Sviluppo di un'architettura efficiente ed estensibile per fornire servizi di rete evoluti per l'internet di prossima generazione

Borsista

Francesco Giacinti

Tutor

Prof. Gianluca Reali

5° Borsisti Day *– 13/05/2014*

GARR

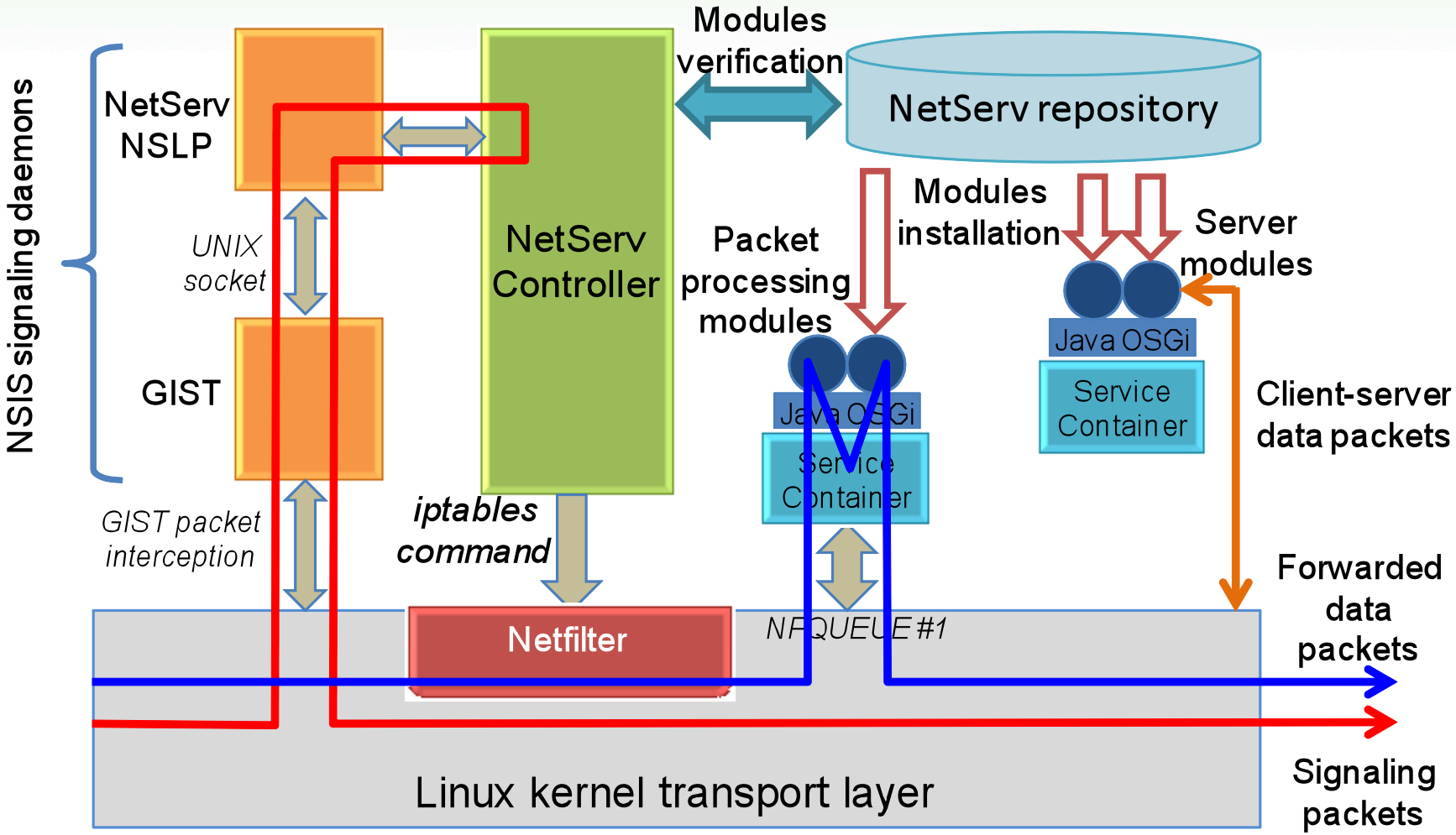
Summary

- NetServ Architecture
- NetServ Signaling
- Integration of Off-path Extension
- CDN (Content Delivery Network) Service
- Porting to Juniper Routers
- Realization of an equivalent router by using OpenFlow
- Activity Phases
- Conclusion and Final Remarks

What is NetServ?

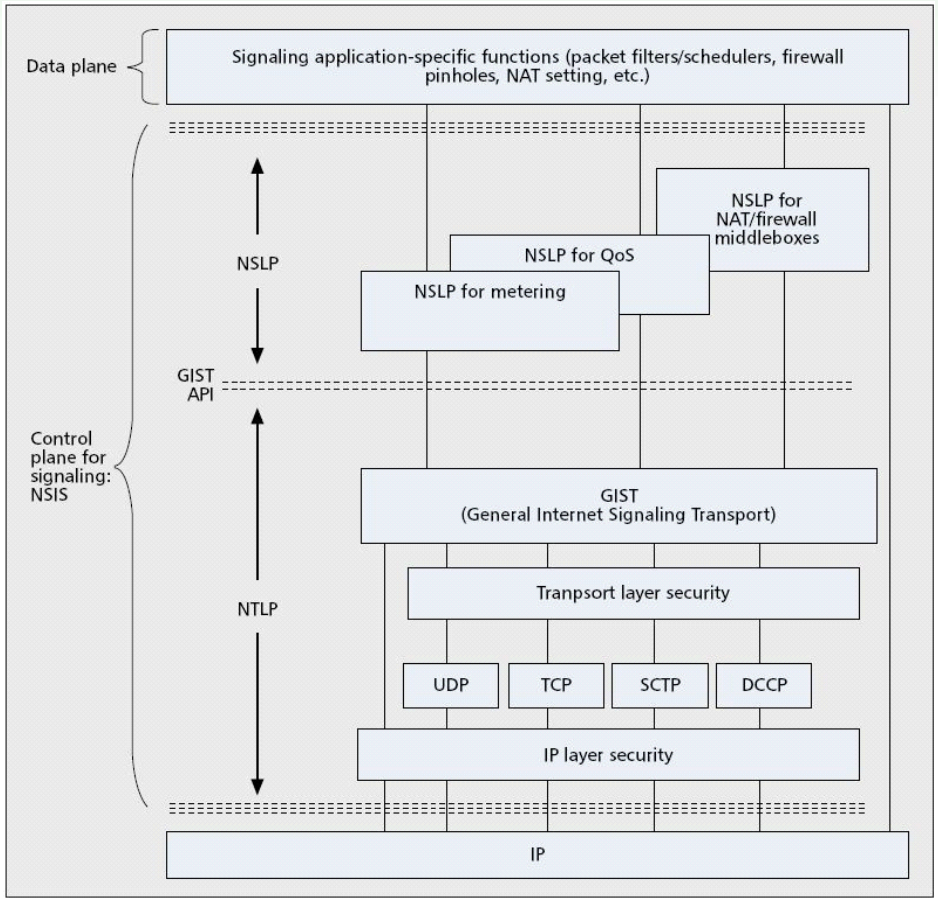
- NSF FIND project led by Columbia University (NY) with Deutsche Telecom – DOCOMO – Bell Labs and **UniPG**!!
- In-network service container
- Java-programmable, signal-driven router
- Resources isolation and virtualization
- NSIS signaling support
- Hot service deployment and removal
  - Service installed only when needed
- Only assumes that a node provides packet transport
  - Future Internet highly evolvable, and decouples services from a specific technology
  - NetServ everywhere from routers to end systems (users and servers)

NetServ Node Architecture



NetServ Signaling 1/3

- NSIS signaling
  - suite of protocols envisioned to support various signaling application
  - IETF RFC 4080
- Two layers:
  - NTLP: NSIS Transport Layer Protocol
    - GIST (Generic Internet Signaling Transport)
  - NSLP: NSIS Signaling Layer Protocol
    - NetServ-specific NSLP
      - On-path based signaling
      - Three messages:
        - » **SETUP + ACK**
        - » **PROBE REQUEST/RESPONSE**
        - » **REMOVE + ACK**



- **Two requirements to be met:**
  - ❑ **Installation, modification and removal of service modules at runtime;**
  - ❑ **Allow the creation of complex logic for services distribution, parameters management, etc.**

NetServ Signaling 2/3

- NetServ protocol provides:
  - Installation and removal of service modules
  - NetServ network probing features, such as:
    - **State of nodes (Topology Discovery)**
    - **State of services (Service Discovery)**
    - **Logs and errors collection for service/ node management system**

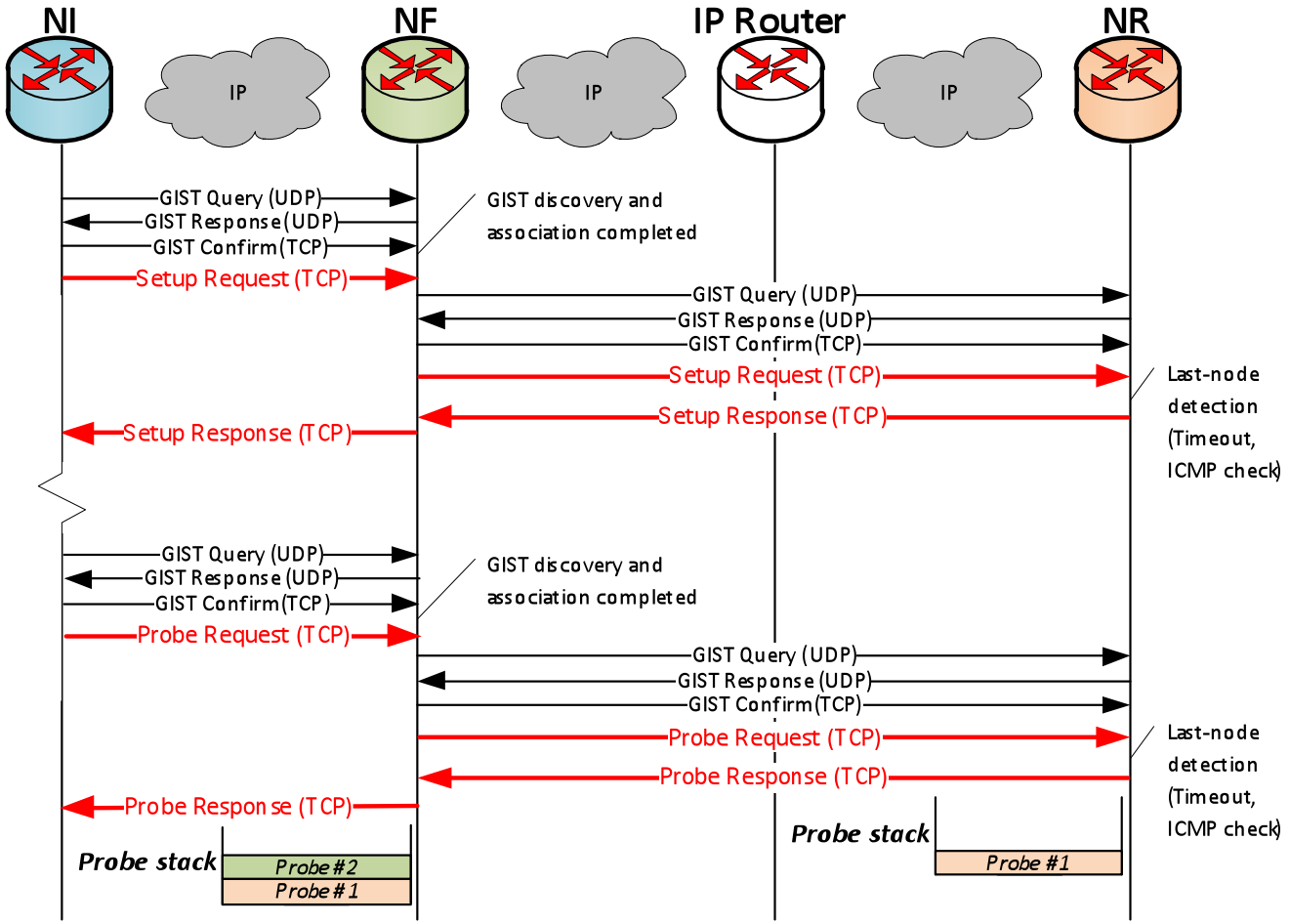
```
SETUP NetServ.apps.application_name NETSERV/0.1
dependencies:
filter-port:5060
filter-proto:udp
notification:
properties:visualizer_ip=1.2.3.4,visualizer_port=5678
ttl:3600
user:user_name
url:http://content-publisher.com/modules/module_name.jar
signature:4Z+HvDEm2WhHJrg9UKovwmMutxGibsA71FTMFykVa0Y\xGcIG8o=
<blank line>
```

Parameter	Message	Description
dependencies	S	List of bundle dependencies necessary to run the bundle
filter-port	S	Destination port for the packet that will be intercepted by the bundle
filter-proto	S	Type of protocol for the packet that will be intercepted by the bundle
notification	S, R	XML-RPC URL invoked after the setup of the module
node-id	S, R, P	Address of the explicit destination of a message (end-to-end signaling)
probe	P*	Code of the probe request
properties	S	List of key=value properties that are passed to the bundle
ttl	S*	Time to live of the bundle (s)
signature	S*, R*, P*	Digital signature of the message
user	S*, R*, P*	The user owning the NetServ Container

Setup (S), Remove(R), Probe (P), Mandatory(*)

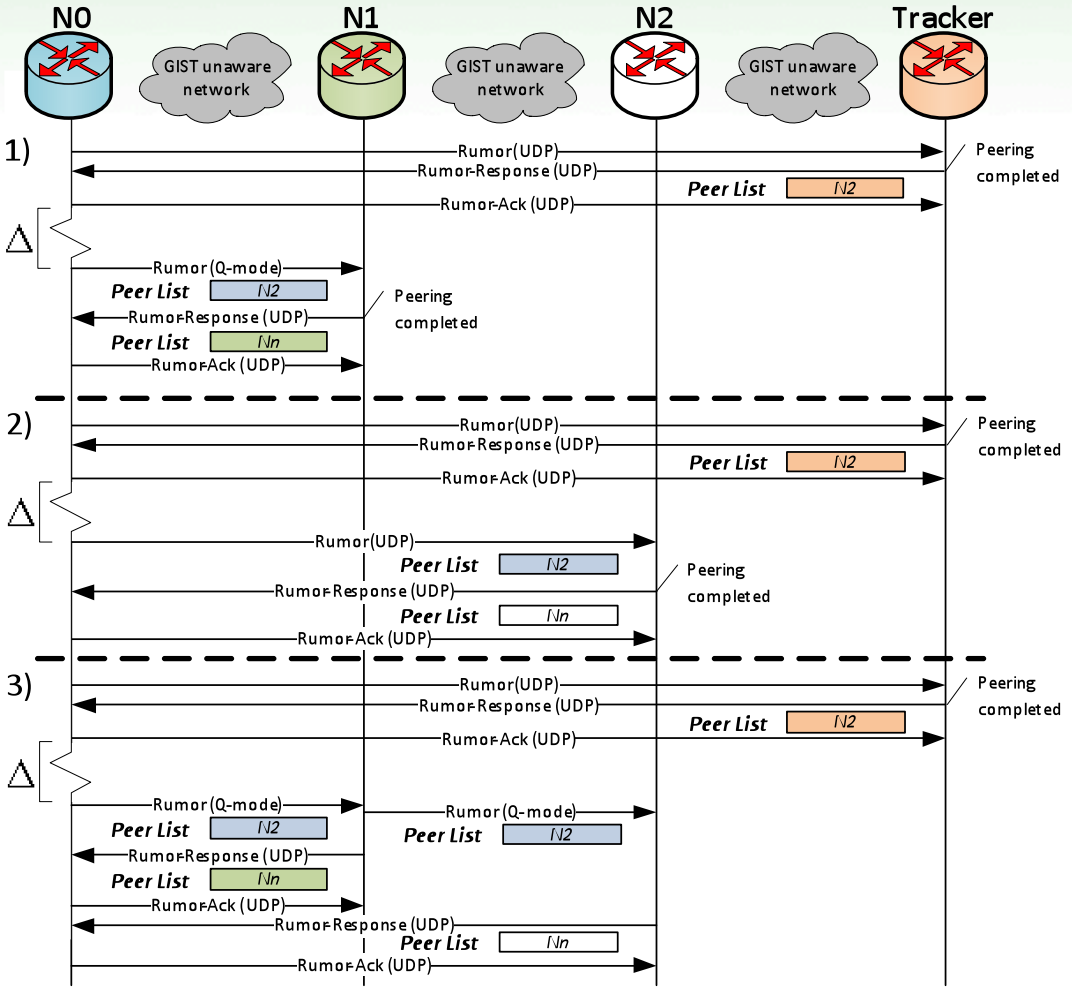
NetServ Signaling 3/3

- NetServ protocol on top of GIST that provides hop-by-hop node discovery, peer association, message transport and security
- Only NSIS nodes with a running NetServ NSLP will process the protocol messages, IP nodes forward the packets transparently



NSIS Off-path Extension

- Applications may need different dissemination techniques
- Off-path goes beyond traditional on-path delivery
- Need for a discovery mechanism
  - Gossip*-based with Trackers
  - 3 models compared

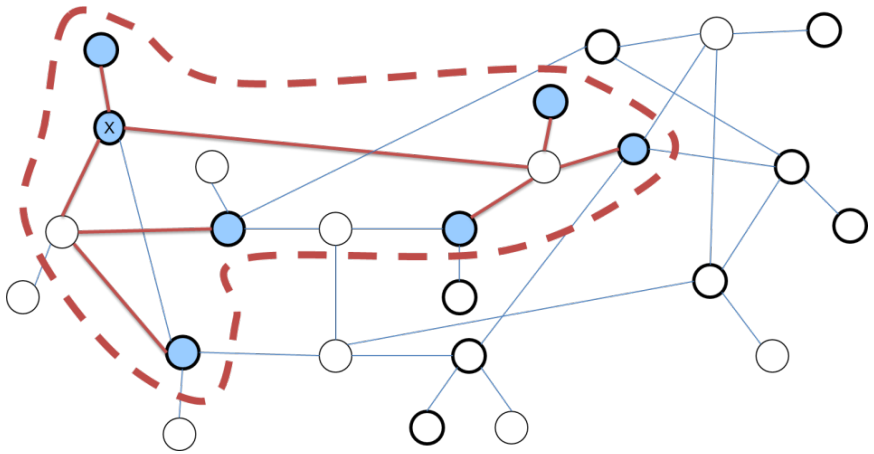


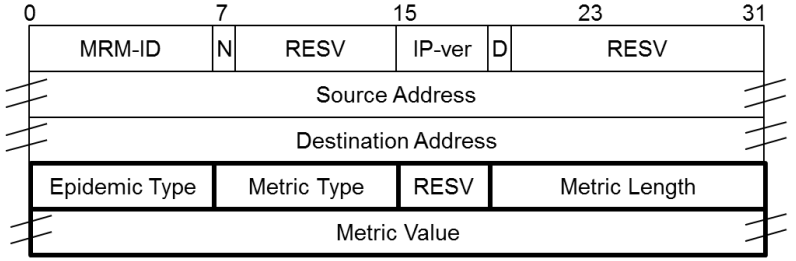
[*] A. Demers, D. Greene et al., “Epidemic Algorithms for Replicated Database Maintenance”, In *Proceedings of the Sixth Symposium on Principles of Distributed Computing*, August 1987.

Off-path Dissemination Mode

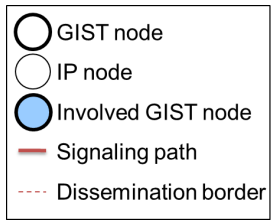
- **Bubble, Balloon, Hose options**
- **Metrics supported are: IP TTL, GIST TTL, Latency**

a) Bubble centered in X with radius = 1 GIST hop

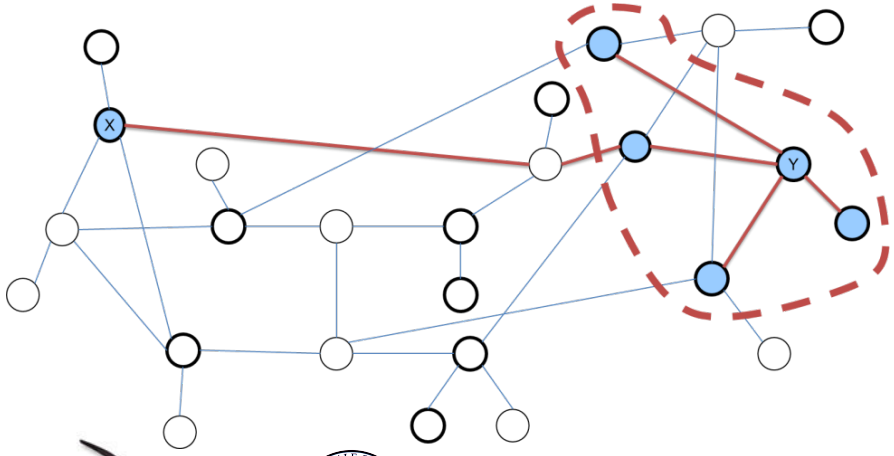




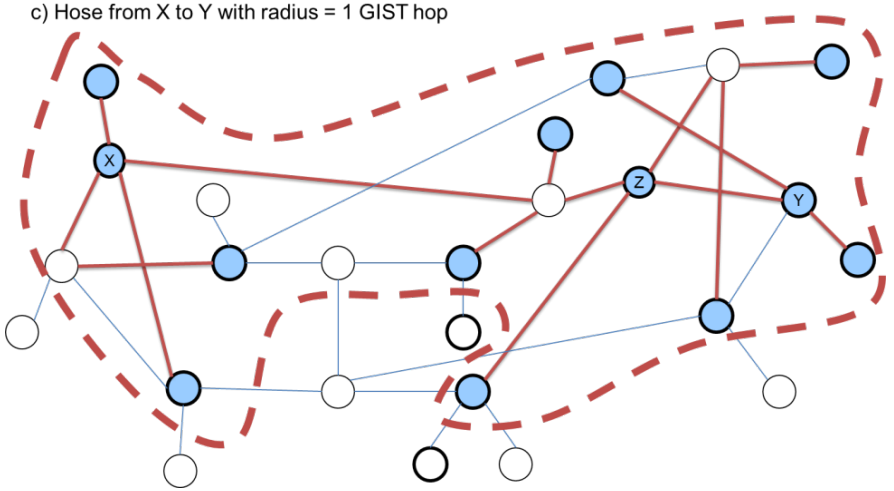
New Epidemic MRM TLV



b) Balloon started in X and centered in Y with radius = 1 GIST hop



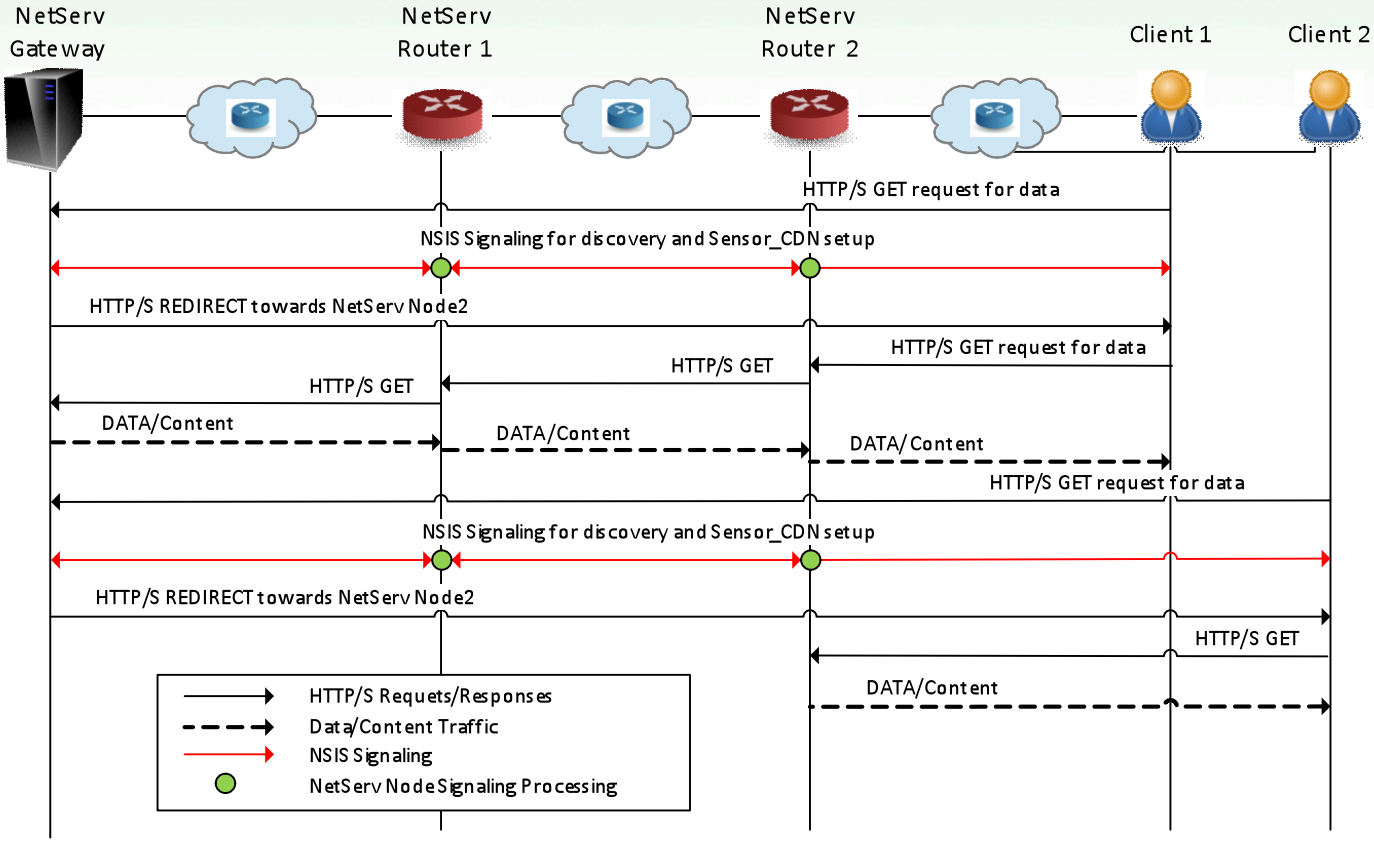
c) Hose from X to Y with radius = 1 GIST hop



CDN Service 1/2

- Aim: store data in Gateways and distribute them to requesting applications
- Real wireless sensors motes sending data to a requesting Gateway
- A single Gateway provides two sets of functions (implemented in a NetServ bundle):
  - Requesting data from sensor motes and storing them in a DB (SQLite) by using the interface with the sensor network
  - Distributing data to applications (clients) requesting them by using the interface with Internet
    - RESTful interface by using Jetty
    - Sensor data payload formatted according to JSON model
- CDN-like application avoiding preloading data
- NetServ bundle implementing CDN functions
- The bundle is stored in the Gateway
- CDN bundle use SQLite and Jetty to:
  - Store data coming from an upstream cache into DB
  - Sending data towards downstream cache/client

CDN Service 2/2



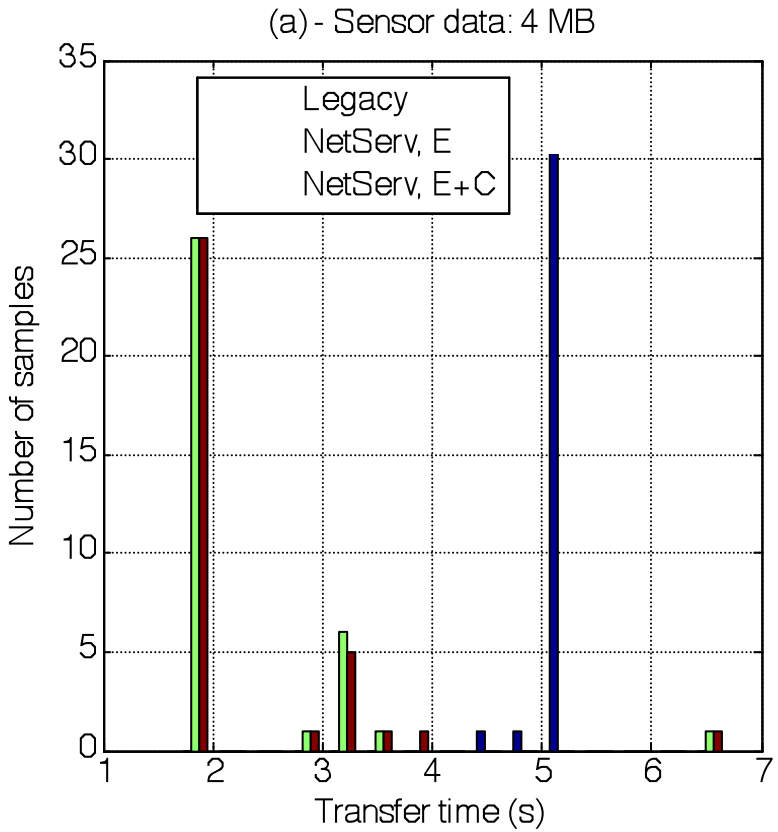
Testbed

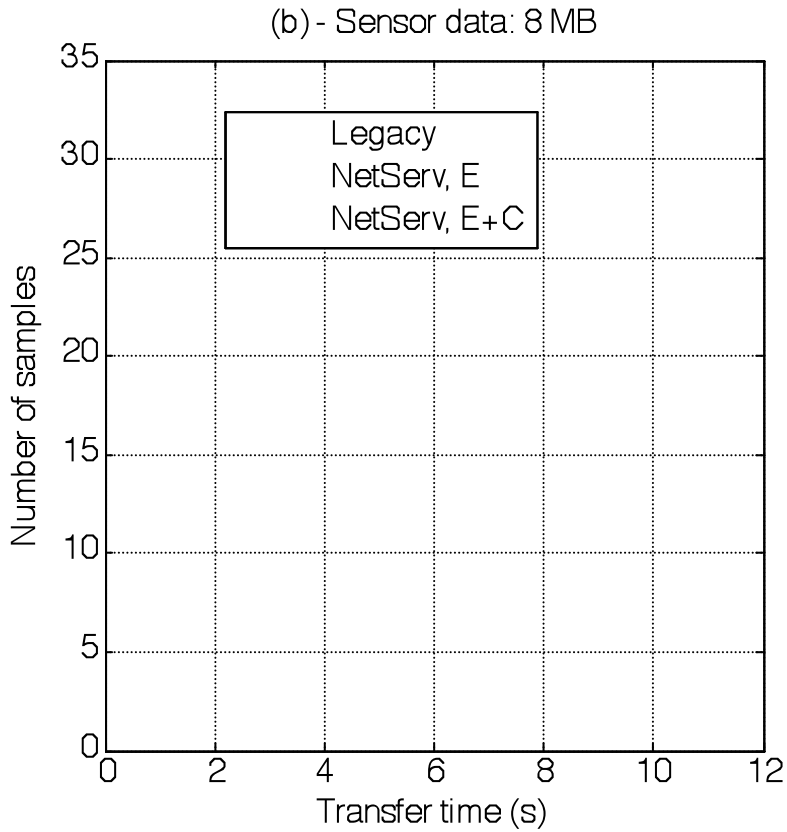
- 60 nodes
  - Physical or VM
- 11 Core nodes
  - 512MB RAM
  - 1 or 2 CPU
- 12 Edge nodes
  - 512MB RAM
  - 1 virtual CPU
- 37 End-nodes
  - 374MB RAM
  - 1 virtual CPU

- 1 end-node is used as control node.
- 1 end-node implements the Gateway (data retrieving and distributing)
- 35 end-nodes represent the client requesting data
- Gateway towards the network is limited to 10 Mb/s (emulating a 3G+/4G cellular connection)

On-path test evaluation 1/3

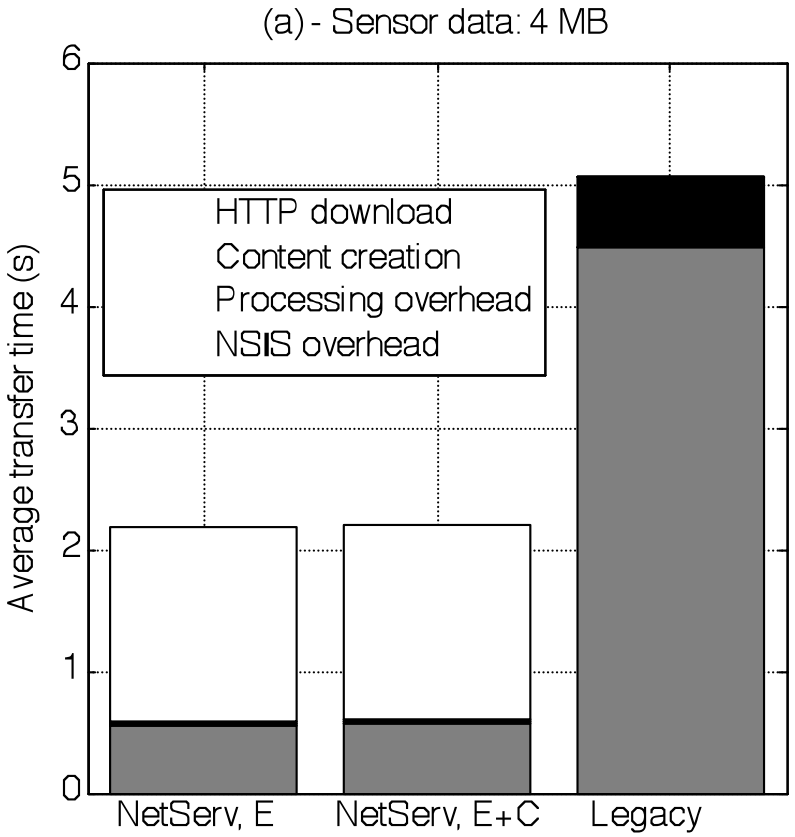
Download time distribution for different solutions

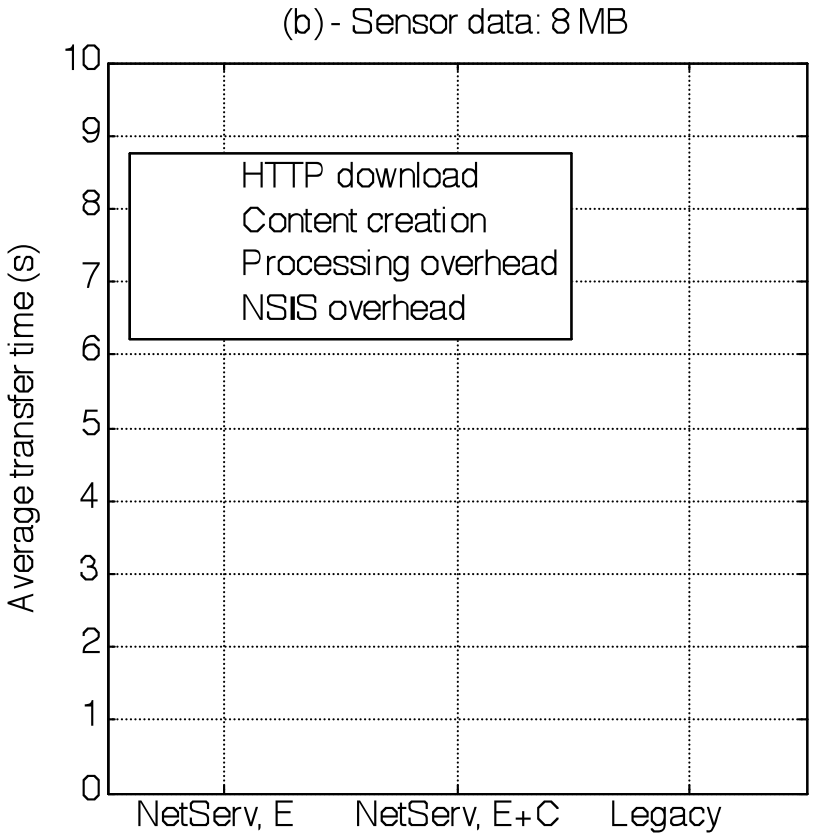




On-path test evaluation 2/3

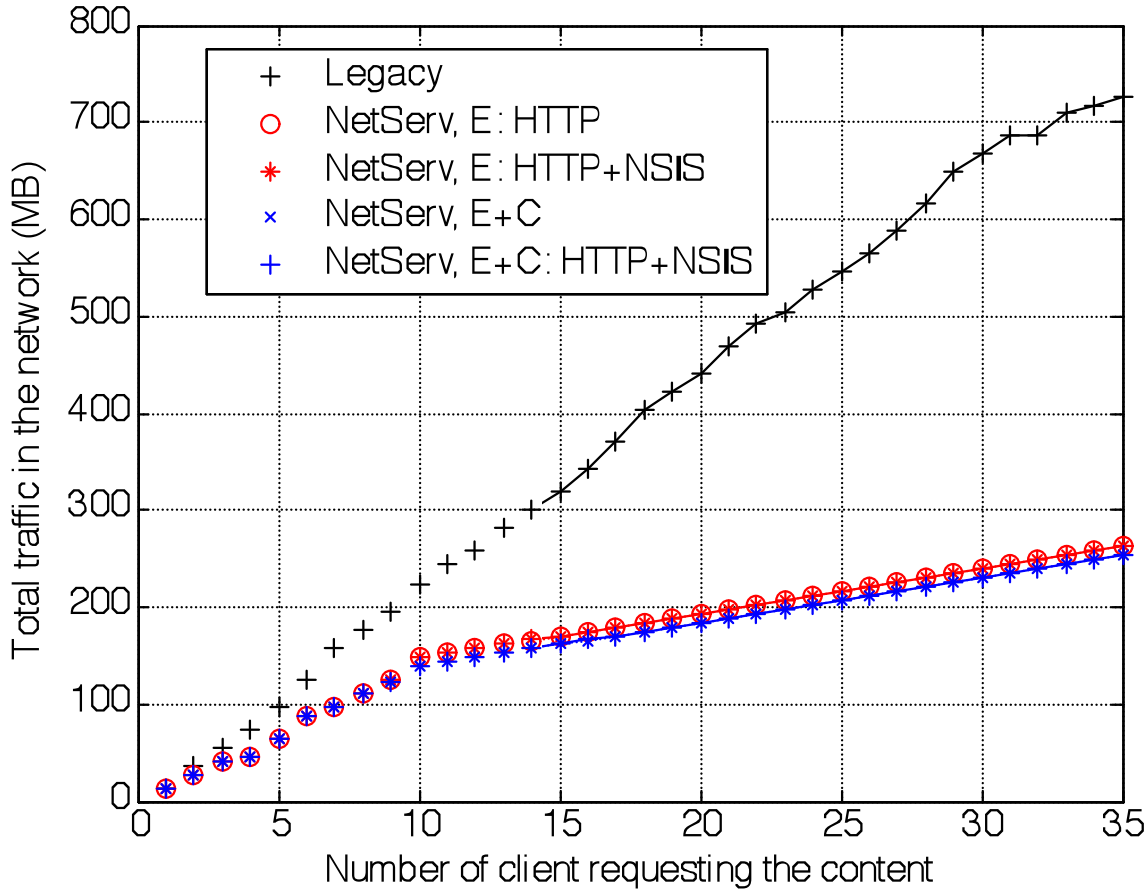
Average download time (individual delay components)





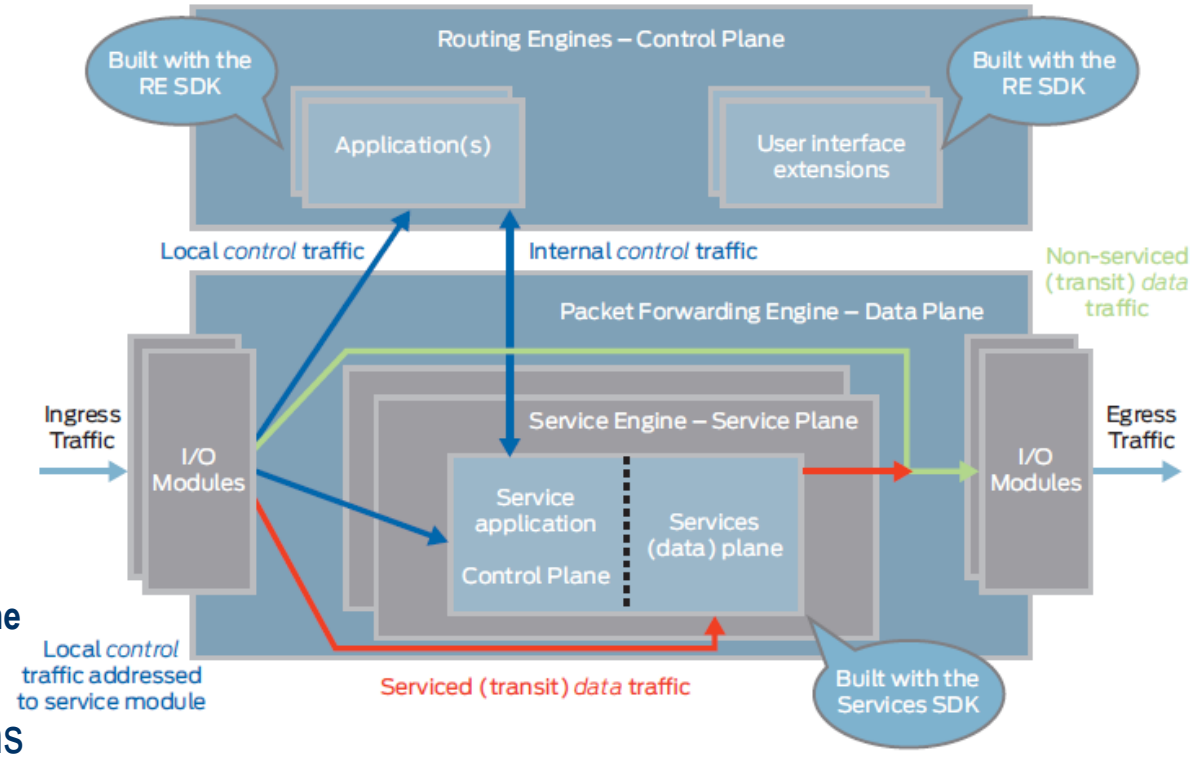
On-path test evaluation 3/3

Network traffic vs. number of clients (Data size 4MB)



Porting to Junos OS 1/2

- Junos OS architecture:
  - Single network operating system
  - Integrating routing, switching, and security
  - Based on FreeBSD version 7.1
  - Modular architecture
  - Logical division into 3 elements:
    - **Routing Engine (RE) Control Plane;**
    - **Packet Forwarding Engine (PFE) – Data Plane**
    - **Service Engine (SE) – Service Plane**
  - Junos SDK for developing applications



Troubles using the Junos SDK:

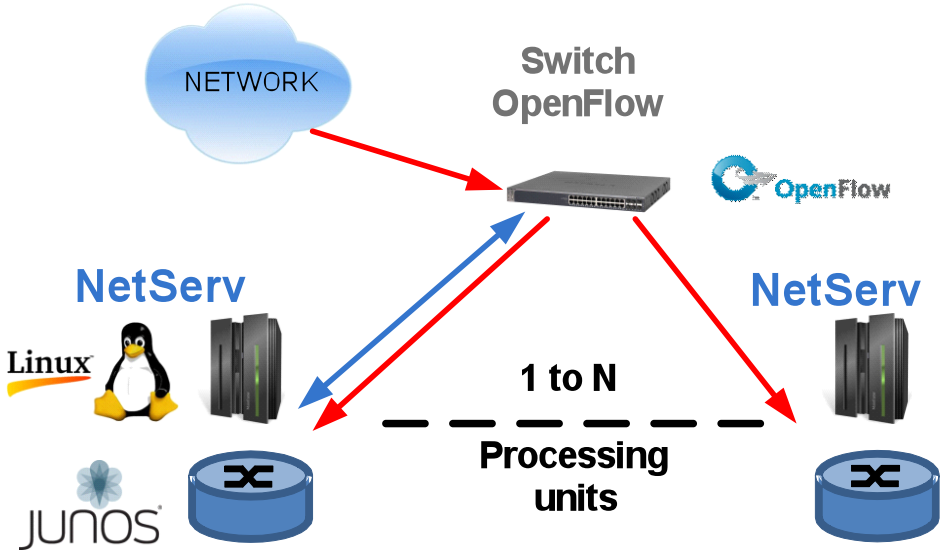
- Limited support from Juniper
- SDK documentation huge and not well organized
- A lot of time required to understand how to develop whole applications

Porting to Junos OS 2/2

- First results:
  - found a replicable way to install and run Java Virtual Machine in Junos OS
  - installation of Oracle JAVA JDK version 6u45 in Junos OS
  - NetServ OSGI container and NetServ service bundles able to run in Junos OS
- Work in progress:
  - porting the NSIS signaling daemons inside Junos OS
  - porting the NetServ Controller inside Junos OS
  - allow NetServ service bundle to automatically add, modify or delete packet routing rules inside the Juniper router
- Junos OS porting benefits:
  - whole NetServ architecture will be able to run also in the common programmable Juniper routers/switches
  - easier and faster in-network service design and implementation
    - **using JAVA programming language instead of proprietary Junos SDK developing system**
    - **“write ones, run everywhere”**
  - creation of a detailed tutorial/report reporting all necessary and organized steps to develop whole application by using Junos SDK

Equivalent Router (OpenFlow) 1/2

- Realization of an equivalent router: server/pc/ with NetServ + switch OpenFlow
- Allows to scale the service capability according to the SDN (Software-Defined Networking) paradigm
- Given a hardware data plane will be possible to add from 1 to N processing units (NetServ) able to elaborate the data traffic
- Service load split among various units
- Due to the simple way to install NetServ services/bundles, the SDN-NetServ paradigm will be more appealing than a paradigm that makes use of simple virtual machines (whose overhead is greater)



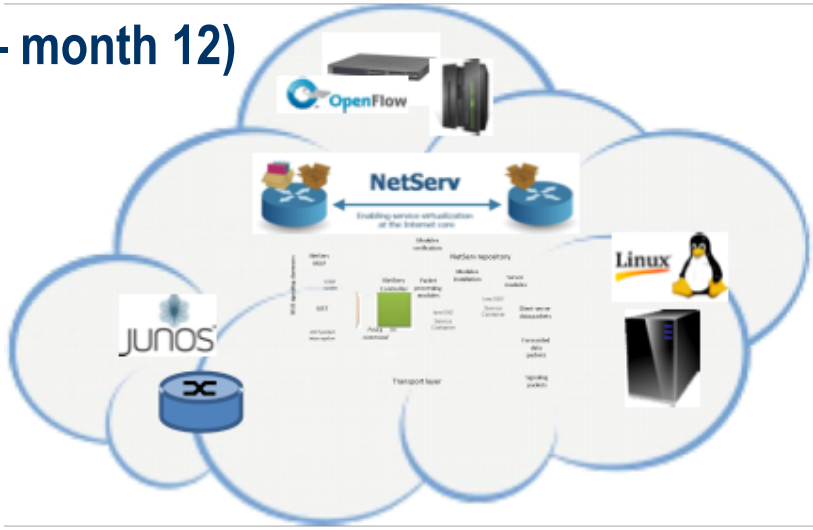
Equivalent Router (OpenFlow) 2/2

- When a new NetServ bundle is installed in a equivalent node, the NetServ Controller must perform two operations:
  - Specify the rules to forward the packets towards its relevant bundle/s
  - Inform the OpenFlow controller to install the rules allowing the packets to reach the NetServ processing units
- The NetServ Controller will exploit the communication API of the OpenFlow Controller
- For example, OpenDaylight platform offer a northbound interface with RESTful API
- The more interesting aspects of this approaching is that a service/bundle could be installed and run without any modification on:
  - Servers/PCs Linux that act as software routers;
  - Programmable Juniper routers;
  - Equivalent SDN nodes/routers with OpenFlow data plane and NetServ processing unit

- Future work
  - Integration of Netmap framework inside the NetServ architecture to allow the developing of packet-processing services

Activity Phases

- **Phase 1 - Porting to Juniper routers through Junos SDK (Month 1 – Month 4)**
- **Phase 2 - Realization of an equivalent router: server/pc with NetServ + switch OpenFlow (Month 5 – Month 8)**
- **Phase 3 - Extensive test campaign of implemented service over proposed architectures, for evaluation of: (month 9 – month 12)**
  - **Network traffic saving;**
  - **Signaling overhead;**
  - **Reduction of contents delivery time;**
  - **Scalability;**
  - **Nodes resource usage (in particular, CPU and memory usage).**



Thank you!