Active Networking On A Programmable Networking Platform

Tal Lavian, Phil Yonghui Wang
{tlavian, pywang}@nortelnetworks.com
Technology Centre, Nortel Networks Corporation

Abstract **– Current Active Networks research projects are mainly realized in software-based host systems since commercial network devices lack required networking programmability. This paper studies the active networking approach using the Openet programmable networking platform. Openet comprises ORE (Oplet Runtime Environment) and hierarchical services from low-level systems to high-level applications, and provides a neutral service-based programmability to network devices. Moreover, Openet can have customer network services including Active Networks-based services deployed on current commercial network platforms.**

We demonstrate the active networking with commercial network devices by deploying the active network service ANTS onto the Accelar routing switches. The performance of active network communication is examined by the experiment in an Accelar-routed active net and compared with regular non-active network communication. The experimental result reveals that Java network I/O is a bottleneck of enhancing capsule processing capability and ends up a look at what active network services are applicable to current commercial network platforms. Finally we present observations and future works about active networking through the Openet platform.

I. INTRODUCTION

Programmable networking technologies such as *Active Networks* [2] expose a novel approach that allows customers to introduce value-added services into the network "on-the-fly". Typically, through the Active Networks, applications can deploy new protocols and change their services dynamically for specific purposes in terms of active packets. Thus, the exciting opportunity is that the network infrastructure can be changed by network service providers and other third parties, rather than only network device providers.

The present-day trend in commercial-grade routers and switches is to implement ever more functionality of network in hardware, resulting in ever-faster performance, but ever-less flexibility, since only fixed sets of services and protocols are supported. As more of the functionality is frozen in silicon, less is the capability to introduce new service and customization inside the network. This limitation makes these network devices unsuitable for hosting Active Networks services, resulting in that their current implementations are primarily done in host-based systems.

In order to enable programming services, network devices must be, in addition to fast performance, equipped with the networking programmability. The Nortel Networks Technology Center has proposed out a programmable networking platform, *Openet* [1], which is a service-based internetworking infrastructure that delivers such programmability to diversified network devices.

This paper studies the deployment of Active Networks services using the Openet platform onto commercial network hardware. Openet provides the networking programmability by introducing ORE and a stack of hierarchical network services. The ORE is an open, platform-neutral, pure Java runtime environment that is used to customize, download and initiate network services dynamically. In terms of *Oplets*, all network services are encapsulated as ORE-based services. These services are classified into four categories from low-level system services such as *JFWD* (Java Forwarding) and *JMIB* (Java MIB access) to high-level application services such as active network EEs (Execution Environments) and their applications. Finally, services are injected into the network by having ORE download and activate their Oplets on network nodes.

The Nortel Networks *Accelar* routing switches [4] are used with Openet in our investigation. They are commercial multi-gigabit products that provide in hardware L3 routing, switching, filtering and classification. To gain the wire-speed forwarding performance, the Accelar in the data plane employs the ASIC (Application Specific Integrated Circuit) hardware technology that is not re-programmable yet. However, the Accelar control plane is a CPU-based system that can run Java and external program code. This property allows Openet to be integrated so that the Accelar becomes a re-programmable device that allows deploying network services in the control plane.

To demonstrate the active networking capability on the Openet platform, the ORE ANTS service, which implements the MIT ANTS EE [6], is deployed on the commercial Accelar routing switches. Within the Nortel Networks corporate intranet, an experimental active network is constructed with active nodes, non-active nodes and a downloading server. We successfully run ANTS applications to enable the active network communication over the network and to examine the system performances of active and regular network communications through experiment. The result shows that Java network I/O operations transmitting a capsule take much more time than processing a capsule once faster CPU is employed in network nodes. This becomes the number one cause impacting the network performance.

The remainder of this paper is organized as follows. Section 2 briefs the DARPA Active Networks technology and related works. Section 3 introduces the Openet programmable platform, including ORE, hierarchical services and Accelar. Section 4 argues how a service is injected to the Accelar, and details ORE APIs as well as the ANTS service deployment on the Accelar. Section 5 presents experimental results and related discussions. Finally, observations and future works are concluded about Openet and active networking.

II. THE DARPA ACTIVE NETWORKS AND RELATED WORKS

The DARPA Active Networks approach [2] is a major effort to supply the user networking ability under the Internet infrastructure. Through installing multiple active user interfaces or *Execution Environments* (EEs) on *active nodes*, users can flexibly compose new protocols and dynamically deploy new services for their specific purposes. These EEs are referred to virtual machines and “programming interfaces” that are available for the Active Networks applications to process *active packets* or *capsules* and to control the processing.

Significant research works include: the MIT *ANTS* (Active Node Transfer System [6]) toolkit, the UPenn *Switchware* architecture [5], the Columbia University *Netscript* language [7], the USC/ISI *Abone* (Active Backbone) [9], the Active Networks protocol *ANEP* [8] and the BBN *Smart Packet* network management [10]. To date, these developments have been mainly realized in software-based hosts (e.g., Linux systems) that offer the required programmability but lack the performance required in real networks. Nonetheless, the foremost goal of Active Networks is to bring these active networking technologies to commercial network nodes (routers and switches), in which they also gain performance from hardware acceleration.

III. OPENET

Openet is originated from the open programmable architecture for Java-enabled network devices [1]. The Openet architecture depicted in Figure 1 includes two major components of the Openet: *ORE* and *Hierarchical Services*. In this section introduces the two components as well as how Openet works with the commercial hardware *Accelar*.

A. ORE

The Oplet Runtime Environment (ORE) is the core of the Openet architecture. It is an open object-oriented networking environment for customer service creation and deployment. At runtime, it supports injecting customized software, e.g., the Active Networks EEs, into network devices through secure downloading, installation, and safe execution of Java-based service code inside a JVM (Java Virtual Machine).

In order to secure service downloading and management, we define the *Oplet* as a self-contained downloadable unit that embodies a non-empty set of services. Thus, services are encapsulated by one or multiple Oplets, and Oplets in turn publish those services they provide to ORE. Along with the service code, an Oplet also specifies service attributes, authentication information, and resource requirements. Like a Java object, a service can inherit particular functions from other services and offers its interfaces public to them.

The ORE provides the mechanisms to download Oplets, to resolve service dependencies, to manage the Oplet lifecycle, and to maintain a registry of active services. Users can deploy network services to the network by having ORE to download and activate their Oplets on particular network nodes.

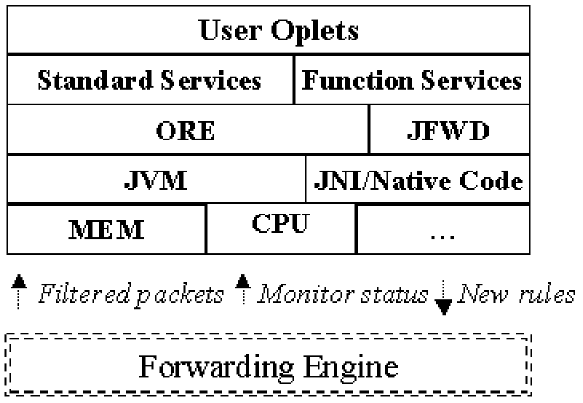

Figure 1: The Openet architecture

B. Service Hierarchy

In Openet, all network services are encapsulated by Oplets and run within the ORE environment. Oplets are objects and provide public APIs accessible to application services.

To ease service creation and gain platform independency, Openet employs a service hierarchy that places these services into four categories: *System*, *Standard*, *Function* and *User*, as shown in Figure 1.

1) System services and JFWD

“System services” are low-level network services that have direct access to the hardware features through JNI or native codes. For re-programmable hardware, they are built over native programming interfaces. Otherwise, for current ASCI-based commercial hardware that is not re-programmable, they wrap the hardware instrumentation that controls the ASIC behaviors. Thus, in fact, they by their neutral APIs determine how much of the programmability Openet brings to hardware. They require particular hardware knowledge, and provide neutral APIs to upper-level services.

- JFWD: *routing and forwarding service, alters hardware packet processing behaviors*
- JMIB*:* *MIB access service, provides access to hardware instrumentation*

- JSNMP*: SNMP client, provides access to the SNMP v2 agent*
- JPCAP*: local packet capturing service using Berkeley* libpcap (if available)

Of the above system services, JFWD is a fundamental one that provides platform-independent Java APIs that customer services use to alter the routing and forwarding behaviors on network nodes. It includes a number of standard service mappings such as MAC address, ARP, IP routing, IP filtering, IP Diffserv and VLAN (Virtual LAN). JFWD implementations on different network platforms (e.g., Accelar/VxWorks and Linux) require use of native codes or communications. On the Accelar routing switches, the JFWD implementations turn out to be a wrapper around the hardware instrumentation interfaces.

A typical use of JFWD is to instruct the forwarding engine to alter packet processing through the installation of IP filters. A filter is composed of MAC address, IP or transport protocol header, or their combination, and a policy that specifies the action executed to the matched packets. The policy can define where the matched packets are delivered or how the packet content (e.g., Diffserv remarking) is altered. Diverting packets to the CPU (at the control plane) allows customer services such as AN EEs to capture packets that match particular filters (e.g., the protocol type is ANEP) from the forwarding plane and thus to process them.

2) Standard services

"Standard Services" provide the ORE standard features for customer service creation and deployment. They are also used to conduct user interaction with ORE.

- *OpletService: Oplet service API, extended to create service descriptions and interfaces*
- ManifestOplet*: Oplet encapsulation abstract interface, implemented to create service-specific Oplet*
- Startup*: ORE startup service, auto-starts specified services when the ORE starts*
- Shell*: telnet-like user interface service, provides shell commands to manipulate Oplets and/or network services (e.g., start and stop)*
- Logger*: ORE log service, provides printout during running services*

3) Function services

"Function Services" provide common functionality or utility used to rapidly create user-level services. They are intermediate services coming with the ORE release or contributed by a third party.

- *HTTP: HTTP service*
- *JDiffServ: Diffserv interface, provides access to the hardware Diffserv feature*
- Jcapture*: Packet capturing service, sets IP filters and diverts packets to CPU*
- IpPacket*: IP packet utility, constructs IP/TCP/UDP header and payload*

4) User services

Namely, "User Services" are the user-end application services for particular purposes. They are built using the other three lower categories, dependent on whether they require use of existing services and hardware features. Typical application services include altering packet forwarding priority, QoS setup, mobile agents, network monitoring, Active Networks EEs, network intrusion detection, protocol composition and personalized communications.

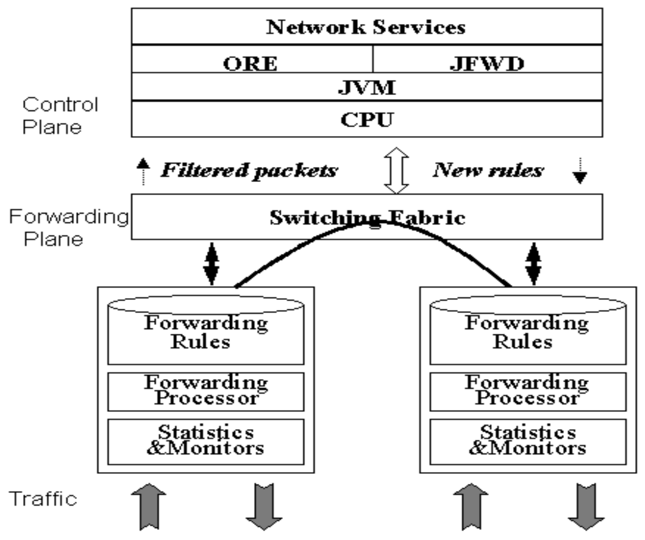

Figure 2: Accelar and Openet

C. Accelar Routing Switch

The Accelar, or Passport[1], achieves a significantly higher level of performance by introducing two separated working planes *control* and *forwarding*, as depicted in Figure 2. The forwarding plane along the data path is implemented using ASICs that can forward packets up to 256 gbps (gigabits per seconds) without consuming any CPU resource. Conventional routers and software-based routing systems involve CPUs in both packet forwarding and forwarding control, hence reduce the level of performance that they can achieve.

The control plane utilizes the whole CPU resource, and resides the VxWorks OS and the embedded Java VM. Moreover, it runs ORE, and houses diversified network applications that make up of customers' intelligences and

[1] Passport/Accelar 1100B, 8600 and other models are available in the commercial market without Openet. Openet is free and source open for the research purpose.

value-added services. As a result, the Accelar brings the Openet programmability to real networks.

How does Openet work with a network node like the Accelar? Basically, ORE and network services are initiated at the control plane. In fact, these services can be divided to two planes: *control* and *data*, according to which plane they serve. Control-plane services deal with network management issues such as changing network configurations (e.g., routes), or altering the forwarding behaviors (e.g., forwarding priority) along the data path. Data-plane services such as new protocol processor cut through the data path and take in and process particular packets before forwarding them.

IV. SERVICE DEPLOYMENT

With Openet, network services are composed using normal Java classes. Before they get deployed to the network, they are encapsulated with the ORE Oplet interfaces to have the Openet programmability. This section describes issues about the service deployment.

It is obviously of interest that commercial network devices like the Accelar embody the Active Networks approach to real networks by hosting these EEs. The MIT ANTS is a typical EE for composing and deploying new network protocols dynamically. It employs mobile code, demand loading, and caching techniques, and provides a software package that comes with a toolkit and several demonstrative applications such as ping and multicast.

To deploy this service, we've built an ORE ANTS service named "AntsNodeService", which provides the same ANTS EE capability as the original MIT ANTS distribution does. Through wrapping the MIT ANTS code, this ORE ANTS implementation is completely injected onto the Accelar 1100B and 8600 routing switches.

A. Two Oplet APIs

The ORE provides two Oplet APIs for service creation and encapsulation.

1) Base service

The first API "OpletService" is a base class of service creation that a new service extends to define its interface. That interface class includes the service description and the service function interfaces.

A service also has another class (called as the object class) to implement its interface class. That object (e.g., AntsNodeServiceImpl.java) realizes the customer functionality that provides a service (e.g., an Active Networks EE) as well as two private methods that the service Oplet internally uses to start and stop the functionality. It also can import Java codes of other services or user programs.

2) Service Encapsulation

The second "ManifestOplet" is an abstract interface of service encapsulation that an Oplet (e.g., AntsNodeOplet) implements to encapsulate the service and register it to the ORE. ManifestOplet has two methods: startService() and stopService(), which are used by the ORE to start or stop a service.

While loading the Oplet, the ORE extracts the service information from a manifest file like "Ants.mf", including the Oplet name, service package and description and dependent services. This file is consistent with the service Oplet.

B. Service Package

The Java code of a service is packed to a jar file for ORE downloading, which may include the below Java classes and other user-defined classes.

- *AntsNodeService.java: the AntsNodeService public interface*
- *AntsNodeServiceImpl.java: the AntsNodeService implementation, wraps "package ants"*
- *AntsNodeOplet.java: the Oplet, provides the "AntsNodeService" service*
- *Ants.mf: the service manifest, provide the service information*

The jar file (e.g., ore-ants.jar) can be stored in the local ORE directory "<*OREROOT*>/ore/jars/", or uploaded to a trusted server for later downloading.

C. Service Injection

The final stage of service deployment is to inject network services to network nodes, which implies downloading and activating their code within the ORE. There are at least three ways to do dynamic service injection, e.g., the ORE shell service, the ORE startup service and a user service initiation service. Once ORE downloads and then activates the service Oplets, they are injected and thus become local services (e.g., on the Accelar).

The "AntsNodeService" is actually the ORE wrapper around the ANTS code and generates an instance of the MIT ANTS EE (version 1.2). The ORE ANTS service is packed with both the MIT ANTS package and the AntsNodeService one. Then, it's stored in a Linux HTTP server (see Figure 3).

The Openet software including the ORE 0.3.3 and the whole ORE ANTS are available via "http://www.openetlab.org/downloads/". More statements about this service deployment are detailed in [11].

D. Our Active Net

Within the Nortel Networks corporate intranet, we construct an experimental active net that mainly includes 3 active nodes and 3 non-active ones, shown in Figure 3. The Accelar 1100B or 8600 routing switch, and 3 PC boxes are located in an experiment network (*net 10*), which is routed to the intranet where working machines such as Sun workstations are. The ORE ANTS service is loaded on the Accelar node and tested with the ANTS applications such as the ANTS Ping (APing).

- *Accelar 1100B: PowerPC 403/66Mhz with 32 MB memory and VxWorks 5.3, as the active router running the ORE ANTS*
- *Accelar 8600: PowerPC 740/266Mhz with 64 MB memory and VxWorks 5.3, as the active router running the ORE ANTS*
- *2 Sun workstations: UltraSPARC I/167Mhz with 128 MB memory and Solaris 2.7, as Source and Destination hosts running MIT ANTS*
- *HTTP server: PII/400MHz system with 32 MB memory and Linux 2.2.14, providing the ORE service jar files and the ORE ANTS configuration*
- *2 PCs: PII/400MHz systems with 32 MB memory and Linux 2.2.14, as source and destination hosts running regular Ping.*

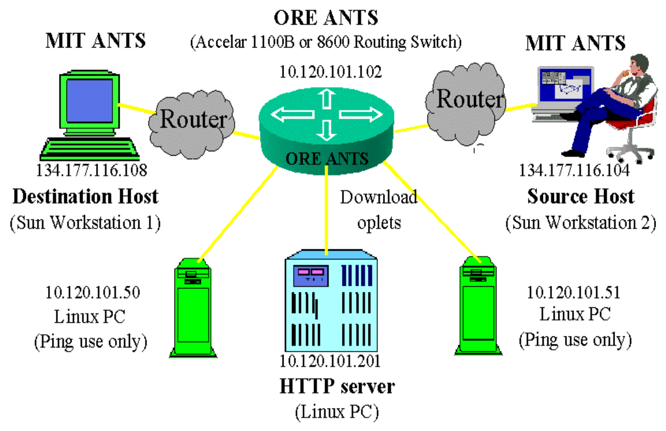

Figure 3: The experimental active net running ANTS EEs

To reflect the above network, we modify some ANTS configuration files such as “ants.config” and “ping.routes” to use the Accelar as the active router and two Sun workstations as the Source and Destination hosts. All the configuration files are also stored in the same HTTP server.

The ORE ANTS service runs as follows. After booting (with the ORE boot image), the Accelar downloads and then starts the ORE core, the ORE ANTS and other services from the HTTP server. When the ORE ANTS service is loaded, it further downloads these configurations and uses them to set up the ANTS EE. On the two hosts, the original MIT package is used to start the ANTS EEs with their own configurations.

To use this service, the ANTS Ping application is started at the Source host and sends 100 capsules to the Destination at a given interval. Initial processing the capsules by the ORE ANTS EE at the Accelar indicates that it encounters a new active service, i.e., the Aping service. Then it sends a request to the capsule’s source for loading the Aping service code. Once the code is transferred to the Accelar, the ORE ANTS EE executes it to process the first and then subsequent capsules so that they are forwarded to next ANTS-enabled node (i.e., the Destination host). When the Destination echoes the APing capsules, the Accelar (now the Aping code is installed) can readily process each feedback capsule and forward to the intended Source host. Those capsules transmitted and received at each node show that all the ANTS EEs are working properly, including the ORE ANTS services on the Accelar.

V. EXPERIMENT AND RESULT

System performance is a very concerned issue to the active networking approach. In this section, we study the performance in terms of delay and throughput and compare active networking communication with regular IP communication, through our experiment based on the active net shown in Figure 3. The Accelar node, which is either 1100B or 8600, routes both active and regular IP packet traffic during the experiment.

A. Experiment

The experiment has two fundamental goals regarding the active networking services with the commercial network hardware platform. The first goal is to verify that through Openet the ANTS EE is deployed on the Accelar and works with other ANTS EEs on the Source and Destination hosts. Section IV has already described how to achieve this goal by loading and activating the ANTS EEs and applications on the Accelar and Sun workstations.

The second goal is to evaluate the service performance and to determine the impact of active communication and capsule processing on the system performance, as compared to regular non-active network communication. To achieve this, both ANTS Ping (APing) and regular Linux Ping (Ping) applications are tested (because other ANTS applications are not easily comparable). In the active net, the Aping test uses two Sun workstations as Source and Destination hosts, the Accelar as the active router and a Linux PC as the HTTP server. The Ping test uses two Linux PCs as source and destination, which are also connected by the same Accelar.

Two Accelar routing switches 1100B and 8600 are used respectively in the active net, with all the tests repeated. Accelar 1100B has a slow CPU while Accelar 8600 has a fast CPU. Through testing with these two Accelars, we can understand how those CPUs perform capsule processing differently.

The two applications are tested by sending at least 100 packets or capsules at some regular intervals. The Aping capsule size is 83 bytes while the Ping packet size is 64 bytes. Initially, when Aping is started the first time, it sends 100 capsules at 1000ms interval. Then, Aping is repeatedly tested and measured under 4 different intervals from 0 to 1000 ms. Ping is tested only twice by sending 100 (or more) packets at two extreme intervals, one interval is 0 (using "*ping –f*") and the other is 1000ms (using "*ping"*). Both APing and Ping are tested under normal traffic rather than overloaded. It seems senseless having background traffic congest the network in this experiment since the Accelar has a throughput as much as 10 gbps.

B. Results and Analysis

Network communications in the experiment are packet- or capsules-based. At each node, all packets or capsules transmitted and received are counted, and their departure and arrival time are measured. Following the experimental result is our analysis.

1) Loss

Having all the capsules delivered is very important to active networks applications, even though the network may not guarantee it. In Aping, when the Destination host receives a capsule, it returns a feedback capsule to the Source host. The numbers of feedback capsules received at the Source host are drawn in Figure 4 and indicate how many capsules are securely conveyed in the active net.

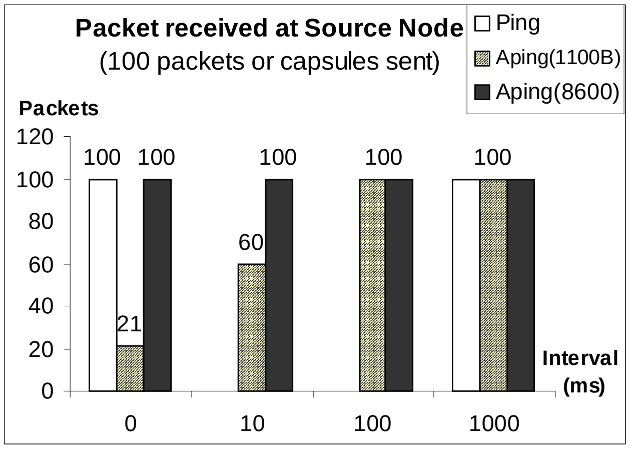

Figure 4: Received packets for Ping and Aping

In this experiment, of all the Aping tests with four intervals, only two tests that use Accelar 1100B and work at other intervals (0 and 10ms, less than the capsule round-trip time) lose their capsules. Other tests that either work at intervals 100ms and 1000ms or use Accelar 8600 communicate all the 100 capsules and their feedback ones.

The worst case that the Source host only receives 21 feedback capsules happens when Aping sends capsules to Accelar 1100B uninterruptedly (i.e., its interval is 0ms). The reason is that the ANTS communication is based on the UDP channel and thus does not guarantee packet delivery. The UDP communication is actually blocked when the next-hop nodes (i.e., the Accelar and the Destination host) are busy in receiving and processing incoming capsules.

In comparison, all the APing tests using Accelar 8600 at all intervals do not lose capsule. This Accelar has sufficient CPU competence to process incoming capsules in time, as a result, no UDP communication is blocked even at 0ms interval.

On the other hand, the two Ping tests with 0 and 1000ms intervals receive all the feedback packets, without loss. This is because that both Accelar and Sun workstation can process the well-defined ICMP packets immediately without blocking the arrival of next packet.

2) Delay and Throughput

Table 1 lists the packet delay and throughput values of the APing and Ping tests. All APing tests are measured after the Accelar and the Destination have loaded the APing service code, except the APing ones "*1000 (startup)*" at the bottom lines in Table 1. These "startup" ones are the first tests in which the Accelar and the Destination need to load the APing service code from the Source host before they can process the first incoming capsule.

Table 1: Packet Delays of Ping and Aping Tests (In milliseconds)

Interval	**Ping**		
	First packet	**Average**	**Throughput (pps)**
0	1.2	0.1	10000
10	-	-	-
100	-	-	-
1000	0.8	0.1	10000
Interval	**APing (1100B)**		
	First capsule	**Average**	**Throughput (cps)**
0	3209	-	-
10	551	-	-
100	139	32	31.5
1000	131	31	32.3
1000 (startup)	760	53	19.6
Interval	**APing (8600)**		
	First capsule	Average	Throughput (cps)
0	47	391	2.55
10	12	11	90.9
100	12	11	90.9
1000	13	11	90.9
1000 (startup)	462	36	27.7

Only the two APing tests that lose capsules cannot calculate their average delays and throughputs. Particularly, for the Aping test at 0ms interval, it is 3209ms past when the Source host receives the first capsule feedback. Why? The experiment shows that the Source host sends out all 100 capsules (consuming 3208 ms totally) before turning to receive capsules. For Accelar 8600, the APing test at 0ms interval has a larger average delay of 391ms, which indicates that capsules are ever buffered heavily before processing.

In other tests that do not lose their capsules, their average delays and throughputs are pretty close, 31ms for Accelar 1100B and 11ms for Accelar 8600.

It's also noticed that the first tests "*1000 (startup)*" at 1000ms interval have bigger delays (i.e., 760 ms for Accelar 1100B and 462ms for Accelar 8600) when the Source receives the first feedback capsule, but further tests "*1000*" at the same interval have much shorter delays, 31ms and 11ms respectively. The reason is that the later tests save time loading the Aping service code on each node along the capsule route.

The maximal throughput of the APing tests is found to be 32.3 cps (capsule per second) for Accelar 1100B and 90.9 cps for Accelar 8600. However the two Ping tests have the same throughput of 10,000 pps (packet per second) and the same average delay of 0.1ms as well. This comparison reveals that packet-by-packet processing in the ANTS service significantly reduces the network throughput.

3) Delay Contributions of APing tests

To look into what contributes the capsule delay, capsule-processing time consumed at each node is measured by comparing the time of receiving and re-transmitting one capsule. Figure 5 depicts the delay contributions among different components in the Aping tests that have the minimal average capsule delays: 31ms for Accelar 1100B and 11ms for Accelar 8600.

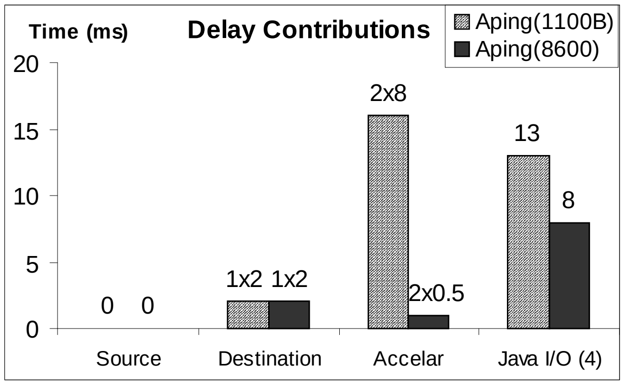

Figure 5: Delay distribution among components

The Source host does not process a capsule (excluding transmission) and takes 0ms. The Destination host takes 2ms to process a capsule before returning a feedback capsule. The Accelar processes a capsule sent from the Source and a feedback capsule returned from the Destination, consuming totally 16ms for Accelar 1100B and 8ms for Accelar 8600. That is, the Accelar 1100B takes averagely 8ms processing each capsule, or 3 times slower than the Destination (i.e., a Sun workstation). However Accelar 8600 takes 0.5ms, 3 times faster than the Destination.

The remaining time (13ms for Accelar 1100B and 8ms for Accelar 8600) is consumed by the round-trip communication of a capsule. In fact, it takes little time (less than 1 ms) transferring a capsule among three nodes by the wire communication. So, that time is supposedly used by 4 pairs of Java network I/O operations (write and read) on the three nodes (1 on the Source, 1 on the Destination and 2 on the Accelar). That is out of our expectation! However, additional network tests based on a Linux PC and a Sun workstation confirms that the Java overhead for a pair of simple UDP socket I/O operations (i.e., a DatagramSocket server writes a 32-byte message and a DatagramSocket client reads it) needs 2~3 ms while the same socket operations using C/C++ takes almost 0 ms.

That is, on Accelar 8600, a capsule and its feedback need 2x0.5ms processing and 2x2ms I/O operation. The total time of a capsule delay related to Accelar 8600 is averagely 5ms, or at a throughput of 200 cps.

Compared with Accelar 1100B, in an APing test using Accelar 8600, the whole Java I/O of a capsule communication takes 8ms reduced from 16ms, however nearly 75% of the total 11ms capsule round-trip time arising from 42% of 31ms. This gives a lesson that JVM implementation, particularly in Java network I/O operations, is the bottleneck of active networking performance once the capsule processing ability is enhanced.

C. Discussion

The above experiment may not be complicated but it is adequate to reach the two experimental goals. It has verified the deployment of the active networks service ANTS through Openet and also examined the performance of the Accelar-based active networking. Here we discuss common issues about the active networking services with the Openet platform and the commercial network platform, including the performance evaluation of this experiment, potential performance improvement through hardware and software approaches, and finally the classification of active networking services that are applicable to the current commercial network hardware platforms.

1) Performance evaluation

Generally speaking, in this experiment, the overall performance of the ANTS service running in current Accelar-based active network is similar to that in conventional host-only active networks, and largely depending on CPU competence and Java runtime execution. The maximal throughput of APing is 32.3 cps using Accelar 1100B or 90.9 cps using Accelar 8600, which is not comparable to the throughput of regular Ping, 10,000 pps. This is not a surprising result. The main reason is that like a host system the Accelar has to use its limited CPU to process every capsule prior to forwarding since its ASIC-accelerated forwarding engines cannot process packets whose protocols are not integrated. Actually those CPUs used in the host systems here are Sun UltraSPARC 1 @167MHz and Intel Pentium II @400MHz, neither of them is the strongest CPU today. Of the Accelar routing switch family, Accelar 1100B is an economic product that is equipped with a PowerPC

403@67MHz CPU and Accelar 8600 is a superior one that is equipped with a PowerPC 740@266Mhz CPU. That's why Figure 5 shows that the Accelar switches process an Aping capsule distinctively, i.e., Accelar 1100B takes four-fold time of a Sun host does While Accelar 8600 takes one quarter of a Sun host does.

In contrast, the regular traffic of Ping maintains the same throughput that is different from the active network communication, as shown in Table 1. This result is little changed in the experiment even if both Ping and APing are tested at the same interval simultaneously, because the Accelar separates control and data planes. The Accelar forwarding engines that deal with the regular traffic are not impacted under the condition that the Accelar CPU is busy in processing the active network traffic.

2) How can we enhance the performance?

Typically, the ASIC technology enables commercial network devices like the Accelar switches with high capability that can forward regular traffic with well-defined protocols at a designated rate of 1 gbps per port, or nearly 2 million 40-byte IP packets per second. In contrast, the active packets that exercise their own protocol are processed by the Accelar CPU system at a rate around 200 cps. Such a low throughput cannot meet the performance requirement of commercial data communications.

How can we enhance the performance? Obviously, integrating stronger CPUs such as latest PowerPC chips into commercial network devices like the Accelar is reasonable and has no problem in implementation. Accelar 8600 coming with faster CPUs such as PowerPC 740@266Mhz already shows a very good performance improvement (see Figure 5). But it is little possible to enhance shortly the active networking performance as high as the Accelar designated rate due to the Java I/O overhead. Recent network development using network processors such as IXP 1200 [12] exposes another hardware approach achieving this performance level, provided that Openet or similar programming platforms become available. But it has to resolve the same overhead issue when it deals with Java applications.

The software approach can also enhance the performance. In Openet, all network services except some low-level service implementation are programmed using Java. Java is good across platforms, but poor at performance (see Figure 5). Improving JVM with code caching and JIT (just-in-time) technologies cannot change this weakness since the active code is dynamically loaded. However improving JVM in expediting Java I/O operations is absolutely imperative and efficiently.

Moreover, re-engineering the active network services to couple tightly with commercial network hardware platforms so that they can make better use of the high-performance forwarding ability. This is potentially a viable solution to the data-plane services, however it must avoid the platform-dependent issue. The Openet service hierarchy can solve it by building high-level application services upon low-level system services that shelter the hardware diversity. See next discussion for detail.

Finally, what can Openet do for that? In fact, Openet does not affect the service execution except starting and terminating. Once a service is activated, the ORE Oplet encapsulation that is just a wrapper of the service code does nothing with the service functionality. In the future, through controlling use of resources (e.g., CPU, memory) per service, Openet can prioritize but improve service performance.

3) With Openet, what active network services are applicable to current commercial hardware?

The Openet's goal is to provide a hardware-independent programming platform that users can create and deploy network services with their intelligence and application purposes onto commercial network devices. Active network services are particularly important but computation-intensive, thus not all of them are sufficiently supported by current commercial hardware due to low CPU competence. However some of them are, particularly those do not generate large traffic and/or perform real-time communication.

As stated in Section III, in Openet, according to the planes they serve all the network services including active network services can also be divided into control-plane and data-plane services, regardless what levels they are. Data-plane services usually are involved in application-specific communication and can apply user-defined protocols rather than well-defined protocols to packet processing. Some data-plane services such as active multimedia services require service quality guarantee such as real-time support, and thus are not applicable to current network devices. Other data-plane services like ANTS have loose or elastic time requirement and are thus applicable.

Control-plane services mainly deal with network management such as configuration, service initiation, policy notification and monitoring. They affect the data-plane behaviors such as forwarding and routing by changing service policies, but do not process data packets directly. These services can be done at system startup, on schedule, or triggered by events, therefore they have few strict requirements such as real-time configuration. They are definitely the most potential network services applicable to current commercial network devices.

Moreover, with Openet, customer services such as active network services are hardware-independent. On commercial network nodes like the Accelar, the core part ORE sits in the control plane. Through the Oplet encapsulation, network services whether they are active networking-based or not are easily encapsulated as Oplets, and ORE treats them in the same way. Furthermore, through the service hierarchy, customer services at the data plane are loaded to the control plane and then access the data plane through invoking low-level services. Some low-level services such as JFWD and JMIB requiring both Java and native codes may rely on platform-dependent implementations but provide platform-independent APIs. They can be used by Active network services such as Netscript [7] that requires access local routing and interfaces and active IP accounting [3] that uses local traffic monitoring features. Since ORE is Java only and platform-neutral, customer services are also programmed using Java so that they are portable to other network platforms.

VI. CONCLUSION

This paper presents two major contributions. The first contribution is that Openet is presented as a programmable networking platform by which customer network services including active networks services can be deployed onto commercial network devices. The deployment of the ORE ANTS service on the Accelar routing switch shows that now it becomes doable bringing the active networking technology to real network nodes. Moreover, these services under Openet are portable across network hardware platforms.

The other is that through experiment the performance of active networking on the commercial network hardware platform is examined using two commercial Accelar routing switches. Since commercial hardware like the Accelar has high forwarding ability for pre-defined network protocols but limited packet processing ability for customized protocols, the throughput of active network traffic is not comparable to that of regular traffic. The main reason is that Java network I/O operations is much more serious in impacting the active networking performance than Java capsule processing. Several hardware and software approaches have been analyzed in commercial network devices improving the performance, particularly employing stronger CPUs in network hardware and re-engineering the active network services to make better use of the wire-speed forwarding hardware.

Through service classification, we realize that some active network services including most control-plane and certain data-plane services are applicable to current commercial network hardware, as long as they do not require real-time communication or heavy packet processing.

At this stage, we’re having some works doing with Openet, active networking and network hardware platforms. First, we’re exploring Openet to develop practical active networking applications such as new service creation, QoS provisioning and monitoring. Second, porting Openet services to more network hardware platforms including non-Nortel brand and network processor-based network devices is highly demanded. Third, Openet will introduce new mechanisms in improving service deployment, e.g., service security enhancement and thread-based resource allocation. Finally, even though latest commercial network products such as Accelar 8600 are equipped with fast CPUs, how to make most use of both CPU and forwarding hardware in enhancing the ability of customized packet processing in both control-plane and data-plane and finally the performance of active networking is another investigating issue.

ACKNOWLEDGEMENT

The authors would like to thank Franco Travostino and Doan Hoang at Nortel Networks Corp. for their reviews and constructive suggestions, and Ramesh Durairaj for his help in the experiment setup. We also thank Norman Brickman at MITRE Corp., Steve Zabele and his team at TASC Inc. for their valuable discussions about deploying the ANTS service on the Accelar routing switch.

REFERENCES

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

[2] David L. Tennenhouse, et al, “A Survey of Active Network Research”, IEEE Communications Magazine, Vol. 35, No. 1, January 1997

[3] F. Travostino, “Active IP Accounting Infrastructure,” IEEE OpenArch 2000, Tel Aviv, March 2000.

[4] Nortel Networks Corp., “Networking Concepts for the Passport/Accelar 8000 Series Switch”, April 2000

[5] D. Scott Alexander, et al , “The SwitchWare Active Network Architecture”, IEEE Network Special Issue on Active and Controllable Networks, vol. 12 no. 3, July 1998

[6] David J. Wetherall, John Guttag, and David L. Tennenhouse, “ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols”, IEEE OPENARCH'98, San Francisco, CA, April 1998.

[7] Y. Yemini and S. da Silva. “Towards Programmable Networks”, IFIP/IEEE Intl. Workshop on Distributed Systems: Operations and Management, L'Aquila, Italy, October 1996.

[8] D. Scott Alexander, et al, “ANEP: Active Network Encapsulation Protocol”, Active Networks Group, Request for Comments, http://www.cis.upenn.edu/~switchware/ANEP/docs/ANEP.txt, July 1977

[9] Active Network Backbone (ABone), http://www.isi.edu/abone/

[10] B. Schwartz, A. Jackson, T. Strayer, W. Zhou, R. Rockwell and C. Partridge, “Smart Packets for Active Networks”, IEEE OpenArch 99, New York, March 1999

[11] Phil Wang, “ORE ANTS service on Accelar”, http://www.openetlab.org/downloads/HOWTO.ore-ants, May 2000

[12] Intel Internet Exchange Arcitecture (IXA), http://developer.intel.com/design/ixa/white_paper.htm