Monitoring Network QoS in a Dynamic Real-Time System†

Hong Chen‡, Brett Tjaden‡, Lonnie Welch‡, Carl Bruggeman‡, Lu Tong‡, Barbara Pfarr‡

†School of Electrical Engineering and Computer Science
Ohio University, Athens, OH 45701
{hc152787|tjaden|welch|bruggema|lt356587} @ ohio.edu

‡Real-Time Software Engineering Branch
NASA GSFC/Code 584, Greenbelt, MD 20771
Barbara.B.Pfarr.1@gsc.nasa.gov

Abstract

This paper presents our design and tests of a real-time network monitoring program for DeSiDeRaTa, an existing resource management system. This monitor will assist DeSiDeRaTa in maintaining an acceptable Quality of Service (QoS) for groups of real-time applications by reporting the communication delays caused by inadequate network bandwidth. The network monitoring application we developed uses SNMP and network topology information gleaned from the DeSiDeRaTa application specification files. Network bandwidth utilization of each real-time communication path is computed, and experiments have been run to demonstrate the accuracy of these measurements.

1. Introduction

Significant work has been done on resource management (RM) of distributed real-time computer systems. Such systems usually contain a large number of computers connected by one or more computer networks. Maintaining adequate quality of service (QoS) for real-time applications requires the resource management middleware to manage not only computing resources but also network resources. In many cases, proper management of network resources is vital for delivering adequate QoS to a dynamic real-time system. A large amount of data communication between the computers and improper use of network resources can lead to congestion and delay, and ultimately a QoS violation. To properly manage network resources, the resource management middleware must monitor their performance.

This paper focuses on a technique for monitoring network resources. It extends the work of DeSiDeRaTa, an existing resource management middleware solution for dynamic, scalable, dependable, real-time systems [1]. DeSiDeRaTa assumed that network resources were never a bottleneck. Our network monitoring program provides real-time network performance information to the resource management middleware and helps the middleware to detect and diagnose potential QoS violations due to a sub-optimal allocation of network resources.

To obtain network performance information, the network resource monitoring program must know the topology of the computer network. DeSiDeRaTa includes a specification language with which to specify the components of a real-time system. For our work, we have extended the specification language to include network-related information of the real-time system, such as computer hosts, network devices, network interfaces, and network connections. Our network monitoring program obtains the topology and connectivity of the real-time system from this specification file. The monitor implements an algorithm that traverses the communication path between hosts based on the topology information in the specification file, and calculates the bandwidth (both available and used) between pairs of hosts.

This real-time network performance is obtained by querying network components periodically using the Simple Network Management Protocol (SNMP) to gather performance information from hosts and network devices. Combining the SNMP query results and network topology information, bandwidth statistics are then calculated.

The outline of the remainder of this paper is as follows. Section 2 presents some background on SNMP and the DeSiDeRaTa resource manager. Section 3 discusses the algorithms and implementation details of our approach. Section 4 provides some preliminary test results for our network monitor. Section 5 presents our conclusions and gives some thoughts on future work.

† Funded by NASA Office of Earth Science, NASA Research Announcement 99-OES-08.
2. Background

2.1. Network Monitoring

We considered many different network monitoring techniques as the basis for our work. SNMP is a traditional technology and it has been widely used for network monitoring [2-4]. Other techniques, such as Java [5,6], agent technology [7], switch monitoring [8], were also considered. We chose SNMP for its simplicity and comprehensiveness.

2.2. SNMP

The Simple Network Management Protocol (SNMP) [9] provides a basic network-management tool for TCP/IP-based environments. SNMP defines the structure of a hierarchically organized database, which is stored in a series of network components. Each network component contains information such as services offered, the device’s routing table, and a wide variety of statistics. Such information can be accessed through a local Management Information Base (MIB) [10] that specifies the structure of the database. More information about SNMP and MIB can be found in [11].

Our network monitor uses the SNMP protocol and the interface table of the MIB to obtain data communication statistics from each SNMP-enabled network component. The interface table provides a static data transmission rate as well as counters for data and packets being sent and received through each network interface. Real-time network information is obtained by polling SNMP servers on each device (e.g., hosts, switches, routers) periodically. This data can then be used to determine the amount of bandwidth used and available for the network component. By combining these metrics (see section 3.3), we compute the available and used bandwidth of a real-time path.

2.3. Resource Management and Network Topology

The work in this paper is part of the effort to build DeSiDeRaTa, adaptive resource management middleware for dynamic, scalable, dependable real-time systems [1]. The middleware performs QoS monitoring and failure detection, QoS diagnosis, and reallocation of resources to adapt the system to achieve acceptable levels of QoS. The current version of DeSiDeRaTa middleware manages only computational resources and assumes no QoS violation is caused by network delays. Our work extends DeSiDeRaTa to allow the management of network resources. We developed an application that does network QoS monitoring and provides the middleware with network metrics regarding data communication information, which enables the middleware to manage the resources based on the network metrics and network QoS specification.

To obtain the network metrics, the network monitoring software has to be able to discover the network topologies and combine them with SNMP data from each network component. Network topology discovery usually is difficult due to the complexity of computer networks. In the DeSiDeRaTa environment, this problem can be solved easily using some of RM infrastructure already in place. The resource management middleware must know details about hardware systems and all the software applications under its control. A specification language was developed to describe such information of hardware and software systems, including network connections and interfaces. The network monitoring software can obtain this network information from the specification files and construct the network topology graph for the system.

3. Methodology

3.1. SNMP Polling

The goal of the network-monitoring program is to obtain real-time network bandwidth usage information. SNMP information can be polled from an SNMP-enabled host or network device (i.e., the host/device has an SNMP demon running). Table 1 shows some of the MIB-II (second version of MIB) objects being accessed during our periodic SNMP polling.

The data transmitted through an interface in both directions, as well as the static bandwidth, can be obtained using these MIB-II objects. Because the polling results are cumulative numbers, this data has to be polled periodically. The old value is subtracted from the new one to determine statistics for the polling interval. The time interval between two polling processes can be found using the system uptime data. The data transmission rate, including packets and bytes per unit time can then be calculated.

3.2. Network Topology and Specification Language

The network-monitoring program was built using a local area network (LAN) model. The topology of such networks is modeled using hosts (or network devices), network interfaces, and network connections.

Figure 1 shows the LAN topology model. Each host or network device has one or more network interfaces. For example, in the figure, hosts A, C, and E each have a single connection (interface) to the network, while B and D have multiple interfaces. B and D can be hosts with multiple network connections, or network devices such as...
A network connection is specified as a pair of interfaces that are physically connected to each other. In this model, the connection must be 1-to-1, i.e., one interface may only be connected to one interface on another host/device.

Utilizing the DeSiDeRaTa specification language is a straightforward approach to obtain network topology. Pure network discovery is not feasible in the DeSiDeRaTa environment because the resource management middleware has to know exactly what resources are under its control, and this requires at least some level of specification of resources. A hybrid approach may be a better solution in the future, however, due to its complexity, we chose the simpler solution using the specification language for this stage of the research.

A new extension to the DeSiDeRaTa specification language [12] was developed to describe the topology of the network resources of the real-time system under control of the resource management middleware. The network topology is defined using the model described above.

### Table 1. MIB-II Objects Used in Network Monitoring.

<table>
<thead>
<tr>
<th>MIB-II Object (Numbers)</th>
<th>Description [10]</th>
</tr>
</thead>
<tbody>
<tr>
<td>system.sysUpTime (1.3.6.1.2.1.1.3)</td>
<td>The time (in hundredths of a second) since the network management portion of the system was last re-initialized.</td>
</tr>
<tr>
<td>interfaces.ifTable.ifEntry.ifSpeed (1.3.6.1.2.1.2.1.5)</td>
<td>An estimate of the interface's current bandwidth in bits per second (static bandwidth).</td>
</tr>
<tr>
<td>interfaces.ifTable.ifEntry.ifInOctets (1.3.6.1.2.1.2.1.10)</td>
<td>Accumulated number of octets received on the interface.</td>
</tr>
<tr>
<td>interfaces.ifTable.ifEntry.ifInUcastPkts (1.3.6.1.2.1.2.1.11)</td>
<td>Accumulated number of subnetwork-unicast packets delivered to a higher-layer protocol.</td>
</tr>
<tr>
<td>interfaces.ifTable.ifEntry.ifOutOctets (1.3.6.1.2.1.2.1.16)</td>
<td>Accumulated number of octets transmitted out of the interface.</td>
</tr>
<tr>
<td>interfaces.ifTable.ifEntry.ifOutNUcastPkts (1.3.6.1.2.1.2.1.17)</td>
<td>The total number of packets that higher-level protocols requested to be transmitted to a subnetwork-unicast address.</td>
</tr>
</tbody>
</table>

Figure 2 shows the pseudo code for data structures that specify the network topology. A host/device is specified by its name, a list of all network interfaces on the host, and other host information. The interfaces are distinguished by their unique local names. A network connection is specified as two host-interface pairs, which give the two ends of the connection. Finally, the network topology can be described as a list of all the hosts/devices and all the network connections among them.

The specification file is parsed, and related network topology information is then passed to the network-monitoring program.

### 3.3. Path Traversal and Bandwidth Calculation

Based on the information from the specification language, the communication path between two hosts can be traversed. A simple recursive algorithm is designed to traverse the path, with a necessary infinite-loop detecting function implemented. The result of the path is described as a series of network connections defined in the previous section.

To calculate the available bandwidth between two hosts, one has to know the bandwidth of each connection in the path. The available bandwidth of the whole path is simply the minimum of all the individual available bandwidths. Assume a communication path consists of \( n \) network connections, and the available bandwidth for connection \( i \) is \( a_i \) (i = 1, 2, \ldots n). Then the available bandwidth of the path, \( A = \text{minimum}(a_1, a_2, \ldots a_n) \). For each individual connection, the available bandwidth \( a_i \) is just the difference between maximum bandwidth \( m_i \) and used bandwidth \( u_i \). A measure of \( m_i \) can be obtained directly through SNMP polling, while the \( u_i \) must be computed by the network monitoring program.

It is relatively easy to calculate used bandwidth of a host connected to switches because a switch does not forward packets for one host to other hosts connected to the same switch. Hence, the amount of bandwidth used on
a host connected to a switch is simply the amount of data transmitted as reported by SNMP polling from either the host or the switch. If the traffic reported is $t_i$, then we simple have $u_i = t_i$. The traffic of another connection $t_j$ ($j\neq i$) will not affect $u_i$.

However, for hosts connected to hubs, all packets that go through the hub will be sent to every host connected to the hub. Therefore, the amount of bandwidth used for a host connected to a hub is the sum of all the data sent to the hub, instead of just the one to this host. Assume there are $n$ hosts connected to the hub and the traffic reported by SNMP polling for each host is $t_i$ ($i = 1, 2, ... n$). Then $u_i = t_1 + t_2 + ... + t_n$. Notice that $u_i$ cannot exceed the maximum speed of the hub.

An algorithm was implemented in our network monitoring program to distinguish these two cases and calculate available bandwidth accordingly. The available bandwidth between two hosts is calculated by traversing the communication path between the two and computing the minimum value.

4. Experiments and Results

4.1. Experimental Setups

The experiments described below were performed in the Laboratory for Intelligent Real-time Secure Systems (LIRTS) at Ohio University. The network is a LAN
system with one 100 Mbps switch and one 10 Mbps hub. As shown in Figure 3, one Linux machine (L1), two Solaris 7 machines (S1, S2), and four Solaris 8 machines (S3-S6) are connected to the switch. Two other Windows NT machines (N1 and N2) are connected to the hub, which is connected to the switch.

Our network monitoring program was running on the Linux machine L1. SNMP demons were available on L1, N1, N2, S1, S2, and the switch at the time of experiment. Such a network arrangement is sufficient for monitoring the bandwidth between any pair of hosts in the system. For example, even though there is no SNMP demon on either S4 or S5, the bandwidth between S4 and S5 can still be monitored by polling the interfaces on the switch that are connected to S4 and S5.

4.2. Network Load Generator

To test our network monitoring program, a simple network load generator program was developed. It sends data streams to a designated host at a given speed. The data are sent as UDP packets to the DISCARD port (UDP port number 9) on the host. The real speed of traffic generated by the program is slightly larger than the specified value due to the extra bytes of UDP and IP headers and acknowledgements. The size of these extra bytes may vary depending on the speed of data traffic, but is small compared to the amount of data being sent in our experiments.

4.3. Experiment Results

Preliminary experiment results are presented in this section. Communication paths through both the hub and switch were tested, and end-to-end bandwidth was observed using our network monitoring program, and compared to expected values.

4.3.1. Dynamically varying network load. A set of experiments was performed to observe the network traffic between a Windows NT machine, N1, and the Solaris 7 machine, S1. The communication path between these two hosts was computed by our program using network topology information from the specification file. The path that data followed was: S1 – switch – hub – N1 (Figure 3).

To check the correctness of the network bandwidth usage reported by our network monitoring program, network traffic was generated from L1 to N1 using the network load generator. Starting at 0 Kbytes/second for 120 seconds, we increased the amount of data sent by the load generator by 100 Kbytes/second each 60 seconds. After 360 seconds, the load generator was sending 500 Kbytes/second from L1 to N1. The entire load was eliminated at 420 seconds (Figure 4a).

The output of our network monitor exhibits a similar pattern to the actual amount of data being transmitted (Figure 4b). The reported value of used bandwidth is slightly larger than the generated traffic load due to the background traffic over the network and the extra data in the packet headers. The fluctuation and spikes seen in Figure 4b is caused by the fluctuation of data transmission over the network and a slight delay in SNMP polling.

Table 2 gives some statistics of the measured results. The background traffic was calculated as the average of measured values at 0 generated load. The average traffic was obtained for different generated load by subtracting the background from the average of measured traffic. The average measured load less background was about 4% larger than the values of generated load. Part of the

![Figure 4](4.png)

**Figure 4.** Experiment results for our network monitoring program. (a) Pattern of traffic load generated by the load generator. (b) Measured traffic between hosts according to our network monitoring program.

<table>
<thead>
<tr>
<th>Generated Load (Kbytes/second)</th>
<th>Average Measured Load (Kbytes/second)</th>
<th>Average Load Less Background (Kbytes/second)</th>
<th>% Error</th>
<th>Maximum % Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11.824</td>
<td>103.639</td>
<td>3.64%</td>
<td>16.4%</td>
</tr>
<tr>
<td>100</td>
<td>115.463</td>
<td>207.514</td>
<td>3.76%</td>
<td>8.4%</td>
</tr>
<tr>
<td>200</td>
<td>322.512</td>
<td>310.688</td>
<td>3.56%</td>
<td>16.6%</td>
</tr>
<tr>
<td>300</td>
<td>428.280</td>
<td>517.454</td>
<td>4.11%</td>
<td>5.1%</td>
</tr>
<tr>
<td>500</td>
<td>529.278</td>
<td>517.454</td>
<td>3.49%</td>
<td>5.7%</td>
</tr>
</tbody>
</table>

**Table 2.** Statistics of Measured Traffic Load (Kbytes/second).
difference is due to the packet headers of generated traffic. The IP and UDP headers in a system with 1,500-byte MTU size can contribute about 2%. The other 2% may come from the traffic caused by SNMP queries and acknowledgements.

Table 2 also shows maximum percentage error of individual value of measured traffic. The large error (16%) was caused by delays in SNMP polling. Occasionally, some data bytes are counted in a later SNMP message instead of an earlier one, resulting in an abnormally small value followed by and abnormally large one.

4.3.2. Hosts connected by a hub. A hub forwards data packets to all the connected hosts, not just the one for which a packet is destined. This affects the bandwidth of all hosts connected to a hub if data is sent to any host connected to the same hub. Our monitoring program considers this by summing the traffic through a hub when computing the amount of bandwidth used on any communication path through the hub.

An experiment was run to monitor the amount of bandwidth used by the two Windows NT machines connected to the hub. Data was sent from the Linux machine to the two NT machines as shown in Figure 5a-b. We started with no data being sent to either NT machines. After 20 seconds, we began to send 200 Kbytes/second from L1 to N1. 20 seconds later, we began to send 200 Kbytes/second from L1 to N2. After another 20 seconds, the traffic from L1 to N1 was reduced to 0. 20 seconds later the traffic from L1 to N2 was also eliminated. The observed traffic load for the two paths (S1—N1, S1—N2) is as we expected.

The statistical analysis, the same as the one in previous section, shows 3.7% error on average values of measured traffic (less background), with maximum individual error of 7.8%.

4.3.3. Hosts connected by a switch. A switch only forwards packets to the host for which they are destined, not all the hosts connected to the switch. Therefore, our monitoring program treats switches differently than hubs. The traffic through a switch is not summed up. Instead,

![Figure 5. Experiment results for hub-connected hosts. (a-b) Patterns of traffic loads generated by the load generator. (c-d) Measured traffic between hosts according to our network monitoring program.](image)

![Figure 6. Experiment results for switch-connected hosts. (a-c) Patterns of traffic loads generated by the load generator. (d-e) Measured traffic between hosts according to our network monitoring program.](image)
only traffic going to and from a particular host is considered when computing the amount of bandwidth being used.

An experiment was performed on two hosts connected by the switch. Traffic between S1 and S2, and between S1 and S3 was measured by our program. All three machines are Solaris machines and are connected to the switch. 2,000 Kbytes/second of traffic was generated at time 20-60, 40-80, and 100-120 seconds from L1 to S2, S3, and S1 respectively (Figure 6a-c). As shown in Figure 6d-e, the load sent to S2 can only be seen between S1 and S2, and the load to S3 appears only between S1 and S3, while the load to S1 is present in both paths because S1 has only one connection to the switch.

The statistical analysis shows 2.2% error on average values of measured traffic (less background), with maximum individual error of 7.8%. The smaller percentage error on average values is due to the much larger volume of traffic generated compared to the previous experiment.

5. Conclusions and Future Work

This paper has presented our approach to monitoring the network QoS in a dynamic real-time system. Our network monitoring program uses SNMP and network topology information, and computes network bandwidth usage information of a communication path in real-time. Different algorithms are used for host pairs connected by different network devices (hub and switch). The results of experiments on a LAN environment show the effectiveness and correctness of our program. Future work includes measurement of network latency, network QoS violation detection, dynamic network topology discovery, and distributed network monitoring.

6. References