Pythia: Diagnosing Performance Problems in Wide Area Providers

Partha Kanuparthy, *Yahoo Labs;* Constantine Dovrolis, *Georgia Institute of Technology*

https://www.usenix.org/conference/atc14/technical-sessions/presentation/kanuparthy

This paper is included in the Proceedings of USENIX ATC '14: 2014 USENIX Annual Technical Conference.

June 19–20, 2014 • Philadelphia, PA

978-1-931971-10-2

Pythia: Diagnosing Performance Problems in Wide Area Providers

Partha Kanuparthy
Yahoo Labs

Constantine Dovrolis
Georgia Institute of Technology

Abstract

Performance problem diagnosis is a critical part of network operations in ISPs. Service providers typically deploy monitoring nodes at several vantage points in their network, to record end-to-end measurements of network performance. Network operators use these measurements offline; for example, to troubleshoot customer complaints. In this work, we leverage such monitoring infrastructure deployments in ISPs to build a system for near real time performance problem detection and root cause diagnosis. Our system works with wide area inter-domain monitoring, unlike approaches that require data sources from network devices (SNMP, Netflow, router logs, table dumps, etc.). Operators can input operational and domain knowledge of performance problems to the system to add diagnosis functionality. We have deployed the system on existing monitoring infrastructure in the US, diagnosing over 300 inter-domain paths. We study the extent and nature of performance problems that manifest in edge and core networks on the Internet.

1 Introduction

End-to-end diagnosis of network performance is a significant part of network operations of Internet service providers. Timely diagnosis information is integral not only in the troubleshooting of performance problems, but also in maintaining SLAs (for example, SLAs of enterprise network and cloud provider customers). Knowledge of the *nature* of performance pathologies can also be used to provision network resources.

Performance diagnosis, however, is a challenging problem in wide area ISPs, where end-to-end (e2e) network paths traverse multiple autonomous systems with diverse link and network technologies and configurations. In such networks, operators may not have: *(i)* performance metrics from all devices comprising e2e paths, and *(ii)* labeled training data of performance problems. In this work, we explore a new system for troubleshooting based on domain knowledge, relying on e2e measurements and not requiring training data.

We leverage the monitoring infrastructure that ISPs deploy to record network health – consisting of several commodity hardware *monitors*. These monitors run low-overhead network measurement tools similar to *ping*, to record e2e delays, losses and reordering of monitored paths. Operators place monitors at vantage points in the network to maximize network coverage. An example of such deployments common in wide area academic and research networks is the perfSONAR software [6]; there

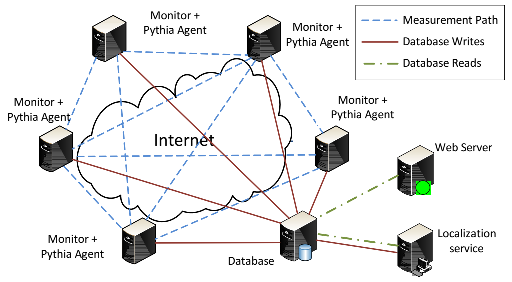


Figure 1: *A deployment of Pythia.*

are currently over 1,000 perfSONAR monitors spread across 25 countries [1]. Monitors running perfSONAR usually run the *One-Way Ping* (OWAMP) measurement tool, which probes over UDP every 100ms and records send/receive timestamps and sequence numbers.

Pythia works with e2e measurement streams recorded at monitors to do two tasks. First, it *detects* whether there is a performance problem on a monitored path at a given time. Second, it *diagnoses* the root cause(s) of detected performance problems. Pythia also *localizes* problems to network interfaces (using traceroutes recorded by monitors) [25]. In this paper, we focus on near-real time detection and diagnosis of short-lived and sporadic performance problems. Sporadic problems may not always be noticed by network operators, but can result in long-term failures and network downtime (e.g., gradual failure of an optical line card [1]).

A typical deployment of Pythia includes *lightweight* agent processes that run on monitors, and centralized database and web services (see Figure 1). Deployment can be done incrementally in ISP networks since deployment only involves adding the agent to new monitors. Pythia allows the operator to input diagnosis definitions using an expressive specification language as functions of *symptoms* in the measurements. This is useful since operators understand performance problems (and new ones that arise after network upgrades). Symptoms could include, for example, statistical changes in the delay timeseries, packet losses and/or packet reordering.

Pythia generates a *diagnosis forest* from the pathology specifications in order to reduce the number of symptoms that are tested on a measurement timeseries. The diagnosis forest reduces the agent's computation overhead at the monitor; and becomes important as the number of specifications grows with time or in large monitoring deployments. In practice, at a perfSONAR monitor probing a nominal set of 10 paths, each at 10Hz (default for

OWAMP), the Pythia agent receives a measurement every 10ms; we design a per-measurement agent run time of less than 100us. We describe common pathology specifications in Pythia that we wrote based on operator input: broadly related to congestion and buffering, loss nature, routing and reordering.

We make the following contributions in this paper. We design an efficient and scalable system and algorithms for real time detection and diagnosis (§2,4,5,6). We deploy Pythia in production wide area backbone networks to diagnose several pathologies. We use Pythia to do a first large-scale study of performance problems in edge and backbone networks (§9).

2 System and Monitoring Model

Pythia is a distributed system that works on top of existing monitoring infrastructure in ISPs. We design Pythia to scale to large monitoring deployments (potentially hundreds of monitors), without affecting the accuracy and timing of measurements taken at the monitors; and at the same time, with low network communication overhead. Pythia consists of a agent processes that run on monitors, a central database and a web server that renders real time diagnosis summaries (see Figure 1).

The agent is a lightweight process that performs two tasks. First, when the agent bootstraps, it parses the pathology specifications and generates efficient diagnosis code. Second, at runtime, agent reads measurements recorded at a monitor to detect and diagnose performance problems in near real time. It writes diagnosis output to the database. In order to minimize diagnosis-related traffic at agents, the agent runs diagnosis computation on the node that records the measurements; the agent computes diagnosis that requires information from other monitors using the database[1]. We design detection and diagnosis algorithms for the agent in Sections 4 and 5.

We consider a simple but general model of ISP monitoring. Suppose that N monitors are deployed by the ISP. Measurement tools in these monitors send active probes to other monitors, potentially over $N \times (N-1)$ end-to-end paths. For each monitored path $A \rightarrow B$, monitor A sends probes at a reasonably high frequency[2] to monitor B, and B records measurements of the probes; we do not require a specific probe sampling process. For each measurement probe that monitor B receives, a measurement tuple of sender and receiver timestamps, and a (sender) sequence number is recorded by B. The sequence number is incremented by one at the sender for each probe in the flow. We require loose clock synchronization (error margin of seconds) between the sender and receiver to

[1]An alternative design is to ship measurements to a centralized compute cluster; this may not be feasible in ISPs due to network policies.

[2]The probing frequency for a path is expected to be high enough to observe short-lived performance problems, but at the same time, the average probing traffic is expected to be low-overhead.

Listing 1 Pathology specification grammar.

```
1. 'SYMPTOM' symptom
2. 'PATHOLOGY' pathology 'DEF' ( symptom |
   'NOT' symptom )
3. symptom → symptom_1 'AND' symptom_2
4. symptom → symptom_1 'OR' symptom_2
5. symptom → ( symptom )
6. 'PROCEDURE' symptom func
```

correlate pathologies between monitors. Monitor M_i collects information about all probes sent to it; a lost packet is either "marked" by M_i as lost after a pre-defined timeout interval, or implicitly marked by a missing sequence number at M_i for that flow. We expect that a suitable *interface* exists on each monitor so that Pythia's agent can read measurements; the interface could be an API, a local cache, or simply files written to the disk.

As an example, the perfSONAR monitoring software follows this model. It runs the OWAMP tool; an OWAMP endpoint A sends 40B UDP packets at 10Hz to endpoint B, and B records timestamps and sequence numbers in a file. Reading these, the Pythia agent at B computes one-way delay, loss and reordering for $A \rightarrow B$.

3 Pathology Specification

One of the design goals of Pythia is to allow the operator to *add* performance pathology definitions to the system based on, for example, domain knowledge of the network or the network performance. To do this, we would need a simple model for a pathology; the model would enable us to design a pathology specification language.

We model a performance pathology as *a unique observation on a set of symptoms*. Given a measurement timeseries $\mathscr{E}$, a symptom is a boolean-valued test $T(\mathscr{E}) : \mathscr{E} \rightarrow \{0,1\}$ such that T returns `true` if the symptom exists in the timeseries. A pathology is a logical expression on one or more symptoms. Pathologies differ either in the set of symptoms on which they are defined, or on the logical expression. Examples of symptoms include "interquartile of delays exceeds 100ms", "loss probability exceeds 0.5" and "non-zero reordering metric".

The pathology specification language is based on the pathology model. A pathology can be specified using the language as conjunction or disjunction operations on symptoms. The language also allows negations of symptoms. Listing 1 shows the pathology specification language grammar. An example of a pathology specification for a form of congestion is the following:

```
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 statement specifies a rule for the pathology `CongestionOverload` using seven symptoms.

The language keyword `PROCEDURE` specifies subroutine names for symptom tests. We define 11 pathologies in Pythia by default along with their symptom tests (§6). In order to add a new pathology definition, the user adds a pathology in the specification language and writes subroutines for boolean-valued tests that the pathology uses (or the user could reuse existing symptom tests). The parser replaces specifications of each pathology P containing disjunctions with multiple specifications of P that only contain conjunctions. Expressions of conjunctions allow us to represent specifications as a decision tree.

The diagnosis module generates a diagnosis forest, a compact intermediate representation of the specifications (§5), and generates diagnosis code from it.

4 Detecting Performance Problems

The problem of *detection* is the first step towards performance diagnosis. The agent process at a monitor reads path measurements of packet delays, loss and reordering to test *if there is a performance problem at a given point of time in a monitored path*. A challenge in detection is to define a generic notion of a performance problem – which is not based on the symptoms or pathologies. We define detection using a performance *baseline*.

We define detection as *testing for significant deviations from baseline end-to-end performance*. Suppose that we have a timeseries of measurements from a sender $\mathscr{S}$ to a receiver $\mathscr{R}$ (our methods are robust to measurement noise in timestamping). Under non-pathological (normal) conditions, three invariants hold true for a timeseries of end-to-end measurements of a path:

1. The end-to-end delays are *close to* (with some noise margin) the sum of propagation, transmission and processing delays along the path,
2. No (or few) packets are lost, and
3. No packets are reordered (as received at $\mathscr{R}$).

These invariants define baseline conditions, and a violation of one or more of these conditions is a deviation from the baseline. We implement three types of problem detection: *delay*, *loss* and *reordering* detection, depending on the baseline invariant that is violated.

The agent divides the measurement timeseries for a given path into non-overlapping back-to-back *windows* of 5s duration, and marks the windows as either "baseline", or as "problem" (i.e., violating one or more invariants). The agent then *merges* problem windows close to each other into a single problem window.

Delay detection: Delay detection looks for *significant deviations* from baseline end-to-end delays. We use the delay invariant condition (1) above, which can be viewed as a condition on modality of the distribution of delay measurements. Under normal conditions, the distribution of delay sample in the window will be unimodal with *most* of the density concentrated *around* the delay baseline $d_{\min}$ of the path (sum of propagation, transmission and processing delays). If there is a deviation from the delay baseline, the delay distribution will have additional modes: a *low* density mode around $d_{\min}$, and one or more modes higher than $d_{\min}$. The lowest mode in the delay sample's pdf is used as an estimate of the *baseline delay* for the window.

We use a nonparametric kernel smoothing density estimate [21] to find modes; with a Gaussian kernel (a continuous function) and the Silverman's rule of thumb for bandwidth[3] [21]. A continuous pdf enables us to locate modes (local maxima), the start and end points of a mode (local minima), and density inside a mode with a single pass on the pdf. The module also keeps track of the previous window's baseline for diagnosis of problems with duration longer than a window. To discard self-queueing effects in probing, the agent pre-processes the timeseries to check for probes sent less than $100\mu s$ apart.

We note that the delay detection algorithm has limitations; in particular, it may sometimes detect a long-term problem (minutes or longer) as multiple short-term problems. For example, a pathology such as a queue backlog (congestion) that persists for minutes may cause a level shift in delays – which could "shift" the baseline. The agent merges "problem" windows that are close to each other, and the operator may tune the window size to overcome this limitation. Our focus in this work, however, is on short-term performance problems.

Loss detection: Loss detection looks for *significant deviations from the baseline loss invariant condition* (2). Under normal conditions, the number of lost packets measured by monitors depends on several factors, including the cross traffic along the path, link capacities and the probing process. Since ISPs deploy low probing rates (e.g., 10Hz), Pythia looks at every lost probe. The loss detection algorithm marks a window as "problem" if the window contains at least one lost packet. Similar to delay detection, the agent merges "problem" windows close to each other into a single "problem" window.

Reordering detection: Reordering detection looks for *significant deviations from baseline reordering invariant condition* (3). The reordering module computes a *reordering metric R* for each 5s window of sequence numbers in received order, $\{n_1 \dots n_k\}$, based on the *RD* metric definition in RFC 5236 [9]. For each received packet i, it computes an *expected sequence number* $n_{\exp,i}$; $n_{\exp}$ is initialized to the lowest recorded sequence number in the timeseries, and is incremented with every received packet. If a sequence number i is lost, $n_{\exp}$ *skips* the value i (assume that the window starts with $i = 1$). A *reorder sum* is computed as: $\eta_{sum} = \sum_{i=1}^{k} \mid n_i - n_{\exp,i} \mid$. The reordering module estimates the number of reordered

[3]In case the bandwidth estimate is large, the delay distribution under the case of baseline deviation may be unimodal, but with a *large range* of delays under the mode.

packets in the window based on mismatch in the two sequence numbers: $\eta = \sum_{i=1}^{k} I\left[n_i \neq n_{\exp,i}\right]$. The reordering metric R for the window is defined as the ratio of the above (RFC 5236 [9]): $R = (\eta_{sum}/\eta)\, I\left[\eta \neq 0\right]$.

Note that R is zero if there was no reordering, and R increases with the *amount* of reordering on the path (i.e., both number of packets and how "far" they are reordered). Our goal is not to estimate the number of reordered packets (which may not have a unique solution), but to quantify the extent of reordering for a window of packets (and use R in diagnosis). The detection algorithm marks a window as "problem" if $R \neq 0$ for that window. The algorithm appends R to a reordering timeseries.

System sensitivity: Pythia provides a simple knob to the user to configure *sensitivity* of detection towards performance problems without asking the user for thresholds. This functionality reduces the number of problems that the system reports to the user, while ignoring relatively *insignificant* problems. Sensitivity is defined on a scale of one to 10, based on the fraction of delays higher than the baseline or the loss rate in the problem time window.

5 Diagnosis of Detected Problems

Diagnosis refers to the problem of finding the root cause(s) of a detected performance problem. The agent triggers diagnosis whenever it detects a (delay, loss or reordering) problem in the measurements of a path. The agent performs diagnosis by matching a problem timeseries with the performance pathology specifications, which network operators can extend using operational or domain knowledge. When the agent bootstraps, it generates diagnosis code from the specifications by building an efficient intermediate representation. We focus on the intermediate representation and code in this section.

A key aspect of diagnosis is to design algorithms that have low resource (CPU and memory) consumption, since the agent should not affect the measurement accuracy or probe timing at the monitor on which it runs. This becomes particularly important when tests for symptoms are resource intensive, or as the list of pathologies to test for gets large.

The diagnosis forest: A brute-force approach to diagnose a performance problem is to test the problem timeseries against *all* symptoms, and subsequently evaluate each pathology specification to find matching pathologies. This can be computationally expensive, since symptom tests could be expensive. Our goal is to *reduce the number of symptoms the agent tests for when diagnosing a problem.* An efficient way to do this is to build a decision tree from the pathology specifications. We evaluate the overhead of the algorithms in Section 8.

In order to generate diagnosis code, the agent generates an intermediate representation of the specifications: a *diagnosis forest*. A diagnosis forest is a forest of decision trees (or *diagnosis trees*). A diagnosis tree is an 3-ary tree with two types of nodes: the leaf nodes are pathologies, and rest of the nodes are symptoms. The tree edges are labeled either `true`, `false` or `unused`, depending on whether that symptom is required to be true or false, or if the symptom is not used. Hence, a path from the root node to a leaf node L in a diagnosis tree corresponds to a logical conjunction of symptoms for pathology L (specifically, symptoms that have non-`unused` edge labels).

Why not existing methods? There are several variants of decision tree construction methods such as the *C4.5* and *ID3*. These algorithms iteratively choose an attribute (symptom) based on the criteria of one that best classifies the instances (pathologies). They generate small trees by pruning and ignoring attributes that are not "significant". We found that existing construction methods are not suitable for the pathology specifications input for three reasons. First, pathologies may use only a small subset of symptoms (hence, we cannot treat `unused` as an attribute value in existing tree construction algorithms). In addition, not all outcomes of the symptom may be used in diagnosis; for example, Pythia does not include any pathologies which require the "loss exists" symptom to be `false`. Second, a pathology is required to be diagnosed using *all* symptoms in its specification (existing decision tree methods consider the *smallest* set of symptoms). Third, pathologies may exist *simultaneously* in an end-to-end path, and hence can be diagnosed in parallel (unlike a decision tree). Two pathologies can be diagnosed in parallel if both of their specifications match the problem timeseries.

Forest construction: The diagnosis forest is constructed in two steps. In the first step, the agent divides the set of pathologies $\mathscr{P}$ into disjoint subsets $\{P_1 \ldots P_k\}$, such that: (1) pathologies in each P_i use overlapping symptoms, and (2) sets P_i and P_j ($i \neq j$) do not use common symptoms. Since no two members of the set $\{P_1 \ldots P_k\}$ use overlapping symptoms, we can run tests for P_i and P_j independently, and potentially have multiple diagnoses for a problem.

In the second step, the agent constructs a decision tree for diagnosing members in each pathology subset P_i. The initial tree is constructed such that the root node is the symptom that is *most frequently* used by pathologies, and such that the frequency of symptom usage drops as we go towards the leaves (Fig. 2). At the end of this step, the trees will contain *all* symptoms that are required to diagnose each pathology. We may, however, have some symptoms with `unused` outgoing edge labels.

Finally, the tree construction algorithm prunes as many unused symptoms as possible in the decision tree(s). This consists of two rounds of pruning on each tree (Fig. 2; the leaves are pathologies, and shades show symptoms). First, we ensure that each symptom node has

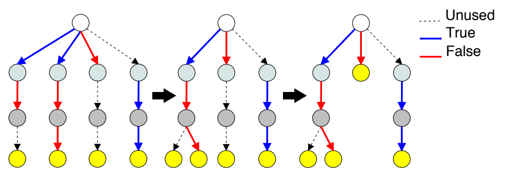


Figure 2: *Decision tree construction: pruning and merging.*

unique outgoing edge labels by *merging* edges with same labels. Second, we delete all edges with labels `unused` where feasible[4].

The default diagnosis rules configured in Pythia result in a forest of two trees – the first for diagnosis of delay and loss-based problems, and the second for diagnosis of reordering-based problems. The agent generates diagnosis code by traversing the decision forest. Each symptom node in a tree corresponds to a procedure call implemented by the agent for that symptom; and each leaf (pathology) node indicates a diagnosis output.

Unknowns: In practice none of the pathology definitions may match a detected problem timeseries. In such cases, the problem is tagged by the diagnosis logic as "unknown". For each "unknown" problem, the agent checks whether the problem originated from a monitor, by looking at diagnoses across monitored paths in the system (see Section 6.6). In our data, we have observed that less than 10% of problems are tagged as "unknown".

6 Common Diagnosis Specifications

We configure Pythia with a set of pathology specifications based on our domain knowledge of performance problems, and based on inputs from network operators. We expect that the network operator would add more based on her domain and operational knowledge of the network. Our goal behind the default choice of pathologies in Pythia is to provide the network operator with useful diagnosis information of the monitored networks.

In this section, we cover five classes of pathology specifications, and the statistical tests for matching the associated symptoms. We design boolean tests for symptoms by extracting salient and noise-resilient features of pathology models. The symptoms are defined over the measured timeseries for an end-to-end path - which includes delay, loss and reordering measurements. Table 1 lists the symptoms we test for. Some of the symptoms use domain knowledge-based thresholds, and can be fine-tuned by the operator.

Our network path model is as follows. An end-to-end path consists of a sequence of store-and-forward hops having a limited buffer size. We do not assume that links are work conserving, FIFO, or of a constant capacity (for example, 802.11 wireless links violate these assumptions). Monitoring hosts may induce delays, losses and "noise" in measurements.

[4]More specifically, for each edge $A \rightarrow B$, we delete A and move B upwards if (1) $A \rightarrow B$ has label `unused` (i.e., symptom A is unused in diagnosis of the sub-tree of A), and (2) B does not have any siblings.

6.1 End-host Pathologies

Our experience deploying Pythia on production monitors showed occurrence of short-term end-host pathologies – significant delays induced by effects inherent to commodity operating systems and userspace tools. Such pathologies occur at the monitoring nodes on which a Pythia agent runs. End-host pathologies may also refer to significant delays induced due to measurement application behavior (e.g., due to delays in timestamping or delays in sending probe packets). End-host effects may not be useful to the network operator; however, it is important for Pythia to identify and discard them, and not report them as pathological network delays[5].

End-host effects: A common artifact of commodity OSes is *context switches*. A busy OS environment may lead to significant packet *wait* delays at the sender and/or the receiver-side measurement end points. For example, these could be delays: *(i)* after a userspace `send()` call till packet transmission, or *(ii)* after the network delivers a packet to the OS until userspace `recv()` call (and corresponding timestamping).[6]

We can model an end-host induced delay symptom as follows. We model the buffered path from a measurement application to the NIC buffer (and the reverse path) as a single abstract buffer. Under normal conditions, this buffer services packets as they arrive from the measurement tool (or from the network). Under pathological conditions (e.g., when the OS is busy scheduling other processes or not processing I/O), the buffer is a non-work conserving server with "vacation periods". If a vacation period of W is induced while packet i is being served, successive packet arrivals will see steadily decreasing wait delays (T_i is the send timestamp of i):

$$d_{i+k} = \max\{W - [T_{i+k} - T_i], 0\} \quad (1)$$

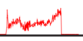
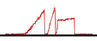
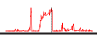
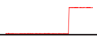
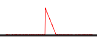
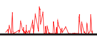
at the end-host (assuming the other packets do not see new vacation periods, and no other sources of delay variation). This behavior manifests in end-to-end delay measurement timeseries as a "triangular peak" symptom of height W, and the duration of this peak is also W.

It can be argued that a vacation period could be a burst of cross traffic that arrived in the inter-probe duration δ. We choose our threshold for W to avoid matching such cases. Suppose that a burst arrived at a rate λ at a link of

[5]An alternative approach is to tackle end-host pathologies by rewriting the monitoring tool to reduce OS-level noise; e.g., by running in kernel space. It is, however, not feasible to do this in production.

[6]Note that context switches may also occur in network devices due to wait periods when the OS resources are busy; in practice, the likelihood is much higher in end-hosts, since they may be running multiple resource intensive processes (other than measurement tools).

Table 1: *Default symptoms and their boolean-valued tests. Tests take input delay sample $\mathscr{D} = \{d_1 \dots d_n\}$, and the estimated baseline delay is b (both in ms). Tests work on a reordering metric timeseries $\mathscr{R} = \{R_1 \dots R_l\}$. $I(x)$ is the 0-1 indicator function, and $m(\dots)$ is the sample median. The default thresholds are tuned using empirical observations on perfSONAR data.*

Symptom	Sample	Boolean-valued Test
High delay utilization		More than 50% are *large* delays: $\sum_{i=1}^{n} I(d_i > b+1) > 0.5n$
Bursty delays		Largest delay hill duration less than 70% of delay hill duration
Extreme delay range		Very small: $\mathscr{D}_{0.95} - \mathscr{D}_{0.05} < 1$ms; very large: $\mathscr{D}_{0.95} - \mathscr{D}_{0.05} > 500$ms
Delay-loss correlation		For a lost packet i: $d_j > b + 1$ms, for majority of $j \in [i-5, i+5] - \{i\}$
Small loss duration		All packets between i and $j > i+1$ are lost, and $T_j - T_i > 1$s
Delay level shift (LS)		Estimate LS: first/last $d_i < b \pm 10$ms; 10% points before/after LS
Large triangle		Sudden rises in delay over 300ms: $\sum_{i=1}^{n} I(d_{i+1} - d_i > 300) < 0.1n$
Single-point peaks		Median of neighbors of i:$d_i > b + 1$ms $\approx$ median of d_j:$d_j \leq b+1$
Reordering shift		One-proportion test for: $\sum_{i=l/2+1}^{k} I\left(R_i > m(R_1 \dots R_{l/2})\right) = 0.5(l/2)$

capacity C. If the delay increase was *due to cross traffic*, we have the following condition for the queue backlog: $\left(\frac{\lambda - C}{C}\right)\delta \geq W$; in other words: $\lambda \geq \left(1 + \frac{W}{\delta}\right)C$. In our monitoring infrastructure, $\delta = 100$ms; so we can define a threshold for W by choosing an upper bound for the input-output ratio λ/C. We use a ratio of 4, giving us $W \geq 300$ms in case of an end-host pathology.

Depending on the magnitude of the vacation period W, we can have two end-host pathology symptoms. First, if W is of the order of 100s of milliseconds (e.g., when the receiver application does not process a probe in time), we will observe a "large triangle" delay signature (see Table 1), described by Equation 1[7]. Second, if W is much smaller – of the order of 10s of milliseconds (typical duration of context switches in a busy commodity OS) – and if the inter-probe gap is close to W, the delay symptom is a set of "single-point peaks": delay *spikes* that are made of a single (or few) point(s) higher than the baseline delay. The number of spikes is a function of the OS resource utilization during measurement.

6.2 Congestion and Buffers

Network congestion: We define congestion as a *significant* cross traffic backlog in one or more queues in the network for an extended period of time (few seconds or longer). Pythia identifies two forms of congestion based on the backlogged link's queueing dynamics. First, congestion *overload* is a case of a *significant and persistent* queue backlog in one or more links along the path. Overload may be due to a single traffic source or an aggregate of sources with a persistent arrival rate larger than the serving link's capacity. The congestion overload specification requires a high *delay utilization* above the baseline, and a traffic aggregate that is *not bursty*.

[7] We assume that the inter-probe gap is much lower than such W.

Second, *bursty* congestion is a case of a significant backlog in one or more network queues, but where the traffic aggregate is bursty (i.e., high variability in the backlog). We use the term "bursty" to refer to a specific backlog condition that is *not persistent*. The bursty congestion specification requires a high *delay utilization*, and "bursty" delays, i.e., the timeseries shows multiple delay *hills* each of reasonable duration. Both congestion specifications require that the timeseries does not show end-host pathology symptoms.

Buffering: A buffer misconfiguration is either an over-provisioned buffer or an under-provisioned buffer. Over-buffering has the potential to induce large delays for other traffic flows, while under-buffering may induce losses (and thus degrade TCP throughput). Pythia diagnoses a path as having a buffer that is either over- or under-provisioned based on two symptoms. First, the delay *range* during the problem is either too large or too small. Second, an under-provisioned buffer diagnosis requires that there is *delay-correlated* packet loss during the problem (see Section 6.3). We do not make any assumption about the buffer management on the path (RED, DropTail, etc.). We choose thresholds for large and small delay ranges as values that fall outside the typical queueing delays on the Internet (the operator can tune the values based on knowledge of network paths).

6.3 Loss Events

Random / delay-correlated loss: Packet losses can severely impact TCP and application performance. It is useful for the operator to know if the losses that flows see on a path are *correlated* with delays - in other words, delay samples in the neighborhood of the loss are larger than baseline delay. Examples of delay-correlated losses include losses caused by buffering in a network device, such as a full DropTail buffer, or a RED buffer over the minimum backlog threshold.

On the other hand, *random losses* are probe losses which do not show an increase in neighborhood delays. Random losses may be indicative of a potential physical layer problem, such as line card failure, bad fiber or connector, or a duplex mismatch (we defined these based on operator feedback [2]). In theory, a random loss is the conditional event: P[loss | delay increase in neighborhood] = P[loss], where the *neighborhood* is a short window of delay samples. In practice, we may not have sufficient losses during the problem to estimate the probabilities; hence we look at the delay neighborhood[8]. Our loss-based pathologies may not have one-to-one correspondence with a root cause; however, operators have found them useful in troubleshooting [2].

Short outage: Network operators are well aware of long outages. They may overlook infrequent *short* outages, possibly caused by an impending hardware failure. A short outage can disrupt existing communications. Pythia diagnoses loss bursts that have a small loss duration (one to few seconds) as short outages.

6.4 Route Changes

The symptom of a route change is a *level shift* in the delay timeseries (and possibly, packet loss during the level shift). Routing events could be either long-term route changes, or route flaps. Pythia currently diagnoses long-term route changes. It does so by finding significant changes in the baseline and in the propagation delay. Note that delay level shifts can also occur due to clock synchronization at the monitors; Pythia currently reports delay level shifts as "either route change or clock synchronization". We do not support identification of clock synchronization events[9].

6.5 Reordering Problems

Reordering may occur either due to a network configuration such as per-packet multi-path routing or a routing change; or it could be internal to a network device (e.g., switching fabric design in high-speed devices) [4]. Reordering may not be a pathology, but it can significantly degrade TCP performance. If it exists, reordering will be either *persistent and stationary* (e.g., due to a switching fabric or multi-path routing), or *non-stationary* (e.g., routing changes).

Pythia diagnoses the above two types of reordering. The detection logic computes a reordering metric R for each time window of measurements (Section 4). R is zero if there is no reordering, and it increases with the *amount* of reordering on the path. Pythia diagnoses reordering non-stationarity by looking at the set of 10 most recent reordering measurements. Pythia uses the "reordering shift test" to diagnose non-stationarity (or stationarity) in reordering (see Table 1).

[8]For a loss burst, Pythia considers delays before and after the burst.

[9]Identification of clock sync. events is an expensive operation. It can be done, for example, at a monitor M by correlating delays from all timeseries destined to or starting at M; if there is a clock sync, *all* timeseries will show a level shift at about the same time.

6.6 Unknowns

In practice, there may be detected problems that do not match any of the input pathology specifications. We call such problems as "Unknown" problems. When an agent finds that a problem is unknown, it performs an additional check across the monitored network to diagnose if the problem is a result of an end-host (monitor) pathology. Specifically, the agent checks whether *a significant number of paths ending at/starting from its monitor show a performance problem at the same time*. It does this by querying the Pythia database. The agent tags all unknown problems as end-host pathologies if a majority of paths were diagnosed as having an "Unknown" or an end-host problem.

7 Live Deployment

Many wide area ISPs consist of geographically distributed networks connected using inter-domain paths. Monitoring infrastructure in such ISPs consists of nodes in the constituent networks. For example, the US Department of Energy's Energy Sciences Network (ESnet), a wide area research backbone, connects several stub networks, each hosting several perfSONAR monitors.

We deploy Pythia on a wide area perfSONAR monitoring infrastructure that spans seven ESnet monitors across the US, a Georgia Tech monitor and monitors in 15 K-12 school district networks in GA, USA[10]. The current deployment uses the default corpus of 11 performance pathology definitions, some of which were formulated based on ESnet and Internet2 operational experience. We are in the process of expanding Pythia to several monitors in ESnet, Internet2 and other networks.

Our live deployment showed some interesting patterns from K-12 networks. We found using Pythia that in a typical week, about 70% of network-related pathologies are related to congestion, leading to packet losses. In particular, about 29% of the problems are due to traffic burstiness. Pythia also found that about 5% of the problems are packet losses not related to buffering, which may be due to physical layer-related problems. Moreover, Pythia's localization shows that almost 80% of the network interfaces are affected by one or more performance problems. Pythia also showed diurnal and weekday-related congestion patterns. Pythia's findings confirm with a prior study on the same K-12 networks in 2010 [18]. We use monitoring data to do a large-scale study of pathologies in Section 9.

[10]The infrastructure is a part of the Georgia Measurement and Monitoring (GAMMON) project, which aims to assess the feasibility of online learning requirements of Georgia Department of Education.

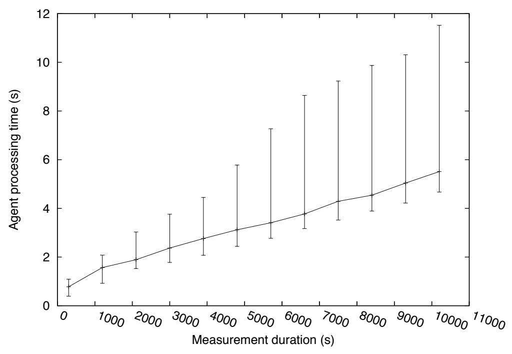


Figure 3: *Agent run time vs. input measurement duration: median, 5th and 95th percentile across 100 sample measurement files, excluding database commit times.*

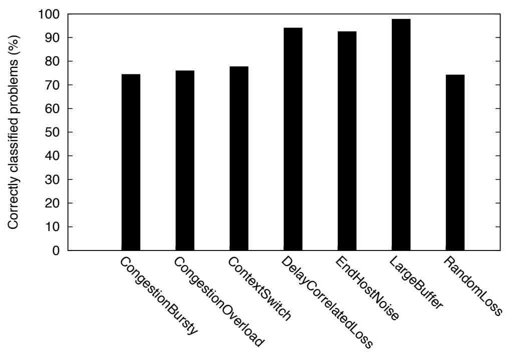


Figure 4: *Validation of diagnosis logic: classification accuracy of different diagnoses. Datasets include academic, research, commercial and residential broadband networks.*

8 Monitoring data and Validation

In the remainder of the paper, we look at how the different performance pathologies that Pythia diagnoses (§6) manifest in core and edge networks in the Internet. In order to do this, we collect data from production monitoring deployments in two backbone networks and leverage a popular measurement tool run by home users; we run the agent on the data. We collect data since we use it for offline validation and accuracy estimation.

Datasets: We collect data from production wide area monitoring deployments in backbone networks, and we build our own monitoring infrastructure to collect data from edge networks. We use four sources of data:

Backbone: This is data from production perfSONAR monitoring infrastructure in ESnet (33 monitors, 12 days) and Internet2 (9 monitors, 22 days). This data represents high-capacity, wide area inter-domain networks.

Residential: We use residential user-generated data from ShaperProbe [11]. ShaperProbe includes a 10s probing session in upstream and downstream directions. We consider 58,000 user runs across seven months in 2012.

PlanetLab: We build and deploy a full-mesh monitoring infrastructure similar to perfSONAR, using 70 nodes. We collect data for 12 hours in March 2011. We monitor commercial and academic networks.

Our data comes from 40 byte UDP probes, with an average sampling rate of 10Hz per path. OWAMP (perfSONAR) uses a Poisson process, while ShaperProbe and PlanetLab tools maintain a constant packet rate. All monitored networks have wide area inter-domain paths.

Agent overhead: We first measure the run time of detection and diagnosis algorithms in the agent. Figure 3 shows the agent run time as a function of input size – duration of measurements for a path – on a 3GHz Xeon processor. We use 100 randomly picked measurement timeseries from ESnet and Internet2. On an average, the agent takes about $60\mu s$ to process a single e2e measurement. Hence, the agent can handle large monitoring deployments (i.e., several measurement paths per monitor).

Validation: It is hard to validate an inter-domain diagnosis method, since it is not feasible to get systematic "ground truth" diagnoses (labeled data) of performance problems across the network. This is further complicated by our focus on *(i)* short-term problems which are typically unnoticed by network operators, and *(ii)* wide area ISPs, which include problems from multiple networks and inter-domain links.

In order to overcome paucity of labeled data, we use manually classified pathology data. We first run Pythia to find and classify performance problems in the data into different pathology types. For each of the four data sources, we choose a uniform random sample of 10 problems from each pathology type. We select a total of 382 problems for manual classification. Note that we would not be able to evaluate the false negative detection rate; it is infeasible to find problems that go undetected given the size of our data. Our detection methods, however, are not based on symptoms, and hence do not introduce systematic biases towards/against certain pathologies.

We manually observe the delay and loss timeseries of each problem and classify it into one or more pathologies (or as "Unknown"). For each problem, we mark as many valid (matching) pathologies as we see fit. We consider this as our ground truth. This approach has limitations. First, a high delay range in the timeseries (e.g., due to few outliers) may visually *mask* problems that occur over a smaller delays. Second, if the problem is long-duration, it may be hard to visualize small timescale behavior (e.g., a small burst of packet losses will be visualized as a single loss). Third, if there are unsupported diagnoses (e.g., short-term delay level shifts), the matching pathology definitions would be wrong, while the ground truth would be correct. Finally, the ground truth generation does not include "cross-path" checks that Pythia uses. An example of such checks is for diagnosing end-host pathologies in "Unknown" problems (§6.6).

To validate, we compare the diagnoses generated by Pythia for each problem with the manually classified di-

Table 2: *Occurrence of non-network problems in different datasets. Acronyms: "E.H.N." - "EndHostNoise", "C.S." - "ContextSwitch", "Unk." - "Unknown".*

Dataset	# Problems	E.H.N.	C.S.	Unk.
ESnet	465,135	52%	43%	3%
Internet2	18,774	1%	3%	13%
PlanetLab	718,459	56%	16%	1%
ShaperProbe	8,790	54%	9%	9%

agnoses. We did not find any false positive detections in manual inspection. We define "accuracy" as follows[11]. We ignore problems that were manually classified as "Unknown". We show two measures of diagnosis accuracy. For each problem:

- A diagnosis is correct if *at least one* of Pythia's diagnoses exists in the manually classified diagnoses. The diagnosis accuracy across all problems is 96%.
- A diagnosis is correct if *all* of Pythia's diagnoses exist in the manually classified diagnoses. The diagnosis accuracy across all problems is 74%. There were 1.42 diagnoses per problem on average.

Figure 4 shows validation accuracy for each pathology; we ignore pathologies for which we have less than 35 instances. A diagnosis for a problem is marked as "correct" if it exists in the ground truth. The "CongestionOverload" and "CongestionBursty" diagnoses include short-term delay level shifts; these are false diagnoses, since Pythia does not support them, and they comprise 42% of the total false diagnoses of the two congestion types.

9 Case Studies of Networks

In this section, we use Pythia to study performance problems in the Internet. We use measurements from production monitoring deployments in backbone and edge networks (see §8 for details of the data).

9.1 Monitor-related problems

Before we look at network pathologies, we study end-host (monitor) pathologies across different monitoring infrastructures. Our data comes from a wide variety of monitor platforms: Linux-based dedicated servers (ESnet and Internet2), desktop and laptops running different OSes at homes (ShaperProbe), and virtual machines (PlanetLab). Table 2 shows the fraction of detected problems that are diagnosed as end-host pathologies. We show frequencies separately for the two forms of end-host pathologies: "EndHostNoise" (short wait periods) and "ContextSwitch" (long wait periods).

The table shows an interesting feature that validates known differences between monitor types. The Internet2

[11]Our diagnosis is a multiclass multilabel classifier that includes the "Unknown problem" output. It is not straightforward to define precision and recall in this case. We define accuracy as fraction of classified samples that are not unknown and are "correct".

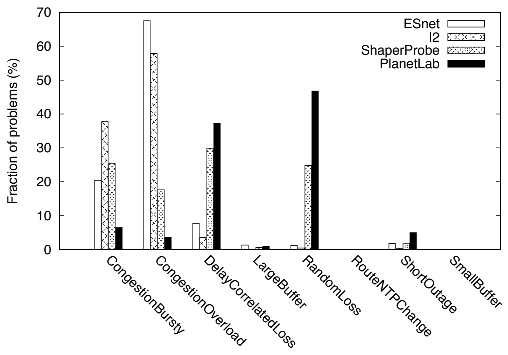


Figure 5: *Breakdown of network-based pathologies among the four datasets (omitting "Unknown" and end-host pathologies).*

monitors are dedicated resources for measurement tools, while the ESnet monitors also run MySQL databases for indexing measurement streams. Hence, Internet2 data shows a smaller fraction of end-host-related pathologies than ESnet, since the OS environment is more likely to be busy in ESnet monitors. The PlanetLab environment is also likely to be busy, given that the resources are shared among virtual machines. ShaperProbe data comes from a userspace tool running on commodity OSes, and where the users run other processes such as web browsers.[12]

The table also shows that the fraction of problems that Pythia diagnoses as "Unknown" are typically lower than 10%. In the rest of this section, we focus on network-related problems (i.e., excluding end-host and "Unknown" pathologies).

Implications: Production monitoring infrastructure is dedicated to measurements, and hence is not expected to induce large delays in measurements (other than monitor downtime). While this may be true for long-term monitoring (minutes to hours), we find that when the focus is on short-lived problems, we see a nontrivial proportion of monitor-induced delays. It hence becomes important to diagnose and separate such problems.

9.2 Core vs. Edge networks

We look at the composition of network-related pathologies in the different networks in Figure 5. We see that the high-capacity backbone networks, ESnet and Internet2, show a high incidence of congestion pathologies (both overload and bursty congestion). Moreover, there is no significant difference in the composition of pathologies between the two (similar) core networks.

The residential edge (ShaperProbe) shows a high incidence of both congestion and loss pathologies. Note that we do not see a significant fraction of "LargeBuffer" pathologies in home networks, though the presence of large buffers in home networks has been shown before

[12]We note that "ContextSwitch" problems in ShaperProbe may include problems arising from either end-hosts or 802.11 links inside the home (Section 6.1). Separating end-host pathologies in ShaperProbe data allows us to focus on the ISP access link.

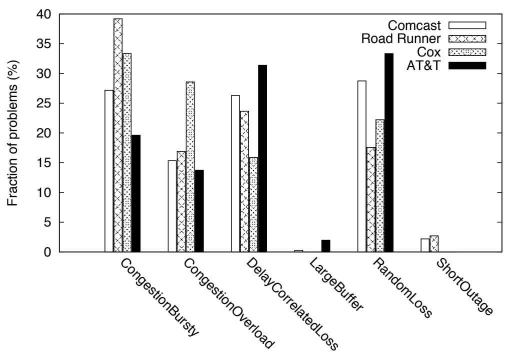


Figure 6: *Composition of different network-based pathologies among the four residential ISPs (both up and downstream).*

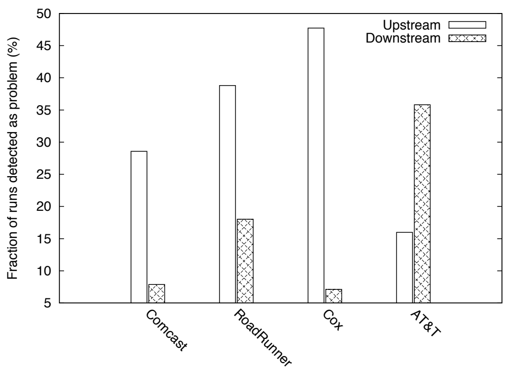


Figure 7: *Residential broadband networks: fraction of runs detected as pathology in upstream and downstream directions, for three cable and one DSL provider.*

[12]. This is because Pythia can diagnose a large buffer problem only if cross traffic creates a significant backlog in the buffer during the 10s probing time. PlanetLab data shows a high incidence of loss pathologies, but not congestion. The pathologies include delay-correlated and random losses, as well as short outages.

Implications: The results show differences in problems between backbone and edge networks. Despite the presence of large buffers, home networks are prone to loss-based problems, which can significantly degrade performance of web sessions. This may be due to 802.11 wireless links inside homes. A real time e2e diagnosis system can help quickly address customer trouble tickets.

9.3 Residential network pathologies

We look at four large residential broadband providers in our dataset: cable providers Comcast, Road Runner and Cox; and DSL provider AT&T. The number of network problems in our data depends on the number of runs ShaperProbe receives from ISP users (problems per link ranged from about 250 in AT&T to 12,000 in Comcast). Note that ShaperProbe data has an inherent "bias" – the tool is more likely to be run by a home user when the user perceives a performance problem.

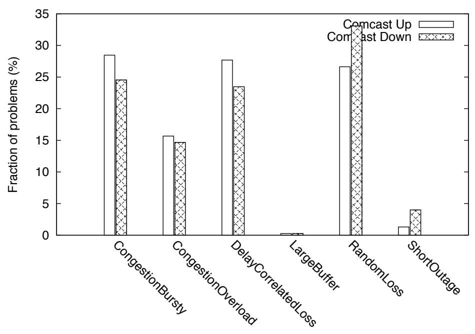


Figure 8: *Composition of different network-based pathologies in Comcast, as a function of link direction.*

Figure 7 shows fraction of user runs showing performance problem in the upstream and downstream directions for the four ISPs. We use runs which had recorded over 50% of expected number of measurement samples in 10s (at 10Hz). We see a difference in the frequency of problems in upstream and downstream directions between cable and DSL providers. A plausible explanation is that in the case of cable, DOCSIS uplink is a non-FIFO scheduler, while the downlink is multiplexed with neighborhood homes; DSL uses FIFO statistical multiplexing in both directions, but the link capacity is relatively more asymmetric. To cross-check the problem detection numbers, we look at the difference between 95th and 5th percentiles of the delay distribution during the problem. We find that about 59% upstream and 25% downstream Comcast runs have a difference exceeding 5ms (the default delay detection threshold); while for AT&T, the figures were 45% and 63% respectively.

Figure 6 shows composition of performance pathologies in each of the four ISPs. We see that the DSL provider AT&T shows a higher incidence of loss pathologies than the cable providers (both delay-correlated and random loss pathologies).

We next look at whether cable links show different pathology distributions among problems in the upstream and downstream directions. Figure 8 shows composition of pathologies across runs from Comcast. We do not see a significant difference in the composition of pathologies, even though there are more problem detections in the cable upstream than downstream.

Implications: We see that within residential ISPs the nature and composition of performance pathologies varies. Hence, we cannot build a one-size-fits-all system for residential ISPs; the system should allow operators to input domain knowledge-based pathologies.

10 Discussion and Limitations

In this paper, we presented Pythia, a system for detection and diagnosis of performance problems in wide area providers. Prior work has taken two key approaches to performance diagnosis in ISPs (see §11). The first approach involves designing and deploying *new probing tools* that diagnose specific pathologies. This enables detailed diagnosis, but new tools add probing overhead. The second approach involves *mining network-wide data*, from every network device and other data such as trouble tickets. This enables detailed diagnosis as well, but does not work in wide area ISPs, where paths can be inter-domain.

We explore a new complimentary approach that works with existing monitoring infrastructure and tools, and works in the inter-domain case. Pythia uses diagnosis rules, which are defined as logical expressions over problem symptoms. Operators can add these rules to the system using a specification language. This design enables ISPs to *incrementally deploy diagnosis* in production: not only when adding new monitors to the network, but also as and when *new* performance problems are seen by operators. At the same time, this approach eliminates the need for training data, which is hard to have in wide area ISPs. Pythia provides real time diagnosis summaries by using non-invasive detection and diagnosis algorithms at the monitors. In the course of building Pythia, we have noted some limitations and considerations when building and deploying the system.

Monitoring: Pythia relies on existing monitoring in the ISP. This could mean that the *diagnosis may be limited by probing* (e.g., probing frequency). For example, our results on home networks show a low incidence of large buffer-related problems, since not all large buffer delay increases may not sampled by the probing process.

Specifications: Language specifications of diagnosis rules may have limitations, despite the flexibility that they offer. When adding diagnosis rules, operators need to consider the tradeoff between specificity and generality of new diagnosis rules relative to existing rules – in particular with large number of rules. Further, it may not be feasible to specify some pathologies, since the monitored feature set, and hence the symptom set, is limited.

Sensitivity: In practice, Pythia's detection logic can lead to a large number of reported pathologies. We address this by including a knob that allows the operator to tune *sensitivity*, defined as the magnitude of deviation from the baseline (§4). We leave it to future work to rank pathologies based on operator interest and criticality.

Symptoms: The symptoms used in diagnosis rules may be based on models of performance (e.g., the end-host class), or may be based on static thresholds. We note that symptoms based on static thresholds may be common in practice, since they are likely to be based on operator experience. Since these thresholds could change with time, an open feature in Pythia is to extend the specification language to support threshold declarations.

Pathologies: Our deployment experience has shown that signatures induced by monitors may be common in practice when the focus of diagnosis is on short-lived problems. We leave open the analysis of problems that Pythia finds "Unknown". Pythia could augment these with a *similarity* measure over a specified space of features. Similarity compares the problem against a representative set of diagnoses to find the most similar diagnosis.

11 Related work

There has been significant prior work on detection and diagnosis of performance problems. The prior work falls into two classes of design. We present representative work in each class.

Data-oriented methods: These are diagnosis methods that use significant amount of data sources that are usually available in enterprises and single administrative domains, but *not* in wide area inter-domain settings. These methods give detailed diagnosis; there is a tradeoff, however, between how *detailed* the diagnosis can be and the wide-area applicability (*generality*) of a diagnosis method. A summary of some of these methods follows. AT&T's G-RCA [24] works on data from a single network such as SNMP traps, syslogs, alarms, router configurations, topology and end-to-end measurements. It mines dependency graphs from the data and constructs diagnosis *rules* from the graphs. SyslogDigest [19] mines faults from router syslogs. NetMedic [10] and Sherlock [3] diagnose faults in enterprise settings by profiling end-host variables and by mining dependencies from historic data. NICE [16] enables troubleshooting by analyzing correlations across logs, router data and loss measurements. META [23] looks at spatial and temporal features of network data to learn fault signatures. A recent study [22] uses email logs and network data to analyze routing-based failures. Learning methods may require prior training; however, they can complement Pythia by classifying problems that cannot be diagnosed using domain knowledge.

Data-oriented methods have also been used to diagnose specific pathologies in the context of a single network. Feather et al. look at diagnosing soft failures in a LAN using domain knowledge on a pre-defined set of features [5]. We take a similar approach to diagnosis using domain knowledge, but in the more general wide area inter-domain context. Lakhina et al. used unsupervised clustering methods on packet-level features in packet traces to classify performance anomalies [14]. Huang et al. use structural properties of packet traces to detect performance problems in a LAN [7]. Huang et al. identified inter-domain routing anomalies using BGP updates [8]. There has been extensive work on structural methods to detect anomalies in volume data in an ISP; for example, the influential work by Lakhina et al. uses dimensionality reduction on volume data [13].

Active probing methods: These methods rely on active probing, and are typically based on domain knowledge. They can reveal detailed and accurate diagnosis, since they provide the choice of carefully crafting probing structures. It is, however, hard to widely deploy a new active probing tool in a large monitoring network, especially if some of the monitors are in other ASes. We summarize a few representative tools below. Netalyzr [12] probes to help a user with troubleshooting information. Tulip [15] diagnoses and localizes reordering, loss and congestion using a single end-point. PlanetSeer [26] uses a combination of active and passive methods to monitor path failures. Prior work designed probing tools for specific diagnoses such as Ethernet duplex mismatch [20] and buffering problems [17].

12 Conclusion

In this paper, we have designed a performance problem detection and diagnosis system for wide area ISPs, that works in conjunction with deployed monitoring infrastructure. Pythia only requires a lightweight agent process running on the monitors. We have designed efficient detection and diagnosis algorithms that enable such an agent without affecting measurements. Pythia provides an expressive language to the operator to specify performance pathology definitions based on domain knowledge. We have deployed Pythia in monitoring infrastructure in wide area ISPs, diagnosing over 300 inter-domain paths. We used Pythia to study performance pathologies in backbone and edge networks. Our experience with Pythia has shown that existing monitoring infrastructure in ISPs is a good starting point for building near real time wide area problem diagnosis systems, enabling incremental diagnosis deployment.

Acknowledgements: We thank Jason Zurawski from Internet2, and Joe Metzger, Brian Tierney, and Andrew Lake from ESnet for providing us with OWAMP data sets. We thank Warren Matthews for his work on Pythia deployments. We are grateful to the anonymous reviewers for their valuable comments. This research was supported by the U.S. Department of Energy under grant DE-FG02-10ER26021.

References

[1] *Deployments of Network Monitoring Software perfSONAR Hit 1,000*. http://1.usa.gov/Qt94Nk.

[2] *perfSONAR Deployment on ESnet*. Brian Tierney. Presented at AIMS Workshop, CAIDA. 2011.

[3] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. Maltz, and M. Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *ACM SIGCOMM CCR*, volume 37, pages 13–24, 2007.

[4] J. C. R. Bennett, C. Partridge, and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM ToN*, 7(6):789–798, Dec. 1999.

[5] F. Feather, D. Siewiorek, and R. Maxion. Fault detection in an Ethernet network using anomaly signature matching. In *ACM SIGCOMM CCR*, 1993.

[6] A. Hanemann, J. Boote, E. Boyd, J. Durand, L. Kudarimoti, R. Łapacz, D. Swany, S. Trocha, and J. Zurawski. PerfSONAR: A service oriented architecture for multi-domain network monitoring. *Service-Oriented Computing*, 2005.

[7] P. Huang, A. Feldmann, and W. Willinger. A non-intrusive, wavelet-based approach to detecting network performance problems. In *ACM SIGCOMM IMW*, 2001.

[8] Y. Huang, N. Feamster, A. Lakhina, and J. Xu. Diagnosing network disruptions with network-wide analysis. In *ACM SIGMETRICS PER*, volume 35, pages 61–72, 2007.

[9] A. Jayasumana, N. Piratla, T. Banka, A. Bare, and R. Whitner. Improved Packet Reordering Metrics (RFC 5236), June 2008.

[10] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl. Detailed diagnosis in enterprise networks. In *ACM SIGCOMM CCR*, volume 39, pages 243–254, 2009.

[11] P. Kanuparthy and C. Dovrolis. ShaperProbe: end-to-end detection of ISP traffic shaping using active methods. In *ACM SIGCOMM IMC*, 2011.

[12] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: illuminating the edge network. In *ACM SIGCOMM IMC*, 2010.

[13] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM CCR*, volume 34, pages 219–230, 2004.

[14] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM CCR*, volume 35, pages 217–228, 2005.

[15] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. User-level Internet path diagnosis. In *ACM SIGOPS OSR*, volume 37, pages 106–119, 2003.

[16] A. Mahimkar, J. Yates, Y. Zhang, A. Shaikh, J. Wang, Z. Ge, and C. Ee. Troubleshooting chronic conditions in large IP networks. In *ACM SIGCOMM CoNEXT*, 2008.

[17] M. Mathis, J. Heffner, P. O'Neil, and P. Siemsen. Pathdiag: automated TCP diagnosis. *PAM*, 2008.

[18] R. Miller, W. Matthews, and C. Dovrolis. Internet usage at elementary, middle and high schools: a first look at K-12 traffic from two US Georgia counties. In *PAM*, 2010.

[19] T. Qiu, Z. Ge, D. Pei, J. Wang, and J. Xu. What happened in my network: mining network events from router syslogs. In *ACM SIGCOMM IMC*, 2010.

[20] S. Shalunov and R. Carlson. Detecting duplex mismatch on Ethernet. *PAM*, 2005.

[21] B. Silverman. *Density Estimation for Statistics and Data Analysis*. Monographs on Statistics and Applied Probability. Taylor & Francis, 1986.

[22] D. Turner, K. Levchenko, A. Snoeren, and S. Savage. California fault lines: understanding the causes and impact of network failures. In *ACM SIGCOMM CCR*, volume 40, pages 315–326, 2010.

[23] T. Wang, M. Srivatsa, D. Agrawal, and L. Liu. Learning, indexing, and diagnosing network faults. In *ACM KDD*, 2009.

[24] H. Yan, L. Breslau, Z. Ge, D. Massey, D. Pei, and J. Yates. G-RCA: a generic root cause analysis platform for service quality management in large IP networks. In *ACM SIGCOMM CoNEXT*, 2010.

[25] S. Zarifzadeh, G. Madhwaraj, and C. Dovrolis. Range tomography: Combining the practicality of boolean tomography with the resolution of analogue tomography. In *ACM SIGCOMM IMC*, 2012.

[26] M. Zhang, C. Zhang, V. Pai, L. Peterson, and R. Wang. PlanetSeer: Internet path failure monitoring and characterization in wide-area services. USENIX OSDI, 2004.