# Enabling Active Networks Services on A Gigabit Routing Switch

Tal Lavian and the Openetlab Team

# CONTENTS

- Challenges of Customized Networking and The Active Networks Approach
- The Gigabit Routing Switch: Accelar
- The ORE Programmability
- ORE services and Customer Deployment
- The ORE ANTS: an example of injecting AN services into network nodes
- Summary

# Challenges of Customized Networking

- Ever more functionality done in hardware
  - good: bring faster processing ability
  - bad: reduce the opportunity to introduce new services inside the network
- Legacy network nodes employing a static and well-defined set of protocols
  - closed systems that allow configuration of existing services but do not allow service addition
  - Unsuitable for hosting the deployment of customer services including Active Networks services

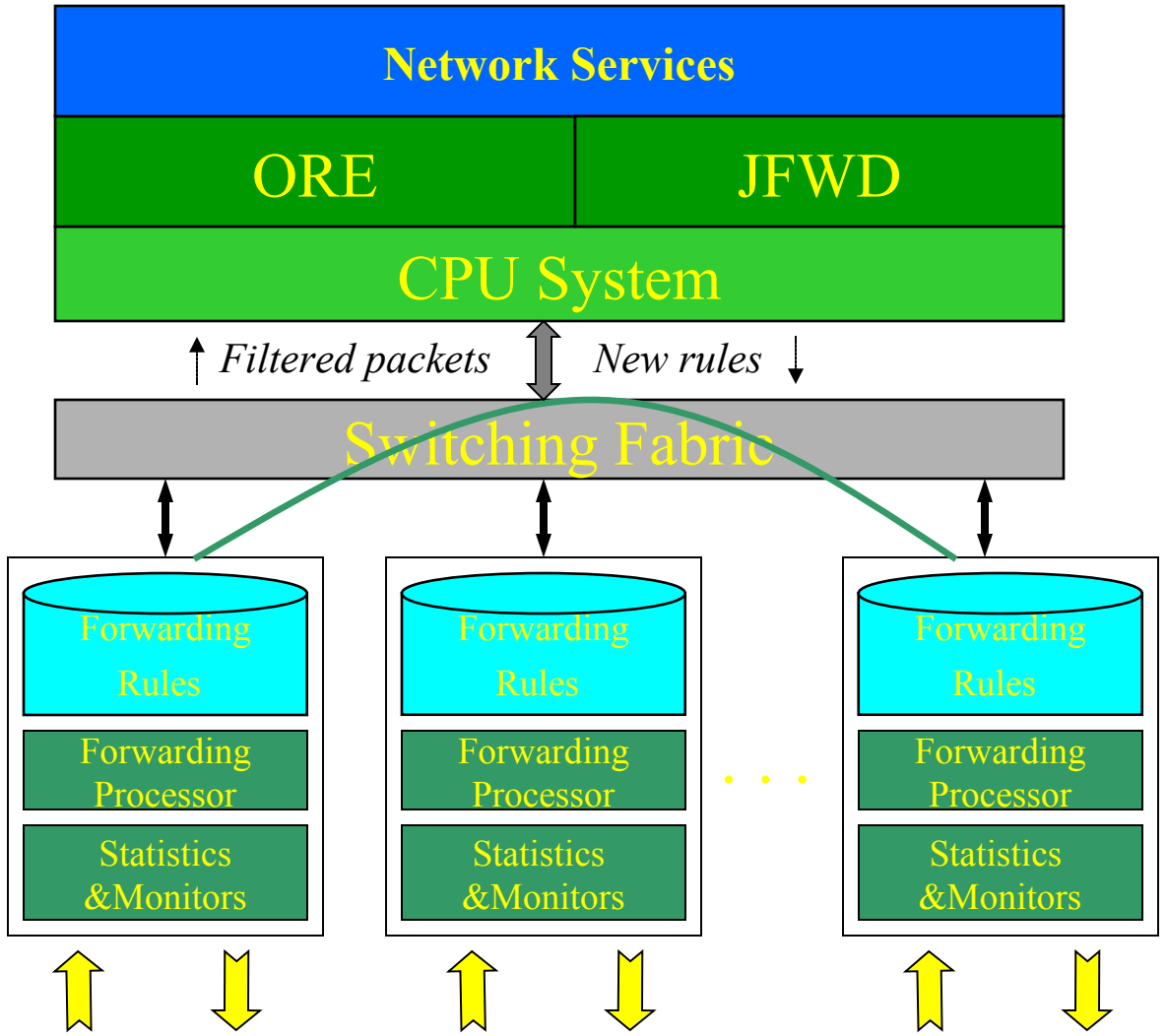
# Active Networks

- A “programmable” user-networking approach
  - injects network services to the network “on-the-fly”
  - supports per-flow service customization
  - enables ISPs and individuals to add their services
- To support AN, hardware should provide
  - Fast processing ability to compete AN computation
  - the programmability with open networking APIs

# The Accelar Routing Switch

- A Nortel Networks L3 Routing Switch Family
  - distributed ASIC forwarding architecture
  - packet forwarding up to 256 gbps
  - VxWorks real-time OS
  - ORE networking programmability
- High performance by two separated planes
  - Forwarding: forwards packets at a wire speed
  - Control: processes policy control as well as supports the ORE services

# Accelar Programmable Networking



# The ORE Programmability

- ORE: an Oplet Runtime Environment for injecting customized software into network
  - an open platform for secure downloading, installation, and safe execution of Java code
  - provide user-level service API
  - network services implemented using Java code

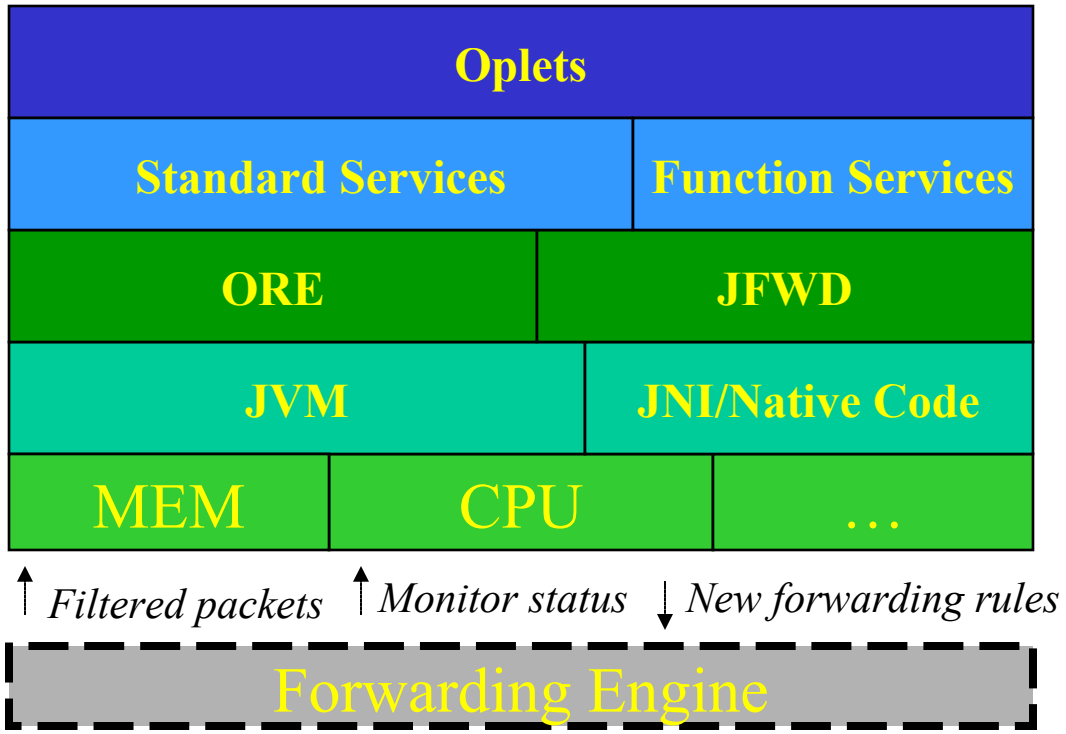
# The ORE Programmability

- *Oplet:* a self-contained downloadable unit
  - encapsulates one or more services
  - contains service attributes such as dependency
  - Secure downloading, service installation
- *Service:* a downloadable code that implements a specific functionality
  - includes Active Networks services: EE
  - Can be built on the top of other services
  - examples: filtering packet, altering forwarding priority and diverting packets

# ORE Architecture

User-defined services

OpletService,
Shell, Logger




ANTS
Firewall, DiffServ

Jcapture, HTTP,
IpPacket

# ORE Services

- Three categories
  - Standard: ORE-specific APIs for customer service encapsulation and management
  - System: low-level or underlying access APIs such as packet forwarding and processing services
  - Customized: user-level service APIs
    - Function: ORE or user services for common use
    - Oplets: application-specific customer services

# ORE Services

- System Services
  - JFWD: Java Forwarding API, see next slide
  - *JMIB*: platform MIB access, provides access to hardware instrumentation
  - *JPCAP*: packet capturing, provides use of local Berkeley *libpcap*

# ORE Services

- JFWD: a system service
  - Java Forwarding API, platform-independent
  - controls packet processing and forwarding
  - provides access to the hardware instrumentation
  - typical network mappings
    - IP filters: drop, forward and capture packets
    - IP routing
    - MAC address, ARP and Vlan
  - native implementation on Accelar and Linux

# ORE Services

- Standard Services
  - *OpletService*: Oplet service API, extended to define service descriptions and interfaces
  - *ManifestOplet*: Oplet encapsulation abstract interface, implemented to create service-specific oplets
  - *Start*: ORE startup service, loads given services at startup
  - *Shell*: telnet-like user interface, provides shell commands to manipulate oplets and start or stop network services
  - *Logger*: ORE log service, provides runtime logs

# ORE Services

- Customized services
  - *HTTP*: HTTP service
  - *Jcapture*: packet capturing service
  - *IpPacket*: IP packet utility, constructs IP/TCP/UDP header and payload
  - *JMIB*: platform MIB access, provides access to hardware instrumentation
  - *JPCAP*: packet capturing, provides use of local Berkeley *libpcap*

# Customer Service Deployment

- Customer service programming
  - regular Java programming
  - two ORE APIs: OpletService and ManifestOplet
- Service code packed in jar and stored in downloading servers
- ORE downloads service code and starts particular services as instructed
- A service can be built using other services

# Customer Deployment: ORE API

- OpletService: the ORE base service
  - Extended by customer service interface classes to define service description and interfaces
  - customers also provide the service implementation classes to implement those interface classes
  - service implementation classes should include two additional private methods for starting and stopping the service function respectively

# Customer Deployment: ORE API

- ManifestOplet: the abstract *oplet* interface
  - implemented by customers as concrete oplets to encapsulate the service code
  - has two methods *startService*() and *stopService*() to register or deregister a service at runtime
  - accompanied by manifest files to cover service information, e.g., oplet name, service description, dependency and package name

# Customer Deployment: package

- What are includes in a service package?
  - *Hello.class*: the service interface class, extends OpletService
  - *HelloImpl.class*: the service implementation class, implements the interface Hello
  - *HelloOplet.class*: the Oplet class, implements Manifest and encapsulate service Hello
  - *HelloOplet.mf*: the service manifest file, provides the service information

# Customer Deployment: start

- How to start customer services? 2 ways at least
  - at startup
    - the ORE startup service (start) starts those services specified in “start.properties”, which is in the same directory of the service package “start.jar”
    - edit “start.properties” to add or remove your service packages
  - at runtime
    - customers can use the ORE shell service to manipulate those services by “telnet *OREHOST* 1999”
    - the whole service lifecycle can be instructed
  - through the ORE API by remote applications

# Customer Deployment: To Accelar

- Injecting customer services onto the Accelar
  - service code (i.e., jars) stored in external servers for downloading
  - services can be activated at startup or runtime
  - once activation successfully, those services work like native services on the Accelar

# ORE ANTS on the Accelar

- Deploying the ANTS on the Accelar using ORE
- MIT ANTS distribution
  - version 1.2
  - no modification to the ANTS code
- on the Accelar 1100B routing switch
  - ORE version 0.3.3
  - ORE ANTS package
  - URL: "http://www.openetlab.org/downloads/"
- An Active Networks service implementation

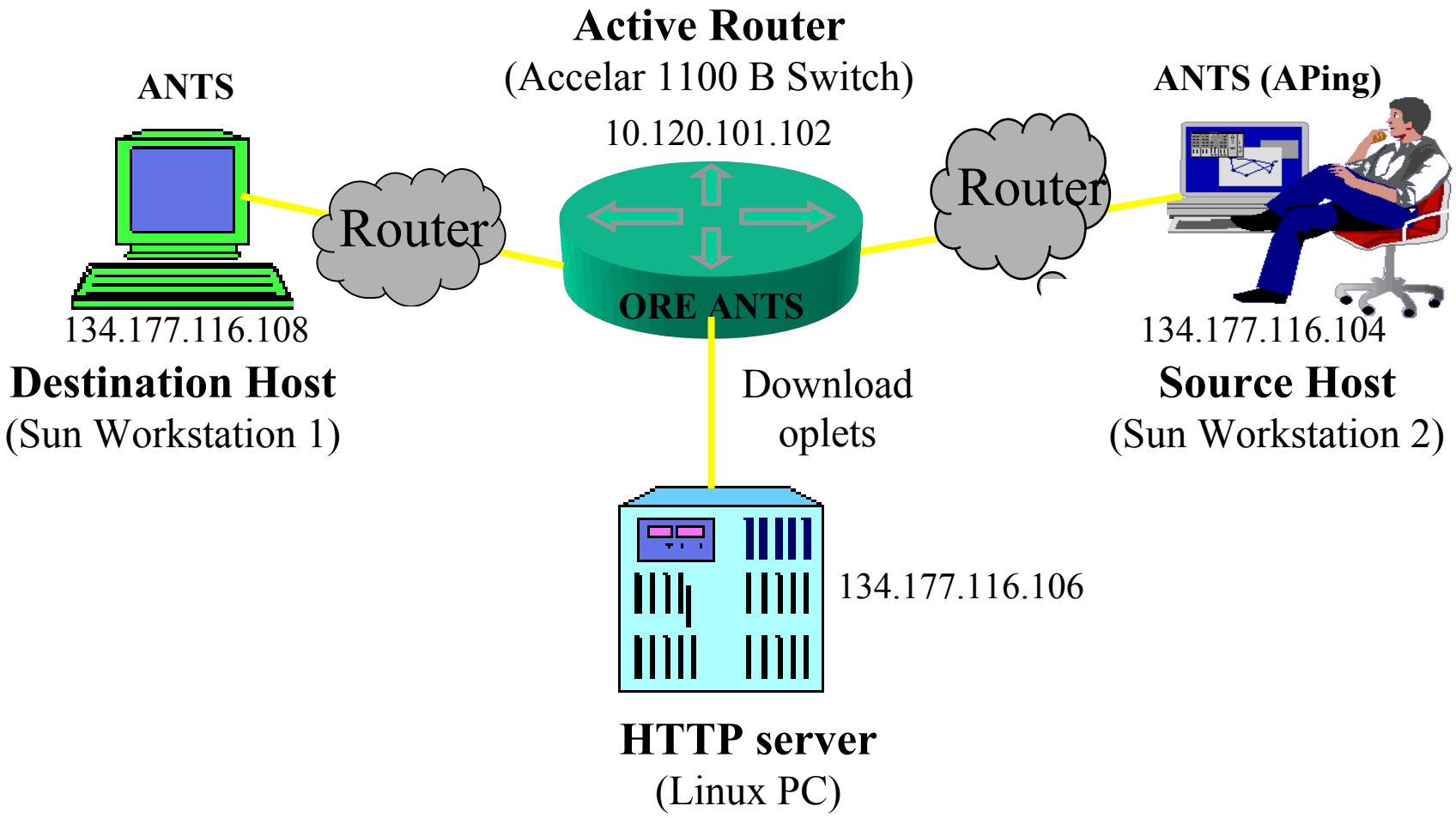
# ORE ANTS: service

- Service: “AntsNodeService”
  - wrapping the MIT ANTS code
  - package “com.nortelnetworks.ore.service.ants”
    - *AntsNodeService.class*: the AntsNodeService interface
    - AntsNodeServiceImpl.class: the service implementation
    - AntsNodeOplet.class: the Oplet
    - AntsNode.mf: the manifest
  - service interfaces
    - getNode(): connect to the ANTS code
    - getConfiguration(): set up the service using ANTS configuration

# The ANTS Ping (Aping) Test

- **The ORE ANTS service tested by APing**
  - **an experimental active net built within Nortel**
  - ***Accelar 1100B:*** the active router with ORE ANTS
  - ***Sun workstations 1:*** destination active node with MIT ANTS
  - ***Sun workstations 2:*** source active node with MIT ANTS (and APing)
  - ***Linux PC:*** the HTTP server providing the ORE service jar packages and the ORE ANTS configuration

# ORE ANTS Testbed



# Summary

- ORE brings the programmability to network
- The ORE ANTS deployment on the Accelar is a successful instance of injecting Active Networks (AN) services to network nodes
- Porting AN services to ORE is rather easy
- If necessary, JFWD or other system services are used by customers to access underlying resource or hardware instrumentation
- Accelar is still working on strong CPU competence & flexible ASIC programmability