AFRL-IF-RS-TR-2003-26
Final Technical Report
February 2003

RESOURCE MANAGEMENT UNDER LANGUAGE AND APPLICATION CONTROL (REMULAC)

Carnegie Mellon University

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J342

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2003-26 has been reviewed and is approved for publication.

APPROVED: Raymond A. Liuzzi

RAYMOND A. LIUZZI
Project Engineer

FOR THE DIRECTOR: Michael Talbert

MICHAEL L. TALBERT, Maj., USAF
Technical Advisor, Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE
FEBRUARY 2003

3. REPORT TYPE AND DATES COVERED
Final Aug 96 – Oct 02

4. TITLE AND SUBTITLE
RESOURCE MANAGEMENT UNDER LANGUAGE AND APPLICATION CONTROL (REMULAC)

5. FUNDING NUMBERS
C - F30602-96-1-0287
PE - 62301E
PR - E314
TA - 01
WU - 01

6. AUTHOR(S)
Thomas Gross and David O'Hallaron

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Carnegie Mellon University
500 Forbes Avenue
Pittsburg Pennsylvania 14213

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFTB
26 Electronic Parkway
Rome New York 13441-4514

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2003-26

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Raymond A. Liuzzi/IFTB/(315) 330-3577/ Raymond.Liuzzi@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

An ensemble of networked systems is an attractive platform for a wide range of applications. The "GRID" approach is one example of an attempt to harness a large number of systems at different sites together to form a practical computing environment. Networked systems expose applications to the realities of resource availability. Networks either deliver best-effort service (then it is up to the application how to shield the user) or they allow reservations (then the application must figure out what resources to reserve). However, when this project started, networked systems did not support network-side end-to-end resource reporting and management, making life difficult for application developers and user. The Remulac (Resource management Under Language and Application Control) developed a uniform method to provide resource management for applications so that they can benefit from the availability, replication, and cost-effectiveness of networked environments.

14. SUBJECT TERMS
Remulac, Remos, Collectors, Modelers, Predictors

15. NUMBER OF PAGES
29

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

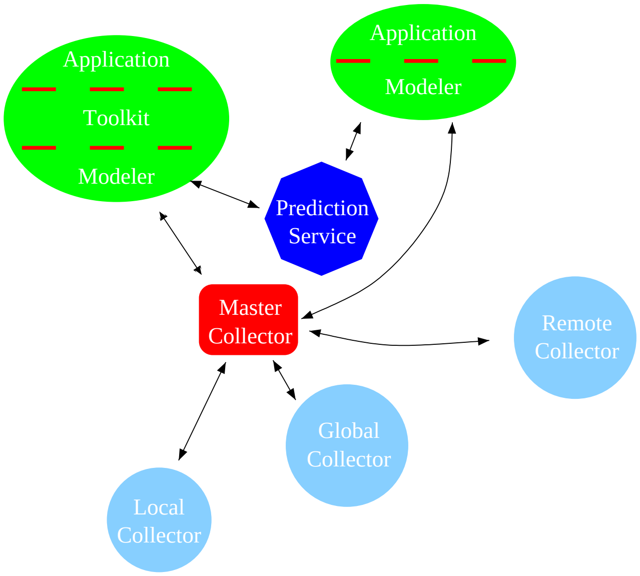
List of Figures and Tables

Executive Summary

An ensemble of networked systems is an attractive platform for a wide range of applications. The "GRID" approach is one example of an attempt to harness a large number of systems at different sites together to form a practical computing environment. Networked systems expose applications to the realities of resource availability. Networks either deliver best-effort service (then it is up to the application how to shield the user) or they allow reservations (then the application must figure out what resources to reserve). However, when this project started, networked systems did not support network-wide end-to-end resource reporting and management, making life therefore difficult for application developers and users. The Remulac (REsource Management Under Language and Application Control) developed a uniform method to provide resource management for applications so that they can benefit from the availability, replication, and cost-effectiveness of networked environments.

The core of the Remulac approach is the Remos (Resource Monitoring System) that provides information to the application about the network environment. This information is then used either by the application directly, by a virtual machine (that provides, e.g., load balancing or process migration), or by a toolkit or framework (e.g., to provide other services like cost-effective or secure communication). Remos uses a query-based interface so that any overhead is experienced only when an application has a clear need for interaction with the performance monitoring system. The two main queries provide information about the (logical) topology and the resource availability for flows.

Remos introduces the concepts of *Collector"* and *Modeler* to address a wide range of networking scenarios. Collectors are responsible for obtaining raw information; modelers support the Remos API for clients and translate application-level requests in network-level requests (and vice versa). A number of collectors have been implemented, including one that is based on an industry standard for LANs (SNMP) and one for wide-area settings that uses benchmarks. Ongoing work (supported by other contracts) has extended the Remos system to deal with wireless LANs.



Remos is being used on a regular basis by several groups (both inside the Quorum effort and elsewhere): QuO (BBN), the Aura Project (CMU), the HiPer-D Testbed (NSWC and S/TDC), CACTUS (University of Arizona), and the Desiderata Project (Ohio University).

1 Introduction

The Remulac (REsource Management Under Language and Application Control) approach provides a uniform framework for resource management that allows us to support applications that demand real-time resource guarantees (e.g., when transmitting a video stream) as well as applications that demand a uniform view of some resources (e.g., using the memory of many workstations as a single, distributed, shared-memory). The core of the Remulac project was the development of *Remos* (resource monitoring system).

The Remos system was designed to provide resource information to distributed applications. Every distributed application that wants to react to changes in resource availability must be able to determine the usage of network resources. Networked systems expose applications to the realities of resource availability and provide applications with explicit information about network resources that allows them to adapt, or tolerate, fluctuations in resource availability in an intelligent way.

The Remos system serves as a foundation for a range of application-specific approaches to deal with network resource changes. These approaches are beyond the scope of Remos, which aims to provide resource measurements across a wide range of network architectures, environments, and implementations. While designing Remos, we considered the needs of many different applications and the capabilities of a range of networking and computing environments. The ability of Remos to support resource measurement in a variety of environments and for a variety of applications makes it an appropriate measurement tool for Grid environments.

A measurement system must address a number of conflicting priorities to be of practical use. Before we present the architecture and implementation of Remos, let us briefly review our design objectives:

- Scalability: Resource monitoring in distributed systems necessarily involves many machines, a large network infrastructure, and many users. The monitoring system should scale well with both the size of the infrastructure and of the user population.

- Usability: The users of the collected information are application developers, not network managers. Our goal is to only provide applications with the information they need, without swamping them with unnecessary details.

- Flexibility: Different users require different types of information. For example, synchronous multiprocessing, real-time video, and bulk data transfer have distinctly different bandwidth, latency, and loss requirements, and require that information across different timescales.

- Portability: Network technology continues to develop at a fast pace. Remos must be able to allow the integration of new networking technologies or new measurement techniques. As a consequence, the Remos design must isolate those parts where modifications are likely (many parts that interact with the base networking technology) and provide a stable API to clients. This will improve portability.

- Robustness: The system must degrade gracefully under load (both on the network and on the resource measurement system).

- Cost-effectiveness: The measurement system should not perturb the measured system to the point that the measurements are meaningless. The application's runtime and space overhead should be proportional to its demands.

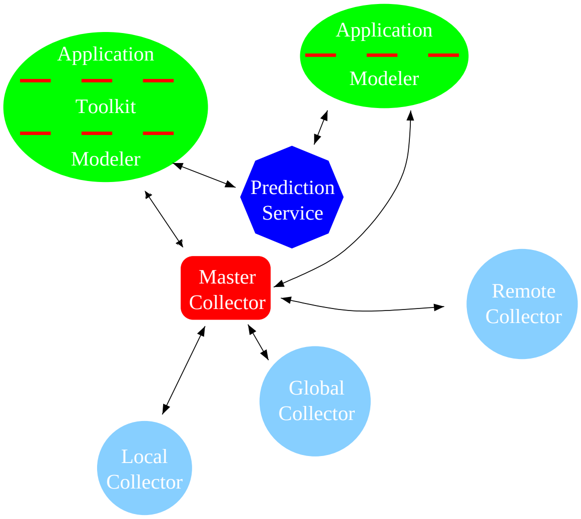


Figure 1: Overview of the components in the Remos architecture.

Remos has been used on a regular basis by several groups: the Aura Project (CMU), QuO (BBN), the HiPer-D Testbed (NSWC and S/TDC), CACTUS (University of Arizona), Rainbow (CMU), and the Desiderata Project (Ohio University). These projects are quite diverse, both with respect to the networks they use and their application information needs. Our evaluation along the "portability" and "flexibility" dimension is in part based on interactions with these users.

In the rest of the report, we describe how the design of Remos addresses the above challenges. In Section 2, we describe the general architecture of Remos. Section 3 describes the techniques used by Remos to implement the architecture. Section 4 describes related work and discusses Remos in terms of the Grid Monitoring Architecture proposed by the Grid Forum. Section 5 evaluates how the design and implementation of Remos meet the original design goals. Finally, Section 6 discusses the lessons learned in the development of Remos as well as issues that require further work.

2 Architecture

An overview of the Remos architecture is presented in Figure 1. The Remos architecture divides the services needed between *collectors*, *modelers*, and *predictors*. The Remos API, which is exposed to applications, is implemented only in the Modeler. This design allows considerable flexibility in varying the design of the other components.

2.1 The Remos API

From most applications' perspective, networks are diverse, complex, shared, heterogeneous black boxes that serve to move data between two points. The goal of Remos is to open up the black box so distributed applications can make appropriate decisions based on the network's capabilities. However, there are inherent tradeoffs in providing detailed information about a network to an application. Too little information may not allow a complex application to accurately predict the performance advantages or disadvantages of various options for data placement, communication pattern, algorithm, or image quality. On the other hand, too much information can make simple questions, such as the

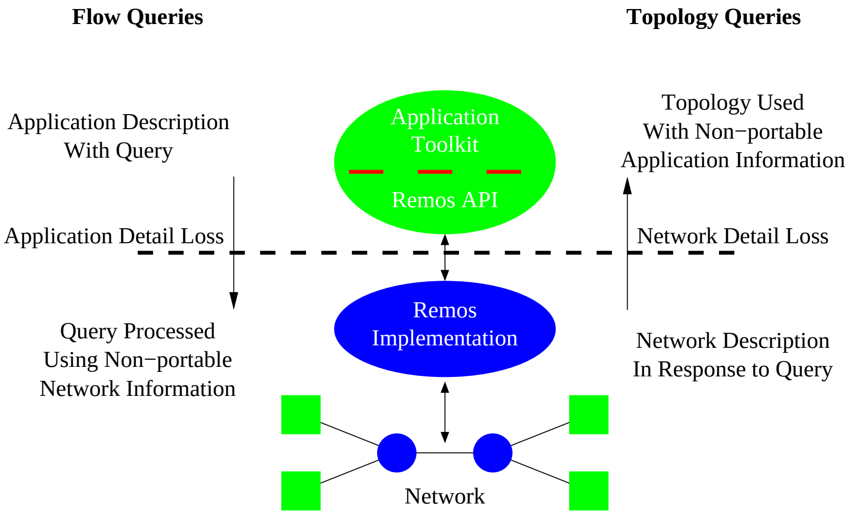

Flow Queries
Topology Queries
Application Description With Query
Application Toolkit
Remos API
Topology Used With Non−portable Application Information
Application Detail Loss
Network Detail Loss
Remos Implementation
Query Processed Using Non−portable Network Information
Network Description In Response to Query
Network

Figure 2: The two query abstractions supported by the Remos API are illustrated here. As each query passes through the abstraction layer between the application and network levels, information is lost. A user should select the best query for an application by evaluating the complexities of the application's adaptation options and whether any unusual network support might be available that is not reflected in the standardized topology description.

bottleneck bandwidth between two hosts, difficult or impossible for an application programmer to answer without learning many details of network behavior. The question for the Remos API, then, is determining the proper level of abstraction for representing network information to the application.

Because we neither wished to sacrifice information detail or the ease of developing applications, we developed two fundamentally different ways for applications to access the data available through Remos, shown in Figure 2. The first, flow-based queries, is used when the application itself is fairly simple, or wants to evaluate the performance a particular communication pattern will receive from the network. These queries require a standardized description of a communication pattern to be used by the application. This introduces information loss, but if the application's communication needs are simple, that loss should be minimal. The standard description is then passed to the network layer, which is free to use whatever network-specific knowledge it has to respond to the query.

The second type of query is the topology query. The topology query is useful for the opposite problem, when an application is rather complex, and its options for network utilization are too complex or would take too many separate queries to evaluate using the flow-based queries. In the topology query, the network's representation, including topology, link capacity, and utilization, is passed to the application layer in a standardized format. Again, this process introduces information loss, but it enables applications to make decisions such as task placement without incurring exponential costs.

Both queries include an age parameter, which allows the application to specify the maximum age of any cached information used to answer the query. The age parameter allows the application to decide on the tradeoffs between the speed and accuracy of the response.

2.2 Collectors

The collectors are responsible for acquiring and consolidating the information needed by the application. Collectors can use a variety of methods of collecting information, e.g. they may incorporate or control *sensors* that perform the actual measurements, but from an architectural view they have a single function: collect information and forward it on to the Modeler. For scalability reasons, collectors can be organized in a hierarchical fashion (Figure 1). At the lowest level, collectors are responsible for collecting information about specific networks. For example, a local collector is responsible for obtaining performance information about its LAN. Global collectors are responsible for obtaining performance information about the networks connecting LANs. Local or global collectors at remote sites can be contacted to obtain information about those remote sites.

The Master Collector is responsible for gathering information from different collectors and coalescing it into a response to a modeler's query. The Master Collector maintains a database of the locations of other collectors and the portion of the network for which they are responsible. When a request comes from a modeler, the Master Collector queries the appropriate collectors and replies without revealing that the response was obtained from multiple collectors. Using this technique, it is possible to build several layers of collectors. For example, the remote collector in Figure 1 might be another Master Collector that in turn contacts a variety of local collectors when queried about its network.

One important advantage of this architecture is that it blurs the line between inter- and intra-site measurements. Because the collectors assume responsibility for contacting remote sites and for aggregating all available information into a single response, neither the Modeler nor the application must determine whether the query concerns nodes at a single site or at remote sites or consider the most appropriate measurement technique. If the WAN link is the only bottleneck along the path of the query, then the appropriate measurement will automatically be returned.

2.3 Modeler

The modeler sits between the application and the Collectors. It implements the API and is responsible for modeling the collector-gathered information about the network into the information abstractions required by the application. Its first task is to communicate with the collector to send requests and receive information about the network. Its second task is the conversion of the raw network information into abstractions with which the application can work.

From an architectural view, the modeler functionality and the abstractions it defines are related to the application, whereas the collectors work with network-related abstractions. A collector discovers network nodes and links, which are formed into a network topology graph by the modeler. Similarly, a collector measures the resources of individual nodes and links inside a network; the modeler uses these resource measurements to calculate information useful to the application, such as the resources available to a set of network flows, or to identify a bottleneck along a path.

An application communicates with exactly one modeler, which runs on the same node as the application. In contrast, the modeler may gather information from many collectors, which are scattered across the network. This division of labor allows Remos to reduce the overhead imposed on the application by performing most of its work on separate nodes. It also eliminates the need to store application state in collectors, allowing each collector to deal with queries in a stateless fashion, while the modeler retains the ability to customize its responses, query intervals, and predictions based on the application's state.

2.4 Predictors

Predictors are responsible for turning a measurement history into a prediction of future behavior. The predictors used with Remos are part of the RPS Toolkit developed by Dinda [9]. Although the API supports requests for predictions of network behavior, in practice we only implemented the predictors for host-load. Because the core of Remos focuses on network monitoring, we will not discuss the predictors further here. Readers interested in the host load prediction are referred to Dinda [8]. The effectiveness and quality of bandwidth predictions based on SNMP and benchmark measurements are discussed elsewhere [21, 23, 36].

3 Implementation in Remos

The implementation of Remos is diagrammed in Figure 3. This figure illustrates how the various collectors used by Remos interact when used in a grid-like environment. As described in the previous section, the complexities of the system are hidden from the user by partitioning Remos into application-side (Modeler and predictors) and network-side (collectors) components. To cover the wide variety of networks and administrative domains in which Remos must run, we developed a set of three collectors.

SNMP Collector Collects topology and bandwidth information from SNMP-enabled IP routers.

Bridge Collector Determines the topology of bridged Ethernet LANs and forwards the topology information to the SNMP Collector to augment its bandwidth monitoring capabilities to include SNMP-enabled Ethernet switches.

Benchmark Collector Performs active TCP probes by transferring data between two machines and recording the bandwidth obtained during the transfer. Used in networks where no SNMP access is available.

This combination of collectors provides support for the vast majority of networking situations. The majority of current LANs are built using bridged Ethernet. The combination of the Bridge Collector and SNMP Collector provide topology and bandwidth information in these networks. Most campus networks are built with routers connecting various Ethernet subnets, which are supported by the SNMP and Bridge Collectors. When two sites are connected across the Internet, it is typically impossible for an end-user to obtain SNMP access to the relevant routers, but the Benchmark Collector performs probes across that span of the network to the remote site, where a local SNMP Collector performs its monitoring.

In the remainder of this section, we describe how each collector is implemented and how they communicate with one another. We conclude with a brief description of the Modeler implementation.

3.1 SNMP Collector

The SNMP Collector is the basic collector that Remos uses for most of its network information. SNMP is a database protocol designed to provide network administrators with direct access and control over the status of network devices [29]. The SNMP Collector uses these features to obtain network-level information about topology and performance directly from routers and switches. Because the SNMP Collector has direct access to the information the network itself stores, this collector is capable of

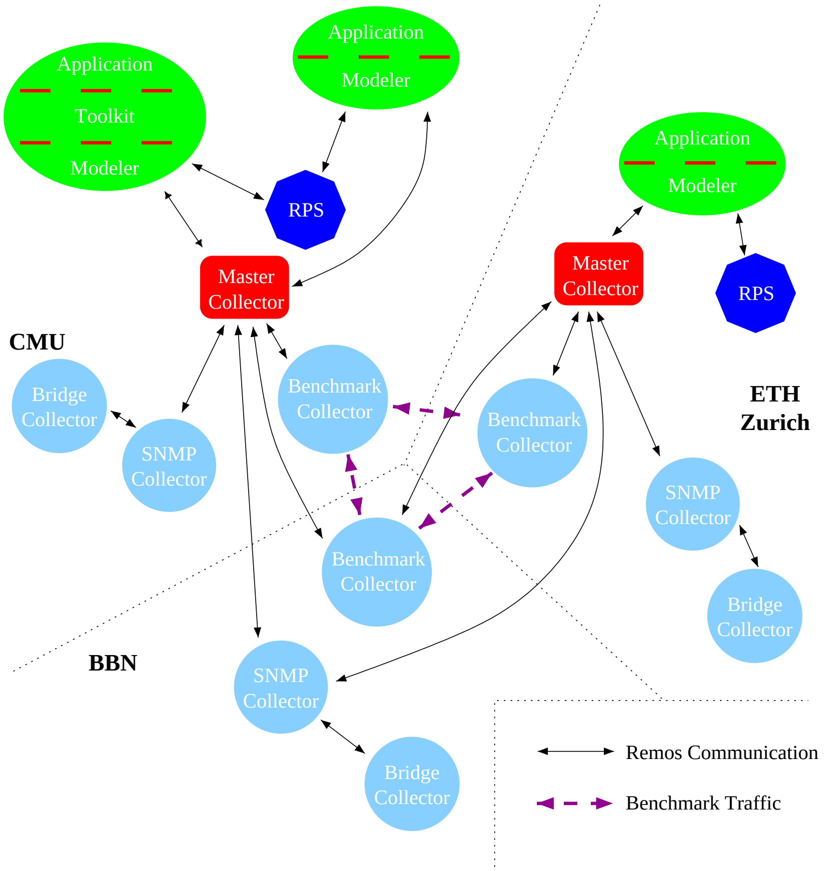


Figure 3: A detailed illustration of how the components of the Remos architecture are connected. Shown here are applications running at CMU and ETH making use of resources at CMU, ETH, and BBN. Each application is using prediction services to provide information about the future network availability. The applications at CMU are using machines at CMU and BBN, and the application at ETH is using machines at ETH and BBN. The benchmark measurements sent across the Internet are shown, but, for clarity, the connections between the SNMP and Bridge Collectors, and the network components they monitor are not shown.

answering the flow and topology queries that require an understanding of the details of the network's structure [23]. The SNMP Collector operates on routed networks (level 3).

An SNMP Collector is assigned to monitor a particular network, generally an IP domain corresponding to a university or department. Because SNMP agents are normally only accessible from local IP addresses, these administrative restrictions dictate the location and areas of responsibility for the SNMP Collectors.

The SNMP Collector monitors the network on an on-demand basis. It waits for queries, then explores and begins monitoring the network components needed to respond to that query. Once it begins monitoring parts of the network, it will continue with periodic monitoring to collect history of that network for use in predictions. The collector can also be configured to begin monitoring specific resources at startup for use in a computational center or with other known resources.

The first and most complex step the SNMP Collector must take upon receiving a query is topology discovery. Using the IP addresses of the nodes in the query and the routers they are configured to use, the collector follows the route hop-to-hop between each pair of nodes in the query. While simple, the algorithm is quite expensive since it has a running time of $D \times N^2$, where D is the diameter of the network and N is the number of endpoints in the request. However, to help reduce the actual running time, our algorithm stops a path search between a pair of nodes when we reach an earlier discovered path to the same destination. Also, in subnets where routes are symmetric, half the queries can be eliminated.

Once the collector has discovered the routes used between the nodes, it queries the routers along the path for the link bandwidth between each pair of routers. It then periodically monitors the utilization of each segment by querying the octet counters for each interface on the routers. By default, the utilization is monitored every five seconds, although this is a configurable parameter.

The final responsibility of the SNMP Collector is representing the network with a virtual topology graph. When the collector discovers nodes connected to a shared Ethernet, or connected to routers it cannot access, it represents their connection with a virtual switch. In the case of shared Ethernet, this switch can be annotated with the bandwidth capacity and utilization of a shared Ethernet, representing its functionality with a standard graph format.

The SNMP Collector is implemented with Java threads, so it is capable of monitoring a number of routers and responding to many queries simultaneously.

3.2 Bridge Collector

By itself, the SNMP Collector is only capable of monitoring level 3 routed networks. While many research and campus networks are connected using only routers, the majority of LANs are implemented using level 2 switched Ethernet. Unfortunately, Ethernet switches do not provide explicit topology information as is provided by the IP routing tables. The Bridge Collector addresses this problem by determining the topology of the Ethernet switches using their forwarding databases. In the Remos architecture, it exists entirely to inform the SNMP Collector of the Ethernet topology, allowing the SNMP Collector to obtain the actual utilization measurements from the switches.

The Bridge Collector begins its topology discovery at startup. It is capable of determining the topology between any switches that implement the forwarding database specified in the RFC1493 Bridge MIB. The complete forwarding database is downloaded from each bridge in the Ethernet LAN. Using this information, the Bridge Collector determines how the bridges are connected, thus deriving the topology for the entire Ethernet network [22]. Once the Bridge Collector has determined how the bridges are connected, it then finds the location of all the hosts in its monitoring list. The Bridge Collector now has a complete picture of the level 2 topology provided by the Ethernet LAN.

3.2.1 Topology Maintenance

After completing the initial discovery phase, the Bridge Collector begins monitoring the location of all the hosts it is aware of on the LAN. It selects appropriate monitoring intervals for each host based on historical information indicating its likelihood to move, leave the network, or go down.

To update the location of a previously discovered host, the Bridge Collector performs a "quick check," which consists of sending a ping and immediately querying the bridge to which the host was last known to be connected. If the host has not moved, the bridge immediately reports it to be in the same position. If the host has moved, the Bridge Collector waits for a response to its pings and then searches through the bridges in the same manner as for an unknown host.

For previously unknown hosts, the Bridge Collector pings the host until it receives a response, which places the host's MAC address in the ARP cache of the machine and ensures that at least some bridges have seen the host. The Bridge Collector then begins at the root of the switch topology and locates the host in the topology.

The Bridge Collector code assumes that while hosts may move, the switches themselves do not move. This assumption is almost always true for any non-mobile network. In the event that system administrators reconfigure the network, the Bridge Collector must be restarted to repeat its initial topology discovery.

3.2.2 Communicating with the SNMP Collector

The Bridge Collector informs the SNMP Collector of LAN topology only when presented with a query. When the SNMP Collector receives a query from the Modeler for a host or hosts of which it is unaware, it forwards a topology query request to the Bridge Collector. The Bridge Collector replies to the SNMP Collector with the bridges and links used in the topology between the hosts in the query. The SNMP Collector caches its knowledge of these bridges and hosts, and adds them to the list of links that it monitors.

To simplify the implementation, the Bridge Collector does not maintain state for which portions of the topology the SNMP Collector has queried. Instead, when it observes a host it is monitoring move, it sends an *invalidate* message to any SNMP Collector with an open connection. If the SNMP Collector had previously cached information about that host, it will delete it and optionally request the new information. If it is unaware of the moved host, then it merely ignores the *invalidate* message.

3.2.3 Runtime costs

Analyzing the runtime of the Bridge Collector is difficult because the actual topology algorithms runs very quickly. The time-consuming portion of the Bridge Collector is the process of retrieving information from the bridges without overloading them (which is sometimes perceived as an attack by the bridges). The initial download of the forwarding databases from the bridges takes several minutes on a typical 100 host network and somewhat under two hours on a 3000 host network. Because the Bridge Collector answers most queries from its cache of topology information, after the initial discovery phase is run, this cost no longer matters to the runtime of Remos.

The most important cost of the Bridge Collector is the time it takes to reply to a query from the SNMP Collector. In cases where the Bridge Collector has all hosts being queried in its cache, and the query does not require more recent updates than are currently cached, the response is immediate. If the query requests a host the Bridge Collector has not seen before, or requests more information than is currently cached, then the Bridge Collector must search for or update the location of each host.

Performing a quick check to confirm a host is in the same location as before is of minimal cost, only the time it takes to issue and receive a response to one SNMP query. Discovering the location of an unknown host can take longer, as the time for this operation depends on the time before a ping response is received, the time to send an SNMP query to each bridge, and the number of bridges that must be queried before the host is located in the topology. Typically, however, an active host can be located within one or two seconds. If a host is not responding to pings, the Bridge Collector waits a predetermined time before reporting the host as unavailable.

For queries of very large numbers of new hosts, the search could be parallelized, however that optimization has not been made to reduce the load placed on the bridges.

3.3 Benchmark Collector

While SNMP offers excellent information, Remos generally cannot obtain SNMP access to network information for WANs or other networks where the Remos administrator does not have an account on a machine. In that case, we fall back on a Benchmark Collector, which does explicit testing to determine the performance characteristics of the network. A Benchmark Collector is run at each site where an SNMP Collector is. When a measurement of performance between multiple sites is needed, the Benchmark Collector exchanges data with the Benchmark Collector running at the other site of interest. By measuring the bandwidth and latency between sites, the Benchmark Collectors determine the performance of the links connecting the network and report this information to the Master Collector. This technique is similar to the techniques used by NWS [39].

To measure bandwidth we use Nettest [5], and for delay we use traceroute. The Benchmark Collector must have permission to run code on the endpoints because some benchmarks require that specific programs are run on the source or destination. The Benchmark Collector is also expensive: the algorithm is N^2 with a large constant (time to execute a benchmark). In practice we only use this collector as a “WAN collector,” so endpoints correspond to subnets and N is in practice small. Also, in an environment with many Remos users, we would rely on caching to keep track of connectivity to a larger number of subnets.

There are some interesting differences between the SNMP and the benchmark collector:

- The Benchmark Collector can add a considerable load to the network traffic (while it is probing). The SNMP collector on the other hand, adds very little traffic but places an additional load on the routers since they have to respond to SNMP queries.

- The Benchmark Collector measures user-level performance. In contrast, the SNMP collector collects historical data on bandwidth use, which then has to be translated into an estimate of how much bandwidth a new user can expect. While our results show that this is possible, more experience is needed to show how accurately this can be done across a range of networks.

- The information from the Benchmark Collector is less detailed. For example, suppose we have a three node query (nodes A, B, C). If benchmarks show that the A-C bandwidth is 4 Mbs and

B-C is 5 Mbs, the Benchmark Collector cannot predict what the result would be if A-C and B-C stream data at the same time. An SNMP collector would return a logical topology that shows where the bottleneck is, i.e. whether it is shared between the two flows or not.

Overall, our experience indicates that SNMP collectors are less intrusive and provide more accurate information, although it is difficult to evaluate the impact of the SNMP queries on router performance.

3.4 Master Collector

Queries involving a single subnet can be handled entirely by the collector responsible for that subnet. However, distributed applications that need to obtain information about multiple subnets cannot get all the information they need from a single collector. The Master Collector was designed to solve this problem. Despite its name, a different Master Collector can be used in each network where Remos applications are running.

The Modeler used by the Remos application submits a query to its Master Collector. When a query is received, the first task of the Master Collector is to identify the IP networks and subnets involved in the query, along with the associated SNMP and Benchmark Collectors for those networks. The current implementation of the Master Collector uses a database to keep track of all the collectors it knows about. The collectors register with the database, giving information that includes the type of collector and the domain it is responsible for, represented by one or more subnet addresses and netmasks.

The Master Collector uses the IP addresses of the hosts requested to identify the collectors needed to answer the query. It chooses the SNMP Collectors that are responsible for the domains containing hosts listed in the query (leaf subnets), and then also picks Benchmark Collectors to get information about the wide area networks between the leaf subnets.

Once the Master Collector has identified relevant collectors, it breaks up the query and sends the correct piece to each collector. This problem is trivial if we know the IP addresses of (the relevant ports on) edge routers that connect the subnets. However, this information is not part of the query request. One solution is to have the Master Collector discover this information, e.g., using traceroute, or have it request the information from the data collectors. This approach has the drawback that the Master Collector has to learn about and keep track of information that is subnet specific. Instead, we place the responsibility for identifying edge routers with the data collectors. For each data collector, the Master Collector formulates a request that contains not only the endpoints of the subnet it's responsible for, but also one endpoint from each of the other leaf subnets. Then the Master Collector uses a separate thread to send the queries to all the collectors in parallel.

When an SNMP Collector gets a query from the Master Collector, it uses the IP addresses of the hosts in the query to determine if any of them are outside its domain. If there are any, it discovers the routes within its own network up to and including its edge router. In its response to the Master Collector, it returns both the relevant local topology and information about the edge router. Benchmark Collectors have to replace the endpoints outside their domains with the addresses of nodes in the same subnets that can be used to run benchmark programs (e.g., a peer Benchmark Collector).

Once all the queries have been sent out, the Master Collector waits for each thread to finish receiving the responses from all the collectors. Then the Master Collector merges the results of each query and removes duplicate node and link information. In this process, node and link information from an SNMP Collector takes precedence over the same information from a Benchmark Collector since the SNMP Collector's data is more accurate. Furthermore, if the response from an SNMP Collector contains an edge router, it will also include a list of all of the IP addresses associated with that edge router. This is important because a router that is identified by one IP address in the response from a Benchmark Collector might be identified by a different IP address in a response from an SNMP collector. The Master Collector uses the edge router information to determine that these two different IP addresses belong to the same node.

Once the query results have been merged, the response that is sent back to the Modeler is a seamless picture of the network connecting the hosts listed in the original query.

3.5 Modeler

The application queries information about the network via the API that is provided by the Remos Modeler. The Modeler is a single-threaded entity that opens and maintains a TCP connection to its Master Collector. The location and the port of the Master Collector are configured at application startup. The Modeler is available in C and Java implementations.

The Modeler uses the raw network information provided by the collector to build up a topology structure of the underlying network. It connects nodes and links to form a network graph, and it may add virtual switches to simplify the network topology.

The Modeler implements both the topology and flow queries. Although our intention was to allow information that could not be represented as part of the topology response to be used to answer flow queries, in practice we have not been able to explore this option because we have collected little information that cannot be represented in our topology graphs. However, we have found that the most important advantage of the flow queries is their ability to simplify application programming.

The Modeler's implementation of the flow queries can be quite useful for applications. The algorithm assumes that the application will receive the unutilized or available bandwidth on each link of the network and uses the max-min fair share algorithm [16] to determine how the messages presented in the flow query will use the available bandwidth. By implementing this complex approximation in the Modeler, a Remos application developer can quickly form a complex query and rely on the Modeler to perform the analysis, rather than performing it within the application. The API allows the user to specify fixed-rate, proportionally adjustable, and best-effort flows to mimic the behavior of a variety of application types.

The topology information returned by the Modeler is summarized to describe only the portion of the network required by the list of nodes it is interested in (e.g., the end systems of a network). The routers and bridges used to connect those hosts are returned, and large uninteresting parts of the network (a series of bridges or routers with no queried hosts attached) are compressed with virtual switches representing their bandwidth and latency restrictions to simplify the topology description.

All Remos queries contain a *age* field. This age specifies the allowed age of a measurement. That is, an application can specify how old a measured value can be to be considered accurate for a given application. Surveying a LAN may require new measurements at seconds or minutes intervals, whereas WAN measurements may only be required at intervals of minutes. This age parameter allows the customization of Remos to the needs of different applications.

4 The Grid and related work

Grid-based distributed computing has brought about the need for systems that monitor and predict both application and resource information. In addition to Remos, a number of systems have been developed that address various information needs of grid applications [37, 33, 25, 34]. One of the principle differences between Remos and these systems is that Remos was intended to provide applications with end-to-end data derived from component sensors across the network, and integrate these measurements with traditional sensor-based data and end-to-end benchmarks.

While other projects have developed techniques to derive Internet topology [11, 31, 17, 24], Remos was the first to integrate LAN topology information with performance measurements. Because the link-sharing found on LANs can have a profound influence on an application's performance, providing this information as well as site-to-site performance measurements has proven useful for predicting application performance.

Research into resource prediction has focused on determining appropriate predictive models for host behavior [10, 38, 26], and network behavior [35, 1, 12]. The RPS toolbox used by Remos incorporates many of the models studied by this research. RPS is also available as an independent tool for other research requiring predictive models.

One of the products of the Global Grid Forum is the Grid Monitoring Architecture [32], which is being developed by the performance working group. In this architecture each Collector is a producer. The Master Collector is a joint consumer/producer, as its responsibility is to contact the other collectors as a consumer, before aggregating the information together and providing it to another layer. Although we view the Modeler as a consumer, it could also be another joint consumer/producer, providing end-to-end performance predictions using the component data available from the collectors as a service to other applications. In the Remos architecture, the collectors also implement a limited form of directory service to locate each other. The directory service of the GMA would be natural to use for this purpose.

Overall, we find that the Remos architecture is quite compatible with the GMA, and should interoperate well with other monitoring systems once appropriate interfaces are designed. The biggest challenge presented by Remos is describing the information available through it in the directory service. Because Remos provides end-to-end data derived from component-level data, it would be difficult to describe all possible measurement pairs in the directory service. However, several solutions to this problem have been discussed by the Grid Performance Group and others, therefore we are confident that the GMA's directory service will support Remos well when fully developed.

Associated with the format of the GMA is the method used to store grid information in the first place. Significant discussion is ongoing about the advantages and disadvantages of a hierarchical approach, such as MDS-2 [6], or a relational approach [7]. Both proposals present models that are capable of associating Remos with the resources it monitors, which is the fundamental requirement Remos has for a directory service.

5 Evaluation

The flexibility and portability aspects of Remos have been discussed in other Sections, especially Section 3. Here, we discuss scalability and functionality results for the different Remos components and the system as a whole.

5.1 LAN scalability

In a first set of experiments, we look at the response time of the SNMP Collector deployed in the local area network in the School of Computer Science at CMU. The network is a very large bridged network so the Bridge Collector must also be used to get complete topology information.

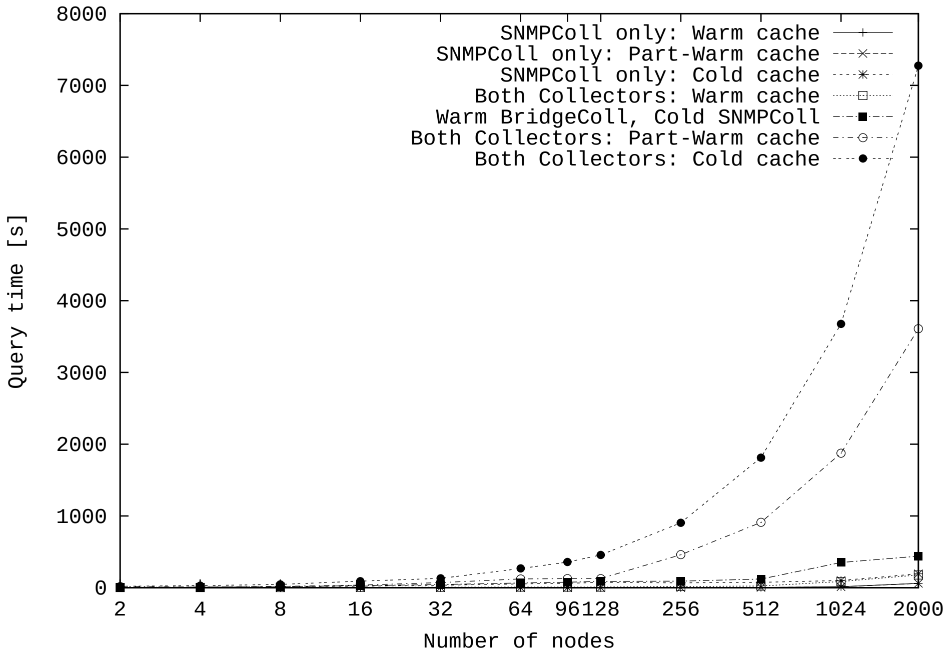


Figure 4: SNMP and Bridge Collector response times

Figure 4 shows how the response time increases with the number of nodes specified in the query. All measurements are averaged over at least 10 runs. There are seven scenarios; the first 3 were run with the SNMP Collector only, and the last four included results from the Bridge Collector as well.

- *SNMPColl only: Cold cache*: the SNMP Collector has just started up so it has no information on either the static topology or the dynamic performance metrics.
- *SNMPColl only: Part-Warm cache*: the SNMP Collector has some cached information, namely the result from the previous query (typically about 1/2 of the data).
- *SNMPColl only: Warm cache*: the SNMP Collector has both the static and dynamic data in its cache.
- *Both Collectors: Cold cache*: Both the SNMP Collector and Bridge Collector have just started up and have no information on either the static topology or the dynamic performance metrics.
- *Both Collectors: Part-Warm cache*: Both the SNMP Collector and Bridge Collector have some cached information, namely the result from the previous query (typically about 1/2 of the data).
- *Both Collectors: Warm cache*: Both the SNMP Collector and Bridge Collector have the static and dynamic data cached.
- *Warm BridgeColl, Cold SNMPColl*: The Bridge Collector has all the topology data already cached, but the SNMP Collector has just started up and has no static or dynamic information.

We can make a number of observations. First, it clearly pays off to cache information. The warm-cache results are a factor of three or more better than the cold cache results. Second, the worst case cost of a cold cache query is $O(N^2)$. However, we implemented a number of optimization that reduce

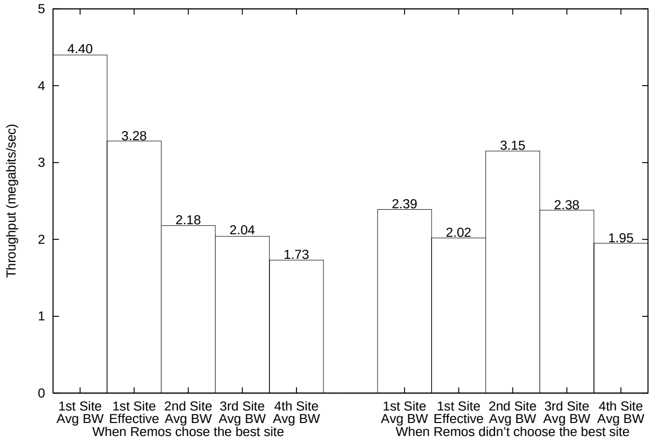


Figure 5: Average transfer rates for well-connected sites

the cost, especially for large N; the measurements show the effect. Finally, the cost of warm-cache queries should be $O(N)$. We see that the cost actually grows faster, probably because of increasing memory requirements which reduce execution efficiency.

5.2 Mirrored server experiment

One simple use of Remos is to help applications choose a remote server based on available network bandwidth. We have written a simple application that reads a 3MB file from a server after using network information obtained from Remos to choose the best server from a set of replicas [23].

We ran two sets of mirror experiments: one that used remote sites with good network bandwidth, and another experiment using sites with poor bandwidth. For the first experiment, we ran the application at Carnegie Mellon and servers at Harvard, ISI, Northwestern University (NWU), and ETH. Averaged over all 108 trials, we observed an average throughput of 2.03 Mbps from Harvard, 2.15 Mbps from ISI, 4.11 Mbps from NWU, and 1.99 Mbps from ETH. For the second experiment, we ran the application at Carnegie Mellon and the servers at the University of Coimbra, Portugal (average throughput 0.25 Mbps), the University of Valladolid, Spain (average throughput 1.02 Mbps), and the third server was run on a machine in Pittsburgh connected via a DSL link with a maximum upstream bandwidth of 0.08 Mbps. We ran 72 trials using the poorly connected sites.

In order to be able to evaluate the quality of the Remos information, we modified the application to read the file from all three servers, starting with the server that, according to Remos, has the best network connectivity. In the first experiment using well connected sites, Remos chose the remote site that ended up having the fastest transfer rate 83% of the time. Figure 5 shows the difference in throughput between the 1st place site Remos chose and the other 3 sites. The left half of the graph shows the throughput when Remos chose the best site, and the right half of the graph shows the throughput when Remos did not choose the fastest site. The second bar in each group shows effective bandwidth for the site Remos chose. This bandwidth includes the time it took to get an answer back from the Remos system.

In the second experiment, which used sites that were not well connected to CMU, Remos chose the remote site that ended up having the fastest transfer rate 82% of the time. Figure 6 shows the difference in throughput between the 1st place site Remos chose and the other 2 sites. As in Figure 5, the left half of the graph shows throughput for when Remos chose the best site, and the right half shows throughput

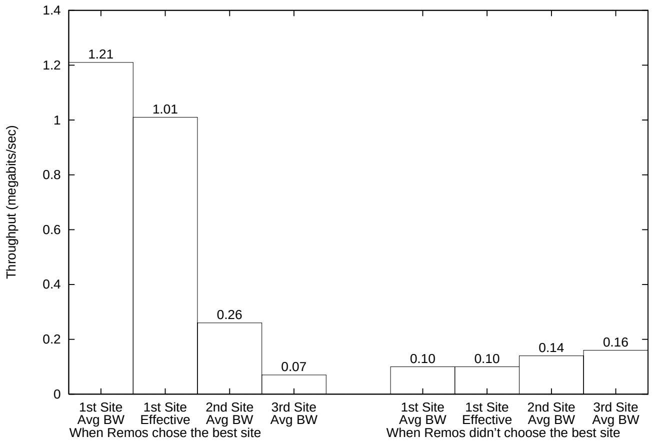


Figure 6: Average transfer rates for poorly-connected sites

for when it didn't. The second bar in each group once again shows the effective bandwidth for the site Remos chose.

We included the effective bandwidth measurement to show that even though it takes some time to consult Remos to choose a server, performance is still better than choosing one of the slower sites. These experiments also show that using Remos to pick a site is effective even when all of the sites have poor connectivity.

5.3 Application Experiment—Video transfer

In the previous example, Remos used the available bandwidth as a metric. This metric, however, does not always directly correspond to the metric in which the application is interested. For example, the quality of a video application that downloads and plays the video in real time may be rated by the number of correctly received frames at the client [14]. This experiment shows how the Remos metric corresponds to such an application-defined metric.

For the experiment, the video client is located at ETH. Servers from which the videos can be downloaded are placed at different locations in Europe and the U.S (see Table 1). The video server is able to adapt the outgoing video stream to the available bandwidth by intelligently dropping frames of lower importance [14]. It thereby maximizes the numbers of frames that are transmitted correctly.

The bandwidth of the local server at ETH is an order of magnitude higher than EPFL, which in turn is an order of magnitude larger than the others.

Before downloading a video, the client issues a Remos query to measure the available bandwidth

Server Location	average bandwidth	standard deviation
ETH Zurich	63.1	5.61
EPFL Lausanne	3.03	0.17
CMU	0.50	0.28
University of Valladolid, Spain	0.37	0.28
University of Coimbra, Portugal	0.18	0.07

Table 1: Server location, the available bandwidth and the standard deviation, measured by Remos.

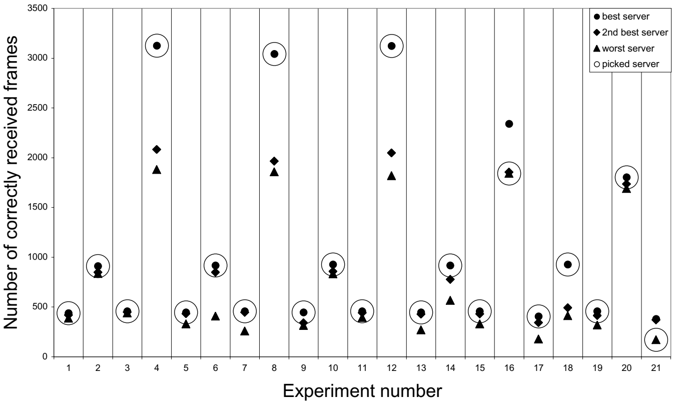


Figure 7: Server picked according to the measured bandwidth (large circle) and the number of correctly received frames in the following download.

to all servers. It then downloads the movie from the server with the best connectivity. To compare the results, the client subsequently also downloads the same video from all other sites in the decreasing order of the available bandwidth. This experiment was run several times within 24 hours with different movies.

Figure 7 shows the number of correctly received frames for each experiment. The server that is selected first according to the bandwidth measurements by Remos is indicated by a large circle. The figure excludes the results from ETH and EPFL because the bandwidth is always higher that the bandwidth required by the application. If ETH is included, the client always picks the server at ETH. The downloaded video does not lose any frames. If ETH is excluded, the system always selects EPFL and also gets the video without dropped frames. If both ETH and EPFL are excluded, the client-perceived quality corresponds to the reported bandwidth in 90% of the cases, i.e. the client receives the most frames correctly from the server with the highest bandwidth. In the 2 cases where the best server is not picked, an inspection shows that the server only sent about half of the packets, probably due to a high load on the server.

The results show that the available bandwidth corresponds well to the application-perceived quality. However, the two wrong picks indicate that the bandwidth alone does not guarantee a good video download. Other parameters may influence the download as well and must be taken into account.

Figure 8 shows 2 experiments in detail. The same movie is downloaded from 2 different servers, a local server with a high- bandwidth connectivity and the remote server with a limited bandwidth. Each packet that arrives at the client is timestamped and the application-perceived bandwidth is calculated as the average over 3 different time intervals: 1, 2 and 10 seconds.

The download from the local server is not limited by the bandwidth. The average over small intervals shows that the bandwidth requirements vary over time. These fluctuations can be explained by the variation of the movie content. Averaging the bandwidth over a larger interval smooths the variations.

For the remote server experiment, the bandwidth measured by Remos is the horizontal line at 0.15Mbps. This line corresponds well to bandwidth measured by the application if it is averaged over a large interval. The 10 seconds interval corresponds to the time interval that Remos uses to measure the available bandwidth. Calculating the average over smaller intervals shows higher fluctuations. The reported bandwidth does not correspond well to the bandwidth of these small intervals.

This experiment demonstrates that optimal results can only be achieved when not only the metric of

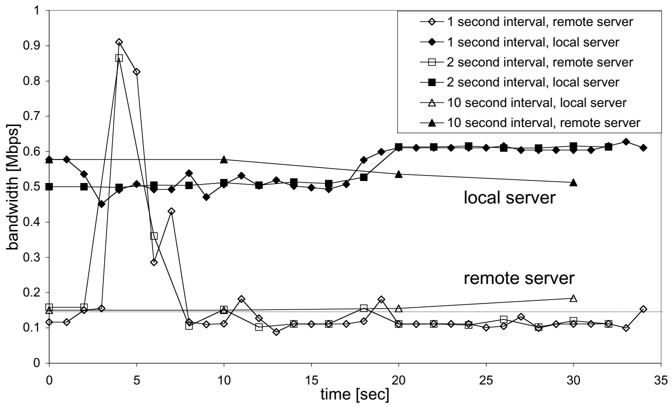


Figure 8: The bandwidth measured by the application, averaged over different time intervals, and the bandwidth reported by Remos.

Remos and the application correspond, but also when the interval over which the bandwidth is reported matches the varying needs of the application. Although Remos is not currently able to fully address these points, this experiment still shows that Remos is well able to provide useful guidance to this type of application. It can help the video client to select the server. In addition, it might similarly be used to determine alternate servers and routes for a dynamic video handoff [19].

5.4 Support for Application-layer Routing

The traditional IP routing algorithm minimizes the number of hops in a connection. However, many applications, e.g., multimedia applications, are sensitive to bandwidth rather than the number of hops. Studies, e.g., by Savage et al. [28], have shown that alternative paths can be constructed in the Internet which provide a better bandwidth than the default routing path.

Recent approaches in overlay network advocate the routing via end-systems in a network [4, 40, 18, 27, 3]. That is, data is sent from a sending host via one or multiple third-party host inside the network to the final destination. Default IP routing is used to tunnel data between overlay nodes.

We claim that Remos is well suited to support such overlay networks because Remos provides the necessary network information while hiding the details of the information gathering from the overlay structure, e.g., the use of the right tool to measure bandwidth in a LAN or in a WAN.

To show this suitability, we have implemented an application-specific routing protocol for a collaborative application using Remos. The collaborative application must transmit data of different types, and hence of different sizes, from one sender to multiple receivers. The set of collaborating nodes forms a kind of an overlay network. Depending on the size of the data and depending on the available resources along the paths, the application must decide which path is the best. The algorithm for the path selection uses the following equation to determine the transmission time of a data item:
$t_{transmission} = \frac{datasize}{bw} + latency$
For small amounts of data, as for text, the latency is the dominant factor, whereas the bandwidth becomes more important with the increasing data size. The current values of the available bandwidth and the latency can easily be gathered using Remos.

To show the effects of the routing, we perform an Internet experiment. We have collected a set of Internet traces between several hosts in Europe and the U.S.. Every host can act as sender, proxy or receiver. Alternative paths are constructed by concatenating two path, as proposed by Savage et

src/dst	ETH	EPFL	CMU	NWU	UVA	UFMG
ETH		165.5	305.3	358.4	290.1	182.7
EPFL	1180.6		331.8	385.4	316.5	146.8
CMU	514.8	349.8			686.7	
NWU	667.1	406.8	390.6		1365.2	1037.2
UVA	401.5			347.2		384.4
UFMG	150.5	167.5	1125.2		355.6	

Table 2: Data threshold [KB] for which the alternative path has a better download time than the default routing path.

al. [28]. The bandwidth of an alternative path is the minimum of the two individual bandwidths and the latency is the sum of the individual latencies. As in [28], we found that alternative paths exist with a significantly better bandwidth than the default routing path.

The gain of alternative path routing can be shown by comparing the transmission time of the default routing path to the best alternative path. The default path has typically a low latency since it crosses a small number of hops. For small data items, the default path is preferably used, whereas alternative paths are often used for large data sizes because of the greater bandwidth capacity. That is, there is a threshold above which the alternative path provides a better connectivity than the default routing path. The threshold depends on the differences in the latency and the bandwidth.

Table 2 shows this data threshold for 6 hosts in the experiment. The numbers, expressed in KB, are average values of 250 measurements. A field is empty if the default path is better than the alternative paths even for large data sizes. The results show that most thresholds lie between 400 KB and 1.5 MB. According to these results, text and small images should be sent over the default path whereas larger images and video should be sent over the alternative path. For large data sizes, such as multimedia streams, the increased bandwidth availability can have a significant effect on the quality of the video.

The effects on the application-perceived transmission time are shown in Figure 9. This figure compares the default path with the best alternative path, as a function of the destination host and the data size. The logarithmically scaled y-axis denotes the transmission time in seconds. For the first host (EPFL), e.g., the default path is always better than the alternative path. For all others destinations, however, the default path is only better for small data sizes because it has a better latency. As soon as

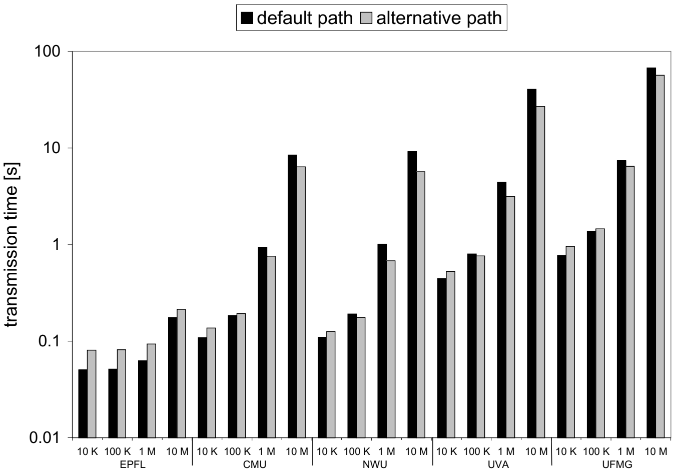


Figure 9: Transmission time over default and alternative paths, as a function of the data size.

the data size increases, the better bandwidth availability of the alternative paths pays off.

The conclusion from this experiment is that Remos is able to provide resource information about a network that can be used for application-layer routing. This information can be used for overlay networks. However, it can also be used directly by network-aware applications to deploy application-specific routing. Overlay networks are typically transparent to an application and route the data based on a single metric. However, this experiment also stresses the importance of separate routing schemes, e.g., depending on the size of the data.

6 Reflections

In this section we try to capture what we learned about resource monitoring systems in the last four years. While these comments are of course quite subjective, we hope our thoughts will help others working in the same area.

6.1 What worked

The first step in the Remos development was the definition of the Remos API [20]. We kept the API simple and focused on simple network performance properties that are of interest to applications. While the API supports several performance metrics, our initial implementation focused on bandwidth. Our experience suggests that these were the right design decisions. The API provides a good balance between simplicity and amount of information provided. The API works for all the networks we have encountered so far, i.e. it is network independent. Finally, bandwidth is by far the most important metric for many applications.

Underneath the fixed API, we decided to use a systems architecture that was modular and extensible. This choice also worked well. Our initial system consisted of just an SNMP Collector, and later we were able incorporate Benchmark, Bridge, and Master Collectors, without changes to the API. Because of the modular design, we were also able to use different data gathering techniques for different networks. While benchmarks are an effective way of collecting bandwidth information, they are too expensive and intrusive for many types of networks, and we need to utilize more lightweight techniques such as the SNMP Collector.

6.2 What needs more work

We discovered that one of the most difficult challenges in building a resource monitoring system is making the system easily portable and robust across diverse environments. Our goal was that Remos must be able to report resource information for any networked environment, with minimal, if any, manual configuration. In practice, we discovered that bringing up Remos in a new environment can be challenging. Problems range from: network features that we had not encountered before (e.g. VLANs), misconfigured network elements, and non-standard features (e.g. non-standard SNMP implementations). To some extent, these portability problems should not be a surprise: there are many network vendors and many ways to configure a network, so this problem is inherently hard. However, we have learned that, while useful, neither the SNMP MIBs nor the vendors commitment to support them properly are sufficient to allow the development of totally portable utilities, such as Remos, that automatically parse and use the information presented by SNMP agents. After many years, we still encounter hardware from major networking vendors with new mistakes, omissions, or unusual interpretations of MIB standards that require a new kludge or test in Remos to ensure support. Improving the robustness and portability of Remos is an ongoing effort.

Remos currently relies on SNMP MIB and benchmark information. Many other sources of information could be tapped, including measurements collected by ISPs for traffic engineering purposes, application-level information [30], and network information that is collected in vendor-specific ways. The emerging DMTF CIM standards offer the promise of better portability. Also, for certain types of networks, such as shared Ethernets, we need better techniques for performance prediction.

There are many ways in which the Remos system could be improved. A first issue is that communication between the Remos components is currently based on a single-purpose, ASCII based protocol. While this was convenient for debugging and development, using standard solutions such as SOAP or XML over HTTP would ease interoperability and extensions. Second, we are working on a new collector for wireless networks, and improving our existing collectors to support mobile hosts. Third, the Benchmark Collector could be improved by adding support for other types of benchmarking, for example, lighter weight probing techniques based on packet pairs [15]. Finally, it would be interesting to evaluate techniques for sharing and caching of prediction results, and to explore how well this allows the architecture to scale to large numbers of diverse applications.

6.3 When is Remos most useful?

Many applications (e.g., video streaming) only care about the performance of a single flow between two nodes that are currently exchanging data. In such cases, Remos is probably overkill, because the application can get the required information more cheaply and more accurately by monitoring its own performance [2]. However, for applications that have to select a server from a set of options, that have to select and assign a set of compute nodes with certain connectivity properties, or that have to make critical configuration decisions (e.g. to use remote or local execution, to use video plus audio, or audio only), Remos provides explicit connectivity information that would be difficult and expensive to collect otherwise [13].

We end up with a model of an adaptive application that combines two types of adaptation using different information sources. The application performs node and network selection, and high-level self-configuration based on explicit, Remos-provided resource information. This type of decision is typically made when the application starts up, or, for long running applications, periodically during execution. During execution, the application can fine-tune its performance based on direct measurements. This model is in part driven by the cost of adaptation: adaptation that does not involve changes in node usage can be cheap and fast, while changing nodes or high-level application configuration will be more expensive.

7 Conclusions

The Remos architecture is designed to provide the information needed by Grid applications across many diverse environments. Remos has been implemented and tested in a variety of different networking environments and has been used to support a variety of applications, thus demonstrating the flexibility and portability needed for emerging applications. We have used Remos to support both large numbers of machines at a single site as well as to support several sites simultaneously and find that the architecture scales well. While our architecture differs somewhat from the proposed Grid Monitoring

Architecture (GMA), a comparison indicates both that Remos should interact well with GMA-based monitoring tools and that the future development and performance of tools such as Remos will be easily supported within the framework of the GMA.

The availability of the Remos API allows application developers to address new aspects of the environment. Without sacrificing portability for performance (or vice versa), it is now possible to develop applications that use information about the status of the network to determine the next adaptation steps. The availability of and experience with the Remos architecture backs up the claims made by the Remos API and provides a practical demonstration that it is possible to find a workable compromise between the conflicting objectives of functionality, performance, and portability. As networks grow in complexity, and as efforts like the Grid bring more application developers into this domain, the interest in infrastructure systems like Remos is likely to increase. Dealing with and obtaining performance information will remain an important topic; Remos provides both a set of abstractions and an architecture that have proven their value in practical settings.

References

[1] S. Basu, A. Mukherjee, and S. Klivansky. Time series models for internet traffic. Technical Report GIT-CC-95-27, College of Computing, Georgia Institute of Technology, February 1995.

[2] J. Bolliger and T. Gross. Bandwidth monitoring for network-aware applications. In *Proc. 10th IEEE Symp. High-Performance Distr. Comp.*, San Francisco, CA, August 2001. IEEE CS Press.

[3] Y. Chu, S. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the Internet using an overlay multicast architecture. In *Proceedings of ACM SIGCOMM ’01*, pages 55–67, San Diego, CA, Aug. 2001.

[4] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *ACM Sigmetrics 2000*, pages 1–12, Santa Clara, CA, June 2000.

[5] I. Cray Research. Nettest Networking Benchmark. ftp://ftp.sgi.com/sgi/src/nettest, 1992.

[6] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing (HPDC 10)*, August 2001.

[7] P. Dinda and B. Plale. A unified relational approach to grid information services. GWD-GIS-012-1. http://www.cs.northwestern.edu/˜pdinda/relational-gis/, February 2001.

[8] P. A. Dinda and D. R. O’Hallaron. An evaluation of linear models for host load prediction. In *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC ’99)*, pages 87–96, August 1999. Extended version available as CMU Technical Report CMU-CS-TR-98-148.

[9] P. A. Dinda and D. R. O’Hallaron. An extensible toolkit for resource prediction in distributed systems. Technical Report CMU-CS-99-138, School of Computer Science, Carnegie Mellon University, July 1999.

[10] P. A. Dinda and D. R. O’Hallaron. Host load prediction using linear models. *Cluster Computing*, 3(4), 2000. An earlier version appeared in HPDC ’99.

[11] R. Govindan and H. Tangmunarunkit. Heuristics for internet map discovery. In *IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[12] N. C. Groschwitz and G. C. Polyzos. A time series model of long-term NSFNET backbone traffic. In *Proceedings of the IEEE International Conference on Communications (ICC’94)*, volume 3, pages 1400–4, May 1994.

[13] T. Gross and P. Steenkiste. A perspective on network/application coupling. In *Proc. 8th NOSSDAV Workshop (Network and Operating System Services for Digital Audio and Video), www.nossdav.org/1998/-tech*, 1998. Short paper.

[14] M. Hemy, P. Steenkiste, and T. Gross. Evaluation of adaptive filtering of MPEG system streams in IP networks. In *Proceedings of the IEEE International Conference on Multimedia and Expo 2000 (IDME 2000)*, pages 1313–1317, New York, NY, Aug. 2000.

[15] N. Hu and P. Steenkiste. Estimating available bandwidth using packet pair probing. Technical Report CMU-CS-02-166, Carnegie Mellon University, School of Computer Science, September 2002.

[16] J. M. Jaffe. Bottleneck flow control. *IEEE Transactions on Communications*, 29(7):954–962, July 1981.

[17] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. On the placement of internet instrumentation. In *IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[18] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole. Overcast: reliable multicasting with an overlay network. In *Proceedings of the 4th Symposium on Operating System Design and Implementation (OSDI 2000)*, pages 197–212, San Diego, CA, Oct. 2000.

[19] R. Karrer and T. Gross. Dynamic handoff of multimedia streams. In *Proc. of NOSSDAV '01*, Port Jefferson, NY, 2001.

[20] B. Lowekamp, N. Miller, D. Sutherland, T. Gross, P. Steenkiste, and J. Subhlok. A resource query interface for network-aware applications. *Cluster Computing*, 2(2):139–151, 1999.

[21] B. Lowekamp, D. O'Hallaron, and T. Gross. Direct network queries for discovering network resource properties in a distributed environment. In *Proc. 8th IEEE Symposium on High-Performance Distributed Computing (HPDC-8)*, Redondo Beach, CA, Aug. 1999.

[22] B. Lowekamp, D. R. O'Hallaron, and T. Gross. Topology discovery for large ethernet networks. In *Proceedings of SIGCOMM 2001*. ACM, August 2001.

[23] N. Miller and P. Steenkiste. Collecting network status information for network-aware applications. In *IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[24] K. Obraczka and G. Gheorghiu. The performance of a service for network-aware applications. In *Proceedings of the ACM SIGMETRICS SPDT'98*, October 1997. (also available as USC CS Technical Report 97-660).

[25] R. Ribler, J. Vetter, H. Simitci, and D. Reed. Autopilot: Adaptive control of distributed applications. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC 7)*, pages 172–179, 1998.

[26] M. Samadani and E. Kalthofen. On distributed scheduling using load prediction from past information. Abstracts published in Proceedings of the 14th annual ACM Symposium on the Principles of Distributed Computing (PODC'95, pp. 261) and in the Third Workshop on Languages, Compilers and Run-time Systems for Scalable Computers (LCR'95, pp. 317–320), 1996.

[27] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: informed Internet routing and transport. *IEEE Micro*, 19(1):50–59, Jan. 1999.

[28] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The end-to-end effects of Internet path selection. In *Proceedings of ACM SIGCOMM '99*, pages 289–299, Boston, Massachusetts, Aug. 1999.

[29] W. Stallings. *SNMP, SNMPv2, and RMON*. Addison-Wesley, 2nd edition, 1996.

[30] M. Stemm, S. Seshan, and R. Katz. Spand: Shared passive network performance discovery. In *USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, June 1997.

[31] W. Theilmann and K. Rothermel. Dynamic distance maps of the internet. In *IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[32] B. Tierney, R. Aydt, D. Gunter, W. Smith, V. Taylor, R. Wolski, M. Swany, and The GMAWG. A grid monitoring architecture. DRAFT. http://www-didc.lbl.gov/GGF-PERF/GMA-WG/, August 2002.

[33] B. Tierney, B. Crowley, D. Gunter, M. Holding, J. Lee, and M. Thompson. A monitoring sensor management system for grid environments. In *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC 9)*, pages 97–104, 2000.

[34] B. Tierney, W. Johnston, B. Crowley, G. Hoo, C. Brooks, and D. Gunter. The netlogger methodology for high performance distributed systems performance analysis. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC 7)*, pages 260–267, 1998.

[35] R. Wolski. Forecasting network performance to support dynamic scheduling using the network weather service. In *Proceedings of the 6th High-Performance Distributed Computing Conference (HPDC97)*, pages 316–325, August 1997. extended version available as UCSD Technical Report TR-CS96-494.

[36] R. Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1(1):119–132, 1998.

[37] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *J. Future Generation Computing Systems*, 15(5-6):757–768, October 1998. Published also as UCSD Technical Report Number CS98-599.

[38] R. Wolski, N. Spring, and J. Hayes. Predicting the CPU availability of time-shared unix systems. In *Proceedings of the Eighth IEEE Symposium on High Performance Distributed Computing HPDC99*, pages 105–112. IEEE, August 1999. Earlier version available as UCSD Technical Report Number CS98-602.

[39] R. Wolski, N. Spring, and C. Peterson. Implementing a performance forecasting system for metacomputing: The network weather service. In *Supercomputing '97*, 1997.

[40] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: an infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of Berkeley, CA, Apr. 2001.