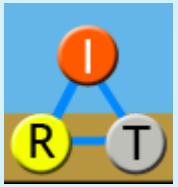
NetServ: Dynamically Deploying In-network Services

Suman Srinivasan⁞, Jae Woo Lee⁞, Eric Liu⁞, Mike Kester⁞, Henning Schulzrinne⁞, Volker Hilt⁞, Srini Seetharaman †, Ashiq Khan ‡

⁞*Columbia University,* ⁞*Bell Labs,* †*Deutsche Telekom R&D Lab,* ‡*DOCOMO Labs Europe*



NetServ overview

Extensible architecture for core network services

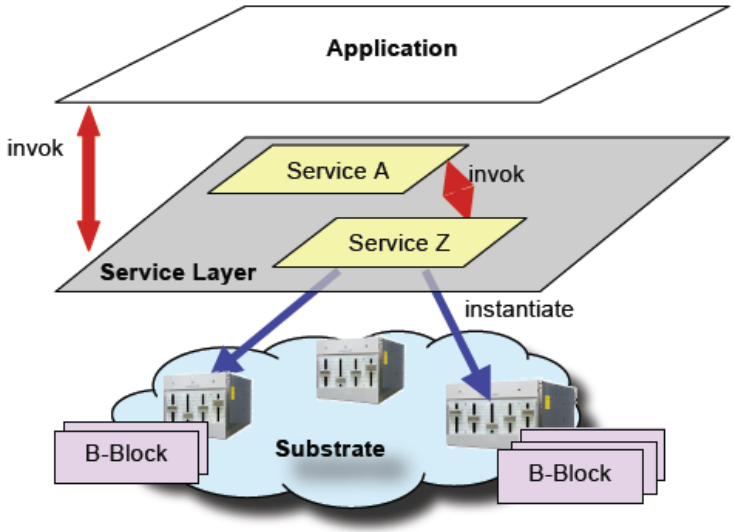


Figure 3: Instantiation of services over tunable building blocks.

Modularization

- Building Blocks
- Service Modules

Virtual services framework

- Security
- Portability

NSF FIND four-year project

- Columbia University
- Bell Labs
- Deutsche Telekom
- DOCOMO Euro-Labs

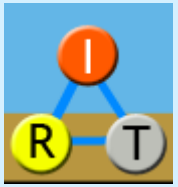
No more *ossification* in NGI

Different from Active Networks?

- **Active Networks**
  - Packet contains executable code
    - Can modify router states and behavior
  - Not successful
    - Per-packet processing too expensive
    - Security concerns
  - Notable work: ANTS, Janos, Switchware
- **NetServ**
  - Virtualized services on current, passive networks
    - Service invocation is signaling driven, not packet driven
  - Service modules are stand-alone, addressable entities
    - Separate from packet forwarding plane
    - Extensible plug-in architecture

Building Blocks

- Key components of network services
  - Access to network-level resource
  - Implementation of common functionality
- For example:
  - Link monitoring and measurement
  - Routing table
  - Packet capture
  - Data storage and lookup

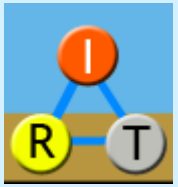


Service Modules

- 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

First prototype implementation

- 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



Technology: Click router

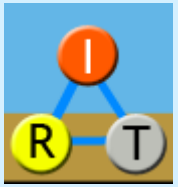
- Runs as a Linux kernel module or user-level program
- Modules written in C++ (called *Elements*) are configured in a text file
- Elements are arranged in a directed graph, through which packets traverse
- Example:
  - Click router command:

    ```
    sudo click print.click
    ```

  - Configuration file print.click:

    ```
    FromDevice(en0)->CheckIPHeader(14)->IPPrint->Discard;
    ```

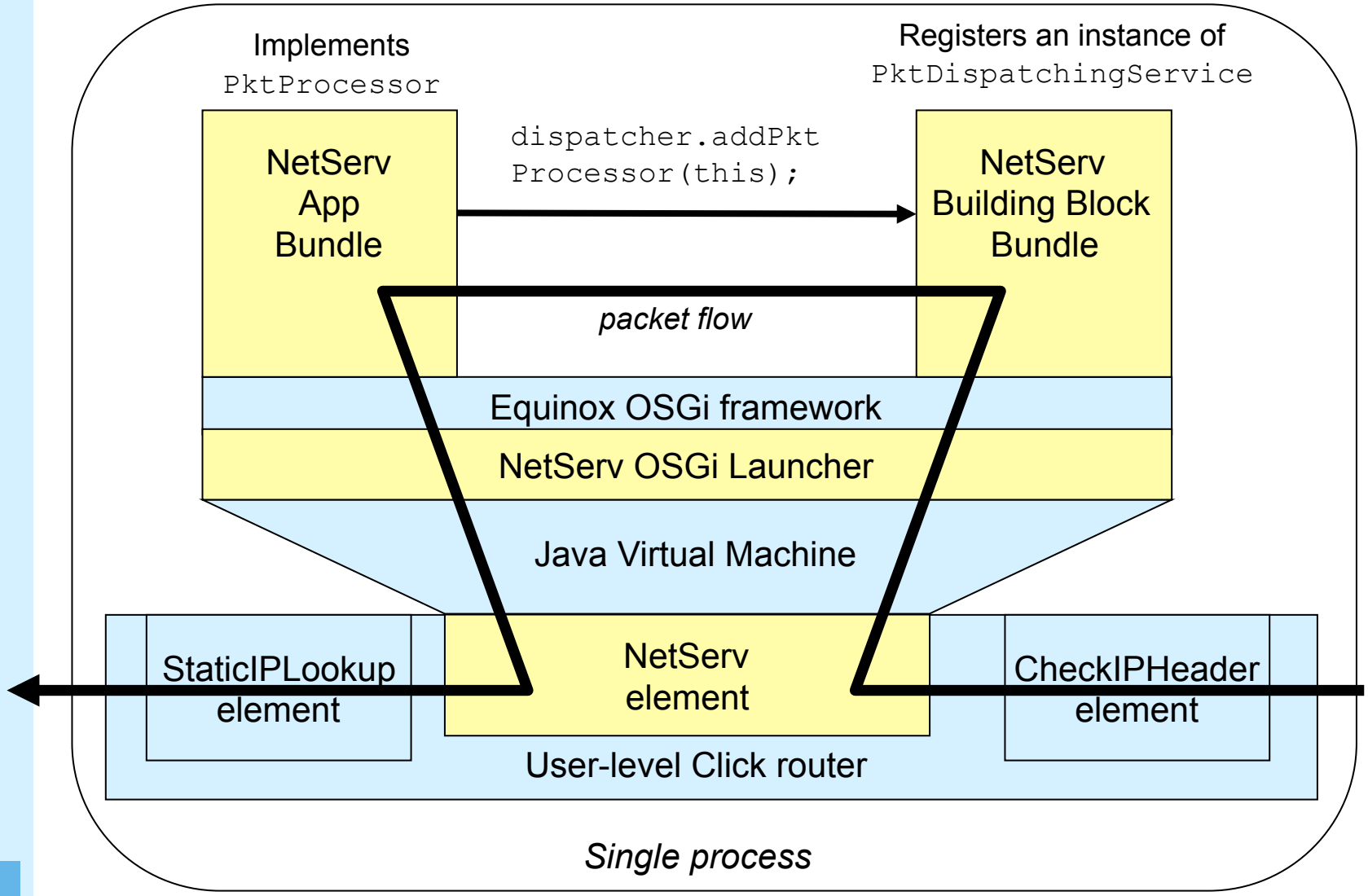
- http://www.read.cs.ucla.edu/click/



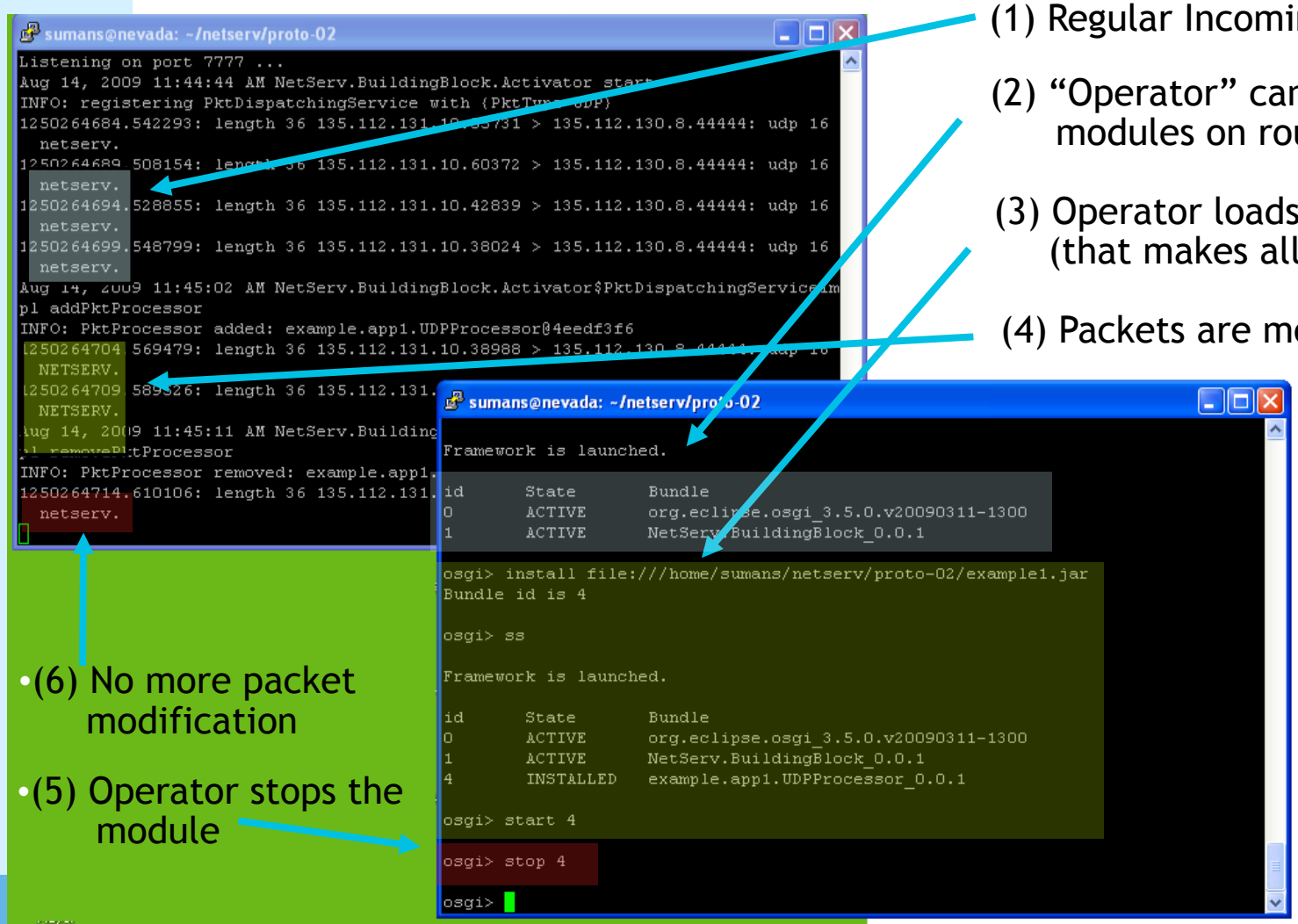
Technology: OSGi

- 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

1st prototype implementation



Demo: NetServ prototype



(1) Regular Incoming packets

(2) "Operator" can view modules on router

(3) Operator loads a new module (that makes all data uppercase)

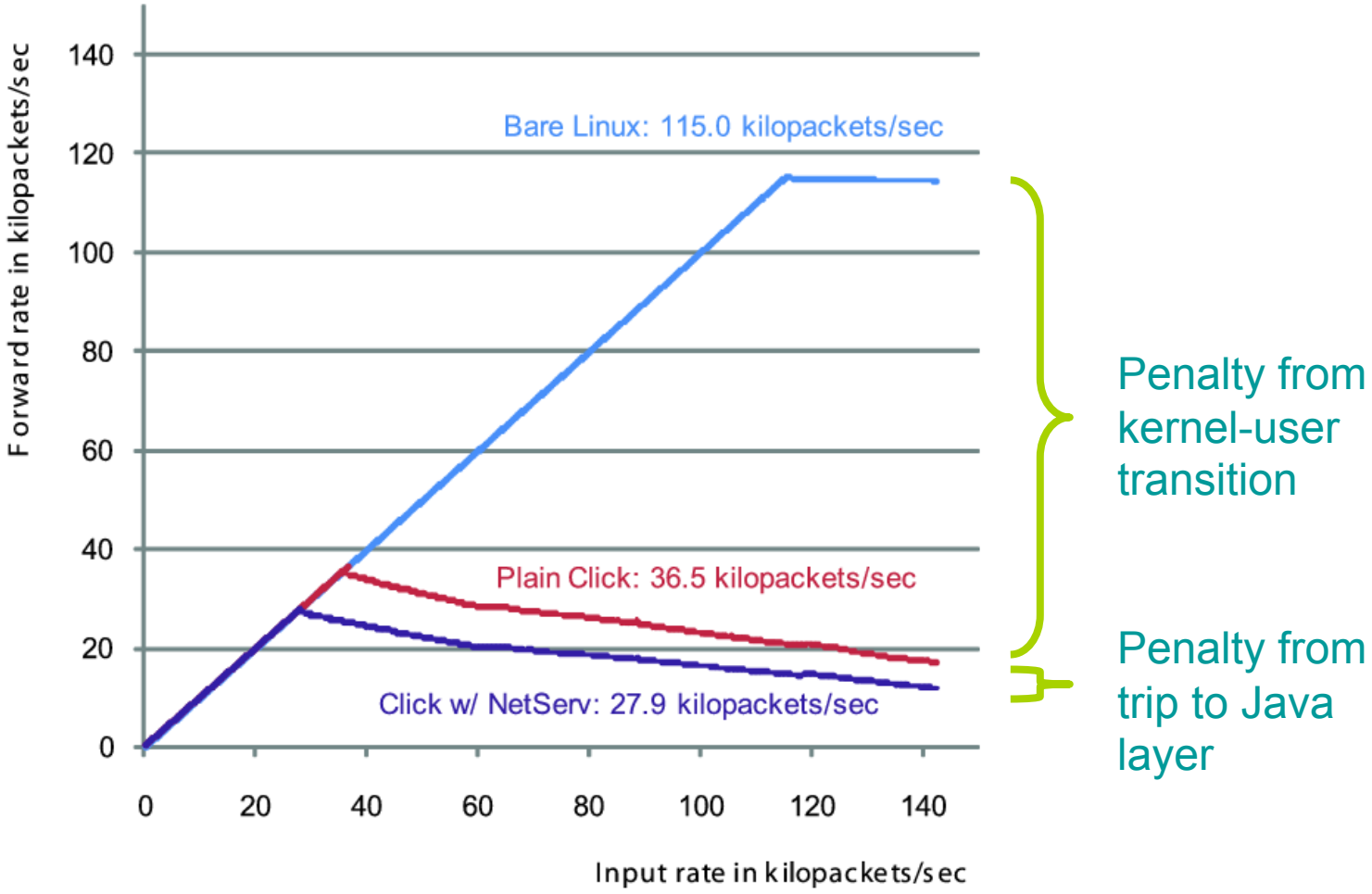
(4) Packets are modified

- (6) No more packet modification
- (5) Operator stops the module

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

MLFFR Comparison



Penalty from Java/OSGi overhead is extremely small compared to kernel-user transition.

NetServ Deployment Scenarios

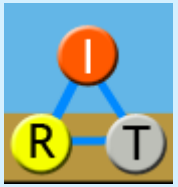
- CDN application scenario with publisher/provider
- Three actors
  - Content publisher (e.g. youtube.com)
  - Service provider (e.g. ISP)
  - End user
- **Model 1: Publisher-initiated deployment**
  - Publisher rents router space from providers
- **Model 2: Provider-initiated deployment**
  - Publisher 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

Current Work: CDN on NetServ

- On-Path CDN
  - Prototype implemented during summer 2009 at Bell Labs
- Dynamic content migration
  - Moving content closer to the end user according to demand
- Building blocks
  - Network monitoring
  - Content discovery
  - Caching proxy

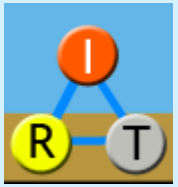
Current Work: NetServ Platform

- Ubiquitous NetServ
  - 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



Related Work

- Cisco's Programmable Overlay Router
- Juniper's JUNOS SDK
- DaVinci project
- VROOM (virtual routers on the move)
- OpenFlow Switch
- Ethane



Summary

- NetServ: architecture for dynamic in-network service deployment
- Modular and extensible
  - Building Blocks and Service Modules
  - Virtualized Services Framework
  - Supports various deployment scenarios
- Prototype implementation: Click and OSGi
- Initial measurements and analysis
- CDN application under development
- www.cs.columbia.edu/irt/project/netserv/