ATMEN: A Triggered Network Measurement Infrastructure

Balachander Krishnamurthy
AT&T Labs–Research
bala@research.att.com

Harsha V. Madhyastha*
University of Washington
harsha@cs.washington.edu

Oliver Spatscheck
AT&T Labs–Research
spatsch@research.att.com

ABSTRACT

Web performance measurements and availability tests have been carried out using a variety of infrastructures over the last several years. Disruptions in the Internet can lead to Web sites being unavailable or increase user-perceived latency. The unavailability could be due to DNS, failures in segments of the physical network cutting off thousands of users, or attacks. Prompt reactions to network-wide events can be facilitated by local or remote measurement and monitoring. Better yet, a distributed set of intercommunicating measurement and monitoring entities that react to events dynamically could go a long way to handle disruptions.

We have designed and built ATMEN, a triggered measurement infrastructure to communicate and coordinate across various administrative entities. ATMEN nodes can trigger new measurements, query ongoing passive measurements or stored historical measurements on remote nodes, and coordinate the responses to make local decisions. ATMEN reduces wasted measurements by judiciously reusing measurements along three axes: spatial, temporal, and application.

We describe the use of ATMEN for key Web applications such as performance based ranking of popular Web sites and availability of DNS servers on which most Web transactions are dependent. The evaluation of ATMEN is done using multiple network monitoring entities called Gigascopes installed across the USA, measurement data of a popular network application involving millions of users distributed across the Internet, and scores of clients to aid in gathering measurement information upon demand. Our results show that such a system can be built in a scalable fashion.

1. INTRODUCTION

Numerous Web performance measurement infrastructures have been built over the years. Entities such as Keynote [12] routinely measure availability and performance of popular Web servers. Content Distribution companies such as Akamai do their internal measurements to balance load on their large set of servers. Measuring and monitoring infrastructures perform the role of watching numerous servers typically by periodically polling them.

Beyond failures at the individual server sites, disruptions in the Internet can lead to sites being unavailable or increase user-perceived latency. Disruptions could be at any of the protocol layers directly involved in the Web transaction such as DNS or TCP. But network failure events can cause a large collection of clients to be cut off from accessing the site even though the Web site itself is functioning flawlessly. Constant measurement and monitoring of collections of Web servers is thus an important task to ensure performance and availability of Web sites.

Current monitoring techniques tend to use distributed infrastructures that coordinate internally in a proprietary fashion. For example, ISPs measure and log information locally and correlate on a intra-net basis. However rarely is there correlation of such information with other monitoring sites. Correlation of measurement and monitoring information gathered elsewhere can aid in handling network events such as worms and flash crowds. Geographically distributed machines [20] are available as part of a testbed for such purposes. Hardware-based packet monitors [4] and associated software can process and analyze high volumes of data. Traditional monitoring devices inside a small administrative entity report all events to a centralized repository. The volume of traffic however prevents this approach from scaling to larger administrative entities or for even selectively sharing across the Internet between multiple administrative entities. Thus there is a need for a *distributed set of intercommunicating measurement entities* that can dynamically react to events and gather application-specific measurement data.

In this paper we report on the design and development of ATMEN—a triggered measurement infrastructure for communicating and coordinating across various administrative entities. ATMEN nodes are capable of triggering new measurements on remote nodes and querying ongoing passive and existing historical measurements. Measurements can be selectively turned on and off for specific durations of time on a subset of co-operating set of measurement sites based on the occurrence of one or more events. Selective triggers tailored to local needs and written to exploit different capabilities of multiple data sites is a more practical, low cost solution. Multiple queries can be run simultaneously in the presence of live traffic. The results of the queries are coordinated before making local decisions in conjunction with local information. Monitoring Web sites for availability and performance are just two exemplar applications of ATMEN.

A key goal of ATMEN is to avoid wasted measurements by judiciously reusing measurements. The same parameter is often measured by multiple applications, and most measured parameters exhibit varying degrees of stationarity across time and space. Earlier proposals include techniques for reducing measurements in the form of a single measure-

*Work done while at AT&T

WWW2005, May 10–14, 2005, Chiba, Japan.
.

ment entity or addition of more primitives into the network to reduce the measurements that overlay services have to make [17],[10]. However, such limited forms of measurement reuse do not envision temporal and spatial stationarity or reuse across applications. Commonalities across clusters of measurement sites may allow the measurements to be reused across space. Many applications share certain common primitive measurement components and these may be reusable by other applications. For example, many applications on the Internet interested in performance may measure a DNS component which may be reusable across applications if there is not considerable variance in the timescale of interest. Reuse in ATMEN is thus along three axes: spatial, temporal, and application.

We began with a preliminary measurement study to examine spatial and temporal stationarity of the constituent elements of measurement, and the ability to partially reuse application-level measurements. The results convinced us of the potential viability of our approach. This paper describes the design, implementation, and evaluation of ATMEN.

The main contributions of our work are:

1. The design and construction of a triggered measurement infrastructure involving hardware and software components, historical collections of past measurements, and a protocol for communication between the nodes. The basis for carrying out quick and low-cost monitoring to aid a variety of application-level tasks are provided by the infrastructure.
2. Identifying the axes of possible reuse of network measurements to reduce wasted measurements.
3. A demonstration of the feasibility of a system reusing measurements across time, location, and application.

The demonstration, a key focus of this paper, is via two distinct experiments. The first experiment monitors download times of popular Websites from numerous PlanetLab client sites. Simultaneously, network logs of a popular interactive application are polled to look for problem areas. Using hints of network delays in certain segments of the Internet from the interactive application, additional download measurements are triggered in the Web site monitoring application. The second experiment, monitors DNS server availability through multiple ATMEN nodes passively measuring DNS traffic. When the number of DNS responses is less than the number of requests at any of the nodes for any DNS name, a set of PlanetLab nodes are used to check the status of the DNS server for that DNS name. The primary aim is to demonstrate the low overhead of ATMEN.

2. RELATED WORK

ATMEN is the first comprehensive application-independent architecture for triggered measurements and measurement reuse. Prior work focused on a particular application domain, was limited to one type of measurement, or lacked a model to compose diverse measurements. Both [17] and [10] focus on reducing the amount of measurements overlay networks have to make. They do not consider the spatial and temporal stationarity of measurements and the reuse of measurements for anything other then overlay applications. Other application domains that have seen increased attention lately are distributed Intrusion Detection [1], large scale worm detection[3], Internet Storm Center [9] for widespread attack detection, and distributed attack suppression [5]. The proposed solutions are application-specific and do not consider the spatial and temporal stationarity of measurements. Similarly services such as Keynote [12] and its affiliated services (Internet health report [8] and Netmechanic [19]) conduct various performance measurements using a distributed infrastructure but do not provide a system which would allow selective reuse or triggering of measurements by third parties. There also exist standalone systems, such as eValid [6], that provide Website capacity performance analysis.

EtE monitor [2] measures Web site performance by passive accumulation of packet traces from a Web server, reconstructing the accesses, and presenting both network and server overhead by a combination of statistical filtering methods as heuristics. The monitor can be placed at a Web server or at any point in the network where it can examine traffic associated with a Web server. The authors do not discuss the possibility of monitors communicating with each other.

A system for distributed measurements [22] uses the peer-to-peer paradigm in the measurement context to measure properties of a network path between two participants. The system does not address other measurement types such as HTTP download time and does not present a model guiding the reuse of the gathered data. A similar end-system based approach is proposed by [11]. This system passively gathers various statistics from the end host's local network and reports them to a central server. This system is also limited in the type of data it can gather and in addition we do not believe that such a fully centralized approach will scale.

SPAND [21] is similar to ATMEN in that it reuses passive measurements over space and time to improve replica selection. SPAND shows the benefits of sharing network measurements across time but does not share the general framework provided by ATMEN. Our survey of related work indicates that there are many non-interacting application specific solutions for distributed measurements. Therefore, we feel the time is ripe to combine these application specific solutions within a common framework.

3. DESIGN OF ATMEN

Before implementing ATMEN, we identified the basic architecture of an application-independent triggered measurement infrastructure. There are essentially two kinds of entities in ATMEN - applications (*App*) that require measurement data for some particular parameter and sources (*DS*) which provide measurement data for some parameter, where we employ the generic term *parameter* to refer to any individual type of measurement data. Since ATMEN would span across administrative entities, it is not feasible to enforce every participating entity to construct Apps and DSes using some standard code base. A practical solution is to instead define the language, *i.e.*, the protocol for communication between different entities in the system. All entities would only need to ensure that their Apps and DSes can speak and understand this protocol. To develop a protocol suitable for ATMEN, we consider the flow of control in any application that makes use of triggered measurements.

```
repeat forever or upon a client request {
    Request data for a set of parameters P from DSes S
    if (event e is detected) {
        Request data for a set of parameters P' from DSes S'
    }
}
```

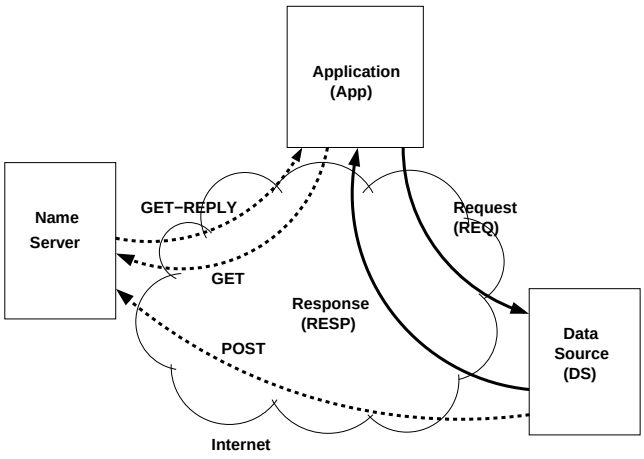


Figure 1: Basic architecture of ATMEN

ATMEN needs to provide the following three basic primitives to support such an application.

1. Any DS should be able to advertise its existence and its measurement capabilities, *i.e.*, the parameters it can provide measurement data for and the language it supports for querying on these measurements.
2. Any App should be able to discover the existence of the DSes which provide measurement data for its parameter of interest.
3. An App should be able to send a request for data to the relevant DS and the DS should be able to send back a response.

Keeping these primitives in mind, we introduce a name server into the system to which DSes would send their advertisements and from which Apps would learn the existence of DSes relevant to them. To support the three primitives, we propose the use of five types of messages in ATMEN's communication protocol – *POST*, *GET*, *GET-REPLY*, *REQ* and *RESP*. The site hosting a DS advertises its existence by sending a *POST* message to the name server. An App can learn about DSes providing measurement data of interest by sending an appropriate *GET* message to the name server and receiving a *GET-REPLY* message in response. The App then sends a *REQ* message to the site hosting the appropriate DS, requesting measurement data. The DS sends data back to the site hosting the App, in a *RESP* message. The architecture of ATMEN is shown in Figure 1.

A key objective in the design of ATMEN is to make it application and measurement data independent. Hence, we chose XML as the language in which the messages are specified. We also require that all communication be over TCP connections. We now formalize each of the five messages:

POST: The POST message facilitates a DS to advertise itself to the rest of the system. To enable Apps to request data from this DS, the POST message must contain the parameter *param* for which the DS supplies measurement data and the address *ip_address:port* to which requests for this data should be sent. One issue that this throws up is how does an App comprehend the semantics of the name *param* for the concerned DS. We believe that all measurements can be expressed in terms of a globally understood namespace of commonly used parameters. To resolve this issue, we also include a string *descr* in the POST message which gives a precise description of how the DS measured the parameter *param*. For example, the description for *DNS_lookup_time* measured by some DS could be the time taken by a call to gethostbyname() within a program invoking it.

In most cases, just requesting data for parameter *param* does not make much sense. For example, in the case of the parameter DNS lookup time, no value can be determined for this parameter unless the address prefix the measurement was made from, the DNS name for which the measurement was made and the time at which the measurement was made are specified. Hence, the POST message also includes the arguments that characterize any particular instance of the measurement. In the case of DNS lookup time, these arguments would be *src_prefix*, *DNS_name* and *timestamp*.

Apart from providing raw measurement values, a DS may support some language for querying on the measurement data. We could either enforce all DSes to support the same language for querying or include the format of the query language for the particular DS in the POST message. Both of these are too restrictive, and so we chose an approach in between these two extremes. We believe that the query language supported by most DSes would be decomposable into a standard set of capabilities, performing the basic functions of selection, transformation and aggregation. To advertise these capabilities, the POST message would include a capability vector for the capabilities supported by the DS.

GET: The GET message enables Apps to learn the existence of DSes providing measurement data for their parameter of interest from the name server. GET should include the name of the parameter for which a list of DSes is required.

GET-REPLY In response to a GET message with parameter *param*, the name server returns a GET-REPLY message that contains all the entries of the POST messages of all DSes providing data for the same parameter *param*.

REQ The REQ message is sent by an App to a particular DS to request data from that DS. This message is sent over a TCP connection setup with the *address:port* specified in the POST message for that DS. The REQ message specifies the name of the parameter *param* for which measurement data is solicited and a set of values for each of the arguments specified in the POST message for this DS. The REQ message should also specify the capabilities that need to be enforced along with appropriate values as arguments for these capabilities.

RESP The DS sends back its response in a RESP message with a list of tuples. Each tuple comprises of a value for the measurement parameter that data was requested for along with the values taken by the arguments for the particular instance of the measurement that yielded this value. Since ATMEN is a cooperative best-effort system, the DS might choose not to return any data. The RESP message would then include an error code with the reason for not returning data. Such reasons include invalid values for the arguments, the requested data being unavailable, the server being overloaded, or refusal to provide data to a particular IP address based on some administrative policy.

Figure 2 shows the exact format of the POST, REQ and RESP messages in the context of a particular App-DS interaction. The DS measures DNS lookup times from the prefix *123.456.0.0/16* for a set of DNS names *www.foo.com*, ..., *www.bar.com*, as specified by the *src_prefix* and *dns_name* arguments. The *timestamp* argument does not have any

```
<POST>
<SITE>
<ADDRESS>123.456.101.202</ADDRESS>
<PORT>12345</PORT>
</SITE>
<PARAM> DNS_lookup_time
<ARG value="src_prefix">123.456.0.0/16</ARG>
<ARG value="dns_name">www.foo.com, ....., www.bar.com</ARG>
<ARG value="timestamp"></ARG>
</PARAM>
<DESCR>Time taken by call to gethostbyname in httperf</DESCR>
<CAP value="threshold"></CAP>
<CAP value="count"></CAP>
</POST>
```

(a)

```
<REQ>
<PARAM>DNS_lookup_time
<ARG value="src_prefix">123.456.0.0/16</ARG>
<ARG value="dns_name">www.bar.com</ARG>
<ARG value="timestamp">[1089210957, 1091038673]</ARG>
</PARAM>
<CAP value="threshold">3</CAP>
</REQ>
```

(b)

```
<RESP>
<TUPLE>4.037
<ARG value="src_prefix">123.456.0.0/16</ARG>
<ARG value="dns_name">www.bar.com</ARG>
<ARG value="timestamp">1090078345</ARG>
</TUPLE>
</RESP>
```

(c)

Figure 2: (a) POST (b) REQ (c) RESP

value indicating that it accepts requests for historical measurements made any time in the past. The DS uses the httperf [16] application for measuring DNS lookup times and does so by measuring the time taken by the gethostbyname() call within httperf. This DS supports two capabilities. The *threshold* capability implies that given a threshold value, the DS can supply values for all measurement instances with the required values for the arguments such that the value of the parameter is greater than the threshold. The *count* capability implies that given a set of values that the arguments can take, instead of providing the values of the measurement parameter, the DS can provide a count of the number of measurement instances with these values for the arguments. The App makes a request for instances of DNS lookup time measurements, made for one of the DNS names within some specific interval of time, that were greater than 3 seconds.

4. IMPLEMENTATION

We implemented a realization of the ATMEN architecture described above. We could choose any possible implementation for the name server, the Apps and the DSes as long as they communicate in conformance with the basic protocol described in the previous section. The two primary objectives that we had in mind while deciding on one particular implementation are extensibility and minimizing communication overhead. The system should permit new Apps to make use of the infrastructure with ease and adding new DSes should be easy for any site. The second objective of reducing communication, useful in any system, is all the more significant in a triggered measurement setup as it is critical to be able to react to events quickly.

The components in our implementation of ATMEN are shown in Figure 3. To simplify addition of new Apps as well

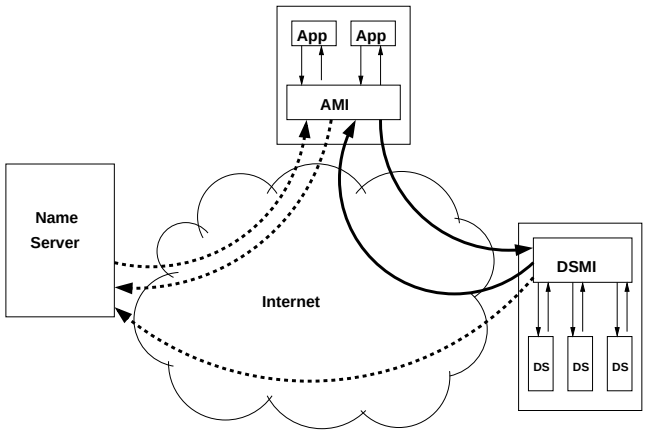


Figure 3: Our implementation of ATMEN

as new DSes, we have interfaces on nodes which host Apps as well as on those which host DSes. We refer to these as the Application Measurement Interface (AMI) and the Data Source Measurement Interface (DSMI), respectively. When an App needs to retrieve some measurement data, it issues the request to the local AMI which sends a REQ message to the DSMI on the appropriate node. The DSMI retrieves the data from the appropriate DS, and sends back a RESP message to the AMI, which forwards it to the App. We next present details of our implementation of the name server, the AMI and the DSMI.

The name server runs a TCP server at a globally known address:port. DSes advertise themselves by setting up a TCP connection with the name server and sending a POST message on the connection. Similarly, Apps set up a TCP connection, send a GET message and receive a GET-REPLY message. A script to be executed by the administrator of the site hosting the DS, is used to send out an advertisement for any DS. Name server resiliency is vital as it cannot be a single point of failure of the whole system. It would have to be implemented in a hierarchical manner, much in the same way as the DNS root servers are currently implemented.

The AMI is realized as a C library to minimize system overhead; Apps can link with it to access the exported API. The interface exported by the AMI essentially has two calls – *getDSes* and *getMeasurementValue*. The App can first learn about the existence of DSes it is interested in by making a *getDSes* call on the AMI. Based on the parameter *param* passed to it by the App, the AMI constructs an appropriate GET message and sends it to the name server. The AMI then parses the GET-REPLY message returned by the name server. The AMI returns to the App an array of an abstract datatype, called DStype, which is defined in the header file of the AMI that Apps would include. DStype captures all the properties a DS could specify in its POST message.

The App then makes a *getMeasurementValue* call on the AMI to request measurement data. Based on the DStype object passed as argument, the AMI constructs a REQ message and sends it to the appropriate DS. To reduce the connection setup overhead, the AMI maintains persistent TCP connections, timing them out if unused for a long period of time. To enable the App to make multiple requests in parallel without having to use threads, the *getMeasurementValue* call on the AMI returns a file descriptor to the App as soon as the

REQ message is sent out successfully. A separate thread within the AMI polls all the connections on which responses are expected. When a RESP message is received, the AMI identifies the file descriptor the tuples received should be written out to. Since multiple requests and responses are sent and received over the same TCP connection, the AMI implements appropriate locking to ensure that there is at most one outstanding request to a given DS at a given time.

On the site hosting the DS, the DSMI runs a TCP server at the port advertised in the POST message. When it receives a REQ message, the DSMI parses the message and launches the appropriate command to obtain the required measurement data. The DSMI then reads in the output of the command, constructs the corresponding RESP message and sends it back on the connection it received the REQ message. In comparison with the latency suffered by the REQ and RESP messages in traversing the Internet and the time taken to execute the process that provides the measurement data, the overhead due to the AMI and DSMI is extremely low, as we show in our results later.

5. REUSE OF MEASUREMENTS

Sharing of measurements in the context of a triggered network measurement infrastructure enables reuse across time, space, and application. This is based on the observation that many measurement parameters display some degree of temporal and spatial stationarity and many of the components measured are the same across applications. We present a rough outline of how this can be achieved to reduce the number of measurements that need to be made.

Suppose an application needs the value for measurement of some parameter Z from a site s^* at time t^*. Z can be a function $f(X_1, \ldots, X_n)$ of some n component measurements $X_1, \ldots, X_n$. Several historical measurements might exist for each of these components, X_i, as part of measurements of the same/different parameter by the same/different application. To reuse measurements, we need to characterize the variability of each component with respect to space and time with probability distributions P_i^t and P_i^s, respectively. $P_i^t(t^*|t)$ is the probability of the measurement made at time t being still valid at time t^*, while $P_i^s(s^*|s)$ is the probability of the measurement made at site s being valid at site s^*. These probability distributions have to be determined *a priori* based on measurement studies of each component.

Based on these probability distributions, we can compute the confidence value associated with each historical measurement and choose the value with which is associated the highest value of confidence. Assuming that historical measurements exist for all the component measurements $(X_1, X_2, \ldots, X_n)$, a value of Z can be inferred and the confidence interval associated with this value can be computed. If the range of this interval is below a chosen threshold fraction of the inferred value of Z, then additional measurements are unnecessary; the inferred value for Z can be used. This threshold would be chosen based on the quality of measurement desired by the application or by the intrinsic variability associated with Z (the variation observed in measurements made at the same time at the same site). On the other hand, if the *"quality"* of the inferred value is deemed to be low, new measurements have to be triggered for a subset of the components $(X_1, \ldots, X_n)$. This subset would be determined based on the costs associated with measuring each of these components and the increase in "quality" desired in the value for parameter Z.

Such an approach towards measurement reuse would work for applications where the parameter of interest can be expressed as a function of measurable components, the variability of each of which can be characterized with respect to the time and location where the measurement was made.

Though the above approach completely captures reuse of measurements across space and time within the same application, it can be applied for reuse across applications only when measurements can be directly reused. Full reuse of measurements across applications is applicable only when a component measured by an application A_1 is also a component of the parameter of interest of another application A_2. On the other hand, partial reuse of measurements across applications is possible where A_1 infers the need to trigger new measurements based on measurements made by A_2. For such reuse A_1 would require *a priori* knowledge of the impact changes in the measurements made by A_2 have on the components it measures. Thus, when A_1 measures a component X_i, it would have to store the current estimates of measurements by A_2 that influence X_i. When A_1 reuses this measurement of component X_i based on temporal and spatial stationarity, it has access to the new estimates for measurements made by A_2 that affect X_i. Based on the degree of change compared to the stored estimates, A_1 can infer whether new measurements need to be triggered.

6. EXPERIMENTS

We deployed and evaluated two measurement applications that can take advantage of triggered measurements. Our objective was not only to highlight the applicability of a triggered measurement infrastructure such as ATMEN, but also to choose applications which are in some way representative of all applications that make use of Internet measurements. For this, we classified all measurement applications based on their parameter of interest and, implemented and deployed one application each from two of these categories.

Monitoring DNS server availability: This is an example of applications that are interested in the success/failure of a measurement. We use passive network monitoring to detect the failure of a DNS lookup for a particular DNS name. If we detect such a failure we trigger active measurements from a set of geographically distributed sites to check if the DNS server for that particular DNS name is truly unreachable. This experiment allows us to monitor a large number of DNS names for availability with a minimal amount of active measurements.

Performance-based ranking: This is an example of applications that are interested in the value/result of a measurement. We monitor the performance of a set of Web sites by ranking them in the order of the download times as measured from a distributed set of sites. As download time can be broken down into DNS, TCP and HTTP components, there is significant potential for measurement reuse. Temporal and spatial stationarity of these components can be taken advantage of to reduce the number of measurements required. To ensure that disruptions in the network are not missed, measurements made in other applications can be used to trigger new measurements.

Applications in the other two categories where the parameter of interest is either some aggregate/summary statistic (such as total traffic per port or average link utilization), or the output/content of a measurement (such as IP address

returned by DNS lookup or Web page returned by a HTTP transfer), can also benefit from ATMEN.

6.1 Study of Measurement Reuse Potential

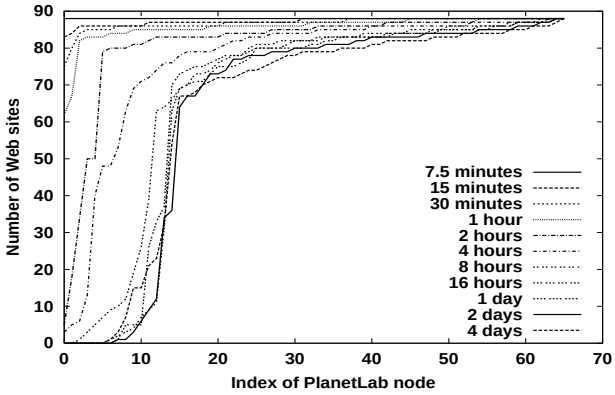
We first studied the potential for measurement reuse in the performance-based ranking application. Download time is decomposed into sum of individual components [13]. We obtained *"index.html"* from Web sites and focused our attention on the main components of download time – DNS lookup time, TCP connection setup time, and HTTP transfer time. We first examine measurement reuse across the temporal and spatial axes and later show how measurements made by another application can trigger new measurements showing application-level partial reuse of measurements.

The study involved periodic download of the home page of a set of popular Web sites from several PlanetLab nodes chosen as client sites, geographically distributed around the globe. We merged several popular Web site rankings—Netcraft [18], Mediametrix [15], and Fortune 500 [7] to obtain the 88 most popular Web sites. From each client site we downloaded the home page of these sites, roughly every 7.5 minutes for 17 days between July 3–19 2004. For downloading, we used *httperf* [16] and logged the DNS lookup time, TCP connection setup time, and HTTP transfer time. We divided the data into two parts—July 3rd to 11th and July 12th to 19th. We analyzed the data in each part in isolation and the inferences that we drew based on the data of the first part were confirmed on the data of the second part.

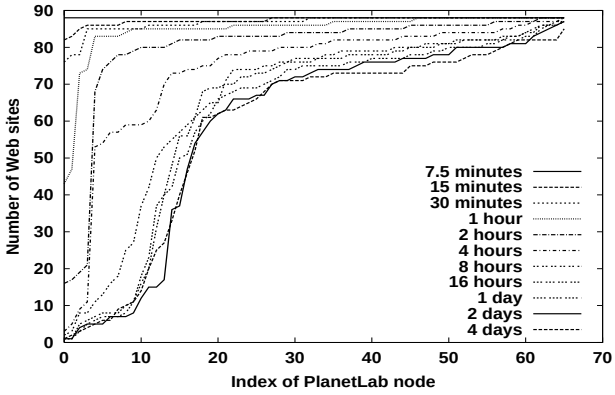
6.1.1 Reusability across time

We analyzed the data collected to study the temporal and spatial stationarity of each of the components of download time. For temporal stationarity we looked at the variability of each component over various timescales. As a benchmark for variability, we considered the intrinsic variability of each component, *i.e.*, by how much do two successive measurements of the same component vary. We computed the ratio of every pair of successive measurements and found the value below which more than 80% of these ratios lie. We determined this value for each component for every (client, Web site) pair and refer to this as the intrinsic variability threshold for that component of the download time. We then computed the ratios of every pair of measurements 15 minutes, 30 minutes, 1 hour, 2 hours, 6 hours, 12 hours, 1 day, 2 days and 4 days apart and computed the variability threshold over each of these timescales. Figure 4(a) shows, for each of these timescales, the number of Web sites on each PlanetLab node for which the variability threshold of TCP connection setup time at that timescale was less than the intrinsic variability threshold. On most PlanetLab nodes, the variability of the TCP setup time even across days is the same as that observed in successive measurements. Thus, connection setup time is pretty stable over multiple days. However, near-perfect stationarity across all PlanetLab nodes is observed only across a couple of hours. Figure 4(b) shows that HTTP transfer time too exhibits similar stability.

DNS lookup time did not exhibit similar stability – displaying instead a bimodal distribution, with the two values representing DNS lookup performed with a cache hit and a cache miss. Figure 5 shows that on each PlanetLab node, the fraction of DNS times clustered around the 25th and 75th percentiles (within either 100% or 10ms) is pretty high for almost all Web sites, but the number of Web sites with



(a)



(b)

Figure 4: Temporal stationarity of (a) TCP setup time and (b) HTTP transfer time: High stability across 2 hours and reasonable stability across days.

more than 60% of the measurements clustered around just the 25th percentile is significantly lesser. These cache misses were due to low cache hit rates in the nameservers. This was because the nameservers used on the PlanetLab nodes were different from those used by other machines at the same site. Since DNS time exhibits a bimodal distribution, we can use a randomized algorithm to predict which of the two modal values the next DNS lookup would take.

6.1.2 Reusability across space

To explore spatial stationarity, we studied whether each of the TCP connection setup and HTTP transfer time components from a particular Web site on a PlanetLab node can be predicted given the value of that component for the same Web site on another PlanetLab node close to it[1]. As TCP connection setup and HTTP transfer time were observed to be highly stable over time, we considered the median value to be representative of the distribution. On each PlanetLab node, we computed the median value for both components of download time to each Web site. We then computed for both components, the correlation co-efficient across the 88 Web sites for each pair of PlanetLab nodes. Figure 6 shows that the correlation co-efficient is very close to 1 for almost all PlanetLab node pairs within 20ms of each other.

[1]We do not expect DNS time to exhibit spatial stationarity.

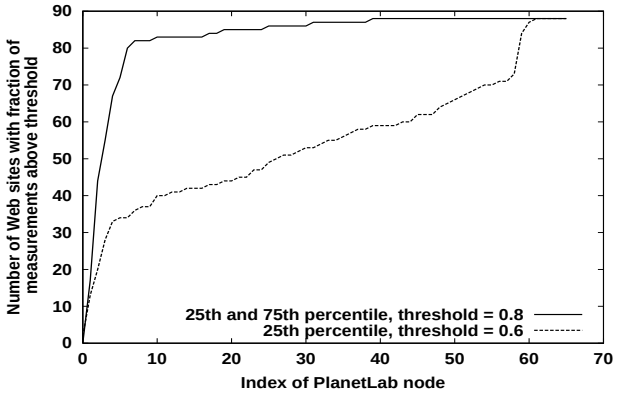


Figure 5: Bi-modal distribution of DNS lookup time

Our study showed that the TCP setup time and HTTP transfer time components of download time are highly stable over the period of a couple of hours, and that they are highly correlated for nearby sites. A similar study could be performed for the DNS availability application too. To examine temporal and spatial stationarity, DNS lookups to a set of Web sites need to be performed repeatedly over a period of time from a distributed set of sites. The results of these would then have to be analyzed to determine the probability that a DNS lookup fails at time t given that it failed at time t', and the probability that a DNS lookup fails at site s given that it failed at site s'. We have not yet carried out a measurement study of such a nature.

6.1.3 Reusability across applications

To demonstrate reusability of measurements across applications, we show how measurements made in an unrelated application can be partially reused to trigger measurements in the performance-based ranking application. We consider a multi-user interactive application where tens of thousands of users from across the world connect to a single server cluster. This application generates a log file every minute containing the IP addresses of the set of clients currently connected to the servers. Our objective was to explore if anomalies observed in this application can be used to trigger new measurements in the ranking application.

We compared every log file with that generated 3 minutes prior to it, and determined the instances where the number of users taking part in the application drops by more than 10%. Our hypothesis was that a sharp drop in the number of users is more likely to be due to a network or server outage rather than users voluntarily leaving the application. We identified 8 such instances during the period from April to July 2004, the period for which we had data. We determined the set of clients that were lost during each of these 3 minute periods and then made use of BGP data to determine the AS-level paths from each of these clients to the server. Based on this, we computed for every AS, the number of clients that were lost whose paths pass through it.

In one of the instances, a significant increase in the number of lost clients was associated with every AS (Figure 7(a)). Since it is highly improbable that a catastrophic event occurred in all ASes simultaneously, this indicates that the problem was most probably with the server cluster. This information is of no value for the performance-based ranking application. On the other hand, Figure 7(b) shows another instance where a significant increase in the number of lost clients is seen in only AS 1, clearly indicating that there has been some event in that particular AS. [2]

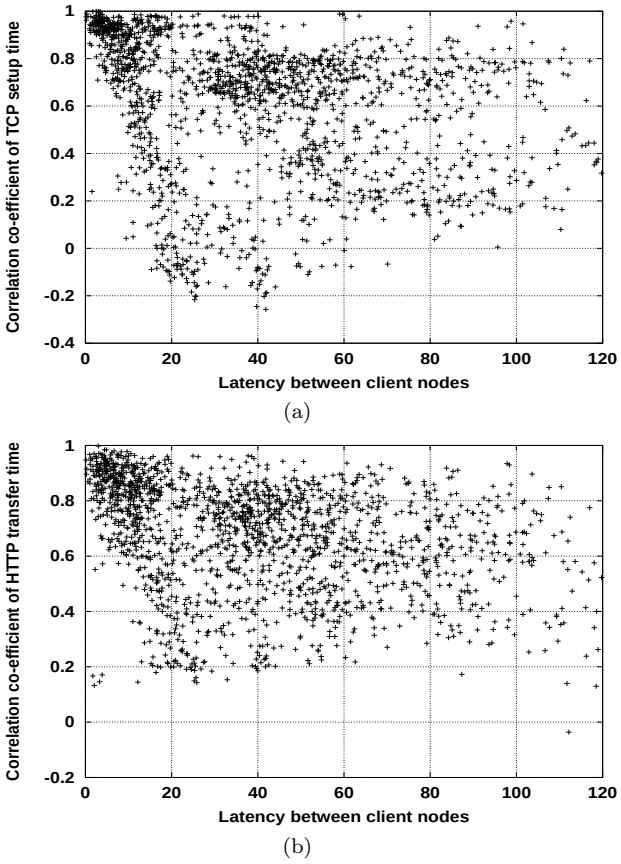
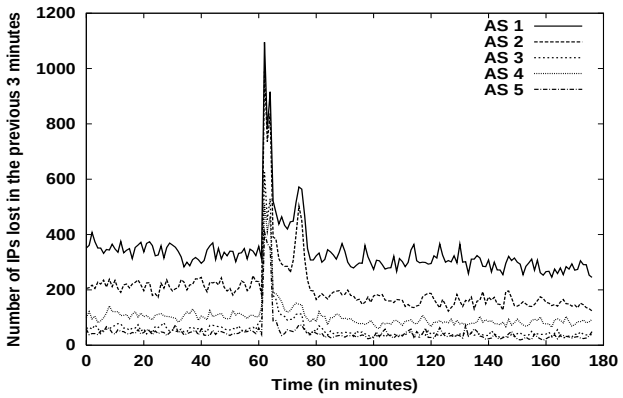


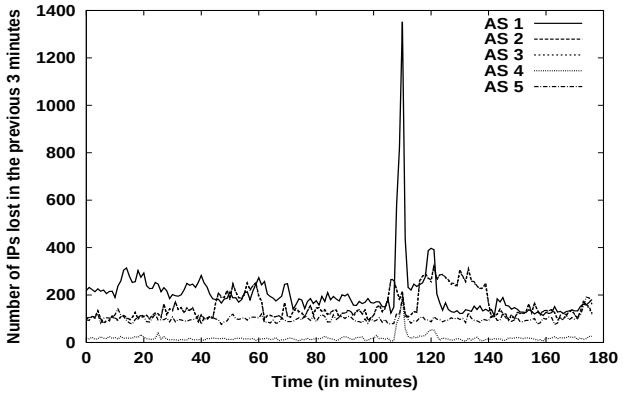
Figure 6: Spatial stationarity of (a) TCP connection setup time and (b) HTTP transfer time: Both show strong correlation when client nodes are within 20 ms of each other.

These observations indicate that the measurements made by the multi-user interactive application can be utilized to build an AS health index. Suppose such an index is available to the performance-based ranking application. To make use of this information, the ranking application needs to have *a priori* input that when the health of some AS is "low", the TCP connection setup time and HTTP transfer time to any Web site, the path to which passes through that AS, could possibly have changed. To be able to make use of this information, whenever the ranking application makes an estimate for the download time to a particular Web site, it also determines the AS-level path to the Web site. Suppose this estimate for download time is reused in the future based on temporal and spatial stationarity. If the health of one of the ASes along the path to the Web site has dropped significantly, then the application infers that the TCP setup time and HTTP transfer time components of the historical measurement are suspect and a new measurement might have

[2]AS numbers have been anonymized.



(a)



(b)

Figure 7: (a) A sharp increase in number of clients lost is seen in all ASes indicating a problem with the server (b) A sharp rise in number of clients lost is seen in a single AS indicating a problem in that AS

to be triggered. This is an instance of how partial reuse of measurements across applications is possible.

6.2 Details of Application deployment

6.2.1 Data Sources

We now briefly describe the measurement data sources that we utilized in the deployment of the two applications we chose - performance-based ranking of Web sites and monitoring of DNS server availability.

- The Gigascope[4] packet monitoring system, which is a highly configurable high speed network monitor, was deployed at 3 locations within the US – 2 of them at a site in the south-west and the third at a location in the north-east. The Gigascope in the north-eastern part, which we refer to as *gs1* henceforth, is monitoring a 45Mbps Internet access link connecting approximately 300 researchers to the Internet. The Gigascopes in south-western US, which we refer to as *gs2* and *gs3* henceforth, are monitoring a cable access network used by a few thousand cable access customers with a typical traffic volume of 200Mbps.

  On these nodes, Gigascope passively monitors DNS requests and responses, binning these requests and responses into time bins of 10 seconds. In each time bin, Gigascope generates a tuple of DNS request and response count seen for every DNS name for which a request or response was observed in that time bin.

- We constructed an AS health index using the logs of the multi-user interactive application outlined above. These logs are generated by a Gigascope running at the server, but we built the index based on the logs stored elsewhere, to which the log files are transferred from the server with a few minutes delay. The basic intuition we employed in developing this index is the fact that the "health" of an AS is "bad" if a significant increase in the number of lost IPs was observed "only" in that particular AS.

- 68 nodes in the PlanetLab testbed[20] were employed for performing active measurements supporting 3 different measurements:

  Measuring download time from a set of Web sites With the download times, the actual IP addresses the downloads were performed from were also returned, to optionally perform traceroutes to the servers that responded.

  Performing DNS lookups for a set of DNS names Along with the status of the lookup (success/failure), the TTL value in the responses were also returned to detect if any of the responses were from the cache.

  Performing AS-level traceroute to a set of IP addresses We used the IP to AS mapping generated in [14] to return the AS-level paths to these addresses.

  Scripts in either perl or ruby were written to perform each of the above measurements.

Note that this set of data sources includes querying from passive measurements and from historical measurements, and triggering of active measurements. In all of these cases, the DSMI had to execute the same sequence of actions. On receiving the REQ message, it parses the message and invokes the appropriate command or appropriate query. The DSMI then reads in its output, constructs the RESP message and sends it back to the node that issued the request.

6.2.2 Applications

We now outline the working of the application code that retrieved measurement data from the above sources and triggered new measurements when required. In the performance-based ranking application, once every 2 hours, the application polls a random subset of the 68 PlanetLab nodes, requesting them to measure the download time from each of 10 popular Web sites. The subset of PlanetLab nodes is chosen such that no two are within 20 ms latency of each other. As mentioned before, along with the download time, the PlanetLab node also returns the IP address the download was performed from. The application then requests the AS-level paths to these IP addresses and stores the AS-level path for every (PlanetLab node, Web site) pair.

Once every 10 minutes, this application also polls the AS health index. When the application finds the health of some AS to have fallen below a threshold, it looks up the stored AS-level paths returned previously by the PlanetLab nodes and triggers new download time measurements from each PlanetLab node to the Web sites, the paths to which pass through the "low-health" AS.

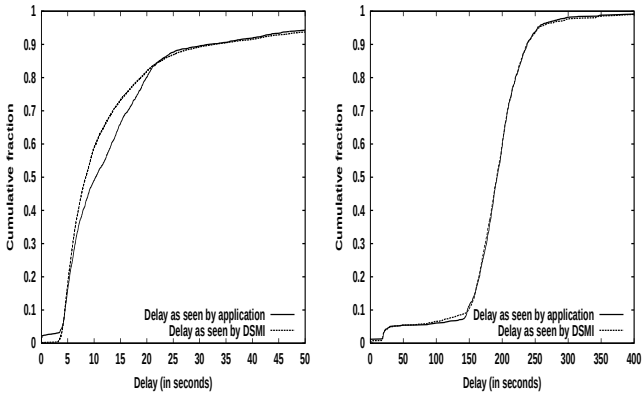


Figure 8: Distribution of delays as observed by the application and by the DSMI for making (a) download time and (b) AS traceroute measurements

In the DNS availability monitoring application, the application initially subscribes to the 3 Gigascopes. It then continuously receives tuples, each of which contains a count of the number of DNS requests and responses observed in that time bin for some DNS name. When the application observes that for some DNS name, the number of DNS requests and DNS responses do not match up over consecutive time bins[3], it polls a random one-third of the 68 PlanetLab nodes to perform a DNS lookup for that DNS name.

7. RESULTS

We now present a summary of the results obtained from our deployment of the two applications – performance-based ranking and DNS availability monitoring. These results were collected based on the deployment of both applications over a period of 7 days between November 3–9 2004. Our primary objectives in the evaluation of either application were to quantify the savings due to our triggered measurement setup and to demonstrate the low overhead of our system. When an App requests measurement data from a DS, the minimum delay it will experience is the RTT between the two sites added to the time taken by the DS to either perform an active measurement or query from passive/historical measurements. We study both the applications we deployed to determine what overhead our system adds over and above this minimum delay.

7.1 Performance-based ranking

Savings in this application were implicit because the setup drew upon the results of our study of the temporal and spatial stationarity of download time. Due to temporal stationarity, we measured the download time from each of the 10 Web sites only once every 2 hours and due to spatial stationarity, these measurements were made only from around 20 of the 68 PlanetLab nodes whenever a measurement needed to be made. Based on our prior experience with the AS health index, we used the threshold to identify a "low-health" AS to be 0.9. During the 7 day period, there was not a single instance when the health of an AS fell below 0.9 and so, no download time measurements had to be triggered in addition to those made once every 2 hours.

[3]Note that irrespective of size of the time bin, the response for a request need not necessarily be in the same time bin.

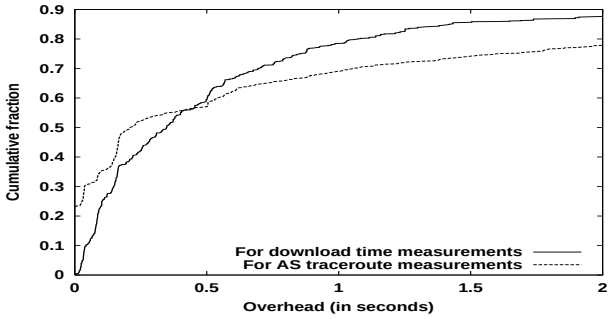


Figure 9: Overhead due to system for measurements made in the performance-based ranking application

To quantify the overhead of ATMEN, we measured the delay in obtaining measurement data as seen from within the application and as seen from within the DSMI. The delay experienced by the application is the time between when the application makes a *getMeasurementValue* call on the AMI to when the application detects that it can read in data on the file descriptor returned to it by the AMI. The delay seen from within the DSMI is the time of execution of the script/command it launches to obtain the measurement data. The difference between these two delays minus the RTT between the sites hosting the application and the data source is the overhead of the system.

In the performance-based ranking application, there are 3 kinds of data sources - download time, AS traceroute and AS health. The machine on which we ran the AS health source is in the same subnet as the one on which we ran the application. Hence, we only consider the overhead in obtaining measurement data from the download time and AS traceroute sources which are hosted on the PlanetLab nodes. Figure 8 shows the distribution of the delay as seen by the application and as seen by the DSMI, both for download time and AS traceroute measurements. Both the distributions are almost identical for either measurement parameter, which indicates that the overhead due to the system is extremely low. We further confirm this in Figure 9 which plots the distribution for either kind of measurement the difference between the two delays minus the RTT between the node hosting the application and the PlanetLab node that made the measurement. The range of values taken by the overhead, in comparison with the range of values taken by the overall delay, show that the overhead due to the system constitutes only a minute fraction of the overall delay.

7.2 DNS availability monitoring

Over the 7 day deployment period, the DNS availability application spanned 59728 time bins. During this period, 822383 tuples were generated by gs1 and 3218237 tuples were generated by gs2 and gs3. Based on this data, the application triggered 86318 and 557796 DNS lookups, respectively. Due to our setup of triggering active measurements based on measurements made passively, we obtain more than 80% savings in the number of DNS lookups that need to be performed.

Like in the performance-based ranking application, here too we plot the distribution of the delays in obtaining mea-

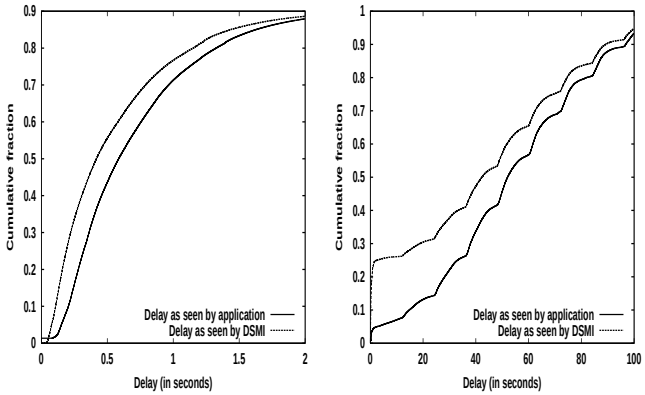


Figure 10: Distribution of delays as observed by the application and by the DSMI for making DNS lookup measurements based on data from (a) gs1 and, (b) gs2 and gs3

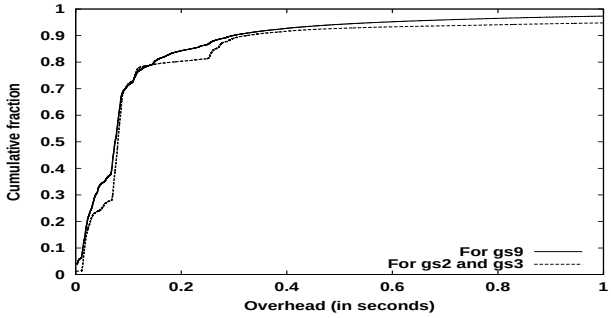


Figure 11: Overhead due to the system for measurements made in the DNS availability application

surement data as seen by the application and as seen by the DSMI. Figure 10 shows these distributions for DNS lookup measurements that were triggered based on data from gs1 and from gs2 and gs3, respectively. The range of delays observed in the latter case is greater because there are a greater number of lookups triggered per request due to the higher traffic volume going through gs2 and gs3 per time bin. As in the performance-based ranking application, these graphs too show that the distribution of delays as seen by the application and as seen by the DSMI are almost identical. We again computed the overhead due to the system by subtracting the RTT between the machine hosting the application and the PlanetLab node on which the DNS lookup was made from the difference between the delays seen by the application and the DSMI. Figure 11 plots the distribution of this overhead. In this application too, we see that the overhead constitutes only a minor fraction of the overall delay.

8. CONCLUSIONS AND FUTURE WORK

We designed and implemented ATMEN—a triggered network measurement infrastructure. We used it to construct two important Web applications – ranking of Web sites according to download time and monitoring of DNS server availability. Our study of temporal and spatial stationarity of the components of download time showed the significant potential for measurement reuse in the performance-based ranking application. We also demonstrated partial reuse of measurements across completely unrelated applications. Results from our deployment of both applications over a 7 day period showed the significant savings in a triggered measurement setup. The low overhead observed due to our system demonstrated its scalability. We are improving the AMI to further reduce the communication overhead. The AMI could cache results received from the name server and from different DSes and store them in a standard configuration file to share across Apps. The AMI can take advantage of the potential for measurement reuse transparent to the App. When the App requests data for some measurement parameter, the AMI can fetch the data for component measurements that the parameter can be broken down into, compose the values obtained for these components and hand the value for the requested parameter back to the App. In other future work, we are examining a range of applications that could benefit from ATMEN including providing early warning for attacks when seen in some part of the network.

9. REFERENCES

[1] S. S. Chen, B. Tung, and D. Schnackenberg. The common intrusion detection framework. In *Proc. of Information Survivability Workshop*, 1998.

[2] Ludmila Cherkasova, Yun Fu, Wenting Tang, and Amin Vahdat. *ACM Transactions on Internet Technology*, 3(4):347–391, 2003.

[3] B. N. Chun, J. Lee, and H. Weatherspoon. Netbait: A distributed worm detection service. Technical Report IRB-TR-03-033, Intel Research Berkeley, 2003.

[4] C. D. Cranor et al. Gigascope: A stream database for network applications. In *Proc. of SIGMOD 2003*, 2003.

[5] C. Papaddopoulos et al. Cossack: Co-ordinated suppression of simultaneous attacks. In *Proc. of Discex III*, 2003.

[6] eValid WebSite Analysis and Testing Suite.

[7] 1999 Fortune 500 companies, 1999.

[8] Internet health report. `http://www.internethealthreport.com/`.

[9] Internet storm center. `http://isc.sans.org`.

[10] J. Jannotti. Network layer support for overlay networks. In *Proc. of IEEE OPENARCH 2002*, 2002.

[11] C. R. Simpson Jr. and G. F. Riley. Neti@home: A distributed approach to collecting end-to-end network performance measurements. In *PAM 2004*, 2004.

[12] Keynote. `http://keynote.com`.

[13] B. Krishnamurthy and J. Rexford. *Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement*. Addison-Wesley, 2001.

[14] Z. M. Mao, J. Rexford, J. Wang, and R. Katz. Towards an accurate as-level traceroute tool. In *Proc. of ACM SIGCOMM 2003*, 2003.

[15] Media Metrix. `http://mediametrix.com`.

[16] D. Mosberger and T. Jain. httperf: A tool for measuring web server performance. *ACM SIGMETRICS Performance Evaluation Review*, 26(3):31–37, 1998.

[17] A. Nakao, L. Peterson, and A. Bavier. A routing underlay for overlay networks. In *Proc. of ACM SIGCOMM*, 2003.

[18] The Netcraft Web Server Survey. `http://netcraft.co.uk/survey`.

[19] Site monitoring tool: Reliable Service by Netmechanic.

[20] Planetlab. `http://planet-lab.org`.

[21] S. Seshan, M. Stemm, and R. H. Katz. Spand: Shared passive network performance discovery. In *USITS'97*.

[22] S. Srinivisan and E. Zegura. Network measurement as a cooperative enterprise. In *Proc. of IPTPS'02*, 2002.