follows. We define detection as the problem of *finding significant deviations from the baseline value* of the metric of interest (delay, loss, or

[2] *A time series summary consists of a (delay, loss, reordering) metric tuple for back-to-back 5 s time windows of measurements.*

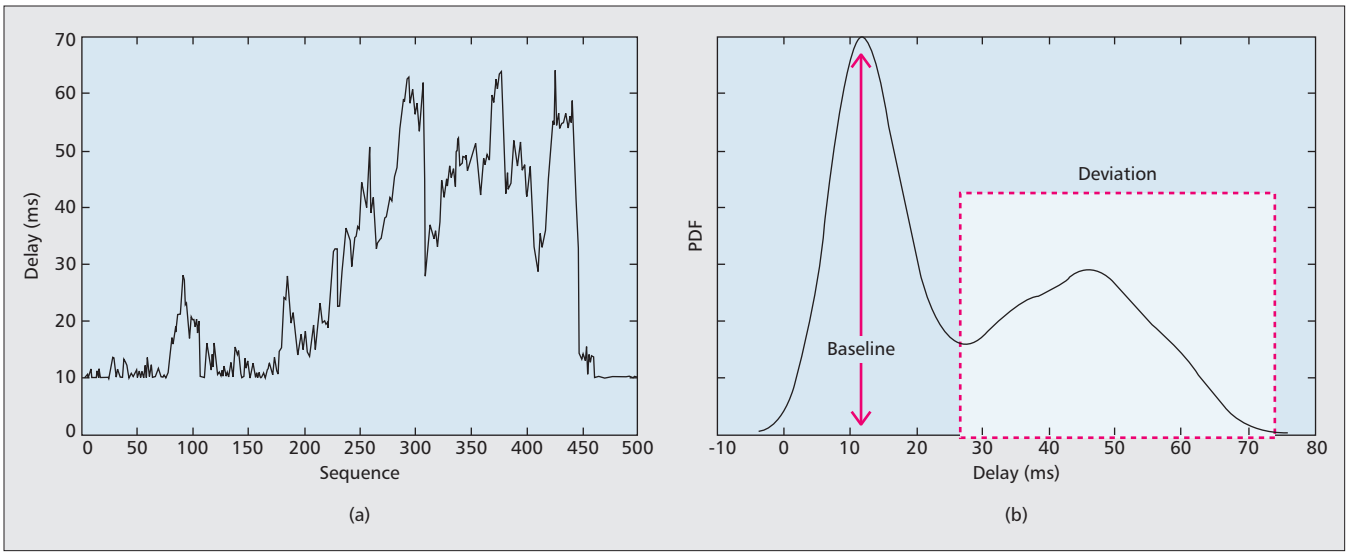

Figure 3. *An example of a delay-based performance problem (timeseries), and the corresponding pdf. Pythia uses the pdf to estimate a baseline performance and to detect deviations from the baseline.*

reordering). The baseline value is the value of the metric when there is no performance problem on that path during the time window. For loss and reordering problem detections, the baseline is simply zero losses and no reordering. Hence, a loss or reordering performance problem is detected when the agent sees a packet loss or packet reordering in the path measurements, respectively.

Delay performance problem detection is not straightforward, since the baseline delay for a path is an additive function of the end-to-end propagation delay, transmission delays across store-and-forward devices along the path, processing delays at each hop, and measurement noise and timestamping granularity at the monitors. The agent would hence need to estimate the baseline delay of the path from measurements, and find deviations from the baseline to detect delay performance problems. The baseline delay for a fixed end-to-end path typically corresponds to a *large density* of delay measurements that have a *similar* value. When there is a performance problem, we can have two baseline scenarios:

- There is a baseline with a non-negligible density *and* a large density of points with delays higher than the baseline.
- There is no baseline (in other words, there is no significant density at any delay value).

We estimate baseline delay and detect if a measurement window includes a deviation from the baseline using the probability density function (PDF) of the delay measurements. The agent estimates the PDF using a non-parametric kernel smoothing density estimate with a Gaussian kernel; the bandwidth is estimated using Silverman's Rule of Thumb [8]. A smoothing PDF estimate allows us to find density "peaks" in the PDF space using a single pass, since the PDF estimate is continuous. The lowest delay peak with a significant density corresponds to the baseline estimate. The agent identifies if there is a deviation from the baseline using one of two conditions above.

The agent allows the operator to tune detection output using a *sensitivity knob* on a scale of 0 (detect all problems) to 10 (detect problems with significant fraction of deviations from the baseline).

Diagnosing Performance Problems

Diagnosis refers to the problem of *determining the root cause(s)* of a detected performance problem. At a high level, Pythia diagnoses performance problems by matching the problem time series (delay, loss, and/or reordering) with a set of pathology definitions. We use domain knowledge to input a default set of 11 pathology definitions to Pythia, and we expect the network operator to input more based on her knowledge of the network and its problems. Domain-knowledge-based diagnosis designed on active measurement infrastructure has a significant advantage over methods that rely on data from devices in the network (e.g., SNMP, Netflow, and router logs), since the former can scale to large wide area networks whose paths span multiple administrative domains.

We define a performance pathology as follows. A pathology can be viewed as the existence of one or more *symptoms*. A symptom is a pattern in the delay, loss, and/or reordering timeseries. More formally, a symptom is a *Boolean-valued test on a measurement time series*. A pathology is a *Boolean expression on one or more symptoms*. The Boolean expression for a pathology can include conjunctions, disjunctions, and negations of symptoms. Pythia provides a language by which a network operator can add new pathology definitions to the system. The agent process reads the input sequence of pathology definitions at bootstrap time and generates efficient diagnosis code from the rules. An example pathology definition for a case of network congestion is the following:

```
PATHOLOGY CongestionOverload DEF
delay-exist AND high-util AND NOT
```

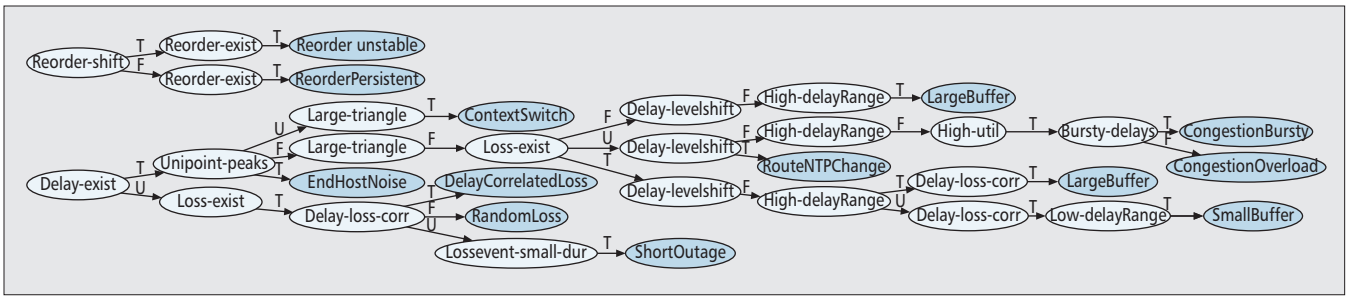

Figure 4. *Diagnosis forest generated using default set of pathology definitions in Pythia.*

```
bursty-delays AND NOT high-delayRange
AND NOT large-triangle AND NOT uni-
point-peaks AND NOT delay-levelshift
```

The default set of pathology definitions input to Pythia span congestion, buffer provisioning, loss, reordering, and routing-based pathologies. We look at the diagnosis specifications later.

The agent parses the pathology definitions and generates an efficient intermediate representation that minimizes the number of times the symptoms are tested to diagnose a performance problem. It is important to minimize the number of symptom evaluations, since the agent process should not affect the monitoring processes on the node by consuming CPU cycles; at the same time, the agent is required to test all symptoms for a particular pathology. The agent generates a forest of decision trees from the pathology definitions. The leaf nodes of each decision tree correspond to pathologies, and the non-leaf nodes correspond to symptoms; hence, a path from the root to a leaf represents a conjunction of symptoms along the path.[3] We can have multiple decision trees in the case when the set of pathologies can be divided into subsets such that there are no overlapping symptoms used by the subsets. The agent prunes redundant symptom nodes and edges in each decision tree.

The diagnosis forest generated using the default pathology set in Pythia is shown in Fig. 4. The figure shows two diagnosis trees, one for delay- and loss-based pathologies, and the other for reordering-based pathologies. Note that symptom nodes could have up to three child edges labeled true (T), false (F), and unused (U). Although the agent prunes unused symptoms, not every unused symptom can be pruned (e.g., if a symptom has subtrees that use the symptom).

We have designed a decision tree construction method from scratch instead of using existing methods such as *C4.5* or *ID3* due to the nature of the pathology definitions. A pathology typically uses only a subset of the set of symptoms. A detected problem may include multiple pathologies as root causes. The pathology definitions may not use both Boolean outcomes for every symptom. We need to consider all symptoms for diagnosing a pathology, instead of the most "useful" subset approach used by traditional tree construction methods.

If a detected problem does not match any of the input pathology definitions, it is tagged by the agent as an "unknown" pathology. The agent processes "unknown" pathologies to see if they originated from one of the monitors; it does this by correlating with diagnoses across other monitors.

Localizing Performance Problems

We define localization as the problem of determining *which IP interface(s) in the network resulted in the detected performance problem.* Pythia uses IP-layer topology data recorded using traceroute at monitors, and combines topology data with measurement summaries across all monitored paths to localize performance.

We implement localization in Pythia as two asynchronous distributed processes:

- The agent writes a summary time series every 5 s of delays, losses, and reordering for each path to the database repository.
- A localization service (on a central node) runs localization algorithms with input as the summary and topology data.

Note that the localization service cannot be triggered by an agent after detection, since localization algorithms need the network topology as well as a snapshot of measurements across the whole network. Since localization runs at the granularity of 5 s time windows, the service can correlate localization output with diagnoses from the database repository.

Our localization algorithms are based on *network tomography*, which is a class of methods for inference of link-level performance from end-to-end measurements. At a high level, our algorithms look at common links (interfaces) between paths that have a performance problem at any given time. The algorithms then iteratively prune a list of faulty links until all paths with performance problems are accounted for.

We use two forms of network tomography in Pythia's localization service: Boolean tomography [9] and range tomography [10]. Boolean tomography assigns a Boolean performance attribute (good or bad) to network links. We use boolean tomography for localizing loss- and reordering-based performance problems. Range tomography takes into account the magnitude of the metric (instead of characterizing it as good or bad) to localize problems and estimate performance of links with problems. We use range tomography for localizing delay-based performance problems in the localization service.

We show the differences between the two localization techniques with an example. Consider the network shown in Fig. 5: s_1 to s_4 are monitors doing delay measurements in a full mesh. Suppose that the three paths $s_1 \rightarrow s_4$, $s_2 \rightarrow s_4$, and $s_3 \rightarrow s_4$ have delay-based performance problems (high end-to-end delays). If we use Boolean

[3] *The agent replaces each disjunction by two expressions of conjunctions in a pre-processing step.*

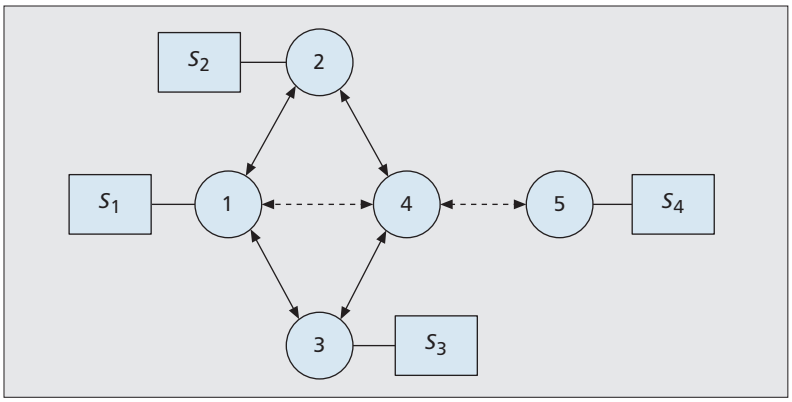

Figure 5. *Localization: a simple network example with four monitors. The dashed lines indicate links with pathological delays. Boolean and Range tomography algorithms find different set of links as pathological.*

tomography, the common link between the three paths, link $4 \rightarrow 5$, is localized as the link with the performance problem. If we use range tomography, we consider the delay measurements: suppose that the end-to-end delays of $s_1 \rightarrow s_4$, $s_2 \rightarrow s_4$, and $s_3 \rightarrow s_4$ are 500 ms, 300 ms, and 300 ms, respectively. Range tomography would localize the performance problem to link $4 \rightarrow 5$, assigning it a delay of 300 ms, and link $1 \rightarrow 4$, assigning it a delay of 200 ms.

Current Deployment

We deploy Pythia on an inter-domain perfSONAR monitoring deployment. The deployment monitors paths between Georgia Tech and 15 K–12 school districts in Georgia as part of the Georgia Measurement and Monitoring (GAMMON) project. The GAMMON project aims to quantify network performance and assess feasibility of online learning requirements of the Georgia Department of Education in underserved rural districts. The monitoring deployment is also used by network managers for troubleshooting and planning.

The current deployment monitors over 250 paths. Each monitor runs two tools, One-Way Ping (OWAMP [2]) and traceroute, to every other monitor in the monitoring network. For a path $A \rightarrow B$ between monitors A and B, OWAMP sends a small UDP probing packet (from userspace) every 100 ms on average as a Poisson process; the packets are timestamped at A and B, and a sequence number is generated for each probe by A. This allows Pythia to observe end-to-end delay, loss, and reordering for each monitored path. The monitoring deployment runs a traceroute every 10 min for a path. We configure our deployment with 11 pathology definitions, which fall into the following classes.

Congestion and buffering: We define two forms of congestion: overload and bursty congestion. Overload is a significant and persistent queue backlog in one or more links on a path. Bursty congestion is a significant backlog that is also bursty. Overload induces high delays in traffic, while bursty congestion induces high delays as well as high jitter. We also define pathologies corresponding to over- and underprovisioning of buffers in the path; overprovisioning can induce very high delays, while underprovisioning induces packet losses.

Loss pathologies: We define two loss-based pathologies. First, delay correlated losses are accompanied by high delays in their neighboring measurement samples (e.g., congestion). Second, random losses are not accompanied by high delays in the neighboring samples; in other words, random losses are accompanied by delays around the baseline. Random losses may be indicative of physical layer failures such as bad fiber. We also define short outages as loss events that have durations on the order of seconds.

Routing pathologies: We define level shifts in delay time series as route changes. Our definitions distinguish between route flaps and long-term route changes.

Reordering pathologies: We define two forms of packet reordering, stationary and non-stationary. Stationary reordering is persistent reordering that can happen due to network configuration such as per-packet multipath routing or switch fabric. Non-stationary reordering may occur due to frequent routing changes.

End-host pathologies: End-host (monitor) pathologies are not network pathologies, and may not be useful to the network operator. We need to diagnose them, however, since they can lead to pathological delay, loss, or reordering signatures. An end-host event can occur at either the sender or the receiver monitoring host due to the operating system (OS) environment. A busy OS environment may lead to excessive application context switches, which in turn induce delays in servicing probing packets (at the sender) or timestamping probing packets (at the receiver).

Unknown pathologies: Unknown pathologies are detected pathologies that do not match any of the pathology definitions configured in Pythia. An agent running at monitor A processes unknown pathologies to check if they are induced by end-host effects using diagnosis data from other agents. Specifically, when A finds an unknown event at time t, it checks if a significant fraction of monitored paths to and from A show end-host pathologies around time t; if it does, agent A marks these unknown pathologies as end-host pathologies. After this step, we typically find less than 10 percent of unknown pathologies in our deployment.

We validate the diagnosis methods and their accuracy in a separate paper [11].

Results

We present results from our GAMMON deployment of Pythia. We consider performance problems that occurred over a week, from March 18 to March 24, 2013. We correlate the diagnosis and localization output by finding the overlaps between paths that had the same diagnosis and links that were localized during that problem. If at least three diagnosis pathologies have a common diagnosis, we consider it as a "single problem."

In terms of diagnosis, we identified 51,961 problems, of which 28,125 are network-related, 26,447 are end-host-related, and 3513 are unknown pathologies. Note that the pathology definitions in Pythia allow a detected problem to be diagnosed

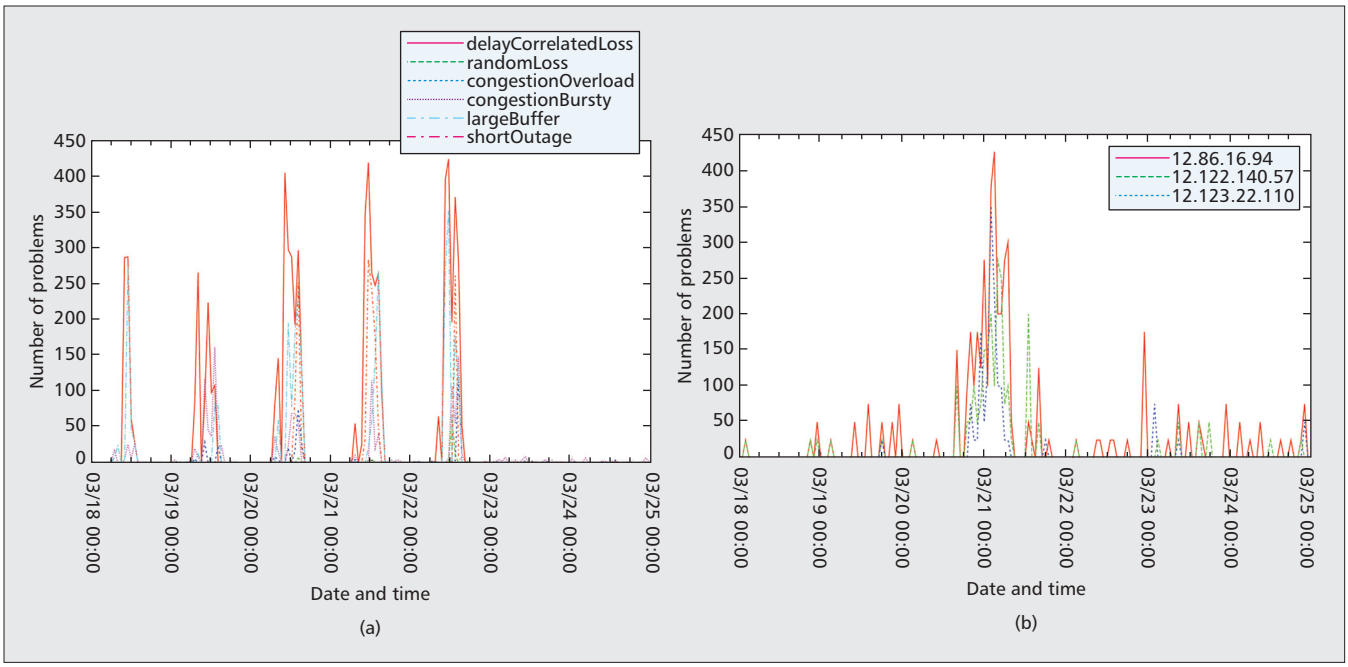

Figure 6. *Network pathologies from the live deployment of Pythia: a) timeline of problems for an interface showing diurnal and weekly patterns; b) timeline for three interfaces showing non-peak-hour congestion.*

as both end-host-related and network-related problems simultaneously.[4] Hence, the total number of detected problems can be smaller than the sum of the network, end host, and unknown pathologies. We find no instances of reordering-based pathologies in the network. The main pathologies affecting the network are delay-correlated losses (19,722 problems), bursty congestion (8138) and large buffer-based (7195) problems. We see a smaller incidence of short outages (3866 problems) and random losses (1381).

From the localization output, out of a total of 152 IP interfaces in GAMMON, 120 interfaces are affected by at least one problem. Although the fraction of interfaces affected by problems in the network is typically high, only 17 interfaces exhibit more than 10 problems/h. Out of those 17, only four interfaces see more than 60 problems/h, and are responsible for the majority of the problems diagnosed in the network. Of these, three interfaces are close to the network edge. We look at two observations from our Pythia deployment next.

We look at the timeline of performance problems during the week for a network interface. We see a diurnal pattern of performance problems, with a rise in network problems during working hours, and no problems during the two weekend days (since schools are closed, leading to low network utilization). We show an hourly time series of problems for an interface in Fig. 6a. We see a significant incidence of congestion-related problems during working hours (8 a.m. to 5 p.m.) on weekdays. In particular, we find a large number of delay-correlated losses and overprovisioned buffer-related problems. In addition, we see a small number (300) of short outage problems from Wednesday to Friday. We checked with the network operators and verified that this interface belongs to a router on the border of Franklin County Schools, which is known to see frequent congestion and packet loss (an upgrade is planned).

We also find multiple network interfaces in GAMMON that show significant amounts of congestion-related problems during non-working hours. Figure 6b shows the hourly incidence of congestion overload pathologies diagnosed in three network interfaces. We see that there are several congestion problems from 8 p.m. to 7 a.m. on the interfaces. According to traceroute, these three interfaces are back-to-back hops on a path; this shows a possible case where the network needs to be reprovisioned.

Next Steps and Call for Deployment

We can mention here what we would like to add next in terms of functionality, and to invite people to deploy our system and work with us in extending its functionality. We would like to extend an invitation to network operators to deploy Pythia in their monitoring infrastructure. We are currently adding monitoring nodes from different networks into our Pythia infrastructure. We expect that the GAMMON deployment of Pythia will grow to over 30 monitors in K–12 networks. We are collecting ground truth from network operators of the different GAMMON networks to improve Pythia. In addition, we are deploying Pythia on perfSONAR hosts in the U.S. Department of Energy's Energy Sciences Network (ESnet), and we are working with the Internet2 community to deploy Pythia. We welcome the opportunity to work with network operators in the wider perfSONAR community, and we would like to extend the Pythia agent to work with monitoring deploymens from other networks. We hope that Pythia will augment troubleshooting and capacity planning efforts in networks, and we

[4] This is true when the problem is a superposition of the two types of pathologies. We found that busy monitors can induce a significant amount of pathologies in measurements, causing such superpositions frequently.

We are extending the Pythia infrastructure to monitoring deployments in more networks, and we would like to invite the network operator community to try Pythia in their monitoring infrastructure.

look forward to improving the system based on operator feedback. A dashboard for our live deployment is at http://pythia.cc.gatech.edu.

Conclusion

In this article, we have presented an overview of Pythia, a distributed system for detecting, diagnosing, and localizing performance problems. Pythia works on top of legacy monitoring infrastructure that is deployed in ISPs. Pythia provides the operator with a real-time view of short-term performance problems. Network operators can easily configure Pythia with performance problem definitions based on their domain knowledge of the network and its performance. We have described our deployment of Pythia that monitors over 250 inter-domain paths. We are extending the Pythia infrastructure to monitoring deployments in more networks, and we would like to invite the network operator community to try Pythia in their monitoring infrastructure.

Acknowledgments

We thank Jason Zurawski from Internet2, and Joe Metzger, Brian Tierney, and Andrew Lake from ESnet for providing us with the OWAMP data sets. We are also grateful to the anonymous reviewers for their constructive comments. This research was supported by the U.S. Department of Energy under grant number DE-FG02-10ER26021.

References

[1] A. Hanemann *et al.*, "PerfSONAR: A Service Oriented Architecture for Multi-Domain Network Monitoring," *Service-Oriented Computing*, 2005, pp. 241–54.
[2] S. Shalunov *et al.*, "A One-Way Active Measurement Protocol (OWAMP)," Network Working Group, RFC 4656, 2006.
[3] H. Yan *et al.*, "GRCA: A Generic Root Cause Analysis Platform for Service Quality Management in Large IP Networks," *ACM SIGCOMM CoNEXT*, 2010.
[4] A. Mahimkar *et al.*, "Troubleshooting Chronic Conditions in large IP Networks," *ACM SIGCOMM CoNEXT*, 2008.
[5] A. Mahimkar *et al.*, "Towards Automated Performance Diagnosis in A Large IPTV Network," *ACM SIGCOMM CCR*, vol. 39, no. 4, 2009, pp. 231–42.
[6] T. Qiu *et al.*, "What Happened in My Network: Mining Network Events from Router Syslogs," *ACM SIGCOMM IMC*, 2010.
[7] P. Bahl *et al.*, "Towards Highly Reliable Enterprise Network Services via Inference of Multi-Level Dependencies," *ACM SIGCOMM CCR*, vol. 37, no. 4, 2007, pp. 13–24.
[8] B. Silverman, *Density Estimation for Statistics and Data Analysis*, Monographs on Statistics and Applied Probability, Taylor & Francis, 1986.
[9] N. Duffield, "Simple Network Performance Tomography," *ACM SIGCOMM IMC*, 2003.
[10] S. Zarifzadeh, M. Gowdagere, and C. Dovrolis, "Range Tomography: Combining the Practicality of Boolean Tomography with the Resolution of Analog Tomography," *ACM SIGCOMM IMC*, 2012.
[11] P. Kanuparthy and C. Dovrolis, "Pythia: Distributed Diagnosis of Wide-Area Performance Problems," 2013, tech. rep., Georgia Tech.

Biographies

Partha Kanuparthy (kanuparthy@gatech.edu) is a research scientist at Yahoo Research. He received his Ph.D. degree in computer science from the Georgia Institute of Technology. His research interests include systems performance measurement, estimation and diagnosis, and building high-performance systems.

Danny H. Lee is a graduate student in the Networking and Telecommunications Group at the Georgia Institute of Technology. He graduated from the National University of Singapore with a B.S. in computing (computer engineering). He has previously done research in novel technologies such as software-defined radio, and is interested in network tomography and network performance analysis.

Warren Matthews is a research scientist at the Georgia Institute of Technology. Since obtaining his Ph.D. in high energy physics in 1997, he has worked in numerous areas of advanced networking. He is currently leading the Georgia Measurement and Monitoring (GAMMON) project, a network performance infrastructure among K–12 school districts in the state of Georgia.

Sajjad Zarifzadeh received his Ph.D. from the University of Tehran, and is currently with Yazd University. He was a visiting student at the Georgia Institute of Technology, during which time he worked on localization in the Pythia project.

Constantine Dovrolis is an associate professor at the College of Computing of the Georgia Institute of Technology. He received his Ph.D. degree from the University of Wisconsin-Madison in 2000. His current research focuses on the evolution of the Internet, Internet economics, and applications of network measurement. He is also interested in cross-disciplinary applications of network science in biology, climate science, and neuroscience.