Dynamic Classification in Silicon-Based Forwarding Engine Environments

R. Jaeger[1,2], R. Duncan[2], F. Travostino[2], T. Lavian[2], J. Hollingsworth[1]
[1]*University of Maryland, College Park, MD 20742*
[2]*Nortel Networks, 4401 Great America Parkway, Santa Clara, CA 9505*
{rfj,hollings}@cs.umd.edu {rduncan, travos, tlavian}@nortelnetworks.com

***Abstract*--Current network devices enable connectivity between end systems with support for routing with a defined set of protocol software bundled with the hardware. These devices do not support user customization or the introduction of new software applications. Programmable network devices allow for the dynamic downloading of customized programs into network devices allowing for the introduction of new protocols and network services. The Oplet Runtime Environment (ORE) is a programmable network architecture built on a Gigabit Ethernet L3 Routing Switch to support downloadable services. Complementing the ORE, we introduce the JFWD API, a uniform, platform-independent portal through which application programmers control the forwarding engines of heterogeneous network nodes (e.g., switches and routers). Using the JFWD API, an ORE service has been implemented to classify and dynamically adjust packet handling on silicon-based network devices.**

***Index terms*-- Programmable Networks, Active Networks, ORE, JFWD, Distributed Applications, Networking Protocols, Differentiated Services**

I. INTRODUCTION

Programmable networks enable the introduction of user computing to network devices to provide new and improved services and distributed application functionality. Traditional networks deliver packets between endpoints using protocol software that is bundled with the hardware. Network devices do not support user customization or introduction of new software applications nor do they facilitate introducing new protocols quickly.

This paper details a policy-based packet classification technique that is downloaded to a programmable network device to dynamically adjust the DS-byte in the IP header of real-time flows on a silicon-based forwarding engine. The technique is implemented using the Oplet Runtime Environment (ORE), a network services platform which supports dynamically loaded Java services. Crucial to the classification engine is the JFWD network API which provides control of the forwarding plane operations. We demonstrate that we can support Java-based services and implement packet sampling, processing, and forward policy adjustment on a commercial-grade, silicon-based, Gigabit L3 Routing Switch.

The remainder of this paper is organized as follows. Section 2 introduces the Oplet Runtime Environment (ORE). Section 3 introduces the JFWD networking API. Section 4 discusses implementation of a technique for doing best-effort packet classification for Quality of Service improvements based on packet sampling and policy-based DS-byte (formerly ToS) modifications. The conclusion is presented in Section 5.

II. THE OPLET RUNTIME ENVIRONMENT (ORE)

The Oplet Runtime Environment (ORE) described in this paper supports dynamically injecting customized software services into network devices. The implemented architecture is composed of an embedded Java Virtual Machine (JVM) and the ORE. The ORE component provides the substrate on the network device to support the secure downloading, installation, and safe execution of services. Since the ORE and its services are constrained to running in the JVM, system stability of the core network device operations is not affected.

Customized ORE services, which run locally on network devices, include monitoring, routing, diagnostic, or other user specified functions. ORE services can monitor and change specific Management Information Base (MIB) variables locally on the device through the Java MIB API [3]. Direct access to MIB variables on network nodes greatly improves scalability and reduces network traffic compared with using SNMP agents to perform remote "get" and "set" operations [3].

The ORE architecture attempts to mediate the shared resources (memory, CPU, persistent storage, and bandwidth) between mutually suspicious services and to provide for inter-service communication including the sharing of objects and calls into other services.

The ORE architecture consists of the ORE environment, oplets, and services. Oplets are self-contained downloadable units that encapsulate one or more services, service attributes, authentication information, and resource requirements. Oplets can provide services to other oplets and can depend on other services. The ORE provides machinery to download oplets, manages the oplet lifecycle, maintains a registry of active services, and tracks dependencies between oplets and services.

III. JAVA FORWARDING API

We have implemented a network API on a commercial-grade, ASIC-based, L3 Routing Switch that allows ORE services to request alternative packet handling from the hardware forwarding plane. The JFWD API specifies interfaces for Java applications to control a generic, platform-neutral forwarding plane. The JFWD API consists of a collection of Java classes specifications. The implementations of these classes is not considered part of the JFWD API technology, because it is code that, unlike the API, is highly platform-specific and thus lend itselfs to other distribution strategies.

JFWD is the first Java API package to explicitly address the needs of contemporary forwarding planes of network nodes which are increasingly implemented in silicon, must support multiple protocols, and must realize Quality-of-Service (QoS) low-level functionality. To drive an ever-growing number of options, more diversified control data will have to be fed into control planes.

The JFWD implementation across multiple hardware platforms brings a) the abstraction of a platform-neutral forwarding engine, b) a framework within which new protocols and control data can be described, and c) a new breed of shrink-wrapped Java programs for controlling network nodes. [2].

IV. JFWD APPLICATION

ORE services can use the JFWD API to instruct the forwarding engine to divert packets from the data plane to the control plane. To support a new breed of router-based programs that do not delay packet forwarding, we use the JFWD API to instruct the forwarding engine to send packets to both the data plane output queue and also to the control plane. This feature, called CarbonCopy, does not divert packets to the control plane; it performs wire-speed forwarding while copying the packets to the control planes.

By allowing ORE services to modify the packet handling in the ASIC forwarding engine, new types of applications are possible that do not require core router functionality to be altered.

In [5], the authors present a control-on-demand architecture in which flow control is performed on a *best effort* basis. An IPv4 flow is determined by five fields in the IPv4 header: the IP source and destination address, the source and destination port numbers, and the protocol. Packets belonging to a flow are copied to the control plane for processing while simultaneously forwarding the packet. This is our approach as well. It deviates from the store-execute-forward paradigm of Active Networks.

We implemented an ORE service, called the DiffServ Classification service (DSC), to classify flows and modify packet headers based on the policy set for a particular protocol contained in the IP header. A policy determines the action to be taken for a particular flow or class of flows. A policy filter executes the actions specified in the policy.

Our implementation samples packets from the data plane using the CarbonCopy feature of the JFWD API. To accomplish this sampling, the DSC service sets the CarbonCopy filter on all ports, which instructs the forwarding engine to copy some number of packets from each port into the control plane.

Next, the DSC service retrieves the delivered packets. Each packet is parsed to determine if a policy filter has been set for this flow. If no filter is set for the flow but a policy exists for this protocol, a policy filter will be created for this flow and will be applied to ports in the forwarding processor(s).

The policy filter created for a particular flow is based on the IP header fields in the packet that determine a flow. The value of the DS-byte field in the IPv4 header is set for all packets in the flow. The new value of the DS-byte value is determined by the policy for that protocol type. In our prototype, our policy is to set the DS-byte field for all packets where the protocol is RTP.

For each RTP packet sampled, the service checks for an existing filter. If no filter exists for this flow, the RTP policy of setting the DS-byte is executed by setting a filter in the forwarding engine. This modifies the forwarding engine behavior to set the DS-byte field for all packets of that flow. It is a *best effort* classification, since the sampling of packets is random and may not acquire the packets from a particular flow in a timely manner.

V. CONCLUSION

We implemented a stable programmable networks platform capable of running dynamically loaded services on a commercial-grade Gigabit L3 Routing Switch. The ORE supports the creation of services in Java that are extensible, portable, and easily distributed over the network. It provides a platform for safely running these services without disturbing the existing core functionality of the router. Through the JFWD API, the packet capture, processing, and modification capabilities are realized at the programmable node. Because of the CarbonCopy and filter mapping features provided in the JFWD API, we were able to create an ORE service to perform dynamic DiffServ Classification and DS-byte adjustment in an effort to improve the performance of specific types of network flows.

VI. 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] R. Jaeger, R. Duncan, F. Travostino, T. Lavian, J. Hollingsworth, "An Active Networks Services Architecture for Routers with Silicon-Based Forwarding Engines", Submitted to OpenArch '00 March 2000.

[3] T. Lavian, R. Jaeger, J. Hollingsworth, "Open Programmable Architecture for Java-enable Network Devices", Stanford Hot Interconnects, August 1999.

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

[5] G. Hja'lmtysson, S. Bhattacharjee, Control on Demand: An efficient approach to router programmability, April 1999.

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

Dynamic Classification in a Silicon-Based Forwarding Engine

Rob Jaeger[1,2] rojaeger@nortelnetworks.com rfj@cs.umd.edu

Tal Lavian[2] tlavian@nortelnetworks.com

[1] Technology Center, Enterprise Solutions, Nortel Networks
[2] Department of Computer Science, University of Maryland

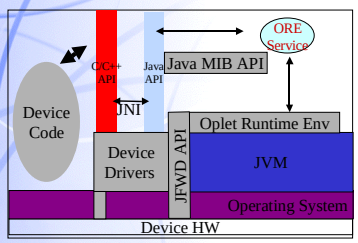
NORTEL NETWORKS

Objectives

- Implementation of Dynamic Classification in Silicon-Based Forwarding
- Perform policy based classification and modification of packets
- Specify interfaces for Java applications to control a generic, platform-neutral forwarding plane
- Enable dynamic downloading of services to network devices
- Provide for inter-service communication including the sharing of objects and calls into other services

NORTEL NETWORKS

Open Device Architecture



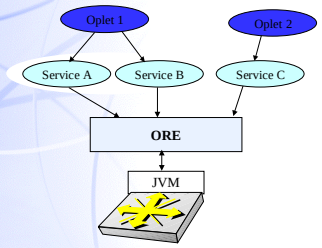
NORTEL NETWORKS

Accomplishments

- JVM on a silicon-based Routing Switch
- ORE - Oplet Run-time Environment
- Java-enabled Device Architecture
- Java SNMP MIB API
  - include proxy mode for devices with no JVM
- Implementation of Network Forwarding API
- All of this enables implementation of **Dynamic** Classification in Silicon-Based Forwarding

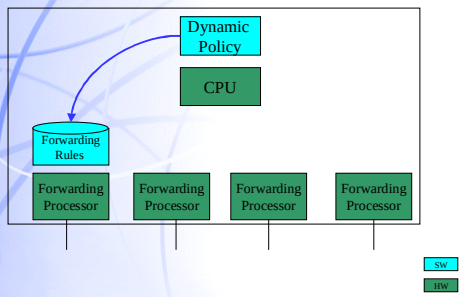
NORTEL NETWORKS

ORE - Oplet Run-time Environment



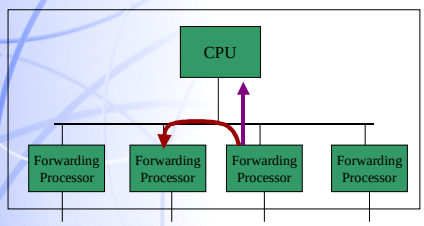
NORTEL NETWORKS

Dynamic Configuration of Forwarding Rules



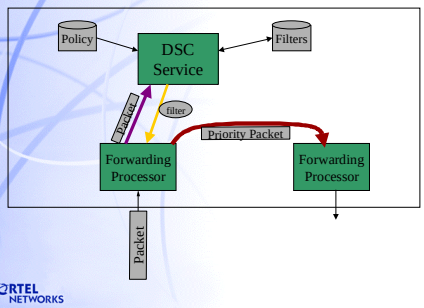
NORTEL NETWORKS

JFWD CarbonCopy Capability



NORTEL NETWORKS

Best-Effort Classification



NORTEL NETWORKS