Open Programmable Architecture for Java-enabled Network Devices

Tal Lavian, Nortel Networks - Advanced Technology Center
Robert F. Jaeger, University of Maryland
Jeffrey K. Hollingsworth, University of Maryland

Keywords: Open Architecture, Programmable Networks, Java-enabled Networking, JVM, Active Networks, Mobile Agents, Intelligent Agents.

1 Overview

Current network devices enable connectivity between end systems given a set of protocol software bundled with vendor hardware. It is impossible for customers to add software functionality running locally on top of network devices to augment vendor software. Our vision is to open network devices so that customized software can be downloaded, allowing for more flexibility and with a focus on industry and customer specific solutions. This brings considerable value to the customer.

We have chosen to use Java because we can reuse its security mechanism and dynamically download software. We can isolate the Java VM and downloaded Java programs from the core router functionality.

We implemented Java Virtual Machines (JVMs) on a family of network devices, implemented an Open Services framework, and developed an SNMP MIB API and a Network API upon which we can demonstate the value of openness and programmability of network devices.

2 Open Network Services

2.1 New Programmable Paradigm

We are entering a new era where servers, switches, and routers are participating in applications. The IEEE P1520 recognized the need for a software infrastructure for programming networks and has initiatives which propose standard APIs to networking devices. [28-29].

Programmable network technology allows dynamic downloading of software programs to network devices and moving intelligence to these devices. Downloading software enables customized application level computational interconnectivity between servers and network devices as well as among network devices. Programmable networks is an enabling technology for a new type of distributed applications that are not technically possible today.

2.2 PC-ifications of Programmable Devices

In network devices like switches and routers, the software and the hardware are bundled and not openly programmable. We have created an Open Service Architecture that allows dynamic downloading of programs to theses devices.

Our vision is that network devices will be open like software for PCs. For example, we buy software and hardware for PCs from different vendors, and then we install and customize the software on the computer. Network devices like switches and routers are also computers of a different type. We have implemented programmability characteristics on these types of computers like the PCs so that additional functionality can be provided to these devices.

2.3 Enabling Technology

Now, we are able to securely download and run Java applications on network devices. We can distribute the intelligence to network devices for close-loop interaction on the nodes. This allows for new types of applications that were not possible until now because of scalability issues.

We have implemented technology that allows for a class of downloadable applications which are not bundled with the vendor hardware including enhanced multicast services, application level filtering, and mobile/intelligent agents [4] [13] [17]. The platform supports the Active Networking reference architecture upon which multiple non-interfering Execution Environments can run concurrently [2-7] [9] [12] [14-18] [25]. Operating systems for computers have recently been developed to allow user customization of services and policies. Exokernel[1]] and Spin[8] provide new kernel architectures to support safety and extensibility.

2.4 Static Agent vs. Dynamic Agents

Usually, we are managing network devices via “get” and “set”, as *static* requests to SNMP agents. Having a *dynamic* agent on the device opens us to a new set of applications. Instead of external SNMP requests, we can download software that use SNMP calls via an internal device loop.

The direction is to open network devices to 3rd party developers. To facilitate this, we have created an Open Service Interface[26] using Java with additional security mechanisms. The interface opens the network devices to 3rd party vendors and allow other domain experts to add value to network devices.

Java has taken the computer world by storm. However, it is mainly used by client s and servers for browser and business applications. Until now, Java did not run on the network devices.

2.5 New type of Applications

We can add value to network infrastructure by allowing tight code interaction between business applications and network devices. This new paradigm allows a new type of distributed applications between servers and swiches/routers. All software in the applications can be dynamically loaded on to the servers and network devices.

2.6 Domain Experts

Domain experts that work closely with customers know the problems the customer faces and may bring ideas from other business domains to the development of solutions. Allowing 3rd party experts to write custom applications allows for innovative solutions to customer business requirements.

2.7 Feature-on-demand

Java beans with specific plug-in features can be developed and released quickly providing feature-on-demand capability to address changing customer needs. This approach circumvents the need for vendor support to add customer desired capabilities to network devices and avoids the need for the customer to wait for new features from the vendor in order to realize capabilities. Also, features can be ported quickly to other products implementing the Open Service Interface.

3 Open Architecture

3.1 What have we implemented?

We have implemented JVMs on the Nortel Networks Routing and L3 Routing Switch family of products. We can securely download and safely run Java applications on these devices and access native code running on these devices [1]. We developed an architecture that allows for downloading of new code to implement desired features. We developed several APIs including an SNMP API and a Network API. The SNMP API allows reading and writing of MIB variables [27]. The Network API allows Java applications to control the hardware based forwarding and selection of packets to be delivered to Java applications based on packet signatures. Consequently, our network devices are programmable allowing for customers to safely run dynamically downloaded Java code to realize additional functionality.

3.2 Scalability

Network Management via SNMP suffers scalability in applications with massive close-loop interactions. For example, assume an NMS application that monitors some network device parameters. Lets assume that we want to gather information on 10 parameters per port on a device with 100 ports and we want to sample 100 times per second. These types of applications are not practical in the current SNMP model of centralized management.

Current technology like RMON provides an interface to allow network devices to collect statistics locally. It is restricted to a set of predefined types of data that can be collected and it does not address the scalability problem.

We overcome these limitations in our distributed intelligence approach. The data samples are local calls with no need for SNMP call for each sample. The NMS will be notified only when needed based on thresholds or trends. This allows an intelligent event notification mechanism rather than a polling mechanism.

3.3 Security

Computer networks have become vital to businesses. It is critical that the network functions properly and a loss of operation is not tolerable. To allow the openness that we are proposing, we needed to implement a strong security architecture. Security is a key component of the Open Service Interface architecture. It includes security for downloadable applications and for the device's code interface. In addition to the Java language security features, the architecture provides additional security including digital signatures and code certification.

3.4 The JVM

Implementing a JVM for embedded and real-time environments presents several challenges. Such environments are especially concerned with reliability, determinism, and restricted memory. It is also crucially important that new Java programs must work without disturbing the existing non-Java functionality.

Our JVM design satisfies these requirements by encapsulating the JVM into one framework. This JVM manages the Java threads and their memory usage within this framework and presents one unified task to the RTOS. The Open Services Architecture was designed to insulate core router functionality from Java and JVM failures. Even if the JVM crashes, the router remains operational.

3.5 But Java is SLOW

Java has often been criticized for its speed and its use in a routing device has been questioned. With the advent of ASIC technology for fast path data forwarding engines that achieve Gigabit forwarding rates, no serious routing platform is based on a software forwarding engine. An active network platform that touches all network packets is doomed to fail. However, control and management applications are well suited for Java based network services locally on the network device as well as active applications which don't require high packet per second rates to operate.

3.6 Java MIB API

The goal is to provide an interface that transparently gives access to the SNMP data, while hiding the SNMP machinery of PDUs, object identifiers and table iterations. The MIB gives a fairly object-oriented model to the data, which fits quite naturally onto Java class decomposition.

The MIB is structured as a collection of groups, with each group containing a number of variables. Each group is modeled as a Java class, and each variable is modeled with accessory methods.

3.7 Inter-operating with SNMP MIB Compiler

There are a large number of variables in the MIB and it is tedious and error prone to generate the Java bindings for them all by hand. To automate the binding, we provide a MIB compiler that takes the standard ASN.1 textual representation of the MIB and generates the necessary Java code. This approach is similar to other mibgen tools and results in a Java API which represents the MIB tree. Javadoc pages document the Java MIB API that results from the input MIB definition.

An incremental strategy for MIB variable access is the result of this research. MIB variables are queried and set by constructing SNMP PDUs that are sent to the loopback interface of the device and delivered to the underlying SNMP stack. This approach allows us to evolve from the loopback strategy by incrementally adding direct low-level implementations of selected time-critical methods.

4 Applications Examples

4.1 Close-loop Applications

Creating a closed-loop mechanism on the device itself allows for the development of tight local management applications. We can develop applications utilizing tight interactions with the traffic pattern and the device capabilities. We can move some of the intelligence of NMS and policy servers to the network devices. This allows for new types of applications that are not feasible with centralized management schemes.

4.2 Example 1: Interactions with Business Applications

Currently, business applications are running on servers and networking applications are running separately on network devices. There is very limited interconnection at the application level. Downloading and running Java applications on network devices enables collaborations between network devices and applications running on other servers. Business applications can take advantage of distributed computing in an environment consisting of applications running on network devices which leverage the local-loop directly on the device. For instance, an electronic auctioning service can use routers to filter bids which are no longer acceptable given that the bid price has already been surpassed [10]. Another application can *dynamically* configure the output priority queues on the network device as part of the application needs.

4.3 Example 2: Interactions with other network devices

Currently, network devices interact using a pre-defined set of protocols bundled with the hardware. There are no interactions or sharing variables that is not within the confines of bundled vendor software. In the Open Service Interface architecture, network devices can run non-bundled distributed applications that interact at the application layer with applications running on other devices. For example, a community of routers can make a *dynamic* configuration decision based on bottlenecks or load. The decision making process can be done as part of a distributed application within this community.

4.4 Example 3: Mobile Agents

The architecture provides a platform for mobile agent technology. Mobile agents to perform information gathering such as L2 connectivity or traceroute are made possible through the Network API. The mobile agent performs some work on the local device and moves to the next device to continue the work. For example, roaming diagnostic agents can gather performance information along the path from a client to the server. This mobile agent can identify and even solve problems along the network path.

4.5 Example 4: Active Networks

By developing an Open Services Interface, we provide a platform upon which Active Networking Execution Environments (EEs) can be hosted. We allow for running Java-based EEs as services on the router that can host active applications downloaded in the Active Network Encapsulation Protocol for distribution to EEs.

4.6 Example 4: Local QoS implementation

Instead of configuring the QoS mechanisms from remote station, we can do it locally on the network device. For example, we can monitor locally the traffic patterns and configure the QoS parameters based on policy that is pushed to the device as an application.

5 Conclusion

We are entering a new era where servers, switches, and routers participate in networking applications. This is a powerful new technology that enables dynamic downloading of end-user applications. We provide an open set of network services upon which new applications can be built to customize the behavior of network devices as well as management networks. We provide tools to developers enabling them to use their domain expertise to create solutions to specific business challenges. Introducing domain expertise into the development process creates a virtual community of developers that can bring innovation to networking customers.

The Nortel Open Service Interface functional prototype implements this enabling-technology which is the foundation of this paradigm shift. The ability to download *dynamic* agents to network devices is much more powerful than the current use of *static* agents. Dynamic program loading into the switching and routing elements of a network bring new opportunities for configuring and managing large, complex, and high performance networks. Pushing intelligence into the network elements themselves solves the scalability problems inherent in a centrally managed network.

6 References

[1] P.Bernadat, D. Lambright, and F. Travostino, "Towards a Resource-safe Java for Service-Guarantees in Uncooperative Environments," IEEE Symposium on Programming Languages for Real-time Industrial Applications (PLRTIA) '98, Madrid, Spain, Dec. '98.

[2] AN Node OS Working Group, NodeOS Interface Specification", June 15, 1999

[3] Active Networks Working Group, "Architectural Framework for Active Networks Version 0.9", August 31, 1999

[4] D. Wetherall et al. ANTS: A Toolkit for Building andDynamically Deploying Network Protocols. OPENARACH'98

[5] S. Bhattacharjee et al. On Active Networking and Congestion. Technical Report GIT-CC-96/02, College of Computing, Georgia Institute of Technology, 1996.

[6] D. L. Tennenhouse and D. Wetherall. Towards an active network architecture. In Multimedia Computing and Networking 96, San Jose, CA, Jan 1996.

[7] Y. Yemini and S. da Silva. Towards Programmable Networks. In IFIP/IEEE Intl. Workshop on Distributed Systems: Operations and Management, 1996.

[8] B. Bershad et al. Extensibility, Safety and Performance in the SPIN Operating System. In 15th Symp. on Operating Systems Principles, Dec. 1995.

[9] D.D.Clark & D.L.Tennenhouse. Architectural Considerations for a New Generation of Protocols. SIGCOMM '90,1990.

[10] eBay Inc. AuctionWeb server. http://www.ebay.com/.

[11] D. R. Engler et al. Exokernel: An Operating System Architecture for Application-Level Resource Management. In 15th Symp. on Operating Systems Principles,1995.

[12] M. E. Fiuczynski and B. N. Bershad. An extensible protocol architecture for application-specfic networking. Proceedings of the 1996 Winter USENIX Conference, 1996.

[13] S. Floyd et al. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. ACM SIGCOMM'95, 1995.

[14] L.-W. Lehman et al. Active Reliable Multicast., INFOCOM'98, 1998.

[15] A. Mallet et al. Operating System Support for Protocol Boosters. Technical Report MS-CIS-96-13, CIS Dept., Univ. of Pennsylvania, 1996.

[16] J. Smith et al. SwitchWare Accelerating Network Evolution. Technical Report MS-CIS-96-38, CIS Dept.,Univ. of Pennsylvannia, May 1996.

[17] D. Tennenhouse et al. A Survey of Active Network Research. IEEE Communications Magazine, 1997.

[18] D. L. Tennenhouse and D. Wetherall. Towards an active network architecture. In Multimedia Computing and Networking 96, San Jose, CA, Jan 1996.

[25] Jonathan Smith, et al, Activating Networks: A Progress Report, COMPUER, April 1999.

[26] T. Lavian, S. Lau, "Java-Based Open Service Interface," Bay Architecture Lab TR98-010. March 30, 1998, Bay Networks. BAL web site: http://forums.baynetworks.com/lobby/main/dispatch.cgi/_T_corp_archlab_docshare

[27] R. Duncan, T Lavian, R. Lee, J Zhou. "Java Open SNMP MIB API." Bay Architecture Lab TR98-038. December 1998.

[28] IEEE P1520 Reference Model, www.ieee-pin.org

[29] Biswas, J., Lazar, A. A., Huard J.-F., Lim, K., Mahjoub S., Pau L.-F., Suzuki, M., Torstensson S., Wang W. and Weinstein S. "The IEEE P1520 Standards Initiative for Programmable Network Interfaces", IEEE Communications Magazine, Vol 36.