Distributed Management with Mobile Components

M. Feridun, W. Kasteleijn, J. Krause
IBM Research Division
Zurich Research Laboratory
8803 Rueschlikon
Switzerland
{fer, wka, jkr}@zurich.ibm.com

Abstract

The increasing importance of networks and the growing numbers of devices and services that run on them necessitate effective network and systems management. The traditional centralized management paradigm alone is no longer sufficient for effective management solutions, primarily as it does not scale well. Distribution of management tasks is a promising approach. The distributed management framework (DMF) presented in this paper provides an environment which allows a broad range of management tasks to move and run anywhere within the managed system. In our approach, management tasks are lightweight applications that can be dynamically configured and downloaded as required, reducing the load on managed resources and simplifying the problem of management software updates. We present an object-oriented, Java-based implementation of the DMF and describe applications developed on this platform.

Keywords

distributed management, Java-based management, code mobility, mobile agents, active network management

1. Introduction

Management of networks and systems consisting of large numbers of entities requires extensive computing and networking capabilities. In a typical centralized system, management data from managed entities are collected, consolidated and analyzed at one or a few servers or ‘consoles’. There are a number of disadvantages to this approach: first, the servers become bottlenecks as they process data from the whole managed system. Second, the network can be overloaded as the management traffic flows towards the server, creating a funnel. Scalability of management solutions is a key issue.

(c) 1999 IFIP

The alternative to centralized management is the distribution of management tasks across the managed system, with the goal of executing tasks as close to the managed resources as possible. Executing management functions on or near managed resources reduces the amount of management traffic as management data is consolidated and refined at the source. The local execution of the management tasks can be more efficient, and make a richer set of local functions available to the tasks. Furthermore, in a distributed management system, dynamic configuration or deployment of management tasks in response to the requirements of the problem at hand is possible.

There are several approaches for distributed management. In [1], an earlier prototype of the DMF, a proprietary protocol is used to instantiate management operations and code is downloaded via HTTP. The architecture allows remote instantiation of management operations but no code movement. The *management by delegation* approach presented in [2] accomplishes a hierarchy-based distributed management by allowing extension of the server-side software using downloadable code fragments. The paper also describes a framework able to support extensions written in different programming languages. Reference [3] provides an in-depth study of code mobility and its impact on network management, with suggestions for applications that can benefit from the mobile agent based solutions. The work described in [4] uses a mobile agent platform to extend the capabilities of a centralized client-server architecture for network management. The architecture presented in [5] uses the 'plug and play' paradigm for distributed management, using server add-ons, delegated code as well as autonomous agents. Several centralized security mechanisms are described, such as a 'shield' to protect the local system from malicious agents, thereby restricting the capabilities of an agent to access local resources. Legal agents also have a protection shield against malicious agents.

The distributed management architecture presented in this paper provides an environment which allows a broad range of management tasks to move and run anywhere within the managed system. In our approach, management tasks are lightweight applications that can be dynamically configured and downloaded as required, reducing the load on managed resources and simplifying the problem of management software updates. Management tasks are supported by a distributed directory and secure communications. The architecture does not force the programmer to design its management applications according to any given programming paradigm. Rather, the programmer is free to use paradigms such as client-server and cooperating peers. Mixed solutions are also possible.

Figure 1 shows an enterprise network which is partitioned into several sub-networks (subnets). We assume that all relevant devices in the system support our architecture. The goal of the example management application is to determine the status of all available NFS daemons running in the enterprise network. The application, running possibly as a background troubleshooter, does not have a priori knowledge of the topology of the network or the location of the NFS servers.

(c) 1999 IFIP

On initialization, it may be given the location of an IP router, or the application may discover the router by looking at local machine settings, e.g., default route.

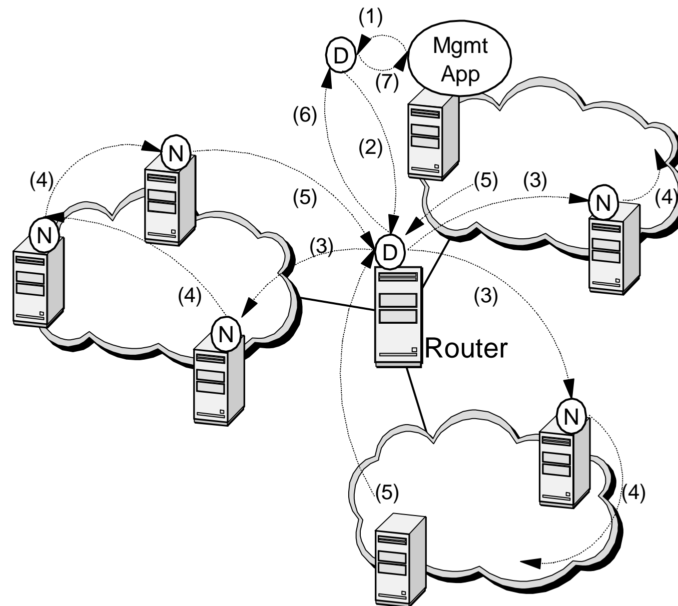

Figure 1: Distributed NFS Status Determination

The management application begins by creating and then sending a discovery agent D to the IP router as shown in Figure 1 (1, 2). From the router configuration data, agent D discovers the different subnets adjacent to the router. It then creates an NFS query agent N per subnet (3). Each such agent visits all hosts on the assigned subnet looking for NFS daemons and if one is present, collects status information (4). Once the collection is complete, each NFS query agent returns to the originating discovery agent D to deliver its consolidated data from the subnet and terminates (5). Agent D waits until all NFS agents have reported back after which it returns to the management application with the results (6, 7).

The simple example described above demonstrates a number of goals that we want to achieve with our architecture:

- The execution of the three major tasks, topology discovery, NFS daemon discovery and NFS status determination are distributed, carried out by lightweight management tasks. There is no single processing bottleneck.
- Management tasks are executed close to the managed resources. Local consolidation of raw data results in reduced network management traffic on the network.
- A management task is *brought* to the managed resource, adapted to run in the environment of the resource and then executed, rather than requiring a large set of pre-installed tools. Once the management task completes its assigned

(c) 1999 IFIP

objectives, it leaves the managed resource to continue its processing elsewhere or terminates.

- Management tasks can be assigned some level of autonomy, so that functions such as discovery can scale and adapt to the managed network without prior configuration information. For example, if there are other subnets attached to the discovered subnets, then the agent D can repeat the above steps (or clone itself) to collect NFS daemon data.

In the next section, we present the Distributed Management Framework (DMF), our architecture for roaming management applications. We then discuss our implementation of this architecture, followed by an example of a prototype management application using the DMF.

2. Distributed Management Framework

Distributed Management Framework (DMF) provides an object-oriented architecture where lightweight, mobile management applications can be dynamically deployed for distributed management. As shown in Figure 2, the DMF

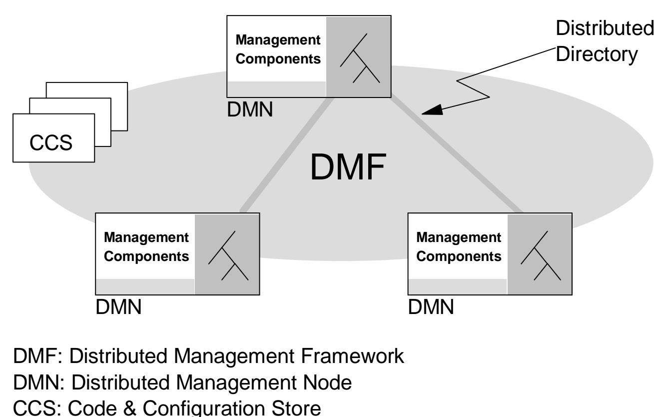

Figure 2: Distributed Management Framework

consists of a collection of *distributed management nodes* (DMNs), where each DMN provides the execution environment and supporting services for *management components* (MCs), i.e., the management applications. The DMNs form a network of peers over which hierarchies or cooperating sets of management components can be run. A *distributed directory*, maintained by each DMN is used in locating management components and services. Components and configuration of the DMF are stored in a set of replicated *code and configuration store*s (CCSs), from where they can be downloaded to the DMNs as required. In the following, we describe these components in detail.

(c) 1999 IFIP

2.1 Management Components

Management components carry out management tasks of different levels of complexity, executing within the DMF. The principal characteristics of an MC are:

- An MC is a *lightweight* task, i.e. the resources it requires (e.g., memory) are minimal, however it can be short term or a long running task.
- An MC can be *mobile*, moving between DMNs if required.
- An MC can be composed on the fly, where its parts can be downloaded as required by the management task. When its task is completed, the MC and its constituent parts can be removed from the DMN.
- An MC can run on any DMN which is configured with the *capability* for that MC. Each capability specifies key parameters for an MC such as the name of the Java class to be invoked or whether the MC is to be initiated at DMN startup. An initial set of capabilities is read from the configuration file at initialization, and can be modified (additions, deletions) during run-time.

An MC may execute simple queries, for example use the interfaces to the local file system to determine the available disk space and report the results back to another MC. An MC may act as a management console, integrating management functions provided by other MCs active within the DMF. A more sophisticated MC may carry out a longer running task such as collecting traffic statistics. Another type of MC can be an *adaptive* agent which travels to a DMN with an assigned management task, discovers the management tools available at that DMN relevant to its task, and then adapts and/or composes a solution strategy for the given environment in order to efficiently carry out its task.

2.2 Distributed Management Node

Machines on which the MCs described above should be deployed need to be instrumented with the appropriate execution environment. A Distributed Management Node (DMN) provides this execution environment for MCs.

The components of the DMN are shown in Figure 3 and are described in this section. Each of these components define functions which can be available at a DMN but need not to be present at all times as they can be dynamically downloaded and installed (or removed) when required.

Bootstrap

Bootstrap is the only component that needs to be installed on a managed resource in order to support a DMN. Its purpose is to reduce the amount of DMN code that has to be present on a given managed resource so that only the necessary components are downloaded when required, reducing resource requirements on the host and making it easier to update DMN code. On DMN initialization, the minimal bootstrap mechanism reads a local configuration file to determine the initial CCS to be used and the mode (secure, non-secure) in which the DMN will run. It then loads the code for the DMN from the CCS using HTTP or FTP if the DMN is non-secure, or HTTPS if operating in secure-mode and instantiates the

(c) 1999 IFIP

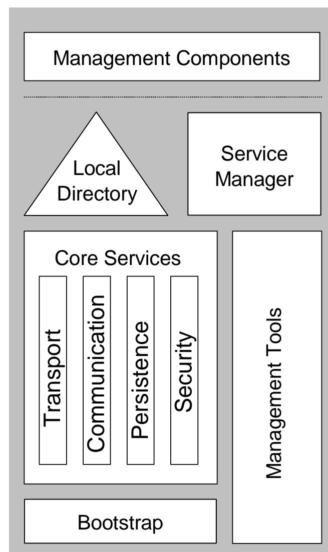

Figure 3: Architecture of the Distributed Management Node (DMN)

core services. The bootstrap mechanism next reads the configuration of the DMN from the CCS to create and initialize the *service manager*.

Core Services

The core services component provides the essential services for all components of the DMN.

The *transport service* is responsible for moving MCs between DMNs. When MCs are moved, they can keep their internal state including the results of the completed management operations.

The *communication service* provides seamless remote and local communication between the management components. Communication includes not only messaging but also event forwarding, remote method invocation and exception forwarding.

The *persistence service* allows storage of relevant DMN components into the persistent storage and reload of stored components back into memory. The primary use of this service is the recovery from system crashes, where persistent parts of the DMN can be reloaded into memory during reboot. Another possible use is to save system resources, where components that are not in use are flushed into persistent storage and reloaded again when they are needed.

The *security service* provides secure communication and transport between DMNs. It also protects different parts of the DMN by applying access control. This helps to limit the damage misbehaving or error-prone MCs can cause.

Management Tools

The component *management tools* consists of service interfaces that enable access to tools provided by the local operating system, e.g. *ping* and *traceroute*, or by

(c) 1999 IFIP

other third party software packages that are available on a machine but are not integrated with the DMN, e.g., a database. These services are registered with the local directory by the service manager, and are available to the management components through their service interfaces.

Service Manager

A very important part of the architecture is the *service manager*. It is responsible for the dynamic installation, configuration and removal of service interfaces to local management tools as well as MCs. The service manager also registers the installed service interfaces and MCs with the local directory.

The service manager is created and initialized by the bootstrap service during DMN initialization, and provided with an initial configuration. First, the service manager creates the local directory and registers it as a subtree of the global, distributed directory. Second, different service interfaces and MCs are installed as specified in the initial configuration file.

Installation of an MC by the service manager consists of a number of steps: First, the service manager downloads the necessary code packages from the central code store, which may be different from the one that was used for bootstrapping. Second, the MC is created and initialized using the capabilities defined for that MC in the local directory. Finally the service manager registers the MC as a “component” with the distributed directory, making the MC and its services (if any) available to other MCs within the DMF.

When loading the code required for an MC, the service manager uses the services of the bootstrap component to mark them as “in use” by the MC. When the MC terminates, any code that was downloaded, if not shared by other MCs, can then be removed from the DMN to save resources.

Local Directory

The local directory tree is maintained by the service manager and is a part of the distributed directory. It contains the following information:

- DMN static configuration, e.g., the location of the distributed directory root node;
- DMN capabilities, e.g., the functions such as MC types or management tools supported by this DMN; and
- DMN dynamic configuration, e.g., active MCs at this DMN and information specific to the active MCs.

In order to facilitate the integration of information related to MCs into the directory, the directory supports a special kind of node called the *ActiveNode*. An ActiveNode acts as a bridge between normal directory operations (e.g., directory query) and the facilities in an MC that provide the corresponding operations. For example, if an MC internally maintains its own directory structure, e.g., an LDAP based directory, it can add itself into the directory as an ActiveNode, which is then used as the translation interface to the MC internal directory. An ActiveNode can also be used to provide dynamic information from an MC into the directory: if a status parameter (e.g., number of active connections) of an MC is entered as an

(c) 1999 IFIP

ActiveNode in the local directory tree, then a query on that node is translated into a component internal action to retrieve the status parameter and return it as response to the query.

Distributed Directory

The local directory subtrees of all the DMNs constitute the distributed directory. The main uses of the distributed directory are location of services and monitoring the status of the components of the DMF.

A single DMN is configured to maintain the root of the distributed directory tree; backup DMNs for this role are also defined. On startup, each DMN constructs its local directory tree, and attaches it to the (first available) root.

The directory provides an event notification service. A management component can subscribe to this service in order to be notified when the subscribed event occurs. Examples of events are addition/deletion of nodes, used in monitoring status and location of management components.

The distributed directory supports generic search mechanism for locating nodes. For example, it enables queries to search for all management components of type "event filters".

Our current approach to directory failures is very simple. Each DMN monitors the distributed directory. If the DMN acting as the root of the directory fails, then the DMNs attach directory subtrees to a pre-configured backup root (there can be several). Failure of any other DMN results in the removal of the link to the directory subtree from the root of the distributed directory. We are currently developing more robust algorithms for handling failures.

3. DMF Implementation

The DMF is implemented using the Java language [6]. The key motivations for using Java are its object-orientedness, platform independence, and networking support including dynamic code downloads.

The DMF is based on the Objectspace Voyager™ platform [7], which through its ORB (object request broker) provides seamless remote communications between objects, and a number of important services such as persistence and directory. It is a Java-based middleware for the development of distributed, object-oriented applications.

Our implementation of the DMF uses and extends the capabilities of the relevant Java APIs and the Voyager platform. In this section, we briefly discuss the important aspects of these extensions.

3.1 DMN

Bootstrap

The bootstrap package consists only of two classes, namely a network class loader and the bootstrap class itself. These two classes and the Java JDK are the only installation requirements for the DMN.

(c) 1999 IFIP

When a device comes up it automatically starts the bootstrap with a URL specified in the local configuration file pointing to a CCS, e.g. an HTTP server, from where a configuration file or object is loaded. The configuration specifies the class that implements the DMN and the location of the CCS (typically the same HTTP server as the one where the configuration was downloaded from) from where the necessary packages can be loaded. After the bootstrap mechanism has instantiated the downloaded classes, it initializes the DMN with the configuration and starts it.

Using our bootstrap mechanism, an administrator can install a new version of the DMN once on the CCS and change the configuration used in the bootstrap process there. Code or configuration changes to every device hosting DMNs are not required, as the bootstrap process will automatically install new versions as needed.

Core Services

The persistence and communication services of the DMF are directly based on those provided by the Voyager platform.

The transportation service of Voyager provides object mobility. This permits movement of MCs between DMNs. MCs can be moved by other MCs or they can move themselves.

Voyager relies on the Java object serialization mechanism to transform objects into byte streams. A proprietary protocol (layered on top of TCP) is used to transport the serialized objects across the network. However custom transports can be installed as well; we use this feature in implementing security enhancements, i.e., a secure transport mechanism for moving objects and messages between DMNs.

The Voyager platform includes a customized security manager which protects (local) system resources similar to the Java Sandbox model. To protect transport of MCs and communications between DMNs we modified Voyagers transport mechanisms to support SSL (Secure Socket Layer). We also changed class loading to work with HTTPS (HTTP over SSL). Our current implementation assumes a closed system and does not handle attacks such as a malicious MC locking resources of a DMN.

3.2 Distributed Directory

The Voyager platform allows the registration of an object in a distributed hierarchical directory structure provided by Voyager’s Federated Directory Service. Subtrees of this directory structure can be spread across different hosts, i.e. DMNs in our architecture.

The objects registered in the directory can also be remote and mobile. This allows us to keep track of MCs even after they have moved to a different DMN. Our enhancements to the base directory services are the addition of search facilities, support for ActiveNode functions as described in section 2 and an event notification mechanism for alerting subscribers of changes in the directory.

(c) 1999 IFIP

4. Implementation Example: Enterprise Manager

In this section, we present a management application called EnterpriseManager that runs on our DMF implementation. EnterpriseManager is a management component that runs on a DMN (DMN-ZRH), and is used to analyze IP traffic characteristics when assigned performance thresholds are exceeded.

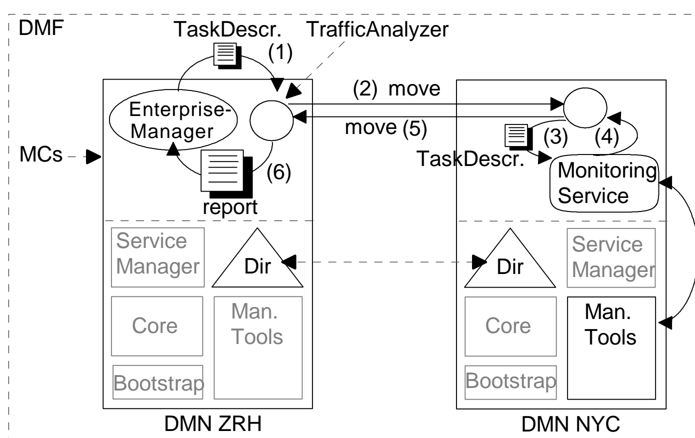

Figure 4: Enterprise Manager Scenario

In our scenario shown in Figure 4, the EnterpriseManager listens for "delay threshold exceeded at LAN NYC" events from other MCs in the DMF. When it receives such an event, the following set of actions will take place:

- The EnterpriseManager creates/activates a TrafficAnalyzer MC which monitors traffic on a LAN segment and measures IP traffic in terms of a selected set of application protocols. The TrafficAnalyzer receives from the Enterprise Manager a TaskDescription object (1). This TaskDescription 'defines' the task, which means that it specifies the subnet to monitor, types of application protocols that need to be observed (e.g., HTTP, SMTP, FTP) and the monitoring duration.
- The TrafficAnalyzer MC now uses the Distributed Directory to find a DMN that is capable of running a MonitoringService MC and can monitor traffic on LAN NYC. It finds DMN NYC.
- The TrafficAnalyzer moves itself to DMN NYC (2).
- On arrival at DMN NYC, the TrafficAnalyzer starts the MonitoringService MC and initializes it with the TaskDescription (3). The MonitoringService initializes and starts monitoring using a set of management tools available at the DMN. The monitoring process is explained in detail in the next sub-section.

(c) 1999 IFIP

- As it receives the results from the subnet, the MonitoringService reports data back to the TrafficAnalyzer which composes a report based on the collected data (4).
- When the specified monitoring period ends, the MonitoringService stops monitoring and terminates, i.e. removes itself (and the associated Java classes) from the DMN.
- The TrafficAnalyzer moves back to the DMN ZRH (5), gives its report to the EnterpriseManager and terminates (6).

4.1 Monitoring Tools

In order to monitor the traffic on a LAN segment, we need monitoring tools that can capture IP traffic from the network and analyze them. In our implementation of the scenario, we have used two such tools, and these are briefly described in this section.

RMON

An RMON (Remote Monitoring) probe is a packet monitor/analyzer with an SNMP agent supporting Internet standard MIBs. There are two versions: RMON MIB [8] defines statistics for the lower protocol layers as well as configurable, sophisticated filter mechanisms to count or capture specific packets; RMON2 MIB [9] provides extensions to RMON by allowing application layer protocol statistics.

The only way to configure and get data from a typical, commercially available RMON probe is through the SNMP protocol. In order to reduce the SNMP traffic between the management components and the RMON probes, in our implementation, RMON probes are always local to (meaning on the same host-machine) or at least on the same subnet as the management components.

Homegrown Packet Sniffer

The homegrown packet sniffer (HPS) consists simply of a C-based tool that captures packets from a promiscuous network adapter, and passes it onto a Java object that acts as a real-time ‘packet server’ for management components. HPS captures IP packets.

Analysis of packets is done using ‘plug-in’ management components. The key advantage of the HPC over RMON probes is that it does not require SNMP. However, its functions are limited to packet capture, and does not include statistics.

4.2 The MonitoringService MC

The MonitoringService MC is responsible for selecting, configuring and collecting data from a monitoring tool. Its key component is the TaskHandler which uses the TaskDescription received from the TrafficAnalyzer to configure the monitoring tool. As an example, assume that the task description specifies HTTP protocol as the one to be observed (TCP port 80), and the monitoring tool is an RMON probe. The TaskHandler creates a set of three Java objects, 2 filter objects representing RMON *filter* and one for a RMON *channel* as shown in Figure 5. The filters are

(c) 1999 IFIP

configured to capture TCP packets with source or destination ports equal to 80 respectively.

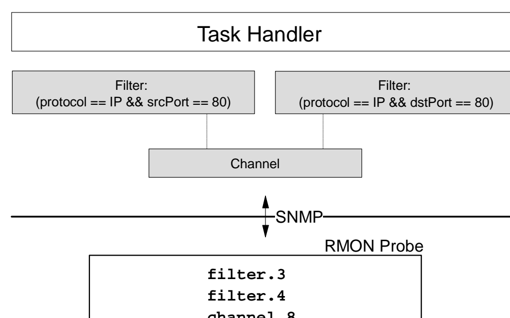

Figure 5: Setting-up and RMON Probe

On creation, these Java objects send a set of SNMP *set* commands to the RMON probe to configure it by creating the equivalent RMON MIB table entries. To collect data, the TaskHandler calls a method on the channel (getData) which using an SNMP *get*, collects the value of the *channelMatches* field (number of HTTP packets) from the RMON channel construct.

Software constructs such as the TaskDescription and the TaskHandler enable us to dynamically create and configure management applications at any DMN. The TaskDescription is a concise description of an management task; the TaskHandler realizes this task by pulling together the appropriate management components.

The Java objects created for the monitoring task, including the TaskHandler can be dynamically downloaded when required, and removed (garbage collected) when the monitoring is complete.

4.3 Implementation Experiences

We implemented the EnterpriseManager application on our site network consisting of 9 LAN segments distributed across two remote campuses. We installed DMNs at a selected set of hosts (running AIX, Windows95, Windows NT and Linux) on the network. The installation has been used to analyze traffic patterns for services such as the HTTP and Lotus Notes servers.

For the scenario described above, a DMN located on the same LAN segment as the RMON capable of monitoring the campus HTTP traffic was selected. Figure 6 shows a browser window with a graph showing the HTTP traffic distribution over a day on three local LAN segments at our site. This graph is generated in real-time by the EnterpriseManager based on the reports received from the TrafficAnalyzer. A set of management components are used as ‘plug and manage’ modules for the TrafficAnalyzer in order to carry out different analysis functions.

(c) 1999 IFIP

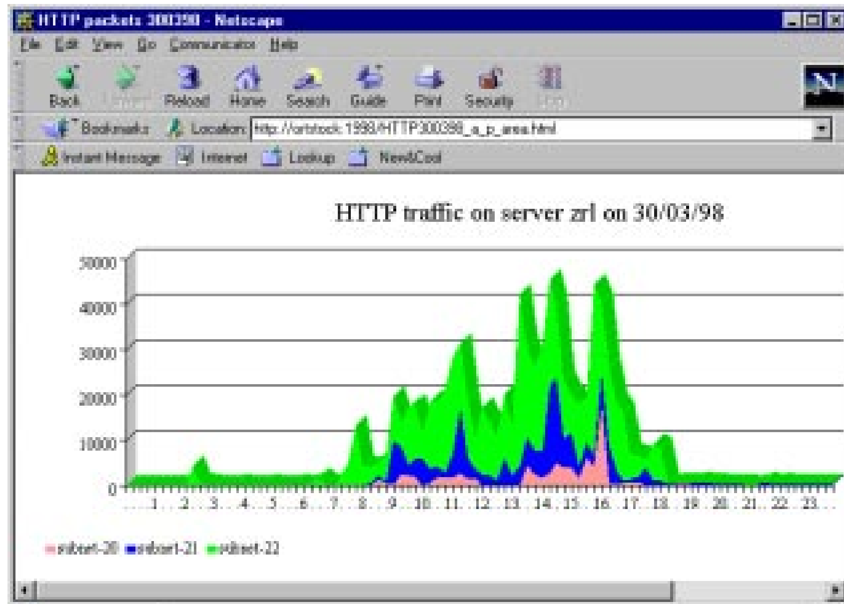

Figure 6: HTTP Traffic Distribution

Our experiences with the EnterpriseManager implementation have been very positive. Using the DMF, we can very easily move different management components and therefore functions to a DMN to carry out a given task. For applications where location dependent management tools such as RMON are not needed, we are able to download all of the DMN including the bootstrap component using a browser and Java applets.

5. Conclusion

The phenomenal growth in the size of networks and our increasing reliance on them for services requires effective means of network and systems management. The architecture presented in this paper is a step in this direction. The distributed management framework (DMF) enables distribution of management tasks and provides the ability to run these tasks practically anywhere in the managed system. The DMF architecture is designed to run on various types of devices that range from embedded systems to desktops.

Our current research effort is focused on two main directions. First, we are developing a suite of management applications that benefit from the distributed management approach. Second, we are looking at implementation options for an "embedded" version of the DMF that can scale down to very small devices such as handheld computers or printers. The overall goal is to investigate the distributed management of environments consisting of huge numbers of (possibly mobile)

(c) 1999 IFIP

devices, working with Tivoli Systems, an IBM company, on the future generation of Tivoli management products.

Acknowledgments

The authors acknowledge contributions of our colleagues (in alphabetical order), A. Bussani, T. Gschwind, C. Hörtnagl, W. Kleinöder, R. Nielsen, S. Pleisch and S. Rooney as well as the suggestions made by anonymous referees.

References

[1] F. Barillaud, L. Deri, and M. Feridun, "Network Management using Internet Technologies," Proc. ISINM'97, May 1997, San Diego, USA.
[2] G. Goldszmidt and Y. Yemini, "Distributed Management by Delegation," Proc. ICDCS'95, June 1995, Vancouver, British Columbia, Canada.
[3] M. Baldi, S. Gai and G. Picco, "Exploiting Code Mobility in Decentralized and Flexible Network Management," Proc. MA'97, April 1997, Berlin, Germany.
[4] A. Sahai and Christine Morin, "Towards Distributed and Dynamic Network Management," Proc. NOMS'98, February 1998, New Orleans, Lousiana, USA.
[5] A. Bieszczad and B. Pagurek, "Towards Plug-and-Play Networks with Mobile Code," Proc. ICCC'97, November 1997, Cannes, France.
[6] K. Arnold and J. Gosling, *The Java Programming Language*, Addison Wesley, 1996.
[7] ObjectSpace, *ObjectSpace Voyager Core Technology User Guide* (V2.0), 1998.
[8] S. Waldbusser, *Remote Network Monitoring Management Information Base*, Internet RFC-1757, 1995.
[9] S. Waldbusser, *Remote Network Monitoring Management Information Base Version 2 using SMIv2*, Internet RFC-2021, 1997.

(c) 1999 IFIP