

TECHNICAL REPORT 1883
April 2002

Managing Quality of Service within Distributed Environments

SSC San Diego

Teknowledge

The Open Group Research Institute

System/Technology Development Corporation

Approved for public release;
distribution is unlimited



**SPAWAR
Systems Center
San Diego**

**SSC San Diego
San Diego, CA 92152-5001**

SSC SAN DIEGO
San Diego, California 92152-5001

Commanding Officer

Executive Director

ADMINISTRATIVE INFORMATION

The work described in this report was performed for the Defense Advanced Research Projects Agency (DARPA) Information Technology Office (ITO) by the Advanced Concepts and Engineering Division (Code 241), SSC San Diego; Teknowledge Corporation, Palo Alto, CA; The Open Group Research Institute, Woburn, MA; and System Technology Development Corporation, Herdon, VA.

ACKNOWLEDGMENT

This report was compiled from research and experiments performed within the Quorum Integration, Testbed and Exploitation (QUITE) project. This DARPA ITO Quorum project effort is structured as follows: DARPA ITO, Sponsor; SSC San Diego, Technical/Contracting Lead; Teknowledge Corporation, Integration Lead; System/Technology Development Corporation (S/TDC) and the Open Group (TOG); Integration.

The individuals who directly contributed to this specific effort included Gary Koob, Quorum Program Manager (DARPA ITO); John Drummond and Al Sandlin (SSC San Diego); Neil Jacobstein, Adam Pease, Lou Coker, Jeff Vu, Joe Marclino, and Jim Reynolds (Teknowledge); Mustafizur Rahman and Jim Carroll (TOG); and Arthur Robinson, Manoj Srivastava, Amarendranath Vadlamudi, and Shivakumar Patil (S/TDC).

Active participation by all the QUITE project personnel included extensive discussion regarding the research and experiments conducted within the QUITE project. This document has drawn upon the research and the numerous quality of service management experiment design planning meetings and other related informal writings developed throughout the QUITE project effort.

AwardBios™ is a trademark of Phoenix Technologies.

ACE™ and TAO™ are trademarks of Washington University and the University of California, Irvine.

Intel® and Pentium® are registered trademarks of the Intel Corporation.

3Com® is a registered trademark of the 3Com Corporation.

Windows NT® is a registered trademark of the Microsoft Corporation.

Linux® is a registered trademark of Linus Torvalds.

Red Hat® is a registered trademark of Red Hat, Inc.

EXECUTIVE SUMMARY

OBJECTIVE

The main objective of this research effort is developing a method for determining effective management of quality of service (QoS) resources within a heterogeneous distributed environment. Problems within the resource management environment have only been partially addressed. Success with components such as the Quorum component, Dynamic, Scalable, Dependable, Real-Time (DeSiDeRaTa), excludes network loading or bandwidth information in management decisions. These decisions should be more reliable if application network QoS requirements and runtime network loading information are considered. The object of this research was to construct an experiment that examined efficiently managing QoS within distributed environments based upon network information.

METHOD

Offering resource management facilities in a heterogeneous environment is a multifaceted problem. Many factors must be considered to successfully solve the problem. The Quorum Integration, Testbed and Exploitation (QUITE) project's Quality of Service Metrics Services (QMS) framework integrated the Quorum component, DeSiDeRaTa, with the Quorum component, REsource MOnitoring System (Remos), to investigate a hypothesis. This research also included the development of a specific test application that combined to create an experiment for determining efficient management of QoS resources within a distributed environment.

CONCLUSION

Experiment results described in this report have successfully demonstrated individual Quorum goals using integrated Quorum components. This experiment has integrated various Quorum technologies from low level to high level and created a layered resource management system. This work shows that this integration supports resource allocation and optimization at several different levels and creates the capability to install and use Quorum technologies effectively together. Results can be accessed through the QUITE framework and toolkit. These components, which include Remos and DeSiDeRaTa, were integrated correctly to introduce new multidimensional QoS management capabilities. The Conclusion section of this report provides a more detailed discussion of the experiment results.

CONTENTS

EXECUTIVE SUMMARY	iii
BACKGROUND	1
INTRODUCTION	3
MAJOR ISSUE.....	3
Experiment Goals.....	3
Experiment Description	3
EXPERIMENT ENVIRONMENT	5
ARCHITECTURE.....	5
RESOURCE MONITORING SYSTEM	7
DYNAMIC, SCALABLE, DEPENDABLE, REAL-TIME.....	8
EXPERIMENT REQUIREMENTS	11
HARDWARE.....	11
SOFTWARE	11
HARDWARE SETUP	12
ANALYSIS PROCEDURE	13
HYPOTHESIS.....	14
ASSUMPTIONS.....	14
DATA.....	14
EXPERIMENT EXECUTION	15
Starting Remos	15
Starting DeSiDeRaTa Middleware.....	16
Starting the Test Application	17
Controlling the Processing Load	19
RUNNING EXPERIMENTAL SCENARIOS: AN OVERVIEW	21
Movement through CPU Stress.....	21
Stressing the Network Link	22
CONCLUSION	25
EXPERIMENT RESULTS	25
DATA.....	26
REFERENCES	27
APPENDIX A	A-1

Figures

1. Software concept	5
2. Testbed architecture	6
3. Remos network monitoring.....	7
4. Resource and QoS management software architecture	9
5. DeSiDeRaTa architecture diagram.....	10
6. Hardware setup.....	12
7. DeSiDeRaTa HCI	17
8. HCI view of processes running on teksd6 host.....	18
9. Plot Dynamic Data pop-up menu.....	18
10. Latency information for the H:W:Testing path and applications.....	19
11. Test application GUI controller	19
12. BP process moved from one host to another	21

BACKGROUND

Many factors must be considered in successfully providing resource management capabilities in a heterogeneous environment. The Dynamic, Scalable, Dependable, Real-Time (DeSiDeRaTa) approach excludes specific network information in its resource management decisions. These decisions should be more reliable if they consider application network quality of service (QoS) requirements and runtime network loading information.

An experiment was developed to examine this premise and determine the effectiveness of adding network information into the QoS resource management decisions. This experiment examined the integration of the Quorum component, DeSiDeRaTa, performing resource management functions, and the Quorum component, REsource MONitoring System (Remos), providing dynamic network loading and bandwidth information with a test application.

The DeSiDeRaTa program controlled applications to manage the latency of execution paths through a cooperating set of applications. The path(s) controlled had to be straight-line execution paths that proceeded from start to finish. The progress of a particular path's execution was monitored by a series of messages sent from the applications to DeSiDeRaTa, which informed it of path progress. It was assumed that each application in a path finished its work before the next application in the path began its work. Multiple iterations of a given path could be active at any time.

Remos determined logical network topology from lists of hosts' routers and switches in the network by using the Simple Network Management Protocol (SNMP) facilities. All network systems monitored required SNMP capability. After topology determination, Remos began to continuously gather SNMP network traffic information from the hosts in the local area or wide area network monitored and from any routers in the configuration. This information determined network traffic volumes on individual segments and data flowing between hosts of interest to Remos clients. Remos provides a client modeling application program interface (API) that specified and retrieved host topology and flow information.

INTRODUCTION

MAJOR ISSUE

The major issue examined in this experiment was whether application management decisions for distributed applications were more effective when network bandwidth loading information and application network bandwidth QoS requirements were considered.

Experiment Goals

The goals of the experiment were as follows:

- Determine whether considering network loading information allows DeSiDeRaTa to make more effective application management action decisions.
- Determine under what network load circumstances this information supports better decisions and when it does not.

Experiment Description

All the test application instances were started on a single DeSiDeRaTa-controlled host. The operator caused DeSiDeRaTa to move a single instance to see where the application was moved. The application's maximum workload baseline was predetermined before execution. The application was first started at a lower workload, then increased so that DeSiDeRaTa moved it again. This process was repeated to gather ongoing data on DeSiDeRaTa management decisions.

Four test application instances were used for the test setup. Logically, only three applications were needed to perform the test. However, due to the way in which DeSiDeRaTa V1.4 worked and the specific experiment executed, the first instance of the application that initiated each path could not be moved from its initial host. To ensure that this did not happen, four instances of the test application were started. These applications were designated AP, BP, CP, and DP in path order. The last three applications in the path were the ones potentially effected by the test changes and so DeSiDeRaTa could move these applications. The message-input file was set large enough to use significant bandwidth when multiple messages per second were sent.

During the test, outputs from all application instances were recorded using the QMS framework's Logging Plugin in a logfile. For analysis, only the output from the CP and DP applications were used because these applications were the ones set to non-default behavior. The default-reporting interval for each test application instance was once per 5 seconds. This interval was changed for CP and DP to be once per second. The number of messages per second to send was manipulated, changing the workload imposed on the application and the execution path. For this test application version, when DeSiDeRaTa started or restarted an application instance, non-default parameters always had to be reset for that instance.

The test sequence to start the application path on a single machine was as follows:

1. Set the reporting interval on DP to once per second.
2. Set the workload of the CP application to exceed the path latency.
3. See where DeSiDeRaTa moved the application.

4. Set the reporting on CP to once per second.
5. Set its workload to a level that should be attainable on an unloaded system with full bandwidth available.
6. See whether DeSiDeRaTa will move the application or not. If not, set the workload (number of messages per second) up to cause DeSiDeRaTa to move the application. Record the actions performed in the note file and iterate.

The experiment tested combining Quorum components such as DeSiDeRaTa and Remos, providing more functionality than either provided alone.

This experiment was run with test application(s) under DeSiDeRaTa control. The test application provided a single execution path that DeSiDeRaTa controlled. It required a user-configurable amount of network bandwidth. Network and central processing unit (CPU) workload varied.

The test application provided an end-to-end flow of network traffic with the user individually controlling the volume of information between application instances. CPU workload could be controlled for each application instance. The user or DeSiDeRaTa could dynamically re-locate application instances. A graphics user interface (GUI) provided all application controls. The application instances communicated through a Common Object Request Broker Architecture (CORBA) interface. The application instances communicated with DeSiDeRaTa by the DeSiDeRaTa Transport Control Protocol/Internet Protocol (TCP/IP) sockets' API. The user controlled the message size for messages between applications. The user supplied a file to be transmitted for each instance of the application. The file could be any useful size.

For this experiment, messages were sent between instances of the test application with different network segments set to different traffic levels. Network (iperf server/client) stressor applications stressed selected network segments during the experiment to simulate reduced network availability.

To determine the performance of the test application, metrics were gathered detailing the network workload over time in the network segments. DeSiDeRaTa measured path latencies and application management actions were recorded. Test application message throughput information was also gathered and comparisons were made with desired network traffic, network stress levels, application latencies, and DeSiDeRaTa control actions during the experiment.

Two versions of DeSiDeRaTa were compared. The first version was standard DeSiDeRaTa V1.4 as provided by Dr. Lonnie Welch and the DeSiDeRaTa project. The second version was V1.4, modified to allow network inputs and thinning of possible hosts, which could be used for application hosting based on data from Remos and test application network requirements specifications. The experiment was run several times with varying loads for each DeSiDeRaTa version. The results were compared to determine how standard DeSiDeRaTa compared with a network-enhanced DeSiDeRaTa in making decisions on where to place applications when path latencies were exceeded and application management actions were required.

EXPERIMENT ENVIRONMENT

The testbed for these experiments was composed of several commercial off-the-shelf (COTS)-available Intel[®]-based Pentium[®] II architectures with single-processor systems running at 400 MHz, containing generic components. The communication of these systems was through a 100 BaseT Ethernet. The network node PCs were inhabited by 3Com[®] 100BaseT network cards, and a typical high-density display system based upon the Intel740[®] video accelerator. The operating system was Microsoft Windows NT[®] version 4.0 build 1381 with Service Pack 6a. Standard generic drivers were used during these tests, and no modifications were performed upon these devices.

The storage devices, primary and secondary, used on these systems were 512-MB SDRAM (60 ns) and 16-GB NTFS format SCSI (with parity check enabled) respectively. The BIOS used by these systems was AwardBIOS[™] version 4.51, with HAL: MPS 1.4-APIC platform. The internal clock mechanism used was the Microsoft Windows NT[®] multimedia timer, which provided clock resolution of greater than 1-ms architecture.

ARCHITECTURE

The software concept for the managed QoS experiment included elements of software developed within the Quorum program. These components included DeSiDeRaTa, Remos, QMS, and specifically developed test applications. Figure 1 shows a simplified diagram of this relationship.

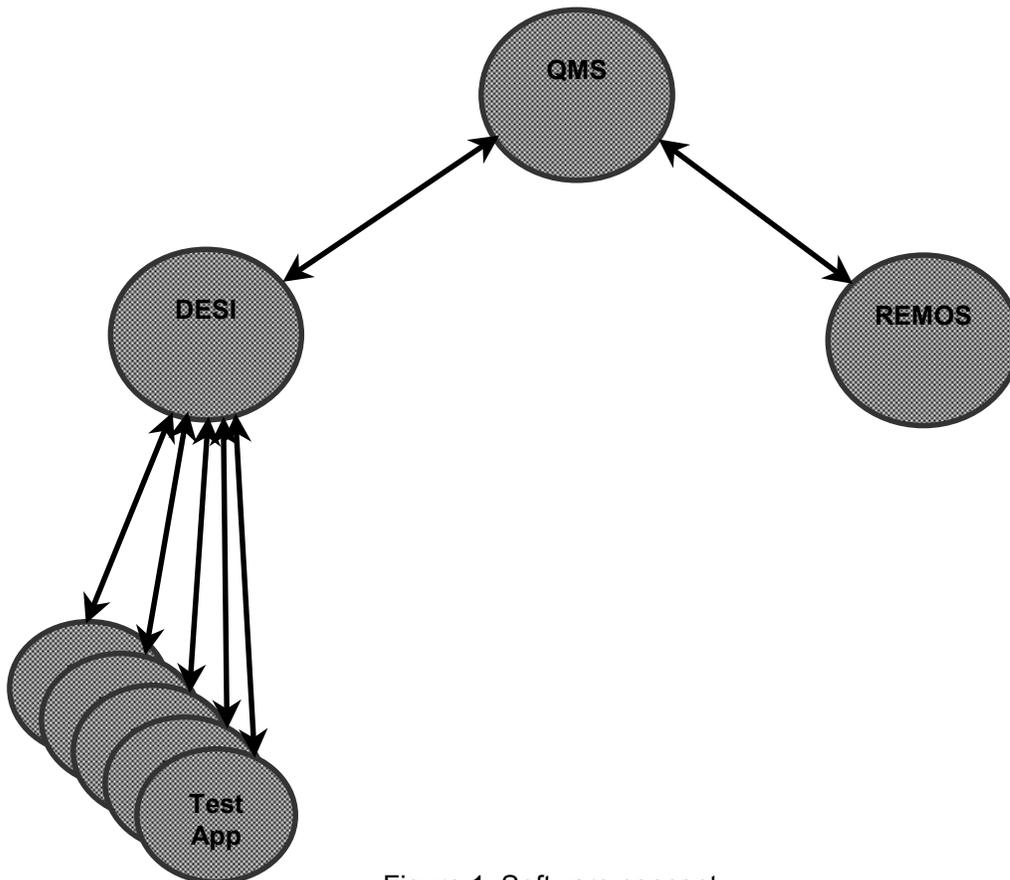


Figure 1. Software concept.

The design rationale for this experiment's software architecture was to use DeSiDeRaTa for the resource management functions, Remos for the feed of dynamic network information, and QMS for the basic integration framework and network/application metrics and data recording. The application program provided a good examination of the efficiency and proper functioning of the main Quorum software components (Figure 2).

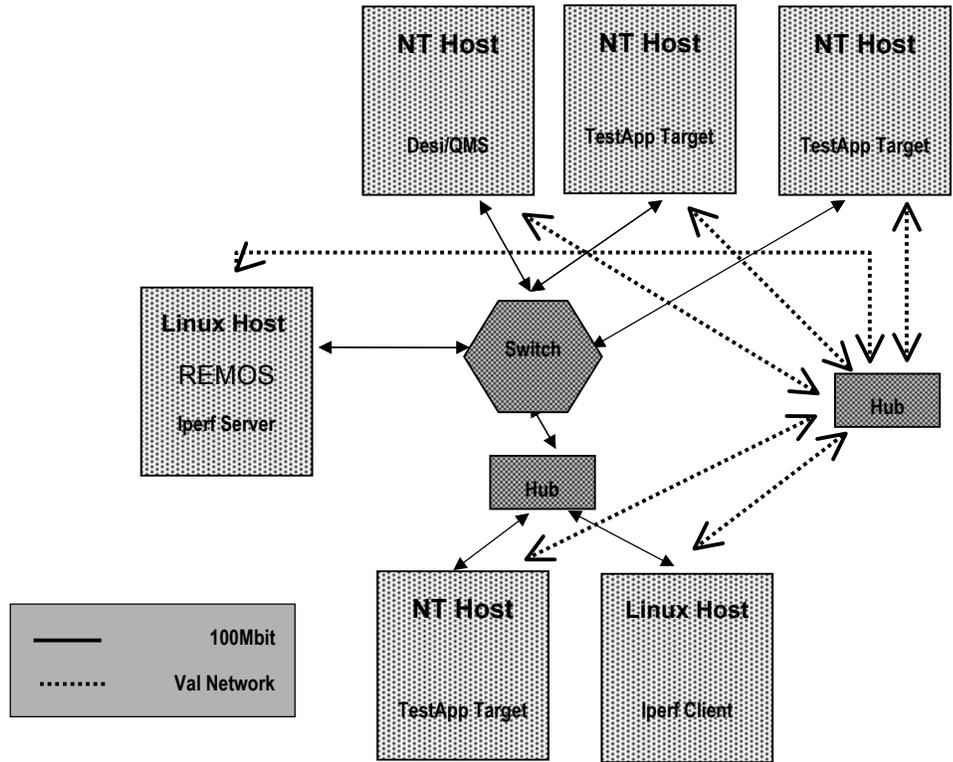


Figure 2. Testbed architecture.

RESOURCE MONITORING SYSTEM

The Quorum program Resource Monitoring System (Remos) project produced the Remos used in this experiment. This project developed the Remos system at the Carnegie Mellon University School of Computer Science.

Figure 3 shows the use of Remos to monitor network-specific information (DeWitt 2000). DeWitt describes Remos as follows: “Remos supports two classes of queries. ‘Flow queries’ provide a portable way to describe a communication step to the Remos implementation, which uses its platform-dependent knowledge to return to the user the capacity of the network to meet this request. ‘Topology queries’ reverse the process, with the Remos implementation providing a portable description of the network's behavior to the application.”

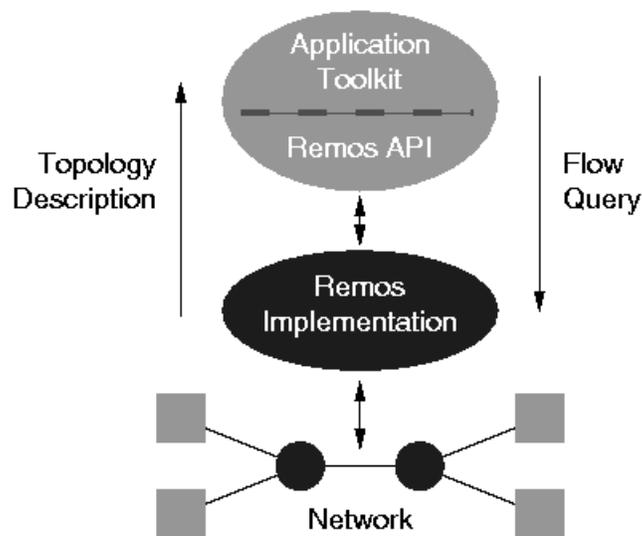


Figure 3. Remos network monitoring.

As further described in DeWitt (1998): “Remos is a query-based interface to the network state. Queries can be used to obtain the structure of the network environment, or to obtain information about specific sets of nodes and communication links on the network. The main features of the Remos interface can be summarized as follows:

Logical Network Topology: Remos supports queries about the structure of the network environment. The structure presented is a “logical topology,” i.e., a representation of the network characteristics from an application’s standpoint, which may be different from the physical network topology.

Flow-based queries: Queries regarding bandwidth and latency are supported for *flows*, which are logical communication channels between nodes. Flows represent application-level connections, and therefore, should be an easy to use abstraction for applications.

Multiple flow types: Remos supports queries relating to fixed flows with a fixed bandwidth requirement, variable flows that share bandwidth equally, and additional independent flows that can absorb all unused bandwidth.

Simultaneous queries: An application can make queries about multiple flows simultaneously. The Remos response will take any resources sharing by these flows into account.

Variable time-scale queries: Queries may be made in the context of invariant physical capacities, measurements of dynamic properties averaged over a specified time window, or expectations of future availability of resources.

Statistical measures: Remos reports all quantities as a set of probabilistic quartile measures along with a measure of estimation accuracy. The reason is that dynamic measurements made by Remos typically exhibit significant variability, and the nature of this variability often does not correspond to a known distribution.

DYNAMIC, SCALABLE, DEPENDABLE, REAL-TIME

The resource management system used in this experiment was a product of the Quorum Program Dynamic, Scalable, Dependable Real-Time (DeSiDeRaTa) Project. This system was developed by the Laboratory for Parallel and Distributed Real-time Systems in the University of Texas Department of Computer Science and Engineering, and Ohio University Laboratory for Intelligent Real-Time Secure Systems School of Electrical Engineering and Computer Science.

As noted by Welch (2000): “The DeSiDeRaTa project is providing innovative QoS management technology, which incorporates knowledge of needs in the distributed shipboard computing systems domain.” The overall targets for this work are systems within environments, which have real-time constraints. Welch provides the following explanation: “DeSiDeRaTa differs from previous work in that it accounts for the complex features of dynamic real-time systems. These features include previously overlooked issues with respect to granularity, variable periods, sporadic processes, priorities, fault management and scalability.”

This resource management system is based upon a logical architecture that provides efficient management behavior (Figure 4) and is further described by Welch: “The application programs of real-time control paths send time-stamped events to the *QoS metrics* component, which calculates path-level QoS metrics and sends them to the *QoS monitor*. The monitor checks for conformance of observed QoS to required QoS, and notifies the *QoS diagnosis* component when a QoS violation occurs. The diagnoser notifies the *action selection* component of the cause(s) of poor QoS and recommends actions (e.g., move a program to a different host or LAN, shed a program, or replicate a program) to improve QoS. Action selection ranks the recommended actions, identifies redundant actions, and forwards the results to the *allocation analysis* component; this component consults resource discovery for host and LAN load index metrics and determines a good way to allocate the hardware resources in order to perform the actions, and requests the actions be performed by the *allocation enactment* component.”

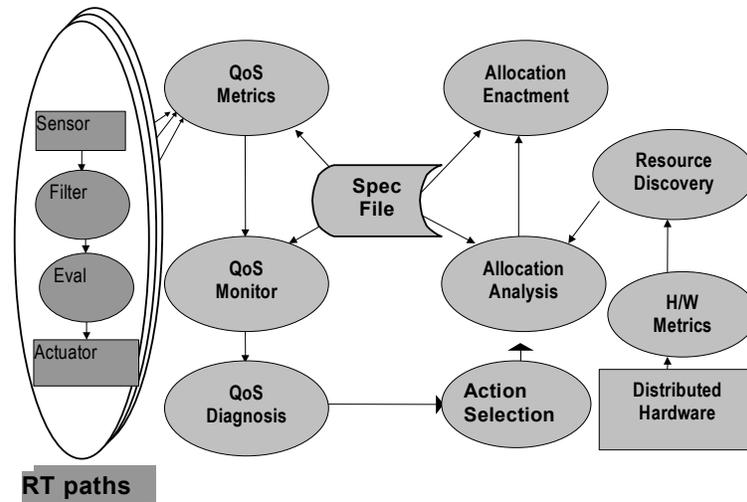


Figure 4. Resource and QoS management software architecture.

Figure 5 shows the element-level integration of the DeSiDeRaTa system into the QUITe project experiment. This diagram builds upon a previous illustration presented by Welch at a Quorum principal investigator meeting in 1999. In this diagram the functional aspects of this experiment with regard to the DeSiDeRaTa element are illustrated. During experiment execution, the Network Broker derived a list of pair wise bandwidth measurements for the hosts that could support an application component instance. The Network Analyzer then used this listing to develop a comprehensive inventory of hosts that had adequate network resources. This list was then used to reduce the existing list of hosts supplied by the Hardware Broker (based on its analysis of CPU use, etc.). This reduced list was then composed of hosts that could support the processing and bandwidth requirements of the application path.

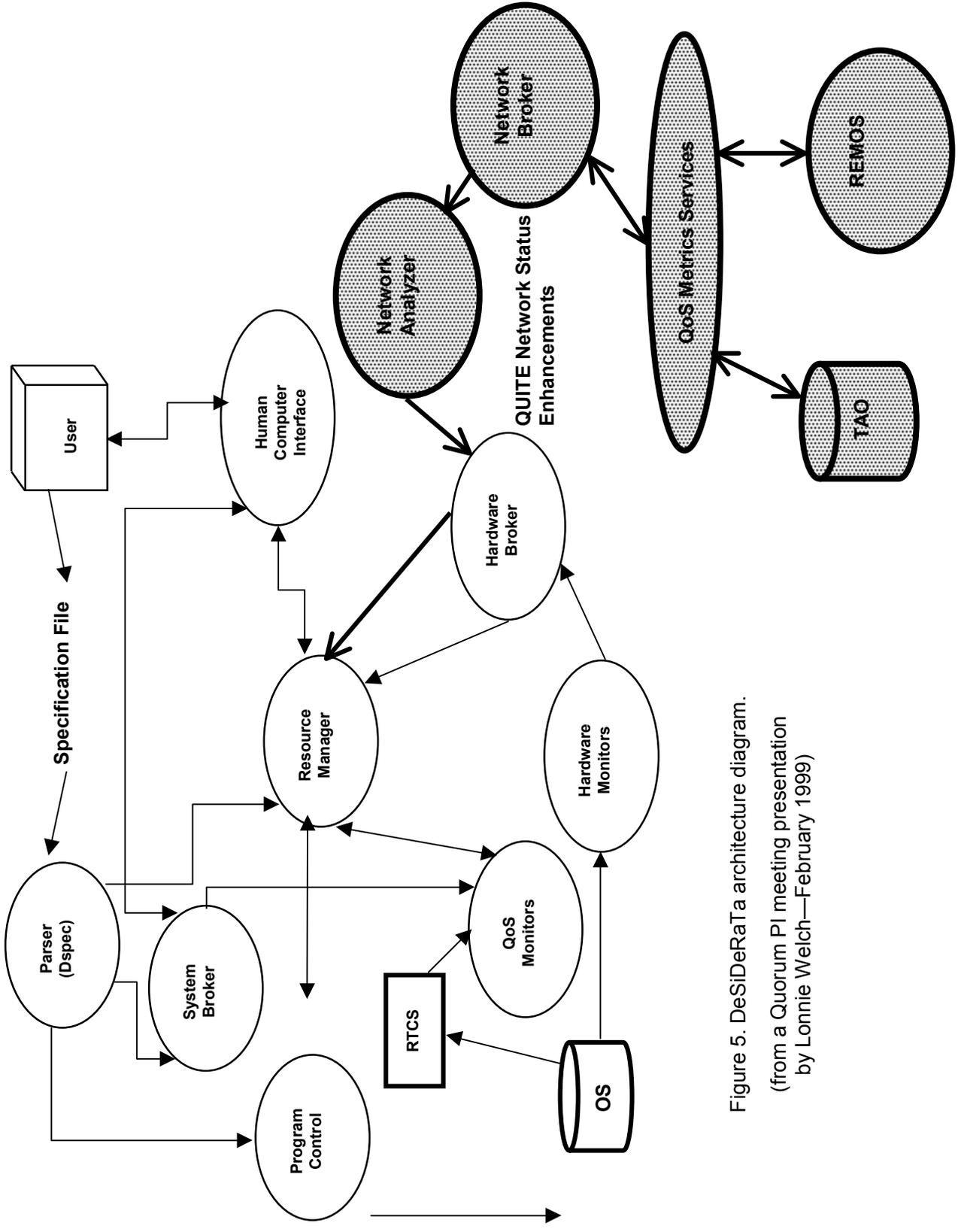


Figure 5. DeSiDeRaTa architecture diagram.
 (from a Quorum PI meeting presentation
 by Lonnie Welch—February 1999)

EXPERIMENT REQUIREMENTS

This section describes the hardware and software requirements for the managed QoS experiment. The description is followed by setup and configuration procedures. Next, this section describes the general execution and use of the software, which familiarizes the user with the software before actual experimental runs are attempted. Finally, an overview of how to perform the experiments is presented.

HARDWARE

The minimum hardware requirements for this managed QoS experiment includes the following elements:

- Five Microsoft Windows NT[®] machines: PII-266, 128-MB RAM, 10/100 local area network (LAN)
- One Red Hat[®] Linux machine: PII-266, 128-MB RAM, 10/100 LAN
- Cisco Catalyst[®] 2900 series XXL
- 10/100 LAN Hub

SOFTWARE

The software elements used for development and execution of these experiments included operating systems, language compilers, middleware, software development environment tools, and QMS. Two main operating systems are used for experiment execution. The Microsoft Windows NT[®] operating system version 4.0 instituting Service Pack 6a and the Red Hat[®] Linux operating system version 6.2. Other software includes the Java Development Kit. This section describes the use of the QUITE project managed QoS experiment software. The fundamental software distribution consists of four main pieces of software:

- Test Application (used to simulate the application path)
- DeSiDeRaTa middleware (modified by the QUITE team to make use of Remos data)
- QMS (used to relay information between Remos and DeSiDeRaTa and experiment data collection)
- Remos with Instrumentation (Remos distribution with QUITE team instrumentation)

Windows NT[®] Machines

- Windows NT[®] 4.0 SP 3
- Java[™] Development Kit (JDK)/Java[™] Runtime Environment (JRE) 1.3
- Visual C++ 6.0
- The Adaptive Communication Environment (ACE[™]) Object Request Broker (ORB): The ACE ORB (TAO[™])

Linux[®] Machines

- Red Hat[®] Linux[®] 6.0
- JDK/JRE 1.3

HARDWARE SETUP

In Figure 5, five Windows NT[®] machines and one Linux machine are used. Two Windows NT[®] machines are connected to a 10-MB hub, which, in turn, is connected to the switch.

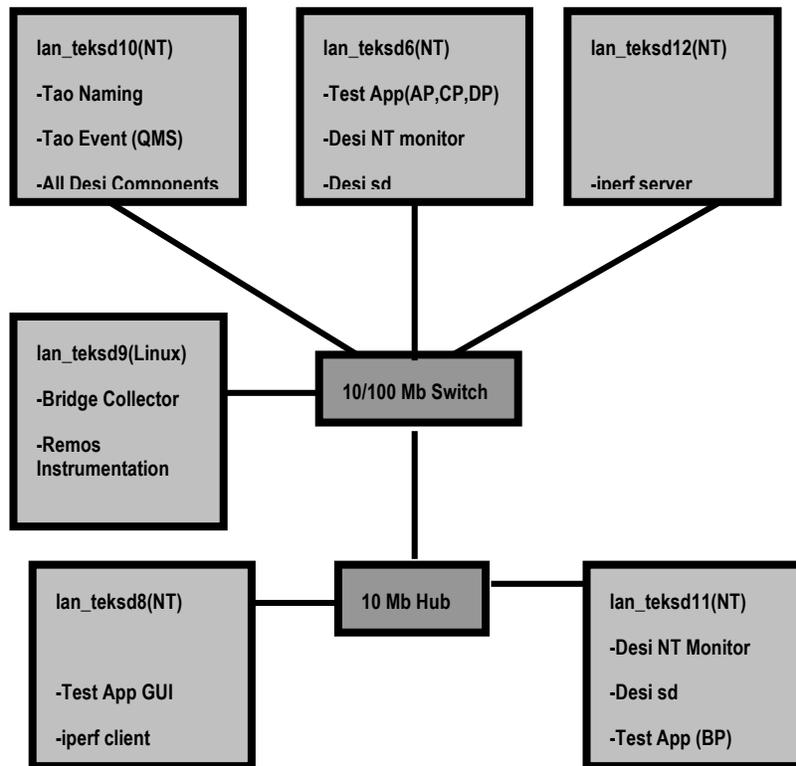


Figure 5. Hardware setup.

ANALYSIS PROCEDURE

This experiment measures the effect of network load requirement information on DeSiDeRaTa management decisions. It is expected that better performance of the application will be seen when the network load requirements of the application are met. The procedural elements of this managed QoS experiment include the following:

- Extract network loads over time from QMS log
- Extract message throughput over time from QMS log
- Extract path latency information from DeSiDeRaTa QoS Manager log
- Thin latency information to one value per second elapsed
- Reconcile time lines of different data streams so that they are in line with consistent data points (one point per second)
- Graph values over time and compare curves for different stress levels
- Compare graphs between DeSiDeRaTa versions to determine whether standard DeSiDeRaTa or network-enhanced DeSiDeRaTa makes better decisions on process placement

The experiment is performed by starting all of the test application instances on a single DeSiDeRaTa-controlled host and causing DeSiDeRaTa to move a single instance and observe where the application is moved. The application workload is set to determine its maximum workload. The workload is then increased so that DeSiDeRaTa will move it again. This process is repeated to gather ongoing data on DeSiDeRaTa management decisions.

Four test application instances are used for the given test setup. Logically, only three applications are required to perform the test. However, due to the way in which DeSiDeRaTa V1.4 works, the first instance of the application, which initiates each path, cannot be moved from its initial host. To ensure that this will not happen, four instances of the test application are started. These applications are designated AP, BP, CP, and DP in path order. The last three applications in the path are the ones potentially effected by the test changes and so DeSiDeRaTa may move these applications. The message-input file is set large enough to use significant bandwidth when multiple messages per second are sent.

During the test, outputs from all application instances are recorded in the QMS log. For analysis, only the output from the CP and DP applications are used because these applications are the ones set to non-default behavior. The default reporting interval for each test application instance is once per 5 seconds. This interval is changed for CP and DP to be once per second and the number of messages per second to send is then manipulated, thus changing the workload imposed on the application and the execution path. For this version of the test application, when an application instance is started or restarted by DeSiDeRaTa, any non-default parameters must always reset for that instance.

The test sequence, therefore, is to start the application path on a single machine. Set the reporting interval on DP to once per second. Set the workload of the CP application to exceed the path latency, see where DeSiDeRaTa moves the application, set the reporting on CP to once per second, set its

workload to a level that should be attainable on an unloaded system with full bandwidth available, and see whether DeSiDeRaTa moves the application or not. If not, set the workload (number of messages per second) up to cause DeSiDeRaTa to move the application. Record the actions performed in the notes file and iterate.

HYPOTHESIS

In distributed environments, applications control environments that take advantage of information about communication requirements between host nodes will do a better job of resource management and provide better QoS than those that do not.

We feel that one practical way to test this hypothesis is to use a working application manager and modify it to consider network loading information and then compare the actions of the modified tool with the unmodified tool. Our chosen tools for this test are DeSiDeRaTa and Remos working within the QUITE QoS Toolkit framework and using QMS as an intermediary. For applications that make significant use of a network, DeSiDeRaTa provides better latency management and resource use with Remos-provided network information than without it.

ASSUMPTIONS

The experiment description in this document emphasized miscellaneous assumptions that directly apply to this managed QoS examination. These assumptions are listed here as they are used during the experiment.

The basic operational assumption for this experiment includes the presumption that DeSiDeRaTa, Remos, and any/all other Quorum program components correctly execute as their design logically requires.

Regarding the Remos element of this experiment, it is assumed that the Remos collector software is functionally loaded and executing at the time the experiment is executing.

The DeSiDeRaTa software controls applications to manage the latency of execution paths through a cooperating set of applications. It is assumed that the paths that are controlled are straight-line execution paths that proceed from start to finish. Therefore, it is assumed that each application in a path finishes its work before the next application in the path begins its work.

The fundamental communication infrastructure assumptions for this experiment include the presumption that all tests will be performed within the QUITE project distributed testbeds. Most (or all) of these experiment tests are LAN-based within a distributed environment.

DATA

This experiment is mainly concerned with the effects of adding network bandwidth awareness to an existing process management and control tool to determine whether this improves or changes the correctness of its management decisions. Bandwidth awareness is provided by integrating sensed bandwidth data into the DeSiDeRaTa process control decision procedure. The bandwidth information comes from the Remos tool. To support the needed analysis, data must be gathered on how DeSiDeRaTa is reaching its decisions. The data required are the actual DeSiDeRaTa commands issued, the network and other host metric data input to DeSiDeRaTa and output from Remos, the

application path status inputs to DeSiDeRaTa, control inputs to DeSiDeRaTa and Remos, configuration inputs, and static hardware configuration data.

Data are gathered from all QoS-related software used in this experiment and from the QUITE validation software infrastructure. The QoS-related software is as follows:

- QUITE QMS
- DeSiDeRaTa
- DeSiDeRaTa instrumentation
- Remos and the QMS Remos probe instrumentation
- QUITE test application

Data gathered from these software applications include control and management data output by these applications as part of their normal management and monitoring process and data gathered from non-intrusive monitoring probes added by QUITE. DeSiDeRaTa has also been modified to output text informational messages from its QoS Manager module detailing its path latency calculation results and its decisions to take a management action for a process (move or copy the process).

The QUITE infrastructure software includes the following:

- Iperf Network Stressor
- QMS Logger Plugin

Data gathered from the infrastructure software consists of various kinds of data metrics describing the host, network, and processing environment in which the experiment is run.

EXPERIMENT EXECUTION

Starting Remos

To launch REMOS/QMS, the TAO naming and QMS event channel should be running. Two scripts are provided on the Windows NT[®] machines to launch the TAO naming and event servers:

1. tao_naming.bat (**launch this first**)
2. qms_channel.bat (**launch this next**)

Next, launch Remos (on the Linux machine) by running the following script files provided:

Run_collector.sh

Run_bridgecoll.sh

Run_remos_int.sh

After running this last script file, you should see the Remos instrumentation connect to the QMS event channel (via text messages displayed in console window).

Starting DeSiDeRaTa Middleware

Once TAO naming, the QMS Event Channel, and Remos are running, the DeSiDeRaTa middleware can be started. To launch the DeSiDeRaTa middleware (including the new QUITE enhancements), the following processes must be started:

- DeSiDeRaTa Name Server
- DeSiDeRaTa System Broker
- DeSiDeRaTa Resource Manager
- DeSiDeRaTa Host Broker
- DeSiDeRaTa Host Analyzer (QUITE-modified version)
- Network Analyzer (new addition)
- Network Broker (new addition)
- DeSiDeRaTa NT[®] Monitor (host monitor)
- DeSiDeRaTa Program Control
- DeSiDeRaTa Startup Daemon
- DeSiDeRaTa Human–Computer Interface (HCI)

A batch file called *testapp_startup.bat* is available that starts each piece. Two GUIs appear: *NT[®] Monitor* and *HCI*. Launch this batch file, pressing the space bar each time to continue launching the various pieces. Once the HCI GUI is displayed, prepare the remote machines to act as potential hosts.

To move processes from one machine to another, all eligible machines must first be specified in the *DesiderataHW.spec* file and listed as a potential host in the *Testapp.spec* file.

Once the HCI GUI on the host DeSiDeRaTa machine appears, the following two DeSiDeRaTa processes must be started on all remote Windows NT[®] migration target machines:

1. NT[®] Monitor – start the NT[®] Monitor.exe on the remote machine. A script called *desi_nt_monitor.bat* executes the monitor and connects to the DeSiDeRaTa name server.
2. SD.exe – start the Start Daemon (SD.exe) on the remote machine by using a script called *desi_sd.bat*.

The above processes (NT[®] Monitor and SD) should be started after the HCI on the host machine is started. Do not start the test applications until all of the desired remote machines have started their NT[®] Monitor and SD processes.

In the HCI GUI, select *Connect* under the *File* menu and enter the appropriate host and port where the DeSiDeRaTa Name Server is running.

In the HCI GUI, various views can be started from the top menu bar:

A *Host Window* shows the various hosts and processes running on each host.

A *Path Window* shows the application path. This view also provides access to a dynamic plot of the latencies through a *right mouse click* inside this window.

A *Console Window* allows viewing text accounts of DeSiDeRaTa events.

Starting the Test Application

Once DeSiDeRaTa has started, from the DeSiDeRaTa HCI, on the top menu bar, select *StartSystems*. To launch the test application path, select *H:W*. Figure 7 shows this action sequence.

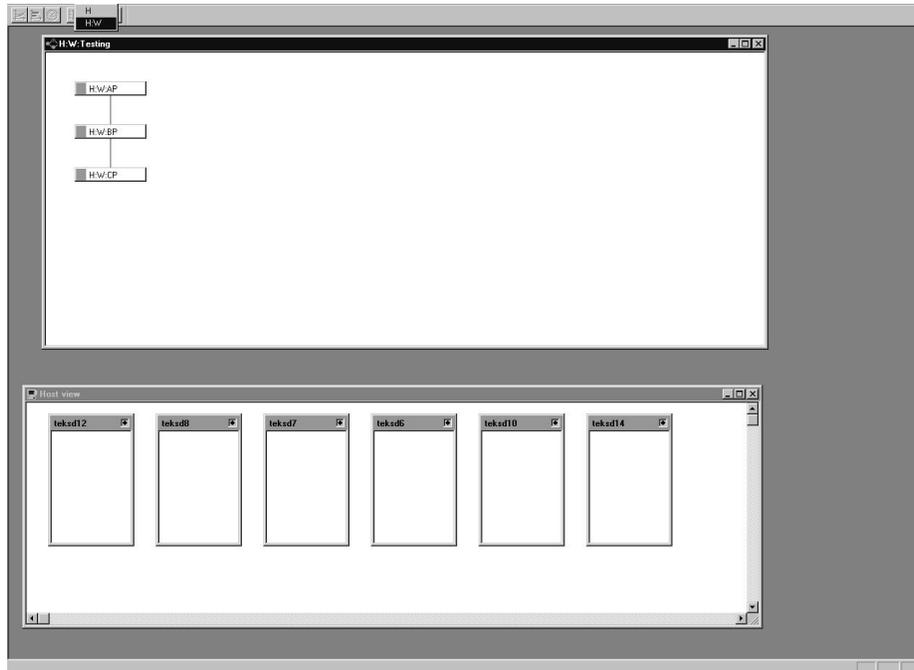


Figure 7. DeSiDeRaTa HCI.

DeSiDeRaTa begins to launch the TestApp path. There are three processes that execute: AP, BP, and CP. In the *Host View* window on the DeSiDeRaTa HCI, each process can be seen executing on a host machine. As Figure 8 shows, the processes are running on host, *teksd6* (*in our particular testbed*).

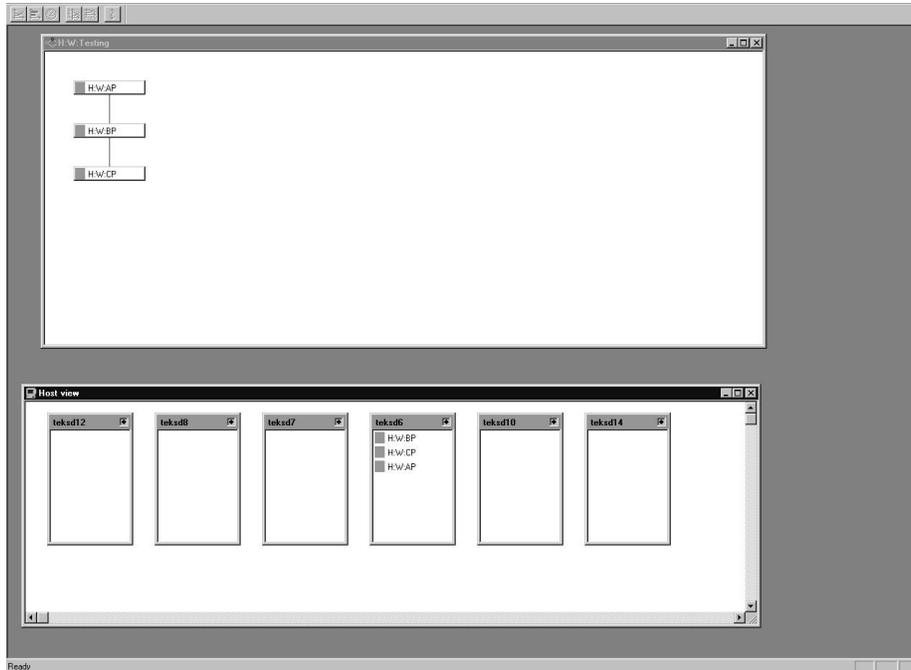


Figure 8. HCI view of processes running on tekcd6 host.

Each process, AP, BP, and CP, execute inside an MS-DOS[®] window. These windows can be seen on the machine corresponding to what is shown in the host view.

The DeSiDeRaTa HCI also features the ability to dynamically plot latency (in milliseconds) information of the entire path and each application. Hitting the right mouse button inside the H:W:Testing window of the DeSiDeRaTa HCI accesses this plot view.

This action brings up the *pop-up menu* shown in Figure 9. Select *Plot Dynamic Data* to view the graphical data. Figure 10 shows an example of the latency graph.

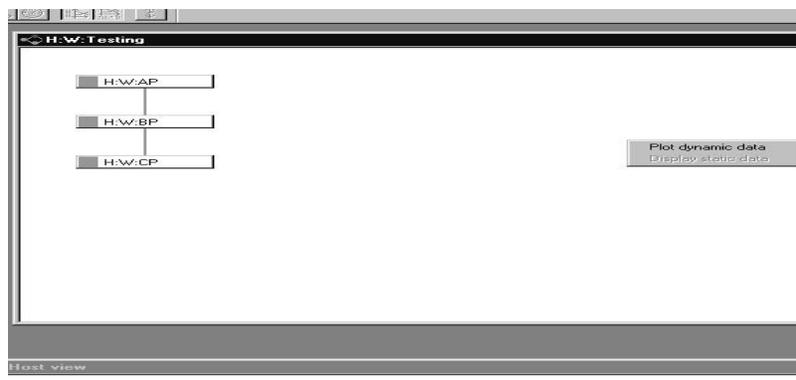


Figure 9. Plot Dynamic Data pop-up menu.



Figure 10. Latency information for H:W:Testing path and applications.

Controlling the Processing Load

The test application can be controlled by the provided GUI. After the applications have been started, the user may launch the GUI Controller. A batch file inside the *scripts* subdirectory has been provided called *testapp_gui.bat*. This file launches the GUI. Once the GUI starts up, click on the *connect* button at the left corner of the GUI (you may need to resize the GUI to see things correctly.) Figure 11 shows the connected GUI.

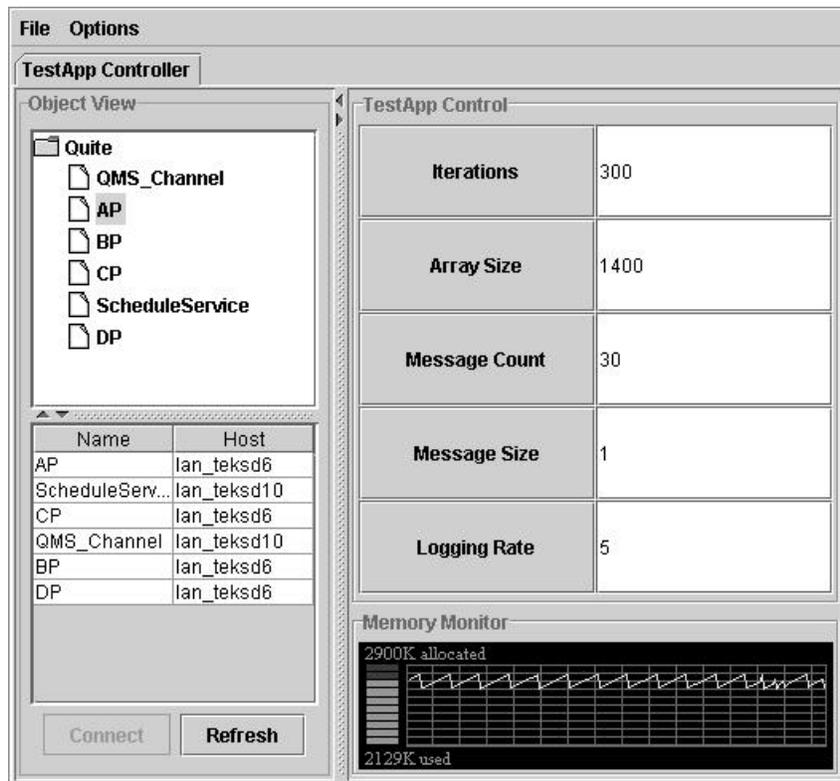


Figure 11. Test application GUI controller.

As figure 11 shows, an *Object View* lists the objects registered with TAO CORBA Naming Service. Below this view is a table describing where each object is executing. To the right of these panels is the *TestApp Control*. This area allows dynamically modifying processing parameters in the test application.

To make a modification to the processing of one of the application pieces (AP, BP, or CP), do the following:

1. Select the process in the *Object View*. (NOTE: We have encountered problems with DeSiDeRaTa in modifying the start of the application path, AP. We suggest that modifications be made only to the middle of the path [in our case, BP]). Selecting *EventService* and *ScheduleService* will have no effect.
2. In the *TestApp Control* panel, modify the desired value in the text box.
 - Iterations = how many times to perform the sorting algorithm on the dataset.
 - Array Size = how many data items to be sorted.
 - Message Count = how many *message units* (see next item) to send *each cycle*.
 - Message Size = how big the *message unit* will be. The *message unit* is defined as the text message multiplied by the value of the message size. (The text message is defined either in a text file or as the default string, *Experiment 1*. See the Appendix chapter section on Experiment Setup and Configuration for more details on setting the text message.). As an example, consider a message size of 4 and the text message as the default string “Experiment 1”. This example would produce the following message unit:

“Experiment 1 Experiment 1 Experiment 1 Experiment 1”
 - Logging Time (seconds) = how often to log the test application data. The following information is logged:
 1. Length of the message.
 2. Number of messages sent each cycle.
 3. How many messages have been sent so far.
 4. Size of the data array
 5. How many times the data array is sorted.
 6. Time stamp information.

When you are ready to send the modification, click on the corresponding button (i.e., for Array Size, click on the Array Size button next to the text box). The modification will affect the process selected/highlighted in the *Object View*.

Once DeSiDeRaTa decides that a move is necessary, the target process starts on a suitable host and is killed on the former host. This move can be seen in the *Host View* window of the HCI. In Figure 12, the *BP* process was moved from *teksd6* to *teksd10*.

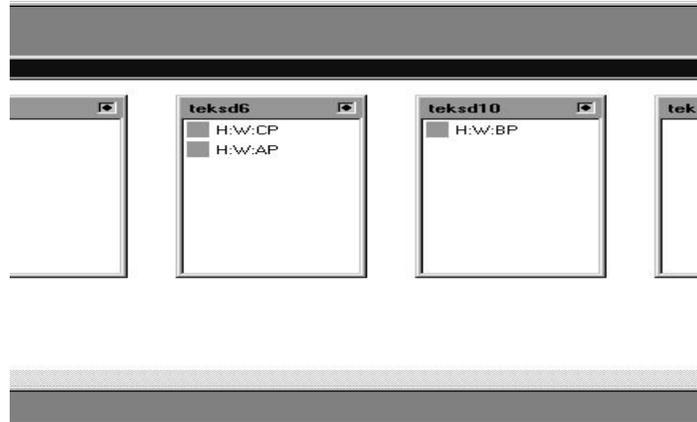


Figure 12. BP process moved from one host to another.

RUNNING EXPERIMENTAL SCENARIOS: AN OVERVIEW

This section describes how to execute the basic experimental scenarios. Please read the previous sections on running the various software components as they give a more detailed view of how to operate each piece of software.

There are two types of scenarios explored in this section:

- Stressing a particular process in the path using the CPU workload parameters, thereby causing that process to move to another machine.
- Stressing a particular link between processes, thereby causing certain hosts to be dropped from consideration for moving.

Movement through CPU Stress

The first scenario involves stressing a particular process in the path (in our example, BP) using the test application GUI. We assume that Remos collector is running. Please refer to documentation on Remos and the QMS/Remos instrumentation for more details on setup and execution of that component. What follows are the steps used to execute this first scenario.

1. Start TAO Name Server.
2. Start TAO Event Channel (qms_channel.bat).
3. Start Remos Instrumentation.
4. Start testapp_startup.cmd. This batch file starts up the DeSiDeRaTa middleware. Press the spacebar to launch each part of the middleware.
5. Prepare the remote machines by launching the Windows NT[®] monitors and Start Daemons on the remote hosts. This launch is accomplished by executing the desi_nt_monitor.bat and the desi_sd.bat files.
6. On the DeSiDeRaTa HCI, select File->Connect and connect to the DeSiDeRaTa Name Server by entering the host machine and port in the dialog box.

7. On the DeSiDeRaTa HCI, select Hosts->Host View.
8. On the DeSiDeRaTa HCI, select Paths->H:W:Testing. When that window pops up, right-click on it and select Plot Dynamic Data.
9. On the DeSiDeRaTa HCI, select StartSystems->H:W:Testing. This selection launches the TestApplication as described in the section, *Starting the TestApplication*.
10. Once all the parts of the path have been launched and are executing (you should see text scrolling rapidly in each of the TestApp windows), execute the TestApp GUI controller. The file testapp_gui.bat launches the GUI.
11. Connect the GUI to the TAO Name Server by clicking on the connect button.
12. Causing the BP process to move to another host will vary depending on processor speed, memory, network information, etc. Changing the *array_size* to 300 and the *Iterations* to 1300 will usually cause DeSiDeRaTa to move the BP process. These values can be changed using the testapp GUI as explained in the above section.
13. Once DeSiDeRaTa detects path latency beyond the desired threshold, the BP process should be moved to another host as shown in the previous section.

Stressing the Network Link

This section describes the second basic scenario in which the network link between processes is stressed, thereby causing DeSiDeRaTa to remove certain hosts from the list of possible targets for migration (*Good Hosts*). Consider the setup described in Figure 6.

In this scenario, there are six machines. Of these six, three hosts are possible migration targets for DeSiDeRaTa. The idea behind this scenario is that by executing an outside network stressor such as *iperf* between lan_teksd8 (connected by hub to lan_teksd11) and lan_teksd12, DeSiDeRaTa removes lan_teksd11 from the list of potential migration targets because the network load on that link has surpassed the desired threshold. Running the network stressor tool on any host will also change the CPU use on that host. To prevent the effects of this CPU use on the experiment, we put tekstd8 and tekstd11 on the same logical network segment using a hub. The approximate steps for executing this scenario are as follows:

Start TAO Name Server.

Start TAO Event Channel (qms_channel.bat).

Start Remos Instrumentation.

Start testapp_startup.cmd (this batch file starts up the DeSiDeRaTa middleware. Press the spacebar to launch each part of the middleware).

Prepare the remote machines by launching the Windows NT[®] Monitors and Start Daemons on the remote hosts. This launch is accomplished by executing the desi_nt_monitor.bat and the desi_sd.bat files.

On the DeSiDeRaTa HCI, select File->Connect and connect to the DeSiDeRaTa Name Server by entering the host machine and port in the dialog box.

On the DeSiDeRaTa HCI, select Hosts->Host View.

On the DeSiDeRaTa HCI, select Paths->H:W:Testing. When the window pops up, right-click in it and select Plot Dynamic Data.

On the DeSiDeRaTa HCI, select StartSystems->H:W:Testing. This selection launches the testApplication as described in the section, *Starting the TestApplication*.

Once all parts of the path have been launched and are executing (you should see text scrolling rapidly in each of the testApp windows), execute the testApp GUI controller. The file testapp_gui.bat launches the GUI.

Connect the GUI to the TAO Name Server by clicking on the connect button.

Run iperf server on a machine connected to the switch by a hub. Use a command similar to *iperf -s*

Run an iperf client on a machine connected directly to the switch by using a command such as the following:

```
Iperf -c [IP address of the iperf server] -w 50000 -t 900
```

(NOTE: see iperf -h for more help on iperf)

In the Host Analyzer DOS window on the DeSiDeRaTa middleware machine, you should notice that the host connected to the hub is no longer a potential migration target (*Good Host*).

Running the CPU stress as described in the previous section to the point that DeSiDeRaTa migrates BP guarantees that DeSiDeRaTa does not migrate BP to the *dropped* host.

Increasing the Message Count and Message Size parameters in the testApp GUI forces DeSiDeRaTa to drop undesirable hosts from the *Good Host* list as well. By varying these parameters, you should get DeSiDeRaTa to move a process to a *good host* without the need for iperf or the CPU stress.

CONCLUSION

EXPERIMENT RESULTS

The conclusive statistics indicate that the resource management system, DeSiDeRaTa, when used within this experiment, produces an elevated perception during the resource management decision process when provided with dynamic network information such as network resource requirements and availability. Experiment results also indicate that this improved resource management decision is dependent upon available bandwidth (e.g., more effective at higher usage).

The network monitoring system, Remos, is a competent device for detecting network resource use and accessibility. The integration of two Quorum program components is an effective approach to efficient resource use, which contributes to overall QoS-level provisioning. The timing of the DeSiDeRaTa and Remos components were well-matched and showed conclusively that efficient information exchange contributes to effective resource control.

Experiment results also indicate that effective use of the QUITE project framework and the tools to create the capability to install and use Quorum technologies work effectively together.

The results from this experiment include numerous run logs and the resulting illustrative chart data. The data files collected for the series consist of following:

Unmodified DeSiDeRaTa series

- PmLog—path latency management log from DeSiDeRaTa’s QoS Manager
- QMS Log—test application output log messages
- Test notes—notes on the test runs taken during the experiment

Modified DeSiDeRaTa series

- PmLog—path latency management log from DeSiDeRaTa’s QoS Manager
- QMS Log—test application output log messages and Remos bandwidth data
- Test notes—notes on the test runs taken during the experiment

After the experiment, the data files were processed to support analysis. The QMS logs were processed to extract data from their original XML format into data files that were suitable for insertion into Excel[®] spreadsheets or some type of database. The QMS log’s original data had a name of the form Runxxxqms.log. The extracted bandwidth data files were of the form Runxxxqms-BW.out. Note that there were no bandwidth data for the series with unmodified DeSiDeRaTa. The extracted test application output message data file was of the form Runxxxqms.out.

The PmLog files were processed to remove superfluous ID information and provided the data in columnar format. The raw data files had a name of the form RunxxxpmLogfile.log. The processed files had a name of the form RunxxxpmLogfile-proc’d.log.

The experiment test notes are in two files: RunI-notes and RunJ-notes. The Run I notes are for the series with the modified DeSiDeRaTa and Remos combination. The Run J notes are for the series with the unmodified DeSiDeRaTa.

These files are provided for analysis in a set of Winzip format files that support their efficient transport. They can be checked out of the Quite CVS repository from the CVS module Experiments/Desi-Remos.

DATA

Besides simple formatting of the data, the data gathered must be processed to analyze it appropriately. This processing is needed due to differences in time stamps and the frequency with which the data were recorded.

PmLogs: Desi QoS Manager path latency data

These data were output whenever a completed path message from the Test Application was processed by Desi. During these experiments, this processing occurred anywhere from twice per second up to about 10 times per second. These data must be thinned manually to allow proper comparison with the test application output data. Messages indicating when the QoS Manager recommended moving an application must be removed from the data.

QMS Logs: Test Application Data

These data were output every 5 seconds by default. For the test, the rate was set manually during the test to once per second. These data were output approximately once per second when set, but some seconds are missed due to scheduling and other uncertainties in the test application. These data must be manually formatted to allow graphing, mostly by adding blank records as placeholders for missing second records. Note that the field used to support data editing was the Event Time field.

These logs and graphs are exceptionally large in size and could not be feasibly included within this report. However, they can be accessed through the QUITE project by contacting Teknowledge or SSC San Diego.

REFERENCES

- DeWitt, A., T. Gross, B. Lowekamp, N. Miller, P. Steenkiste, J. Subhlok, D. Sutherland. 2000. "Remos: Resource Monitoring for Network-Aware Applications," CMU-CS-97-194, Carnegie Mellon School of Computer Science, Pittsburgh, PA.
<http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/cmcl/www/remulac/remos.html>
- Welch, L., et al., 2000. "DeSiDeRaTa: Resource and QoS Management for Dynamic, Scalable, Dependable Real-Time Systems: Manual for Use of Distributed QoS and Resource Management Middleware, Laboratory for Parallel and Distributed Real-time Systems, Department of Computer Science and Engineering, University of Texas at Arlington, and Laboratory for Intelligent, Real-Time, Secure Systems School of Electrical Engineering and Computer Science, Ohio University.

APPENDIX A EXPERIMENT SETUP AND CONFIGURATION

Installing Remos

The Management of Quality of Service experiment requires the proper installation of Remos. The instructions below will allow for the setup and configuration of the environment for Remos execution. These directions should be followed in the sequence noted below.

1. Edit `quite-config/ENV.sh` to reflect the correct path for `REMOS_DIR` and the `PATH` variable for the location of JDK 1.2.2. Also edit the value for `COLLECTOR` to reflect the correct host name and the same port as in the `collector.config` file.
2. Source the `ENV.sh` file by "`source ENV.sh`" in bash.
3. Edit `$REMOS_DIR/Makefile.shared` to locate `ROOT_DIR` - the location of this Remos distribution. The Quite configuration assumes that JDK1.2.2 is located in `/usr/local/share/jdk1.2.2`. If this is not so, edit the `Makefile.shared` to change this path as well in the Linux section.
4. `cd` to the Remos root directory (`$REMOS_DIR`) and run "`gmake rebuild_all`" Everything should compile cleanly with no errors.
5. Edit `bridge.config` to reflect the IP address of your switch and the right SNMP community string. If it is a Cisco catalyst then use the `communitystring@VLAN` convention.
6. Edit `collector.config` and change the line that starts with "`bridgecoll`" to reflect the host that is running the bridge collector. Leave the port number alone.
7. Now you are set to run Remos.
8. Verify that the "`run_bridgecoll`", "`run_collector`" and "`run_visualizer`" programs in `$REMOS_DIR/build` have executable permissions.
9. If you'd like to take Remos for a test spin, `cd` to `quite-config/utilities` and run "`make all`". This will create two binaries "`flowtest`" and "`graphtest`" which will let you query Remos for information.
10. To run Remos, here's the order you execute things - `cd` to `$REMOS_DIR/quite-config`
 - Run Bridge collector with the following command
`"$REMOS_DIR/build/run_bridge_coll ./hostfile ./bridge.config"`
 - Run SNMP collector with the following command
`"$REMOS_DIR/build/run_collector ./collector.config"`
 - Run our test tool `graphtest` with the following command
`"$REMOS_DIR/quite-config/utilities/bin/graphtest -e -i 5 -s 10 -f ./hostfile"`
Run "`graphtest -h`" for an explanation of the options.
 - To run the Java visualizer, use the following command:
`"$REMOS_DIR/build/run_visualizer ./hostfile"`

QMS and REMOS

This section describes the installation and setup of the Remos software and the STDC Remos instrumentation additions. Much of the text included here was taken from the readme files included as a part of the Remos package.

The readme files included with the standard Remos distribution are as follows:

README.desc

README.build

README.collectors

README.required

README.run

README.version

In addition to these files, STDC has provided a Readme.first file located with the Remos component in the quite-qvs repository. This file is located in the Remos/quite-config directory.

Building Remos from Scratch

Most likely, the Remos and the Remos instrumentation will be deployed as executables on the QUITE distribution. If this is the case, skip to the next section, *Configuring Remos*. However, if you want to build Remos *from scratch*, this section covers the steps required to do so. This section will present excerpts from the instructions found in the above files on building Remos from scratch. Please refer to the official Remos readme files listed above and the STDC Readme.first file located in the quite-config directory for more detailed information.

First, install the remos software. This software usually comes in a tar or gzip format. Follow the instructions as detailed in README.build for detailed instructions on how to build the Remos software. The main requirements to build the package are gcc 2.7.2 or higher (ideally, egcs 1.0 or later), gmake (ideally, 3.7 or later), and jdk 1.2.

1. Edit quite-config/ENV.sh to reflect the correct path for REMOS_DIR and the PATH variable for the location of JDK 1.2.2. Also edit the value for COLLECTOR to reflect the correct host name and the same port as in the collector.config file.

2. Source the ENV.sh file by "source ENV.sh" in bash.

3. Edit \$REMOS_DIR/Makefile.shared to locate ROOT_DIR - the location of this Remos distribution. The Quite configuration assumes that JDK1.2.2 is located in /usr/local/share/jdk1.2.2 . If this is not so, edit the Makefile.shared to change this path as well in the Linux section.

4. cd to the Remos root directory (\$REMOS_DIR) and run "gmake rebuild_all". Everything should compile cleanly with no errors.

5. Edit bridge.config to reflect the IP address of your switch and the right SNMP community string. If it is a Cisco catalyst then use the communitystring@VLAN convention.

6. Edit `collector.config` and change the line that starts with "bridgecoll" to reflect the host that is running the bridge collector. Leave the port number alone.

Configuring Remos

If the Remos with `Remos_instrumentation` package is delivered as a "ready to run" executable package, then building Remos from scratch is not necessary.

1. On the Linux machine, `cd` to directory you want to install Remos.
2. Untar the Remos distribution using the command `tar xvzf remos-all.tar.gz`. Please note that this will create the following subdirectories in the current directory:

Bin

Scripts

Config

Lib

3. In the scripts directory, open the file `envsetup.sh`. Edit this file to reflect the correct path for `TOP_DIR` and the `JDK12_HOME` for the location of JDK 1.2.2. Also edit the value for `COLLECTOR` to reflect the correct host name and the same port as in the `collector.config` file (see step 5).

4. In the config directory, edit `bridge.config` to reflect the IP address of your switch and the right SNMP community string.

5. Edit `collector.config` in the config directory and change the line that starts with "bridgecoll" to reflect the host that is running the bridge collector. Leave the port number alone. For example, it should read something similar to:

```
bridgecoll teksd4 41271
```

TEST APPLICATION AND DeSiDeRaTa

This section describes the installation and configuration of the test application and the DeSiDeRaTa middleware modified for use in this experiment. Please note that the QMS dll is also included in the distribution tar file.

This document refers to the *run-time* installation of the TestApplication software and does not cover the building of the software from source code. This *run-time* version will normally be distributed as a zip file. Upon *unzipping* the file, the user will see the following subdirectories (*note: our top directory is desi-cvs-2000, but that may be different on your machine*):

/bin

/config

¹ We experienced a problem in our setup where we received the message: "Network Unreachable." We did the following: `Route add -net 224.0.0.0 netmask 240.0.0.0 dev eth0`.

/lib

/scripts

The QUITE-modified version of DeSiDeRaTa is included in the TestApplication zip file. Setting up the test application and the DeSiDeRaTa middleware to work in your environment mainly involves re-configuring the batch files in the *scripts* directory. In *envsetup.bat*, modify the following variables to match your local environment:

```
set TOP_DIR=d:\desi-cvs-2000
set COTS_HOME=c:\cots
set NT_HOME=C:\WINNT
set VCPP_HOME=c:\program files\Microsoft Visual Studio
set TAO_NAMING_HOST=teksd8

REM desi configuration
set DESI_NS_HOST=teksd8
set NS_MULTI_PORT=10013
set RMSPECDIR=%TOP_DIR%\config\specfiles

set JDK1.1_HOME=%COTS_HOME%\jdk1.1.7B
set JDK1.2_HOME=%COTS_HOME%\jdk1.2
set JDK1.3_HOME=%COTS_HOME%\jdk1.3
```

Note that this *envsetup.bat* file is called by *envsetup1.3.bat*. Since the test application GUI requires jdk/jre 1.1.3, the *testapp_gui.bat* (which starts the Controller GUI) calls *envsetup1.3.bat* which in turn calls *envsetup.bat*.

Additionally, there is a file called *naming_util.properties* located in the *config* directory. This file is used by the test application GUI controller to connect to TAO naming. Replace the IP address with the IP address of the machine where the TAO naming server is running.

DeSiDeRaTa is configured via *specfiles*. The *specfiles* used in DeSiDeRaTa reside in the *config/specfiles* directory. The main things to modify are found in *DesiderataHW.spec* and *TestApp.spec*. In the *DesiderataHW.spec* file modify the following variables to match your environment:

```
HOST teksd12 {
    Type "INTEL_PENTIUM_500";
    OS "MS_Windows_NT";
    Version "MS_Windows_NT_4.0";
    Speed 500; //MHz
    Memory 256; //MB
    NumCPUs 1;
    Threshold 0.1 ;
    SPECint95 20.05 ;
    SPECfp95 14.2 ;
    Default-Network D_N:NH_250_Ethernet
}
```

This file defines all the NT machines that will be potential targets for migration of processes. It is important that the SPECint and SPECfp values match your machines. To find the values for your system, visit www.spec.org. For more detailed information about these and other parameters, see the DeSiDeRaTa documentation.

The *TestApp.spec* file details information about the test application. DeSiDeRaTa uses this information along with the hardware information specified above to arrive at decisions about resource management. Make sure that the hostnames and machine types match those defined in the *DesiderataHW.spec* file. Please refer to the RM-Spec manual in the DeSiDeRaTa distribution for detailed information about each of these parameters.

Make sure that all hosts that you want DeSiDeRaTa to consider a potential migration target are listed under the Application subsection. For example, in the following section defining the CP process:

```
APPLICATION CP {  
  
    ... [ text deleted ]  
  
    //list of hosts that are the only h/w set up to run this app  
    Host lan_teksd10;  
    Host lan_teksd6;  
    Host teksd21;  
    Host lan_teksd11;  
    Host teksd5;  
  
    ... [ text deleted ]  
}
```

The CP process can migrate to any of these 5 machines: lan_teksd10, lan_teksd6, teksd21, lan_teksd11 or teksd5. If the subsection were specified as follows, the CP process would only be able to migrate to teksd21 and teksd5:

```
APPLICATION CP {  
  
    ... [ text deleted ]  
  
    //list of hosts that are the only h/w set up to run this app  
    Host teksd21;  
    Host teksd5;  
    ... [ text deleted ]  
}
```

IMPORTANT NOTE: One important thing to remember is that *DeSiDeRaTa* parses ALL files ending in the *.spec* extension in the specified directory. Therefore, it is important to rename or remove any unwanted *spec* files; otherwise, *DeSiDeRaTa* will parse them.

In the *TestApp.spec* file, the arguments to the test application are as follows:

```
//Ordered list of command line arguments
Arg "test_app"; // NAME OF THE EXECUTABLE
Arg "BP";      // NAME OF THIS NODE IN PATH
Arg "3";       // NUMBER OF MESSAGES TO PASS EACH CYCLE
Arg "10";      // CURRENTLY, NOT USED
Arg "d:\desi-cvs-nt\config\input_message.txt"
// PATH TO FILE WHICH DEFINES THE MESSAGE STRING
Arg "teksd10"; // DESI NAME SERVER HOST
Arg "7300";    // DESI NAME SERVER PORT
Arg "H:W:Testing"; // DESI QM NAME
Arg "AP";     // PREVIOUS NODE
Arg "CP";     // NEXT NODE
Arg "dummy";  // DUMMY STRING REQUIRED BY DESI
```

Note that a default value is used for the array size (200) of items to sort and the number of iterations (10) it is sorted. These values can be modified dynamically via the test application GUI.

The fifth argument defines a path to a textfile which will store a user-defined string to be used as the message to pass to the next node in the path. In the example above, the path is defined as “d:\desi-cvs-nt\config\input_message.txt “. Note that this argument is passed to all *Desi Startup Daemons* therefore this exact filepath must exist on all machines that will be running this particular processing node. Alternatively, you can specify “*DEFAULT_INPUT*” instead of a filepath. In that case, the test application will use the default string *Experiment 1* instead of reading one from a file.

Adding additional parts to the path (advanced / optional section)

One important thing to note is that each process in the application path is defined in the *TestApp.spec* file. That is, AP, BP, CP, and DP each have an APPLICATION section. If desired, it is possible to create more parts to the path by performing the following steps. Note that this procedure is *not* required to perform the experiments and is described here only in the case that the user wants to conduct *modified* versions of the experiments on their own.

In the TestApp.spec file, modify the PATH Testing section to include the additional pieces. For example, the section currently states the following:

```
PATH Testing {
  Connectivity {
    (H:W:AP, H:W:BP);
    (H:W:BP, H:W:CP);
    (H:W:CP, H:W:DP);
  }
}
```

To add an additional piece to the path, this section must be changed to the following:

```
PATH Testing {
  Connectivity {
    (H:W:AP, H:W:BP);
    (H:W:BP, H:W:NEWONE);
    (H:W:NEWONE, H:W:CP);
    (H:W:CP, H:W:DP);
  }
}
```

Notice that *H:W:NEWONE* has been added to the path.

Next, add an additional *APPLICATION* subsection describing the *H:W:NEWONE* node. Again, make sure that the *hostnames* and *machine types* match.

In the section describing the ordered list of command line arguments, make sure that the second argument lists the name you wish to give to this new node. For example:

```
//Ordered list of command line arguments, static
  Arg "test_app";
  Arg "NEWONE"; // New node!!!
  Arg "3";
  Arg "10";
  Arg "teksd10";
  Arg "7300";
  Arg "H:W:Testing";
  // next process name
  Arg "AP";

  // previous process, in this case - this is the end path (keyword) for app
  to find out if it

  // needs to invoke the next process. Again, in this case, it would not
  invoke the next process
```

```

    Arg "CP";

    // need to provide this dummy arg because some strange
    //behaviour of startup daemon the way it handled the arguments

    Arg "dummy";

```

Note that the first argument that describes the actual executable that runs is the same for all of the applications, *test_app.exe*. This is because we use one single executable for all pieces of the path customized by command line argument.

Again, please refer to Rmspec documentation for detailed information on the meaning of the DeSiDeRaTa parameters.

ENVSETUP1.3 BATCH FILE

```

call envsetup1.3.bat

rem this now has been defined in envsetup.bat

rem set RMSPECDIR=%CONFIG%\specfiles

set HOST_PAIR_FILE=nb-host-pair.txt

start "DeSiDeRaTa Name Server" /min NS.exe %DESI_NS_PORT%

pause

start "DeSiDeRaTa System Broker" /min SB.exe SB %DESI_NS_HOST% %DESI_NS_PORT%

pause

start "DeSiDeRaTa Resource Manager" /min RM.exe RM %DESI_NS_HOST%
%DESI_NS_PORT% 7

pause

start "DeSiDeRaTa Hardware Broker" /min HB.exe HB 5 %DESI_NS_HOST%
%DESI_NS_PORT%

pause

start "DeSiDeRaTa Network Analyzer" NA.exe %DESI_NS_HOST% %DESI_NS_PORT% HA NB
60000000

pause

start "DeSiDeRaTa Host Analyzer" /min HA.exe %DESI_NS_HOST% %DESI_NS_PORT% RM
HB NA

rem start "DeSiDeRaTa Hardware Analyzer" /min HA.exe %DESI_NS_HOST%
%DESI_NS_PORT% RM HB

pause

```

```
start "DeSiDeRaTa Network Broker" NB.exe 4 0 %DESI_NS_HOST% %DESI_NS_PORT% NA  
%HOST_PAIR_FILE%
```

```
pause
```

```
start NTMonitor.exe 5 HB %DESI_NS_HOST% %DESI_NS_PORT%
```

```
pause
```

```
start "DeSiDeRaTa QM_Testing" QM.exe N RM SB %DESI_NS_HOST% %DESI_NS_PORT%  
H:W:Testing F
```

```
start "DeSiDeRaTa Program Control" /min PC.exe PC RM %DESI_NS_HOST%  
%DESI_NS_PORT%
```

```
pause
```

```
start "DeSiDeRaTa Startup Daemon" /min SD.exe PC %DESI_NS_HOST% %DESI_NS_PORT%
```

```
REM pause
```

```
REM start "DeSiDeRaTa HCI" /max HCI.exe
```

REPORT DOCUMENTATION PAGE

*Form Approved
OMB No. 0704-01-0188*

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden to Department of Defense, Washington Headquarters Services Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE (DD-MM-YYYY) 04-2002		2. REPORT TYPE Final		3. DATES COVERED (From - To)	
4. TITLE AND SUBTITLE MANAGING QUALITY OF SERVICE WITHIN DISTRIBUTED ENVIRONMENTS				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHORS The Open Group Development Corp. Research Institute				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) SSC San Diego San Diego, CA 92152-5001				8. PERFORMING ORGANIZATION REPORT NUMBER TR 1883	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency Information Technology Office 3701 Fairfax Drive Arlington, VA 92203				10. SPONSOR/MONITOR'S ACRONYM(S) DARPA-ITO	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT The objective of this research was to construct an experiment that examined efficiently managing QoS within distributed environments based upon network information. Experiment results described in this report have successfully demonstrated individual Quorum goals using integrated Quorum components. This experiment has integrated various Quorum technologies from low level to high level and created a layered resource management system. Results can be accessed through the QUITE framework and toolkit. These components, which include Remos and DeSiDeRaTa, were integrated correctly to introduce new multi-dimensional QoS management capabilities.					
15. SUBJECT TERMS Mission Area: Software Development distributed environment network information Quorum components resource management					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
U	U	U	UU	52	(619) 553-4131

INITIAL DISTRIBUTION

20012	Patent Counsel	(1)
20271	Archive/Stock	(6)
20274	Library	(2)

Defense Technical Information Center
Fort Belvoir, VA 22060-6218 (4)

SSC San Diego Liaison Office
C/O PEO-SCS
Arlington, VA 22202-4804

Center for Naval Analyses
Alexandria, VA 22311-1850

Office of Naval Research
ATTN: NARDIC (Code 362)
Arlington, VA 22217-5660

Government-Industry Data Exchange
Program Operations Center
Corona, CA 91718-8000

Defense Advanced Research Projects Agency
Information Technology Office
Arlington, VA 22203-1714

Naval Postgraduate School
Monterey, CA 93943-5101

Teknowledge
Palo Alto, CA 94303 (3)

The Open Group
Woburn, MA 01801 (3)

System/Technology Development Corporation
Herndon, VA 20170-4214 (3)

Approved for public release; distribution is unlimited.