A Scalable Agent-Based Network Measurement Infrastructure

Yulia Indrayani Wijata, Douglas Niehaus, and Victor S. Frost, University of Kansas

ABSTRACT

The rapid growth of computer networks has made the process of understanding the interaction among network components more challenging than ever. The increase in the size of networks is accompanied by the more demanding use of networks by distributed applications that critically rely on the system to function well. Consequently, monitoring the health and stability of networks has become crucial. Tools and probes to measure the performance of networks for the purpose of management, fault diagnosis, or performance evaluation have been developed by several research groups. There is not yet, however, a measurement infrastructure which offers systematic control and management of measurement efforts and performance data focused on supporting distributed network-aware applications. This work addresses the implementation of a scalable and extensible network measurement infrastructure used to capture network state to improve the performance of a distributed application.

INTRODUCTION

Recent developments in the area of high-speed networking stimulate the implementation of large-scale distributed applications. Currently, these applications rely on the best-effort service offered by the network and sometimes suffer low performance when the underlying network behaves unexpectedly. Most current applications are generally oblivious to variation in network conditions. Evolving network-aware distributed applications attempt to alleviate this problem by capturing the state of the network and using this information to *adapt* to the changing conditions of the network.

To support such adaptation, a network-aware application needs to maintain a view of the network state that is generally dynamic, transient, and sometimes tightly coupled with the semantics of the application. An application's view of the network includes knowledge of the topology, availability of resources, and the available quality of service (QoS).

Numerous efforts have been devoted to monitoring and probing the network for the purposes of fault diagnosis, network management, and performance evaluation. Very few, however, are targeted to helping distributed applications make intelligent decisions about how to utilize resources. An application needs an additional support layer through which it can express its network requirements, and at the same time maintain its view of the network state.

Keeping track of the relevant aspects of network state is a challenging task because it deals with vast amounts of information from a large number of network elements that span multiple administrative boundaries. Each piece of information needs to be collected using an appropriate measurement methodology and should be organized systematically to ensure timely retrieval and meaningful interpretation. We approach this problem by deploying a collection of software agents to provide integrated control of monitoring network elements and collected performance information.

In the context of this article, a network monitoring agent is defined as an autonomous entity whose responsibility is to *automate* one or more of the following tasks:

- Continuous monitoring of application components and network characteristics
- Creation and control of network testing and measurement daemons
- Collection and storage of performance data
- Correlation and presentation of performance data to applications and/or users

In particular, knowledge and query manipulation language (KQML) [1] based agents are used to wrap existing tools — in this case, network monitoring and measurement tools — with software, thus enabling them to communicate via a common agent protocol. With this approach, the complex task of monitoring a large distributed system can be decomposed structurally into domain-specific tasks while maintaining a common goal.

A number of systems have been developed for gathering performance measurements on large networks. For example, the National Internet Measurement Infrastructure (NIMI) [2] is aimed at using active measurements for fault diagnosis in very large networks such as the Internet. The Surveyor effort [3] is an infrastructure that measures end-to-end unidirectional delay, packet loss, and route information along Internet paths, globally deployed at about 50 higher education and research sites. The Surveyor system relies on the use of dedicated computers and hardware to obtain accurate measurements, including synchronized clocks for timestamps. The infrastructure

This research is partially funded by the Defense Advanced Research Agency (DARPA) under contract F19628-95-C-0215.

Editorial liaison: T. Chen

0163-6804/00/$10.00 © 2000 IEEE

described here has several common elements with existing measurement systems, for example, scheduling and execution of active performance monitoring sessions, and collection and storage of performance data. There are several important differences, however; the methodology proposed here is aimed at the correlation and presentation of performance data to be used by network-aware distributed applications. Here a specific distributed application drives the measurement process and uses the results of the measurements at both the application and network layers to improve its performance. Thus, the specific monitoring/instrumentation configurations used to obtain the performance information are tailored to the needs of a specific distributed application. The capabilities of the proposed measurement infrastructure were prototyped and demonstrated using the MAGIC-II [1] Distributed Parallel Storage System (DPSS) [4]. Another novel feature of the proposed infrastructure is the use of KQML-based agents to leverage and reuse existing tools. Note that this feature could be used to take advantage of the capabilities of NIMI within the context of a network-aware distributed application.

Design Criteria

The following design guidelines have been adopted to achieve the design objectives:

- Modularity: The software agent framework must promote modular design, which clearly separates policy from mechanism in measurement.
- Portability: Since a distributed system most likely comprises heterogeneous components and systems, the agents should be easily portable to different architectures.
- Distributed: The system must be capable of monitoring network elements in more than one administrative domain.
- Extensibility: The capabilities of the agents in the system should be easily extended to support new types of measurement or testing.

The Approach

This section describes the two major software tools used as components of the monitoring system: NetSpec, a distributed network performance evaluation tool from the University of Kansas [5], and JATLite, a Java Agent Toolkit package from Stanford University [6].

Using NetSpec to Monitor a Wide Area Network

NetSpec [5] was used as the *main control entity* of network testing and measurement because of its capabilities to perform distributed network testing in an integrated and extensible manner. NetSpec provides support for the experimental evaluation of network performance. It was conceived as a general-purpose tool for conducting a wide range of reproducible wide area network experiments. The NetSpec framework permits two types of daemons/probes to collect data from distributed network elements:

- Test daemons generate traffic with different types of characteristics and measure the achievable throughput.
- Measurement daemons perform specific measurements on network elements.

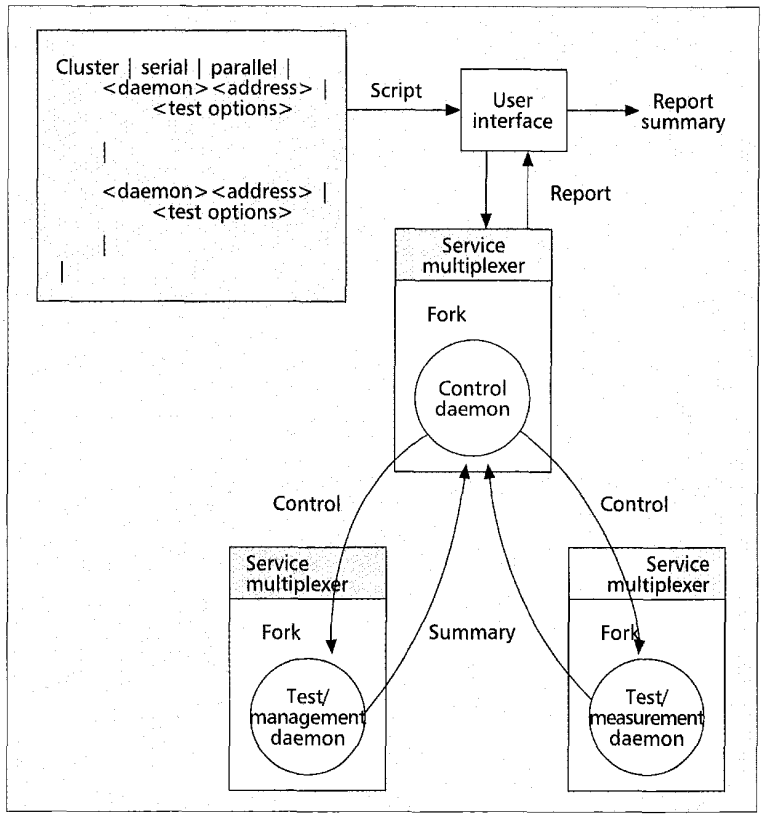


Figure 1. *NetSpec high-level architecture.*

Figure 1 shows the general architecture of the NetSpec framework. The NetSpec *controller* distributes measurement tasks to several network elements based on the testing topology described in a script. In response, the measurement daemons perform the testing or data collection and produce performance reports (similar to the daemons in NIMI [2]). A NetSpec script identifies the nodes involved in an experiment, the roles assumed by each node, the parameters for each measurement or test daemon, and the relationship among daemons.

NetSpec uses a text-based protocol to control the execution of nodes involved in an experiment. The protocol imposes some sequence of phases undergone by each daemon during the execution (Fig. 2). These phases mimic the phases of a typical network connection. In addition, the protocol also supports some administrative commands for control purposes. Table 1 summarizes the commands supported by NetSpec protocol and a description of actions taken when the command is invoked.

JATLite Package

JATLite (Java Agent Template, Lite) [6] is a package of programs written in the Java [7] language that allows users to quickly create new software agents that communicate robustly over the Internet. JATLite provides a basic infrastructure for agents' communication based on

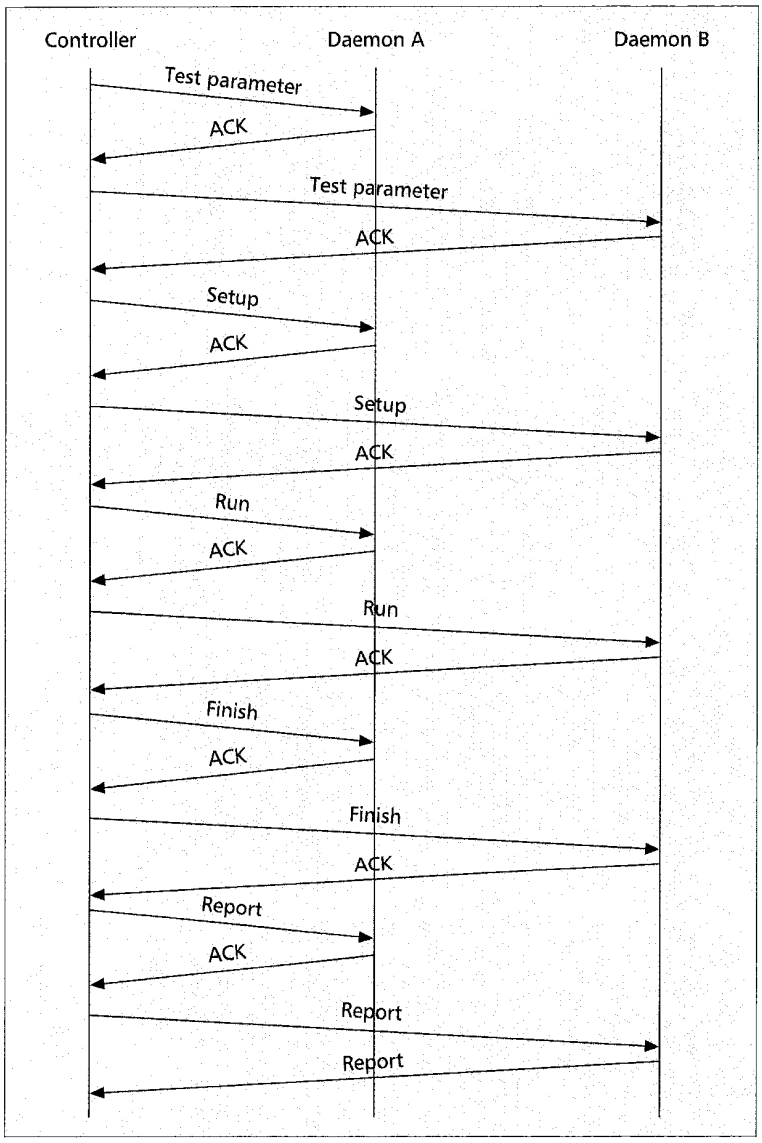


Figure 2. *Distribution of the control protocol.*

Command	Action
Setup	Allocate resources
Open	Establish connection
Run	Start data transfer or measurement
Finish	Finish data transfer or measurement
Close	Close connection
Tear down	Free up resources
Report	Send report summary
Reset	Reset to initial state
Kill	Stop execution
Parameters	Accept test parameters
Config	Return configuration information

Table 1. *NetSpec protocol command and execution phases of a NetSpec daemon.*

TCP/IP and KQML messages. The use of Java allows the agents to run on heterogeneous platforms and thus increase portability. Its modular construction consists of a hierarchy of increasingly specialized layers which may be customized to fit the specific requirements of a given system. One important concept in the JATLite framework is the *agent message router* (AMR), also referred to as the *router*. It provides name registration and message routing or queuing for agents. In this scheme, agents can operate in disconnected mode and still receive the messages addressed to them. Another advantage is that the existence of an agent is transparent to the other agents in the system. An agent can send a message to another agent in the system by indicating the registered name of that agent in the destination field of the message and then sending the message to the router. The router then will forward the message to its intended recipient as long as it has registered itself with the router.

The JATLite package provides a convenient mechanism to wrap network performance measurement tools, such as NetSpec, with software which provides a standard protocol for communication.

An Agent-Based Infrastructure

We approach the problem of creating the measurement infrastructure by defining a collection of software agents to provide integrated control of monitoring and monitored entities as well as the collected information. With this approach, the complex task of monitoring a large distributed system can be decomposed structurally into some domain-specific tasks while maintaining the common goals.

Several types of agents may exist in the system:

- *An application agent* possesses the domain knowledge specific to a network-aware distributed application and performs monitoring at the application level.
- *A measurement agent* creates and controls network experiments on an application agent's demand and manages data collection. The important role of agents of this type is to translate the request from an application agent into test parameters, and to select a network experiment that can fulfill the application monitoring requirements.
- *A presentation agent* correlates and applies application semantics to the collected data for visualization purposes.

Figure 3 shows an example of how these agents interact in the network. A network-aware distributed application has a monitoring demand, for example, to select a *closest* server which is determined by the shortest round-trip time to a client. Its application agent formulates a request to the measurement agent to perform delay measurement between a client and the servers. The measurement agent will then create a network experiment with the requested topology, in this case point-to-multipoint, to measure the round-trip time between the specified network nodes. At a later time, a presentation agent may retrieve the collected data for real-time capture of the network quality perceived by the application.

A Functional Overview

This section provides the high-level architecture of an example monitoring system developed in the MAGIC-II network [1]. As implied by the previous section, monitoring is done at two different levels: the network and application levels. Currently the routers and/or switches in the network are not involved in the measurement process; only end systems participate in the collection of performance information. The focus of this article is on the network level, while both components have been used to support dynamic reconfiguration (i.e., performance tuning and optimization of the distributed application), here the DPSS in the MAGIC-II testbed. A discussion of DPSS performance tuning and optimization can be found in [8].

Figure 4 shows the functional overview of the components in this system. The MAGIC-II testbed cloud represents the wide-area asynchronous transfer mode (ATM) network. The DPSS represents the distributed application being monitored by the agents.

There are four types of agents:

- ***NSAgent*** is a JATLite measurement agent that performs network-level monitoring. It creates and schedules NetSpec experiments and organizes the performance reports. The type and parameters of the experiment can be loaded dynamically. NSAgent collects the information about the DPSS system from the ServerMonitor. The NSAgent will be described in more detail later.
- ***VisAgent*** is a JATLite presentation agent with a front-end applet, which visualizes the state of the network, the distributed application, and the agents' configuration. The information is collected from the NSAgent and ServerMonitor.
- ***ServerMonitor*** is a JATLite application agent, which monitors the status and configuration of the application, in this case, a DPSS system.[1] This agent was developed by Lawrence Berkeley National Laboratory; this agent is discussed in [8].
- ***HostMonitor*** is also a JATLite application agent, which keeps track of the status of the currently connected DPSS clients.

Each of these agents registers itself with the JATLite router when it starts up. The agents exchange KQML messages with other agents in the system via the router.

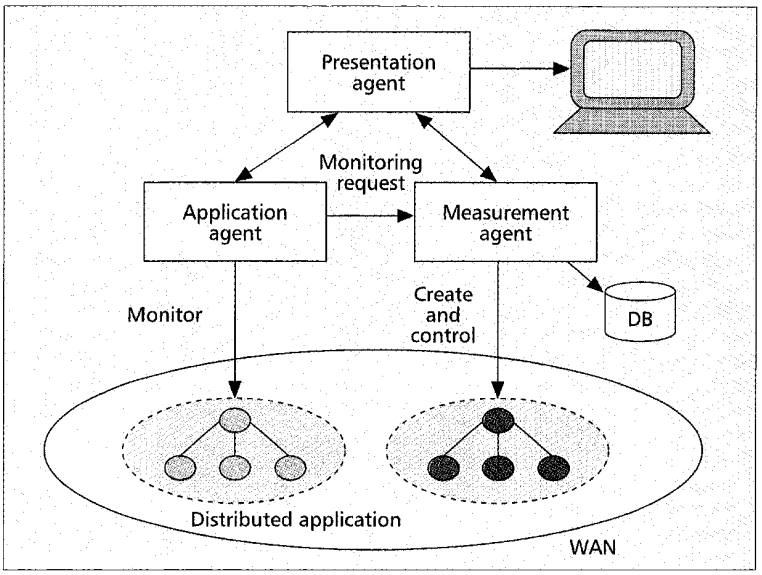


Figure 3. *Relationships among agents in the monitoring system.*

Implementation of NSAgent

The main responsibility of the NSAgent is to capture the state of the network. It does this by performing the appropriate tests and measurements in the network. The results of monitoring and

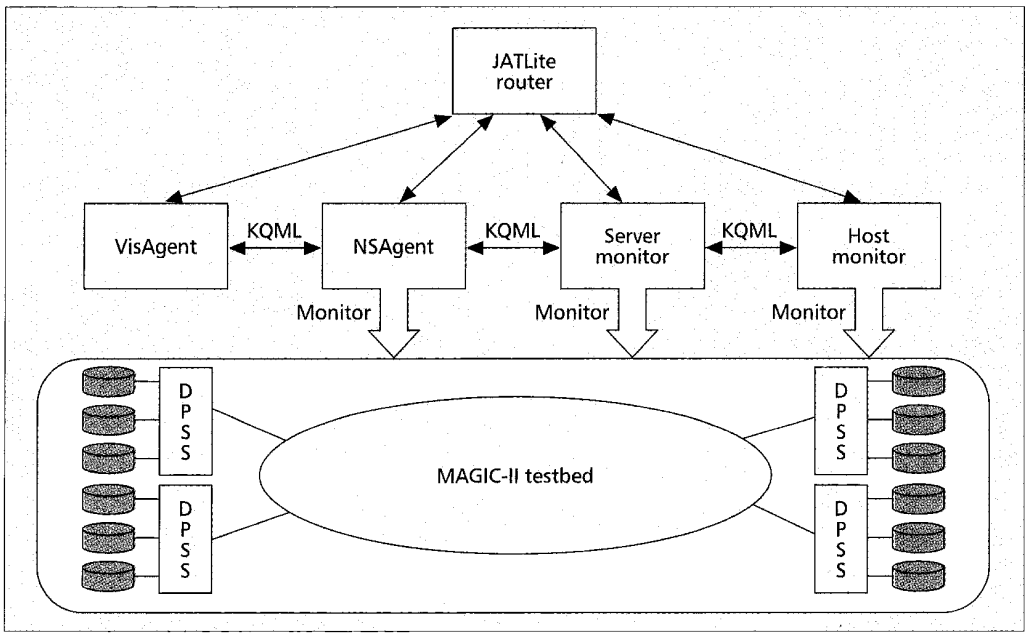


Figure 4. *A functional overview.*

[1] *A DPSS system consists of a DPSS master and one or more DPSS server(s).*

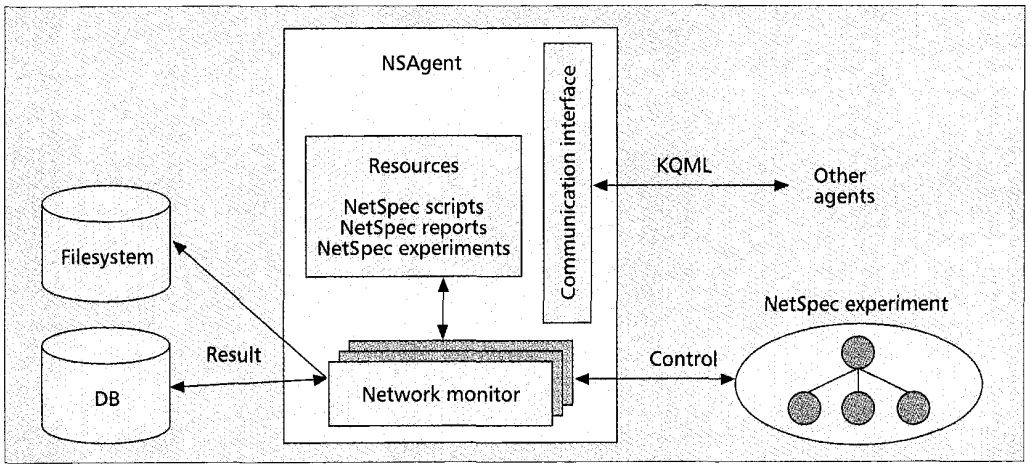


Figure 5. *NSAgent architecture and external components.*

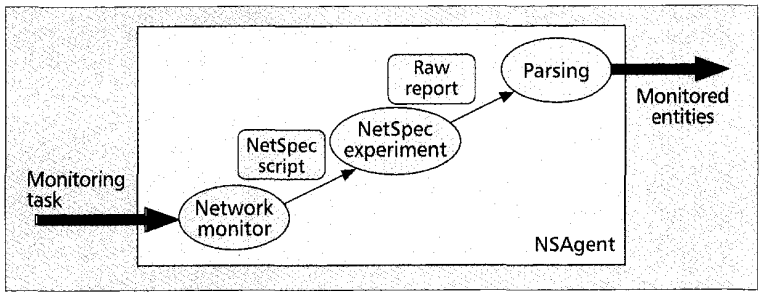


Figure 6. *The network monitoring process.*

measuring the network characteristics can be used for different purposes. In the MAGIC-II testbed, the main objective is to use the knowledge about current conditions in the network to dynamically select the best server in the DPSS system. The best server has the highest available link bandwidth and the lowest round-trip time. The NSAgent is also used to perform general network monitoring tasks such as connectivity or throughput tests to measure the health of the network.

A major consideration in NSAgent implementation is to design an extensible framework which can accommodate future types of network measurements. Although the main objective in this effort was to support the dynamic reconfiguration of the DPSS system, NSAgent should fulfill the ultimate goal of capturing the network state for a wide variety of network-aware distributed applications.

NSAgent Architecture

Figure 5 shows the architecture of the NSAgent. JATLite provides the basic communication interface based on a TCP/IP socket via the router for receiving and sending KQML messages. The NSAgent has a collection of templates for NetSpec scripts which represent network experiments and reports. It receives a monitoring task from application agents and creates the appropriate network monitor script/experiment. The results of the NetSpec experiments are stored in the database or file system for future retrieval.

The message handling of the agent is quite simple. When a KQML message destined for an agent arrives at the router, it stores the message in the incoming message box for that agent and then notifies the agent. The agent is responsible for retrieving its own messages and deleting them afterward.

Knowledge Representation

A main component of a software agent is its knowledge base (KB). It provides the context of agent execution and knowledge about its environment. The representation of knowledge can vary according to its purpose. NSAgent has two types of knowledge representation. *Procedural representation* uses program functions or methods to represent the data and operation associated with an object. This type of representation is particularly useful in defining the capabilities of an agent. The second type of knowledge representation in NSAgent is a *relational database*. It serves as a back-end store of static or runtime data, and provides convenient and structured access and manipulation of data.

In NSAgent, the procedural representation is also called a *resource*. The following sections explain the types of resources defined in NSAgent, and describe the structure and configuration of the database.

Resources — NSAgent uses resources to store the knowledge needed to conduct network experiments. Figure 6 shows the conceptual process of creating a network experiment using the agent framework developed here. Given the task of monitoring a specific network characteristic, NSAgent will create the appropriate *network monitor* object. The monitor object will create the *NetSpec script*, which will invoke the proper NetSpec measurement daemon(s) for the task. Then it will need to parse the performance *report* generated by the NetSpec daemon and interpret the result.

NSAgent defines the following resource abstractions:

- Network monitor: This represents the object that controls a NetSpec experiment. Every network monitor object encodes the *type* and *topology* of a NetSpec experiment. The type is associated with the kind of NetSpec measurement daemon(s) that need be executed. The topology defines the execution

construct and participants of the experiment. A monitoring task must specify the type of monitor object to create. It can also specify the duration of the experiment and storage method. For example, we can define a monitor object that performs a full mesh throughput experiment among N nodes. In this case, the type of experiment (throughput) indicates that we have to use NetSpec test daemons. The full mesh topology will automatically create a script of N^2 end-to-end experiments between each pair of nodes in the set of N specific nodes.

Table 2 summarizes the types of network monitors implemented. *Nstestd* is a NetSpec test daemon that can generate different types of network traffic and measure the achievable throughput. *Nspingd* is a NetSpec measurement daemon that measures round-trip time by sending the ICMP_ECHO message to network nodes. *Nssnmpd* can collect a variety of SNMP data from a network element or host.

- NetSpec Script: A NetSpec script object contains the parameters and options of a NetSpec daemon and the methods to generate the script. A class must be defined for each NetSpec daemon since each test or measurement daemon has different test parameters. For example, a NetSpec script class for the NetSpec test daemon defines the test parameters such as type of traffic, protocol, and end nodes. It also needs to implement the method that generates the topology of the experiment.
- NetSpec report: Since there is no standard report format defined in NetSpec, each daemon generates varying types of report, making parsing and interpretation of the report particularly difficult. To handle this problem, NSAgent defines a NetSpec report class for each type of daemon. Each class provides methods to parse and store the results.

Database — The NSAgent uses the database to store static configuration information about the network (e.g., names and addresses of end hosts or switches) and the configuration of the distributed application. The results of network experiments can also optionally be stored in the database for further retrieval by other agents. Since the goal is to capture the state of the network, only the most recent result is kept in the database. Historical results can optionally be cached in the file system. In our implementation, we have decided to use *msql* (mini standard query language) [9], a lightweight relational database based on SQL for storage.

The Monitoring Task

NSAgent's main capability is to accept a monitoring task and perform the corresponding monitoring action. A monitoring task is encapsulated in a KQML message and can be submitted by other agents in the system. NSAgent also provides a convenient way to load tasks from a file during agent startup. Routine monitoring tasks can be loaded from this initialization file.

A monitoring task must identify the following attributes:

- The type of network monitor
- The frequency of the experiment
- The storage method (to database or file system)
- The parameters of the experiment (specific to the type of network monitor)

Name	Topology	Daemon type
FullMeshThroughput	*N*-to-*N* (full mesh)	nstestd
PointToMultiPointThroughput	1-to-*N* (star)	nstestd
EndToEndThroughput	1-to-1	nstestd
FullMeshDelay	*N*-to-*N*	nspingd
PointToMultiPointDelay	1-to-*N*	nspingd
EndToEndDelay	1-to-1	nspingd
HostMonitor	1-to-*N*	nssnmpd

Table 2. *Implemented network monitors.*

KQML Messages

This section describes the KQML messages implemented in the NSAgent. The types of messages can be broadly categorized into three domains according to the audience of the messages:

- DPSS (application): Messages in this domain deal with the configuration of the DPSS and queries about network status from the DPSS.
- NetSpec: These messages provided an interface to create or terminate NetSpec experiments, including the loading of a monitoring task.
- Visualization/presentation: The NSAgent also serves as the information provider for a visualization agent.

See [10] for a detailed description of the KQML messages.

An Example Configuration in the MAGIC-II Testbed

This section describes the configuration and capabilities of the monitoring system described above. Figure 7 shows the configuration of the demonstration. We had eight DPSS servers distributed over the national-scale high-speed ATM MAGIC-II network deployed from California to Kansas and South Dakota. The agents, JATLite router, database server, and Web server were running on a workstation (a Sun Ultra Sparc running Solaris 2.6) at the University of Kansas, Lawrence. NetSpec was installed on all DPSS hosts and on at least one machine at each site.

NSAgent Configuration

For the demonstration, the NSAgent was configured to schedule and run a number of network experiments using NetSpec. Each network measurement was encapsulated in a monitoring task presented to the NSAgent during its initialization. By default, NSAgent loads a series of KQML messages from an initialization file that defines the agent's initial behavior. The following subsec-

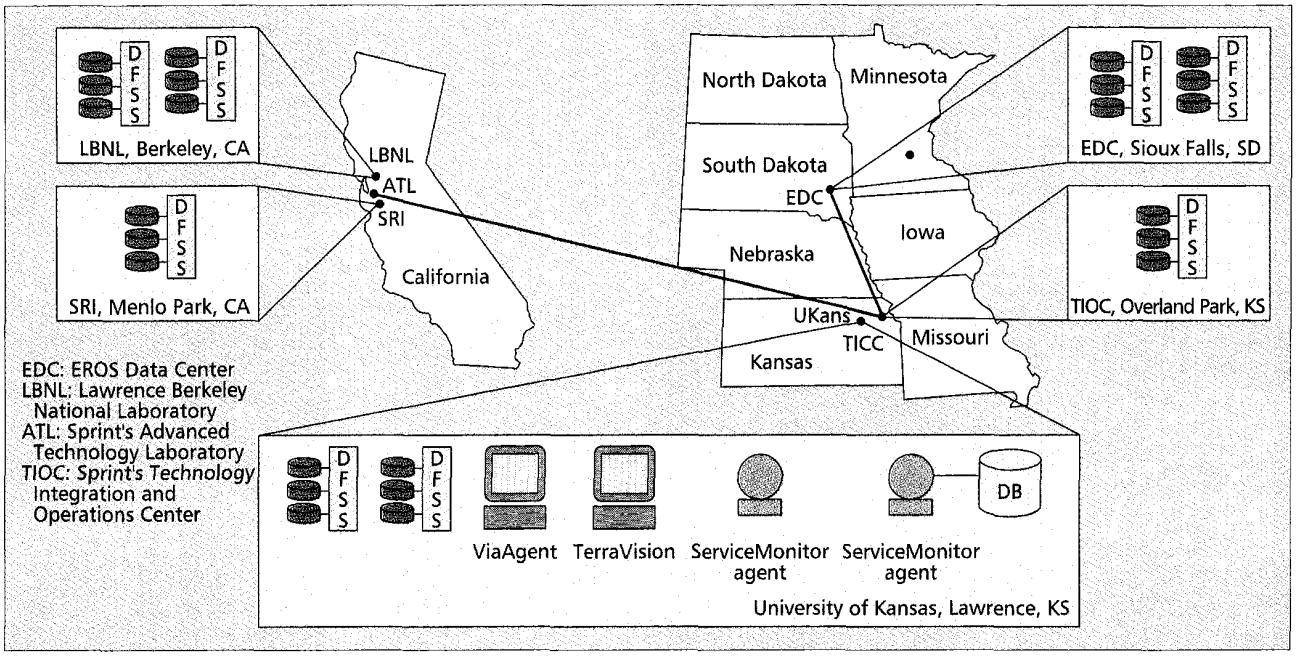


Figure 7. *The demonstration configuration.*

tions describe each monitoring task and its significance as well as some example results obtained from the measurement experiments.

Monitoring Link Quality between DPSS Client and Servers — The objective of this activity was to determine the link quality between a DPSS client and the servers configured in the system in order to select the server with the best connectivity to the client. The link quality is defined in terms of round-trip delay and throughput of the link.

When a new DPSS client joins the system, the NSAgent must be notified about its existence. The entity that must register the client to the NSAgent can be the client itself or a DPSS monitor agent which continually keeps track of the emergence of a new client. However, for these experiments a static configuration was used. The following KQML message was included in the initialization file to register a DPSS client named *tv-client* with only one network interface, *terravision.ukans.magic.net*, which communicates with one of the eight DPSS servers.

```
(evaluate :sender NSAgent :receiver NSAgent
   :content (tell-resource :type client :name tv-client
      :pos (38.963 -95.233)
      :interface (terravision.ukans.magic.net 198.207.143.157)
      :server (lbl-server4 sri-server1 tioc-server1 tioc-server2
         lbl-server3 ku-server1 edc-server1 edc-server2)))
```

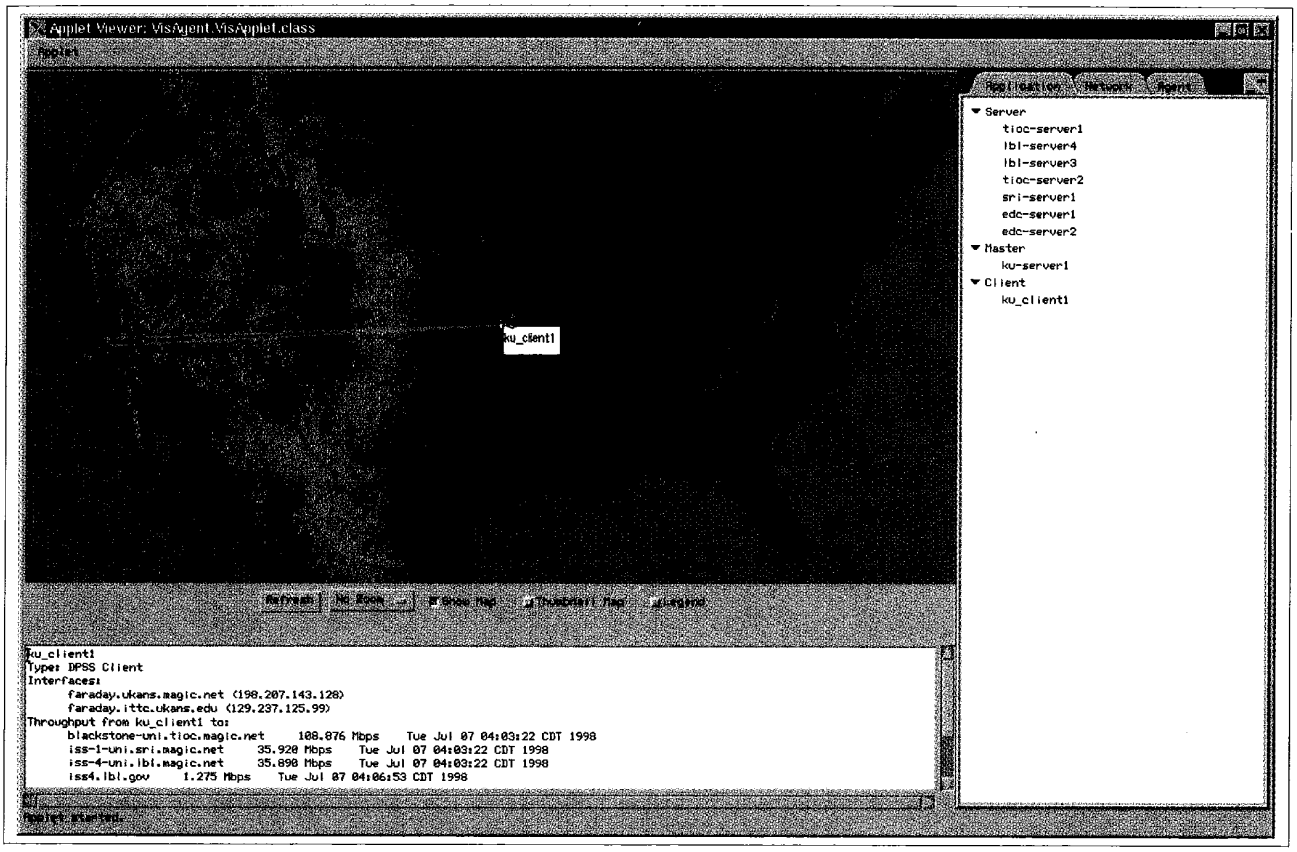
Upon receiving this message, the NSAgent instantiated two types of network monitors (Table 1):

- *A PointToMultiPointThroughput* monitor measures the transfer capacity (throughput) of the links between the client and the servers. If either the client or the server has more than one network interface, each interface will be tested. This experiment was conducted once every 30 min. The results of the experiment are stored in the database.
- *A PointToMultiPointDelay* monitor measures the round-trip time between the client and the servers. Since this type of experiment produces significantly less disturbance to the network, it can be done more frequently (once every 15 min).

Monitoring Connectivity in the Network — The objective here is to monitor the connectivity in the network. Often it is difficult to identify the cause of the problem when an "Unreachable destination" message is received because the network has many potential points of failure. This is especially true in an experimental testbed such as MAGIC-II where the configuration of the network may change frequently. We found that the set of reachable machines/sites varied greatly from one machine to the next in the network.

The objective of the measurement activity was to provide better understanding of how the sites in the network are connected to each other and what kind of connectivity exists at a specific point in time. Here, every 15 min NSAgent schedules a *FullMeshDelay* experiment among the major hosts in each site in the network. Since the *FullMeshDelay* network monitor uses the ICMP_ECHO mechanism to measure round-trip time, it was utilized to test the connectivity from one point in the network to several other points.

Monitoring Transfer Capacity of the Network — Besides monitoring connectivity, another important metric in assessing the general health of a network from the perspective of the target distributed application is accomplished by measuring link throughput. Variation in throughput is generally affected by the amount of traffic and presence of bottleneck links in the network. For this purpose, NSAgent creates a *FullMeshThroughput* experiment among the major hosts in each site. Since this type of experiment



■ **Figure 8.** *A screenshot of VisAgent and the application layer. Nodes represent DPSS master, servers and clients. Lines represent active connection between a DPSS client and servers.*

introduces a large amount of test traffic to the network, it should not be done frequently. It is important, however, to note that the study of variation in throughput during the course of the day should be done at different times during the day.

Monitoring Network Element Status — While the measurement activity described above is aimed at providing information about connectivity between sites in the network, it does not provide detailed information about each machine or other network elements within a site. This type of monitoring can be achieved by doing a *PointToMultiPointDelay* experiment between a host and the remaining hosts and switches in each site. The database contains the static information about the configuration of each site. NSAgent uses this information to create an experiment that sends an ICMP_ECHO message every 15 min from a designated host to the rest of the network elements and hosts at its site.

VisAgent Configuration

To visualize the state of the network from the perspective of the network-aware distributed application, the VisAgent is used from a Java-capable browser. VisAgent collects the performance data and static configuration from three sources: the NSAgent, database server, and ServerMonitor agent. KQML messages are used to communicate with the agents, while standard SQL messages are used to interact with the database server. VisAgent provides layered views at the application and network layers:

- Application layer: Color-coded nodes represent the location of DPSS master, server, and client nodes. The placement of the nodes indicates the geographical location of the corresponding DPSS components. Lines represent the active connection between a client and some DPSS servers. Figure 8 shows the screen capture of the visualization tool displaying the status of the DPSS. In this figure a DPSS client, *ku_client1*, is accessing data sets from two DPSS servers, as shown by the lines connecting the client and the servers. The data panel at the bottom left corner shows detailed information about ku_client1, such as its network interfaces and the results of the throughput measurements. The panel on the right displays the names of the DPSS master(s), server(s), and client(s) that are currently registered with the monitoring system.
- Network layer : Each node at the network layer represents a site in the MAGIC-II testbed. The network layer is further divided into three sublayers:
  - –The topology sublayer shows the physical configuration of the testbed. The width of the lines represents the physical link's capacity (i.e., DS-3, OC-3, OC-12).
  - –The connectivity sublayer shows the results of the connectivity test from a particular site point of view. The color of the lines coming

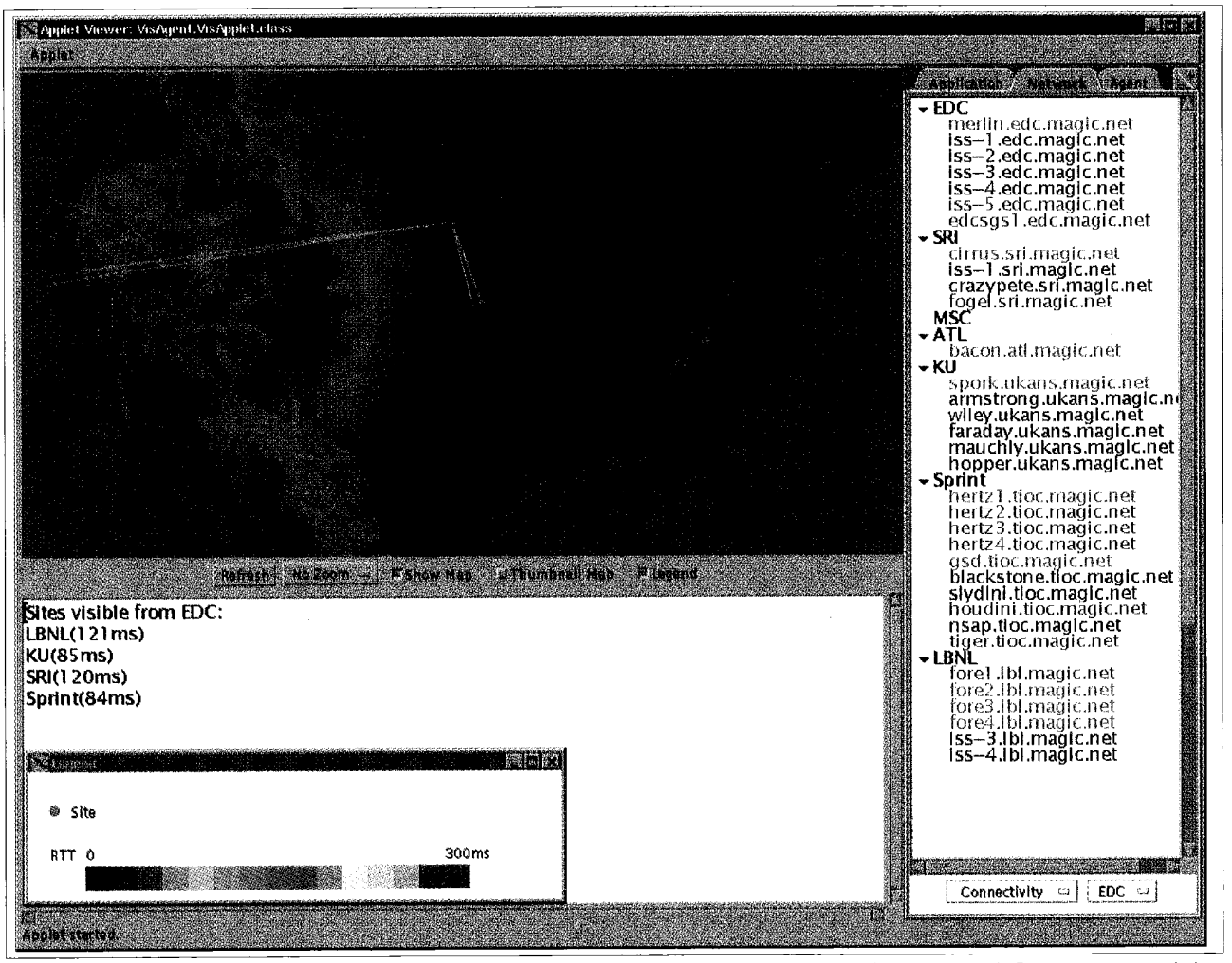


Figure 9. *A screenshot of VisAgent at the network connectivity sublayer. Nodes represent sites (organizations). Lines represent existing connectivity, and their colors represent the round-trip time.*

out from a site to another site represents the round-trip time in milliseconds. Unreachable sites will not be connected by a line. Figure 9 shows a screenshot of the VisAgent displaying the connectivity as seen from the EDC site. The colors of the lines represent the round-trip time values as indicated by the color spectrum in the legend window. The numerical values of round-trip time are also displayed in the data panel.

–The maximum bandwidth sublayer shows the result of the throughput test from the point of view of a particular site. The color of the lines represents the achievable throughput in megabits per second.

Lessons Learned

The complexity, large scale, and accelerating growth of today's computer networks require systematic and automated management of performance measurement for at least three reasons. First, the size of the network and resulting data sets exceeds the capacity of current practice's ad hoc methods. Second, more sophisticated data collection, analysis, and use of results will be required to understand the increasingly complex behaviors of emerging networks that affect the performance of distributed applications. Third, the performance requirements and constraints of emerging distributed applications will also require richer performance data to provide better service through awareness and adaptation to dynamic network conditions. Our experience has shown that the design principles adopted by the system, as described earlier, are vital to success. Modularity that clearly separates policy from mechanism is required to ensure that methods can be conveniently improved, and used as models for extension. Portability is vital because the execution platforms change as quickly as the network to which they are connected. Distribution is necessary because as the target system increases in size, centralized methods are unlikely to remain adequate. Extensibility is required because of the continuous development of new types of tests and data needed to support new types of network behaviors and behavioral analysis. The essential message is that the framework supporting performance measurement must be as capable of dynamic change as the network it measures. Another important lesson distilled from this experience is that it is vital that the instrumentation and other measurement support

be first class design criteria, rather than a sometimes grudging afterthought. Standards for Simple Network Management Protocol (SNMP) based management and data collection fulfill this requirement to a degree, but are not adequate for all types of information gathering. The experience and software described here should guide the design of future network components to help increase the accuracy and decrease the cost of gathering the performance data required to support advance distributed applications.

Conclusion and Future Work

This article describes the design and implementation of a scalable agent-based network measurement infrastructure tailored to support network-aware distributed applications. The agents are implemented in Java, use the JATLite framework, and communicate using KQML. NSAgent's main objective is to perform measurements and tests on the network in order to capture important performance characteristics of the network from the perspective of a distributed application. NSAgent relies on NetSpec to perform the distributed network experiment. The result of the experiment is stored in either the database or the file system. This work describes the implementation of a monitoring agent system as an approach to creating a network measurement infrastructure. The role of the agents in the system is to *automate* the process of:

- Continuous monitoring of application components and network characteristics
- Creation and control of network testing and measurement
- Collection and storage of performance data
- Correlation and presentation of performance data to the application and/or network manager

In the MAGIC-II testbed where this monitoring system was tested, the main purpose was to capture characteristics of the network state to support dynamic reconfiguration of the DPSS, a network-aware distributed application. In addition, the monitoring system was also used to monitor the general health of the MAGIC-II network by measuring some vital metrics of the network, such as connectivity, latency, and throughput. The monitoring system also includes a visualization tool that serves as an interface to the performance data and, more important, provides an integrated view of the distributed application and the underlying network.

References

[1] The MAGIC-II Project, http://www.magic.net
[2] V. Paxon *et al.*, "An Architecture for Large-Scale Internet Measurement," *IEEE Commun. Mag.*, Aug. 1998.
[3] S. Kalidindi and M. J. Zekauskas, "Surveyor: An Infrastructure for Internet Performance Measurements," INET '99, San Jose, CA, June 1999.
[4] T. Finin *et al.*, "Draft Specification of the KQML Agent Communication Language," unpublished, 1993; http://www.cs.umbc.edu/kqml
[5] L. Dasilva *et al.*, "ATM WAN Performance Tools, Experiments, and Results," *IEEE Commun. Mag.*, vol. 35, no. 8, Aug. 1997, pp. 118–25.
[6] Ctr. for Design Res., Stanford Univ., "JATLite: The Java Agent Template," http://java.stanford.edu
[7] J. Gosling and H. McGilton, "The Java Language Environment: A White Paper," Sun Microsystems, 1995.
[8] B. Tierney and B. Crowley, "Java Agents for Monitoring and Management (JAMM) Home Page," Lawrence Berkeley Nat'l. Lab, http://www.didc.lbl.gov/JAMM
[9] D. Hughes, "Mini SQL: A Lightweight Database Server," Bond Univ., Australia, http://www.hughes.com.au/library/msq11/manual/
[10] Y. Wijata, "Implementation of a Scalable Agent-Based Network Measurement Infrastructure to Improve the Performance of Distributed Application," Master's thesis, Univ. of KS, Nov. 1998.

Additional Reading

[1] B. Tierney *et al.*, "An Overview of the Distributed Parallel Storage System (DPSS)," http://wwwdidc.lbl.gov/DPSS/Overview/DPSS.handout.fm.html
[2] CAIDA, The Genmap Package, http://www.caida.org/Tools/Genmap/
[3] B. Crowley, "KQML Messages in the DPSS Agent Monitoring System," http://www-itg.lbl.gov/~crowley/kqml.html

Biographies

YULIA I. WIJATA received her B.Sc. and M.Sc. in computer engineering from the University of Kansas in 1996 and 1998, respectively. In October 1998 she joined Sprint Corporation, Kansas City, as a software engineer in the Service Establishment and Technology Solutions group. She is a member of Eta Kappa Nu and Tau Beta Pi.

DOUGLAS NIEHAUS is currently an associate professor in the EECS Department at the University of Kansas. He has been on the faculty of the university since 1993. His interests include real-time and distributed systems, operating systems, high-performance network implementation and simulation, system and network performance evaluation, and tools for programming environments. Current projects include high-performance simulations of ATM networks on both the signaling and data planes, real-time ORB implementation and performance evaluation, and system support for teams of reflective agents engaged in negotiation about resource allocation. He received his Ph.D. in computer science from the University of Massachusetts at Amherst in 1993, where his thesis addressed the design and implementation of real-time systems. He was a senior software engineer porting UNIX to new platforms at Convergent Technologies in 1986 and 1987, and a member of technical staff doing system, network, and development environment tool programming at Bell Laboratories and AT&T Information Systems from 1981 to 1986. He received his M.S. in computer, information, and control engineering from the University of Michigan in 1981 and his B.S. in computer science from Northwestern University in 1980.

VICTOR S. FROST [F] (frost@eecs.ukans.edu) is currently the Dan F. Servey Distinguished Professor of Electrical Engineering and Computer Science and director of the Telecommunication and Information Technology Center at the University of Kansas. He received a Presidential Young Investigator Award from the National Science Foundation in 1984. His current research interests are in the areas of integrated broadband communication networks, communications system analysis, and traffic and network simulation. He has been involved in research on several national-scale high-speed wide-area testbeds. For example, he was principal investigator on the University of Kansas MAGIC research effort and ACTS ATM Internetwork (AAI). Some of his recent publications have focused on reporting the measured performance of these wide-area broadband networks. His research has been sponsored by government agencies, including NSF, DARPA, Rome Labs, and NASA. He has been involved in research for numerous corporations, including Sprint, NCR, BNR, Telesat Canada, AT&T, McDonnel Douglas, DEC, and COMDISCO Systems. From 1987 to 1996 he was director of the Telecommunications and Information Sciences Laboratory. He has published over 100 journal articles and conference papers, and made several presentations to committees of the Kansas Legislature on telecommunications and the future. He is a member of Eta Kappa Nu and Tau Beta Pi. He served as Chairman of the Kansas City section of the IEEE Communications Society from June 1991 to Dec. 1992. He received his B.S., M.S., and Ph.D. degrees from the University of Kansas, Lawrence in 1977, 1978, and 1982, respectively. In 1982 he joined the faculty of the University of Kansas.

The experience and software described here should guide the design of future network components to help increase the accuracy and decrease the cost of gathering performance data that will be required to support advance distributed applications.