◆ SEQUIN: An SNMP-Based MPLS Network Monitoring System

Marina K. Thottan, George K. Swanson, Michael Cantone, Tin Kam Ho, Jennifer Ren, and Sanjoy Paul

Multiprotocol label switching (MPLS)-based networks are gaining popularity with service providers because they retain the flexibility of Internet protocol (IP) while providing support for quality of service (QoS) through differentiated services (DiffServ). The deployment of such networks requires service providers to ensure that service-level agreements (SLAs) with their customers are being met while guaranteeing cost-effective utilization of their network. The key to achieving these objectives is an efficient monitoring system that tracks QoS metrics such as bandwidth utilization, delay, and packet loss. These metrics can be tracked at multiple granularities, such as per customer, per MPLS tunnel, and per physical network pipe. SEQUIN is a simple network management protocol (SNMP)-based monitoring system capable of tracking QoS metrics in near real time. Some of the challenges addressed by SEQUIN are synthesis of end-to-end QoS using polled data from different network elements, visualization of these metrics across the network, identification of performance problems by correlating the monitored parameters, and design of scalable polling strategies to minimize management traffic. © 2003 Lucent Technologies Inc.

Introduction

Multiprotocol label switching (MPLS)-based networks are gaining acceptance with service providers because they enable switching of packets at high rates while retaining the flexibility of Internet protocol (IP). IP provides support for quality of service (QoS) through differentiated services (DiffServ), and MPLS-based networks enable service providers to preserve this feature. For QoS-capable networks to become widely deployed, service providers must guarantee some form of service-level agreement (SLA) with their customers. Concurrent with providing SLA guarantees, service providers need to ensure that their network is utilized in a cost-effective manner. The key to achieving both of these objectives is an efficient monitoring system that keeps track of bandwidth utilization and QoS metrics, such as delay and loss. These metrics can be tracked at different levels of granularity: for each class of service (DiffServ code points [DSCPs]), for each customer, for each MPLS tunnel, and for each physical network pipe. The SEQUIN monitoring system is designed to track QoS metrics of a service provider network using simple network management protocol (SNMP)-based techniques. It comprises several modules, each of which is responsible for different tasks such as network polling, computation of QoS metrics, and visualization. The

Bell Labs Technical Journal 8(1), 95–111 (2003) © 2003 Lucent Technologies Inc. Published by Wiley Periodicals, Inc. Published online in Wiley InterScience (www.interscience.wiley.com). • DOI: 10.1002/bltj.10049

visualization module presents the computed QoS metrics in a graphical form across the entire service provider's network and helps identify and isolate any performance anomalies or faults.

MPLS technology aims to bring together the best of connection-oriented (asynchronous transfer mode [ATM] and frame relay) and connectionless networking (classic IP networks). The main benefits of MPLS are the ability to provide bandwidth-guaranteed paths and efficient path restoration using mesh routing and shared backup paths. MPLS also simplifies the setup of virtual private networks (VPNs) in a service provider environment. Furthermore, MPLS supports multiservice (frame relay, ATM, IP/MPLS) networking. The momentum behind MPLS technology has now made it possible to build higher-speed switches for MPLS. OC-192 MPLS boxes have been in existence for some time, and OC-768 boxes are feasible and will likely become available in the next year or two. Thus, service providers have come to regard MPLS as the preferred solution for building the core of their networks.

DiffServ, on the other hand, is a framework for assigning traffic to different classes based on IP source address, destination addres, source port, destination port, or protocol type. Packets belonging to different classes are marked with different DSCPs. These markings are typically done at the ingress router/switch of a service provider's network. Based on the SLA for different classes of traffic, metering and policing is done at the ingress router or switch of a service provider's network to ensure that the traffic conforms to the SLA between the customer and the service provider or between two peer service providers. Packets not conforming to the SLA are either dropped or marked with lower priority so that they can be dropped in the event of congestion in the network. Packets belonging to different service levels are marked with different DSCPs and put into different queues. This mapping between the service-level parameters and the DSCPs ensures that the packets are treated according to their QoS requirements by the scheduler.

Network-based VPN is an excellent illustration of how a service provider can combine MPLS and DiffServ to provide a useful network service. Network-based VPNs enable a customer (typically an enterprise) to specify different classes of service. A customer's VPN can begin at the customer premises equipment (CPE) or at the edge of the service

Panel 1. Abbreviations, Acronyms, and Terms

API—application programming interface
AR—auto regressive
ASN—abstract syntax notation
ATM—asynchronous transfer mode
CORBA*—Common Object Request Broker Architecture
CPE—customer premises equipment
CVR—customer virtual router
DiffServ—differentiated services
DSCP—DiffServ control point
FIFO—first in, first out
GLR—generalized likelihood ratio
ICMP—Internet control message protocol
ID—identification
IP—Internet protocol
IS-IS—intermediate system to intermediate system
JDBC*—Java* Database Connectivity
JMS—Java Messaging Specification
LSP—label switched path
LSR—label switched router
MIB—management information base
MPLS—multiprotocol label switching
OC-12—optical carrier digital signal rate of 62.08 Mb/s in a SONET system
OC-192—optical carrier digital signal rate of 9.953 Gb/s in a SONET system
OC-768—optical carrier digital signal rate of 39.813 Gb/s in a TDM system
OID—object ID
OSPF—open shortest path first
PCA—principal component analysis
PDU—protocol data unit
QoS—quality of service
SLA—service-level agreement
SLS—service-level specification
SNMP—simple network management protocol
SONET—synchronous optical network
TDM—time division multiplexing
TEQUILA—Traffic Engineering for QUality of service in the Internet, at LArge scale
TE—traffic engineering
VoIP—voice over IP
VPN—virtual private network

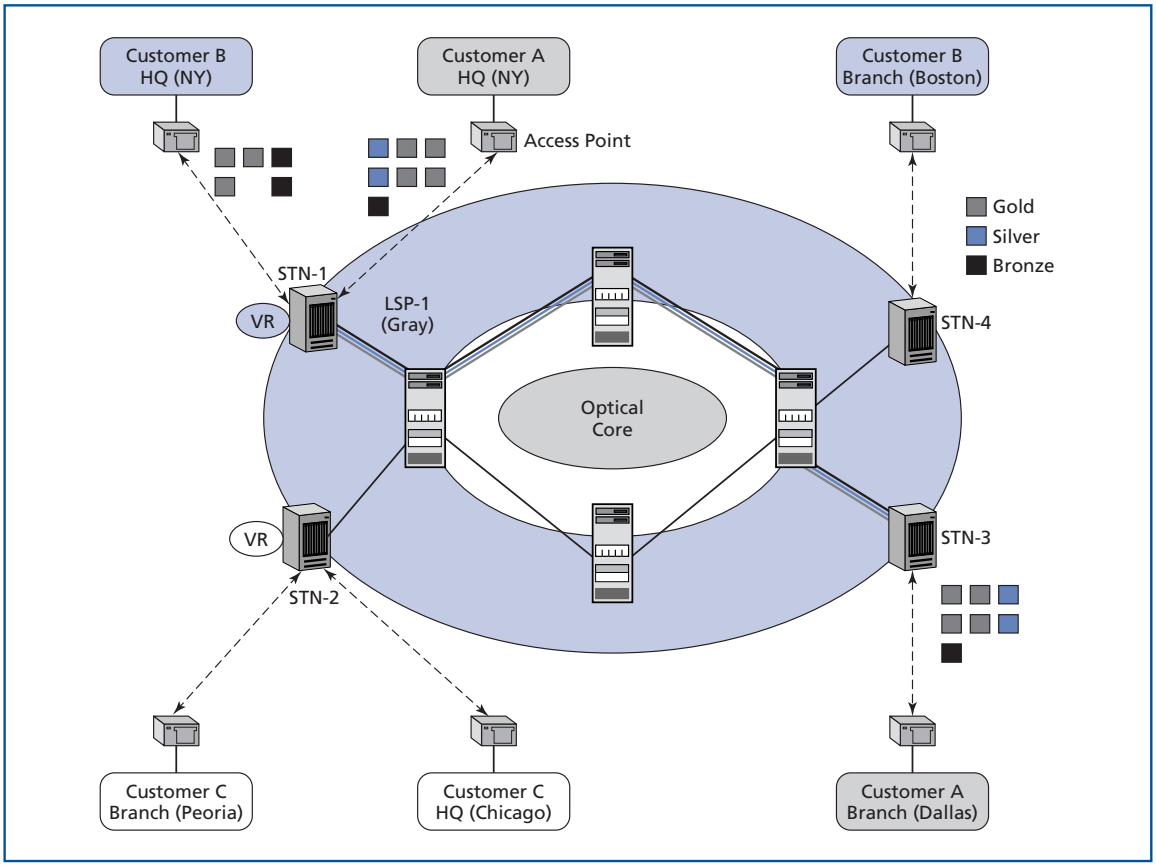


Figure 1.
VPN architecture for a service provider network.

provider's network. In the CPE case, access routers provide the DSCP marking for different traffic classes; in the latter case, the customer virtual router (CVR) located at the edge of the service provider's network provides the marking. **Figure 1** shows a VPN architecture for a service provider network.

In Figure 1, there are three VPN customers—A, B, and C. In particular, customer B generates two classes of traffic (gold and bronze), while customer A generates three classes of traffic (gold, silver, and bronze). Packets are marked at the access router and sent to the switch (STN-1) at the edge of the service provider's network. When DiffServ is combined with MPLS for VPNs, there are several possible ways of mapping DSCPs into MPLS label switched paths (LSPs). Here we show one possible mapping just to set the stage for the subsequent discussion on monitoring of MPLS networks. In the illustration, the focus is on two VPN customers A (gray color) and B (blue color). Both customers have two locations, which are interconnected using VPNs. Customer A generates three traffic classes (gold, silver and bronze) and Customer B generates two traffic classes (gold and bronze). In Figure 1, all three traffic classes of customer A are mapped into the same MPLS LSP at the label edge router (STN-1), and each intermediate MPLS label switched router (LSR) distinguishes among the different traffic classes using the EXP-bits in the MPLS shim header. This mapping is known as an E-LSP mapping [9]. Note that, using the E-LSP scheme, if the traffic from VPN Customer B were also mapped onto the same LSP, the problem of measuring QoS metrics on a per-customer

basis would be further complicated. L-LSP is another approach to combine DiffServ and MPLS. This approach permits the use of an MPLS/DiffServ combination even in the ATM environment. We believe that the SEQUIN monitoring system will extend to such networks as well.

Selection of Monitoring Metrics

In a service provider network, management and performance information can be retrieved using SNMP or other protocols such as Internet control message protocol (ICMP). In this work, we focus on information that can be obtained using SNMP as the method of retrieval. SNMP along with the related MIB information supports a large number of variables that describe and measure various aspects of network behavior. However, it is not possible to monitor all the available information due to the impact of monitoring overhead on network throughput. Therefore, a careful selection of the critical metrics that must be monitored is necessary.

Bandwidth. Typically the amount of bandwidth provisioned for customers is different from the amount the customers actually use. For example, a service provider can allocate bandwidth that is equal to the peak bandwidth requirement for its top-tier customers while allocating bandwidth that is equal to the average bandwidth requirement for its second-tier customers. For even lower-tier customers, the service provider can use an over-subscription model similar to the airline industry. As a result of such varied bandwidth provisioning schemes, the actual bandwidth usage is different from the provisioned bandwidth. The main benefit of bandwidth monitoring is that the knowledge of actual bandwidth usage would help the service providers to determine the amount by which they can afford to over-provision their network. For example, if the allocated bandwidth on an OC-12 (622 Mb/s) link is 400 Mb/s and the actual usage is 300 Mb/s, the service provider has the option to over-provision the network. Without the monitoring of bandwidth usage, this information would not be available and would therefore result in lost revenue for the service provider.

Delay. The importance of end-to-end delay from a customer's point of view will only increase as the customer starts to rely on MPLS-based networks for carrying real-time traffic. Voice-over-IP (VoIP), video-streaming, or video-conferencing applications require support from the service provider's network for bounded delay metrics. Continuous monitoring of delay for MPLS tunnels enables a service provider to ensure that the customer gets what he or she pays for. Monitoring of actual delay would help the service providers determine if they can admit new customers who have specific delay requirements. Without such monitoring, service providers would not be able to provision any additional customers or services with stringent delay requirements and would therefore lose revenue.

Packet loss. Applications such as real-time voice are sensitive to loss and, if a VPN customer wants to get a low-loss connection for transporting loss-sensitive traffic, it is important that the service provider provision its network accordingly. Furthermore, performance problems such as data bursts from different customers or the failure of a switch, router, or link in the network can be detected by monitoring packet loss, delay, and customer bandwidth violations on the MPLS tunnels. Without continuous monitoring, such problems may not be detected in real time. This will lead to lower customer satisfaction and violation of SLAs with customers, and it will also result in a loss of revenue for the service provider.

Relevant Issues for MPLS Monitoring

To implement MPLS monitoring, algorithms are required to determine how frequently the network elements should be polled, how many pollers are required, and how these pollers should be distributed. In addition, new algorithms are needed to synchronize the polled management information base (MIB) variables at different NEs and synthesize the end-to-end QoS. The current MPLS-TE MIB only supports variables that provide traffic accounts for the first hop of the LSP. To obtain a complete picture of network usage, it is necessary to go beyond this limitation of the MIB and provide performance data for all legs of a given LSP. Algorithms that can take several monitored

parameters and correlate them to predict or identify performance problems in the network are also required. SEQUIN addresses all these issues.

The QoS metrics of bandwidth, delay, and loss can be monitored at different levels of granularity:

- Bandwidth and loss per physical pipe,
- Bandwidth and loss per LSP,
- Bandwidth, delay, and loss per class of service, and
- Bandwidth, delay, and loss per class of service per VPN customer (assuming support for MPLS VPNs exists).

Note that, in Figure 1, we only show traffic from one VPN customer (the gray color customer, customer A) that has three classes of traffic (gold, silver, and bronze), and there is one MPLS tunnel from STN-1 to STN-3. It is possible to have multiple VPN customers (customers A and B) with different classes of traffic using the same LSPs. In such cases, it is necessary to monitor QoS metrics per class of traffic per VPN customer per LSP.

Related Work

Several generic monitoring tools are currently available on the market. Only those products that support QoS monitoring are mentioned here. Cisco's NetFlow provides the measurement base for Cisco's new Internet QoS initiatives. NetFlow technology efficiently provides the metering base for key applications such as network traffic accounting, usage-based network billing, network planning, network monitoring, outbound metering, and data mining capabilities for both service provider and enterprise customers. Most of the data collected by NetFlow is obtained using packet sampling techniques, and it is used for post-processing the data on a periodic basis. The SEQUIN monitoring tool analyzes the collected data in near real time. Lucent's Network Operations Software Group uses the VitalSuite® product for monitoring network performance. The current version of the VitalSuite product does not offer a full feature support for monitoring of MPLS and DiffServ. However, its architecture is amenable for the easy incorporation of MPLS support as developed in SEQUIN.

The main objective of the European research initiative called TEQUILA (for Traffic Engineering for QUality of service in the Internet, at LArge scale) [16] is to study, specify, implement, and validate service definition and traffic engineering tools for the Internet. In the context of service-level specification (SLS), the project addresses the problem of designing protocols and mechanisms for negotiating, monitoring, and enforcing SLSs. Primarily, TEQUILA addresses the specification of SLAs, while SEQUIN is a monitoring tool that can be used to do SLA conformance monitoring.

In this paper, we present software techniques, algorithms, and the architecture of our prototype SEQUIN online MPLS monitoring system.

SEQUIN Overview

SEQUIN is a software tool for online performance monitoring and analysis of IP/MPLS networks. It monitors traffic statistics at the edge and core devices and synthesizes the collected information to provide network-wide QoS status. The QoS status of the network, along with the health of the network, is displayed in near real time using multiple visual metaphors such as arcs, tables, and histograms. The SEQUIN architecture is defined with the goal of easy integration with different high-level applications such as provisioning and billing systems through the use of Common Object Request Broker Architecture (CORBA*) APIs.

The SEQUIN monitoring tool requires MPLS-TE MIB support on network devices in order to provide LSP discovery. It also uses the standard MIB II for obtaining performance data of the LSP and the physical network. SEQUIN uses SNMP to collect network device information. The primary motivation for using SNMP as the mechanism to monitor MPLS networks is that SNMP is widely deployed on all IP devices and supports a wide range of management-related information. In order to provide performance metrics at the class-of-service level, it is necessary for the network to have support for DiffServ. However, if DiffServ is not supported, SEQUIN is still capable of monitoring MPLS tunnel specific performance.

SEQUIN Architecture

The SEQUIN architecture is tailored primarily to provide QoS monitoring in a single service provider network. By limiting our scope to a single service provider environment, we can focus on the specific problem of QoS monitoring without the added complexity of inter-domain routing. In the context of a single service provider network, it is possible to provide reasonably good guarantees of QoS in a scalable manner since a synthesized view of the network conditions can be obtained. SEQUIN assumes that there is a basic routing protocol support in the form of either open shortest path first (OSPF) or intermediate system to intermediate system (IS-IS). This assumption is necessary so that route announcements will be made as new tunnels are set up. Currently, we only consider support for pre-provisioned explicit MPLS paths and a single DiffServ domain. A single DiffServ domain assumes that the DSCPs represent the same traffic class throughout the network. This is reasonable since in a single service provider network there is only one administrative domain and therefore only one mapping between the DiffServ code points and traffic classes.

The SEQUIN system performs monitoring, inference, synthesis, and presentation. These tasks need to be implemented at different time scales in the network. The object-oriented, distributed architecture of SEQUIN consists of six different modules working at the appropriate time scales to accomplish these tasks (see **Figure 2**):

- The *NetMon module* polls network devices both at the edge and the core of the network for selected MIB variable information. It collects MIB variable values and saves them in the QoS database.
- The *NetHealth module* monitors the health of the different network devices and detects anomalous conditions such as router overloads and congestion spots in the network. This module uses the MIB variables that are collected by the NetMon module.
- The *NodeQoS module* uses raw MIB variable information to compute current QoS metrics for the head end of the MPLS tunnel and for all the physical links of all the nodes.

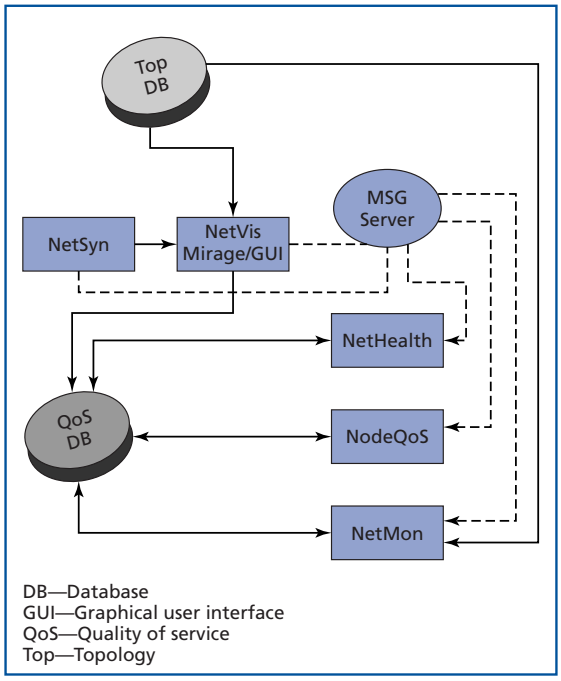


Figure 2.
The modular SEQUIN architecture.

- The *NetSyn module* synthesizes the node-level QoS metrics to provide an edge-to-edge view of network QoS. This is accomplished by correlating the QoS metrics of the different devices as computed by the NodeQoS module.
- The *NetVis module* provides a visual interface that displays the topology of the network and the current state of network QoS metrics.
- *Mirage*, a statistical plotting and online analysis package, provides multiple correlated views of the network measurements.

The SEQUIN monitoring tool is implemented using the Java* build environment, a Java Database Connectivity (JDBC*)-accessible relational database and a Java implementation of the Java Messaging Specification (JMS). Each of the major SEQUIN modules—NetMon, NetHealth, NodeQoS, NetSyn, NetVis, and Mirage—runs in a separate Java Virtual Machine (JVM) and the JVMs are dispersed across two Linux* boxes in our test network.

SEQUIN database table structure. While asynchronous messaging is used to signal events and communicate small amounts (on the order of 100 bytes) of information in the SEQUIN architecture, all significant quantities of information are communicated between modules via database access. Information stored in the database can be classified as either static or dynamic. Static information includes system configuration, network element configuration, network topology, and monitoring agent information. Dynamic information consists of polled SNMP data and information computed with this data.

The polled data corresponds to the values of MIB variables at the various network devices. It is kept in a set of tables that are cleared frequently so they can be held at a reasonable size. Tables that contain computed statistics as well as synthesized MPLS information such as tunnel-level QoS are cleared less frequently as this information is of more enduring interest.

The current SEQUIN implementation uses the SOLID [10] relational database; however, its JDBC interface allows any database to be used as long as a JDBC driver exists for it.

SEQUIN messaging service. SEQUIN modules communicate via the commercial SwiftMQ* JMS [12]. A publish/subscribe messaging model is used where the individual modules subscribe to select messages of interest (different types of data have been written to the database) and also publish messages once the module has completed an operation. When registering with the messaging system, each module provides an asynchronous callback routine for each message type to which the module has subscribed. Typically, when a module receives a notification through this callback, it retrieves data that has just been written to the database. For example, notifications are triggered (publish messages) by the NetMon when it writes collected MIB variable values to the database or when some other module such as NodeQoS has completed a computation of interest and written it to the database.

Network router/MPLS path discovery module. The path discovery module is a standalone Java code that does router discovery and writes the network topology to the database. As it discovers each new router, it uses SNMP to query the router for the existence of the MPLS-TE MIB [7]. If the TE-MIB exists, it then further queries the MPLS-TE MIB to extract MPLS tunnel path information and writes it to the database. The discovery code starts with one or more seed routers and uses filtering to exclude undesired addresses. It accesses the MIB variable *ipRouteNextHop* [8] at each router node to find all the neighboring nodes. It also retrieves all the router interface information such as status and speed by accessing the MIB II *ifTable* [8]. This data is saved to the database for use by clients interested in topology data and for use by other SEQUIN modules that need interface index number, interface speed, and interface type for their processing.

An MPLS tunnel is set up from the head-end router to the egress router, with one or more routers in between. The start or ingress router for the tunnel is referred to as the head-end router, and the last router in the tunnel path is the egress router. Although all the routers in the path have to be configured to handle MPLS, only the head-end router has the MPLS configuration visible in the MPLS-TE MIB (assuming the router's operating system supports it). During the discovery process, all but the last hop (node) in the path are obtained by reading the *mplsTunnelHopIpv4Addr* table in the TE-MIB [7]. The last hop is obtained by reading the *ipNetToMediaNetAddress* table in the *ip* group of MIB II. The last hop node is identified by matching the output interface address of the next to last hop node with the input interface address of the last node. Per-tunnel traffic statistics are maintained at the virtual interfaces of the router. Virtual interfaces act just like any other *ifIndex* [8], except that the tunnel statistics have been separated out from the hardware interface over which the tunnel flows. If the device does not support the MPLS-TE MIB but the LSP path information is stored elsewhere, it can be imported to the SEQUIN database and the system will continue to operate as expected.

NetMon: The SNMP Poller Module

The NetMon module retrieves the MIB variable values via SNMP and stores them in the database to be used by other SEQUIN modules. The NetMon module consists of two main functional areas that

(1) parse the user polling-parameter input file, and (2) perform the SNMP poll of MIB variables specified by the user at the indicated frequencies. Using the publish-subscribe model of the messaging service, NetMon informs the other modules that it has written MIB data to the database. It uses a commercial Java SNMP stack [17] to send SNMP v1 and v2c requests to the subject router nodes.

The polling parameters specification file contains keywords and values for specifying the router node to be polled (Node), MIB variable names to be retrieved by SNMP (Var), and the frequency at which the MIB variables are to be polled (Freq). The node values are specified in the form of Internet addresses (aaa.bbb.ccc.ddd), and the 'Var' values are specified in the form of the complete MIB tree path plus indexing structures (e.g., iso.org.dod.internet.experimental.mplsTeMIB.mplsTeObjects.mplsTunnelTable.mplsTunnel Entry.mplsTunnelName.1.0.0.2173154476). The Freq values are specified in multiples of 5 seconds, up to a limit of 1800 seconds (30 minutes). The minimum polling period is subject to the traffic and capacity characteristics of the network being polled. In our test-bed network, an interval of 15 seconds provided sufficient granularity. The polling parameters file is parsed and the information is stored in Java classes so that the poller can efficiently loop through its polling cycle.

All polling specifications must be in a form suitable for *SNMP Get* Requests. The requested MIB variables (object IDs [OIDs]) are packed into an *SNMP Get* request protocol data unit (PDU), and the PDU is saved internally in its binary byte form to avoid generation of a new PDU every time through the cycle. It is possible to efficiently pack OIDs into an SNMP request PDU and schedule the PDU for transmission to the destination router. However, since we have a relatively small number of router nodes in our test bed, the outgoing PDU requests are bunched together at the beginning of the interval and sent out sequentially. The code can be modified to disperse the PDUs over an interval across the polling period to reduce the burstiness of the SNMP traffic [1]. The SNMP poller loops through the tables created in the poll parameters parsing phase. SNMP Request PDUs are sent to each destination node at the appropriate frequency. The poller sleeps after a request is sent until it is time to send out the next request. After all SNMP responses corresponding to a request PDU are received, the return values are put on a first-in-first-out (FIFO) queue. The FIFO queue *read* thread is awoken, and it pulls the return values off the queue and calls a routine to save the data to the database. After all the results for a router are put onto the queue, the poller puts a special entry in the queue, which signals the FIFO-reading thread to publish a message to the JMS message server indicating that data for a particular router has been written to the data base for the current time slot. Other SEQUIN modules that have subscribed to this JMS message type will be notified of the database update, and they can proceed to use the collected MIB variable values for additional processing or calculations.

NodeQoS: Node-Level QoS Monitoring

Node-level QoS statistics are provided by the NodeQoS module. NodeQoS uses raw polled SNMP MIB data and then calculates interface-level utilization information. Bandwidth usage as well as average and peak is computed for all router interfaces at a configurable frequency using the following formulas:

$$BW_Usage = \frac{8 * \sum_{i=1}^{T/f} (\Delta_i ifIO + \Delta_i ifOO)}{T}$$

$$Avg_Util = 100 \times \frac{BW_Usage}{ifSpeed}$$

$$Peak_Util = 8 * \max_{i=1}^{T/f} (\Delta_i ifIO + \Delta_i ifOO),$$

where f is the polling frequency, and T is the computation interval. Δ is the difference in value over subsequent polling instances, and *ifIO*, *ifOO*, and *ifSpeed* are MIB variables found in the *ifTable* of MIB II. Information on utilization spikes and trends observed over time as well as dynamic algorithms to achieve an optimally load-balanced network can be made available to network administrators. The NodeQoS module, like most of the SEQUIN modules, can either work alone or in a distributed manner. This permits the universe of nodes that must be monitored to be partitioned for optimal computing and message-passing

characteristics. NodeQoS generates messages for downstream modules (NetSyn, NetVis, and Mirage) that provide graphical displays of NodeQoS computations. An issue under study is the computation of near real-time statistical averages based on tunnel setup-rate and holding times.

NetSyn: MPLS Tunnel-Level Monitoring

End-to-end MPLS tunnel-level statistics are generated by the NetSyn module. Currently available MIB data can only provide tunnel utilization and performance data for the first-hop (or head-end) router of a given tunnel. As shown in **Figure 3**, using tunnel topology data, NetSyn algorithms propagate head-end traffic data across all of the links in all the tunnels.

Tunnel bandwidth utilization, non-tunnel bandwidth utilization, and residual capacity are computed for all monitored links using the following formulas.

In the general case, we have:

$$AvgTunLoadLink_j = \sum_{i=1}^{N} t_{ij}$$

$$AvgNonTunLoadLink_j = l_j - \sum_{i=1}^{N} t_{ij}$$

$$RemainingCapacityLink_j = C_j - l_j$$

$$ConfigurableCapacity = C_j - \sum_{i=1}^{N} B_{ij},$$

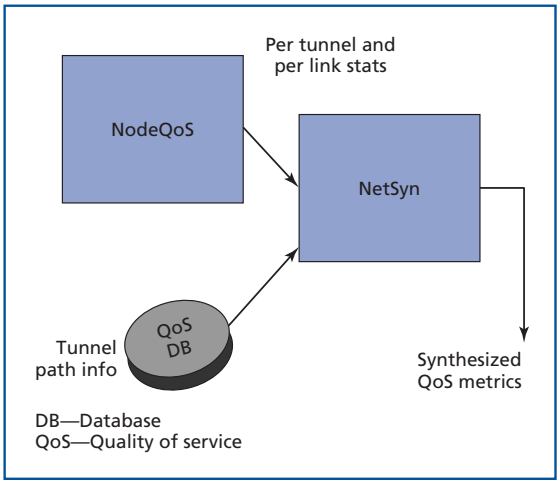


Figure 3.
Schema for NetSyn computations.

where t_{ij} is the average load for tunnel i on physical link j. The value of t_{ij} is obtained using an equation similar to that of bandwidth usage in the *NodeQos* module. Traffic counts associated with a given LSP appear as yet another interface on the router and therefore have all the traffic counter MIB variables, just as in the case of a physical interface. N is the total number of tunnels on physical link j; l_j is the bandwidth usage of physical link j; C_j is the physical capacity of link j; and B_{ij} is the configured bandwidth for tunnel i on link j.

In the case of head-end routers we have:

$$AvgNonTunLoadLink_j = l_j$$

$$RemainingCapacityLink_j = C_j - \sum_{i=1}^{N} t_{ij} - l_j.$$

Once the metrics for the first hop of an LSP are obtained, the LSP path discovery module can be used to reflect the computed information onto all the links on the path of that LSP.

Over time, usage patterns can be identified, and this data can be used as input to either automated or manual provisioning software to determine optimal paths for new tunnels based on complex provisioning and SLA requirements. NetSyn generates messages for NetVis and Mirage that provide graphical displays of NetSyn computations. Additionally, it is possible to create a historical record of computations via a data archiving option. Archival data can be used for replay as well as online simulation scenarios.

NetHealth: Network Health Indicator

The NetHealth module monitors the health of the different network devices and detects anomalous conditions such as router and MPLS tunnel overloads and congestion spots in the network. This module uses the interface and *ip* group MIB variables that are collected by the NetMon module. It implements online learning methods and provides probabilistic indicators of the current health of a given node along with some fault characterization.

A diagrammatic representation of the three-stage NetHealth algorithm is shown in **Figure 4**. The first stage is to detect abnormal changes for each MIB variable. The increments in the MIB variables constitute

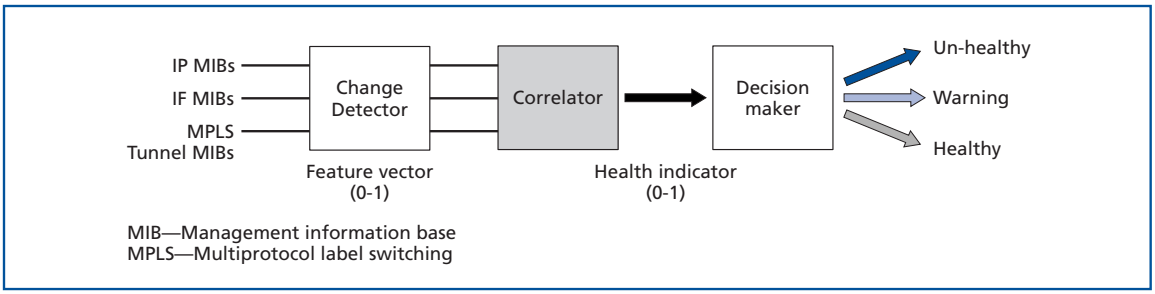


Figure 4.
Schematic diagram of the NetHealth algorithm.

a time series. A piecewise stationary auto-regressive (AR) model is used to describe the non-stationary stochastic time series [2]. In order to detect the subtle changes in the MIB variables that precede a fault, a sequential hypothesis test was performed using the generalized likelihood ratio (GLR) test [14]. Using this test, a time-correlated indicator of the abnormality level is obtained for each of the variables. The indicator takes values between [0, 1]—1 corresponds to abnormal, and 0 corresponds to normal. The variable level indicators of the correlated MIB variables form a feature vector $\vec{\psi}(t)$.

The second stage, called the correlator, incorporates the spatial correlation into the feature vector to compute a health indicator for the network node or MPLS tunnel. A correlator matrix A is designed to represent the interaction among the components of the feature vector. The health indicator is obtained as shown in Eq. (1).

$$f[\vec{\psi}(t)] = \vec{\psi}(t) A \vec{\psi}(t)' \qquad (1)$$

The health indicator is a single value, also bounded between [0, 1]—0 implies healthy, and 1 implies unhealthy. Values in between correspond to increasing abnormality levels. The decision of healthy versus unhealthy is made based on (2),

$$t_a = \inf\{t : \lambda_{f\min} \leq f[\vec{\psi}(t)] \leq \lambda_{f\max}\}, \qquad (2)$$

where $\lambda_{f\min}$ and $\lambda_{f\max}$ are the eigenvalues of the correlator matrix A [15]. In the final stage of the algorithm, the health indicator is fed into the decision maker to determine the network element status as healthy, warning, or unhealthy.

When detecting router abnormalities, the feature vector includes three *ip* group MIB variables—*ipInReceives*, *ipOutRequests*, and *ipInDelivers*. For detecting interface abnormalities, the feature vector uses two interface group variables—*ifInOctets* and *ifOutOctets*. In both cases, the MIB variables are collected from the same router. Detection of MPLS tunnel abnormalities is different from the last two cases, since the MPLS tunnel is unidirectional and the tunnel MIB variables can be obtained only at the head-end router. To provide a tunnel health indicator, the required MIB variables must be collected from the heads of both the forwarding and the reverse tunnels. Therefore, it is necessary to synchronize the polling results from both tunnel head ends. In the current implementation, we use a logical time stamp to synchronize the polling results by attaching the same time stamp to the queries for both routers and sending the queries out simultaneously. Since the monitoring frequency is usually greater than or equal to 15 seconds, it is reasonable to omit network delay (order of ms) and query processing delay (experimentally found to be 20 ms) on the routers. We consider the two polling results with the same logical time stamp to represent a snapshot of the bidirectional tunnel traffic. The MIB variable *ifOutOctets* is used to detect a tunnel health problem.

The fault detection capability is improved by adding some signature-based root cause analysis features. By observing traffic information pertaining to an hour prior to the declaration of failure, we can identify different groups of failures by analyzing their characteristic traffic patterns [13].

NetVis: Visualization Interface

The SEQUIN visualization interface permits the network administrator to work with the visual representation of tunnels, links, and nodes. It is characterized by both its hierarchical nature and its extensive filtering capabilities. The administrator generally begins with the tunnel abstraction. It is possible to filter and search for tunnels based on static information such as the customer ownership and the location connections or various dynamically changing criteria such as effective throughput and utilization. These selected tunnels appear as large arcs between the source and destination locations. By clicking on an arc, information on any tunnel that traverses the arc can be viewed in a tabular format. Selecting a particular tunnel (e.g., row) from the table causes the physical links for the tunnel path to become highlighted. Dynamic color-coding of router nodes and the links that interconnect them give visual insight to the health of the tunnel path. It is also possible to select individual links and view link-specific statistics in a tabular format. The visual interface reads network topology information from external database tables that have been populated through a topology discovery process. A screen shot of our visualization package with the tunnels mapped onto a geographical layout is shown in **Figure 5**. The blue arcs represent the tunnels, and the colors on the routers and interfaces represent the health states (colors are adjusted for this publication). The pop-up table shows the relevant metrics for the selected tunnels.

Online Data Analysis

NetSyn interacts with Mirage [5], a software package for graphical data analysis. Mirage supports direct graphical querying via user interactions, as well as highlighting and dynamic updating. In SEQUIN, these capabilities are used to show multiple correlated views that dynamically track traffic characteristics or display a snapshot of the network load at a particular instance.

In Mirage, a data set is a matrix with entries represented by rows and attributes represented by

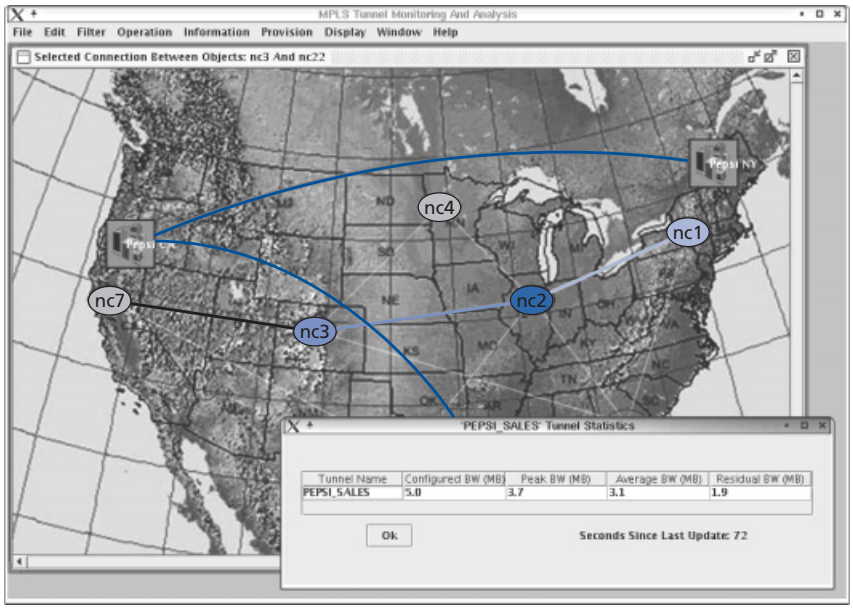


Figure 5.
NetVis display of computed metrics.

columns. Multiple views of SEQUIN data can be obtained. The basic views include:

- A table view of the data matrix, with an optional color tag attached to each row.
- A histogram plot that can be reconfigured to show a one-dimensional projection of the data set to any single attribute with frequencies in a configurable number of bins.
- A scatter plot that displays a two-dimensional projection of the data, where the X and Y axes can be chosen to be any of the attributes via list boxes. Regions in the projection plane can be selected by highlighting with a mouse.
- A feature vector plot (also known as a plot of parallel coordinates) [6] that plots the value of every attribute against the index of that attribute in a chosen group. Data can be selected and broadcasted from this plot by drawing intervals of values in each attribute and composing unions or intersections of such intervals.
- A time series plot that shows the value of a single attribute or values of several related attributes at each entry, assuming that each entry represents a time step with fixed increments.

For example, in a histogram, during the latest polling period we can show how many interfaces experienced an average utilization of 20 to 30%, how many experienced 40 to 50%, and so forth. Any outliers, say, interfaces that have extremely low or high utilization, will be immediately apparent. A user can select the corresponding bars in the histogram and display the selection as a table from which the router IDs and interface IDs can be read. The selection can also be sent to a NetVis display for highlighting on the network map. Updates of the NetSyn variables are appended to another Mirage data set history that stores a historical trace of each variable up to a certain look-back limit and can be displayed as a time series. In **Figure 6**, a screen shot shows Mirage display capabilities (colors are adjusted for this publication).

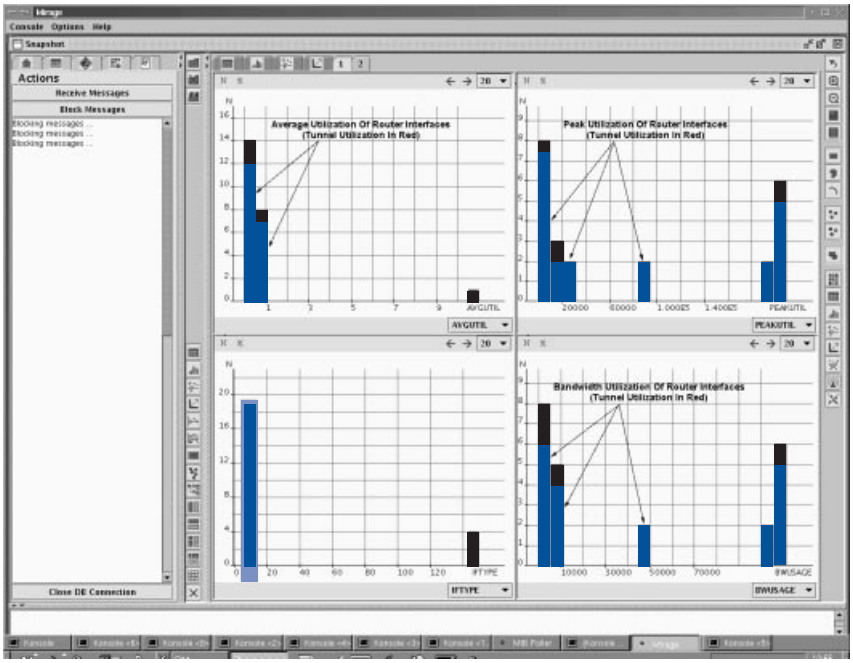


Figure 6.
Mirage display of computed metrics.

Research Results

The research problems addressed by the SEQUIN project range from specific issues related to SNMP-based monitoring to the general problem of network measurements and the infrastructure necessary for scalable monitoring systems. In this section, we provide a brief summary of the key results.

Choice of Variables

Using SNMP MIB for monitoring, network managers have access to a large number of MIB variables that describe the functionality of the network. However, to design a scalable monitoring system, it is essential to restrict frequent polling to a few variables. Therefore, a rigorous mechanism is required to identify a basic set of MIB variables. One technique for reducing the number of variables is principal component analysis (PCA) [3]. Using PCA, we can reduce the number of variables to be polled by choosing those variables that contribute to the highest variability of the data.

Given a set of variables, PCA computes a new set of variables that are linear combinations of the original ones and orders them by the amount of variance in the data set that is explained by these new variables. In our case, the original variables are the MIB variables. The weight of each of the original (i.e., MIB) variables on each particular component (i.e., the newly defined variables) describes the contribution of that variable to the component. Thus, a minimum set of MIB variables can be found by selecting the ones that have the highest weights on the most important components (i.e., those explaining most of the variance). In the *if* group of variables, of the 9 components obtained, components 1 and 2 together, composed of the variables interfaceInOctets, interfaceInUnicastPkts, interfaceOutErrors, interfaceOutOctets, and interfaceOutUnicastPkts, accounted for about 40% variance in the data. Thus, these variables were selected. In the MIB, MPLS tunnel information is represented as yet another interface; therefore, studying the interface group provides the relevant variables to be polled for the MPLS tunnels as well. This choice of MIB variables significantly reduces (by 50%) the bandwidth overhead while still ensuring good conformance monitoring for MPLS traffic profiles.

SNMP PDU Packing Strategies

The efficiency of SNMP-based monitoring systems can be improved by optimizing the use of SNMP PDUs. By optimally packing the SNMP PDUs we can reduce the bandwidth overhead due to network measurements. This is an important step for SNMP monitoring in MPLS networks. In the MPLS-TE MIB [7], the new variables being defined require an increasing number of bytes to specify them in the SNMP request packet, thus increasing the overhead associated with monitoring. Optimal packaging of an SNMP PDU request is dependent on the response values from the query. In determining the response PDU size, the response packet overhead bytes for encoding and the OID size are known since they are user specified. The only unknown is the OID return object value length required for packing it into the response PDU. **Figure 7** shows the maximum number of OIDs that can be packed into a single PDU of a particular abstract syntax notation (ASN) [11]) return type. The number of OIDs that can be packed into a single SNMP PDU also depends on the size of the OID. **Figure 8** shows that the number of OIDs that can be packed into a single SNMP PDU drops as the OID size increases.

Of all the MIB variable types, the most common are ASN type Integer, Integer32, and Integer64. The SNMP agent packs the return value using as few bytes as possible to encode the return value. Therefore, it is difficult to estimate the exact amount of space that is required to pack the response value unless one has some knowledge of the real value to be returned. **Figure 9** shows the impact of the response value size on the number of OIDs that can be packed into a single SNMP PDU.

Therefore, as one builds a request PDU, one can assume a response value length equal to the maximum ASN value type, even though less space may actually be used (e.g., Integer64 = 8 bytes, Integer 32 = 4 bytes). As OIDs to be packed into an SNMP Request packet are accumulated, one must also account for the ASN response value size corresponding to the associated OID and keep the total accumulated size below 1300 bytes (the maximum size allowed by the SNMP agent).

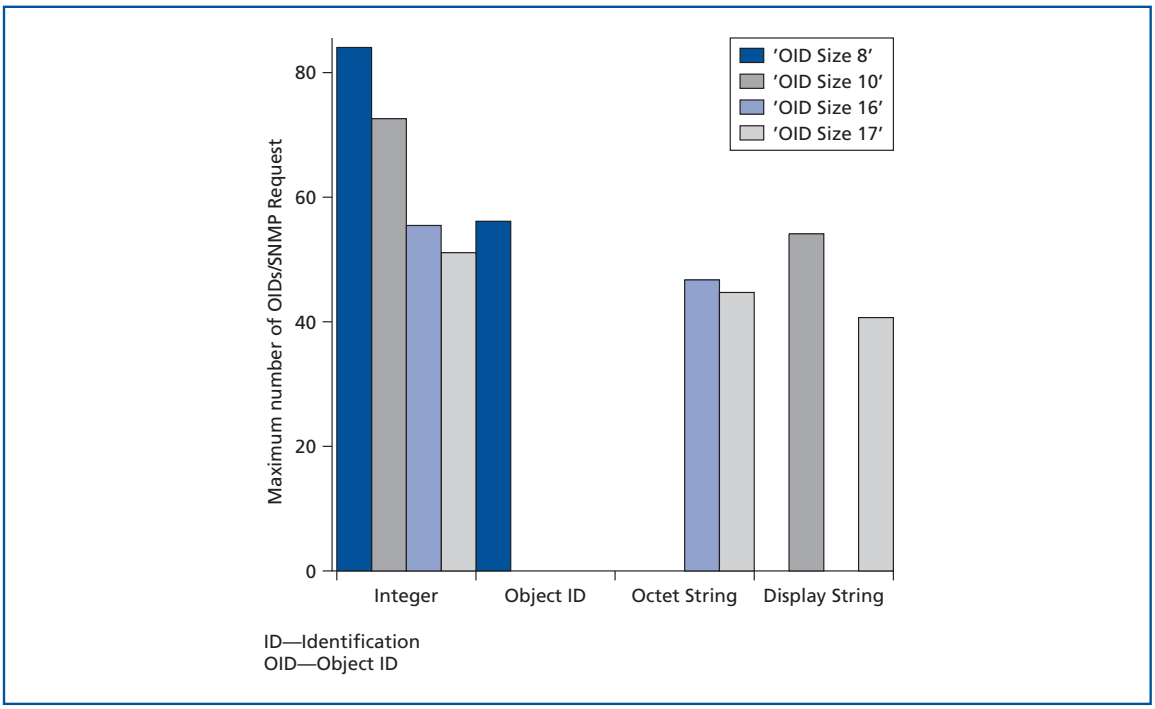


Figure 7.
Maximum number of OIDs that can be packed into a single SNMP PDU for MIB variables of different ASN types.

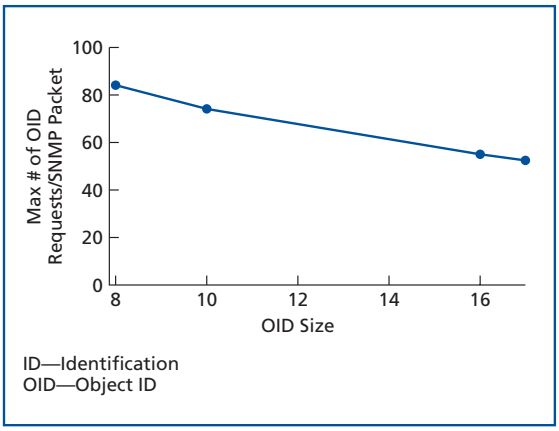


Figure 8.
Maximum number of OIDs that can be packed into a single SNMP PDU as a function of OID size, for ASN-type integer.

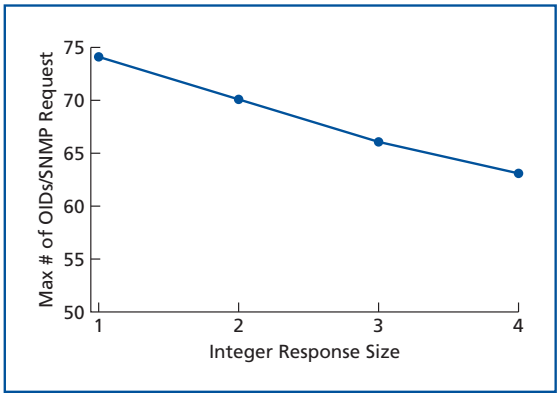


Figure 9.
Maximum number of OIDs that can be packed into a single SNMP PDU as a function of the response value size.

Distributed Poller Location

Typically, in network measurement systems, all nodes are monitored from a single point. Such a centralized scheme does not scale for large service provider networks. With a centralized polling scheme, poll responses have to be forwarded to a central location in the network, thus creating a potential for bandwidth bottle-neck on selected links that are close

to the central manager. Using distributed measurement systems [4] improves scalability by reducing the overall bandwidth overhead. In a distributed measurement system, the polling load is shared among multiple pollers located at different points in the network. However, using distributed pollers increases the cost of network management. Therefore, a good measurement infrastructure must minimize the cost associated with distributed polling. The cost is due to both the deployment of the instrumentation and the bandwidth consumed by the measurement traffic.

Based on simulation studies, we found that the total number of pollers for a given network topology can be reduced by using a MaxPollee assignment scheme. The MaxPollee assignment scheme aggressively chooses the next poller with the intent of assigning all of the remaining nodes to the new poller. However, the reduction in the number of pollers comes at the cost of the bandwidth consumed. We observed that, by increasing the capacity reserved for polling, we can always reduce the number of pollers regardless of the heuristics used. For a given service provider topology, we were able to identify a poller set and a corresponding assignment scheme without violating any bandwidth constraints (only in 40% of the cases was it necessary to use more than 90% of the allocated bandwidth for polling). The distributed poller location scheme can be used in the design phase of a network management system. For a network of about 200 nodes, we can obtain the poller set and the assignment scheme in just a few minutes. Thus, with little effort the network operations center can identify the placement of their measurement instrumentation.

Software Optimizations

For small networks, our current model performs with acceptable latency. However, for larger networks, optimizations are desirable. In the SEQUIN architectural model, there are two time intervals—the polling interval and the computation interval. The computation interval is the interval over which the relevant QoS metrics are computed. It is a processing requirement that all raw polled data be written to the database within the polling interval. Likewise, it is necessary that all computations derived from the raw data be carried out within the computation interval. If the computations exceed this processing window on a regular basis, then any buffers employed to mitigate the effect of traffic bursts will quickly overflow.

System-wide optimization can be achieved by configuring the frequency of messages passed between processes. For example, a single message can be sent to indicate that the data for multiple interfaces has been polled or processed. This step minimizes the amount of messaging traffic, as well as reduces the number of database operations. Since database operations have a high overhead, these optimizations have a significant impact on processing time requirements. Dynamic adjustment of the polling interval and the prioritization of polled data can also improve system operation. Based on trend data, nodes and interfaces that have been identified as risky can be given a higher priority during periods of high bandwidth usage and resource consumption. Another optimization scheme is the additional parallelization of intermodule processing. It is possible for upstream modules to forward certain intermediate results to downstream modules, thus allowing the downstream modules to get a head start on processing. This reduces the overall processing time.

Lastly, like distributed pollers, the use of distributed databases to store and forward polling results can minimize processing latency. Of course, when using multiple databases in peering relationships, system state maintenance and consistency are well-known problems. Because of these consistency issues, it is still desirable to have a master database that regularly receives the intermediate results (we may only send change data) from a network of local databases.

Conclusion and Future Work

In this paper, we have presented the design of an MPLS monitoring system. We have implemented the necessary modules and developed the required algorithms for monitoring MPLS tunnels in service provider networks. The SEQUIN system has been successfully implemented on a test-bed network. Currently, the system is being used to monitor bandwidth metrics. There is ongoing work for developing

algorithms to measure delay and loss. We are investigating novel scheduling strategies with regard to a distributed polling system and the capability of our algorithms for fault modeling and prediction.

Acknowledgments

The authors are grateful to Krishan Sabnani for his continued support of this project as well as Rick Buskens for valuable suggestions during the initial development of this project. We also thank Li Li who helped with the simulations on the distributed poller algorithms.

*Trademarks

CORBA is a registered trademark of Object Management Group, Inc.

Java and JDBC are trademarks of Sun Microsystems, Inc.

Linux is a trademark of Linus Torvalds.

SwiftMQ is a registered trademark of IIT GmbH.

References

[1] M. Cheikhrouhou and J. Labetoulle, "An Efficient Polling Layer for SNMP," Proc. IEEE/IFIP Network Operations and Management Symposium (Honolulu, HI, 2000), pp. 477–490.

[2] P. V. Desouza, "Statistical Tests and Distance Measures for LPC Coefficients," IEEE Trans. Acoustics, Speech and Signal Processing, 25:6 (1977), 554–559.

[3] R. O. Duda and P. E. Hart, Pattern Classification and Scene Analysis, John Wiley, 1973.

[4] G. Goldschmidt and Y. Yemini, "Delegated Agents for Network Management," IEEE Commun. Mag., 36:3 (1998), 66–70.

[5] Tin Kam Ho, "Exploratory Analysis of Point Proximity in Subspaces," Proc. 16th Internat. Conf. on Pattern Recognition (Quebec City, Can., 2002).

[6] A. Inselberg and B. Dimsdale, "Multidimensional Lines I: Representation," "Multidimensional Lines II: Proximity and Applications," SIAM J. of Applied Mathematics, 54:2 (1994), 559–596.

[7] Network Working Group, "Multiprotocol Label Switching (MPLS) Traffic Engineering Management Information Base," IETF draft-ietf-mpls-te-mib-08.txt, Jan. 2002, <http://www.ietf.org/ID.html>.

[8] Network Working Group, "Management Information Base for Network Management of TCP/IP-based Internets:MIB-II," IETF, RFC 1213, Mar. 1991, <http://www.ietf.org/rfc/rfc1213.txt>.

[9] Network Working Group, "Multiprotocol Label Switching (MPLS) Support of Differentiated Services," IETF RFC 3270, May 2002, <http://www.ietf.org/rfc/rfc3270.txt?number=3270>.

[10] Solidtech, SOLID Relational Database, <http://www.solidtech.com>.

[11] W. Stallings, SNMP, SNMPv2, SNMPv3, and RMON I and II, 3rd ed. Addison-Wesley, Boston, 1998.

[12] SwiftMQ, Java Messaging Service, <http://www.swiftmq.com>.

[13] M. Thottan and C. Ji, "Network Fault Classification Using MIB Variables," Proc. 7th IEEE/IFIP Integrated Network Management (Seattle, WA, 2001), pp. 468–482.

[14] M. Thottan and C. Ji, "Statistical Detection of Enterprise Network Problems," J. of Network and Systems Management, 7:1 (1999).

[15] M. Thottan and C. Ji, "Fault Prediction at the Network Layer Using Intelligent Agents," Proc. 6th IEEE/IFIP Integrated Network Management (Boston, MA, 1999), pp. 745–760.

[16] P. Trimintzios, I. Andrikopoulos, G. Pavlou, P. Flegkas, D. Griffin, P. Georgatsos, D. Goderis, Y. T'Joens, L. Georgiadis, C. Jacquenet, and R. Egan, Thales, "A Management and Control Architecture for Providing IP Differentiated Services in MPLS-Based Networks," IEEE Commun. Mag., 39:5 (2001), 80–88.

[17] Westhawk Ltd., freeware Java SNMP stack, <http://www.westhawk.nl>.

(Manuscript approved February 2003)

MARINA K.THOTTAN is a member of technical staff in the Networking Software Research Center at Bell Labs in Holmdel, New Jersey. Currently, she is working in the area of IP and wireless network management. She holds B.S. and M.S. degrees in physics from Bharathiar University in India. She also holds M.S. and Ph.D. degrees in biomedical engineering and electrical engineering, respectively, from Rensselaer Polytechnic Institute in Troy, New York. Dr. Thottan's research interests are in network measurements, IP QoS, time series analysis, and signal detection theory. She is a member of IEEE.



GEORGE K. SWANSON is a member of technical staff



in the Wireless Networking Research Department at Bell Labs in Holmdel, New Jersey. His current job responsibilities include the development of SEQUIN code, investigating research issues in SNMP-based monitoring, and interactions with several Lucent business units interested in using SEQUIN. His research interests are in network management, especially in the areas of SNMP measurements, Java applications for management, and MPLS networks. He received a B.S. in computer science from Fairleigh Dickinson University in Rutherford, New Jersey.

MICHAEL CANTONE was formerly a member of



technical staff in the Networking Software Research Department in the Data Networking Research Center at Bell Labs in Holmdel, New Jersey. His interests include visualization software, information security as it relates to electronic commerce and systems architectures. He holds a B.S. in computer science from Saint John's University in New York City and an M.S. in computer engineering from the University of Pennsylvania in Philadelphia.

TIN KAM HO is a member of technical staff in the



Computing Sciences Research Center at Bell Labs in Murray Hill, New Jersey. She holds a B.B.A in management and applied mathematics from the Chinese University of Hong Kong, an M.S. in systems science from Louisiana State University at Baton Rouge, and a Ph.D. in computer science from SUNY at Buffalo. Her interests are in pattern recognition, data mining, and computational modeling and simulation. She applies computational data analysis to many areas including network management, network device design, and signal processing. She is actively engaged in modeling and analysis of optical line systems via the FROG (Fast Raman Optimized Gain) project. Dr. Ho was formerly an associate editor of IEEE Transactions on Pattern Analysis and Machine Intelligence. *She is currently an associate editor of the* Journal Pattern Recognition. *In 1999, she received the ICDAR Young Scientist Award for her contributions to document image analysis and recognition. She has six U.S. patents and several applications pending.*

JENNIFER REN is a member of technical staff in the



Network and Services Management Research Department at Bell Labs in Holmdel, New Jersey. She conducts research in the design and implementation of a software framework for building reliable and configurable distributed applications. She received a B.S. degree in electrical engineering from Tsinghua University in China, an M.S. degree in electrical engineering from the University of Virginia in Charlottesville, and a Ph.D. degree in computer engineering from the University of Illinois at Urbana-Champaign. Dr. Ren's research interests include distributed systems, IP network and service management, fault-tolerance middleware, and mathematical modeling. She is a member of IEEE.

SANJOY PAUL is the director of Networking Software



Research at Bell Labs in Holmdel, New Jersey, where he leads R&D efforts in wireless networking, MPLS network management, and next-generation IP services. He is also a director in the Forward Looking Group in Lucent's Mobility Solutions Business Unit, where he is responsible for evaluating new technologies for strategic partnerships and setting directions in forward-looking networking research in 3G/4G technologies. He has many years of technology expertise, specifically in the areas of multicasting, media streaming, intelligent caching, mobile networking, and secure commerce. He was formerly recognized as a distinguished member of technical staff in Bell Labs Research, and he is the author of a book on multicasting (Multicasting on the Internet and Its Applications) *and numerous technical papers. He received the 1997 William R. Bennett award from IEEE Communications Society for the best original paper published in* IEEE/ACM Transactions on Networking. *Dr. Paul is an editor of* IEEE/ACM Transactions on Networking *and was a guest editor of* IEEE Network Magazine. *A frequent speaker at conferences and seminars worldwide, he holds a B. Tech. degree in electronics and telecommunications engineering from the Indian Institute of Technology in Kharagpur and M.S and Ph.D. degrees in electrical engineering from the University of Maryland in College Park. Dr. Paul is an adjunct faculty member in the Computer Science Department at Rutgers University, a senior member of IEEE, and a voting member of ACM.* ♦