Jae Woo Lee, Jan Janak, Roberto Francescangeli, Suman Srinivasan, Salman A. Baset, Eric W. Liu, Michael S. Kester and Henning Schulzrinne
Internet Real-Time Laboratory, Columbia University
In collaboration with Wolfgang Kellerer and Zoran Despotovic at DOCOMO Euro-Labs,
Volker Hilt at Bell Labs/Alcatel-Lucent and Srini Seetharaman at Deutsche Telekom

NETSERV: PROGRAMMING NETWORKS (AGAIN)
What is NetServ?

- In-network service container
- Java-programmable, signal-driven router
- “GENI Lite” – deploy modules, not VMs
- Active networking that works™
Overview

- Old world
  - (computation, storage) $\leftrightarrow$ forwarding

- 1990s: active networking
  - no good isolation infrastructure
  - limited applications

- Exploring new opportunities
  - providing additional services in the current Internet $\rightarrow$ NetServ
  - future Internet architecture
Two worlds

1 interface
TB disk
1-32 multi-core processors

10+ interfaces
0 GB disk
1 low-end processor
Storage costs

http://www.mkomo.com/cost-per-gigabyte
Bandwidth costs

- Amazon EC2
  - $100/TB in, $100/TB out
- CDN (Internet radio)
  - $600/TB (2007)
  - $100/TB (Q1 2009 – CDNpricing.com)
- NetFlix (7 GB DVD)
  - postage $0.70 round-trip → $100/TB
- FedEx – 2 lb disk
  - 5 business days: $6.55
  - Standard overnight: $43.68
  - Barracuda disk: $91 - $116/TB
Software: from floppy to autonomous
NetServ overview

Extensible architecture for core network services

- Modularization
  - Building Blocks
  - Service Modules
- Virtual services framework
  - Security
  - Portability
- NSF FIND four-year project
  - Columbia University
  - Bell Labs
  - Deutsche Telekom
  - DOCOMO Euro-Labs
Architecture overview

NetServ controller

Module download

Signaling message to install module

Module install

Signaling message forwarded to next hop

Service modules

Building block layer

Virtual execution environment

Service modules

Building block layer

Virtual execution environment

Service modules

Building block layer

Virtual execution environment

Data packets processed by service modules

NetServ packet transport
Network node example

data center or POP

storage & computation
multiple computation & storage providers
Different from active networks?

- **Active networks**
  - Example: Packet contains executable code or pre-installed capsules
    - Can modify router states and behavior
    - Mostly stateless
  - Not successful
    - Per-packet processing too expensive
    - Limited storage (memory expensive)
    - Security concerns
    - No compelling *killer app* to warrant such a big shift
  - Notable work: ANTS, Janos, Switchware

- **NetServ**
  - Virtualized services on current, passive networks
    - Service invocation is signaling driven, not packet driven
    - Some flows & packets, not all of them
    - Emphasis on storage
  - Service modules are stand-alone, addressable entities
    - Separate from packet forwarding plane
    - Extensible plug-in architecture
How about GENI?

- GENI = global-scale test bed for networking research
  - parallel experiments in VMs
- → long-term, “heavy” services
- Demonstrated NetServ on GENI during GEC8
Related work

- Cisco’s Programmable Overlay Router
- Juniper’s JUNOS SDK
- DaVinci project
- OpenFlow
- ShadowNet (AT&T) – virtual routers
- MillionNode GENI (end systems)
NetServ service containers

- **NetServ service container**
  - User-level processes
  - Embeds a Java Virtual Machine
  - Runs OSGi
  - Service modules are dynamically installed and removed

- **NetServ Service Modules**
  - OSGi-compliant Java code
  - Can be stand-alone server (i.e., listens on a port)
  - Or can be packet processor
    - Packets are first routed from kernel to container using kernel queue (Netfilter queue in Linux and pseudo-device in Click Router)
    - Packets are then passed to Java using Java Native Interface (JNI)
Service modules

- Full-fledged service implementations
  - Use building blocks and other service modules
  - Can be implemented across multiple nodes
  - Invoked by applications
- Examples:
  - Routing-related services
    - Multicast, anycast, QoS-based routing
  - Monitoring services
    - Link & system status, network topology
  - Identity services
    - Naming, security
  - Traffic engineering services
    - CDN, redundancy elimination, p2p network support
Deployment scenarios

- Three actors
  - Content publisher (e.g. youtube.com)
  - Service provider (e.g. ISP)
  - End user

- Model 1: Publisher-initiated deployment
  - Publisher rents router space from providers (or end users)

- Model 2: Provider-initiated deployment
  - Publisher writes NetServ module
  - Provider sees lots of traffic, fetches and installs module
  - Predetermined module location (similar to robots.txt)

- Model 3: User-initiated deployment
  - User installs NetServ module to own home router or PC
  - or on willing routers along the data path
Where does code run?

- All (or some?) nodes in a network
  - AS, enterprise LAN
- Some or all nodes along path
  - data path from source to destination
- Selected nodes by property
  - e.g., one in each AS
How does code get into nodes?

All nodes in (enterprise) network

- gossip
- signaling
How does code get into nodes?
NSIS

- Progress along data path
  - with RAO-based discovery
- Designed to transport large objects
  - supports TCP and UDP
- Security mechanisms
NetServ signaling daemons

- NetServ signaling daemons
  - NSIS (RFC 4080) compliant
  - Two layers: Generic GIST and NetServ-specific NSLP
  - Receive signaling packets to setup/remove/probe modules
  - Command in signal packet handed to NetServ Controller
  - Node can be on-path – signal forwarded to next hop
  - Currently based on FreeNSIS implementation
Module management: OSGi

- “Dynamic module system for Java”
  - originally for set top boxes
- Why OSGi? Why not just JAR files?
  - More than just JAR files
    - much richer encapsulation
    - metadata in manifest
  - Automatic dependency resolution
  - Version management
  - Provides systems services (logging, configuration, user authentication, device access, …)
  - ~ Debian's apt-get or Apple's App Store methods of installation
OSGi features

- Dynamic module system for Java
  - Modules loaded and unloaded at runtime
  - *Bundle*: self-contained JAR file with specific structure
  - Open-source implementations: Apache Felix, Eclipse Equinox
- Security and accounting
  - Security built on Java 2 Security model
    - Permission-based access control
    - No fine-grained control or accounting for CPU, storage, bandwidth
    - Can load native code with appropriate permission
  - Strict separation of bundles
    - Classpath set up by Bundle class loader
    - Inter-bundle communication only through published interfaces
OSGi architecture

- **Architecture**
  - **Bundles**: JAR files with manifest
  - **Services**: Connects bundles
  - **Services Registry**: Management of services
  - **Modules**: Import/export interfaces for bundles

- **Possible to “wrap” existing Java apps and JARs**
  - Add additional manifest info to create OSGi bundle
  - E.g.: Jetty web server now ships with OSGi manifest; now extensively used with OSGi containers and custom bundles
  - For NetServ, we created a OSGi bundle for the Muffin HTTP proxy server
OSGi implementation

- Many core frameworks
  - Eclipse Equinox, Apache Felix, Knoplerfish
- Real-world examples
  - Eclipse IDE uses OSGi for plugin architecture
- Mostly finds use in enterprise applications needing plugin functionality
  - E.g.: IBM Websphere, SpringSource (now VMWare) dm server, Red Hat's JBoss
Getting packets: Click router

- Runs as a Linux kernel module or user-level program
- Modules written in C++ (called *Elements*) are configured in a text file
- Elements are arranged in a directed graph, through which packets traverse
- Example:
  - Click router command:
    ```
    sudo click print.click
    ```
  - Configuration file print.click:
    ```
    FromDevice(en0)->CheckIPHeader(14)->IPPrint->Discard;
    ```
- [http://www.read.cs.ucla.edu/click/](http://www.read.cs.ucla.edu/click/)
1st prototype

- Proof-of-concept for dynamic network service deployment
  - Open-source Click modular router
  - Java OSGi dynamic module system
- Promising initial measurement results
  - NetServ overhead acceptable compared to other overhead
1st implementation

- Implements PktProcessor
- Registers an instance of PktDispatchingService

- NetServ App Bundle
- NetServ Building Block Bundle
- Equinox OSGi framework
- NetServ OSGi Launcher
- Java Virtual Machine
- StaticIPLookup element
- CheckIPHeader element
- User-level Click router
- Single process

```
dispatcher.addPktProcessor(this);
```

packet flow
Performance evaluation

- Initial measurements on the first prototype
  - NetServ on user-level Click router
  - Maximum Loss Free Forward Rate (MLFFR)
- Future work on next-generation prototypes
  - NetServ on JUNOS, kernel-mode Click
  - Ping latency
  - Microbenchmarks
  - Throughput for non-trivial services
For a modular architecture, kernel-user transition is unavoidable since putting a module inside a kernel is not an option.
Current NetServ node architecture

- NSLP daemon
- GIST daemon
- NetServ Controller
- OSGi
- Transport layer
- Linux kernel
- Netfilter
- Packet processing modules
- Server modules
- Client-Server data packets
- Forwarded data packets
- Signaling packets

- Raw socket
- UNIX socket
- iptables command
- NetServ Control Protocol (TCP)
- OSGi control sockets
- NFQUEUE #1
- NFQUEUE #2
- OSGi control sockets
Next Click architecture

Signaling

NetServ Controller Daemon
• Start & stop NetServ Service Container
• Module install & removal

Service modules for
user #1

Building Blocks
OSGi, Java 2 Security
OpenVZ container

Service modules for
user #2

Building Blocks
OSGi, Java 2 Security
OpenVZ container

Service modules for
anonymous users

Flow-based multiplexing
layer
Building Blocks
OSGi, Java 2 Security
OpenVZ container

/dev/fromclick1...N

Kernel-mode Click

PollDevice element
Other Click elements
NetServ Packet Filter
NetServ Packet Injector
Other Click elements
ToDevice element
NetServ controller and kernel

- **NetServ kernel**
  - Pass packets to user-level service container processes
  - Currently Linux kernel with Netfilter queues
  - Click Router version under development
  - Juniper Router version planned

- **NetServ controller**
  - Coordinates between signaling daemons, kernel, and service container processes
  - Receives setup/remove/probe commands from signaling daemons
  - Insert/remove packet filters into/from the kernel
  - Start/stop service modules in service container processes
Packet intercept

- CLICK
  - more functionality than needed
- pcap
  - capture only
- iptables with NFQUEUE
  - allows filtering of packets into different queues
APIs needed!

- Avoid SNMP retrieval problems
  - all or nothing (typical)
  - hard to do selective triggers
- Flow management
  - counters, measurement
- System information
  - like system MIB: geo location, uptime, interface speeds, ...
  - routing table
  - routing table changes ("tell me if route to X changes")
NetServ service (demo) examples
Media relay

- Standard media relay
  - Required due to NAT
  - Out-of-path
  - Inefficient and Costly

- NetServ media relay
  - Closer to users
  - Improved call quality
  - Reduced cost for ITSP
Media relay

Tell:
How VoIP calls are relayed today
- Media relay needed due to NAT
- But relays are often out-of-path
- Inefficient and high cost for ITSP

Show:
1. Make a video call between screens 1 & 3
2. Call flow displayed on screen 2
3. Bad call quality due to far-away relay
Media relay

Tell:
- Media relay running in NetServ
  - Closer to user agents
  - Higher call quality for users
  - Reduced cost for ITSP

Show:
1. SIP server sends signal to install module
2. Screen 2 displays module being installed (blue arrow)
3. Make a new video call
4. Call flow through NetServ relay displayed on screen 2
5. Screen 1 & 3 show much better video quality
Media Relay Signaling

1. UA registers with SIP server
2. SIP server detects NAT:
   • Sends NSIS signal towards UA
   • Netserv node close to UA is discovered
   • Netserv node installs TURN module
   • SIP server remembers to use the TURN module for the UA
3. An INVITE is received for UA
4. SIP server sends relaying request to the TURN server
5. INVITE with updated SDP is sent to UA

NetServ based TURN module is installed at registration time to speed-up processing of INVITE signaling.
NAT Keep-alive Responder

- NAT Keep-alive responder off
  - UA behind NAT must send keep-alive messages
  - Major bottleneck for SIP server
- NAT Keep-alive responder on
  - Module responds on behalf of SIP server
  - No traffic to server
Tell:
- UA behind NAT must keep sending keep-alive message
- Major bottleneck on SIP server today

Show:
- Screen 1: Traffic generators send keep-alive messages to SIP server
- Screen 2: Packet flows shown on screen
- Screen 3: High traffic volume, thus high cost for ITSP
**Tell:**
- SIP server installs modules into NetServ nodes closer to users
- Modules respond to keep-alive messages on behalf of SIP server
- Keep-alive messages do not reach SIP server

**Show:**
- **Screen 1:** Traffic generators keep sending keep-alive messages to the SIP server.
- **Screen 2:** Packet flows terminate at NetServ nodes.
- **Screen 3:** SIP server sees no keep-alive traffic, thus no keep-alive cost to ITSP.

No keep-alive traffic

$0.00$
(1) User requests http://youtube.com/getvideo?id=foo

(2) YouTube sends video file

(3) YouTube sends on-path signal to deploy MicroCDN module

(4) NetServ-enabled routers download the module

(5) NetServ routers notify that the module is active

(6) Another user requests http://youtube.com/getvideo?id=foo

(7) YouTube redirects user to nearest NetServ node running MicroCDN

(8) User requests http://netserv1.verizon.com/youtube/foo.flv

(9) NetServ router relays the video content, while fetching the file and caching it
MicroCDN – watermarking

**Step 1**
User 1 downloads and watches video content from provider.

**Step 2**
Content server sends on-path signal to install MicroCDN module into NetServ router.

**Step 3**
User 2’s request gets redirected to NetServ router, which serves processed video.
MicroCDN – module migration

Tell:
• PlanetLab nodes run scripts fetching small files from the web server
• MicroCDN modules with short TTL keep getting installed and removed
• PlanetLab scripts are choreographed to make modules migrate westward!

Step 1
Screen 2: NetServ nodes popping up in the eastern region
Screen 3: Web traffic generated from eastern PlanetLab nodes

Step 2
Screen 1: Web traffic generated from western PlanetLab nodes
Screen 2: (Flickering) NetServ nodes migrating to the wild west!

This is a rough idea. We are still working out the details to see if this can be done by GEC9.
Looking ahead
Future: NetServ everywhere

- From big to small devices
- Real router: Juniper’s JUNOS
- Personal computer: Kernel-mode Click
- Home router: Linux using iptables
- Security and resource control
  - Enable various deployment scenarios
  - Support different economic incentives
The network services fallacy

- We tried adding network services as protocols:
  - multicast
  - QoS
  - mobility
  - security
- All were, more-or-less, failures
  - (or underperformed expectations)
  - hard to secure, not quite right
Thoughts on architecture

- Long-term constant: service model
  - equivalent of railroad track & road width
- Identify core functions we need
  - routing
  - congestion control
  - name lookup
  - path state establishment
  - ...
- Learn from history
  - why didn’t these get done “right”?
- Need engineering principles
- Requirement list doesn’t help
Future Internet architecture?

- Really closer to urban design
  - zoning, fire codes and infrastructure (rail, water)
    - plus oversight (fire marshal & building inspector)
  - architecture changes, urban designs stay
    - see Washington, DC & Berlin

- “Architecture” must be
  - expressible in one sentence
  - avoid limiting options (unknown unknowns)
  - avoid imposing unnecessary costs
Conclusion

NetServ
- rapid on-path services, with storage
- moderate overhead

(GENI) VMs
- Long-term
- Full access, but not composable

OpenFlow
- rule-based packet handling
- can be controlled by NetServ

NetFPGA
- mainly packet processing
- fastest, least flexible