Using Real-Time Measurements in Support of Real-Time Network Management

James L. Alberi, Ta Chen, Sumit Khurana, Allen Mcintosh, Marc Pucci and Ravichander Vaidyanathan

Abstract **–Increased reliability is necessary if the Internet is to carry information such as voice, video and other enhanced services. Congestion in the network because of the statistical nature of packet forwarding is a serious issue that could impede achieving the reliable, timely delivery of data and quality of service. At present humans monitor the network for congestion with a variety of data collection mechanisms and take corrective actions on an *ad-hoc* basis. Putting humans in the control loop yields corrective actions that are too slow because of delays in collecting data and too error prone because of the complexity of the network.**

This paper presents *Rondo*, an automated control system that manages congestion in core networks in near real time. It discusses the architecture and design of the *Rondo* system with emphasis on the rerouting engine and data-collection subsystem.

***Rondo* relies heavily on MPLS (Multi Protocol Label Switching), a relatively new technology that is intended to provide more efficient control over network routing than the destination-based routing used in the Internet of today. We recognized that MPLS could be used to alter traffic routes dynamically in response to measured or anticipated loads even though its typical application is in a more static environment.**

***Index terms* – automated network control, network management, network performance measurement**

I. Introduction

IP networks are moving into application areas that were formerly reserved to telephone networks by melding voice, video and data. These more capable networks have been termed Next Generation Networks (NGN) in the telephony community. The management of network traffic has been explored in detail in the telephony domain for many years. While this knowledge has some general applicability to the management of IP networks, these networks are, by the very nature of their capabilities, more difficult to control. Individual user traffic carried in an NGN network does not travel over a dedicated pipe of guaranteed bandwidth like a TDM trunk. Instead, traffic of differing characteristics often mingles together along routes that can change with time. Since the transport layer is not reserved but shared, competing flows can consume all available bandwidth across a link. The opposite condition holds at other times, where certain routes become extremely underutilized. While the gain of statistical multiplexing helps to make IP networks more economical than TDM-based networks, such imbalances make IP networks highly inefficient to operate and incapable of providing any quality-of-service guarantees. Time-of-day variations and link outages compound and magnify these imbalances.

IP packets are routed through the Internet using destination-based routing, which typically finds the shortest path through a network. With multiple sources and destinations of traffic, independent paths typically overlap on certain common links, a condition that can lead to traffic overloads, congestion, loss and excessive delay, while other links remain underutilized. Balancing the demands of network traffic across all network paths increases the efficiency of the overall network.

The network planning function typically constructs a set of routes through the network to accommodate expected traffic demands. These plans are relatively static in nature, and are meant to deal with long-term trends in traffic. While reevaluation of planning information using the latest load data is possible, frequent large-scale adjustments to the overall route structure of a network are too disruptive and time consuming. However, if the basic routes are efficient, then the problem of network balancing reduces to managing anomalous conditions that arise from spot load changes and link outages.

Responding to changing network events in near real-time requires a sophisticated monitoring and adjustment process that manipulates traffic flows while preventing the system from becoming unstable. Our goal is to detect and correct an imbalance in time scales approximately 30 seconds to 1 minute. This range strikes a balance between reacting to short-lived events while providing rapid-enough corrections to network flows before service level agreements are violated.

A. Network Management Concerns

IP traffic is often described as self-managing. On an individual flow basis, IP traffic under TCP or similar protocols adjusts to the characteristics of the path between its source and destination. This type of traffic adapts to adverse network conditions (triggered, for example, by packet loss) by reducing the effective transmission rate to a point where

The authors are with Applied Research at Telcordia Technologies, Inc., 445 South Street, Morristown NJ 07960.
(Email: jla@research.telcordia.com).

significant losses do not occur. Similarly, aggregate traffic flows size themselves to the bandwidth capability of the smallest link. While this mechanism throttles flows back to the point where they operate as best they can given current network conditions, it does not balance the higher level demands of managing overall link utilization in the face of multiple network paths. In other words, while flows adjust to the best their local environment has to offer, they are adversely affected by the inefficiency caused by imbalances in their global environment.

A further complication occurs with non-adaptive traffic, such as UDP flows, which do not adjust to network conditions. Under congestion, this type of traffic exhibits loss that can disrupt services like voice-over-IP. Unconstrained non-adaptive traffic also squeezes out adaptive traffic, causing the latter to reduce its effective throughput while UDP traffic continues to dominate transmission[6]. By segregating and managing different classes of service, the network avoids these and other conditions of service degradation.

B. The Trouble with Merely Adding Network Capacity

Internet capacity is rapidly increasing to keep up with growing demands from users. Information from carriers indicates that to accommodate a doubling of capacity at the network-access edge requires that the core expand by a factor of 8 to 10. Placing this capacity exactly where it is needed is a formidable task, as network sources and sinks vary, with not only time of day, but also the advent of new Internet services.

Systems like Napster alter dramatically the load patterns in a network by changing a large number of traffic sinks into a distributed set of traffic sources. Internet sites wax and wane in popularity, causing shifts in overall network demands. Carriers report that loads far from link saturation can adversely affect performance. The chaotic behavior of traffic implies that even extremely lightly loaded links (less than 3% utilization) exhibit loss. Having exactly the right amount of capacity in the right location is difficult in a changing environment. We contend that by adding a layer of global management, we can better respond to and control large-scale networks

C. The Role of Multi-Protocol Label Switching

MPLS[16] offers Traffic Engineering, which provides efficient control over the paths that packets take as they traverse a network[2]. The ability to control these paths is at the heart of *Rondo's* ability to manage network congestion. An optimal set of logical routes or LSPs (Label Switched Paths) through the network leads to efficient utilization if the traffic flows are assigned carefully to the LSPs.* Network administrators may construct multiple paths to the same destination, thereby overcoming a significant shortcoming of conventional destination-based IP routing. Adaptive and non-adaptive traffic may be separated to insure proper quality of service. Bandwidth may be explicitly allocated to meet any service-level agreements in place between the network provider and the uses.

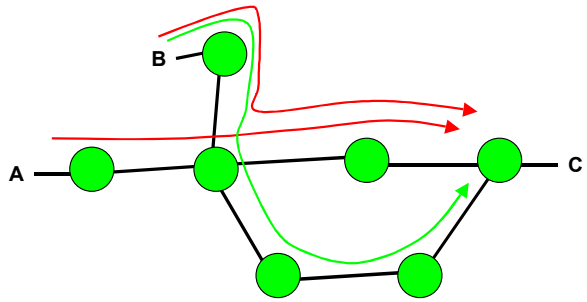

Figure 1. A typical scenario for congestion. Traffic from A to C and from B to C will typically follow the same path causing congestion in the common links. With MPLS, traffic from B to C can be rerouted along the lower routers, eliminating the overused links.

A secondary benefit of MPLS is increased forwarding efficiency. Packets are assigned to FECs (Forwarding Equivalence Classes) and enter into LSPs at the ingress router. Once assigned to an LSP, intermediate routers examine only a minimal header to determine the next hop for the packet. This scheme substantially reduces the amount of processing that occurs at intermediate hops although recent strides in gigabit and terabit IP-routing processors alleviate the concern over processing time.

Figure 1 shows a typical scenario that arises in *Rondo* when using destination routing based on finding the shortest path. Traffic from nodes A to C and from nodes B to C flows along a common set of network segments. With explicit routing through MPLS tunnels, the data from node B to C can be rerouted to a longer but more lightly congested path. The ability to monitor the global state of the network coupled with the fine control afforded by MPLS makes congestion control possible in *Rondo*.

II. THE RONDO ARCHITECTURE

Rondo uses a feedback loop to govern the behavior of traffic in the network core. It manages the flows that originate and terminate between various PoPs (Points of Presence) in the network by directing these flows into the multiple pathways that are created using MPLS Label Switched Paths. These LSPs serve as conduits through the network that are unaffected by the local optimization strategy of shortest path routing. Rather, *Rondo* optimizes performance based on global traffic considerations in the network.

* We use the term LSP and MPLS tunnel interchangeably in the remainder of this paper.

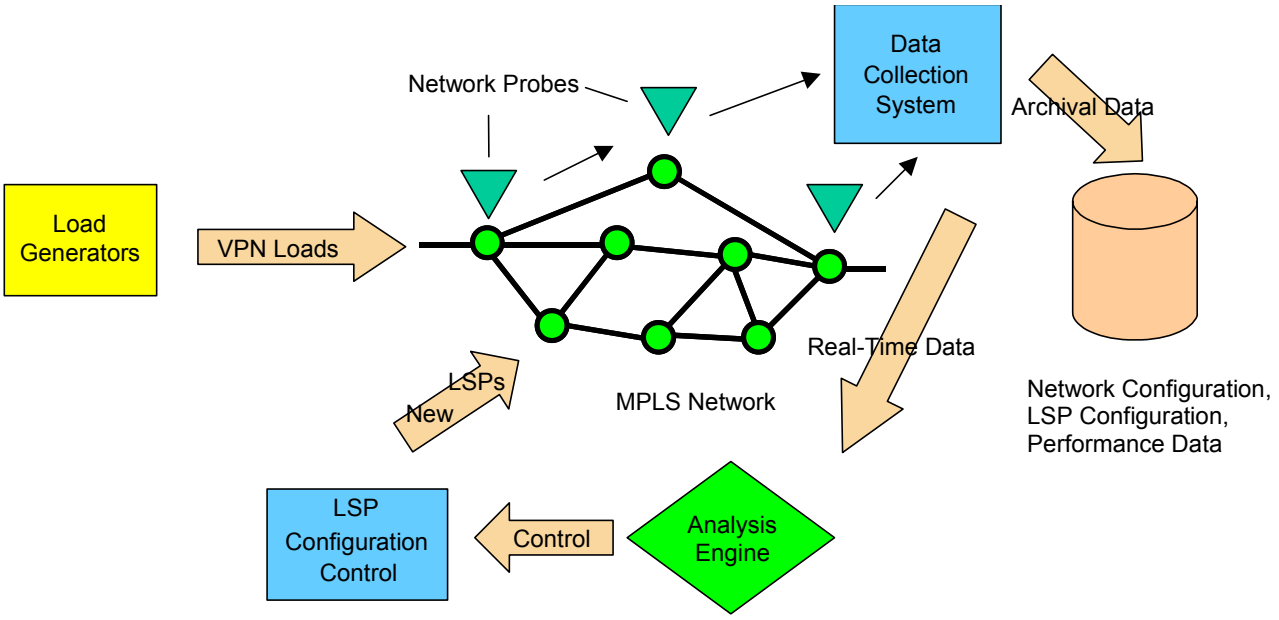

Figure 2. Overall Rondo architecture illustrating the relationship among major functional components.

A. System Components

Rondo is composed of the major parts shown in Figure 2. In the remainder of this paper, we will describe each element with emphasis on the data collection subsystem and the analysis engine.

1) Physical Network

The experimental network is a set of 10 MPLS-enabled routers and interconnections patterned after a much-scaled-down representation of a major service provider's network backbone as depicted on their web site. We note that the provider has 2500 PoPs worldwide so our model has only rough equivalence to reality. However, even with only ten routers, our network exhibits complex and often fascinating behaviors. Routers are connected with 10-megabit links, which makes possible the creation of realistic overload conditions. Each router models a PoP (Point of Presence) on the network where customer nodes are attached. In *Rondo*, each node attached to a PoP is a PC that sends and receives packets.

The network uses a combination of Cisco® 3620 and 3640 series routers. The release of Cisco's IOS (Internet Operating System) available on our routers allows only destination-based selection of MPLS tunnels. Upgrades will ultimately allow selection of the tunnels based on other parameters in the IP packet.

2) Programmable Load Generators and Loading Strategy

We use a collection of PCs programmed[1] to generate time-varying loads similar to those expected in an operational network. Background network traffic on the network is constant in time and is generated by commercially available packet generators. Loads are carefully crafted to cause a buildup of congestion that does not have an overall steady state solution, and are designed to stress the given physical topology.

3) Data-Collection System

The data-collection system uses a variety of devices and techniques to monitor the conditions in the network. These include both active and passive methodologies that capture such characteristics as throughput, loss, delay and jitter. Data collection, a key part of *Rondo*, uses an extensible architecture to provide rapid processing of data under time constraints for its collection, reduction and transmission. Data flow from the network probes through the collection system to the analysis engine with little latency and to archival storage at a lower priority. Data are retained in a database system for other applications such as service-level management that do not require rapid data processing. We describe this part of the system in detail below.

Cisco is a registered trademark of Cisco Systems, Inc.

4) Data Model and Database

Rondo uses the database for a variety of classes of information including physical and logical network topology, configuration information and archived measurement data. The algorithms, displays and other components are driven by the information described by this model, and as such, the organization of this model is crucial to the effectiveness of *Rondo*. The model, which is important for other applications, is realized in a relational database. The most important function of the database is to hold the state of the network topology, which changes as the system reroutes LSPs to alleviate congestion. The analysis and reroute engine periodically updates the topology as the network is reconfigured.

5) Analysis and Rerouting Engine

This element of the system contains techniques for detecting congestion in a network and altering the existing traffic flows to eliminate an overload condition. The engine is designed to focus on more than link utilization, which is the most basic metric of network performance. Utilization indicates the level of activity between network elements and is often viewed as a measure of network congestion.

This view is too simple when one considers the classes of traffic that flow over an IP network. High utilization of a link is one form of congestion, but others might include excessive delay, jitter or high packet loss, all of which could happen at relatively low levels of link utilization. These are measures of congestion that seriously affect proposed services in next-generation IP networks, including voice and video. The engine is designed use any measurable quantity as an indication of a network problem that needs correction.

6) MPLS Configuration and Control

Rondo relies on MPLS to form explicit paths through the core network. Explicit paths allow precise control over the placement of traffic flows within the routed domain of *Rondo*. All traffic in *Rondo* flows through explicitly routed MPLS tunnels, which specify each node along a path from the ingress to egress routers. The network configuration is initially optimal in the sense that all tunnels travel via the shortest path in the network. Once established, packets enter the MPLS tunnels as a function of their destination address and are delivered to the egress router. *Rondo* thus uses MPLS as a mechanism for packet forwarding that is not directly aware of quality of service. Mixing packets with different levels of quality of service in an LSP is possible though but limits the effectiveness of available controls.

Once the initial explicit paths are established, the analysis and reroute engine operates to reroute packets through a path established by a new MPLS tunnel, which may no longer be the shortest path. This action currently takes place via IOS commands that are issued from the controller[4]. When MPLS traffic-engineering MIBs[5][18][19] become available, the controller will use SNMP to establish the new routes.

B. System Operation

The analysis and rerouting engine is in overall control of the system. The engine communicates with the data collection system to establish a schedule of network measurements. As the data collection system takes each measurement, it notifies the analysis and rerouting engine of the presence of new data. The engine combines the new data with the current system configuration and previous data to decide on the appropriateness of rerouting an MPLS tunnel. If a move is appropriate, the analysis engine reconfigures the network through the LSP configuration control and updates the network state in the database.

As we discuss in the following, the route of the new MPLS tunnel does not necessarily preserve overall network optimality. Rather our goal is to reroute traffic as quickly as possible to minimize the congestion at the expense of achieving a theoretical optimum over the whole network. Global optimization might imply moving many or even all the routes in the network. The strategy in *Rondo* is to move from one to a few MPLS tunnels over a period of a few minutes with minimal disruption to network traffic.

III. ALGORITHM

The heart of the analysis and rerouting engine is the rerouting algorithm, which is described in detail in this section. We first outline some assumptions delineating the operation of the algorithm and then give a systematic exposition.

A. Assumptions

1. The network is assumed to contain enough physical nodes and links so that sufficient alternate paths exist to make the possibility of rerouting practical.

2. The path selection algorithm is not required to be optimal. The goal is to reroute traffic as quickly as possible to minimize the amount of affected traffic. The path is selected subject to constraints on bandwidth consumption and hop count, but path selection does not consider the improvements that can be obtained with a global analysis of the network. For example, simultaneously rerouting multiple LSPs is not considered.

3. The state of the system is available and can be used to relate a congested link back to the set of LSPs that contribute to the traffic on that link.

4. Information about the state of each link is measured and available for the analysis and rerouting engine. These measurements include utilization, delay, loss and jitter for each LSP on each link. In the simpler case where delay, loss and jitter are not considered, bandwidth utilization of each LSP on each link will

permit the rerouting of traffic within the available bandwidth space on other links.

5. Where there are several classes of service on a network, the algorithm assumes each class of service is an overlay network that is independent of the others. The algorithm will only manipulate LSPs that are assigned to a single class. It will not attempt to vary the amount of traffic carried in total on each class. The class-of-service settings are established during the configuration process and are not adjusted in response to short term needs. For example, assuming gold, silver, bronze and best-effort network classes, this algorithm will not displace silver traffic to accommodate excessive gold traffic. It also will not permit excessive silver traffic to consume unused gold capacity, as the latter can become unavailable at any moment.

B. *Invocation*

The analysis and rerouting engine holds the current state of the network, so it drives the execution of the algorithm. The engine accepts link congestion data from the data-collection system, which notifies the engine every t seconds. The data levels are divided into two zones: normal and danger. A link is considered to be in the normal zone if its datum is under M, a tunable parameter, and is in the danger zone when it is over or equal to that value. After accepting the data, the engine also checks and collects links that are in the danger zone into a set S. If S is non-empty at the end of an update cycle, then the rerouting algorithm is triggered. When the algorithm reroutes an LSP, the data reading cycle is increased to T to allow time for the network to stabilize.

While several types of data may be collected from the network, *Rondo* currently uses link utilization as its metric. Data on link loads arrive at the engine every 30 seconds (t = 30), at which time links over 80% capacity (M = 0.8) are placed in the set, S, of candidates for rerouting. After rerouting, the network is allowed to settle for 60 seconds (T = 60) to eliminate possible thrashing.

C. *Algorithm Details*

The algorithm discussed here is designed to reduce the utilization on overloaded links to below the threshold of congestion, M. It could as well apply to other metrics of congestion, e.g. delay, loss or jitter, provided a model exists for the composition of these parameters under the aggregation of network traffic flows. Utilization composes by the addition of the utilizations of the individual flows until link capacity is reached.

The goal of this algorithm is to reduce as many overloaded links as possible. Each invocation of the algorithm considers rerouting one LSP in the candidate set, S, and uses link bandwidth in the calculation of the cost. The steps involved are as follows:

1. Examine all links for aggregate indications of congestion. As explained above, a link is congested if its utilization is over the pre-configured threshold M.

2. For all congested links, acquire the set L of LSPs that pass through any congested links.

3. Sort L according to the impact of congestion on particular LSPs. In the simplest case, this is by descending order of consumed bandwidth.

4. Perform a constrained shortest path first (CSPF) search to find an optimal path, l', from the source of L to the sink with the least cost, using a modified Dijkstra's algorithm for single source to single destination shortest path algorithm. The objective function for establishing the link cost is a complex function of link utilization and other constraints to improve the overall distribution of LSP loads. For example, our initial function operated solely on resulting link bandwidth. We are investigating the incorporation of other factors such as avoiding back-up links or mutually interfering traffic flows.

5. Among the paths found in step 3, select the LSP, l, with the least cost alternative optimal path, l'.

6. Reroute the selected LSP.

7. Update the link load statistics to account for the rerouted LSP. The system continues to maintain a short history of the network loads.

IV. Data Collection

The heart of collecting data from the network lies in the network probes. These are usually commercial hardware boxes – often stand-alone, sometimes associated with routers – that measure network traffic at various protocol levels. Probes present the data they collect in a variety of often-inconsistent ways. The job of the data-collection subsystem is to regularize and reduce the data so the other parts of the *Rondo* system have timely and efficient access to it.

The requirement of timely and rapid collection arises from the need for system stability in the network. As a rule of thumb, the delay in corrective action taken in the network should be several times as long as the time taken to perceive the need for a correction. Thus if *Rondo* inspects the network every 30 seconds, the data it has available should not be more than 3 seconds old. We easily meet this requirement with the distributed architecture outlined below. There are further requirements for stability that are explored in the previous section.

In what follows, we outline the diverse nature of some of the probes needed in *Rondo*. The variety of the data, the processing necessary for *Rondo* and the magnitude of the data stream argues for a distributed collection system with

peer-to-peer relationships and significant processing power close to the elements being measured. Reducing the data near its source, aggregating the reduced data into efficient packages and efficiently transmitting the data to the control elements of the system is key to automating the management of a network. The sections that follow cover the issues of synchronous versus asynchronous readout of the data, distributed versus centralized data reduction and methods for archiving the data[14].

A. Probes

This section explores some of the characteristics of probes that make necessary a relatively complex framework for data collection. Network probe is the generic term for hardware or software that contains or uses a network interface to measure data moving through a network. Within this broad definition, there are many styles of probe.

One of the most common types for network measurements is based on SNMP (Simple Network Measurement Protocol[7]), which does not address the form and type of collected data. SNMP has rather low-level functionality and high overhead, which makes it relatively inefficient in large-scale networks. Its primary advantage is its ubiquity. SNMP defines how data move between the probe (often termed an agent) and the client (often referred to as a monitor).

A MIB (Management Information Base) defines the data structures, which are accessed synchronously through an addressing scheme based on a hierarchical name space. MIBS are fundamentally a definition of a global data space that has no inherent operations except reading and writing data cells through the protocol. Relatively recent extensions have added operations that are more complex by defining control variables that implement what amounts to function calls.

In addition to synchronous data transfer, SNMP has a primitive facility that allows the probe to send event notification to its client. These events are termed traps, and are not used in the collection system

1) MIB-2

The IETF (Internet Engineering Task Force) defines a wide variety of MIBs, all of which fall into two broad categories with some overlap, management MIBs and traffic-engineering MIBs. The *Rondo* collection subsystem is concerned only with the latter. The most commonly used MIB with a traffic engineering component is MIB-2[8][9][10][11], which is totally passive and enables packet and octet counts for physical and sometimes logical interfaces at the link level. Data are collected from packets passing both to and from an interface. Most computers and routers define MIB-2 in their SNMP servers. One of the most notable attributes of MIB-2 is its complete lack of awareness of IP and higher-level protocols when monitoring network traffic flows. It measures only link-level packets and records total counts not just the packet payload. Still it is a useful tool and provides important data to *Rondo*.

The notion of an interface is ultimately relatively complex[12]. MIB-2 was originally developed as a tool for counting packets and octets through a NIC (Network Interface Controller) in computers and routers. These serviced Ethernets, token rings, T1's or similar connections. MPLS (Multi-Protocol Label Switching) occupies an unusual position in the protocol stack. MPLS is not a network layer protocol since it lacks end-to-end addressing and routing functionality. Further, MPLS is not a link-level protocol, since MPLS constructs such as LSPs can span multiple routers. Vendors typically represent LSPs in one of two methods. (1) LSPs are represented as logical interfaces or as tunnel interfaces; (2) LSPs are represented as entries in the routing table. Cisco's implementation uses (1) and hence the head-end of each LSP has an interface definition in MIB-2. The data derived from these LSP interfaces are important tools for congestion control in *Rondo*.

2) RMON

RMON[21] and RMON-2[20] are passive SNMP-based probes that are more sophisticated than the MIB-2 probes. These probes are aware of IP, TCP and application-level layers in the protocol stack. Although not aware of individual sessions at these upper layers, RMON-2 can monitor flows between pairs of source and destinations or flows to or from a single address at the network protocol or application level. We define flows as streams of packets with specified source and/or destination addresses without respect to transport-level sessions. Inspecting the well-known port numbers present in the source or destination addresses monitors the application layers. In addition to these functions, RMON and RMON-2 can define arbitrary filters and either count or capture packets that satisfy the filter. They can also fire traps based on thresholds or capture time sequences of network measurements.

RMON and RMON-2 have capabilities defined in the IETF standards documents that outstrip the vendors ability to implement. This statement is particularly true on links with high data rates that transport traffic among many sources and destinations. Typically, RMONs are built into a router, or they may exist as stand-alone hardware devices. In either case, RMONs are prone to exhaust memory or processor resources when configured to measure large amounts of data. Pair-wise flows on busy links can yield an immense traffic matrix. Data captured at high rates exhausts buffer space beyond the ability of the client to retrieve it. Many routers use the same the processor for routing and monitoring functions at the expense of monitoring under heavy load.

Probe Type	Intrusiveness	Activity	Solicitation	Standard	Independent Collection
MIB-2	Out of line	Passive	Synchronous	SNMP	No
RMON	Out of line	Passive	Synchronous	SNMP	Yes
SAA	Out of Line	Active	Synchronous	SNMP	Yes
SoftProbe	Out of Line	Active	Synchronous	Ad Hoc	Yes
Ping	Out of Line	Active	Synchronous	Ad Hoc	No
"Future"	In Line	Active	Synchronous	Ad Hoc	?

Table 1. The attributes of various probes mentioned in the text. MIB-2, RMON, SAA and SoftProbe are used with *Rondo*. The "Future" types of probe are not yet available from their manufacturers.

3) Service Assurance Agent

Cisco's Service Assurance Agent™ (SAA) is an active monitoring agent that is embedded in Cisco's larger routers. Cisco's network monitoring and management products, such as CiscoWorks2000™ use SAA as their source of performance data. Provisioning of and data extraction from SAA can be done through SNMP, and Cisco publishes the MIB description[13], which makes SAA available for use by third-party software. At the network layer, SAA provides a simple echo facility that can compute round trip delays and count packet drops using numerous protocols, including IP, SNA, IPX, appleTalk and DECnet. The equipment at the far end only needs to provide an appropriate echo service, and need not be supplied by Cisco. SAA also can measure jitter using UDP. These measurements are one-way (source to destination and destination to source), so a Cisco router is required at the other end to return the traffic. The data provided can also be used to calculate one-way loss and delay. SAA can also measure response times of higher-level services, including TCP connections, DNS, DHCP, FTP and WWW.

4) In-Line Probes

Certain manufacturers are building probes that intercept the packet stream to insert tagging packets that mark points in the flow. The probes operate in pairs with the sending end inserting the tag and the receiving removing the tag, which carries reference data about the packet stream. For example, these data might consist of time-stamps, counts of packets sent or received since the last tag packet. With this sort of synchronization various quantities become available that are difficult to compute with other methods including absolute measurements of packet loss.

CiscoWorks2000 and SAA are trademarks of Cisco Systems, Inc.

5) Rondo Softprobe

Many other probes (e.g. [3][15]) exist on the net and in the literature that we cannot cover in this paper. However, an especially useful set of measurements to have for congestion control includes delay, loss and jitter for a one-way packet transit of the network. These measurements should be taken through MPLS tunnels if present and at all level of QoS (Quality of Service). Measuring one-way delay, loss and jitter requires accurate, absolute time scales at both the sending and receiving nodes. In the past, this requirement has been difficult to meet, but now with easily available GPS receivers, it is possible to satisfy. The *Rondo* Softprobe provides these measurements to *Rondo* [17]. The SoftProbe is an active probe. It samples the quantities it needs by emitting a stream of packets. Delay, loss and jitter are computed from timing measurements of the packet stream.

6) Probe Attributes

The probes discussed above may be classified according to certain attributes that affect their interaction with the data collection subsystem. These attributes are summarized in the following:

Intrusiveness – A probe may have the packet stream pass through its fabric enabling it to insert and remove packets that function as markers in the flow. The overwhelming majority of probes are out-of-line with respect to the packet stream. They can be physically inserted or removed from the system without disrupting the flow. In-line probes require the stream to be physically disconnected before the probe is inserted.

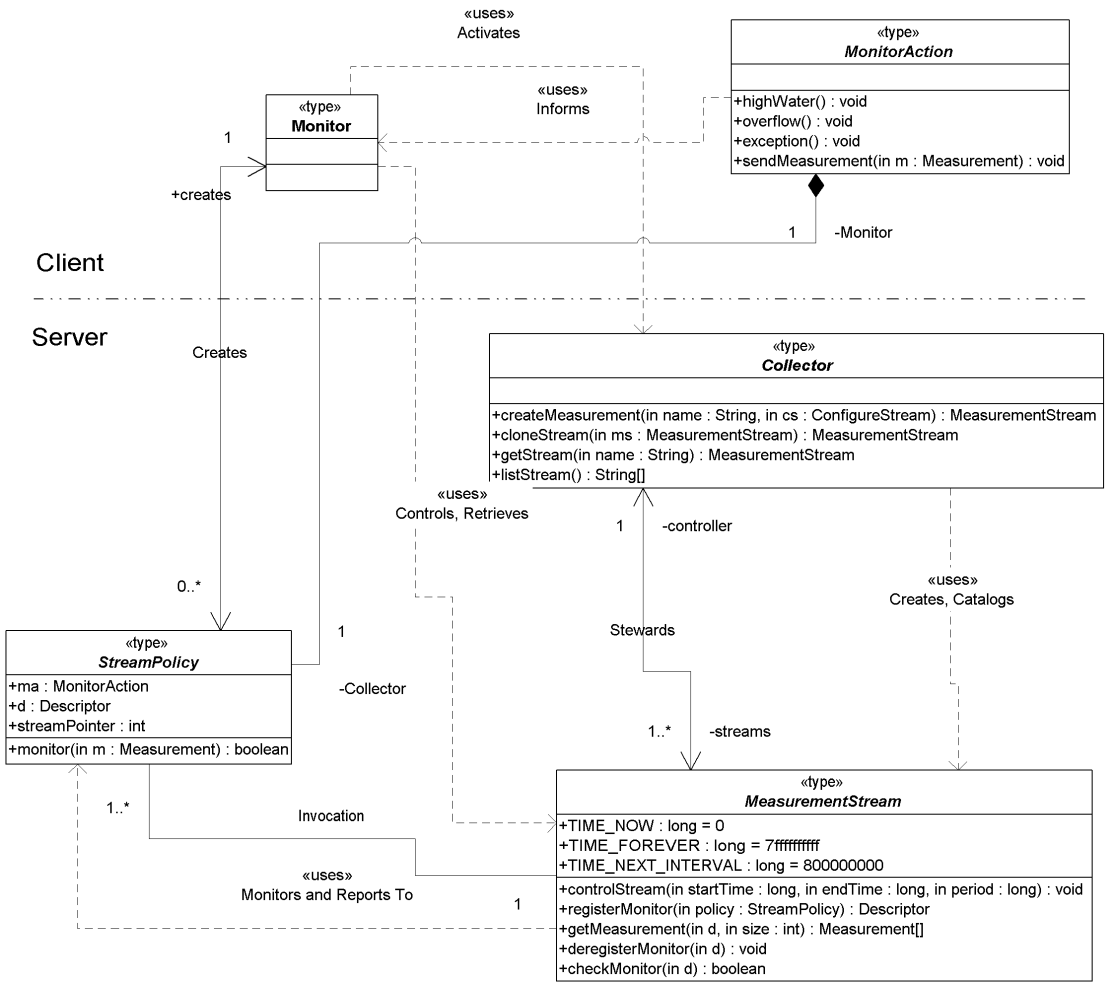

Figure 3. The Rondo architecture showing the principle base classes. Implementation classes are all derived from these. Method calls that cross the client-server boundary are made through Java RMI (Remote Method Invocation).

Activity – A probe may obtain its data by passively monitoring the traffic flow, or it may actively insert packets in the stream to take it measurements. The attribute is distinct from the intrusiveness of the probe. The *ping* command running on a PC connected to an Ethernet is active but out-of-line. One can disconnect the computer without disrupting the network.

Solicitation – A measurement might be requested synchronously from a probe, in which case the client waits for the result, or it might be sent asynchronously as part of a notifying event. The latter might occur if a threshold were crossed as part of a measurement sequence. Most probes operate synchronously, but our architecture allows watching the data stream and emitting notifying events to the rest of the system.

Standard – Most commercial network probes use SNMP to communicate with the client, but other standards are possible including CORBA, EJB, JMS, DCE, etc.

Independent Collection – Rather than counters measuring packets and octets, independent collection implies a higher level of functionality. A probe might collect a table of measurements over a certain period and might further reduce the data into a few derived quantities.

Table 1 classifies the probes discussed in the text according to these attributes. The following sections discuss the architecture of the *Rondo* data collection subsystem and out-lines how this architecture adapts to each of the attributes.

B. Data Collection Architecture

Rondo uses an object-oriented architecture. The data collection subsystem uses the five base classes shown in Figure 3. Each class contains the fundamental methods that manage data collection. Each of these classes is specialized in a fully realized data collection subsystem.

Figure 3 depicts a client-server architecture. The fundamental class on the server side is the *Collector*, which oversees the management of a particular type of data. In *Rondo,* these include data from MIB-2 interfaces, MPLS forwarding tables and SAA timing measurements. We intend that the client-side *Monitor* activates a single copy of a particular *Collector* on each server. For example, a server would have a single instance of the interface-data collector and a single copy of the SAA-data collector.

Each *Collector* uses the *createStream* method to instantiate one or more *MeasurementStream* objects. A *MeasurementStream*, as the name implies, is a sequential, time-stamped set of data of the type specified by the *Collector* that arises from a particular device or *agent.* For example, each MIB-2 interface is the source of a stream of *InterfaceMeasurement* objects. Associated with each *MeasurementStream* is a stream pointer that is analogous to a file pointer in the Unix operating system. The *getMeasurement* method reads a group of data points specified by the *size* parameter in the method and advances the stream pointer by that amount. Other methods present in the *Collector* class include *cloneStream*, which is analogous to the *dup* system call in the Unix operating system, *getStream*, which returns a *MeasurementStream* by the name assigned to it during its creation, and *listStream*, which reports the names of all active *MeasurementStream* objects.

Another important method in *MeasurementStream* is *controlStream*, which determines that starting time, stopping time and frequency of measurements for a given *MeasurementStream*. A stream may have its parameters changed asynchronously. It need not complete the current course of measurements before its collection parameters are modified in any way.

Associated with each *MeasurementStream* are one or more *StreamPolicy* objects, which are passed as arguments from a *Monitor* into the *registerMonitor* method of the *MeasurementStream* class. *StreamPolicy* objects are extensions of a *Monitor.* They should contain logic that is intended to reduce data near to the point of measurement. For example, suppose that the Hurst parameter were important to a particular network-management function. Rather than move all the data to a centralized processor, we believe a superior design processes the collected data close to the source. Another function that might be included in a *StreamPolicy* object is threshold detection, which implies asynchronous notification of the client.

StreamPolicy objects support this capability through associated *MonitorAction* objects, which contain callback methods from the server side to the client. The base class, *MonitorAction*, has a few fundamental methods defined, but the intent is for the application to extend this class in ways specific to the needs of the problem. The methods, *highWater* and *Overflow*, are notifications to the application that data in a *MeasurementStream* need attention. The *exception* method is intended as a general alarm for server-side problems and *sendData* provides an asynchronous update capacity.

The *MeasurementStream* class also contains methods to register and de-register *StreamPolicy* objects (*registerMonitor* and *deregisterMonitor*) as well as a method to check the status of a *StreamPolicy* (*checkMonitor*). Individual *StreamPolicy* objects are referenced by an identifier that is unique in time and space.

StreamPolicy objects and their associated *MonitorAction* objects form the basis of a general and flexible computational model. Because both classes are logically part of an application, they can be tailored to meet the needs of a particular network management problem without having to custom tailor the data collection servers to deal with the changing needs of applications. The servers can concentrate on efficient data collection and stewarding. The applications have the ability to offload some processing to the servers, thereby reducing the flow of data across the management network and the amount of data reduction necessary. This strategy is particularly effective when coupled with automatic code distribution as we discuss in the following section.

C. Design and Implementation

The *Rondo* data collection architecture is realized as a set of Java classes. Each of the types discussed above is either an abstract base class or an interface definition. We construct the collection system using standard Java-based tools, but other implementations and programming paradigms fit the architecture as well.

1) Programming Paradigms

Data collection in *Rondo* uses the Java RMI (Remote Method Invocation) subsystem as its method of distributed computing. Both advantages and disadvantages arise from this decision. RMI is fundamentally formulated with remote procedure calls and threads as the most important entities. Each thread holds state, and the several threads comprising a process must synchronize their operations on shared data. Certain threads are allowed to block as in the class that contains the timed measurement loop, because others threads are live and waiting to process other remote-procedure calls. This sort of design is relatively easy for developers to handle because of the sequential nature of the processing, but can lead to pitfalls when dealing with access to shared data. We chose this design as a vehicle for a prototype implementation because it was easy to set up in a laboratory environment.

However, our architecture is not limited to this style of programming. A more common approach for large systems is the more loosely coupled *software bus*, which is a message passing system at its base. Each entity on the bus registers its interest in receiving certain classes of message through a publish-and-subscribe mechanism. The method calls devolve into lower-level messages and, while threads are not as important in this programming style, they are still available to the developer.

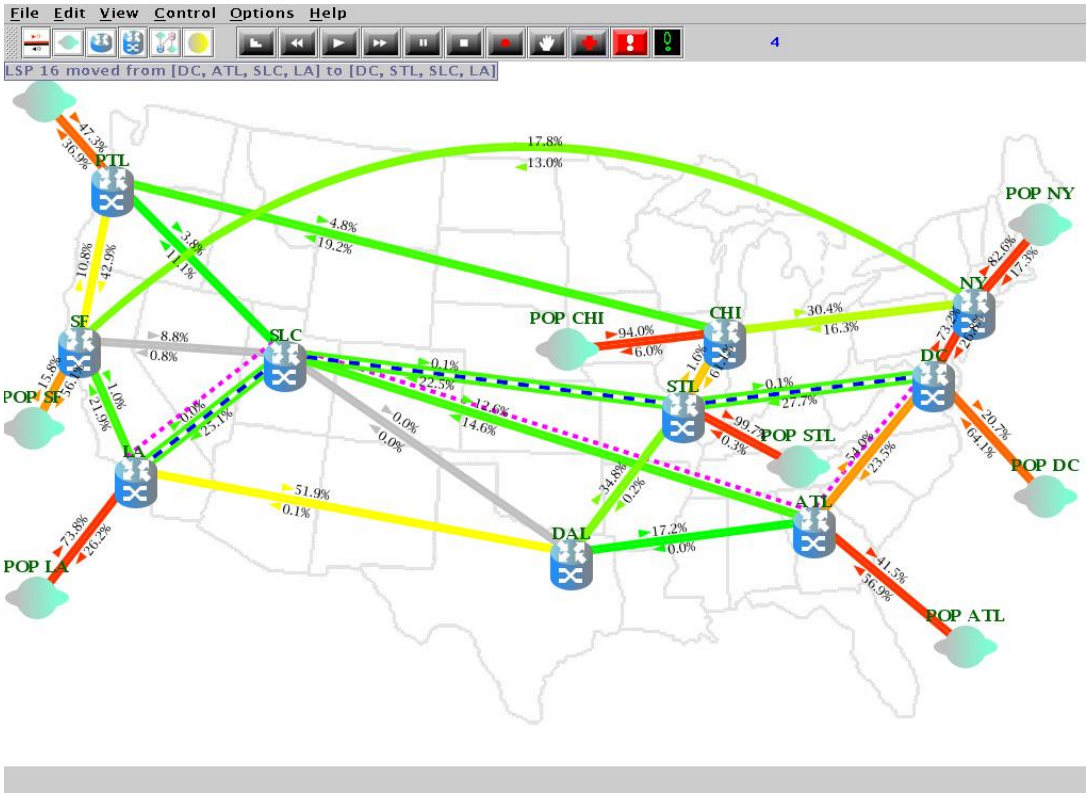

Figure 4. The main screen in the Network Operations Center for Rondo. The Rondo laboratory network is based on Cisco routers using 10 MBit Ethernets as links. Our experimental network is modeled after a large network provider's core infrastructure. Although depicted here in black and white, the colors on each link in the display represent utilization from the highest, red, to lowest, dark green, according to the scale at the bottom of the display.

For example, timed measurement loops work differently in this paradigm. Instead of sleeping within a thread, the message requesting a measurement waits in a queue for delivery at a specified time. At message delivery the system takes a new measurement and requeues a new measurement message for delivery at the next time interval. All state required to control the measurement loop is retrieved at the beginning of each measurement and stored at its end.

Independent of programming paradigm, Java has a large advantage over other systems because of object serialization and automatic code distribution, which is a concomitant feature of serialization. RMI passes local objects by value and has a feature allowing the receiving process to find the object-method code through a standard URL. Remote objects, i.e. those with remotely invoked methods, are passed by reference through a standardized stub generation technology. In our application, *StreamPolicy* objects, which are local Java objects, pass by value through the system. Because these tend logically to be part of an application, its functionality is automatically distributed throughout the system without recourse to explicit software upgrades. Each application can customize its view of the same data stream without explicitly changing the functionality of the server.

2) *Probes and Data Collection*

We discussed the characteristics of various types of network probe in Section IV. Four of the five attributes affect the design of the data collection system – *Activity*, *Solicitation*, *Standard* and *Independent Collection*. The probes listed are consistent with the architecture of the data collection subsystem. The *MeasurementStream* class and its descendents have the capacity to isolate the functional details of collection technology from the rest of the system.

V. Discussion and Issues

A prototype of the *Rondo* system has been running in a laboratory environment since September of 2000. *Rondo* collects packet and octet counts for physical network interfaces and virtual LSP interface using MIB-2. Delay and jitter are collected using SAA. Packet forwarding data for LSPs are collected at intermediate routers in an LSP using an ad-hoc connection to the command-line interface of IOS. All of these sources fit within the *Rondo* measurement framework, which forwards the data to the analysis and rerouting engine.

We have constructed a graphical user interface, shown in Figure 4, that might be used in a Network Operations Center as part of *Rondo*. In the current implementation, the analysis and rerouting engine controls the display. In a system that controls a deployed network, the analysis and rerouting engine would be separate from the user interface.

The NOC display shows a map of the United State with the "location" of each PoP in our experimental network. Controls at the top of the display allow the operation to start and stop automatic congestion control. Although the figure is black and white here, the inter-PoP links indicate their utilization as a series of colors from highest utilization, red, on down through the colors of the spectrum to the lowest utilization, dark green. Links are bi-directional with the attached numbers giving link utilization in each direction.

At the moment depicted, *Rondo* has just rerouted an MPLS tunnel

- from LA→SLC→ATL→DC
- to LA→SLC→ATL→DC

because the ATL→DC link was overloaded. The new route has a series of dashes running through it and the old route has the dashes alongside.

Although *Rondo* works well as a proof of concept in a laboratory environment, there are many issues remaining as it moves towards deployment in a full-scale network. We outline some of the more difficult ones here.

- Although *Rondo* has proven stable under rerouting of single LSPs with a small network, what happens in a larger network?
- Can *Rondo* move several LSPs in one operation to handle cases of simultaneous overloads?
- Can *Rondo* move several LSPs in one operation for a single overloaded link to achieve a more optimal solution? Over what domain of LSPs should optimization take place?
- We believe our method of pushing application functionality to servers distributed in the network is a scalable solution, but it should be tested in a production environment with a larger data flow.
- What strategies are there to reduce the amount of data that the analysis and rerouting engine needs to make intelligent changes in MPLS tunnels?

REFERENCES

[1] B. Adamson, *The MGEN Toolset*, http://manimac.itd.nrl.navy.mil/MGEN.

[2] D. Awduche, J. Malcolm, J. Agobua, M. O'Dell, J. McManus, *Requirement for Traffic Engineering for MPLS*, RFC 2702, September 1999.

[3] N. Brownlee, *Traffic Flow Measurement: Meter MIB*, RFC2720, October 1999.

[4] Cisco™ Configuration Guides and Command References, http://www.cisco.com.

[5] J. Cucchiara, H. Sjostrand & J. Luciani, *Definitions of Managed Objects for the Multiprotocol Label Switching, Label Distribution Protocol (LDP)*, draft-ietf-mpls-ldp-mib-07.txt, August 2000.

[6] J. P. Curtis, J. G. Cleary, A. J. McGregor & M. W. Pearson, *Measurement of Voice Over IP Traffic*, Pam2000: Passive and Active Measurement Workshop, Hamilton, New Zealand, pp 43-59, April 2000.

[7] D. Harrington, R. Presuhn & B. Wijnen, *An Architecture for Describing SNMP Management Frameworks*, RFC 2571, April 1999.

[8] K. McCloghrie & M. Rose, *Management Information Base for Network Management of TCP/IP-based Internets: MIB-II*, RFC 1213, March 1991.

[9] K. McCloghrie, *SNMPv2 Management Information Base for the Internet Protocol using SMIv2*, RFC 2011, November 1996.

[10] K. McCloghrie, *SNMPv2 Management Information Base for the Transmission Control Protocol using SMIv2*, RFC 2012, November 1996.

[11] K. McCloghrie, *SNMPv2 Management Information Base for the User Datagram Protocol using SMIv2*, RFC 2013, November 1996.

[12] K. McCloghrie & F. Kastenholtz, *The Interfaces Group MIB using SMIv2, RFC2863,* June 2000.

[13] L. Metzger, *Response Time Monitor MIB*, ftp://ftp.cisco.com/pb/mibs/v2/CISCO-RTTMON-MIB.my.

[14] V. Paxson, G. Almes, J. Mahdavi & M. Mathis, Framework for IP Performance Metrics, RFC 2330, May 1998.

[15] V. Paxson, A. Adams & M. Mathis, *Experiences with NIMI*, PAM 2000: Passive and Active Measurement Workshop, Hamilton, New Zealand, pp. 87-97 April 2000.

[16] E. Rosen, A. Viswanathan & R. Callon, *Multiprotocol Label Switching Architecture*, RFC 3031, January 2001.

[17] B. Siegell, J. DesMarais, M. Garrett, P. Seymour & D. Shallcross, *Felix Project: Topology Discovery From One-Way Delay Measurements*, PAM 2000: Passive and Active Measurement Workshop, Hamilton, New Zealand, pp 107-115, April 2000.

[18] C. Srinivasan, A. Viswanathan & T. Nadeau, *MPLS Traffic Engineering Management Information Base Using SMIv2*, draft-ietf-mpls-te-mib-05.txt, November 2000.

[19] C. Srinivasan, A. Viswanathan & T. Nadeau, *MPLS Label Switch Router Management Information Base using SMIv2*, draft-ietf-mpls-lsr-mib-07.txt, January 2001.

[20] S. Waldbusser, *Remote Network Monitoring Management Information Base Version 2 using SMIv2*, RFC 2021 January 1997.

[21] S. Waldbusser, *Remote Network Monitoring Management Information Base*, RFC2819, May 2000.