A Remote Resource Monitoring Approach in Active Networks

Yuhong Li, Lars Wolf, and Fangming Xu

Institute of Operating Systems and Computer Networks
Braunschweig University of Technology
38106 Braunschweig, Germany
{li,wolf}@ibr.cs.tu-bs.de, f.xu@tu-bs.de

Abstract. Monitoring the resource status in active network nodes is important both for the network administrator to maintain and deploy active networks and for the normal end-users to acquire better performance. Hence, a method that can monitor the resource status at a remote node is needed. Moreover, due to the complexity of the resource usage in active networks, the corresponding method should be able to monitor the status of various resources at remote node dynamically and in a user-specified way. This paper presents the design and implementation of such a resource monitoring method. The method adopts the active application technique to obtain the desired resource status information at a remote network node and transfer them back to the user in the user-specified way. It also allows the user to control the content, the procedure and the precise of the monitoring.

1 Introduction

In active networks, the resource status information at an active node is important for both the end-users and the network administrators. Active applications use the networks more aggressively than the simple applications in the traditional passive networks, for example, they need various types of resources to inject and execute their programs in the networks. Therefore their performance becomes more sensitive to the network conditions, such as the available resources at the network nodes. In order to acquire better performance, these applications may need sufficient information about the resource status within the networks. For the network administrators, for instance, if they have enough information about the different types of resources at the concerned network nodes, they can decide dynamically when, where and what kind of services can or cannot be executed, so that the congestion and scarcity of a specific type of resources in a certain time can be avoided. In addition, the resource status may be also beneficial for the provision of new services, such as routing at an active node according to the resource status at the neighbor nodes etc. Hence, a method that can monitor the resource status at a remote node is necessary in active networks.

Due to the complexity of the resource usage at active nodes, e.g., more types of resources are needed by an application simultaneously, it is difficult to gather, describe and express the resource status information. Moreover, different end-users and network administrators are interested in different resource status information at

A. Marshall and N. Agoulmine: MMNS 2003, LNCS 2839, pp. 462–474, 2003.

different time. This requires that no constraints should be employed on the content and format of the resource information needed to be observed using this monitoring method. Furthermore, users should also be able to control the procedure of the monitoring, e.g, to specify or change the content of the information desired to be observed or the precise of the monitoring at run time.

Some efforts have been made in the traditional IP-based networks to gather information in the networks, such as using the implicit network feedback (e.g., the packet loss rate), the well-known traceroute program and the SNMP (Simple Network Management Protocol). However, they are not satisfying because of limitations concerning the amount, content or authority of the information. Some systems designed for collecting the network information, such as the Globus [8], Prophet [18] and NWS [20], can still not satisfy the need of the complex and intelligent users more or less. We will consider these methods in more detail in section 5.

This paper presents a new method to monitor the resource status at a remote network node. Different types of resources, such as computation, memory and network bandwidth are taken into account. The system resources at different levels, as well as the resources consumed by each active application at a remote network node, can be observed dynamically. Moreover, users can also specify the content of the information needed to be monitored and control the procedure of monitoring. The remainder of the paper is organized as follows: section 2 analyses the requirements to the resource monitoring approach in detail and introduces our implementation mechanism. Then in section 3, the design and implementation of our mechanism are presented, including the active node architecture and the RemoteMonitor application. Following this in section 4, the evaluation of the method is given. Section 5 discusses some related work and finally a summary is made in section 6.

2 Requirements to the Resource Monitoring Approach

In active networks, users, including the normal end-users and network administrators may have the following wishes with regard to the resource monitoring.

1. Resources can be monitored remotely and dynamically. As introduced above, a user needs to be able to monitor the resource status at a network node for the purpose of performance and management. Moreover, due to the continual arrival and departure of applications at the network node, as well as the start and end of other applications and services inside the network node system, the resource status of the node varies permanently. Hence, the users should be able to acquire the resource information at any remote network node and at any time point as they want.
2. Resources can be monitored in a user-specified way. This involves two meanings. One is the content of the resource information needed to be monitored. Different users may care about different resource information. Even for the same user, he may be interested in different resource information at different times. The other is the procedure of the monitoring. A user may also want to control the procedure of the resource monitoring at runtime. One example is, at the beginning the network administrator wants to observe the resource status at a network node every 30 seconds, after some time, when he believes that the network situation is stable, he

may want to change the interval to 2 minutes. He can also decide to suspend the monitoring for 20 minutes and then resume the monitoring.
3. Observing multiple types of resources. Because of the introduction of application-specific computation into the networks, multiple types of resources, such as computation, memory and network bandwidth, play an important role for applications at the active nodes. Therefore, a resource monitoring approach should be able to supervise multiple types of resources simultaneously.
4. Observing at fine granularity. A user should be able to observe the resource status not only at the system level, e.g., resources available in the node system, but also at application level, e.g., resources consumed by a single active application.

Considering the first two requirements, we adopt the active application mechanism to monitor the resource status at a remote node. Active networks allow applications to carry their own programs in the packets to the network nodes and maintain the infrastructure to load and execute the programs. Therefore they provide the possibility for users to specify what kind of resource information they want and how to get the information. Moreover, since the parameters to control the execution of the program can be sent to the nodes in separate packets during the execution of the programs, the procedure to obtain the resource status at the remote nodes and the way to deliver them back to the users can be controlled in a user-specified way.

The last two requirements need help from the underlying active nodes. Some basic functions and services regarding the resource status in the system should be provided by the nodes, just like other normal network services. Since normally users are not permitted to access the resource status at the network nodes directly for the safety and security reasons. Through invoking these basic functions and services, users can obtain the needed information about the resource status without threatening the system's security.

In the following we describe our implementation of the resource monitoring method, beginning with the architecture of the active nodes, which can provide the basic functions related to the resource status to the end-users.

3 Implementation

3.1 An Active Node Architecture

Basically, an active network consists of a group of network nodes. Some of them own a mechanism for loading and executing user-specific programs, these nodes are called active nodes. The general architecture for active nodes [5] consists of three components: node operating system (NodeOS), execution environment (EE) and active applications (AA). The NodeOS [2] can access all the node's communication, memory and computation resources and provides some basic services and functions for the applications from end-users. The EE acts as an interface between applications and the NodeOS. It provides an execution environment for the applications. The basic services and functions provided by the NodeOS can only be invoked by the active applications through the EE.

Fig.1 illustrates the active node architecture (ANwithRM) providing some resource query functions. It follows the general three-component active node architecture. Janos [17] and ANTS2.0 [3] have been selected as the basis of the NodeOS and EE,

respectively. The most important reasons are that Janos provides some basic resource control functions and ANTS2.0 bases on Janos and takes also the resource control functions from Janos. A resource management subsystem with three modules is integrated in the NodeOS part. Among them, the resource management (RM) module processes all the system resource information and therefore is responsible for providing all the functions regarding the resource status information.

The basic functions regarding resource query provided by the RM module includes appsResUsed(), anNodeResUsed(), and osResAvailable(), osResUsed() etc. Applications from the users can invoke these functions through the EE. Currently, only the three major types of resources, i.e., the network bandwidth, memory and computation, are considered in our system. After invoking these functions, resource status about all the three types of resources are returned. The osResUsed() returns the total resources consumed in the underlying operating systems, and the osResAvailable() returns the resources still available in the whole system. The anNodeResUsed() returns the total amount of resources used by the ANwithRM system, including the system costs and the resources consumed by all the active applications running in the ANwithRM. And the appsResUsed() returns a list of all the active applications running within the ANwithRM, together with the amount of resources consumed by each application. Application's programmers can invoke these functions to acquire the corresponding resource information and select the desired information that should be sent back to the user through programming.

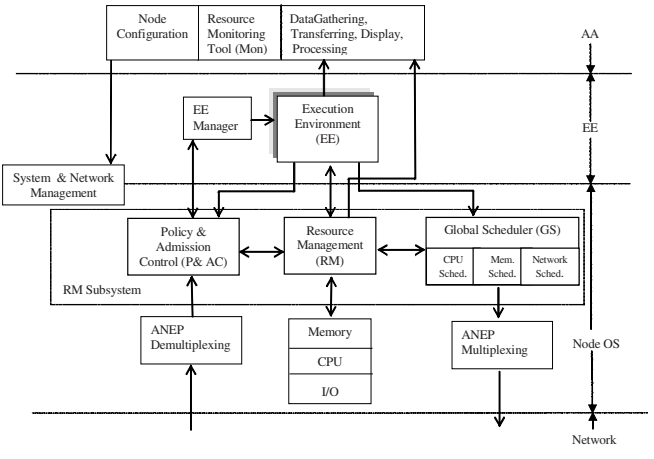

Fig. 1. The active node architecture with resource management

In our implementation, the resources consumed by an active application running within the ANwithRM consists of two parts, one is those consumed by the node system on behalf of the application, such as the Janos and ANTS threads. The other is the resources consumed by the threads initiated due to the execution of the application from the user. The information about the former can be acquired by the RM module with the help of the NodeOS and the underlying operating system. The information about the later can only be acquired through accounting, namely in order to be able to provide resource consumption information about each active application running in

the node system, the RM module has to count the various resource consumptions during the running of the applications in the node system.

The RM module uses the bytecode rewriting and a few native code methods to acquire the various resource statuses. Bytecode rewriting is used to signal the allocation and deallocation of memories, the initiation of application's threads, and when needed, some native codes are used to record the amount of the resources.

Note that in Fig.1, we illustrate the resource monitoring tool in the AA part. In practice, the program code for this application, with the functions of, e.g., resource information gathering, transferring, display and processing, can be injected by the end-users, as explained in the next section. In addition, providing basic resource query services to users is only a part of the functions of the node architecture introduced here. The node architecture can also realize some resource management functions such as admission control. More details about this architecture can be found in [12].

3.2 Remote Monitor Application

3.2.1 Principle of the Application

We have developed the RemoteMonitor application to monitor the resource usage at a remote active node. In order to transfer the user-specific programs and control parameters as well as the resource status data, following types of packets are designed with regard to this application.

MonitorStart: this is the first packet to ask for the remote resource monitoring. It carries the program code, which tells the remote node what kinds of resources are desired and how to acquire them. The program can also process all the packet related to the RemoteMonitor application. In addition, this packet carries also a dynamic precise parameter, namely a value denoting the time interval, at which the resource status information at the remote node should be inquired and sent back periodically to the user.

When this packet arrives at a network node, an execution environment (EE) is created for the RemoteMonitor application, and the program code carried is loaded into the execution environment. Then the program begins to run to gather the specified information. Fig.2 illustrates part of the program code. In this example, the user wants to obtain the resources consumed by the whole ANwithRM system and the resources consumed by the first active application running within it. Here RV stands for resource-vector, it presents all types of resources concerned in the node system. Note that we have adopted the general methods for developing active applications using ANTS [19], which is the basis of our EE.

ResourceData: this packet carries the resource status data. As the result of the execution of the program, the needed data about the resource status are acquired. The data are sent back to the user in the ResourceData packet. In order to be able to observe the resource information in "real-time", we send back a ResourceData packet immediately after the resource information is acquired in our example. In practice, users may also select to send the resource information gathered in several time points together with the according time stamp in one ResourceData packet.

MonitorSuspend/MonitorResume: since a user may not be interested in the resource status at a remote node for some time, we have designed the

```
public boolean evaluate(Node n) { //n is the node where the packet arrives
   if ( n.address()!= concernedNodeAddr() ) {
      return n.routeForNextNode(getDst());
   }     // ensure that only the information at the desired node are collected

   else if (n.address()==concernedNodeAddr ) {
      while (!STOP) {
         sendResourceDataCapsule(n, destination);
                 // destination is equal to the source address from where
                 // the MonitorStart capsule is received
         sleep(INTERVAL);
      }
   }
}

public void sendResourceDataCapsule(Node n) {
    ResourceDataCapsule cap= new ResourceDataCapsule()
    RV r1= n.anNodeResUsed();
    RV r2=n.appsResUsed().elementAt(0);
    cap.pack(r1);
    cap.pack(r2);
    n.send(cap);
}
```

Fig. 2. Program code carried in the MonitorStart packet

MonitorSuspend and MonitorResume packet to the application. After receiving the MonitorSuspend packet, the program stops gathering and transferring the status information until the MinitorResume packet is received. This action can reduce the resource overhead caused by the RemoteMonitor application itself.

Note that using the MonitorSuspend/Resume, the user may, e.g., store the acquired data in the same file for analysis after the resumption. In addition, the resume procedure is faster than to start the application again. However, since the RemoteMonitor application does not end, the resources it occupied at the remote node are not released.

FrequencyChange: after the program carried in the MonitorStart packet begins to run at the remote node, the user receives the ResourceData packet periodically from the remote node. Obviously, the smaller the interval between the two ResourceData is, the more precise can the resource status at the remote node be reflected at the user. We call this time interval the monitoring precise. For some reasons, the user may want to change the monitoring precise. E.g., the network administrator may want to lower the monitoring precise when he thinks that the resource status at a network node is stable enough. Therefore we designed the FrequencyChange packet. After receiving this packet, the program continues to gather and deliver the resource status data according to the new frequency. Note that if the program is already suspended, the program will record this new value and begin to use it when the program is resumed. The monitoring precise parameter has also been integrated in other packets such as the MonitorStart and MonitorSuspend/Resume packet.

MonitorStop: this packet stops the execution of the program at the remote node. As the result, no more resource status data will be transferred to the user.

Fig.3 illustrates the general format of these packets. We have defined a ConcernedNode option field in the ANEP header [1] to stress the addresses of the nodes, where the resource information should be monitored. The program code and data are the payload of the ANTS packet [19]. The TimeStamp field is used in the ResourceData packet to indicate the time when the status data are gathered. This is helpful to recovering the resource statuses. The program code is used only in the

MonitorStart packet to specify the content of the information to be monitored and how the information can be acquired and sent back to the user.

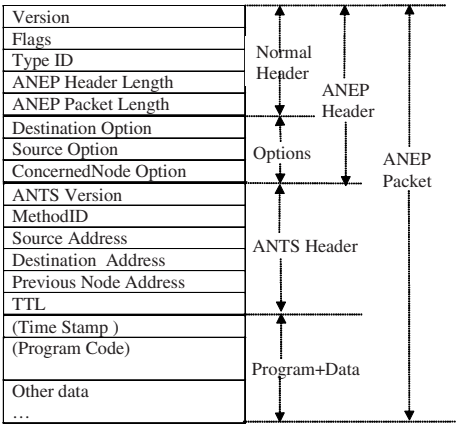

Fig. 3. General packet format

Note that here the program code and the data to be processed are carried in the same packet. We called this mechanism the associated code loading. ANTS [19] adopts an on-demand code loading mechanism to propagate code needed to process a packet in the networks. I.e., the program code (called method in ANTS) for processing various packets related to an application is sent separately from the packets. When a node receives a packet, it checks if the method needed for processing it is already available in the node according to the method identifier carried in the packet. If yes, the method is invoked and the packet is processed. If no, the packet is first saved in a waiting queue, and the node asks for the corresponding method from the previous node using a simple code loading protocol. After the needed method is received and cached, the queued packet will be processed. This mechanism limits the distribution of the code to where it is needed. However, the startup performance of this mechanism is relative poor. There exists a relative long delay until the first packet of an application is processed if the program code has not been cached in the node to be monitored. We have added an associated code loading mechanism to the ANTS EE. It allows each packet to carry the program code needed to be executed in the network nodes together with other data, and therefore omits the separate code loading procedure. This mechanism is suitable for short programs (all the code can be carried in one capsule). The startup performance comparison between the two code-loading methods can be found in [11].

Since the active application mechanism is used to monitor the resource information and the content and format of the information needed to be monitored are specified by the program code, to some extent, this method seems more complicated for users than the methods that need only to specify that users want to monitor some network information. However, due to the flexibility, the method is more applicable to the situation, where the user-specific information needed to be monitored, e.g., the information not defined by the SNMP[6] etc.

Furthermore, in our current implementation, besides invoking the resource query functions, the RemoteMonitor application acts just like other normal active applications. That means the application may be rejected if the active nodes implement an admission control mechanism and there are not enough resources for this application. In practice, according to the implementation of the active nodes, the resources consumed by the RemoteMonitor application can either be considered as the system resource cost and therefore already reserved by the system node, or the application can be given a suitable priority, such that it has a certain possibility to be accepted by the active nodes.

3.2.2 User-Interface

Besides injecting code into the remote node to instruct it to collect the needed resource information, the RemoteMonitor application has also a graphical user-interface at the local system. Through this interface, users can configure the execution of the RemoteMonitor application and the acquired resource data can be displayed graphically in time or stored in files for later analysis and statistics. The main functions of this interface include:

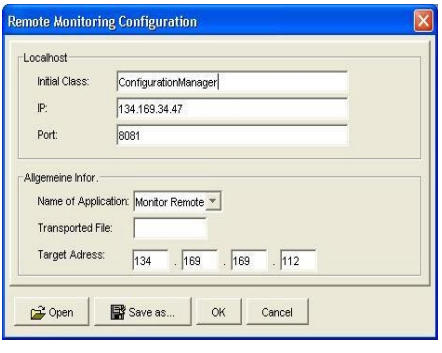

Fig. 4. User-application interface

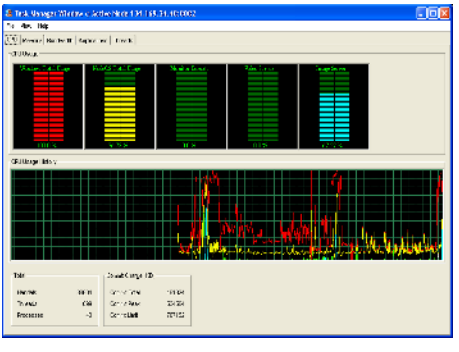

Fig. 5. Display resources dynamically

- Configure and start the RemoteMonitor application. E.g., to specify the address of the node where the resource should be monitored. Fig.4 illustrates this window. Since the RemoteMonitor is an active application, it runs atop an active node. If the active node ANwithRM is not yet running, it can be started together with the application. After the application begins, a MonitorStart packet will be sent to the remote node
- Send different control packets to the remote node, such as MonitorSuspend/ Resume, FrequencyChange, during the resource monitoring.
- Display the resource status information received from the remote node in time. After the ResourceStatus packet is received, the time stamp and the resource data are unpacked, passed to the interface, and displayed graphically in time, as shown in Fig.5. Users can also select which type of resources they want to observe.
- Store and process the received data. The data displayed in Fig.5 can also be saved as XML and Java Object file in order to be processed or analyzed later.

4 Evaluation

4.1 Test Examples

To evaluate our resource monitoring approach, we have used two machines in the same city to construct a small active network; both run our implementation of the active node architecture. One acts as the end-user and has the IP address of 134.169.34.47. The other acts as the remote active node with the address of 134.169.169.112. The end-user starts the RemoteMonitor application by sending the MonitorStart packets to the destination 134.169.34.112. In order to be able to illustrate the results clearly, we do not run any other active applications at the remote node.

In this example, the user wanted to observe the resource usage of the underlying operating system, the ANwithRM, as well as the RemoteMonitor application itself. At the beginning, i.e., in the MonitorStart packet, he specified a precise of one second, namely the resource status at the remote node should be checked and transferred back every one second. After two minutes, the user suspended the program for one minute and then resumed it. At the same time, the precise was changed to two seconds. After two minutes, the suspending and resumption procedure was repeated and the monitoring precise was set to four seconds.

To compare the resource status data observed remotely and locally, we implemented also a simple local application, which inquired directly the local resource status every 1s and display the results on time. Since the CPU usages vary relative heavily, we illustrate only the CPU usages at the remote active node (134.169.169.112) in Fig.6 and Fig.7, which are observed locally at the node 134.169.169.112 and by the user remotely at the node 134.169.34.47, respectively. From these figures we can see that when the monitoring precise is high, the resource usage at a remote node can be well reflected at the user. Otherwise the dynamic variation can be observed roughly. Note that there is still a little difference between Fig.6 and Fig.7, this is because the sampling points of the data displayed on the figures are a little different, they are not strictly synchronized.

In the next sections, we evaluate the monitoring method in the context of delay and the resource costs caused by the method itself.

4.2 Delay

We have studied two types of delay concerning the resource monitoring method. The first is the starting delay, i.e., the time interval between the MonitorStart packet is sent and the first ResourceData packet is received. The average value of tests for 5 times is 6269 ms. The value includes the transmission delay, i.e., from the user to the network node and back to the user, and the time used for loading and executing the RemoteMonitor program in the network node. The second is the observation delay, i.e., the time interval between the resource information is acquired at the remote node and when it is displayed locally, namely the time when the ResourceData packet is received and unpacked at the end-user. This value includes the transmission delay from the remote node to the user and the processing time at the remote node and at the

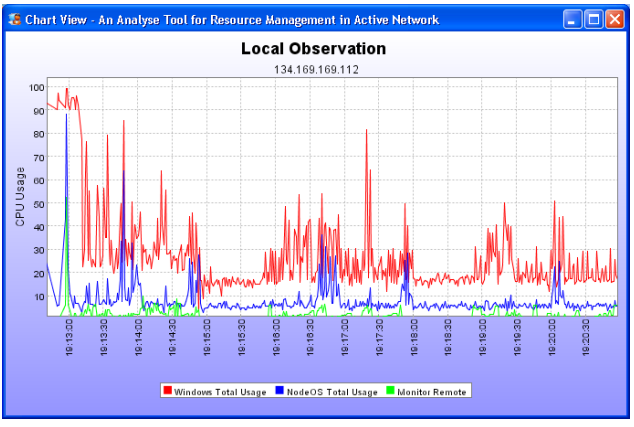

Fig. 6. CPU usages observed locally (in the node 134.169.169.112)

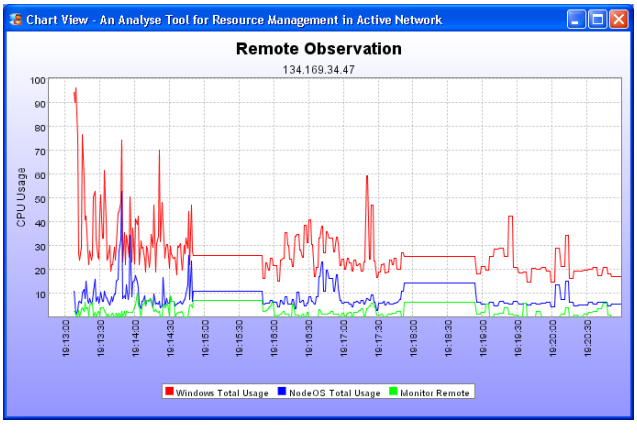

Fig. 7. CPU usages observed by the user remotely (in the node 134.169.34.47)

user, such as the resource data is packed at the remote node and unpacked at the local node. In each of our test, 210 ResourceData packets are received by the local user as the result of the 6 minutes monitoring with the frequency of 1s, 2s and 4s respectively. The average value of the delay of these 210 packets is 21ms after 5 times test. Now we are doing the tests and study these values in the wide area.

4.3 Monitor Precise vs. Resource Cost

Since we adopt the active application mechanism to realize the remote resource monitoring, the method itself has some resource cost. Generally, the cost of the three main types of resources depends on the monitoring precise, i.e., the frequency that the resource status data are gathered and transferred. The network bandwidth cost depends mainly on the length of the ResourceData packet and the number of the

packets sent per unit time, because the length and number of other types of packets are relative small and fixed. The length of each ResourceData packet varies with the content of the resources that end-users want to monitor. And the number of the packets depends on the monitoring precise. In our case, i.e., the user wants to monitor the resource usage of the underlying operating system, the ANwithRM and there is only one active application running in the ANwithRM, and the required dynamic monitoring precise is 1 second for 2 minutes, 2 seconds for 2 minutes and 4 seconds for 2 minutes, and together with 2 minutes suspend, the average network bandwidth is 765bit/s. The total computation cost consists of the time used for the load and execution of the RemoteMonitor code at the remote node for gathering and transferring the resource status data, as well as for processing the controlling packets, such as the MonitorSuspend/Resume and FrequencyChange. It is mainly related to the monitoring precise. The lowest curve in Fig.6 illustrates the computation cost of our example. The memory is mainly used for the buffers for packet receiving and sending, thread stacks, heap memory for objects during the program code execution and heap memories for "soft states", such as inside communication etc. In our example, the average memory usage by the RemoteMonitor application at the remote node is 4.3MB.

5 Related Work

Because of the importance of the network information, several methods related to the monitoring and measurement of network information have been developed in the traditional IP-based networks. [10] presents an architecture to monitor the network resource. It consists of a sensor director, several network sensors and a database. The sensor director initiates the collection of data by the network sensors in response to requests from the resource manager. To some extent, each user in our method acts as both the resource manager and the sensor director, and the program sent by the user and executing at the remote node has the same function as the network sensors. However, since our method benefits from the active networks, it is much simpler and more flexible than the approach presented in [10]. Moreover, our method permits any user in the networks to monitor the resources at any other node in the networks. Remos [7][13][14] is a system specially designed for collecting different kinds of information in networks. It uses different methods such as SNMP [6] and benchmarks to collect network information and provides a standard interface format independent of the details of any particular type of network. One important characteristic of this system is that it provides a query-based interface for its applications and allows them to specify what kind of information they need. Our approach can also provide this function: users can specify what kind of information on which nodes they want. However, the Remos system provides the collected information only in the format of a logical topology. Though it is more attractive and direct than other methods such as listing bandwidth or latency for each pair of endpoints, however, to some extent, it cannot meet the needs of all users because different applications have different requirements. In this point, our approach is more flexible: the user can specify the desired format of the network information in the program code sent to the network. NWS [20] and Prophet [18] provide applications with benchmark-based predictions by sending messages to make measurements between pairs of computation nodes.

These methods concentrated on the predictive accuracy and forecast of the performance of available resources but pay little attention to the requirements of the users. Globus [8] and Legion [9] provide a wide range of functions such as resource location and reservation, authentication and remote process creation mechanisms, but they do not focus on supporting application level access to network status information. Compared with these works, our method is more flexible. It allows the users to specify both the content and the format of the information they need. The users can also observe and process the information in their specified way and in time.

In recent years active networks have gained a rapid development [15]. Many works related to the enabling techniques, such as platforms, languages as well as reliability etc., have been done in order to implement and integrate active networks into the current networks. On the contrary, works related to the applications in the active networks are relative few. The existing applications, such as congestion control [4], network management [16], multicasting and caching, are just to use the idea of the active network technology to solve some hard problems in the network. From this point of view, our work presents an application in the active networks.

6 Summary

This paper suggests a method for monitoring the resource status at remote active nodes. The active application technique is used during the gathering and transferring the resource status data, in order to satisfy the various resource monitoring requirements from users. The RemoteMonitor application has been developed, which maintains different types of packets to notify the remote nodes what kinds of resource data are needed by the user, how to acquire and transfer them back to the user etc. Moreover, the application also allows users to control the procedure of the monitoring, such as specifying the monitoring precise, suspending and resuming the monitoring. In addition, the active node architecture that implements this method has been presented, including the consisting modules and basic functions regarding resource information query. Some initial test results about this method have been given. Now tests related to this method in the wide area are being performed.

References

[1] D.Alexander, B.Braden, C.A.Gunter, A.W.Jackson, A.D.Keromytis, G.J.Minden and D.Wetherall, “Active Network Encapsulation Protocol (ANEP)”, July 1997.
[2] AN Node OS Working Group. Node OS Interface Specification. Jan. 10, 2001.
[3] http://www.cs.washington.edu/research/networking/ants/
[4] S.Bhattacharjee, K.Calvert and E. Zegura, “Congestion Control and Caching in CANES”. In ICC’98, 1998.
[5] K.Calvert et al. “Architectural Framework for Active Networks”, available from http://www.dcs.uky.edu/~calvert/arch-docs.html.
[6] J.Case, K.McCloghrie, M.Rose and S.Waldbusser. “Structure of management information for version 2 of the simple network management protocol (SNMPv2)”. RFC1902, Jan. 1996.

[7] Tony Dewitt, Thomas Gross, Bruce Lowekamp, Nancy Miller, Peter Steenkiste and Jaspal Subhlok, “ReMos: A Resource Monitoring System for Network Aware Applications”, Tech. Rep. CMU-CS-97-194, Carnegie Mellon University, December,1997.
[8] I.Foster and C.Kesselman, “Globus: A Metacomputing Infrastructure Toolkit”, International Journal on Supercomputing Applications, vol.11, no.2, 1997.
[9] A.Grimshaw, W.Wulf and Legion Team, “The Legion vision of a worldwide virtual computer”. Communications of the ACM. Jan. 1997, Vol. 40, No. 1.
[10] Philip M. Irey IV, Robert W. Hott, and David T. Marlow, “An Architecture for Network Resource Monitoring in a Distributed Environment”. Lecture Notes in Computer Science 1388, Springer-Verlag, 1998. Pages 1153–1163.
[11] Yuhong Li and Lars Wolf, "Collection of Network Information in Active Networks", Operating Systems Review 35(4): 39–49 (2001).
[12] Y.Li and L.Wolf, “An Active Network Node System with Adaptive Resource Management”. In Proc. of ICT2002, Beijing, China.
[13] B.Lowekamp, N.Miller, D.Sutherland, T.Gross, P.Steenkiste and J.Subhlok “A Resource Query Interface for Network-Aware Applications”, proceeding of IEEE symposium on High-Performance Distributed Computing, July, 1998.
[14] Nancy Miller and Peter Steenkiste, “Collecting Network Status Information for Network-Aware Applications”, IINFOCOM 2000.
[15] Konstantinos Psounis, “Active Networks: Applications, Security, Safety, and Architectures”, IEEE Communications Surveys, First Quarter 1999.
[16] D.Raz and Y.Shavitt, “Active Networks for Efficient Distributed Network Management”, IEEE Communications Magazine, Mar. 2000.
[17] P.Tullmann, M.Hibler and J.Lepreau. “Janos: A Java-oriented OS for Active Network Nodes”. IEEE Journal on Selected Areas of Communication. Vol.19, No.3, Mar.2001.
[18] J.Weissman and X.Zhao, “Scheduling Parallel Applications in Distributed Networks”. Cluster Computing, vol.1, No.1, pp. 95–108, May, 1998.
[19] D.Wetherall, J.Guttag and D.Tennenhouse, “ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols”, IEEE OPENARCH’98, San Francisco, Apr. 1998.
[20] R.Wolski, N.Spring and C.Peterson, “Implementing a performance forecasting system for metacomputing: The network weather service”, Tech.Rep.TR-CS97-540, University of San Diego, May 1997.