PYTHIA: DETECTION, DIAGNOSIS, AND LOCALIZATION OF NETWORK PERFORMANCE PROBLEMS.

Warren Matthews, Danny Lee, Partha Kanuparthy, and Constantine Dovrolis.
The Georgia Institute of Technology, Atlanta, Georgia, USA.
e-mail: warren.matthews@oit.gatech.edu

Research Paper

Abstract
Networks around the world, especially Research and Education (R&E) networks are increasing capacity and striving for higher availability and reliability. However, improved performance is not guaranteed. Network monitoring services often deploy a distributed infrastructure of nodes that perform end-to-end path measurements such as one-way delay, packet losses, and achievable throughput. An example of such a deployment is the perfSONAR infrastructure in several academic and commercial networks, including Internet2 and ESnet in the US and GEANT in the EU. The Pythia infrastructure is designed to operate on top of monitoring infrastructure deployments to solve three objectives: detect performance problems, diagnose root cause(s) of detected performance problems, and localize performance problems to network interfaces. Pythia has been deployed on perfSONAR hosts within the Georgia Measurement and Monitoring (GAMMON) community. Currently 15 hosts are deployed around the U.S. State of Georgia in the South East United States.

Keywords
“Network Performance”, “Active Monitoring”, Detection, Diagnosis, Localization, perfSONAR.

Introduction
High-performance networks have revolutionized research and are poised to have a similar impact on education. Networks around the world, especially Research and Education (R&E) networks are increasing capacity and striving for higher availability and reliability. However, improved performance is not guaranteed. Quick resolution of network performance problems remains a critical part of network operations. Networks and Internet “health” monitoring services often deploy a distributed infrastructure of nodes that perform end-to-end path measurements such as one-way delay, packet losses, and achievable throughput. The goal of such infrastructure is to detect and alert network operators of network downtime; operators use these alerts to troubleshoot and maintain the network. Network operators typically also deploy passive monitoring infrastructure such as 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. However, passive data is typically not available to external operators and therefore cannot be used to construct and end-to-end path. 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.

An example of such deployments is the perfSONAR infrastructure in several academic and commercial networks, including Internet2 and ESnet in the US and GEANT in the EU. The perfSONAR infrastructure supports many measurement tools; a commonly deployed tool, OWAMP, measures one-way delays between pairs of monitors in the infrastructure.
The Pythia infrastructure is designed to operate in conjunction with such a monitoring systems. The implementation discussed in this paper uses OWAMP and traceroute data collected by a particular community of perfSONAR hosts. Pythia is a distributed system that aims to fill a gap in existing network monitoring approaches by incorporating novel techniques to identify and explain performance problems in inter-domain paths, attributing probable causes to the router interfaces responsible. Pythia identifies performance problems that include standard ones like link failures and route changes, but also less obvious performance problems like router misconfiguration, intermittent congestion and under-provisioned buffers. Specifically, Pythia works on top of monitoring infrastructure deployments to solve three objectives: detect performance problems, diagnose root cause(s) of detected performance problems, and localize performance problems to network interfaces.

Pythia is designed to augment existing troubleshooting procedures that network operators use. Pythia provides operators with a 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 performance problems may not be evident to the operator, unlike performance problems that last for hours; moreover, short-term performance problems may indicate a potential failure. Pythia provides operators with a visualization of performance 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.

**Overview of Pythia**

At a high-level, the Pythia system consists of the following components: An agent process analyzes data at each monitoring node; The agent reports useful summaries in real time to a central database; A web server interfaces with the database to generate an interactive dashboard showing statistics of performance problems in the monitored network.

**Agent:** A lightweight agent process on each monitoring node analyzes 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, and write real time diagnosis and summaries of path performance to the central database. An efficient algorithm optimizes CPU-bound logic in the agent process. On 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.

**Database:** The agents write performance problem diagnosis and time-series summaries to a central database. The database is used to interface diagnosis information from other monitoring nodes, for localization and for generating the front end. Certain classes of pathologies require data from other agents in order to be diagnosed. The time-series summaries are used for localization of performance problems.

**Localization service:** The localization algorithms require measurement data from the complete monitoring mesh. This simplifies the design by eliminating inter-agent communication, but necessitates an additional central server to collect and analyze the full-mesh data. The localization service periodically queries monitors for the most recent topology data and caches it before running localization. The localization process interfaces with time-series summaries and topology data.

**Front end:** Pythia uses a browser-based front end to show summary statistics and visualizations of performance problems. 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 heat-maps visualizing frequency of diagnoses. The front end shows localization data using a network map, with a heat-map overlay for performance problem frequency.
Operators can select paths or IP interfaces to view performance problem summaries.

**Detection**

Detection is the first step towards diagnosis and localization. The agent running on the monitor divides incoming measurements for a path into back-to-back windows of 5 seconds duration, and the agent schedules detection on windows across paths in least recently used order.

Three types of detection are defined: delay, loss, and reordering. The goal is to implement lightweight detection algorithms that are simple and do not expect the network operator to define and constantly fine-tune static thresholds. Performance problem are defined using first principles.

Detection can be defined as the process of *finding significant deviations from the baseline value* of the metric of interest (delay, loss or 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 performance 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 time-stamping 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, which have a *similar* value. When there is a performance problem, we can have two baseline scenarios: either there exists a baseline with a non-negligible density and a large density of points with delays higher than the baseline, or there is no baseline (in other words, there is no significant density at any delay value).

**Figure 1:** An example of a delay-based performance problem and the corresponding PDF.

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 the Silverman’s Rule of Thumb [1]. 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. Figure 1 shows an example of a delay-based performance problem and the corresponding...
PDF. In this case, the deviation is a significant deviation from the baseline, triggering a detection event.

**Diagnosis**

Diagnosis refers to the process of determining the root cause(s) of a detected performance problem. At a high level, Pythia diagnoses performance problems by matching the time-series (delay, loss and/or reordering) with a set of pathology definitions.

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 timeseries. 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:

```plaintext
PATHOLOGY CongestionOverload DEF delay-exist AND high-util AND NOT bursty-delays AND NOT high-delayRange AND NOT large-triangle AND NOT unipoint-peaks AND NOT delay-levelshift
```

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. We can have multiple decision trees under 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.

If a detected performance 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.

The implementation is configured with 11 pathology definitions which fall into the following classes:

**Congestion and buffering:** Two forms of congestion are considered: 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 under-provisioning of buffers in the path; over-provisioning can induce very high delays while under-provisioning induces packet losses.

**Loss pathologies:** Two loss-based pathologies are defined. 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. Also, a short outages is defined as loss events that have durations in the order of seconds.

**Routing pathologies:** Level shifts in delay time-series as defined as route changes. The definitions distinguish between route flaps and long-term route changes.

**Reordering pathologies:** Two forms of packet re-ordering are defined, stationary and non-stationary reordering. 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.

**Monitoring-host pathologies**: Monitoring-host 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 monitoring-host event can occur either at the sender or the receiver host due to the 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 time-stamping 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 a monitor A processes unknown pathologies to check if they are induced by monitoring-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 monitoring-host pathologies around the time t; if it does, agent A marks these unknown pathologies as monitoring-host pathologies. After this step, we typically find less than 10% of unknown pathologies in our deployment.

**Localization**

The aim of localization is to determine *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.

Localization in Pythia is implemented as two asynchronous distributed processes: the agent writes a summary timeseries every 5s, of delays, losses and reordering for each path to the database repository, and a localization service (on a central node) runs localization algorithms with input as the summary and topology data. 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 5s time windows, the service can correlate localization output with diagnoses from the database repository.

The 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, the 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. Two forms of network tomography are used in Pythia’s localization service: boolean tomography [2], and range tomography [3]. Boolean tomography assigns a boolean performance attribute (good or bad) to network links. Boolean tomography is used 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 performance problems and estimate the performance of links with performance problems. Range tomography is used for localizing delay-based performance problems in the localization service.

Using Range Tomography in Pythia attempts to combine the higher resolution of analogue tomography with the practicality of boolean tomography. The path measurements are still classified as good or bad, to exploit the fact that most network paths typically operate without congestion. This allows us to quickly remove a large number of links from the localization, as long as they only appear in good paths. For the remaining paths, the aim is not only to identify bad links, but also to estimate a range (interval) for their actual performance. The width of the resulting range depends on the variability in the underlying path measurements: more accurate measurements result in narrower range estimates.

**Visualization**

The front end is a dynamically generated interface for the operator to the performance problem data; it summarizes the data visually, shows recent events, and allows the operator to specify *filters* using one or more of
monitor, path, diagnosis type and time window. It is written in PHP and runs on top of an Apache web server.

The front end visualizes different types of information. First, it shows time series of network and all events, and the fraction of different diagnosis types each day. Second, it shows a heat map of per-path events by diagnosis type to visualize the significant performance problem types associated with each path. Third, it summarizes the distribution of event types – for all events and for network-only events. Finally, it shows a list of latest events, with path, timestamp, diagnosis and a delay+loss timeseries plot.

**Deployment**

Pythia has been deployed on perfSONAR hosts within the Georgia Measurement and Monitoring (GAMMON) community. Currently 15 hosts are deployed around the U.S. State of Georgia in the South East United States. 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. Network managers also use the monitoring infrastructure for troubleshooting and planning.

The current deployment monitors over 250 paths. Each monitor runs two tools: One-Way Ping (OWAMP) and traceroute to every other monitor in the monitoring network. For a path A → 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 time-stamped 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 minutes for a path.

**Results**

The data analyzed is that between March 31 and April 6. In that week, 160,271 detections were reported to the database with a total of 200,532 diagnoses. There were 84,984 (42.4%) network-related diagnoses, 101,843 (50.8%) host-related diagnoses, and 13705 (6.8%) were classified as unknown. Of the network-related diagnoses, the most common were random loss (38,433 diagnoses, 45.2%) and delay-correlated loss (21,434 diagnoses, 25.2%). No reordering was observed.

Earlier weeks were also examined. It was found that the number of reported detections varied significantly. The week chosen had the smallest number of detections reported. During the week from February 24 to March 2, 385,882 detected events were reported to the database. Variation is expected, but the large variation is probably due to the fact that several districts were on spring break and schools were closed.

From the localization output, out of a total of 152 IP interfaces in GAMMON, 105 interfaces are affected by at least one performance problem during the week. Out of the 105 interfaces, 57 (54%) have IP addresses within the AT&T network. This is not surprising as AT&T provide the connectivity for many of the school districts in Georgia. Out of the total 152 interfaces, 80 (52%) are in the AT&T network, i.e. the number of interfaces with performance problems is roughly representative of the total.

The top 10 most problematic interfaces reported between 291 and 534 performance problems during the week. However, most of these were close to the edge. Comparing again to the previous weeks, the number of interfaces appearing to have performance problems varied significantly.

Furthermore, a diurnal pattern of performance problems is typically observed, with a rise in network performance problems during working hours, and no performance problems during the weekend since schools are closed, and there is low network utilization. Significant incidences of congestion-related performance problems are seen during working hours (8am to 5pm) on weekdays. In particular, a large number of delay-correlated losses and over-provisioned buffer-related performance problems were seen. The network operators were able to verify that this one problematic interface belonging to a router on the border of the Franklin County
School System, is known to see frequent congestion and packet loss. The district is planning an upgrade.

**Future Work**

The Pythia team continues to work on the system, particularly methods to horizontally scale the data and to optimize the read paths. Furthermore, the developers are gathering case studies of how Pythia can augment troubleshooting and capacity planning efforts in networks, and will improve the system based on operator feedback.

The GAMMON deployment of Pythia is expected to grow to over 30 monitors. In addition, Pythia will be deployed on perfSONAR hosts in the US Department of Energy (DoE) Energy Sciences Network (ESnet); and with the Internet2 community. The Pythia team welcomes the opportunity to work with network operators in the wider perfSONAR community.

**References**


**Biographies**

**Warren Matthews** is a research scientist at the Georgia Institute of Technology. Since obtaining his PhD in High Energy Physics, Dr. Matthews has spent 15 years working on numerous areas of advanced networking. Warren is currently leading the GAMMON project, a network performance measurement and monitoring infrastructure amongst school districts in the US State of Georgia. Warren is also the technical lead for the Direct-to-Discovery program, an outreach program connecting scientists in their labs with students in their classrooms via HD video conferencing.

**Danny H. Lee** is a graduate student at the Networking and Telecommunications Group at the Georgia Institute of Technology. Before attending Georgia Tech, he spent 5 years at the Institute for Infocomm Research, Singapore, performing government-funded research and development in novel wireless technologies such as software-defined radio. His research interests include network tomography and performance analysis. He graduated from the National University of Singapore with a Bachelors in Computing (Computer Engineering).

**Partha Kanuparthi** is a recent graduate from the Networking and Telecommunications Group in the College of Computing at the Georgia Institute of Technology. Dr. Kanuparthi is now working at Microsoft Research. His research interests include network inference and its applications. He was at Intel Labs Pittsburgh during Summer 2010, and spent Summer 2008 in the Systems and Networking group at Microsoft Research Cambridge (UK). Prior to grad school, he was a member of the Research Division, Real-Time Collaboration Products, Oracle Corporation. Partha is also a graduate of the Indian Institute of Technology Kharagpur, completing his undergraduate degree in Mechanical Engineering.

**Constantine Dovrolis** is an Associate Professor at the College of Computing of the Georgia Institute of Technology. He received the Computer Engineering degree from the Technical University of Crete (Greece) in 1995, the M.S. degree from the University of Rochester in 1996, and the Ph.D. degree from the University of Wisconsin-Madison in 2000. His research interests include Internet economics, understand the evolution of the Internet ecosystem, measurement tools for the detection of network neutrality violations, and the scalability of interdomain Internet routing. His research has been supported from NSF, DOE, Cisco Systems, Google, Intel,
Telchemy, and others.