Accelar Programmable Networking and Active Networks
—— A Technical White Paper

The Openetlab Team, Nortel Technology Center
Revised date: June 20, 2000

Table of Contents

1. Introduction....1
- Challenges of legacy networking technology....1
- Next generation networking: AN and Opensig/PIN....2
- What does Nortel offer? Accelar, ORE and OpenetLab....2

2. Active Networks....2
- Initiatives....2
- Technology: "Smart Packet" and "Smart Node"....3
- Typical research projects....3

3. Accelar --- A Programmable Gigabit Routing Switch....3
- What's Accelar?....3
- Architecture....4
- Why Accelar is good? Forwarding and control planes separated!....4
- Why AN needs Accelar?....4
- How is a new service injected?....5

4. ORE, JFWD and Network Services....5
- Service and Oplet....5
- ORE architecture....5
- JFWD....6
- ORE standard services....7
- Other ORE services....7

5. Interesting Issues....8
- How a service works on Accelar?....8
- How Accelar does application aware networking?....8
- How can I create a service and its oplet....8
- Simple examples....9

6. An Active Networks service: ANTS on Accelar....9
- ANTS....9
- AntsNodeService....9
- ORE ANTS package....10
- Test: APing....10
- Downloads....11

7. Related Resources....11

End of Paper....12

1. Introduction

- ***Challenges of legacy networking technology***

The Internet is one of the most successful innovations in the 20th century, and makes the global connected in one network. However, current networks including the Internet are virtually static and built using those networking technologies that were originated several decades ago. Their network nodes provide moreover fixed network services (e.g., IP forwarding) that are programmed only by the equipment vendors. As these networks are inflating (still at a high speed), they are obviously demanded to provide the advanced networking abilities such as QoS (quality of Services), dynamic composite protocols, rapid network service deployment, robust network security and flexible network management.

- ***Next generation networking: AN and Opensig/PIN***

Since the mid-1990s, two major communities, DARPA ITO and IEEE Comsoc (Communication Society) (following Opensig) have been contributing their efforts on next generation networking technologies to meet the above challenges. The DARPA AN (Active Networks) Program explores the active or "smart" networking architecture technology. The Opensig (Open Signaling) working group argues open network control issues as they arise in signalling, middleware and service creation, and is pursued by the IEEE P1520 project on PIN (Programming Interfaces for Networks) that attempts to standardize the programming interfaces for ATM, IP and mobile networks.

Moreover, both technologies assume the future network nodes are open and programmable so that one can deploy new network services and programming interfaces.

- ***What does Nortel offer? Accelar, ORE and OpenetLab***

Nortel Networks Corporation ("Nortel" for short) is a worldwide leading network technology and equipment provider, and fully supports those investigations towards next generation networking technology. Furthermore, Nortel's new generation network products are equipped with the network programming ability that meets the needs of two above research communities. Of these Nortel products, the Accelar routing switch (see section 3) is one typical product of the programmable network devices and the ORE SDK (see section 4) is one typical of the software development kits.

On the other hand, the "OpenetLab" team at the Nortel Technology Center is concentrated on integrating the latest research results with the industrial activities. OpenetLab has its own website www.openetlab.org, and also provides support of Nortel products Accelar and ORE, related technologies and resources.

Return to TOC.

2. Active Networks

- ***Initiatives***

The DARPA AN (Active Networks) Program has two main initiatives in (1) composite protocols using "smart packet" and (2) enhanced network services in order to quickly and safely deploy new services. One obvious benefit is that with AN the Internet service providers (ISP) even individual users can deploy or compose new services and protocols "on-the-fly" without long-time waiting for their standardization or the participation of the equipment vendors.

- ***Technology: "Smart Packet" and "Smart Node"***

The core technology of Active Networks is that it introduces two terms "Smart Packet (a.k.a. Active Packet or Capsule)" and "Smart Node (a.k.a. Active Node)". A smart packet carries not only a protocol header and a payload (as a regular IP packet does) but also a program code segment. The program code is executed at smart network nodes as a particular service to processes the packet data or oncoming packets. A smart node is equipped with or can dynamically download and execute the active codes required by smart packets.

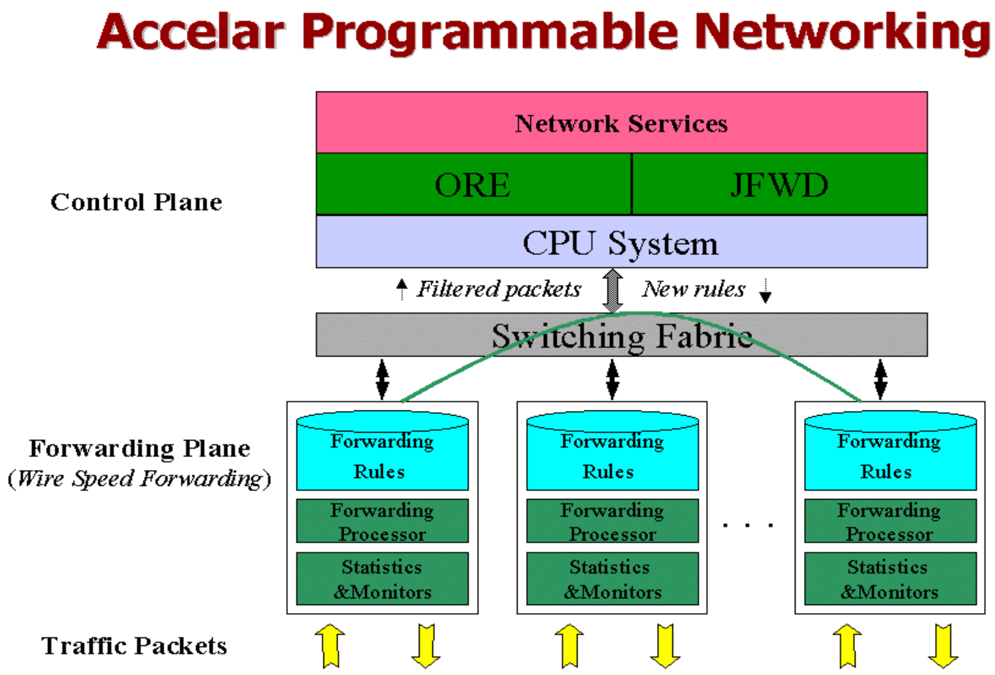
- ***Typical research projects***

The Active Networks community has conducted a wide range of research projects (some of them are listed below). The hyperlinks to these projects are available here.

- *OS*: JanOS (U. Utah), Scout(U. Princetion), AMP (NAI)
- *EE and AA*: ANTS (MIT), Netscript (U. Columbia)
- *Security*: Sands (NAI)
- *Network Management*: Actiware (U. Columbia)
- *Application Infrastructure*: Ninja (UC Berkeley)
- *Application Middleware*: Panda (UCLA)
- *Architecture and Testbed*: Switchware (U. Penn), Abone (USC/ISI)

Return to TOC.

3. Accelar --- A Programmable Gigabit Routing Switch



- ***What's Accelar?***

Accelar is a new Nortel product family of L3 Routing Switches and employs a distributed ASIC (Application Specific Integrated Circuit) -based forwarding architecture that can do packet forwarding at 5.6-256 Gbps. Each ASIC is responsible for four physical 10/100 Ethernet ports or a single gigabit port. The switches scale up to 384 10/100 ports or 96 Gigabit ports (or some combination of the above). There are up to eight hardware-forwarding queues per port corresponding to normal and high priority packets. The hardware is controlled using the VxWorks real-time OS, the networking programmability is provided by the ORE and related service APIs.

- ***Architecture***

The Accelar layered architecture is depicted here, and includes two separated planes: control and forwarding. These forwarding engines at the forwarding plane process the incoming packets and then forward the outgoing packets according to the forwarding rules. All the processing at the forwarding plane is done in ASIC at a wire speed.

The CPU system is utilized by the control plane that covers ORE, JFWD and network services. Network services are managed by ORE, and access the forwarding plane by invoking the JFWD APIs. They can alter the hardware instrumentation (MIBs) such as setting packet filters in order to divert filtered packets to the CPU, change the forwarding priority and provide new protocol support.

- ***Why Accelar is good? Forwarding and control planes separated!***

Traditional networks devices such as routers use CPU to perform two tasks simultaneously, one is to forward packets and the other is to control the forwarding policies. The two tasks compete the CPU power, memory and other resources so that they cannot reach high performance.

Accelar is different from those devices by separating the two tasks into two planes: control and forwarding. The “forwarding” plane is wholly implemented using ASIC that forwards packets at a wire speed, and does no longer consume CPU. The “control” plane occupies the whole CPU so that it has sufficient processing power to control the forwarding plane in real time.
Moreover, the Accelar control plane supports ORE that is an open network programming environment used for deploying customer network services.

- ***Why AN needs Accelar***?

Current AN technologies are mainly realized in the computer boxes (or host systems) using a software approach that costs large computation power. The AN goal is to inject these technologies to the real network nodes. On the other hand, they also gain the high performance from the network nodes that use specific hardware technologies.

In order to meet the needs of Active Networks, next generation network nodes must have both high performance and programmability with open networking APIs (Application Programming Interfaces). However, current network nodes are low-performance, closed systems that do not allow any party rather than their makers to add any network service. It’s impossible to deploy Active Networks services onto those nodes.

Exceptionally, Accelar is a high-performance, open system where network services can be reprogrammed and deployed externally. It provides the platform-independent runtime environment (ORE) and Java APIs. Through Java programming, the AN services can be injected to Accelar.

- ***How is a new service injected***?

A new service programmed using Java is encapsulated (by Oplet) using the ORE interfaces. The encapsulated service package is then stored in an external downloading server (e.g., HTTP or FTP) rather than in the Accelar box. When the Accelar starts, the ORE downloads the service package using URL and activates the service and its dependent services if the service is specified at startup. Or, a network administrator can load that service package into the ORE and activates it using a telnet style shell or the ORE control API. It's noted that, if a service is dependent on other services, the ORE must download their codes and activates them before downloading and activating this service. Once the activation is done, that service has been injected and becomes a native service on the Accelar. Thereafter, this service can access the hardware instrumentation or conduct other works provided that it includes the appropriate APIs such as JFWD.

Return to TOC.

4. ORE, JFWD and Network Services

The Oplet Runtime Environment (ORE) supports injecting customized software, including the AN EEs, into network devices. It is actually a platform for secure downloading, installation, and safe execution of Java code within a JVM (Java Virtual Machine). The Java code or software is called as a "service" and wrapped as an "oplet" using the ORE interfaces.

- ***Service and Oplet***

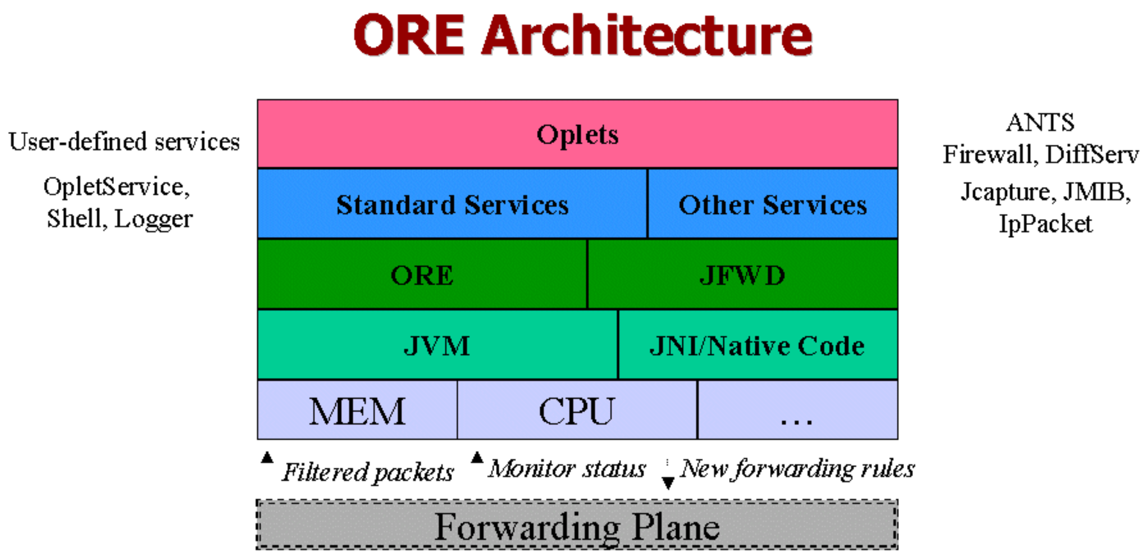
A service is a piece or package of Java code that implements specific functionality. A service may depend on other services and offer public interfaces for other service uses.

In order to securely download and impose policy, an oplet is defined as a self-contained downloadable unit that embodies a non-empty set of services. Along with the service code, an oplet specifies service attributes, authentication information, and resource requirements. Note that an oplet can encapsulate a service even it depends on some other services; in this case, an oplet also contains service dependency information.

- ***ORE architecture***

The ORE architecture consists of the Oplet Runtime Environment (ORE), oplets, and services. Services are encapsulated by one or more oplets, and oplets publish those services they provide to the ORE.

The ORE services can be classified into three categories: ORE-specific, system and customized. In Figure 1, "JFWD" is the system service for underlying packet forwarding and processing, and "Standard Services" are the ORE-specific services and APIs for customer service encapsulation and management. The customized services include both "Other services" and "Oplets". "Other services" are the ORE funcational services coming with the ORE packages while "Oplets" stand for the encapsulated user-defined services. The ORE provides the mechanisms to download oplets, resolve service dependencies, manage the oplet lifecycle, and maintain a registry of active services. The ORE downloads "trusted" oplets using URLs (e.g., HTTP, FTP or FILE), before it starts a new service or activates its dependent services.



On the network nodes, the ORE manages to install, activate, update, deactivate, kill, and uninstall oplets. Users can specify oplets that are automatically installed and activated when the ORE is started. One oplet can use a number of other services that are specified as the oplet attribute. Prior to activating an oplet, all services upon which it depends must be activated. When the oplet is activated it can then register the services it provides with the ORE. The ORE tracks these dependencies between oplets and services, so that if a service becomes unavailable the oplets that are dependent on it are notified.

- ***JFWD***

The JFWD (or Java Forwarding API) is a low-level service that provides platform-independent Java APIs that customer services use to control the forwarding (plane) behavior by accessing the underlying hardware instrumentation on various platforms. It includes several fundamental mappings such as MAC address, ARP, IP routing, filters and VLAN (Virtual LAN).

A major use of the JFWD API is to instruct the forwarding engine to alter packet processing through the installation of IP filters. On Accelar, those IP filters are set to the hardware filters that execute "actions" specified by a filter policy. A filter is based on the MAC, IP or transport protocol header, or their combination. The policy can define where the matched packets are delivered and can also be used to alter the packet content. Packet delivery options include discarding matched packets, or conversely forwarding matched packets if the default behavior was to drop them, and diverting matched packets to the

CPU system (i.e., the control plane). Diverting packets to the control plane allows customer services such as AN EEs to process packets. Additionally, packets can be "carbon copied" to the control plane or to a mirrored interface. The filter policy can also cause packet and header content to be selectively altered (e.g., the Type of Service or the DSCP bits on matched packets can be set).

JFWD implementation on different platforms (e.g., Accelar and Linux) requires use of native codes or communications. On the Accelar, the JFWD native codes turn out to be a wrapper around the hardware instrumentation interface.

- ***ORE standard services***

These standard services provide ORE-specific APIs such as service encapsulation and features such as service startup. They are essential to encapsulate customer services and conduct user interaction with the ORE.

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

- ***Other ORE services***

The ORE also comes with other services that provide some common function APIs that can be used to program high-level application services conveniently. Some of them (e.g., HTTP) are platform-neutral, and the others rely on particular system supports (e.g., Jpcap).

- *HTTP* (ahttp.jar): HTTP service
- *Jcapture* (jcapture.jar): Packet capturing service, sets IP filters and diverts packets to CPU
- *IpPacket(util_packet.jar)*: IP packet utility service, constructs IP/TCP/UDP header and payload
- *JMIB (jmib_*.jar)*: platform MIB access service, provides access to local hardware instrumentation
- *JPCAP (jpcap.jar)*: packet capturing service, provides use of the local Berkeley *libpcap* if possible

Return to TOC.

5. Interesting Issues

- ***How a service works on Accelar?***

In the Accelar architecture, the user-defined services are on the top level. Once they are injected, they are native applications running through the ORE, and monitor and control the ASIC hardware via a low-level service API (JFWD). The JFWD API hides the hardware-specific details, and provides generic access to hardware instrumentation variables (including routing tables and forwarding policies) by giving the application services a one-to-one mapping to hardware functions. For example, through JFWD, a service can install packet filters that have the Accelar forwarding engines to inspect and modify packet headers at wire-speed.

- ***How Accelar does application aware networking?***

The Accelar control plane supports TCP/IP-based socket communications so that an end application can communicate with specific services in order to perform given network operations and monitor the network status. A user can program a particular network service, and inject it to the Accelar using ORE. At a remote node (e.g., a Linux box), an application can invoke the service API to perform the requested network operations. The service (on the Accelar) can also send feedback to that application. A typical example is that a traffic monitor can request the Accelar firewall service to divert incoming packets and have the forwarding planes stop a network attack (such as DoS). Another example is that an RTP/UDP application requests the Accelar DiffServ service to set a given forwarding priority to forward those UDP packets matching the RTP filters.

- ***How can I create a service and its oplet***

Since ORE is a pure Java environment, a network service should be programmed using Java codes. The Java package of a service includes the service interface, the service implementation, and its oplet that has a Java class and a manifest file (service description and dependency). Generally, a service may include three Java classes at least and one manifest file.

➢ *MyService.java*:	the service public interface class, extends OpletService
➢ *MyServiceImpl.java*:	the service implementation class, implements MyService
➢ *MyServiceOplet.java*:	the Oplet class, encapsulates the MyService
➢ *MyServiceOplet.mf*:	the Oplet manifest file, contains the service info

A service may have other files such as Makefile and some documents (e.g., README). Finally, a completed service is packed into a jar file, which is installed to a downloading server (e.g., HTTP or FTP) or the local ORE directory ”ore/jars/”.

If you have already the service code, “MyServiceImpl.java” can be simply a Java wraper that encapsulates the service code, as we did in the ORE ANTS service (on Accelar). And, you may not have to change the service code.

- ***Simple examples***

Here two examples are presented, one is a simple service (examples.hello.Hello) that returns a message on the number it has been called since started. The other is a testing package (examples.helloclient.HelloClient) that acquires this service. Strictly speaking, the later one is also a service, indeed a testing service even though it provides an empty service to ORE.

These files are listed below and can be downloaded from the Openetlab website. Click here to download the two packages that are packed as a gzipped tar file.

Service 1: examples.hello.Hello

- *examples/hello/Hello.java*: the service interface class
- *examples/hello/HelloImpl.java*: the service implementation class
- *examples/hello/HelloOplet.java*: the Oplet class
- *examples/hello/hello.mf*: the service manifest file
- *examples/hello/Makefile*: the make file

Service 2: examples.helloclient.HelloClient (no service class)

- *examples/helloclient/Oplet.java*: the Oplet class
- *examples/helloclient/Oplet.mf*: the service manifest file
- *examples/helloclient/Makefile*: the make file

Return to TOC.

6. An Active Networks service: ANTS on Accelar

For the first time, we've deployed the ANTS (Active Node Transfer System, proposed by MIT) successfully onto the Accelar 1100B routing switch. We do not change the original ANTS code, instead add a new service "AntsNodeService" that encapsulates the code. Thus, the ORE ANTS service includes two packages.

- ***ANTS***

ANTS is a typical Active Networks project for building and deploying new network protocols dynamically at routers and end systems. It uses mobile code, demand loading, and caching techniques, and provides a software package that comes with a toolkit and several active applications such as ping and multicast.

We download the MIT Java package, and do not change the Java code that is still named "package ants". We have to change the ANTS configuration files "ants.config" as well as "ping.routes" in order to reflect our testbed.

- ***AntsNodeService***

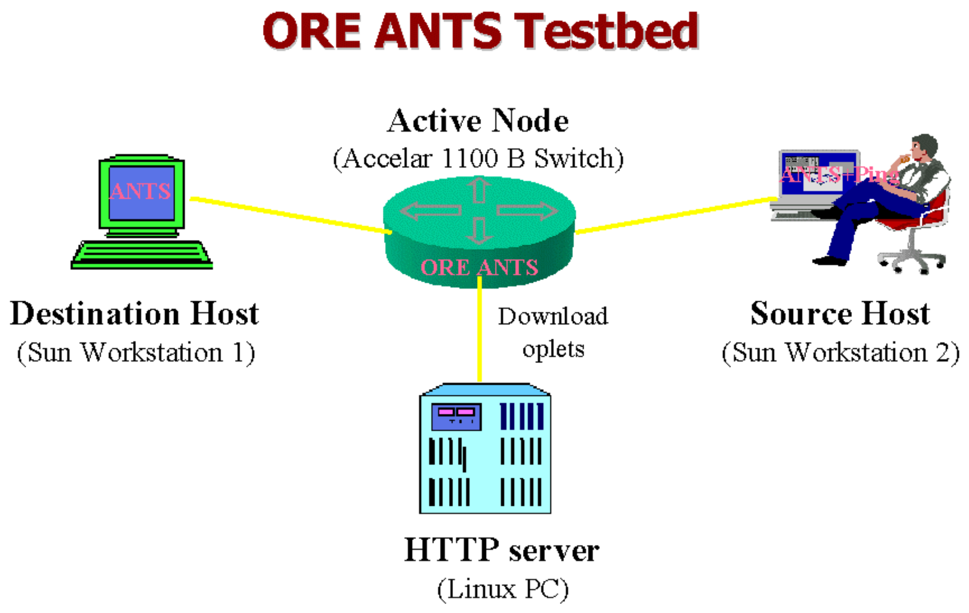
"AntsNodeService" is the ORE wrapper service that covers the above ANTS code. The Java code is named "package com/nortelnetworks/ore/service/ants", and includes the below files.

- *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 class "AntsNodeService.java" in Figure 7 includes the ORE ANTS service description, and two methods: getNode() that connects the ANTS code, and getConfiguration() that reads the ANTS configurations to set up the service.

- ***ORE ANTS package***

We use a jar file "ore-ants.jar" to pack the above two Java packages, "package ants" and "package com/nortelnetworks/ore/service/ants". This jar file is stored in a Linux HTTP server for downloading. When the ORE starts the "AntsNodeService", it firstly loads "package com/nortelnetworks/ore/service/ants". Then, class AntsNodeServiceImpl invokes the "package ants" to set up the real ANTS service.



- ***Test: APing***

When booting the Accelar (using the ORE boot image), we input the URL of the ORE startup service (e.g., http://192.168.1.1/jars/start.jar) so that the ORE on the Accelar downloads the ORE ANTS and other service jar files from the HTTP server (a Linux box). Before doing this, we edit the ORE startup file "jars/start.properties" to include the ANTS service package "ore-ants.jar". We also change the ANTS configuration files "ants.config" and "ping.routes" to use the Accelar as the active node and specify the source and destination hosts. These files are also stored on the same Linux box.

After the ORE ANTS service is loaded to the Accelar 1100 B switch (i.e., the ORE activates the ANTS service), we experiment that service with a small active net that consists of four computer boxes (as in the above figure).

- ➢ *Accelar 1100 B switch*: the active router, runs ORE and loads AntsNodeService
- ➢ *Sun Solaris workstations* 1: destination active node, runs MIT ANTS
- ➢ *Sun Solaris workstations* 2: source active node, runs MIT ANTS and ANTS Ping
- ➢ *Linux PC*: an HTTP server, provides AntsNodeSerivce codes and configuration

We also install the MIT ANTS package onto the two hosts, and configure the source host to run the APing application sending capsules and the destination host to run the ANTS daemon receiving the capsules. On the source node, APing pops up a GUI window where we set the input fields "Num of Iterations (10)" and "Ping Interval (1ms)". Then, click the "Ping" button to start.

The source host sends out the ANTS ping (APing) capsules to the Accelar. Initially, the Accelar does not know how to process the first capsule and requests the APing application from the source of the ANTS ping capsule. The source forwards the APing code to the ORE ANTS service that installs the code in the Accelar ANTS EE. Once installed, the APing application on the Accelar "executes" the program code of those ping capsules resulting in forwarding them to the next ANTS-enabled node. When the destination sends the ANTS ping capsules back through the Accelar (now with the ANTS EE service installed), the capsules are processed and forwarded to the source ANTS EE.

The receipt of all the ANTS ping capsules at the source verified that we had successfully implemented an interoperable ANTS EE implementation on a commercial-grade gigabit routing-switch. So, the test result shows that the ANTS is successfully deployed on the Accelar and the Accelar becomes an active node now!

- ***Downloads***

The whole ORE ANTS package is available for downloading via "http://www.openetlab.org/downloads/". It is a gzipped tar file "ore-ants.tar.gz", and includes the ORE ANTS code and configuration (for Accelar), the MIT ANTS code and configuration (for end hosts), and READMEs.

Return to TOC.

7. Related Resources

The current Accelar routing switch is Accelar 1100 B with the VxWorks real-time OS. The development tools can be downloaded from the Openetlab website, including the ORE boot image, ORE SDK and other services. The current JVM version is 1.1.7 and the ORE is 0.3.3.

- ➢ *Website*: http://www.openetlab.org
- ➢ *Registration*: http://www.openetlab.org/registration.htm
- ➢ *ORE*: http://www.openetlab.org/ore.htm

- *Examples (Hello and HelloClient):* http://www.openetlab.org/downloads/examples
- *Downloads (ORE, docs and software):* http://www.openetlab.org/downloads
- *Related Projects*: http://www.openetlab.org/related_resources.htm
- *Mailing lists*: active_networks@openetlab.org , open_networks@openetlab.org
- *Reference Papers*: http://www.openetlab.org/related_resources.htm#papers

Please send your comments to the editor ***Phil Wang*****.**

Return to TOC.

End of Paper