Discovering Dependencies for Network Management

Paramvir Bahl, Paul Barham, Richard Black, Ranveer Chandra, Moises Goldszmidt,
Rebecca Isaacs, Srikanth Kandula[†], Lun Li[‡], John MacCormick, David A. Maltz, Richard Mortier,
Mike Wawrzoniak[¶], Ming Zhang.
Microsoft Research; also [‡] Caltech, [†] MIT and, [¶] Princeton.

This paper presents the Leslie Graph*, a simple yet powerful abstraction describing the complex dependencies between network, host and application components in modern networked systems. It discusses challenges in the discovery of Leslie Graphs, their uses, and describes two alternate approaches to their discovery, supported by some initial feasibility results.*

1 Introduction

It is lamentable that Leslie Lamport's famous quote [9] *"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable"* describes a scenario familiar to almost every computer user. As IT systems are increasingly distributed, it is not only the clients and servers themselves that can render a computer useless for an afternoon, but any of the many routers, links and network services also involved.

In distributed systems, the underlying problem is the absence of tools to identify the components that "can render your own computer unusable": the implicit web of dependencies among these components exists only in the minds of the human experts running them. The complexity of these dependencies quickly adds up, requiring more help than traditional IT management software provides. Listing the contents of a single DFS [1] directory, for example, can involve a minimum of three hosts and eight network services (WINS, ICMP Echo, SMB, DFS, DNS, Kerberos, ISA key exchange, ARP). Existing management solutions focus on network elements, topology discovery, or particular services, but what is needed are tools to manage and improve the user's end-to-end experience of networked applications.

In deference to Lamport, this paper defines the *Leslie Graph* as the graph representing the dependencies between the system components, with subgraphs representing the dependencies pertaining to a particular application or activity. Nodes represent the computers, routers and services on which user activities rely, and directed edges capture their inter-dependencies. Different versions of a Leslie Graph can express different granularities of dependence for an activity — for some analyses, an Leslie Graph capturing inter-machine dependences at the granularity of IP addresses might be sufficient, while for others an Leslie Graph capturing inter-service dependencies at the granularity of software processes might be desirable.

This paper makes three contributions: (*i*) we define Leslie Graphs and discuss the challenges in finding them, (*ii*) we suggest important problems that Leslie Graphs could help solve, and (*iii*) we describe two ongoing projects that are exploring different approaches to automatically infer the Leslie Graphs.

[1] Windows Distributed File System

1.1 Existing Approaches

It might seem that the Leslie Graph for an application could easily be constructed if its designer generated rules that spell out the application's dependencies. Indeed, a number of commercial products such as MAM[2] and the DSI "System Definition Model"[3] do just this. However, this approach has several problems: the system could evolve faster than the rules; deployment of various forms of middlebox (e.g., firewalls, proxies) can change the application's dependencies without the rule writers even being aware; and rules are unavailable for legacy systems.

Similarly, analysis of configuration files to determine the Leslie Graph is insufficient as many dependencies among components are dynamically constructed. For example, web browsers on enterprise networks are often configured to communicate through a proxy, sometimes named in the browser preferences but frequently contacted through automatic proxy-discovery protocols that themselves rely on resolution of well-known names.

Systems have been proposed to expose dependencies by requiring all applications to run on a middleware platform instrumented to track dependencies at run-time [1, 4, 7]. However, heterogeneity defeats most such efforts in practice. Networks run a plethora of platforms, operating systems, and applications, often from a wide range of vendors. While a single vendor might instrument their software, it is unlikely that all vendors will do so in a common fashion; similarly, building all distributed applications over a single common middleware platform is infeasible. Furthermore, many underlying services on which others depend are legacy services and cannot easily be instrumented or ported to run over an instrumented layer.

1.2 Challenges Finding the Leslie Graph

In contrast to the above approaches, but also without explicitly defining some notion of a Leslie Graph, others have argued that a promising approach to inferring dependencies is to observe externally visible behavior of system components without parsing the contents of packets they send—the "black-box" approach [2, 13]. We follow this general approach, relying mainly on correlation of observed network

[2] http://www.mercury.com/us/products/business-availability-center/application-mapping/

[3] http://www.microsoft.com/windowsserversystem/dsi/sdm.mspx

traffic to infer system dependencies, and augmenting as required with other techniques such as active probing. However, there are several challenges with this approach.

False positives. The Leslie Graph is expressed in terms of dependency between components, which requires understanding their *causality*. However, using observed traffic results in measuring their *correlation*, which is not the same. For example, it is perfectly possible for unrelated conversations, such as periodic background maintenance traffic, to exhibit misleading timing correlations.

False negatives (caching). Statistical correlations require a substantial number of observations in the presence of noise (unrelated background traffic). However, much critical control plane and session setup behavior occurs relatively rarely. For example, it is difficult to determine that a web-browser's use of HTTP depends on both DNS and ARP through traffic observation alone, as the services are typically invoked once and the results cached.

Granularity. Many modern IT deployments use clusters of servers to implement load-balancing and resilience for critical tasks. Alternatively, in smaller systems multiple services will be hosted as separate processes on a single server. Thus, a vertex of the Leslie Graph might need to represent other than a single computer: at some times an entire cluster of computers will be appropriate, at others a single process on a single computer.

Complexity. Enterprise networks use a wide variety of applications [11], including complex services like authentication (Active Directory, IPSEC, Kerberos, RADIUS), remote file systems (AFS, DFS, NFS, SMB), web applications (Sharepoint, Wikis), communications (VoIP, IM, email) and utilities (printing, DHCP, ARP). The inter-dependencies between these are extensive and poorly specified.

Trust. To compute the Leslie Graph hosts must share information about their activities and are expected to do so truthfully. In an enterprise network this trust can be established and enforced by the company's network policy and administration procedures.

We restrict our subsequent discussion to discovery of Leslie Graphs in enterprise networks precisely because the latter two challenges, complexity and trust, make enterprise networks both a useful and feasible place to do so. We assume that we can place agents on a reasonable fraction of the computers on the network to monitor packets sent and received. These agents can also be used for tomography: taking measurements and enabling probing from many vantage points to discover network topology and resolve ambiguities in the Leslie Graph.

2 Leslie Graphs and Their Uses

Studies show that $\sim$ 70% of enterprise IT budgets are spent on maintenance.[4] The ability to create an enterprise's Leslie Graph could have a major financial impact by enabling the following techniques for management and troubleshooting.

Fault localization. A common source of frustration for users is when an application temporarily hangs for no readily apparent reason. The hardest part of resolving such problems is often locating the problem in the first place. Is it in an overloaded server? A policy configuration? A failed router or link? The Leslie Graph for an application not only summarizes the components that are involved, but also allows information from multiple clients to be combined to pinpoint faults through tomography.

Reconfiguration planning. A classic tale of unexpected consequences [10] involves an old machine configured to backup an SQL database. Since the dependency of the primary server on this old machine for backup service was not explicit, operators re-imaged and recycled the old machine. Unfortunately, the primary server failed around the same time, and the database was completely lost. Companies are continually adding, reorganizing, or consolidating services. Frequently, changes are disruptive to services beyond those directly involved due to unexpected and previously hidden interactions. Planning these changes and diagnosing the problems that inevitably result is expensive.

Leslie Graphs can be expected to help in two ways. First, by automatically detecting dependencies, unexpected consequences can be identified in advance and planned for. Second, Leslie Graphs allow IT departments to warn ahead of time the users who will be affected by changes.

Helpdesk optimization. The fact that many users are active at the same time means that failures are likely to result in many calls to the helpdesk — initiating a new diagnostic effort for each call would be wasteful. Knowing the dependencies among components means that new reports can be rapidly chained to the trouble ticket of a known issue: eliminating time spent investigating dependent issues. It also reduces the likelihood of inappropriate remediation such as unnecessarily rebooting the user's computer, and it helps to prioritize trouble tickets by the numbers of users affected.

Anomaly detection. If Leslie Graphs are automatically constructed based on the observed behavior of hosts, anomalies and changes in the graphs point to hosts that are worthy of more detailed human investigation. For instance, differences between clients can be used to find policy issues. If a set of clients cannot reach a server while everything is fine for another set of clients, our algorithms will localize the problem to the clients. The structure of the Leslie Graph can then help guide a human to determine if the cause is a middlebox/firewall common among the clients or a policy (e.g., IPSEC) on the clients themselves.

[4] Forrester Research, "Governing IT in the enterprise" (July 2004) and http://research.microsoft.com/events/snmsummit

3 Implementation Considerations

We are exploring two different approaches to approximating the Leslie Graph using low-level packet correlations. The Constellation system uses a distributed approach, reactively constructing the Leslie Graph of any node on-demand. In contrast, the AND system, which stands for Analysis of Network Dependencies, proactively maintains the approximate Leslie Graph at a centralized inference engine. The rest of this section describes these systems in more detail.

3.1 Constellation

In the Constellation system, local traffic correlations are inferred by passively monitoring packets and applying machine learning techniques. The basic premise is that a typical pattern of messages is associated with accomplishing a given task. Therefore, it is possible to approximate the Leslie Graph by taking the transitive closure of strongly correlated nodes, and furthermore we can detect or diagnose faults by observing the *absence* of expected messages.

In order to explore the class of machine learning approaches that are applicable, we have formalized the problem. Space precludes a complete presentation, but the following three concepts are critical:

Channel. A channel represents the entities between which messages flow and thus between which an edge exists in the Leslie Graph. For example, all packets sharing the same source and destination address might be designated as belonging to a single channel. Alternatively, at a finer granularity we might additionally use application protocol to identify a channel. Channels are described as *input* or *output* channels based on whether they represent messages received at or transmitted by a host.

Activity pattern. We assign a value of either *active* or *inactive* to each channel in the network over some fixed time window. A set of such assignments to channels at a node is an activity pattern for that node, indicating whether or not a packet was observed on each channel during the observation time window.

Activity model. The activity model for a node is a function mapping the activity pattern of the input channels to a vector of probabilities for each output channel being active.

The idea is that by repeatedly observing whether an output channel is active for a given input activity pattern, we can learn the activity model on a host. To do this we are investigating a number of alternative mechanisms including Naive Bayes Classifiers [8] and Noisy-OR models [12]. Our results so far show promise, but have also highlighted some of the inherent trade-offs for this approach. Since activity patterns discard all packet timings and counts within the observation window, picking a suitable duration for the window is critical. Over a very long time window we will learn that all channels are related, whereas selecting a window size that is too small will cause correlations to be missed. We are tackling this problem by building activity models simultaneously for a range of window size and working on good ways to combine the resulting models.

Constellation uses activity models on hosts to approximate Leslie Graphs in a completely distributed manner. The correlation coefficients in the activity model encode the confidence level for a dependency between two nodes. When a host wishes to learn its Leslie Graph for a particular service, it queries its relevant peers to find strong next-hop correlations in their activity models for when only the input channel on which the query was sent is active. This query is then forwarded to those peers who repeat the process, and the resulting transitive correlations combine to give a Leslie Graph from the point of view of the local host. When the Leslie Graph is large this has the advantage that we can order the search by "most likely" path. Leslie Graphs are generated *on-demand* and give a snapshot of recent history at each member host.

One of the challenges when combining local activity models to form a Leslie Graph is choosing an appropriate threshold for deciding that a correlation is strong enough to be part of the graph. At some correlation value for a given edge there is insufficient evidence to assume a causal relationship, and so the edge should be excluded. We are currently investigating this and several other issues, including statistical hypothesis tests for detecting both anomalous and normal changes to an activity model.

3.2 AND

The AND system consists of a centralized *inference engine* and a set of *agents*, one running on each desktop and server. Each agent performs temporal correlation of the packets sent and received by its host and makes summarized information available to the engine. The inference engine serves as an aggregation and coordination point: assembling the Leslie Graph for applications by combining information from the agents; ordering agents to conduct active probing as needed to flesh out the Leslie Graph or to localize faults; and interfacing with the human network managers.

Computing the Leslie Graph. Using the terminology of Section 3.1, we define a *channel* as a 3 tuple of [RemoteIP, RemotePort, Protocol]. Each agent then continuously updates a matrix of the frequency with which two channels are active within a 100 ms window. [5]

To construct the Leslie Graph, the inference engine polls the agents for their matrices. Figure 1 illustrates how aggregating matrices from multiple agents over a long period of time can find dependencies that might be obscured by caching, since even infrequent messages to a server become measurable when summed over many hosts. For example, many hosts will have a matrix similar to $H3$'s that shows

[5] We have found values from 100 ms to 1 s produce the same dependency graphs on clients, but servers that are heavily loaded may cause dependences to be spread over a larger time window.

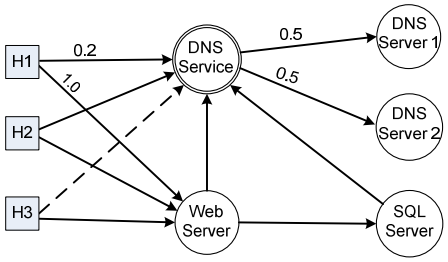

Figure 1: Part of a Leslie Graph discovered by AND when clients access some web server. The dashed line indicates a dependency found by aggregating information across hosts, $H1, H2, H3$.

a strong dependence on the web server, but no dependence on DNS as the web server's address has been cached. However, the matrices for $H1$ and $H2$ show that when these hosts communicated with the web server they also communicated with DNS in the same 100 ms window. If enough hosts that communicate on channel A (e.g., the web server) also communicate on channel B (e.g., DNS) within the same 100 ms, then the engine infers that any host depending on A most likely depends on B as well and will add to the Leslie Graph a dependency on B, as shown by the dashed line in the figure. Each edge in the Leslie Graph also has a weight, which is the probability with which it actually occurs in a transaction. In Figure 1, for example, $H1$ contacts the DNS server 20% of the time before it accesses the web server.

Networks that include either fail-over or load-balancing clusters of servers (e.g., primary/secondary DNS servers, web server clusters) are modeled by introducing a meta node into the Leslie Graph to represent each cluster, for example, the DNS Service node in Figure 1. Currently we use heuristics based on DNS names, port numbers, and stemming URLs to identify clusters and leave automatic detection of cluster configurations for future work.

In addition to user machines and application servers, AND extends the Leslie Graph by populating it with network elements, such as routers, switches and physical links. This broadens the applications of the Leslie Graph as, e.g., link congestion faults can now be localized. We can map the layer-2 topology by using the agents to send and listen for flooded MAC packets as in [5], and the layer-3 topology using traceroutes. Other techniques [6] could be used if SNMP data is available.

Using the Leslie Graph. Of the scenarios described in Section 2, our current focus is on efficient fault localization. Each agent observes the experiences of its own host (e.g., measuring the response time between requests and replies). When a user on the host flags the experience as bad, the agent sends a triggered experience report to the inference engine. For example, a negative experience report might be generated when a user restarts their browser or hits a button that means "I'm unhappy now", or when automated parsing identifies that something wrong has happened (e.g., too many "invalid page" HTTP return codes). A small number of randomly selected positive experiences (e.g., the time to load a web page when the user did not complain) are sent to the engine every 300 s.

The engine batches experience reports from multiple agents and applies Bayesian inference to find the most plausible explanation for the experience reports (i.e., the minimum set of faulty physical components that would afflict all the hosts, routers and links with poor performance while leaving unaffected the components experiencing acceptable performance). Space prevents a full description, but although Bayesian models typically require training, initial results (Section 4) show the structure of the Leslie Graph and the number of viewpoints provided by agents cause the results to have little sensitivity to the training process.

Scalability. The use of a centralized inference engine clearly makes it easier to aggregate information, but it raises scalability concerns about CPU and bandwidth limitations. On a single CPU, our system localizes faults on a Leslie Graph of 160 nodes (see Section 4 for details on the experiment setup) within 200 ms, with the time growing linearly in the number of nodes in the Leslie Graph.

Back-of-the-envelope calculations show the bandwidth requirements are feasible even for large enterprise networks. Experience reports are about 100 B and are sent to the inference engine every 300 s by each agent. The full co-occurrence matrix is polled from each agent every 3600 s. Most hosts in our network use fewer than 100 channels (i.e., use < 100 servers), so the matrix is less than 100x100 floats. Even for an extremely large enterprise network with O(100,000) computers and O(10,000) routers/switches, this results in an average bandwidth of only 10 Mbps. Busy servers have much more than 100 channels, but compression can be used if needed.

3.3 Discussion

As two separate projects, Constellation and AND are exploring two different points in the design space of approximating Leslie Graphs. While both approaches compute Leslie Graphs by aggregating the activities of multiple nodes, their differences highlight how the overall design space can be broken down into three axes, namely timing, structure, and granularity.

The first axis in the construction of a Leslie Graph is the time when it is constructed. Constellation constructs the Leslie Graph reactively, while AND proactively maintains it at the inference engine. If the Leslie Graph is constructed reactively, it imposes little overhead on hosts. However, since nodes log packets over a short period of time, a reactive scheme might miss out on dependencies affected by cached state. For example, in Figure 1, a reactive scheme might not determine the dependency between H3 and DNS. Furthermore, by proactively maintaining the Leslie Graph, the inference engine can respond to faults even before they are detected by all the users.

The second axis is the structure of the system, i.e. whether the Leslie Graph computation is centralized or distributed. Constellation uses a distributed approach to compute the Leslie Graph, while AND computes it at the centralized inference engine. A distributed (unstructured) approach is more robust to network and machine failures that might affect connectivity to the centralized server, while a centralized approach is simpler and easier to manage. We are improving the fault tolerance of AND by implementing the inference engine as a distributed cluster of machines.

The third axis is the granularity of the Leslie Graph, as discussed in Section 1.2. The nodes in the Leslie Graph could be a cluster of servers, a particular machine or a process on a machine. Similarly, for network elements, a node could be end-to-end connectivity between machines, or all the routers and switches in the path. The analysis on a more granular Leslie Graph will be more precise, although it might add complexity to the algorithm and unnecessary detail to the results. Constellation represents hosts and processes in the Leslie Graph, while AND also includes the routers and switches.

A notable trend is the increasing popularity of peer-to-peer applications. These are designed to achieve reliability by dynamically changing the set of servers with which a client communicates based on the content being exchanged or congestion levels in the system. As a result, the Leslie Graph is not stable across long time-periods. A system like Constellation, with its on-demand creation of the Leslie Graph using only recent observations, will report the set of peers currently in use. AND, which aggregates information across time, may show a dependency on servers no longer in use.

Our schemes do have limitations. We do not expect our techniques to find servers that return incorrect answers unless these errors lead to performance or fail-stop problems. For example, if a DNS server holds the wrong IP address for a name and only one client looks up the name, our approach will help only if the Leslie Graph changes as a result. Even if many clients lookup the wrong IP address and thus are unable to establish a connection, our fault localization algorithm will only point to the clients — although human examination of the Leslie Graph would reveal the affected clients share a DNS server. Here our tools and the structure of the Leslie Graph may help human investigators, even if they cannot automatically find the root cause.

4 Initial Results

Existence of correlations. The first step in validating our approach is determining if there is detectable correlation between input channels and output channels that are known to be related, and no correlation between unrelated channels. If this were not true, then black-box techniques would be infeasible. Figure 2 shows the results of plotting the time

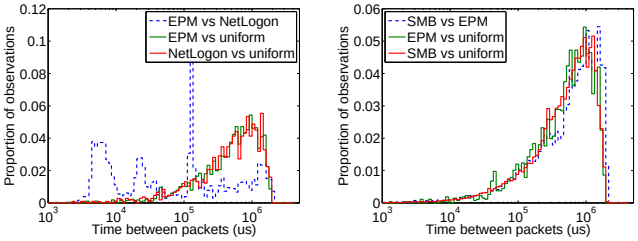

Figure 2: Evaluation of correlation between EPM packets & NetLogon packets (left) and EPM packets & SMB packets (right), using correlation with random noise as a control. EPM vs. NetLogon is significantly different from the control correctly validating their correlation while EPM vs. SMB is indistinguishable from the control.

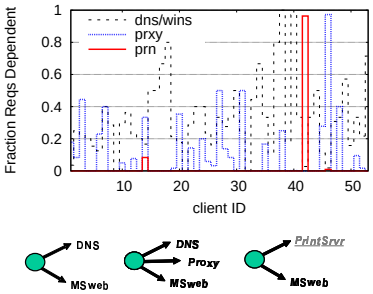

Figure 3: Example of finding dependencies of 53 hosts that contact an internal webserver. The figure on top shows the probabilistic dependencies discovered at each client. The figure below graphically represents the typical dependencies and also illustrates a false-dependency.

difference between receiving a packet of one protocol and sending a packet of the other protocol for three protocols: EPM (the RPC portmapper), NetLogon and SMB. Traces were collected for 1 hour from a busy server. The time differences are also computed against a packet stream whose timestamps are drawn from a uniform random distribution, representing background noise.

The right figure shows that SMB and EPM correlate as strongly with the random packet stream as they do with each other, implying they are not correlated. This is correct, as SMB and EPM are unrelated protocols. The left figure shows EPM and NetLogon have a very different distribution than the comparison with the random stream, implying EPM and NetLogon are closely related — in fact, NetLogon clients use EPM to locate the port to which they send their requests. We obtained similar validation for many other protocols, implying that packet-correlation techniques are feasible.

Finding dependencies. As a first test of AND's technique for finding dependencies in presence of caching, we ran the Leslie Graph generation algorithm against data from 53 hosts collected over one hour. Figure 3 shows the fraction of requests made by each client to the DNS, proxy, and PrintServer that co-occur with a request to a common webserver, called MSweb. As we can see, most clients invoke DNS when making web requests, although not 100% of the time due to caching. However, we still extract the

correct dependency. The data also show some clients are dependent on the proxy that is normally used for external access, even when accessing the internal web server. Investigation showed that these clients were misconfigured with an out-of-date list of internal names, indicating how our approach can be useful for detecting some classes of policy/configuration faults. Two misbehaving hosts made so many requests to the PrintServer that their dependencies for MSWeb became abnormal, showing that even false positives can yield valuable management information.

Usefulness of the Leslie Graph. To evaluate the ability of AND to find and use the Leslie Graph for fault localization, we have created a testbed with 23 clients that are evenly divided between two subnets connected by a router. Each subnet has a web server running Sharepoint (a wiki-like application), with data for the web sites stored on a single SQL database server on one of the subnets. Network services (DHCP, authentication servers, DNS) are connected to the subnet without the SQL server. Using packet-droppers, rate-shapers, and load generators we can deterministically create scenarios where any desired subset of the clients, servers, router, and links appears as failed or overloaded.

We evaluated five scenarios where combinations of one or more web servers, SQL server, routers, and links were set to an overloaded or failed state while robots on the clients made accesses to the web servers and each agent observed the response times seen by its client. These scenarios have Leslie Graphs with about 160 nodes, each of which is a component that could potentially fail. A small portion of the Leslie Graph for the testbed is shown in Figure 1. In all five scenarios, our fault localization algorithms run over the Leslie Graph correctly determined the problematic component. In three scenarios, the algorithm reported one more potentially problematic candidate than the number actually afflicted, but the algorithm also proposed the correct set of active probing tests to resolve this ambiguity.

5 Related Work

Project5 [2] proposes finding performance bottlenecks in a distributed system by using black-box approaches to track requests as they move between servers in the system. WAP5 [13] extends Project5 by developing a new message correlation algorithm for determining which arriving packets trigger which outgoing packets on a host. In contrast, this paper identifies the importance and challenges of discovering the Leslie Graph to support a broad range of management functions. By computing Leslie Graphs at different granularities, our techniques can uncover dependencies which might be overlooked by WAP5 and Project5, such as those masked by caching. We also present several new scenarios where the Leslie Graph can be applied. The notion of "Communities of Interest" (COIs) in enterprise networks is studied by Aiello *et al* [3]. A COI is defined more narrowly than the Leslie Graph, as a collection of interacting hosts, and the authors do not explicitly consider the problem of finding network dependencies.

6 Conclusion

As the web of dependencies between hosts, applications and network elements increases in size and complexity, building tools to automatically discover and reason about these dependencies will be invaluable for network operators and normal users. In this paper, we introduce the Leslie Graph as a generic representation of this web of dependencies. We present two complementary approaches to computing Leslie Graphs, and highlight the differences between them in three dimensions in the design space. We also present several applications of Leslie Graph and the challenges in discovering its approximation that have not been addressed in prior work. We are now gaining experience using the Leslie Graph, and so far have had success finding anomalous configurations and localizing performance faults, which tend to be transient, hard to debug, and annoying to users. We believe that the general problem of discovering and using Leslie Graphs presents a rich field of future research.

References

[1] A.Brown, G.Kar, and A.Keller. An active approach to characterizing dynamic dependencies for problem determination in a distributed environment. In *IFIP/IEEE IM*, May 2001.

[2] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed systems of black boxes. In *SOSP'03*, pages 74–89, Oct. 2003.

[3] W. Aiello, C. Kalmanek, P. McDaniel, S. Sen, O. Spatscheck, and J. V. der Merwe. Analysis of communities of interest in data networks. In *PAM'05*, Mar. 2005.

[4] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using Magpie for request extraction and workload modelling. In *OSDI'04*, Dec. 2004.

[5] R. Black, A. Donnelly, and C. Fournet. Ethernet topology discovery without network assistance. In *ICNP'04*, Oct. 2004.

[6] Y. Breitbart, M. Garofalakis, B. Jai, C. Martin, R. Rastogi, and A. Silberschatz. Topology discovery in heterogeneous IP networks: The NetInventory system. *IEEE/ACM ToN*, 12(3), June 2004.

[7] M. Y. Chen, A. Accardi, E. Kıcıman, J. Lloyd, D. Patterson, A. Fox, and E. Brewer. Path-based failure and evolution management. In *NSDI'04*, pages 309–322, Mar. 2004.

[8] R. Duda, P. Hart, and D.G.Stork. *Pattern Classification*. Wiley-Interscience, 2nd edition, Oct. 2000.

[9] L. Lamport. Quarterly quote. *ACM SIGACT News*, 34, Mar. 2003.

[10] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why do Internet services fail, and what can be done about it? In *USENIX SITS*, 2003.

[11] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney. A first look at modern enterprise traffic. In *IMC'05*, pages 15–28, Oct. 2005.

[12] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, Sept. 1988.

[13] P. Reynolds, J. L. Wiener, J. C. Mogul, M. K. Aguilera, and A. Vahdat. WAP5: Black-box performance debugging for wide-area systems. In *WWW'06*, May 2006.