Learning, Indexing, and Diagnosing Network Faults

Ting Wang[†] Mudhakar Srivatsa[‡] Dakshi Agrawal[‡] Ling Liu[†]
[†]College of Computing, Georgia Institute of Technology, Atlanta, GA
[‡]IBM T.J. Watson Research Center, Hawthorne, NY
[†]{twang, lingliu}@cc.gatech.edu [‡]{msrivats, agrawal}@us.ibm.com

ABSTRACT

Modern communication networks generate massive volume of operational event data, e.g., alarm, alert, and metrics, which can be used by a network management system (NMS) to diagnose potential faults. In this work, we introduce a new class of indexable *fault signatures* that encode temporal evolution of events generated by a network fault as well as topological relationships among the nodes where these events occur. We present an efficient learning algorithm to extract such fault signatures from noisy historical event data, and with the help of novel space-time indexing structures, we show how to perform efficient, online signature matching. We provide results from extensive experimental studies to explore the efficacy of our approach and point out potential applications of such signatures for many different types of networks including social and information networks.

Categories and Subject Descriptors

C.2.3 [**Computer-communication Networks**]: Network Operations—*Network monitoring*; H.2.8 [**Database Management**]: Database Applications—*Data mining*

General Terms

Algorithms, Experimentation

Keywords

Network topology, fault signature, online diagnosis

1. INTRODUCTION

The motif of networks is ubiquitous in our lives [4]. In its simplest form, a network can be modeled as a graph where the vertices of the graph represent *network entities* and the edges represent pairwise interactions between network entities. It turns out that the simple, local, pairwise interactions between network entities can give rise to complex, interesting global phenomena [4]. Models of such global phenomena as a function of local interactions is one of the key issues being investigated in the area of networks. In this work, we propose a new class of models suitable for learning, indexing, and diagnosing a wide range of network phenomena while focusing on faults in the communication networks to exemplify our techniques.

Large communication networks have hundreds of thousands of network entities, and they are typically managed by a centralized network management system (NMS) that collects (local) monitoring data from network entities to diagnose network faults. When a fault occurs at a network entity, it tends to influence the "neighboring" entities. Consequently, faults often results in a large burst of messages being sent to the NMS from the affected entities. Each message contains a timestamp, an identifier of the affected device, and a *type* that signifies an event at the affected device[1]. The goal of NMS is to correlate the events occurring in the whole network, and identify the root-cause fault event(s), suppress dependent events, and discard routine operational events.

A key challenge faced by today's NMS is that of scalability. All widely deployed NMSes maintain a cache of "unresolved" events, and as each new event arrives, they use a rule-based mechanism to correlate the incoming event with all cached events to suppress dependent events and retain only the (unfixed) root-cause events in the cache. The resulting computation complexity is quadratic in network size since the number of unresolved events in the cache as well as event arrival rate is typically proportional to the network size.

Events triggered by a fault are typically generated by a small, constant-size subset[2] of nodes that are topologically related to the faulty node. Thus, each event arriving at the NMS only needs to be correlated with a small, constant-size subset of events in the cache, yielding a linear complexity of event correlation. Intuitively, achieving the linear complexity would require data structures that encode and exploit network topology. In this paper, we propose a framework META (Monitoring network Events with Topology Assistance) that, to the best of our knowledge, is the first proposal to utilize topologically-aware event patterns to perform *scalable* network fault diagnosis.

[1]Henceforth, we will use the words "message" and "event" interchangeably, while ignoring the semantic distinction that events occur at network entities resulting in messages that are received by the NMS for diagnosis.

[2]The size of this subset depends on the degree distribution, etc., of the network and is independent of the size of the network.

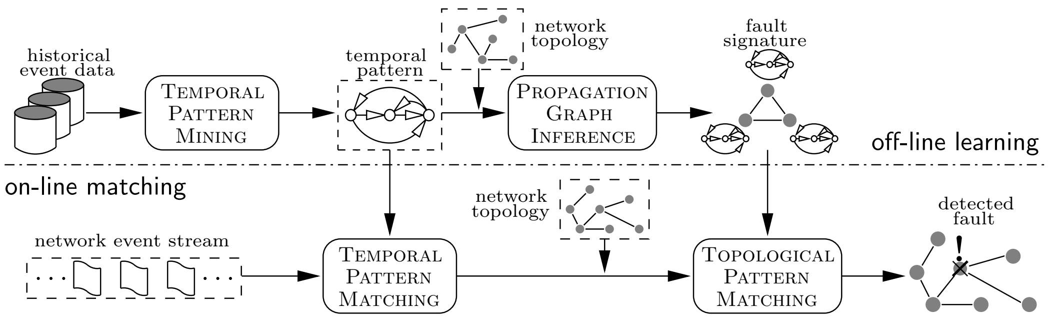

Figure 1: Illustration of the main architecture of Meta.

The remainder of the paper will be organized as follows. Section 2 formalizes the problem of network fault diagnosis; Section 3 and Section 4 describe in detail the offline learning and the online matching components of Meta, respectively (see Figure 1). An empirical analysis of our approach is presented in Section 5. Section 6 discusses related work. The paper is concluded in Section 7.

2. PROBLEM FORMULATION

This section introduces fundamental concepts and notations used in the paper, and formalizes the problem of online network fault detection and localization.

Definition 1 (Network). *A* **network** *is modeled as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V}$ representing the set of nodes, each corresponding to a network entity*[3]*, and $\mathcal{E}$ representing the set of edges over $\mathcal{V}$, each corresponding to a network link.*

Definition 2 (NMS/Agent/Sink). *A* **network management system** (NMS) *consists of a set of monitoring* **agents** *and a* **sink***. The agents are deployed on various network entities to collect monitoring data, and send them to the sink which is responsible for diagnosing potential faults.*

Definition 3 (Network Event). *Each event x is a tuple of the form $x = \langle e, v, t \rangle$, where e represents the event type, v the network node generating the event, and t the timestamp of the event.*

In the rest of this paper, we use e_x, v_x, and t_x to denote the type, entity, and timestamp of an event x, respectively.

Definition 4 (Event Stream). *We model the event stream as a sequence of events, $(x_1, \ldots, x_n)$, where x_n is the most recent event. Real-time fault diagnosis focuses on events occurring within a time window of length ω, i.e., an event subsequence $(x_i, x_{i+1}, \ldots, x_n)$, with $t_{x_n} - t_{x_i} \leq \omega$ and $t_{x_n} - t_{x_{i-1}} > \omega$.*

Now we are ready to formally define the problem of real-time network fault detection and diagnosis:

Problem Definition. *For each time-window, by analyzing the events occurring within the window and exploiting network topological information, detect potential faults (if any) and identify the fault types and failed entities.*

3. OFFLINE LEARNING

In this section, we describe in detail our method of distilling the essential features of network faults manifested in network events and composing them into compact, indexable fault signatures. The signatures are designed to capture two critical aspects of network phenomena, namely, the *temporal evolution* of fault-triggered events, and the *topological relationship* of nodes associated with fault-triggered events. Furthermore, signatures are *universal* for a class of networks; that is, signatures learnt using data from one network instance can be used to diagnose faults in other network instances in the same class.

The feature learning process can be roughly divided into three phases. First, from the noisy historical data, we identify event subsets that correspond to network faults (with high probability) to train our feature extractor. Second, we extract temporal patterns embedded in the training data by extending the classical *expectation maximization* (EM) framework. Third, we combine the discovered temporal patterns with the topological information available to form compact fault signatures that are amenable to efficient indexing and matching. The details of the three phases are presented in Section 3.1, 3.2, and 3.3, respectively.

3.1 Preparation of Training Data

For the purposes of training fault signatures, it is necessary to separate events caused by faults and those triggered by regular network operations. In Meta, this step is achieved by applying three filters, *interval* filter, *support* filter, and *periodicity* filter to events generated by **each node**.

Interval Filter

Typically, the operational events caused by network faults occur in "bursts". The interval filter segments the event sequence generated by each node into a series of "event bursts".

Definition 5 (Event Burst). *An* **event burst** *is an ordered sequence of events wherein two successive events are separated by no more than δ time units.*

As will be discussed in Section 5, an appropriate value for the parameter δ (typically a few milliseconds for communication networks) can be obtained by analyzing the distribution of inter-arrival time of events in a historical dataset. Since events in a burst occur within a very small time window, we arbitrarily define the timestamp of an event burst as the timestamp of its first event.

Support Filter

Given a set of event bursts $\mathcal{B}$ generated by a node, support filter treats each burst $s \in \mathcal{B}$ as a set and ignores the temporal ordering of events. Without ambiguity, we use s to denote both the event burst and its corresponding event set. We have the following definition.

Definition 6 (Support). *Given a set of event bursts $\mathcal{B}$, the*

[3]Without ambiguity, in the following we use "entity" and "node" interchangeably.

support *of an event set s,* $\mathsf{sup}(s)$*, is the number of bursts in* $\mathcal{B}$ *with event sets identical to s.*

To make implementation resilient against noise, we consider two event sets s_i and s_j identical if their *Jaccard similarity coefficient*, $\frac{|s_i \cap s_j|}{|s_i \cup s_j|}$, is close to 1.

Support filter separates fault-triggered and regular event bursts at a node by exploiting differences in their support. Specifically, it selects the subset of bursts in $\mathcal{B}$ with support within a range $[\underline{\lambda}, \overline{\lambda}]$, since event sets with extremely low support are usually the noise component while over-frequent event sets typically correspond to regular network operations occurring at the node.

Periodicity Filter

The goal of the periodicity filter is to further reject event sets that occur periodically[4] since periodic event sets typically correspond to regular network operations, e.g., heartbeat messages.

Definition 7 (PERIODICITY)**.** *Let* $t_0^s, \ldots, t_k^s$ *be the timestamps when the event set s occurs, and* $d_i^s = t_i^s - t_{i-1}^s$ *denote the interval between the* $(i-1)$*-th and i-th occurrences. The periodicity of s,* $\mathsf{prd}(s)$*, is defined as the relative standard deviation of intervals:*

$$\mathsf{prd}(s) = \frac{1}{\overline{d^s}} \cdot \sqrt{\frac{\sum_{i=1}^{k}(d_i^s - \overline{d^s})^2}{k-1}}$$

where $\overline{d^s} = \sum_{i=1}^{k} d_i^s / k$ *denotes the average interval. We reject an event set s if* $\mathsf{prd}(s) < \gamma$.

3.2 Modeling of Event Bursts

Taking the event bursts $\mathcal{B}$ at a node (after the filtering of the first phase as input), this phase utilizes Markov chains to model $\mathcal{B}$ and produces a set of chains $\mathcal{MC}$ as the summarization of event bursts at a node. Markovian properties have been verified to be common in network operational events, e.g., [16, 19]. However, note that while we use Markov chains to model event bursts, we do not claim its optimality. It is worth emphasizing that our framework is flexible enough to support many other models (see Section 7) that can be used to summarize events at a node.

We adopt a mixture model that contains multiple Markov chains $\mathcal{MC} = \{c_k\}_{k=1}^{K}$. We assume that each event burst is independently generated by one specific chain. Each chain $c \in \mathcal{MC}$ describes one type of sequential behaviors; thus, the mixture model is able to capture diverse behaviors embedded in event bursts.

Now we proceed to describing the structure of the mixture model. Without ambiguity, let $\mathcal{B}$ represent the collection of event bursts after initial filtering, and let $\Sigma = \{e_i\}_{i=1}^{M}$ represent the event types that appear in $\mathcal{B}$. All chains in $\mathcal{MC}$ share the same structure: in a chain c_k, for each event $e_i \in \Sigma$, there is a corresponding state $o_{k,i}$; each state $o_{k,i}$ is associated with an initial probability $\theta_{k,i}^{I}$, indicating the probability that a burst starts with e_i; each state $o_{k,i}$ can transit to all other states $o_{k,j}$ (including to the state $o_{k,i}$ itself) with certain transition probability $\theta_{k,i,j}^{T}$; there is a special ending state $o_{k,0}$ for which all transition probabilities are zero. In the mixture model, each chain c_k is associated with a prior probability π_k, which satisfies $\sum_{k=1}^{K} \pi_k = 1$.

[4]A failed entity may also periodically generate failure events (e.g., syslog messages, ping fails, etc.); however, the NMS eliminates such duplicate failure events from the event stream using a standard process known as *de-duplication* before passing them to the diagnosis engine.

Let $\pi = \{\pi_k\}_{k=1}^{K}$, $\theta_k^I = \{\theta_{k,i}^I\}_{i=1}^{M}$, and $\theta_k^T = \{\theta_{k,i,j}^T\}_{i=1,j=0}^{M}$; the parameter space of this mixture model is represented as $\theta = (\pi, \{\theta_k^I, \theta_k^T\}_{k=1}^{K})$. Given an event burst $s = (e_1^s, e_2^s, \ldots, e_L^s)$, the likelihood that chain c_k generates s is given by:

$$\mathsf{like}(s|c_k, \theta) \propto \theta_{k,e_1^s}^{I} \cdot \left(\prod_{i=1}^{L-1} \theta_{k,e_i^s,e_{i+1}^s}^{T} \right) \cdot \theta_{k,e_L^s,0}^{T}$$

Meanwhile, the posterior probability that c_k generated s can be calculated as:

$$\mathsf{prob}(c_k|s, \theta) = \frac{\pi_k \cdot \mathsf{like}(s|c_k, \theta)}{\sum_{k'=1}^{K} \pi_{k'} \cdot \mathsf{like}(s|c_{k'}, \theta)}$$

The optimal setting of the parameters θ and the number of chains K remain to be determined. In the following, we first discuss how to determine θ that maximizes the posterior probability of the given set of event bursts. Let

$$\theta^* = \arg\max_{\theta} \mathsf{like}(\mathcal{B}|\theta) \cdot \mathsf{prob}(\theta)$$

where $\mathsf{prob}(\theta)$ is a prior distribution over θ and $\mathsf{like}(\mathcal{B}|\theta)$ represents the likelihood of observing the whole set of event bursts $\mathcal{B}$ under this model: $\prod_{s \in \mathcal{B}} \left(\sum_{k=1}^{K} \pi_k \cdot \mathsf{like}(s|c_k, \theta) \right)$. Unfortunately, no closed-form solutions exist for such maxima. Here, as sketched in Algorithm 1, an *expectation maximization* (EM) [7] algorithm can be used to iteratively search for the maxima (in the pseudo code below, Q denotes the objective function over the posterior distribution using current parameter estimation θ_{old}).

```
Input: event bursts $\mathcal{B}$, number of chains $K$
Output: parameter setting $\theta^*$
initialize $\theta_{old}$;
while not converged yet do
    compute $Q(\theta, \theta_{old})$ =
    $\sum_{s \in \mathcal{B}} \sum_{k=1}^{K} \mathsf{prob}(c_k|s, \theta_{old}) \log[\pi_k \cdot \mathsf{like}(s|c_k, \theta)] + \log(\theta)$ ;
    compute $\theta_{new} = \arg\max_\theta Q(\theta, \theta_{old})$;
    set $\theta_{old} = \theta_{new}$;
return $\theta^* = \theta_{new}$;
```

Algorithm 1: MODELING EVENT BURSTS

Now, we discuss how to set K. Essentially, by controlling the number of chains, K determines the complexity of the mixture model. Here, we apply the *Akaike's information criterion* [3, 16] to select K. Specifically, the information criterion of the mixture model is given by: $\mathsf{aic}(\theta) = 2|\theta|$ - $2\log[\mathsf{like}(\mathcal{B}|\theta)]$, where $|\theta|$ is the number of parameters to be estimated. The setting of K leading to a minimum $\mathsf{aic}(\theta)$ is considered as optimal.

3.3 Incorporation of Topological Dependency

A novel feature that significantly distinguishes META from existing solutions lies in its incorporation of network topology information in learning and matching faults. In this paper, we consider the following set of relationships, {*self-ing, neighboring, containing/contained, down/up-streaming, tunneling*}, with brief descriptions listed in Table 1. Note that the relationships *down/up-streaming* are referred from the perspective of the sink, i.e., u is at v's down-stream side if the route from the sink to u contains v. In the following, we use $\mathcal{R} = \{SE, NE, CN/CD, DS/US, TN\}$ to denote this set of topological relationships. Each relationship $r \in \mathcal{R}$ is associated with an inverse counterpart $\bar{r}$, e.g., 'down-streaming' to 'up-streaming', 'contained' to 'containing', etc. Given a node v, we refer to the set of network nodes with a specific relationship r to v as a *topo-set*, denoted by $\mathcal{N}_r(v)$.

relationship	description
selfing	u and itself
neighboring	u and v are directly connected
containing/contained	u/v contains v/u as a sub-component (e.g., a router and its interfaces)
down/up-streaming	u is at v's down/up-stream side (route from sink to u/v contains v/u)
tunneling	u is on a tunnel (a special type of network connection) with v as one end

Table 1: Topological relationships and descriptions.

Intuitively, we construct our fault signature based on the following two fundamental observations. (1) Typically, when a fault occurs at a root-cause node u, symptom events may be triggered in affected nodes that are topologically related to u. (2) The triggered event burst at an affected node v differs depending on the topological relationship between u and v. For example, in an Internet Protocol (IP) network if v is a direct neighbor of u (*neighboring*), the failure of u may lead to the event burst of ("*OSPF Interface Down*", "*OSPF Neighbor Down*") at v; while if u is on a tunnel with v as one end (*tunneling*), the failure of u may cause the event burst of ("*Failed Connection Attempt*", "*Open Tunnel Failure*") at v. Therefore,

Definition 8 (Fault Signature). *For a specific type of fault f, we define its signature $\mathsf{sig}(f)$ as a series of tuples $\langle c, r, \mathsf{prob}(c|f, r)\rangle$, where $c \in \mathcal{MC}$, $r \in \mathcal{R}$, and $\mathsf{prob}(c|f, r)$ is the probability of observing an event burst with temporal pattern c at an affected node that has the topological relationship r with the root-cause node where the fault f occurs.*

To learn fault signatures from historical data, we make the following assumptions: each event burst $s \in \mathcal{B}$ (observed at a node v) has been classified into a Markov chain c_v, and represented as a pair (v, c_v); the number of fault types $|\mathcal{F}|$ is known and all faults are reflected in the historical data; the time-window size ω is set as the maximum delay between observing the first and the last event bursts triggered by a single fault.

Algorithm 2 sketches our solution. (i) The event bursts $\mathcal{B}$ are first divided into subsets, each within a time-window less than ω, i.e., the event bursts in the same subset are possibly triggered by a single fault. (ii) In every subset, for each involved node v, one identifies the topological relationship r^*_v that leads to the minimum non-empty intersection of all the topo-sets, i.e., the set of candidate causes. Note that the principle of *minimum explanation* is applied here. (iii) All the tuples $\langle v, r^*_v, c_v\rangle$ in a subset $\mathcal{B}_i$ are then used to compute a potential signature S_i (a $|\mathcal{R}| \times |\mathcal{MC}|$ matrix). (iv) A K-means (with $K = |\mathcal{F}|$) clustering algorithm is applied to the set $\{S_i\}$; the centers of the clusters are regarded as the signatures for the $|\mathcal{F}|$ faults.

4. ONLINE MATCHING

The online matching component of Meta attempts to detect and localize faults as follows: (1) the incoming events are aggregated into event bursts, and for each burst s occurring at an affect node v, the probability $\mathsf{prob}(c|s)$ is calculated for all $c \in \mathcal{MC}$; (2) topologically-aware fault signatures are used to compute the probability $\mathsf{prob}(f|\bar{r}, v \leftarrow s)$ that the faulty node incurred the fault f and has a topological relationship r to the affected node v. If this probability is

```
Input: event bursts $\mathcal{B}$, window-size $\omega$, number of fault types $|\mathcal{F}|$
Output: $|\mathcal{F}|$ fault signatures
divide $\mathcal{B}$ into a set of subsets $\{\mathcal{B}_i\}$ according to $\omega$;
for each subset $\mathcal{B}_i$ do
    // apply the principle of minimum explanation
    compute $\{r^*_v\}_{(v,c_v)\in\mathcal{B}_i} = \arg\min_{r_v} |\bigcap_{(v,c_v)\in\mathcal{B}_i} \mathcal{N}_{r_v}(v)|$;
    for each $r \in \mathcal{R}$ and $c \in \mathcal{MC}$ do
        compute $S_{i,r,c} = \frac{\sum_{(v,c_v)\in\mathcal{B}_i} \mathbf{1}(r^*_v=\bar{r}, c_v=c)}{\sum_{(v,c_v)\in\mathcal{B}_i} \mathbf{1}(r^*_v=\bar{r})}$;
apply $K$-means clustering to $\{S_i\}$ with $K = |\mathcal{F}|$;
set fault signatures as the cluster centers;
```

Algorithm 2: Learning fault signature

greater than a certain threshold, then $\langle f, v, \bar{r}\rangle$ is termed an *evidence* that points to the set of all nodes with relationship r to v as the set containing the faulty node; (3) all collected evidences within a short time window are used to narrow down the set of nodes that include the faulty node.

We need four main data structures to accomplish the online matching: a buffer for aggregating incoming events into event bursts and to compute probability of a temporal model generating an event burst; an index of fault signatures to compute evidences; and an index of network topological dependency and a signature matching tree to enable efficient fault localization. Due to the space constraint, we will only describe the latter three data structures.

4.1 Indexing Fault Signature

To support efficient model-to-fault lookup, we devise an inverted fault signature structure $\mathcal{I}_s$ which maintains the association between models and possible faults. Recall that the signature of a fault f is a series of tuples of the form $\{\langle c, r, \mathsf{prob}(c|f, r)\rangle\}_{c\in\mathcal{MC}}$, where c and r represent a chain and a topological relationship, respectively, and $\mathsf{prob}(c|f, r)$ is the probability of observing c at a node with topological relationship r to the faulty node. Corresponding to each signature, we create an entry in $\mathcal{I}_s$: $\{\langle f, \bar{r}, \mathsf{prob}(f|\bar{r}, c)\rangle\}_{c\in\mathcal{MC}}$, where $\bar{r}$ is the inverse relationship of r, and $\mathsf{prob}(f|\bar{r}, c)$ is the posterior probability that f occurs at a node with topological relationship $\bar{r}$ to a given node observing c. Its computation is given by:

$$\mathsf{prob}(f|\bar{r}, c) = \frac{\mathsf{prob}(c|f, r) \cdot \mathsf{prob}(f)}{\sum_{f'\in\mathcal{F}} \mathsf{prob}(c|f', r) \cdot \mathsf{prob}(f')}$$

where the prior probability of the occurrence of f, $\mathsf{prob}(f)$, can be derived from the overall statistics of network faults.

Now, the posterior probability that a fault f occurs at certain node with relationship r to a node v which observes an event burst s can be calculated as follows:

$$\mathsf{prob}(f|\bar{r}, v \leftarrow s) = \sum_{c\in\mathcal{MC}} \mathsf{prob}(f|\bar{r}, c) \cdot \mathsf{prob}(c|s)$$

For each f and v (with event burst s), we select the set of topological relationships $\mathcal{R}_v$ that satisfies $\mathsf{prob}(f|\bar{r}, v \leftarrow s) \geq \kappa$ ($\bar{r} \in \mathcal{R}_v$). We term such a triple $\langle f, v, \mathcal{R}_v\rangle$ as an *evidence*.

4.2 Indexing Network Topology

While the incorporation of network topological information significantly boosts the precision of fault diagnosis, such improvement incurs extra computation cost in terms of 1) storing the topological information, and 2) correlating event bursts according to their underlying topological relationships. Here, we introduce novel space-efficient indexing structures for topological correlation.

As will be shown Section 4.3, a key operation heavily involved in the fault localization is computing the intersection of two topo-sets, e.g., joining the down-streaming neighbors of one node and the direct neighbors of another; therefore, for each indexing structure, we are particularly interested in analyzing its storage demand and the cost of retrieving (constructing) a topo-set from it. Here, we assume network configurations to be static, and consider incremental maintenance of indices for evolving networks as one direction for our further research. Due to space limitations, we focus our discussion on building indices for up/down-streaming and tunneling relationships.

Up-Streaming/Down-Streaming

A naïve solution that stores the up/down-streaming neighbors for each network entity, results in $O(1)$ retrieval cost and totally $O(|\mathcal{V}|^2)$ storage cost. We construct our indexing structure based on the following two observations: (1) the shortest path routes from the sink to all the nodes form a spanning tree rooted at the sink, i.e., a tree cover of $\mathcal{G}$ [1]; (2) the diameter ϕ of a typical management domain (as observed in four large enterprise networks) is about 3-7 hops. Therefore, in this setting, the set of up-streaming

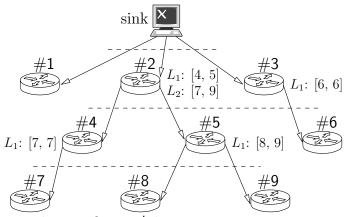

Figure 2: Index of up/down-streaming neighbors.

neighbors (utmost ϕ) of a node can be directly cached. We then traverse the routing tree in a level-order (breadth-first) assigning each node a traversal-order number. The down-streaming neighbors of a given node u can be summarized as ϕ intervals, $\{[l_i, r_i]\}_{i=1}^{\phi}$, where l_i (r_i) denotes the order number of its left-most (right-most) descendent on the ith level below u (see example in Figure 2).

Clearly, this indexing structure requires $O(|\phi \cdot \mathcal{V}|)$ space; maintaining a mapping (sorted on traversal-order number) that projects traversal-order numbers to node-identifiers, this scheme achieves retrieval cost of $O(\phi)$, since the neighbors on the same level can be retrieved in one consecutive chunk.

Tunneling

For tunneling relationship, we are interested in retrieving the set of nodes on tunnels with a given node u as one end, or, reformulated as: *given two nodes u and v, what set of nodes are on the tunnel (if any) connecting u and v?*

Without loss of generality, we assume that all the tunnels follow approximately shortest paths (e.g., OSPF and IGP routing [10]); hence, the problem is cast as indexing a set of shortest paths. Our solution is constructed atop the notion of hop cover of a collection of paths [5].

Definition 9 (Hop/Hop Cover). *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph and $\mathcal{P}$ be the set of shortest paths we intend to index. A* **hop** *is a tuple (p, u), where p is a shortest path with u as one end. A collection of hops $\mathcal{H}$ is said to be a* **hop cover** *of $\mathcal{P}$ if for any $P \in \mathcal{P}$, there is a subset of $\mathcal{H}$ such that P is a concatenation of these hops.*

We construct the collection of hops as follows: starting from an arbitrary path $P_1 \in \mathcal{P}$, we incrementally add in paths $P_2, P_3, \ldots$ from $\mathcal{P}$; the intersected segments between P_i and previous ones $P_1, \ldots, P_{i-1}$ break paths into disjoint hops, or divide existing hops into smaller ones; we collect the set of hops after inserting all paths of $\mathcal{P}$ as $\mathcal{H}$. A moment of reflection shows that (1) two paths can have at most one intersected segment and (2) the set of hops are invariant of the insertion order of paths.

Within this setting, the space-time tradeoff is achieved by caching a subset of $\mathcal{H}$ and leaving uncached hops to online computation; therefore, we are interested in selecting the optimal subset $\mathcal{H}^* \subseteq \mathcal{H}$ that leads to the minimum overall cost as follows. For a hop h, let $\mathsf{len}(h)$ be its length and $\mathsf{sup}(h)$ be the number of paths in $\mathcal{P}$ that contain h as a component. For simplicity, we model storage cost $\mathsf{cost}_{space}(h)$ $= \alpha \cdot \mathsf{len}(h)$ and computation cost $\mathsf{cost}_{time}(h) = \beta \cdot \mathsf{len}(h)$. Assuming that all the paths are queries with equivalent frequency, the overall cost of caching a subset $\mathcal{H}'$ of $\mathcal{H}$ can be modeled as: $\mathsf{cost}(\mathcal{H}') = \sum_{h \in \mathcal{H}'} \mathsf{cost}_{space}(h) + \sum_{h \in \mathcal{H} \setminus \mathcal{H}'} \mathsf{sup}(h) \cdot \mathsf{cost}_{time}(h)$; and the optimal subset $\mathcal{H}^*$ leads to the minimum overall cost: $\mathcal{H}^* = \arg\min_{\mathcal{H}'} \mathsf{cost}(\mathcal{H}')$. Clearly, in this model, a hop h should be cached if and only if its storage cost exceeds its computation cost with respect to all the paths, formally: $\mathsf{cost}_{space}(h) > \mathsf{sup}(h) \cdot \mathsf{cost}_{time}(h)$.

4.3 Correlating Relevant Evidences

With the help of the network topology index, in each evidence $\langle f, v, \mathcal{R}_v \rangle$, $(v, \mathcal{R}_v)$ can be replaced with the corresponding topo-sets $\bigcup_{r \in \mathcal{R}_v} \mathcal{N}_r(v)$ ($\mathcal{N}_{\mathcal{R}_v}(v)$ for short). Two evidences $\langle f, \mathcal{N}_{\mathcal{R}_u}(u) \rangle$ and $\langle f', \mathcal{N}_{\mathcal{R}_v}(v) \rangle$ are considered *relevant* if (1) $f = f'$, (2) $\mathcal{N}_{\mathcal{R}_u}(u) \cap \mathcal{N}_{\mathcal{R}_v}(v) \neq \emptyset$, and (3) they are within a time-window of size ω. This concept can be generalized to multiple evidences.

While it is straightforward to check conditions (1) and (3), computing the intersection of $\mathcal{N}_{\mathcal{R}_u}(u)$ and $\mathcal{N}_{\mathcal{R}_v}(v)$ is expensive: even if both sets are stored in a hash-table, the complexity is $O(\min\{|\mathcal{N}_{\mathcal{R}_u}(u)|, |\mathcal{N}_{\mathcal{R}_v}(v)|\})$. Moreover, following the naïve pair-wise comparison paradigm, each incoming evidence is compared with all existing ones to detect relevance, and thus scales poorly with the network event rate.

We devise a novel structure, signature matching tree $\mathcal{T}_s$, which enables efficient correlation of relevant evidences. Our design follows the one-pass clustering philosophy, [11, 25], which endows $\mathcal{T}_s$ with high throughput and scalability.

4.3.1 *Basic Structures and Operations*

$\mathcal{T}_s$ is a hierarchical structure, with the highest level containing $|\mathcal{F}|$ buckets, each corresponding to one fault $f \in \mathcal{F}$. Within each bucket is a height-balanced tree $\mathcal{T}_s^f$, into which evidences of the form $\langle f, \mathcal{N}_{\mathcal{R}_v}(v) \rangle$ are inserted. Each leaf of $\mathcal{T}_s^f$ corresponds to a cluster of relevant evidences; each non-leaf represents the union of all the clusters in its subtree.

For each leaf (cluster) C containing a set of evidences, we maintain the intersection of their topo-sets, called its aggregation, $\rho(C) = \bigcap_{\langle f, \mathcal{N}_{\mathcal{R}_v}(v) \rangle \in C} \mathcal{N}_{\mathcal{R}_v}(v)$. For each non-leaf (super cluster) SC, we maintain the union of the aggregations of the clusters in its subtree, $\rho(SC) = \bigcup_{C \in SC} \rho(C)$.

The signature matching tree supports two basic operations, insertion and deletion. In an insertion operation, a newly coming evidence $\langle f, \mathcal{N}_{\mathcal{R}_v}(v) \rangle$ recursively descends down $\mathcal{T}_s^f$ by testing $\mathcal{N}_{\mathcal{R}_v}(v) \cap \rho(SC)$ for each non-leaf SC encountered, until being clustered into an appropriate leaf that can absorb it; if no such leaf exists, a new one is created which solely contains this evidence; it then updates the

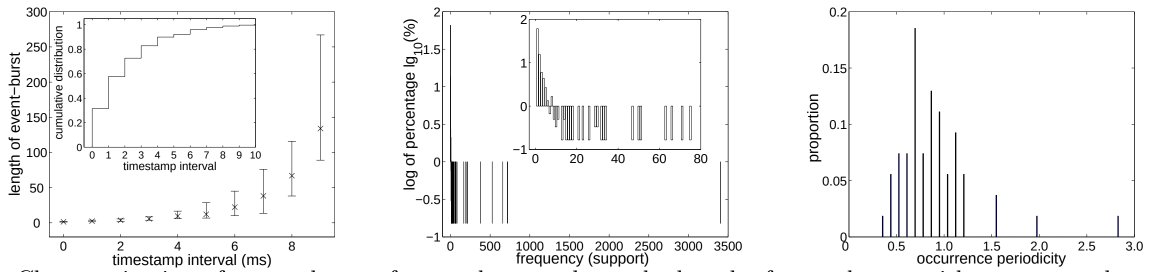

Figure 3: Characterization of a sample set of network event data: the length of event bursts with respect to the maximum timestamp interval, the distribution of frequencies of event sets, and the normalized histogram of the periodicity of event sets.

aggregations of the nodes on the path from the leaf to the root of $\mathcal{T}_s^f$. Those evidences with timestamps out of the current time-window are considered as expired. In a deletion operation, expired evidences are removed from the tree, and the aggregations of the nodes on the paths from the affected leaves to the root are updated in a bottom-up manner.

4.3.2 Optimizations

Two expensive operations involved in the signature matching tree are (1) testing the intersection of the topo-sets of an evidence and the aggregation of a (non-)leaf, and (2) updating the aggregations of the affected (non-)leaves when deleting expired evidences. Here, we introduce two-folded optimizations to ameliorate these two operations.

Filtering-and-Refining

Instead of performing direct comparison of two sets, we follow a filtering-then-refining paradigm: in the filtering phase, we perform fast check to determine if the intersection is non-empty, which may contain false positive results, but no false negative ones; in the refining phase, we make the real comparison. To this end, for each evidence $\langle f, \mathcal{N}_{\mathcal{R}_v}(v)\rangle$, we maintain its bloom filter encoding, $\mathsf{bf}[\mathcal{N}_{\mathcal{R}_v}(v)]$; for each leaf C, we maintain the bloom filter encoding of its aggregation, $\mathsf{bf}[\rho(C)]$; while for each non-leaf SC, a counting filter [8] encoding (to support efficient update) of its aggregation, $\mathsf{cf}[\rho(SC)]$, is maintained. Therefore, the intersection of $\mathcal{N}_{\mathcal{R}_v}(v)$ and $\rho(SC)$ (or $\rho(C)$) can be easily pre-tested using $\mathsf{bf}[\mathcal{N}_{\mathcal{R}_v}(v)] \cap \mathsf{cf}[\rho(SC)]$ (or $\mathsf{bf}[\mathcal{N}_{\mathcal{R}_v}(v)] \cap \mathsf{bf}[\rho(C)]$).

Slotted-Aggregations

To ameliorate the impact of frequent deletions of expired evidences over updating the aggregations of (non-)leaves, we introduce the slotted-aggregates mechanism [2]. Assuming that the sliding window size is ω, a slot cache maintains the aggregations in m slots, the i^{th} slot corresponding to the evidences with timestamp falling in the i^{th} sub-window of size ω/m time units. Now, the deletion of expired evidence affects at most one slot, and the aggregations in all remaining slots can be reused.

5. EMPIRICAL ANALYSIS

This section presents an empirical evaluation of META by using it in the context of communication networks. The experiments are specifically designed to center around the following metrics: (1) the efficacy of the signature model in capturing real network-faults, (2) the effectiveness of online matching in detecting and localizing network faults, and (3) its space and time complexity. We start with describing the datasets and the setup of the experiments.

Attribute	Description
IPaddress	address where the agent resides
PeerIPaddress	address of the master peer (if any)
Event-Count	sequence number of the event
specific-trap	enterprise specific snmp trap type
RawCaptureTimeStamp	time-stamp of the trap message

Table 2: Format of the network event data.

5.1 Experimental Setting

Our experiments used two datasets collected from real-life communication networks to evaluate the offline learning and online matching components of META. The first dataset is an archive of SNMP (Simple Network Management Protocol) trap messages collected from a large enterprise network (7 ASes, 32 IGP networks, 871 subnets, 1,268 VPN tunnels, 2,068 main nodes, 18,747 interfaces and 192,000 entities) over several days in 2007; this dataset is used to extract fault signatures. Event attributes of interest to us are listed in Table 2. The second dataset is a European backbone network consisting of 2,383 network nodes (spans 7 countries, 11 ASes and over 100,000 entities). We generate a synthetic event stream for this network (with tuneable failure rate) to quantify the efficacy and scalability of the online matching component.

A majority of the algorithms are implemented using Java. All the experiments are conducted on a Linux workstation running 1.6GHz Pentium IV and 1G memory.

5.2 Experimental Results

5.2.1 Offline Learning Component

In this set of experiments, we studied the effectiveness of our methodology in all three phases of the learning component of META: preparation of training data, modeling of event bursts, and incorporation of topological information.

Preparing Training Data

The first set of experiments studied the distribution of timestamp intervals, frequency, and periodicity of event sets to demonstrate the effectiveness of *interval* filter, *support* filter, and *periodicity* filter, respectively.

The left plot in Figure 3 illustrates the impact of interval size δ on the average length of event bursts. It is clear that the average length increases significantly as the interval grows, e.g., 50 events for δ = 8ms; meanwhile, a wider interval also enlarges the length deviation of the event bursts. We are interested in an optimal setting of δ that filters spurious event bursts. In our implementation, we used the following heuristic: in the cumulative distribution function (CDF) of intervals, find the interval value with the largest derivate, i.e., the one resulting in the most significant change of the number of event bursts. For example, in the CDF plotted

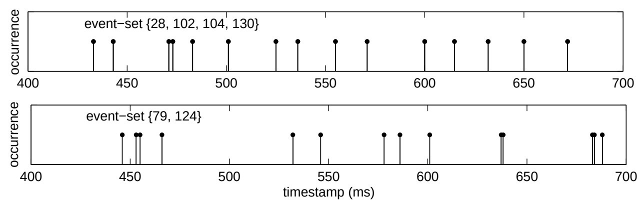

Figure 4: Occurrences of two sample event sets.

in Figure 3, we selected $\delta = 1$ms as the optimal setting.

The normalized histogram of event sets with respect to support (in log scale) is depicted in the center plot of Figure 3, which approximately follows a *power law* distribution. Observe that more than 60% event sets have fairly low support, e.g., below 5, which, as we confirmed by examining the definition of traps, are mainly caused by infrequent network operations, e.g., the event set {3} represents "the cisco NetReg server has started on the host from which this notification is sent", or certain non-recurrent faults, which are of modest interest for our purposes. Meanwhile, the event sets with significantly higher support than others are typically due to regular network operations, e.g., the event set {102, 104} which appears with support 348 indicates "data from the remote side is available for the TDM channel".

specific-trap	description
28	linkModeSniffingTrap
79	bsnAPRegulatoryDomainMismatch
104	metro1500TDMLocModuleData1
124	metro1500TDMLocModuleClockFail

Table 3: Specific-trap codes and descriptions.

The distribution of the periodicity of event sets is illustrated in the right plot of Figure 3. Observe that most of the event sets demonstrate low deviation of occurrence intervals, i.e., they are resulted from normal network operations. We randomly selected two event sets {28, 102, 104, 130} and {79, 124} with periodicity 0.43, and 1.09 (lower periodicity ⇒ more regular), respectively, and examined their implications. Figure 4 compares their occurrences. From the descriptions of the traps as shown in Table 3, it is confirmed that the event set {79, 124} indicates potential network faults, while the event set {28, 102, 104, 130} is caused by regular network operations, e.g., link mode sniffing.

Modeling of Event Bursts

We verified the Markovian assumption on event bursts using the *event burst length histogram* metric. More specifically, by running Monte Carlo simulation, we derived the histogram of event burst length from the learned Markov model, and compared it against that extracted from the real data.

The upper plot of Figure 5 illustrates the comparison of these two histograms (normalization is applied). It is clear that the distribution of the model-generated data fits that of the underlying data fairly tightly. Furthermore, we analyzed the distribution of individual events for real data and model generated data, respectively. As shown in the lower plot of Figure 5, these two distributions demonstrate strong consistency, which empirically proves that our learning model can capture the essential features of the real data.

Incorporation of Topological Dependency

Now, we proceed to verifying the imperative need of incorporating topologically correlated nodes in detecting and localizing network faults. Figure 6 illustrates the fractions of fault-triggered events reported at network nodes in different categories: the fault node itself (SE), and nodes with specific relationships (neighboring - NE, down-streaming - DS, up-streaming - US, and tunneling - TN) to the fault one using event data collected from real enterprise networks. Note that Enterprise 3 is a small-scale network where no VPN tunnels are deployed. Observe that the fault node itself reports only 18-29% of overall events, while those topologically correlated nodes take up to an overwhelming 71-82%.

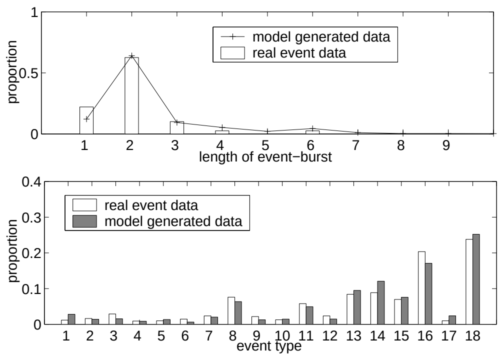

Figure 5: Histograms of lengths of event bursts and individual events in real event data and model generated data.

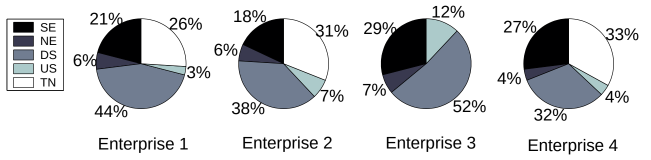

Figure 6: Fractions of fault-triggered events reported at fault nodes (SE) and nodes with specific relationships to fault ones (NE, DS, US, TN) in four real enterprise networks. The network sizes are listed as follows. Enterprise 1: 2514 nodes, Enterprise 2: 3200 nodes, Enterprise 3: 141 nodes, Enterprise 4: 12,444 nodes.

5.2.2 Online Matching Component

In evaluating the online matching component of META, we first generate synthetic event data for a European backbone network using the features of the real-life event data extracted in the previous phase, including 1) the temporal models for generating event bursts as symptoms of network faults, 2) the topological correlations for selecting the network entities where the symptoms will be observed, 3) the frequencies of individual events (type), and 4) the frequencies and periodicities of event sets. By controlling failure rates, we simulate network environments under both well-regulated and unstable conditions. The setting of major parameters is listed in Table 4.

Parameter	Value
number of fault types	50
number of event types	42
significant chains	16
fault occurrence rate	$0.02 \sim 0.1$
candidate size	$3 \sim 7$

Table 4: Setting of parameters for synthesizing network event data.

Accuracy of Fault Diagnosis

This set of experiments evaluated the effectiveness of the META framework in detecting potential network faults by analyzing streaming network event data. We aim at achieving fault detection and localization in a unified framework;

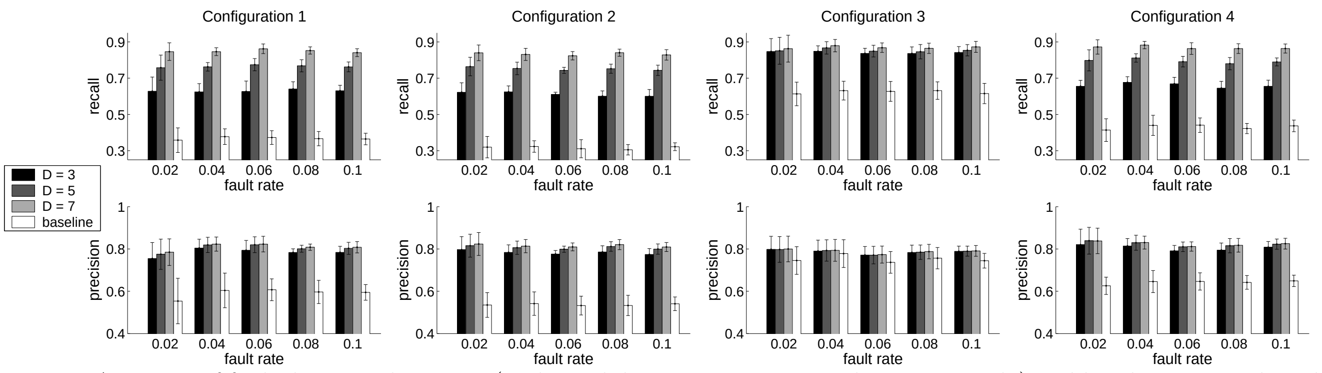

Figure 7: Accuracy of fault diagnosis by META (with candidate size $D = 3$, 5, and 7, respectively) and baseline approach with respect to fault occurrence rate and configuration of topological correlations.

therefore, we consider that a fault is successfully diagnosed only if the fault type is correctly determined and the fault node is localized with sufficient accuracy (in our implementation, we require that for every detected fault, the system suggest no more than D potential fault nodes (candidate size)). We refer to a successfully diagnosed fault as a *hit*. Viewing fault diagnosis as an information retrieval task, we measure the exactness and completeness of fault diagnosis using *precision* and *recall*. Formally,

$$\text{precision} = \frac{\#\ \text{hits}}{\#\ \text{reported faults}} \quad \text{recall} = \frac{\#\ \text{hits}}{\#\ \text{actual faults}}$$

Moreover, we measured the impact of topological information on fault diagnosis. We construct a baseline approach that is agnostic to topology information as follows: it has access to complete knowledge associating network faults with the observed symptoms, but has no possession of topological information. Given a symptom, the baseline approach attempts to identify the minimum set of faults that may trigger these symptoms; we regard it as a hit if the fault type suggested by the baseline approach is correct.

We measured the performance of META and the baseline approach under varying configuration of fault occurrence rate and topological correlations. The fault occurrence rate indicates the frequency of network faults (resulting in abnormal event bursts) relative to regular network operations (leading to normal event bursts); the configuration of topological correlation refers to the fractions of fault-triggered events observed at nodes with various topological relationships to the faulty node, e.g., SE, NE, DS, US, etc. Here, we adopt four different configurations as observed in real enterprise networks (shown in Figure 6).

Figure 7 compares the accuracy of fault diagnosis by META and the baseline approach. We make the following observations: (1) META achieves steady precision and recall scores under all the four configurations; the accuracy of the baseline approach is strongly correlated with the fraction of SE (fault node itself) events − even under configuration 3 (29% SE events), its recall (0.6) is substantially lower that of META (0.85). (2) The recall of META increases significantly as we increase D, for example, under configuration 2, the recall score of META grows from 0.65 to 0.84 as the candidate size D varies from 3 to 7. (3) The precision of META also increases as the candidate size D grows, which at the first glance may seem to contradict the inverse relationship between precision and recall typically observed in information retrieval systems; however, this can be explained by the fact that a larger D essentially provides more leeway in identifying the fault node.

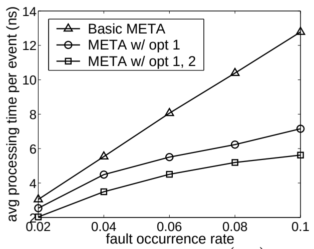

Figure 8: Average processing time (ns) per event with respect to fault occurrence rate, where opt_1 and opt_2 refer to the filtering-then-refining and the slotted aggregation strategies, respectively.

Efficiency of Execution

This set of experiments are designed to measure the scalability of the fault diagnosis in META. Specifically, we evaluate the average processing time of each incoming event, under varying condition of fault occurrence rate, with and without the multi-folded optimizations introduced in Section 4.3.2. Here, the fault occurrence rate refers to the fraction of event bursts caused by network faults.

Figure 8 shows the average processing time per event by three variants of META (the basic version, the one with the filtering-then-refining strategy, and the one with both optimization strategies) as the fault occurrence rate varies. We can obtain the following observations. (i) The processing cost of the basic META grows approximately linearly with the fault rate. (ii) The multi-folded optimizations significantly boost system performance, and the processing cost of both optimized variants of META manifest sub-linear growth rate with respect to the fault rate. (iii) The cost saving achieved by the optimization strategies demonstrates an increasing trend as the fault occurrence rate grows, which can be explained by the fact that a higher fault rate results in a greater number of evidences being fed to the diagnosis engine, thus resulting in superior performance gains.

6. RELATED WORK

For anomaly detection, a plethora of work has been done that uses analysis of low-level metric data, e.g., traffic or routing data, for anomaly detection. For example, in [9], BGP update messages are clustered along three dimensions, time, prefix, and views to detect network disruptions; in [14, 12], multivariate analysis is applied to model normal network traffic and detect deviations; in [24], a wavelet-based clustering algorithm is used to detect abnormal routing behavior. Nevertheless, targeting static analysis of low level metric data, these techniques are not suitable for real-time analysis of high-level event stream. Meanwhile, anomaly detection using historical data has also been an important topic for computing systems in general [6, 23], whose application to networked systems, however, is not clear.

Another line of research is specifically dedicated to fault localization from a set of observations or symptoms (detailed survey in [20]). The existing solutions can be categorized roughly as expert-system techniques and graph-theoretic techniques. The first category of approaches attempt to imitate the knowledge of domain experts, with examples including rule-based systems, e.g., [21], cased-based systems, e.g., [15], and model-based systems, e.g., [17]. The graph-theoretic techniques rely on a graphical model of the system, which describes the propagation for each specific fault, with examples including dependency graph [13], codebook technique [22], and belief-network [18]. These techniques suffer from two main drawbacks: first, they require accurate dependency information amongst network entities, which is usually not available for large scale enterprise networks; second, fault inference typically involves complicated computation and scales poorly with network size and complexity. In contrast, our approach only requires elementary topological information and fault signatures to support matching over high-volume event data.

7. CONCLUSION AND FUTURE WORK

To the best of our knowledge, this work presents the first “network-aware” signatures or patterns that incorporate topological relationships of nodes participating in a network phenomenon in conjunction with temporal evolution of events. Extensive experiments using data from communication networks demonstrate multiple benefits of “network-aware” signatures including a linear computational complexity algorithm for real-time detection.

This work also opens up several directions for further research: 1) incorporation of domain knowledge in training fault signatures; 2) exploration of alternative models of temporal evolution, e.g., hidden Markov chains, frequent itemsets; 3) search for data structures that can be incrementally adapted as network evolves; and 4) incorporation of a richer set of topological relationships derived from multi-layer networks, e.g., mining information diffusion or providing risk-aware access-control in socio-information networks.

Acknowledgements

The authors would like to thank Matthew Duggan (Principle Software Engineer, IBM Software Group) for overall guidance and domain expertise, Venkateshwara Madduri and Dipyaman Banerjee (IBM Research) for the network fault simulator, Dr. Charu Aggarwal (IBM Research) for in depth technical discussions, Dr. Kristian Stewart (Manager, IBM Software Group) for the event datasets, Dr. William Berriss (Tivoli Software Engineer, IBM Software Group) for the topology datasets, Daniel Martin (Tivoli Solutions Architect, IBM Software Group) for introducing the communication networks problem, and Dr. Dinesh Verma (Senior Manager, IBM Research) and Dr. Matthew Ellis (Vice President, IBM Sales and Distribution) for supporting this research effort. The first and last authors are partially sponsored by grants from NSF CyberTrust, IBM Faculty Award, and IBM SUR program.

8. REFERENCES

[1] R. Agrawal, A. Borgida, and H. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. *SIGMOD Rec.*, 18(2), 1989.

[2] Y. Ahmad and S. Nath. COLR-tree: communication-efficient spatio-temporal indexing for a sensor data web portal. In *ICDE*, 2008.

[3] H. Akaike. A new look at the statistical model identification. *IEEE Trans. Auto. Cont.*, 19(6), 1974.

[4] A.-L. Barabási. *Linked: The New Science of Networks*. Perseus Publishing, 2002.

[5] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. *SIAM J. Comput.*, 32(5), 2003.

[6] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox. Capturing, indexing, clustering, and retrieving system history. In *SOSP*, 2005.

[7] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the *em* algorithm. *J. Royal Stat. Soci. B*, 39(1), 1977.

[8] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary cache: A scalable wide-area web cache sharing protocol. In *IEEE/ACM Trans. Netw.*, 1998.

[9] A. Feldmann, O. Maennel, Z. Mao, A. Berger, and B. Maggs. Locating internet routing instabilities. *SIGCOMM Comput. Commun. Rev.*, 34(4), 2004.

[10] I. E. T. Force. OSPF version 2. *http://www.ietf.org/rfc*.

[11] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, 1984.

[12] Y. Huang, N. Feamster, A. Lakhina, and J. Xu. Diagnosing network disruptions with network-wide analysis. *SIGMETRICS Perform. Eval. Rev.*, 35(1), 2007.

[13] I. Katzela and M. Schwartz. Schemes for fault identification in communication networks. *IEEE/ACM Trans. Netw.*, 3(6), 1995.

[14] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. *SIGCOMM Comput. Commun. Rev.*, 35(4), 2005.

[15] L. Lewis. A case-based reasoning approach to the resolution of faults in communication networks. In *IM*, 1993.

[16] X. Meng, G. Jiang, H. Zhang, H. Chen, and K. Yoshihira. Automatic profiling of network event sequences: algorithm and application. In *IEEE INFOCOM*, 2008.

[17] Y. Nygate. Event correlation using rule and object based techniques. In *IM*, 1995.

[18] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., 1988.

[19] F. Salfner. Event-based failure prediction: an extended hidden markov model approach. *Department of Computer Science, Humboldt-Universität zu Berlin, Germany*, 2008.

[20] M. Steinder and A. Sethi. A survey of fault localization techniques in computer networks. *Sci. Comput. Prog.*, 53, 2004.

[21] P. Wu, R. Bhatnagar, L. Epshtein, M. Bhandaru, and S. Zhongwen. Alarm correlation engine. In *NOMS*, 1998.

[22] S. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie. High speed and robust event correlation. *Communications Magazine, IEEE*, 34(5), 1996.

[23] C. Yuan, N. Lao, J.-R. Wen, J. Li, Z. Zhang, Y.-M. Wang, and W.-Y. Ma. Automated known problem diagnosis with event traces. In *EuroSys*, 2006.

[24] J. Zhang, J. Rexford, and J. Feigenbaum. Learning-based anomaly detection in BGP updates. In *MineNet*, 2005.

[25] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *SIGMOD*, 1996.