POLITECNICO DI TORINO 1859 1906

unifying cloud
and carrier networks

Supporting Fine-Grained Network Functions through Intel DPDK

Ivano Cerrato, Mauro Annarumma, Fulvio Risso - Politecnico di Torino, Italy
EWSDN 2014, September 1st 2014

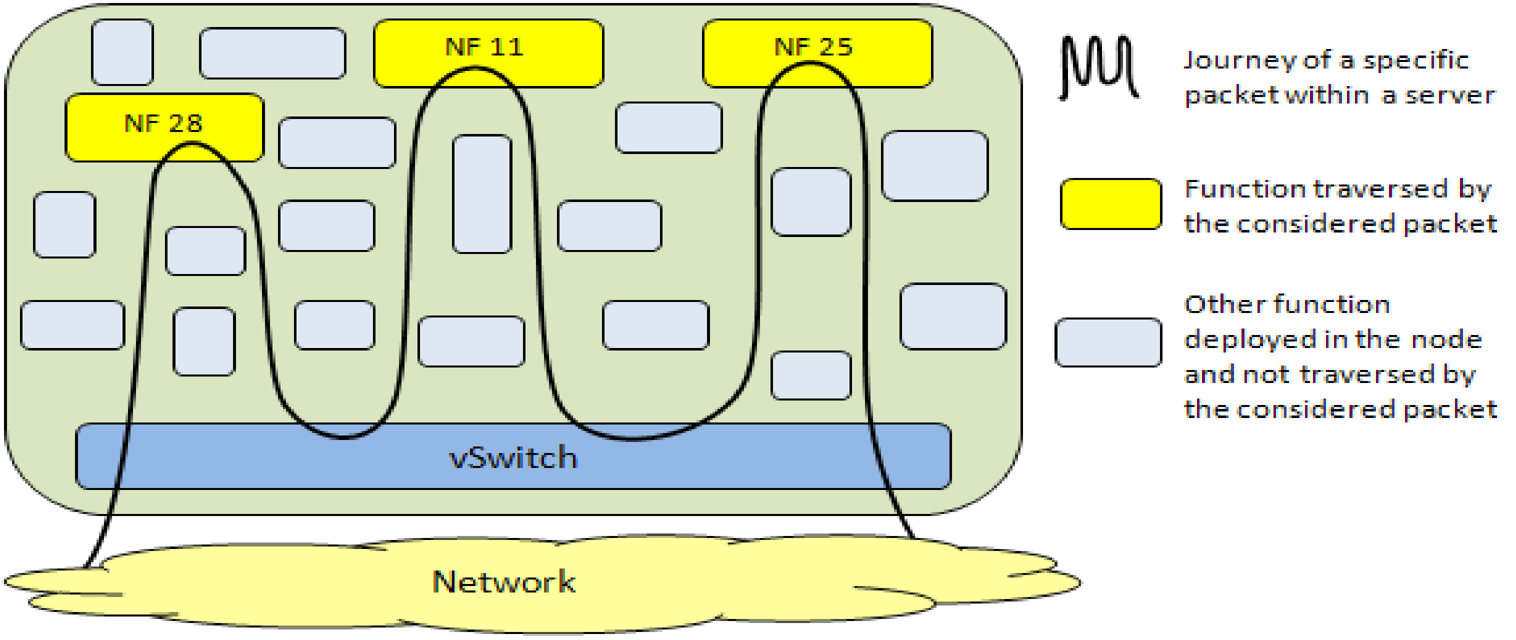
This project is co-funded
by the European Union

Outline

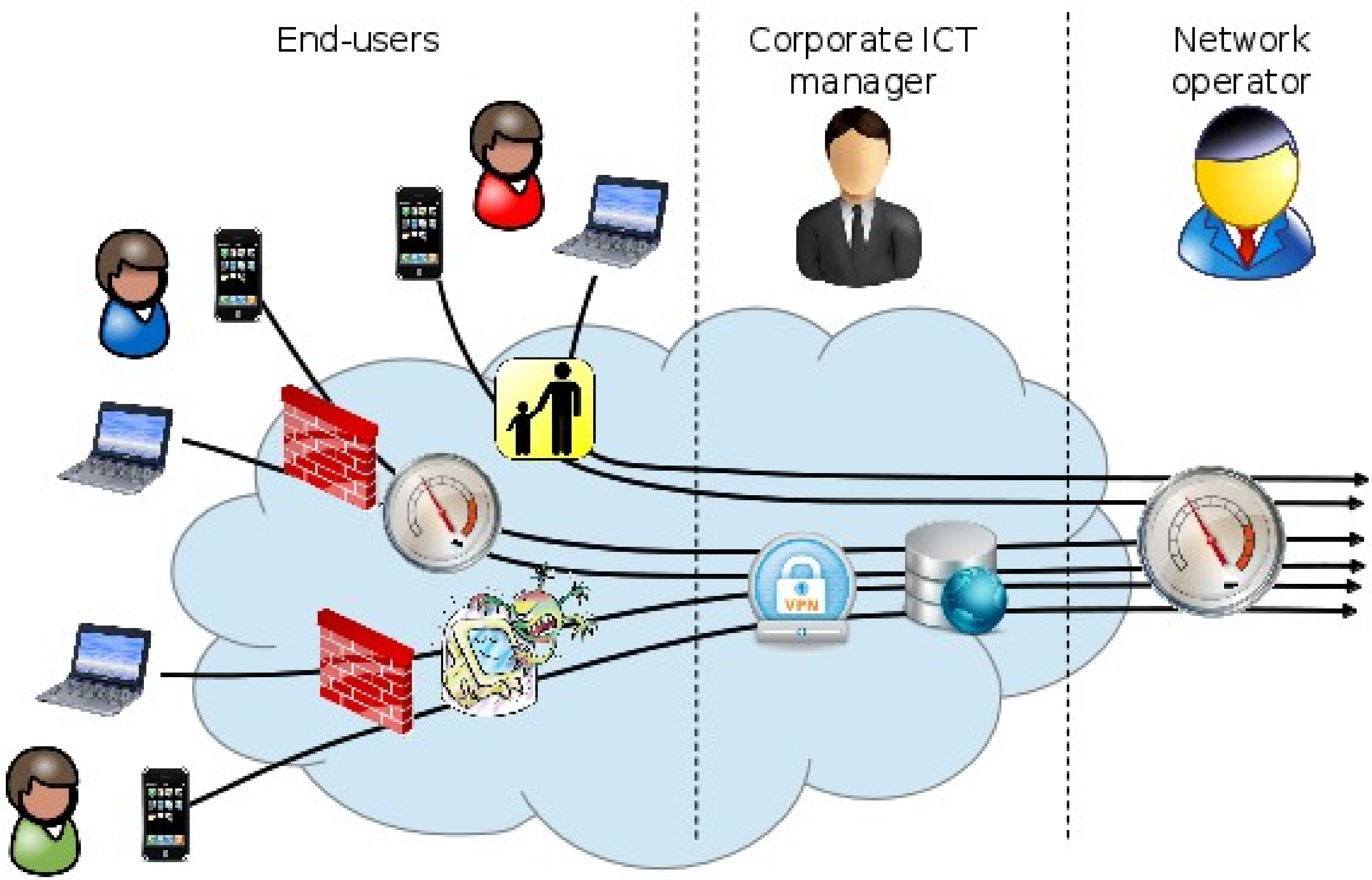
- Introduction (NFV)
- Goals of the work
- Intel DPDK
- Proposed architectures
  - “double buffer”
  - “double buffer + semaphore”
  - “double buffer + FDIR”
  - “isolated buffers + semaphore”
- Results
- Conclusion

Introduction

- **N**etwork **F**unctions **V**irtualization
  - transform network functions (e.g., NAT, firewall) into software images to be deployed on general purpose hardware
  - consolidate several network functions on the same node

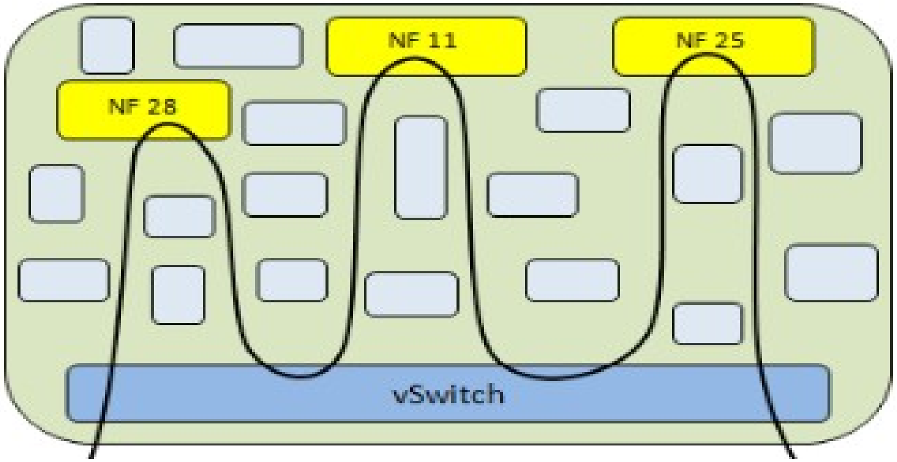


A possible target scenario



Goals of the work

- Propose and evaluate different architectures of the mechanism that transfers packets between the vSwitch and Nfs
  - constraints
    - latency and throughput
    - possible support a **huge number** of **fine-grained NFs**
- Exploit, whenever possible, the primitives offered by the DPDK framework



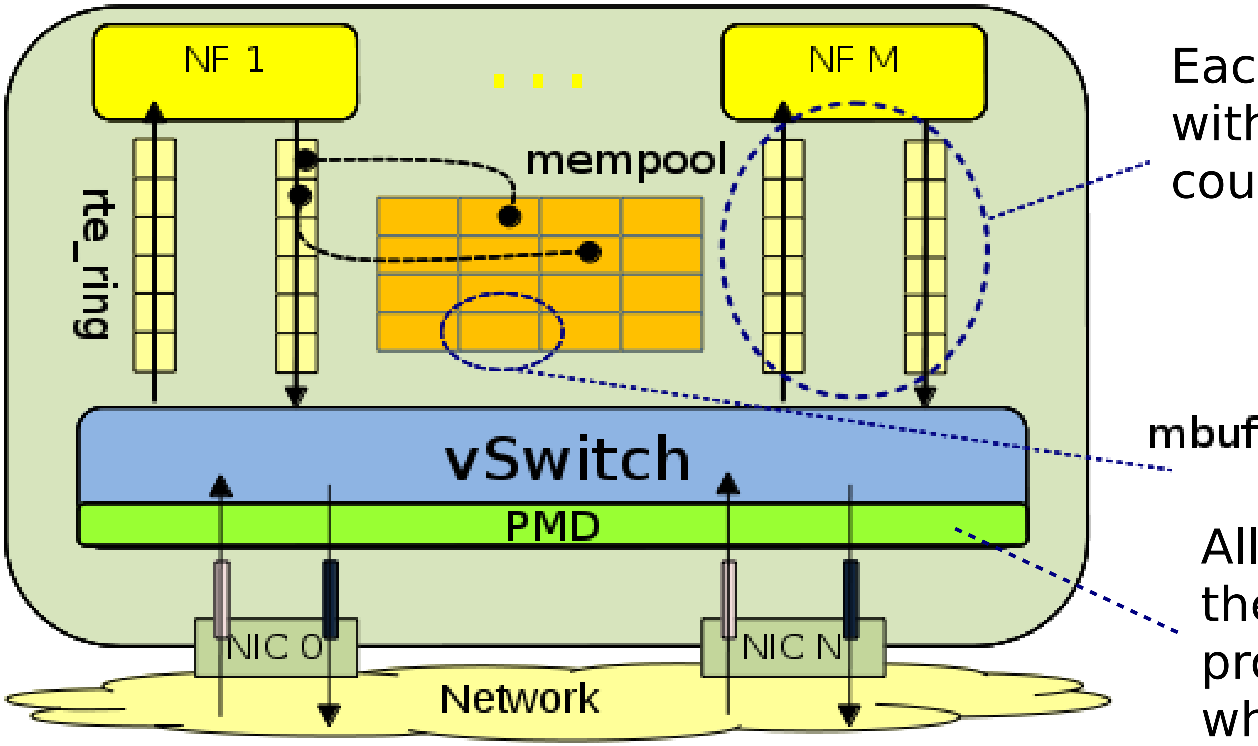
Intel DPDK

- Intel **D**ata **P**lane **D**evelopment **K**it
  - framework that offers efficient implementations for a set common packet processing functions
    - NIC packet input/output
    - memory allocation
    - packet queuing
    - easy access to hardware features (e.g., SR-IOV, FDIR)
- What we use in the proposed architectures:
  - **multi-process** support: the vSwitch is the primary process, NFs are secondary processes
  - **rte_mempool** to store packets
  - **rte_rings** to move packets in a zero-copy fashion
  - **PMD** to access to the NICs

Design choices

- The vSwitch:
  - supports only forwarding rules based on MAC addresses
  - operates in polling mode
- Network functions:
  - are UNIX processes
    - the massive amount of NFs that we need to handle, hence the pressure on CPU and memory occupancy of each NF would make VMs unpractical
  - may follow either the polling or interrupt-based model
    - depending on the number (and type) of NFs we expect to be active in the server

#1 “Double buffer” architecture



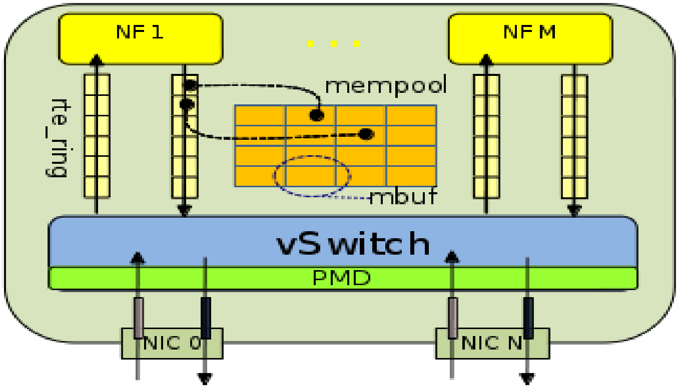
Each NF exchanges packets with the vSwitch through a couple of **rte_rings**

All the packets entering in the node are first processed by the vSwitch, which accesses to the NICs through the **PMD** library

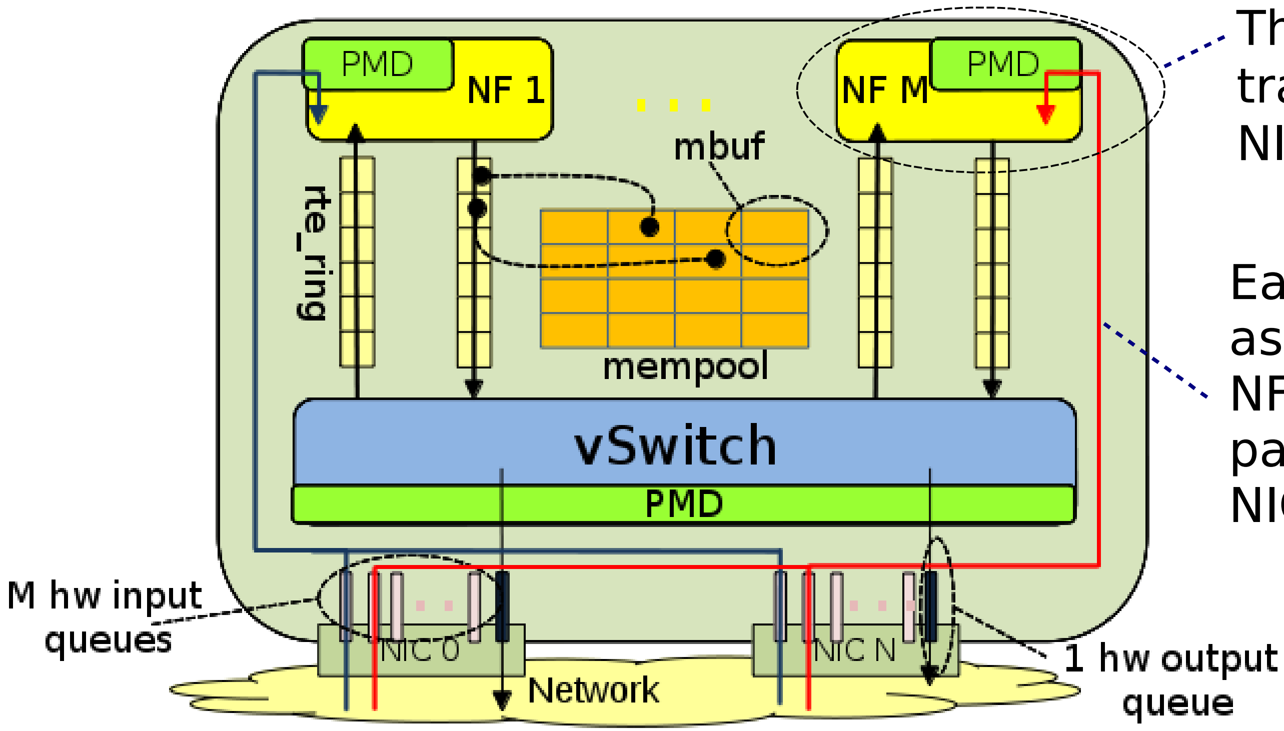
- ***NFs operate in polling mode***
- Implementation appropriate when a limited number of NFs is active (not more than the number of CPU cores available)

#2 “Double buffer + semaphore” architecture

- ***NFs operate in blocking mode***
  - the vSwitch wakes up, through a POXIS named semaphore, a NF when a given number of packets is available
  - a NF suspends itself when all the packets in the buffer have been processed
  - a timeout wakes up the NF if there are packets waiting for too long, to avoid data starvation
- Implementation appropriate when NFs need to process a limited amount of traffic
  - the polling model would unnecessarily waste a huge amount of CPU resources
- The blocking model allows to increase the density of the NFs active on the same server



#3 “Double buffer + FDIR” architecture

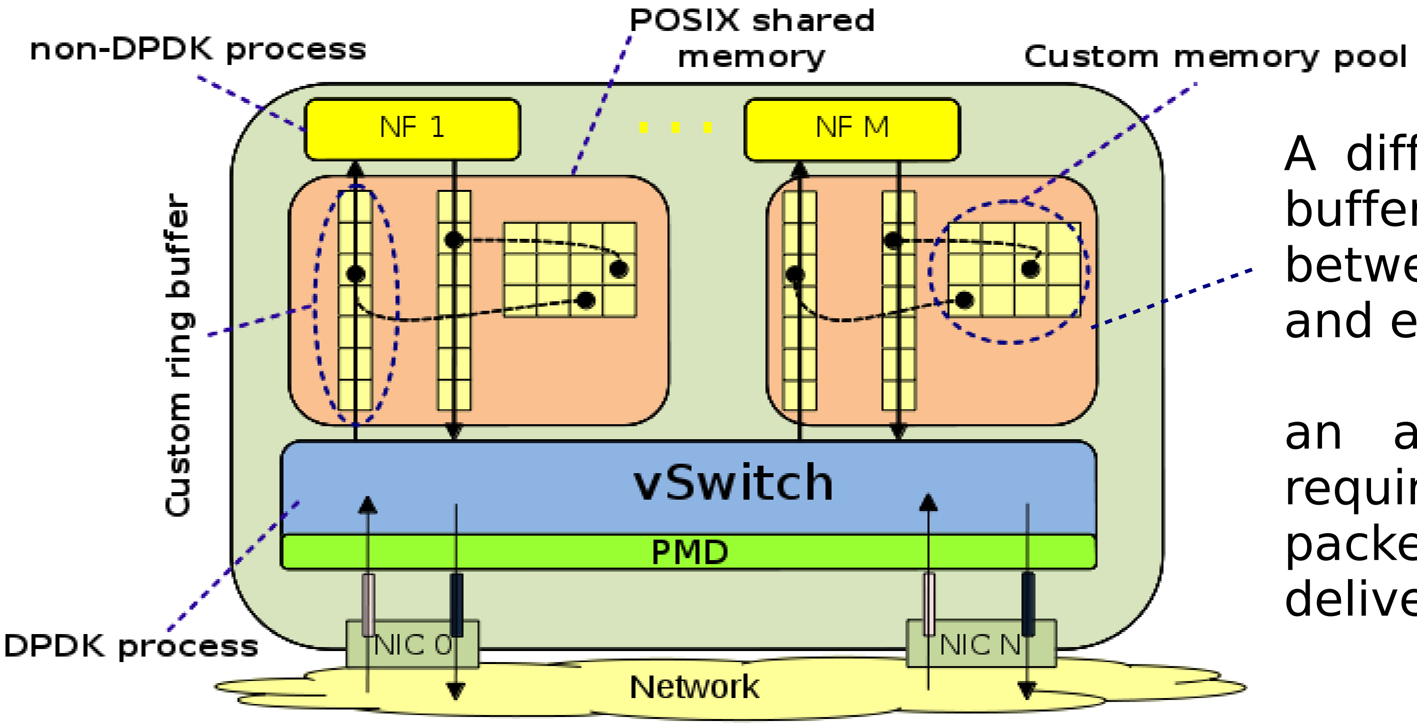


The NFs receive directly traffic coming from the NICs

Each input queue is associated with a different NF; the first classification of packets is offloaded to the NIC

- Remove an hop in the server, with a potential impact on the throughput and latency
- The number of hardware queues available on the NICs is limited
  - the architecture supports a small number of NFs

#4 “Isolated buffers + semaphore” architecture



A different set of three buffers is shared between the vSwitch and each NF

⬇

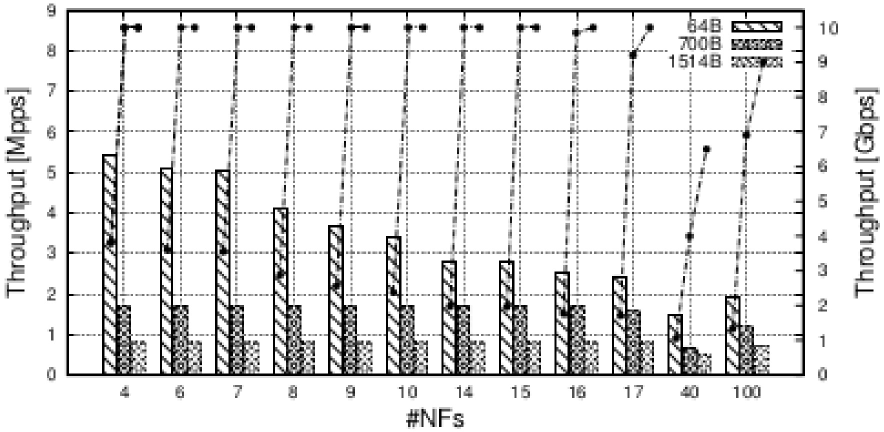
an additional copy is required each time a packet has to be delivered to the next NF

- Appropriate when NFs are not trusted
  - NFs cannot share a portion of memory with the rest of system
  - previous implementations allow any NFs to access and modify all the traffic flowing through the server
    - DPDK data structures shared among all the DPDK processes

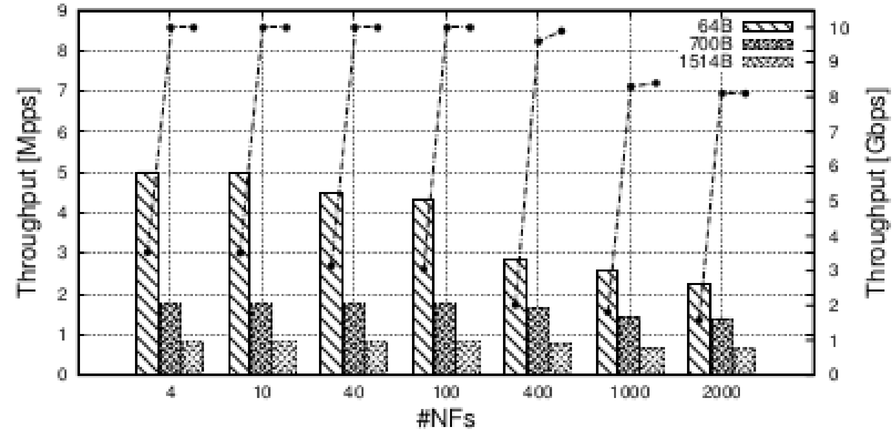
Results – Test conditions

- Machine:
  - Dual E5-2660 Xeon @ 2.20 GHz (8+8 cores)
  - Kernel Linux 3.5.0-17-generic 64 bits
  - 32GB memory
  - connected through a 10Gbps Ethernet link to a traffic generator, and through a 10Gbps Ethernet link to a traffic receiver
- 1 CPU core entirely dedicated to the vSwitch
- NFs distributed among the other cores to maximize the throughtput
- Each packet is processed in two NFs, according to its MAC address
- Two consecutive packets from the network are processed by different NFs
- Each NF calculates a signature on the first 64B of pkts

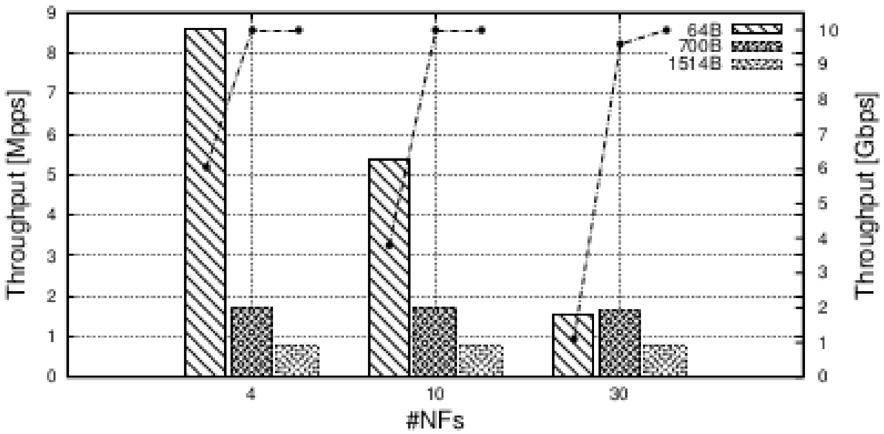
Results - Throughput



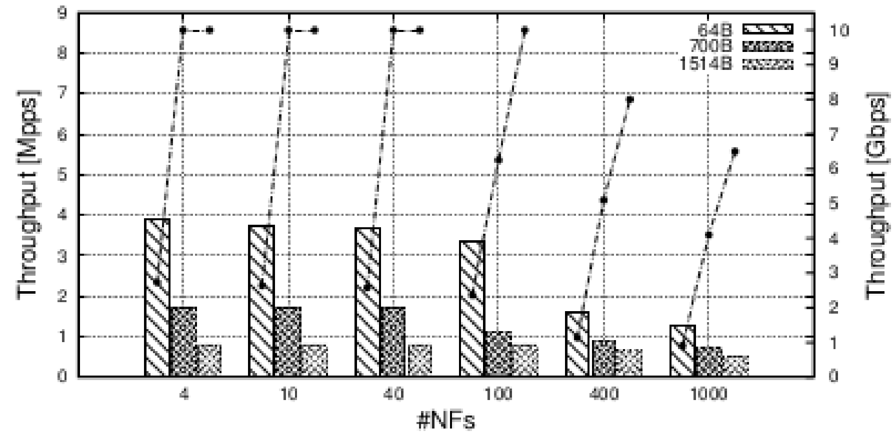
(a) "Double buffer" architecture.



(b) "Double buffer + semaphore" architecture.

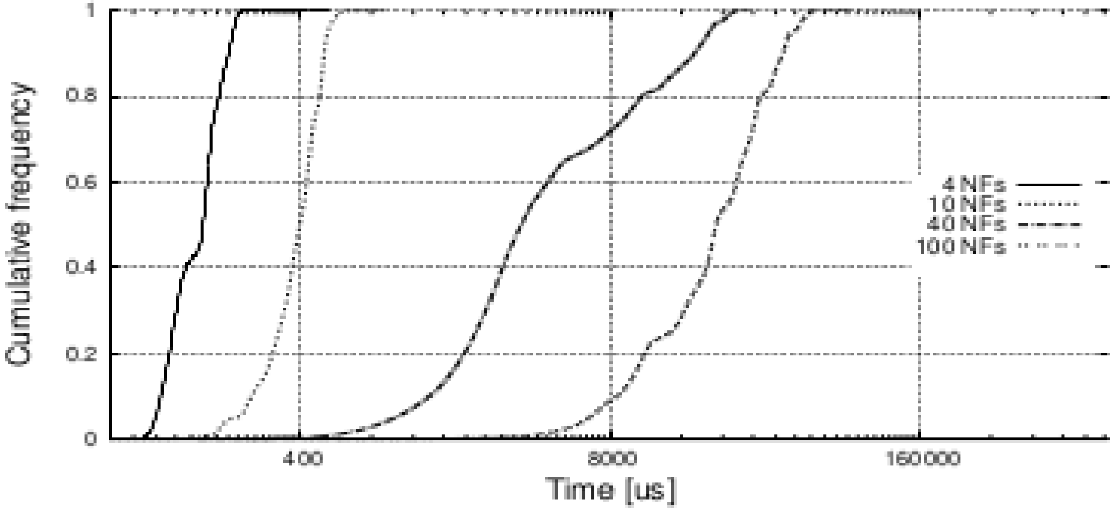


(c) "Double buffer + FDIR" architecture.

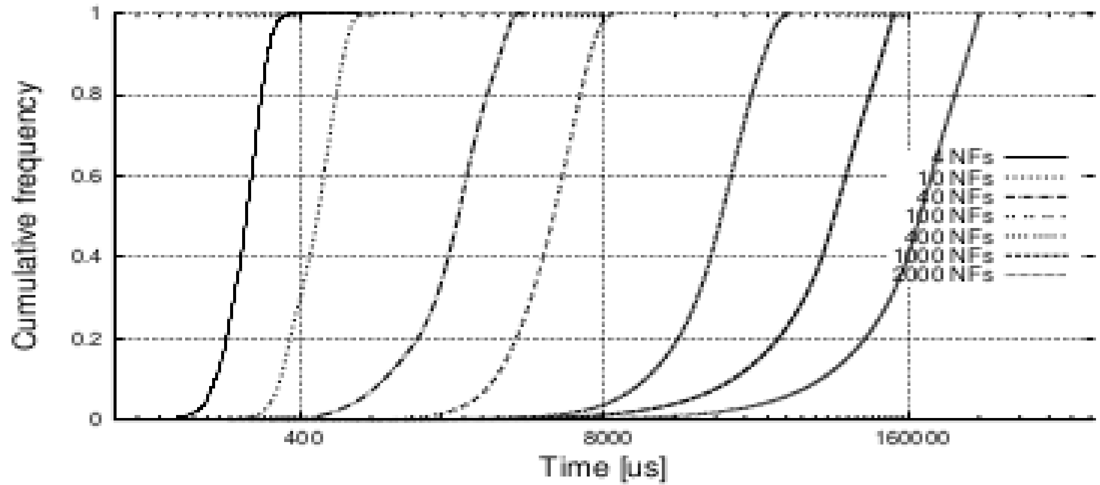


(d) "Isolated buffers + semaphore" architecture.

Results - Latency



(a) Latency introduced by the "double buffer" architecture.



(b) Latency introduced by the "double buffer + semaphore" architecture.

Results - Discussion

- NFs in “semaphore” mode seem more appropriate than in “polling” mode for our use case
  - limited performance loss with a few NFs
    - but we want to scale the number of NFs up
  - much better **scalability** when increasing the number of NFs
- Latency becomes rapidly unacceptable when packing the server with too many NFs

DPDK - Discussion

- DPDK seems to be engineered to support a **few NFs**
  - FDIR
  - a singe CPU core cannot be shared across multiple DPDK secondary processes (more details in the paper)
  - DPDK processes are not free to “float” across cores
  - binding of a NF to a precise CPU core
- DPDK provides limited support for the case of a massive number of NFs
- DPDK secondary processes MUST share some memory structures (isolation is not possible)

Conclusion

- Evaluated several architectures to exchange packets between the vSwitch and the **many tiny** NFs executed on a single server
- All the implementations are based, as much as possible, on the Intel DPDK
- Results are quite satisfying especially in terms of throughput
  - this also confirms the goodness of the primitives exported by the DPDK
- Latency becomes unacceptable when more than 100 NFs are deployed

Questions?