What's Going On? Learning Communication Rules In Edge Networks

Srikanth Kandula
MIT CSAIL
kandula@mit.edu

Ranveer Chandra
Microsoft Research
ranveer@microsoft.com

Dina Katabi
MIT CSAIL
dk@mit.edu

ABSTRACT

Existing traffic analysis tools focus on traffic volume. They identify the heavy-hitters—flows that exchange high volumes of data, yet fail to identify the structure implicit in network traffic—do certain flows happen before, after or along with each other repeatedly over time? Since most traffic is generated by applications (web browsing, email, p2p), network traffic tends to be governed by a set of underlying rules. Malicious traffic such as network-wide scans for vulnerable hosts (mySQLbot) also presents distinct patterns.

We present eXpose, a technique to learn the underlying rules that govern communication over a network. From packet timing information, eXpose learns rules for network communication that may be spread across multiple hosts, protocols or applications. Our key contribution is a novel statistical rule mining technique to extract significant communication patterns in a packet trace without explicitly being told what to look for. Going beyond rules involving flow pairs, eXpose introduces templates to systematically abstract away parts of flows thereby capturing rules that are otherwise unidentifiable. Deployments within our lab and within a large enterprise show that eXpose discovers rules that help with network monitoring, diagnosis, and intrusion detection with few false positives.

Categories and Subject Descriptors

C.2.2 [**Computer Communication Networks**]: Network Protocols; G.3 [**Probability and Statistics**]: Correlation

General Terms

Algorithms, Design, Management, Measurement, Performance

1. INTRODUCTION

Perhaps the single defining aspect of edge networks today is that they are complex to manage. Today's enterprise and campus networks are built from multiple applications, protocols and servers which interact in unpredictable ways. Once the network is set-up, there are few tools that let an administrator keep track with what is going on in the network. Configuration errors seep in, software gets upgraded and servers get phased out leaving the administrator with the unenviable job of ensuring that traffic in the network conforms to a plan. Of course, scripting cron jobs and correlating server logs to figure out what's going on is a tedious option that does not scale [19].

SIGCOMM'08, August 17–22, 2008, Seattle, Washington, USA.

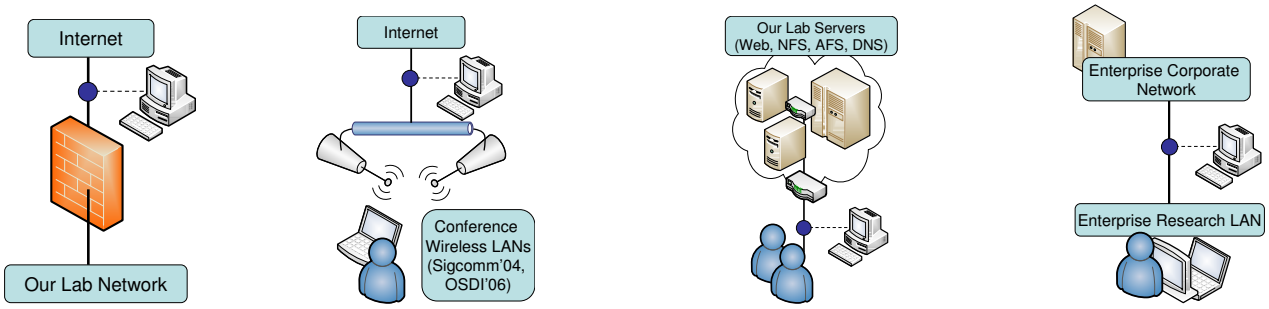
We advocate an alternative approach to manage the complexity in edge networks. Suppose we shift focus away from individual servers and configurations and focus directly on packets on the wire. Suppose that from a packet trace, we could learn communication rules that underlie a majority of the user activities present in that trace. Such a broad picture would provide an administrator with a reality-check; he can see how traffic for major applications traverses the network, identify configuration errors or unwarranted communication, and perhaps detect malicious activity. This paper is a first step towards this goal.

Existing trace analysis tools however cannot reveal to an administrator the communication patterns in his network. They focus on traffic volume and do not reveal the implicit structure in edge network traffic. Tools like MRTG [15] and NetFlow Analyzers focus on the heavy hitters, i.e., flows that exchange a lot of data. They may report that 30% of the traffic is to the web-server and that 70% of the traffic uses TCP. Advanced tools like AutoFocus [5] adapt their granularity of search. They can reveal IP subnets (e.g, 10.0.0.0/8) or port ranges that contribute lots of traffic.

This paper introduces eXpose, a new analysis technique that extracts significant communication rules in a network trace, without being told what to look for. A communication rule is a predicate such as $Flow_i.new \Rightarrow Flow_j.new$ indicating that whenever a new $Flow_i$ connection happens, a new $Flow_j$ connection is likely to happen. For example, eXpose would deduce rules like a DNS connection often precedes new connections, an AFS client talks to the root-server (port 7003) before picking files up from the appropriate volume-servers (port 7000), an End-Point-Mapper RPC call precedes mail fetch from Microsoft Exchange Servers, and viruses such as the mySQLbot probe flood the entire network looking for vulnerable mySQL servers.

Our key insight is simple–if a group of flows consistently occurs together, the group is likely dependent. Of course, correlation does not always imply dependence, but this is the best one can do lacking knowledge of the applications and the network layout. The challenge lies in applying this insight to a network trace where millions of flows show up every few hours. To do this, eXpose selectively biases the potential rules it evaluates and does not evaluate rule types that are unlikely to yield useful information. Second, eXpose abstracts away extraneous flow details to make useful patterns more discernible. Third, eXpose uses an appropriate statistical measure to score the candidate rules and mines efficiently. Finally, eXpose aggregates the discovered rules into a small number of useful clusters that an administrator can corroborate and use.

Not all dependencies are visible at the granularity of a flow. For example, suppose whenever a client talks to a sales server, the server fetches data from a backend database. Yet no individual client accesses the server often enough to create a significant rule. To capture such rules that are otherwise unidentifiable, eXpose introduces templates that systematically abstract away parts of flows. For example, one of our templates replaces the client's IP with a wild-card character creating a *generic* whenever any client talks to the sales



(a) Lab's Access Link **(b) Access Link of Conference LANs** **(c) Before Lab's Enterprise Servers** **(d) Within a Major Enterprise**

Figure 1: We evaluated eXpose on traces collected at each of the locations above (shown by the blue circle). The locations included *access links* at a large university and at two different conference wireless LANs and, *links carrying enterprise traffic* towards servers at a university and at a large company.

server. Leveraging these generics, eXpose searches for rules like $* : Sales \Rightarrow Sales : Database$, meaning whenever *any one* of the clients talks to the Sales server, the Sales server talks to its Database. We show how to apply templates without any prior knowledge about the hosts involved in the trace and find that these templates improve the expressiveness of our rules considerably.

eXpose has three unique characteristics. First, eXpose extracts communication rules with zero guidance, i.e., without being told what to look for. Hence, eXpose can identify communication rules that are unknown to the administrator, including configuration errors, anomalies, and exploits. Second, while prior work [1, 10] focuses on individual applications, a single source-destination pair [10], or the dependencies for clients accessing a given server [1], eXpose learns patterns that may involve multiple applications, servers or protocols. Extracting this broad picture by applying prior techniques that learn rules one application at a time or one server at a time does not scale and may miss out on important dependencies if the admin forgets to point the tool towards the correct servers (or applications). Third, eXpose is in a sense future-proof. By making minimal assumptions about applications, protocols and servers, eXpose's techniques to learn rules can deal with evolving networks.

We deployed eXpose on the server-facing links of two edge networks, in the CSAIL lab at MIT and in the Microsoft Research network for a couple of months. We also run eXpose on traces collected on the access-links (i.e., Internet facing links) of our lab at MIT and two conference hot-spots. We corroborated rules output by eXpose with the admins at the corresponding locations and aggressively report every rule that we could not explain as a false positive. Our results show the following:

- eXpose discovered rules for enterprise configurations and protocols that are deployed and used in mainstream operating systems that we did not know existed, such as Nagios [14] monitors and link-level multicast name resolution [12].
- eXpose discovered rules for the major applications such as email, web, file-servers, instant messengers, peer-to-peer and multimedia distribution in each of the two edge networks.
- eXpose detected configuration errors leading to bug fixes.
- eXpose detected malicious traffic, machines infected by trojans, mysql bot scans and ssh scans targeting key routers.
- eXpose mines for rules in time that is much smaller than the duration of traces (0.1% – 23.8%). This means that although our current implementation works offline by feeding off packet traces, an online implementation is feasible.

We begin by describing eXpose's learning algorithm.

2. LEARNING COMMUNICATION RULES

Our goal is the following: Given a packet trace, discover the communication rules inside the corresponding network. We define a communication rule as a dependency between flow activities (see Table 1). A communication rule $X \Rightarrow Y$ occurs in the trace if flow activity X implies flow activity Y. For example, X may be a new http connection from a client to a web-server, which implies Y, a new connection from the client to the DNS server. The rules themselves are probabilistic to mimic the underlying phenomena. For example, new connections trigger DNS lookups only upon a miss in the local cache. Some communication rules are at the granularity of flows such as whenever Host1 talks to Host2, Host2 talks to Host3; whereas others are more abstract, such as whenever *any client* talks to the web-server W, the web-server talks to its backend B. We call the latter *generic rules* and define simple templates for such generic rules. Our tool discovers all significant instantiations of these templates in the given packet trace. Some of the rules we discover describe the normal behavior of the network, while others identify attacks and mis-configurations in the network.

2.1 From Dependency to Association

How do we detect flow dependencies in a trace? Lacking knowledge of the applications and the network layout, it is impossible to say that two activities seen in a trace are dependent. The best one can do is to find correlated occurrences in the trace. At a high-level, our idea is to identify flow groups that co-occur with frequency significantly greater than chance and repeat consistently over time.

But there are simply too many flows in a trace, for e.g., in all of the links that we collected traces at (see Fig. 1), we saw more than 10^5 flows within a few hours. Examining every pair of flows for dependence doesn't scale, let alone examining larger groups of flows.

To design a scalable solution, we make a Markovian assumption. Whether a flow is absent, present, or new at any point of time is only influenced by flows that have recently occurred. Said differently, we partition the trace into time windows and look for dependencies that occur within the same time window (default window size is 1s). This assumption focuses on identifying dependencies that are separated only by a short period of time. We believe that most network dependencies are machine-generated and are indeed separated by a short period of time. For example, we learnt the communication rules for BitTorrent, viruses and, email clients (See §4) by looking within 1s-wide windows. A systematic evaluation of sensitivity to window size is in §4.5. We acknowledge, however, that some flow dependencies occur over longer periods. For example, the gap between a user reading a web-page and clicking on a link may exceed 1s. Extending our technique to look at dependencies over multiple time windows is possible but is outside the scope of this paper.

Using this Markovian assumption, eXpose converts the input trace into an *activity matrix*. The rows of the activity matrix denote time windows in the trace, whereas the columns correspond to flows in the trace. Each entry in the matrix takes one of three values: new, $present$ or $absent$ denoting the activity of a flow (column) in

Term	Meaning
Flow	A five-tuple <local IP, local Port, remote IP, remote Port, Protocol>
Flow Activity	A flow is either *absent, present* or *new*. Note *new* $\subset$ *present*.
Rule $X \Rightarrow Y$	X, Y are tuples of <flow, flow_activity>. $X \Rightarrow Y$ is a strong rule if whenever flow F_X has activity A_X, flow F_Y is more likely to have activity A_Y than normal.
Activity Matrix	Rows denote time windows, columns denote flows in the trace. Each entry holds the activity of a flow (column) in the corresponding time window (row)

Table 1: Definitions used in this paper

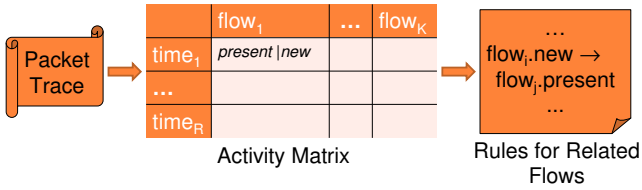


Figure 2: Schematic of eXpose's work-flow.

a time window (row). We observe that this representation (Fig. 2) is typical of a data-mining problem. Hence eXpose finds dependent flows by mining for association rules [4] in the activity matrix.

Flows and their activities: To be concrete, we identify a flow in terms of its source and destination IPs, ports, and protocol. A flow is *present* in a time-window if it sends non-zero packets during that period. For connection-oriented flows like TCP that have an explicit start and end (syn, fin), a flow is *new* in the time window that overlaps its beginning. For datagram-based flows and also when traces don't have detailed headers, a flow is *new* if it becomes present after a period of inactivity. Note that if a flow is new, it is also present in that time window (*new* $\subset$ *present*).

2.2 Significance of a Communication Rule

How does one tell which dependencies are significant? One might be tempted to say that a communication rule, $X \Rightarrow Y$, exists if X and Y occur together often, that is if $Prob(X \wedge Y)$ is high. This intuition is incorrect, however, and causes both false positives and false negatives. Two dependent flows may always occur together in the trace, yet happen so rarely (low $Prob(X \wedge Y)$) that their dependence is not identified. On the other hand, an unrelated pair of flows can have a high joint probability simply because one of the flows is very popular.

We take an information-theoretic approach to identify statistically significant dependencies—pairs of flows that have much higher joint probabilities than merely due to chance. Specifically, we set up candidate rules of the type $X \Rightarrow Y$, where both X and Y are tuples of <flow, flow-activity> and use JMeasure [21], a known metric in the data-mining community, to score candidate rules.

$$JMeasure(X \Rightarrow Y) = I(Y; X = 1), \tag{1}$$

where $I(Y; X = 1)$ is the mutual information, i.e.:

$$I(Y; X = 1) = P(X \wedge Y) \log \frac{P(Y|X)}{P(Y)} + P(X \wedge \neg Y) \log \frac{P(\neg Y|X)}{P(\neg Y)}.$$

Intuitively, the JMeasure score of rule $X \Rightarrow Y$ is the reduction in entropy of the random variable Y when the random variable X happens. If X and Y were independent, the JMeasure score would be zero. At the other extreme, if Y is completely dependent on X, the JMeasure takes the largest possible value, $P(X) \log \frac{1}{P(Y)}$.

Unlike other measures of statistical significance, such as the Chi-Square Test [9] and the F-Measure [9], the JMeasure score does encode the directionality of the relationship. This is crucial because we expect dependencies in practice to be uni-directional. For example, a HTTP connection may trigger a DNS connection but not all DNS connections happen due to HTTP.

Unfortunately, the JMeasure rule that works well for general data-mining comes short when identifying network dependencies for the following reasons.

(a) Negative Correlations: Reduction in entropy occurs in one of two ways. In the extreme, when X happens, Y may always happen or Y may never happen. Unfortunately, JMeasure does not differentiate between these two cases and scores both highly. While the first type of rules are interesting as they correspond to co-occurring flows which are likely to be true dependencies, the latter kind happens trivially often in network traces and are generally not meaningful. There are so many flows that are much shorter than the duration of the trace (low $P(Y)$) that it is easy to find another flow X that happens only when Y does not occur, spuriously leading to a high JMeasure rule. To avoid such rules, we only use the positive correlation part of the JMeasure rule:

$$Score(X \Rightarrow Y) = P(X \wedge Y) \log \frac{P(Y|X)}{P(Y)}. \tag{2}$$

(b) Long-Running Flows: Long-duration flows pose a challenge unique to mining network traces. Every one of our traces had flows that were contiguously active over a significant part of the trace–long downloads (FTP, SMB) and long-running shell sessions (telnet, ssh, remote-desktop). Given the prevalence of short duration flows, it is often the case that a short flow (X) happens only when a long running flow (Y) is active. The above scoring technique scores this pair highly, yielding a spurious rule. To avoid such rules we employ the following principles. First, we note that spurious rules like the above happen only when the activity on either side (both X and Y) is *present*. Long-running flows are *present* in many more time-windows than they are *new*, and a short flow that starts many times, i.e., has many *new* time-windows, is unlikely to completely coincide with a long flow. So, we prefer to report rules involving *present* activities only when there is little mis-match between the frequencies on either side: $1/3 \leq \frac{P(X)}{P(Y)} \leq 3$. Second, we do not report rules involving flows that happen in more than 90% of the time-windows of the trace. It is unlikely that anything reasonable can be said about such dominant flows.

(c) Too Many Possibilities: Any statistical test, including our technique, is based on a simple principle. How likely is the null hypothesis (in this context, the hypothesis that X and Y are independent) given the score? If a rule has a score value that makes the probability of the null hypothesis vanishingly small, the rule is reported as statistically significant. Unfortunately, in network mining there is a complementary challenge. There are so many flows, and so many potential rules, that even if each null hypothesis has very small probability, overall the likelihood of false positives is quite large. eXpose selectively biases the possible rules it considers to reduce false positives. First, we note that a rule involving two flows can have upto four unique IPs–the sources and destinations of the two flows. It is unlikely that flows which have none of their ends in common are dependent (example: does IP A talk to IP B whenever IP C talks to IP D?). Hence, we only report rules involving flows that have at-least one IP address in common. Not only does this shift focus away from

potential rules that are more likely to be false-positives (see §4.3 for evaluation), but it also greatly reduces the number of potential rules, improving scalability. Second, we note that most flows happen in such few time windows that there is too little data to predict relationships for such short flows. So, instead, we focus on the heavy tail– the K most active flows in the packet trace. K can be set by the network administrator, otherwise it defaults to 5000 flows. See §4.5 for an analysis of eXpose's sensitivity to K.

(d) Cut-off: eXpose only reports rules that have a significance score (Eq. 2) greater than a threshold α which defaults to $.01 nats$. [1] To put this threshold in perspective, note that if two flows are completely dependent on each other, i.e., $P(X) = P(Y) = P(X \wedge Y)$, then both rules $X \Rightarrow Y$ and $Y \Rightarrow X$ will be output for all non-trivial values of $P(X)$. More interestingly, for general flows such as the flows from a client to a DNS and an HTTP server, the rule $HTTP \Rightarrow DNS$ will be output only when two conditions hold: (1) co-occurrence better than chance, i.e., $P(DNS|HTTP) > P(DNS)$ and, (2) frequent co-occurrence, i.e., $P(HTTP \wedge DNS) > \frac{.01}{\log P(DNS|HTTP) - \log P(DNS)}$. Note the interplay between the two conditions, the more likely a DNS flow is when an HTTP flow happens, the larger the gap between $P(DNS|HTTP)$ and $P(DNS)$, the less stringent the constraint on frequency of co-occurrence.

2.3 Generic Communication Rules

So far, we have been looking at dependencies between pairs of flows. But, not all dependencies are visible at this granularity. For example, suppose whenever a client talks to a sales server, the server fetches data from a backend database. Clearly this is an important dependency, yet it may be missed if no *single* client accesses the server frequently (recall that we score based on flow frequency). Our idea is to relax the granularity at which we look for dependencies. In particular, we report the above dependence as long as all the clients *together* access the server often enough. To achieve this we introduce generics.

The idea behind generics is similar to wild-cards in regular expression matching—relax some of the fields in a flow's five-tuple. As an example, whenever the flow $Client.SomePort : Sales.80$ is active in a time window, we introduce a generic $*.* : Sales.80$ as being active in that time window. Further, we consider rules like

$$*.* : Sales.80 \Rightarrow Sales.* : Database.*, \quad (3)$$

and say that this rule happens whenever *some client* talks to Sales and Sales talks to the Database within the same time window. A more interesting example of a generic rule is:

$$*.* : WebServer.80 \Rightarrow *.* : DNS.53. \quad (4)$$

For rules that relax IPs on both sides, we say the rule happens in a time window only if the *missing* fields on both sides are the same. It does not matter which client(s) lead to the two relaxed flows, as long as there is *at least one client that talked to both* the web-server and the DNS in that time-window.

2.3.1 Templates for Generic Rules

When to instantiate generics and how to combine them into communication rules? Relaxing exhaustively explodes the number of flows and rules. Instead, we define simple templates for relaxing flows and for combining the generics into rules. Whenever a flow matches a relaxation template, we instantiate the corresponding generic. Rule templates prescribe which combinations of generics and flows are worth considering as a communication rule. We introduce one relaxation template and two rule templates. Our tool automatically learns all significant rules that are formed through instantiations of these templates in the packet trace.

[1] nats stands for units of entropy when natural logarithms are used. For e.g., $\log(2) = .693\ nats$.

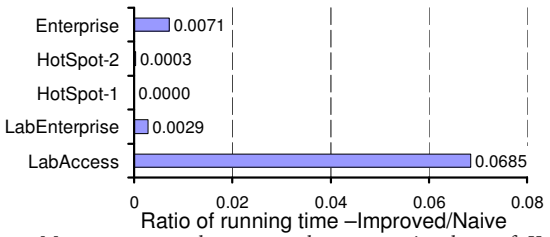


Figure 3: Measurement results compare the computational cost of eXpose's mining algorithm with that of a baseline.

Relaxation Template to Create Generics: Whenever one end of a flow is a well-known port (80 for http, 53 for DNS, /etc/services has exhaustive list), we relax the IP at the opposite (client) end. The idea is to create a generic that abstracts away the client's IP and focuses on all accesses to the server. Instead of relying only on the standard list of well-known server ports, we learn from each trace the ports used by a large number of flows in that trace and consider them well-known. This lets us learn the ports used by peer-to-peer applications. Finally, ports that are not in the well-known list are considered to be the same for the purpose of matching.

Rule-Templates to build Rules from Generics and Flows: Our first template helps to identify server's backend dependencies. Analogous to the example in Eq. 3, this template allows rules that combine a flow with a generic if the un-relaxed IP in the generic (i.e., the IP corresponding to the server) matches one of the IPs in the flow. The second template identifies dependencies that are visible to a client. Analogous to the example in Eq. 4, this template allows rules involving two generics. Such a rule is active in a time window only if at-least one client accesses both the server IPs/Ports in that time window.

2.3.2 Scoring Generic Rules

Rules involving generics, such as the example in Eq. 4, become more interesting as more unique clients conform to the rule. Hence, we supplement statistical significance by a *support* metric. The support of a rule involving one or more generics is the number of unique clients whose communication can be abstracted in the form of the rule. Clearly, a generic rule with support 1 is a trivial generalization. Hence, we only report generic rules that have support greater than a threshold β (default 3).

3. ALGORITHMS TO MINE FOR RULES

So far we introduced generics and the scoring function for statistical significance but how do we mine for communication rules? The costliest part of rule-mining, in our context, involves computing how often each candidate rule occurs in the trace. For a simple rule that involves only two flows, we need to count the time-windows when both flows happen; if the rule involves generics, we need to count time windows that contain flows matching the generics as described above.

Our key contribution here is a more efficient way to compute frequencies for all potential rules. Recall that we focus on the top K active flows (and generics) after discarding the really long flows (those that happen in more than 90% of the time windows). Suppose the trace consists of W consecutive time windows. Naively checking whether each of the $O(K^2)$ pairs of flows occur in each of the time-windows, takes $O(W * K^2)$ time. The square term, K^2, dominates the running time and can be quite long. Instead, we observe that in

any time-window only a handful of the K flows are active. Thus, instead of counting frequencies of all K^2 pairs at each time-window, we need only count flow pairs that are active in a time window. If the w^{th} window has S_w flows, our algorithm computes frequencies for all rules in $O(\sum_{w=1}^{W} {S_w}^2)$ time. Fig. 3 shows the improvement in computational efficiency with this algorithm on our traces. For concreteness, we show the pseudo-code of our algorithm.

Procedure 1 ExtractCommRules(Packet Trace)

```
Find F – the set of top K active flows and generics
Compute Activity Matrix M for all flows in F
Use Rule Templates to Create Candidate Rule set R
for all Time Windows w ∈ rows (M) do
    for all <flow, activity> tuples X, Y ∈ Window w do
        if X ⇒ Y ∈ Candidate Rules R then
            UpdateStats Freq(X ⇒ Y)
        end if
    end for
end for
for all X ⇒ Y rules in Candidate Rules R do
    if Score(X ⇒ Y) > α, Support(X ⇒ Y) > β then
        Output Rule X ⇒ Y as Significant
    end if
end for
```

3.1 Composing Communication Rules

The communication rules discussed above relate only a pair of flows. Our algorithm and statistical significance score naturally extend to rules involving more than two flows. To see this, note that if the left side of the rule $X \Rightarrow Y$ were to be replaced with a conjunction $X_1 \wedge X_2 \wedge \cdots \wedge X_N \Rightarrow Y$, both the algorithm and the significance score still apply.

The trade-off is computational complexity versus rule exhaustiveness. Looking for rules involving up to N flows would roughly take $O(W * K^N)$ time, since we would have to check each of the K^N flow groups generated from the top K flows on each of the W windows in the trace. On the other hand, doing so would help only when a set of flows are related but none of the pair-wise relationships are significant; for e.g., say F_3 is *present* only when both F_1 is *new* and F_2 is *present* and never otherwise. Here, $F_1.new \wedge F_2.present \Rightarrow F_3.present$ is a significant rule but neither of the pair-wise rules are significant enough. We believe in Occam's razor, the more complicated a rule, the less likely it would happen; hence we focus on pair-wise rules.

Complementary to rule exhaustiveness is the problem of grouping similar rules. Our analysis generates hundreds of statistically significant rules on a packet trace; hence it is worthwhile to attempt clustering similar rules into easier-to-understand groups. We found two techniques to be helpful. The first is rather simple. We transform the discovered rules into a rule-graph, wherein each rule is a directed edge joining the nodes corresponding to the <flow, activity> tuples that comprise the rule. We then take a transitive closure of the rule-graph to identify clusters of rules that involve related flows, for e.g., $X \Rightarrow Y_1$, $X \Rightarrow Y_2$, $Y_1 \Rightarrow Z$ will belong to the same transitive closure. The second technique is more complex and is motivated by our observation that the rule-graph consists of highly clustered components that are connected by "weak" edges, i.e., rules with low statistical significance. Hence, we apply spectral partitioning [23] to the rule-graph, to remove such weak edges between strongly connected components. Specifically, if A is the adjacency matrix of the rule-graph, where each matrix entry is the significance score of the corresponding rule, we recursively partition

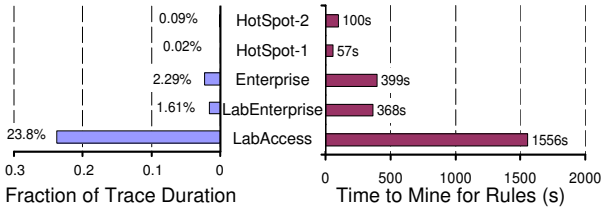


Figure 4: eXpose mines for communication rules within a small fraction of the duration of the trace, so an online solution is feasible.

the graph based on the second smallest eigen value of A's laplacian, as described below.

Procedure 2 RecursiveSpectralPartition(RuleGraph A)

```
if dim(A) < 10 or A is tightlyConnected then return
end if
Laplacian L = diag(colSums(A)) − A
Find EigenVector ν for 2^nd smallest eigen value of L
Split A's rows based on the sign of ν into A_pos, A_neg
Recursively SpectralPartition A_pos and A_neg
```

3.2 Towards a Streaming Solution

As presented so far, eXpose runs offline on packet traces. But, can eXpose work in an online fashion, i.e., can we learn rules from packet streams? Clearly, the chief obstacle is whether rules can be mined quickly enough. Fig. 4 shows how long eXpose takes to mine for rules in each of the five sample traces on a 1.8MHz Dual Core AMD Opteron with 16GB of RAM. Note that the time to mine rules is always much smaller than the duration of the trace and is often many orders of magnitude smaller.[2] Hence, an online implementation seems feasible. To flesh out a streaming version of eXpose, besides mining for rules, we need to compute rows of the activity matrix (§2.1) online, maintain counters for the currently active flows to determine their *new, present* status and frequency counters to identify the K highly active flows (§2.2); all of these seem solvable.

4. EVALUATION

We deployed eXpose on the links carrying traffic towards internal servers at our lab and a major Enterprise. Further, we analyzed traffic traces from three additional locations–the access links at our lab, and at two wireless hot-spots (conference venues for SIGCOMM'04 and OSDI'06). Here, we describe the rules that we corroborated through discussions with admins at each of the locations and highlight sources for both false positives and false negatives.

4.1 Dataset

Our traces encompass packet headers (pcap format) and connection records (BRO [16] format) while some traces anonymize IPs. Since eXpose does not perform deep packet inspection, summary information showing the flows active in each time window suffices. Table 2 summarizes our traces. All our traces were collected at links carrying traffic for thousands of clients, but the connectivity patterns and traffic mix vary. Traffic on the access links is mostly web browsing and email access and exhibits a high degree of fan-out with connections going off to many unique IPs. Traffic on the enterprise links has greater diversity in applications but is directed towards fewer IPs. Traffic in the Enterprise trace is dominated by Windows/NT machines whereas the others show a wider mix of operating systems. eXpose uncovered many interesting flow rules in the

[2] eXpose takes longer to mine rules in the LabAccess trace because unlike the other traces, there is little locality in this trace with flows going off to many unique destinations in each time window.

Trace	Collected at...	Type	Length	Unique Flows	Size
LabAccess	Our Lab (CSAIL@MIT)'s access link to the Internet	Conn. Records	3 hrs	2,832,931	2 GB
LabEnterprise	Link facing internal Lab Servers	Conn. Records	7 hrs	909,563	2 GB
Enterprise	Link between Microsoft's Research and corporate LANs	Pkt Headers	3 hrs	622,959	30 GB
HotSpot-1	Access link for the SIGCOMM'04 wireless LAN	Pkt. Headers	3 days	204,318	604 MB
HotSpot-2	Access Link for the OSDI'06 wireless LAN	Pkt. Headers	3 days	603,127	2.1 GB

Table 2: Dataset of traces we have tested eXpose with.

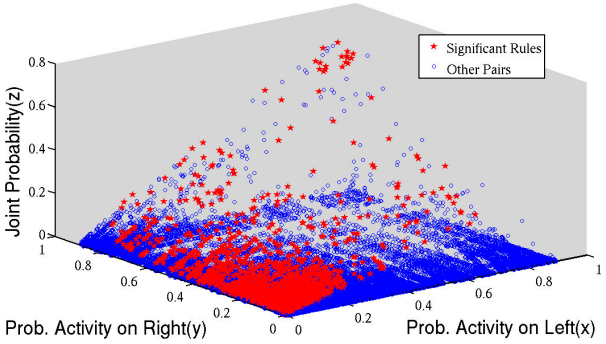


Figure 5: Rules identified by eXpose from among the many possible rules. The figure has a circle (in blue) for each possible rule and a star (in red) for rules that are identified as significant. Significant rules come in different forms; simply looking for high joint probability (high z) is not enough.

traces. The differences in the rules from the various locations provide insights into the characteristics of each environment.

4.2 Metrics

One of the metrics we care about is breadth; we want to extract broad rules that represent patterns for a majority of the traffic. In each of our traces, we were able to discover dependencies for a significant proportion of user actions— web browsing, email, file-server access, instant messaging, peer-to-peer traffic and multimedia content. We care about both correctness and completeness of our rules. False negatives are patterns that are expected to exist in a trace but are not discovered by eXpose. We checked with administrators at both our lab and the corporate enterprise and report the patterns missed by eXpose. False positives are flow-pairs that are scored highly by eXpose but have no reasonable explanation. We aggressively assume every rule we could not explain to be a false positive and mention the sources for such rules in our traces.

4.3 Nature of the Rule-Mining Problem

Identifying significant rules from among the many possibilities is tricky. For the LabEnterprise trace, Fig. 5 plots in (blue) circles each of the potential rules and in (red) stars each of the rules eXpose identifies as significant. Clearly, significant rules come in different forms, some involve activities that happen rarely, both individually and together (near the (0,0,0) corner), others involve one rare activity and one frequent activity (near the (0,1,0) and (1,0,0) corners), and yet others involve a pair of frequent activities (close to the (1,1,1) corner). Simply looking for pairs with high joint probability (points with $z > const$) or looking for pairs with high conditional probability ($\frac{z}{x} > const$) does not suffice.

Before detailing the kinds of rules discovered by eXpose, we present the bigger picture. eXpose augments a packet trace with generics–abstract versions of the real flows, evaluates many potential flow/generic pairs to extract the significant rules and clusters together rules that are similar to one other. Table 3 shows for each of our traces the progression of eXpose through each of these phases.

In this context, it is easy to place our chief contribution—a technique to identify the few hundred significant patterns from among the $10^{10} - 10^{12}$ possibilities. To achieve this, eXpose selectively biases search by not evaluating rules that are unlikely to be useful.

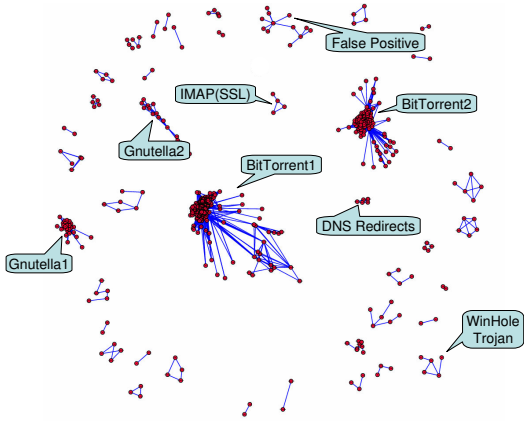


Figure 6: Annotated Snapshot of **eXpose's output** showing the rules learnt by eXpose for the HotSpot-1 Trace. A user can click on a pattern to see more information about the corresponding rule. Nodes in the graph represent flow activities and edges representing rules join the two activities involved in each rule.

Second, eXpose abstracts away extraneous flow details to make useful patterns more discernible. Third, eXpose scores the candidate rules with an appropriate statistical measure and mines efficiently. Finally, eXpose aggregates the inferred rules into a small number of useful patterns that an admin can corroborate and use. Fig. 6 shows an annotated snapshot of eXpose's output.

4.4 Evaluation in Controlled Settings

A key obstacle in evaluating a rule miner like eXpose is the lack of ground-truth information for traces in the wild. To circumvent this, we ran eXpose on a three hour trace from a single client desktop. We discovered all the expected communication rules including, the dependence with DNS; the accesses to the yp server during logins; and the rules for NFS access. Unexpectedly, we found dependencies for certain often browsed web-sites. eXpose found rules for how advertisements and images are synchronously fetched from other servers whenever the client browsed the main page of a web site. Further, we injected artificial traffic wherein pairs of flows at random would either be independent of each other or dependent. eXpose was successful at discovering rules for the dependent flows. We also crafted traffic that occurs together always but is separated by a time gap greater than eXpose's choice of time window size. As expected, eXpose did not discover these dependencies. Similar to prior work [1, 10], we share the belief that *dependent yet separated by long time gap* flow pairs are not common in practice and defer finding such pairs to future work.

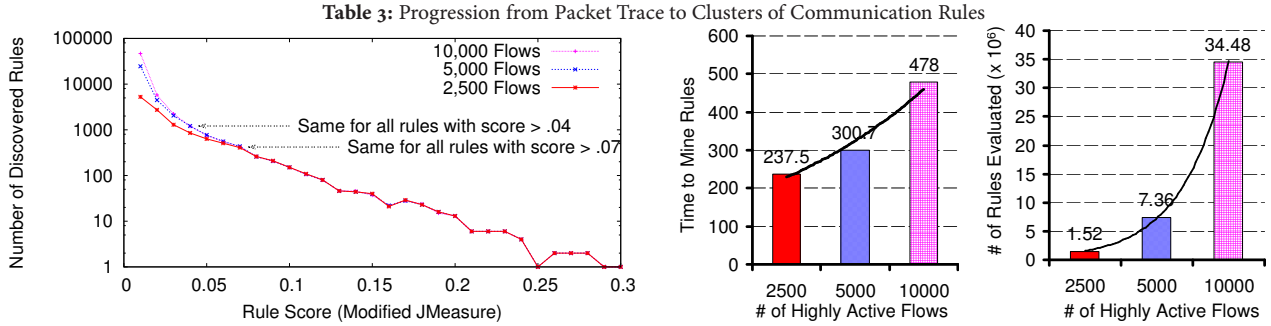
4.5 Micro-Evaluation

We first evaluate some of eXpose's design choices.

Is Selective Biasing Useful? Recall eXpose selectively biases the search to avoid pairs that are unlikely to be dependent. While the details of how eXpose picks rules to evaluate are elsewhere (§2.2), here we verify its usefulness. On the HotSpot-1 trace, eXpose finds 20,094 significant rules from among the 513,647 flow pairs that it evaluates for a *hit-rate* of 3.9×10^{-2}. Doing away with some of our biasing constraints (specifically the constraint that flow pairs have

Trace	# Flow Pairs	#Generics Added	#Rules Evaluated	#Rules Output	#Rule-Clusters
LabAccess	6.76×10^{12}	730	944,681	31,904	301
LabEnterprise	8.26×10^{11}	830	899,034	29,797	346
Enterprise	3.88×10^{11}	1,952	2,000,089	6,289	504
HotSpot-1	4.17×10^{10}	761	513,647	20,094	101
HotSpot-2	3.64×10^{11}	1,040	662,134	1,409	151

Table 3: Progression from Packet Trace to Clusters of Communication Rules



(a) #Discovered Rules vs. #Highly Active Flows Considered (K) (b) Time to Mine (c) Memory Footprint

Figure 7: Sensitivity of eXpose to the number of highly active flows (K, §2.2). Recall that eXpose mines over the top $K = 5000$ highly active flows by default. The more statistically significant a rule (higher score), the fewer the number of active flows eXpose has to consider to discover the rule!

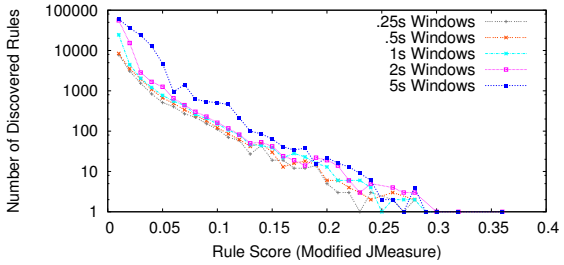


Figure 8: Sensitivity of eXpose to the size of time windows used in discretizing the trace (§2.1). Window sizes in the [.25s, 2s] range seem to not change results appreciably, verifying the assumption that most dependent flows happen within a short duration of each other.

atleast one matching IP to be considered for a rule) caused eXpose to evaluate an order of magnitude more (5,409,049) flow pairs. Yet, only an additional 165 rules were found for a much smaller hit-rate of 3.3×10^{-5} and a large increase in the time to mine.

How Sensitive is eXpose to the Number of Active Flows? Recall that eXpose mines over the top $K = 5000$ highly active flows by default. But, do the discovered rules change appreciably if eXpose mines over many more or many fewer flows? Fig. 7 plots the number of discovered rules and their scores when eXpose mines over the LabEnterprise trace with three different choices of K. As eXpose mines over more active flows, its search space broadens and it discovers more rules. But, the more statistically significant a rule (higher score), the fewer the number of active flows eXpose has to consider in order to reveal that rule! In Fig. 7 (a), we see that when eXpose mines over twice the default number of highly active flows ($K = 10,000$), none of the new rules have score higher than .04. Figures 7(b, c) show the cost of expanding the search space. Since eXpose has to now evaluate many more rules, the time to mine for rules increases by 59% (from 300.7s to 478s) and the memory footprint increases by 368%. We believe that an administrator can choose the number of active flows to tradeoff higher resource cost for better fidelity in revealing the less significant rules. We picked $K = 5000$ as the default.

Why pick $1s$ wide time-windows? Fig. 8 plots the number of rules discovered at each score level by eXpose on the LabEnterprise trace when discretizing at time windows of different sizes. We see that smaller windows lead to fewer rules while larger windows lead to more rules, at almost every score level. The reason is twofold. First, at larger time windows, eXpose would discover rules for more dependent flows, i.e., all those that are separated by no more than the time window. Second, since more flows are present in each of the larger time windows on average, it is more likely that a flow pair will co-occur merely due to chance. Lacking ground truth it is hard to distinguish between these two cases—whether a rule that was newly discovered at a larger window size is a true rule that was missed at the smaller window size or is a false positive that was added merely by chance. Regardless, at all window sizes in the [.25s, 2s] range, eXpose reveals very similar rules, showing that dependencies that occur within a short period are highly stable.

4.6 Case Study: Patterns in the Enterprise

In every one of our traces, we found patterns of typical behavior describing the network setup and the dependencies between applications, and also patterns of atypical behavior indicating configuration problems or malicious activity. We have been running eXpose inside Microsoft on the link connecting a thousand node research LAN with the rest of the corporate network (Fig. 1d) for a few months. Here, we present rules learnt by eXpose on a 3hr trace.

Load Balancing: Perhaps the most significant pattern in the Enterprise trace was due to the proxies used for external web access. There are about thirty different proxies, and eXpose found this rule cluster of generics:

$$Proxy1.80 : *.* \Rightarrow Proxy2.80 : *.*;\ Proxy3.80 : *.*;\ \ldots$$
$$Proxy2.80 : *.* \Rightarrow Proxy1.80 : *.*;\ Proxy3.80 : *.*;\ \ldots$$
$$\ldots \quad \text{and so on.}$$

This rule-cluster was a clique; the generic corresponding to each proxy was linked by a rule to every one of the generics corresponding to the other proxies. These rules show that whenever a client talks to one of the proxy servers, the client is very likely to talk to some other proxy server. The explanation for this rule is simple – a client browsing the web fetches multiple HTTP objects in quick succession, and the load-balancing aspect of the proxy cluster spreads the client's HTTP requests across multiple proxy servers.

We note two things. First, this rule could not be discovered without abstraction. No one client talks to all the proxies, nor does any one client browse for very long. Second, without any knowledge of the network setup in the Enterprise, eXpose reveals the existence of load balancing web proxies and further shows that the load balancer

works fairly well. Such information helps in understanding the network setup and in troubleshooting performance problems.

Within the Enterprise, many services are load-balanced. We found similar load-balancing patterns for DNS servers on port 53, WINS (windows name lookup) servers on port 137 and domain-controller servers (certificates) on port 138.

Application Dependencies: We believe that one of the main contributions of eXpose is the ability to automatically find dependencies for all the important applications. eXpose learnt rules for web activity, email viewing, video lecture broadcasts, printers and dependencies at all the prominent servers.

eXpose discovered rules indicating that name lookup is a key ingredient of activities ranging from distributed file-systems to email and web access.

$$*.* : Server.port \quad \Rightarrow \quad *.* : DNS.53$$
$$*.* : Server.port \quad \Rightarrow \quad *.* : WINS.137$$

The first rule shows that a client accessing any server does a DNS lookup for names, while the latter shows the client doing a WINS (windows name server) lookup. Names in the Enterprise are distributed across DNS and WINS tables. Such rules were ubiquitous for every one of the servers mentioned in the rest of the trace, so we omit further mention.

Web: For web-browsing, we found that a majority of the proxies requested authentication credentials before access.

$$Proxy1.500 : *.* \Rightarrow Proxy1.80 : *.*$$

This generic rule shows that when fetching HTTP data (at port 80) from a proxy, clients exchange Kerberos credentials (at port 500).

E-Mail: eXpose found dependencies for e-mail access.

$$Client.* : Mail.135 \quad \Rightarrow \quad Client.* : DC.88$$
$$Client.* : Mail.135 \quad \Rightarrow \quad Client.* : Mail.X$$
$$Client.* : Mail.X \quad \Rightarrow \quad Client.* : PFS_1.X, Client.* : PFS_2.X$$
$$Client.* : Mail.X \quad \Rightarrow \quad Client.* : Proxy.80$$

The rules show that whenever a client talks to a mail server, he talks to a domain-controller (DC); the DC servers on port 88 issue Kerberos credentials after authenticating the user. The second rule shows that the actual mail server is on a "custom port", so the clients first look up the end-point mapper on port 135 at the server to learn which port the actual exchange server process is located at. Enterprise-wide public mail is stored elsewhere on *public folder servers*. Many clients browse through public mail simultaneously with their personal mail. Finally, we found that most mail contains HTTP content, or has links to content, so the users connect to the proxies when reading mail.

A couple of points are worth noting here. The e-mail dependencies are the first instance of multi-host dependencies and show the ease with which eXpose extends across hosts and applications. eXpose discovered these rules without being told that email was an important activity and without knowing which ports/servers are involved in email. Clients within the Enterprise are distributed across multiple mail servers, so we found multiple instances of the above rules, one for each mail server. Finally, eXpose found the port used by most of the exchange servers and our admin was particularly interested in the exceptions.

$$*.* : Mail_1.135 \Rightarrow *.* : Mail_1.49155$$

Most mail servers were running out of a default configuration file that created exchange server processes on port 49155. The few exceptions were probably due to legacy servers that had out-of-date configurations. Ability to detect such fine-grained configuration helps to understand and debug problems.

File-Servers: For Windows SMB (think NFS for Windows), eXpose discovers that

$$SMBServer.445 : *.* \Rightarrow SMBServer.139 : *.*.$$

The rule indicates that clients ask which server has the file by querying the server (at port 139) which in turn responds with a file-handle that the clients use to fetch the file (at port 445). Of course, the file-handle may point to another server.

$$OtherSMBServer.445 : *.* \Rightarrow SMBServer.139 : *.*$$

Such SMB redirection is common practice so that logical names share a hierarchy, while the files are distributed across servers. eXpose uncovered this tangle of re-directions.

Video Lecture Broadcasts: eXpose discovered multiple cliques corresponding to video lecture broadcasts.

$$Video.rtsp : Client_1.* \quad \Leftrightarrow \quad Video.rtsp : Client_2.*$$
$$Video.rtsp : Client_1.* \quad \Leftrightarrow \quad Video.rtsp : Client_3.*$$
$$Video.rtsp : Client_2.* \quad \Leftrightarrow \quad Video.rtsp : Client_3.*$$
$$\ldots \quad \text{and so on.}$$

It turns out that the Video server live-streamed talks and other events within the enterprise over Real Time Streaming Protocol (rtsp). Each clique corresponded to the audience of a particular talk. Rules linking a pair of clients who were tuned in for a long overlapping period had higher scores than those involving clients who tuned out of the broadcast quickly.

Mailing List Servers: eXpose discovered multiple instances of cliques involving email servers.

$$ListServ.* : Client_1.* \quad \Leftrightarrow \quad ListServ.* : Client_2.*$$
$$ListServ.* : Client_1.* \quad \Leftrightarrow \quad ListServ.* : Client_3.*$$
$$ListServ.* : Client_2.* \quad \Leftrightarrow \quad ListServ.* : Client_3.*$$
$$\ldots \quad \text{and so on.}$$

It turns out that each of these mail servers was responsible for a particular mailing list. Whenever a mail would be sent to the list, the mail server forwards the mail onto all the participants of that list. eXpose discovered when each list was active and the participants for each list.

DHCP: Clients that were searching for a DHCP server by broadcasting on the network caused the pattern,

$$NetworkBroadcast.137 : *.* \Rightarrow DHCPServer.137 : *.*.$$

Printers: eXpose found a clique involving print spoolers:

$$IP_1.161 : PrintServ.* \quad \Leftrightarrow \quad IP_2.161 : PrintServ.*$$
$$IP_1.161 : PrintServ.* \quad \Leftrightarrow \quad IP_3.161 : PrintServ.*$$
$$IP_2.161 : PrintServ.* \quad \Leftrightarrow \quad IP_3.161 : PrintServ.*$$
$$\ldots \quad \text{and so on.}$$

It turns out that each of the IP's corresponded to the network interfaces of the printers throughout the Enterprise. The print-server appears to periodically poll the SNMP (161) port of these printers for usage and other information.

Workload Clusters: eXpose found cliques for file-servers.

$$Pool_1.* : FileServ.445 \quad \Leftrightarrow \quad Pool_2.* : FileServ.445$$
$$Pool_1.* : FileServ.445 \quad \Leftrightarrow \quad Pool_3.* : FileServ.445$$
$$Pool_2.* : FileServ.445 \quad \Leftrightarrow \quad Pool_3.* : FileServ.445$$
$$\ldots \quad \text{and so on.}$$

File-servers showing up in the above rules are centralized data stores for various groups. They were accessed by pools of clients who were crunching the data in parallel.

Presence Server: eXpose found many such rules,

$$Presence.5601 : Client_1.* \Leftrightarrow Presence.5601 : Client_2.*$$

It turns out that a presence server is part of Windows Office Com-

municator. Whenever a client logs in, logs out or goes idle, his machine sends an update to the presence server which forwards the information onto the user's friends. So, each rule above links users who are in each others *friend list*.

Oddities: eXpose found that a particular user Bob's machine does:

$$\begin{aligned} Bob.* : Home_1.* &\Leftrightarrow Bob.* : Home_2 \\ Bob.* : Home_1.* &\Leftrightarrow Bob.* : Home_3 \\ Bob.* : Home_2.* &\Leftrightarrow Bob.* : Home_3 \\ &\ldots \text{ and so on.} \end{aligned}$$

It turns out that Bob's machine was talking to many IPs belonging to DSL users and cable-modem pools (Verizon DSL Customers, Road Runner Customers). eXpose found this pattern because the accesses to these home machines were periodic and synchronized. Either Bob is running an experiment that probes many home machines, or he is part of a zombie bot-net and the communication here is keep-alive messages between bots in the botnet.

4.7 Case: Patterns in the LabEnterprise

We have deployed eXpose within the CSAIL lab at MIT such that eXpose sees traffic to and from the major internal servers, including web, email, Kerberos, AFS, NFS servers, managed user workstations and a Debian/Fedora mirror (see Fig. 1c). Here we report rules learnt by eXpose on a 400 minute trace.

Fig. 9 depicts rules learnt by eXpose. Graph nodes correspond to a flow or a generic in the trace, an edge between nodes indicates that eXpose discovered a rule linking the two nodes. The graph on the left plots the output of eXpose's rule-mining algorithm–i.e., all the statistically significant rules, whereas the graph on the right plots the output of our recursive spectral partitioning algorithm. We found in practice that eliminating weak rules which strangle otherwise disconnected node groups reduces false positives.

syslogd clique: All the managed workstations within the lab and most of the internal servers ran the same version of Debian with similar configuration. In particular, these machines export their system log files to a server via port 514. Cron jobs in the config ran at specific times causing synchronized updates to the individual system logs and synchronized updates to the syslogd server.

$$\begin{aligned} IP_1.514 : Syslogd.514 &\Leftrightarrow IP_2.514 : Syslogd.514 \\ IP_1.514 : Syslogd.514 &\Leftrightarrow IP_3.514 : Syslogd.514 \\ IP_2.514 : Syslogd.514 &\Leftrightarrow IP_3.514 : Syslogd.514 \\ &\ldots \text{ and so on.} \end{aligned}$$

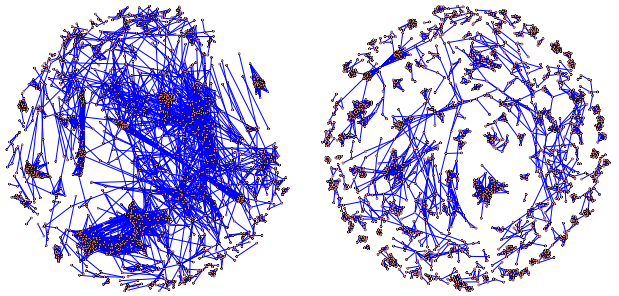
As eXpose could list all the machines uploading logs, the admin could chase down machines that were supposed to be managed but were not uploading syslog files synchronously.

AFS accesses: The following pattern repeated across many clients:

$$\begin{aligned} Client.7001 : AFSRoot.7003 &\Rightarrow Client.7001 : * \\ Client.7001 : AFS_1.7000 &\Rightarrow Client.7001 : AFS_2.7000 \\ Client.7001 : AFS_1.7000 &\Rightarrow AFS_1.7000 : AFSRoot.7002 \end{aligned}$$

These rules show that a client talks to the root server (at port 7003) to find which of the many volume servers have the files he needs and then talks to the appropriate volume server. The lab has a single root server but many tens of volume servers with files distributed across the volume servers. A user's content is often spread across more than one servers, causing simultaneous accesses from his cache manager (at port 7001) to the AFS servers (at port 7000). Finally, creating new files, or browsing into different parts of the AFS tree initiate connections to a permissions server (at port 7002).

Besides identifying the underlying structure of AFS traffic, eXpose's rules let us understand where each user's content was located and how the accesses were spread across the various servers. The lab admins were happy to see that the accesses to volume servers matched up with the hard-disk sizes on the servers (larger servers got more accesses).



(a) Rules Discovered By Mining (b) After Spectral Partitioning

Figure 9: Efficacy of Rule Pruning. The graph of left represents rules discovered by mining. Each edge corresponds to a discovered rule and joins nodes corresponding to the <flow, activity> pair involved in the rule. The graph on the right depicts the same rule-set after recursive spectral partitioning. Eliminating low-scored rules between node-groups that are otherwise strongly connected breaks the rule-set into understandable pieces.

File-Server as the Backend: The fact that many users have all their data on AFS leads to many neat patterns:

$$\begin{aligned} Remote_1 : WebServ.80 &\Rightarrow WebServ.7000 : AFS.7001 \\ Client : LoginServ.22 &\Rightarrow LoginServ.7000 : AFS.7001 \\ Compute_1 : AFS.7001 &\Leftrightarrow Compute_2 : AFS.7001 \\ Compute_1 : AFS.7001 &\Leftrightarrow Compute_3 : AFS.7001 \\ Compute_2 : AFS.7001 &\Leftrightarrow Compute_3 : AFS.7001 \end{aligned}$$

The first shows that accesses to home pages on the lab's web-server cause the web-server to fetch content from the AFS servers. While the home-pages are stored on the web-server, it is likely that users link to content (papers, cgi-scripts) present elsewhere in their AFS share. Also, ssh connections into the lab's login server caused the login server to mount directories from AFS. Finally, compute clusters synchronously accessed many volume servers, most likely because a data-intensive job was parallelized across the cluster.

E-Mail Flow: eXpose discovered how email flows through the lab.

$$\begin{aligned} Incoming.25 : *.* &\Rightarrow Webmail.2003 : Incoming.X \\ *.* : Webmail.143 &\Rightarrow *.* : Webmail.993 \\ *.* : Webmail.110 &\Rightarrow *.* : Webmail.995 \end{aligned}$$

The first rule shows that whenever a connection is made on port 25 (SMTP) to a mail server in the lab, the mail server connects with different server. It turns out, the lab has one Incoming SMTP server that receives mail from outside but does not store the mail. Instead, mail is forwarded on to a webmail server via LMTP (enterprise version of SMTP, port 2003). The webmail server in turn provides an interface for users to read their mail. The next two rules show the webmail server responding to both IMAP (port 143) and POP (port 110) connections only to force clients to use the corresponding secure versions IMAPS (port 993) and POPS (port 995).

Outgoing E-mail: eXpose tracks patterns in outgoing mail.

$$\begin{aligned} OutMail.* : dns0.mtu.ru.53 &\Leftrightarrow OutMail.* : dns1.mtu.ru.53 \text{ (many)} \\ OutMail.* : Mail_1.25 &\Leftrightarrow OutMail.* : Mail_2.25 \text{ (clique)} \end{aligned}$$

The first rule above shows that whenever the outgoing server does a DNS (port 53) lookup at one remote name server, it does a name lookup on another redundant name server in the same domain. It turns out that the lab's mail server implementation simultaneously looks up multiple DNS servers in the hope that at least one of them would respond. Further, the second rule shows the outgoing mail server simultaneously connecting to multiple mail servers in domains like *messagelabs.com*. Apparently, messagelabs (and many others) are out-sourced email providers who receive email for companies(e.g., Akamai). Several lab mailing lists have employees of these companies, and hence a mail to such lists makes the outgo-

ing server deliver mail simultaneously to multiple SMTP servers at the outsourced email provider.

Nagios Monitor: The Lab admins use a Nagios machine to monitor the health of the lab's key servers.

$$Nagios.7001 : AFS_1.7000 \Rightarrow Nagios.7001 : AFS_2.7000 \text{ (AFS clique)}$$
$$Nagios.* : Mail_1.25 \Rightarrow Nagios.* : Mail_2.25 \text{ (mail clique)}$$
$$Nagios.* : AD.139 \Rightarrow Nagios.* : AD.389 \text{ (active directory)}$$

These rules show that the Nagios machine periodically connects to the servers of each type as a *client* and probes their health. The first rule is part of a full clique, the Nagios machine connected as an AFS client with every one of the AFS servers in the lab. Similarly, the second rule is part of a clique wherein the Nagios box sent mail out through each of the SMTP servers in the lab. The third rule shows Nagios checking the windows server for both its name (netbios, port 139) and LDAP services (directory lookup, port 389).

Discovered Bugs: eXpose discovered many instances of configuration problems and malicious users that the administrators were able to act upon. Here are examples of two such rules.

(1) IDENT Dependency: eXpose found many instances when a client's connection to a server is followed by the server initiating a connection at port 113 on the client. For example,

$$Client.* : MailServer.25 \Leftrightarrow Client.113 : MailServer.*$$

It turns out that port 113 is IDENT traffic.

> Despite the fact that IDENT was never very useful, even today …UNIX servers, most commonly IRC Chat, but some eMail servers as well, still have this IDENT protocol built into them. Any time someone attempts to establish a connection with them, that connection attempt is completely put on hold while the remote server attempts to use IDENT to connect back to the user's port 113 for identification [7].

Shown this rule, an admin changed configuration to disable IDENT.

(2) Legacy Configuration: A legacy DHCP server was active and responding to requests on a subnet. Even more, the legacy server was accessing the lab's authoritative name server that was supposed to be used only by the front-end DNS servers, which pull the master name table and respond to clients. Shown this rule, an admin disabled the legacy server.

$$NetworkBroadcast.68 : Legacy.67 \Rightarrow Legacy.* : MasterNS.53.$$

False Positives: We were able to corroborate 45 out of the 346 rule groups that eXpose discovered on the labEnterprise trace, for a false positive rate of 13%. Some of the false positives were due to dependencies that appeared true but we could not explain such as pairs of flows that always happened together in 5% of time windows. Others were due to high volume servers, such as a debian mirror hosted within the lab. There are so many connections to the debian mirror server from so many different IPs that invariably we find many connections overlapping with each other. We do know that many operating systems have software that automatically touches the distribution mirrors at fixed times of day to check for updates, yet it is hard to say for sure why a pair of IPs access the debian mirror synchronously. A natural improvement to eXpose is to automatically scale the score threshold per server, i.e., high volume servers that are at a risk of false positives are reported only if they are in rules that have much higher scores.

False Negatives: Backend dependencies at the web-server hosting personal web-pages were too fine-grained for eXpose. We discovered the bulk dependency, i.e., that most web-page requests cause the web-server to fetch content from AFS. But, we were unable to isolate the more complex dependencies of individual web-pages carrying dynamic content. We believe that this is because of too few connections to such dynamic content and readily admit that eXpose is likely to miss specific dependencies while looking for the broad ones. But, one can *white-list* servers that eXpose should pay closer attention to by adding that server's flows to the list of flows that eXpose builds rules for.

4.8 Case: Patterns on the Lab's Access Link

mySQL worm: We found an instance of the mySQL worm; the host pD9E9D641.dip.t-dialin.net performed a port scan throughout the network on port 3306 which is the default mySQL port on Unix.

> The mySQL Bot scans this port looking for mySQL servers with weak passwords and if it is successful in logging in as root . . . uses an exploit to install the bot on the system. [13].

Unlike other worms such as the SQL Sapphire Worm [20], this worm causes little traffic and may not show up in tools that detect heavy-hitters. Yet, eXpose detects it because the remote host scanned many lab hosts simultaneously.

Cloudmark: The mail server of a group in the lab was involved in an interesting pattern. Whenever the server received mail (at port 25), the server would connect to one of a bunch of servers owned by cloudmark.com at port 2703:

$$MailServer.25 : *.* \Rightarrow MailServer.* : CloudMark.2703.$$

Apparently, cloudmark is a filtering service for spam, phishing and virus-bearing mails to which this group subscribes.

NTP synchs: The lab runs a major Network Time Protocol (NTP) server. To keep system clocks up-to-date, clients periodically probe an NTP server, and adjust clocks based on the server's response and the round trip time estimate. Client implementations vary widely though. Most clients query the server infrequently and the probe/response pairs are just one UDP packet. But eXpose found

$$Client_1 : NTP.123 \Leftrightarrow Client_2 : NTP.123,$$

indicating that pairs of clients were accessing the server synchronously and often (one probe in every couple of seconds). Most likely, these machines have the same poorly written NTP client.

TOR: The lab contributes servers to the TOR [3] anonymity network. The ability to identify temporally correlated pairs strips one-hop anonymity, we can identify the next-hop for flows routed through the lab's TOR server. For example, eXpose finds rules,

$$IP_1.9001 : TOR.* \Leftrightarrow TOR.* : IP_2.9001$$

showing that traffic flows from IP_1 to IP_2 or vice versa. TOR's anonymity is not broken though. eXpose-like traffic analysis has to be done at every TOR server on a flow's path to identify the participants. But, this does highlight a weakness–TOR seems to not use cover traffic and since there isn't a lot of traffic to begin with, it is easy to correlate flows across one hop.

FTP Session Dependencies: The lab provides a mirror for both debian and fedora linux distributions. Clients around the world download OS images and packages. eXpose found that

$$IP_1.* : Mirror.* \Rightarrow Mirror.21 : IP_1.*.$$

It turns out that an ftp control connection (on port 21) to exchange commands precedes ftp data connections that do the actual data transfer. Further, data connections are started either actively, i.e. started by the server on port 20, or passively, i.e., started by clients at ephemeral ports. The rules show that most data connections, in practice, are passive perhaps to let clients behind NATs access data.

Discovered Bugs: Again eXpose discovered exploits and configuration problems. **(1) Legacy Addresses in Mailing Lists:** We found

that our university's outgoing mail server was simultaneously accessing a couple of mail servers in the lab.

$$UnivMail.* : OldMail_1.25 \Leftrightarrow UnivMail.* : OldMail_2.25$$

This was consistent with other mailing list patterns we had seen, the university server was delivering mail to a list that had users at those machines in the lab. Unfortunately, these older mail servers were no longer operational and the email addresses had been invalid for a long time. When shown this rule, the university's admin responded that the mailing lists would be cleaned.

(2) Selective SSH Scans: eXpose identified several hosts in South Asia that were selective scanning the lab's main routers.

$$*.* : Router_1.22 \Leftrightarrow *.* : Router_2 : 22 \text{ (three router clique).}$$

During the scan a host would simultaneously initiate ssh connections and try login/password pairs on all the main routers. eXpose also found co-operative scanning, wherein multiple hosts would scan a router at the same time.

$$Attack_1.* : Router.22 \Leftrightarrow Attack_2.* : Router.22$$

Given the patterns, the lab's admin blacklisted the scanning IPs.

(3) Web Robots: eXpose discovered rules for web crawlers.

$$Robot.* : Web_1.80 \Leftrightarrow Robot.* : Web_2.80$$

These rules indicate that a crawler bounces between multiple web-servers perhaps as it follows links on the lab's web content. Most of the robots belonged to well-known search sites, but one of them was a machine in south-east asia that had neither a name record nor was pingable after the fact. These robots neither make too many connections nor pull down a lot of data and hence are indistinguishable from normal web traffic. eXpose found them by their characteristic access pattern–synchronous accesses to multiple web-servers while chasing down links. The lab's admin flagged the *unknown IP* to be inspected more carefully by the intrusion detection box.

False-Positives: Our lab hosts Planetlab machines. Some dependencies, such as access to the CODEEN CDN are discernible. Yet, it is almost impossible to figure out from the packet trace which currently active slice caused which packets. So, we did not attempt to corroborate rules involving Planetlab.

4.9 Case Study: Rules for HotSpot Traces

Note that both the Sigcomm'04 and OSDI'06 traces are anonymized, so we corroborated rules based on the port and protocol numbers of flows. Fig. 6 graphically shows all the patterns that eXpose learned from the Sigcomm'04 trace.

Peer-to-peer Traffic: Most of the high-density clusters were due to peer-to-peer traffic. In Fig. 6, the two large clusters were due to two wireless hosts using BitTorrent, whereas the two smaller clusters were due to Gnutella users. Each of these hosts connected to many tens of unique peers. eXpose found that whenever a peer communicates on one of the ports in the 6881-6890 range, the peer is likely to communicate on another port in the same range. Presumably, this is due to multiple flows between the host and BitTorrent peer. Gnutella's clusters are similar, except in a different and smaller port range; most flows here are on ports 6346-6349.

Suspicious activity on port 1081: One of the wireless hosts communicated synchronously with four different machines on port 1081. Popular wisdom [18] says that the port is used by the WinHole–"A trojanized version of Wingate proxy server". The traffic volume of each of these flows is fairly small, yet eXpose discovered the pattern from what appear to be synchronous heart-beat messages between this victim and other machines:

$$Victim.* : Other_1.1081 \Leftrightarrow Victim.* : Other_2.1081$$

DHCP Servers: Almost all the clients in the OSDI trace were involved in a simple pattern; the client sends out a broadcast message to port 68 and gets a response from either IP_1 or IP_2.

$$*.67 : 255.255.255.255.68 \Leftrightarrow IP_1.68 : *.67$$

It appears that both IP_1 and IP_2 carry a DHCP Server at the well known port 68. The DHCP daemon responds to requests for new IP addresses sent by DHCP clients from port 67. DHCP traffic is quite infrequent and involves few bytes, yet the synchronous accesses in time lead to this rule. Further, note that eXpose discovered the rule with no knowledge of what to expect in the trace.

Apple iTunes: eXpose found hosts talking on port 5353.

$$H_1.5353 : H_2.5353 \Leftrightarrow H_1.5353 : H_3.5353; H_1.5353 : H_4.5353$$

It turns out that the Apple iTunes application advertises to other Apple machines on the subnet if configured to share its music. Some users appear to have forgotten to disable this feature causing their laptops to advertise music at the conference. eXpose discovers this rule by the temporally correlated advertisements and had no knowledge of iTunes before hand.

Link-level Multicast Name Resolution: We found what appears to be a new form of looking up names. We saw earlier that windows hosts query both the local DNS server and the local WINS server to lookup names. In addition, eXpose observed these rules:

$$\begin{aligned} Host.* : Multicast.5355 &\Leftrightarrow Host.* : DNS.53 \\ Host.137 : WINS.137 &\Leftrightarrow Host.* : DNS.53 \\ Host.137 : WINS.137 &\Leftrightarrow Host.* : Multicast.5355. \end{aligned}$$

A few hosts were sending out packets to a multicast address on port 5355 along with the other name lookups. It turns out that this is link-level multicast based name resolution—a new feature in Vista [12] that is designed specifically for ad-hoc networks. Of course, this is the first time we ever heard of this protocol.

Day-Long Traceroutes: We found two hosts sending what appear to be day-long traceroutes. eXpose discovered these rules.

$$\begin{aligned} Host.0 : IP_1.0 : 1 &\Leftrightarrow Host.0 : IP_2.0 : 1;\ Host.0 : IP_3.0 : 1 \ldots \\ Host.0 : IP_2.0 : 1 &\Leftrightarrow Host.0 : IP_3.0 : 1;\ Host.0 : IP_4.0 : 1 \ldots \end{aligned}$$

The rules show that $Host$ was receiving ICMP (protocol 1) messages from a bunch of IPs all within the same one-second period and repeated many times throughout the day. Our best guess is that somebody in the conference was doing some measurements, maybe to check if a certain path changes during the day. eXpose found this low-volume event and also the fact that the path did not change.

IM: eXpose found these rules:

$$Host.* : MSNServ.1863 \Leftrightarrow Host.* : AOLServ.5190.$$

It turns out that ports 1863 and 5190 are well known ports for MSN and AOL Instant Messaging Servers respectively. It appears as if a couple of users were using aggregated instant messaging (IM) clients like GAIM [6] that can connect to multiple IM networks. There is little traffic in these flows, the IM client appears to refresh the servers periodically and synchronously leading to the rule.

5. POSSIBLE EXTENSIONS OF eXpose

From interactions with our administrators, we have some ideas on how to integrate eXpose into everyday operations. We see much commonality in patterns. We saw eXpose extract similar rules from different locations (e.g., the IMAP-IMAPS, POP-POPS) rules. Even more, a vast majority of the rules extracted on different days but at the same location are similar. This suggests that we should build a database of known rules at each location. Rules extracted on each new packet trace, perhaps in a streaming fashion, can be matched with existing rules in the database. This lets the administrators focus on the *novel* patterns and also gain further confidence in patterns that are seen repeatedly.

6. RELATED WORK

A few tools aggregate traffic volumes and visualize the resulting aggregates. FlowScan [17] takes as input NetFlow data and breaks down the traffic volume according to the application (e.g., HTTP, FTP), the IP prefix, or the AS identifier. CoralReef [2] and IPMON [8] produce similar traffic breakdowns based on packet traces. Autofocus [5] also breaks the traffic volumes into different categories but adapts the breakdown boundaries to zoom-in or out on interesting subnets and port ranges. eXpose extends these tools along a new dimension by identifying temporally correlated clusters of flows.

Other related work attempts to find traffic matching a pre-defined communication pattern. Venkataraman et.al. [24] and Staniford et.al. [22] present streaming algorithms to identify SuperSpreaders, i.e., machines infected by a worm or virus that in turn infect many other hosts. Another line of work [25, 26] detects *stepping stones* whereby an attacker compromises a machine and uses it to launch a very different attack on other machines. Blinc [11] uses hints from multiple levels to tag each flow with the application that created it. More generally, intrusion detection systems like Bro [16] use a database of signatures for malicious activity and find matching traffic. Rather than identifying traffic that matches a given pattern, eXpose automatically extracts the underlying patterns in a trace.

Perhaps the closest to eXpose is work that finds detailed dependencies for individual applications. Kannan et. al. [10] analyze network traces to identify the structure of a particular application session (e.g., FTP, HTTP). For individual applications between one source and one destination, they build state machines detailing how the session progresses. When pointed at a server, Sherlock [1] finds dependencies for clients accessing that server even when such dependencies involve multiple other servers or protocols.

Fundamentally, eXpose is different as it finds dependencies without guidance. Without pre-focusing on any given application or server, eXpose looks for all statistically significant clusters of flows. This lets eXpose find patterns that spread across multiple hosts, protocols and applications, and even those that an admin did not know or expect such as configuration errors. Both Sherlock [1] and Kannan et. al. [10] can bring out detailed dependencies that eXpose's broader search might not highlight. But, to obtain output similar to eXpose, one would have to repeatedly apply these techniques to learn dependencies for one server or one application at a time. This is unlikely to scale and also misses out on dependencies at servers that the admin may forget to point towards.

7. CONCLUSION

We advance the state-of-the-art in traffic analysis by presenting a general mechanism to identify temporally correlated flows in a packet trace. While just looking at temporally correlated flows is unlikely to capture the myriad kinds of structures in a packet trace, we show that this is a powerful primitive that is able to capture many useful patterns. Our tool eXpose uniquely defines the concept of generic rules, focuses only on the statistically significant flow pairs and presents an algorithm that scales to large traces. Results from deploying eXpose within MIT and Microsoft Research show that eXpose uncovers many configuration errors and lets operators get a quick read of what is going on in their network without having to understand logs from the various servers.

8. ACKNOWLEDGMENTS

We thank Noah Meyerhans, Garrett Wollman at CSAIL and Geoffrey Nordlund, Xu Chen at Microsoft Research for their help in collecting the traces and validating the discovered communication rules. We thank the collectors of Sigcomm'04 and OSDI'06 traces. We also thank Albert Greenberg, Arthur Berger and Asfandyar Qureshi for insightful comments on early drafts. This work was supported in part by Microsoft Research and by NSF Career Award CNS-0448287. Opinions and findings in this paper are those of the authors and are not necessarily shared by NSF or Microsoft.

9. REFERENCES

[1] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards Highly Reliable Enterprise Network Services via Inference of Multi-level Dependencies. In *SIGCOMM*, 2007.
[2] CoralReef - Workload Characterization. http://www.caida.org/analysis/workload/.
[3] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security*, 2004.
[4] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. John Wiley, 2002.
[5] C. Estan, S. Savage, and G. Varghese. Automatically Inferring Patterns of Resource Consumption in Network Traffic. In *SIGCOMM*, 2003.
[6] GAIM/Pidgin. http://www.pidgin.im/.
[7] IDENT. http://www.grc.com/port_113.htm.
[8] IPMON. http://ipmon.sprintlabs.com.
[9] Jose Bernardo and Adrian F. M. Smith. *Bayesian Theory*. John Wiley, 2000.
[10] J. Kannan, J. Jung, V. Paxson, and C. E. Koksal. Semi- Automated Discovery of Application Session Structure. In *IMC*, 2006.
[11] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: Multilevel Traffic Classification in the dark. In *SIGCOMM*, 2005.
[12] Link-Level Multicast Name Resolution. http://www.windowsnetworking.com/articles_tutorials/Overview-Link-Local-Multicast-Name-Resolution.html.
[13] PortPeeker Capture of mySQL Bot attack. http://www.linklogger.com/mySQLAttack.htm.
[14] Nagios: Host, Service, Network Monitor. http://nagios.org.
[15] T. Oetiker and D. Rand. Multi Router Traffic Grapher. http://people.ee.ethz.ch/~oetiker/webtools/mrtg/.
[16] V. Paxson. Bro: A System For Detecting Network Intruders in Real-Time. *Computer Networks*, 1999.
[17] D. Plonka. Flowscan: A Network Traffic Flow Reporting and Visualization Tool. In *USENIX System Admin. Conf.*, 2000.
[18] Port 1081. http://isc.incidents.org/port.html?port=1081.
[19] P. Reynolds, C. Killian, J. L. Wiener, J. C. Mogul, M. A. Shah, and A. Vahdat. Pip: Detecting The Unexpected in Distributed Systems. In *NSDI*, 2006.
[20] Analysis of the Sapphire Worm. http://www.caida.org/analysis/security/sapphire/.
[21] P. Smyth and R. M. Goodman. *Knowledge Discovery in Databases*. MIT Press, 1991.
[22] S. Staniford, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS: A Graph-based Intrusion Detection System for Large Networks. In *National Information Systems Security Conference*, 1996.
[23] S. Vempala, R. Kannan, and A. Vetta. On Clusterings Good, Bad and Spectral. In *FOCS*, 2000.
[24] S. Venkataraman, D. Song, P. Gibbons, and A. Blum. New Streaming Algorithms for Fast Detection of Superspreaders. In *NDSS*, 2005.
[25] K. Yoda and H. Etoh. Finding a Connection Chain for Tracing Intruders. In *ESORICS*, 2000.
[26] Y. Zhang and V. Paxson. Detecting Stepping Stones. In *USENIX Security*, 2000.