NetBricks: Taking the V out of NFV

Authors: Aurojit Panda and Sangjin Han, Melvin Walls and Sylvia Ratnasamy, Scott Shenker **(UoC, Berkeley),** & Keon Jang **(Google)**

Presented By: Anmol Mahajan & Sachin Meena

What will we cover

- Network Functions and their virtualization
- Challenges in virtualization
- Methods invented to overcome these challenges
- NetBricks:
  - Design & Implementation
  - Evaluation

Introduction

Background & Motivation

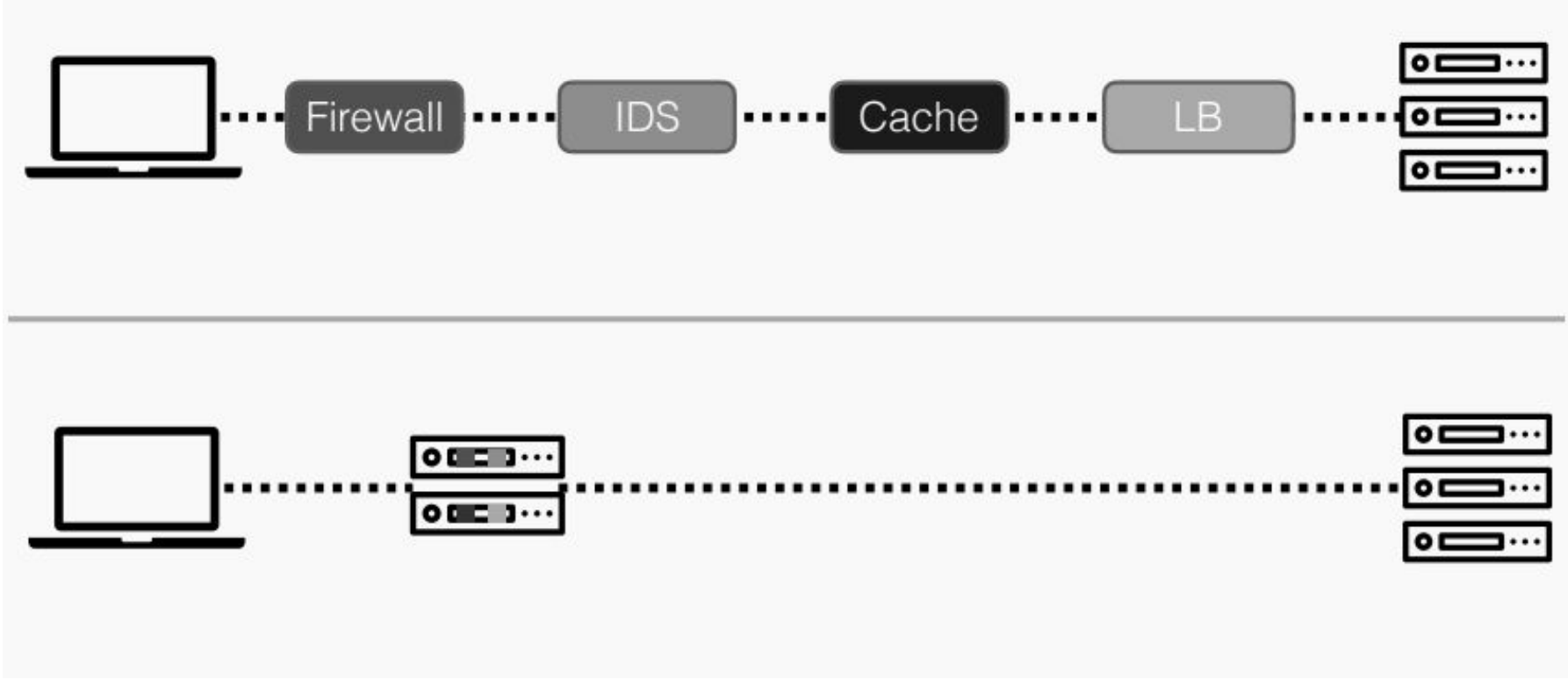
Design

Implementation

Evaluation

What are NFs?

- Networks need more functionality than forwarding packets
- Additional functionality implemented by middleboxes
- Middleboxes
  - Security, firewalls, IDS/IPS, caches, etc
  - Implemented by dedicated hardware
- Functions to replace these hardware devices - ***Network Functions***


Firewall
IDS
Cache
LB

What is NFV?

- Replacing dedicated routers (and firewall h/w, etc) with s/w on servers
- Aim: transform network architecture
  - No new hardware needed
- Example: Firewall, checks for packets from known malicious source, discards accordingly

Why NFV?

- Simplifying deployment, as new functionality only needs new s/w
- Cost reduction due to consolidation of many NFs on single machine
- Cost reduction in h/w setup
- Faster development

Why isn't NFV popular?

- Pre-reqs of carrier networks
  - **Performance**: latencies of O(10 μs) and throughput of O(10 Gbps)
  - **Chaining**: each packets needs to be processed by sequence of NFs
  - **Efficiency**: maximize number of NFs on single machine
- Current tools for building NFs fall short of these requirements

Introduction

Background & Motivation

Design

Implementation

Evaluation

Background

- State of art for NFV is much more primitive than that for programming
- Click - does not provide easily customizable low-level optimizations
- DPDK - fast and optimized I/O only
- NFV developers spend much time in code optimization
- More code tweaks may lead to more bugs

Building NFs

- Tools do not support
  - rapid development (achieved through high level abstractions)
  - high performance (requiring low-level optimizations)

Building NFs

- Click allows NF development by assembling various modules
- I/O is optimized, but developers responsible for other optimizations
- Modules support only limited customization, through parameters
- Developers often need to implement & optimize new modules

Running NFs

- Isolation between NFs is critical
  - as they might be from different vendors (*memory isolation*)
  - as each must be able to work in parallel (*performance isolation*)
- Current deployments rely on VMs for isolation
- VMs incur substantial overheads

Running NFs

- NICs are abstracted - multiple NFs can independently access network
- Allows existence of several NFs on one machine
- Allows *chaining* operations

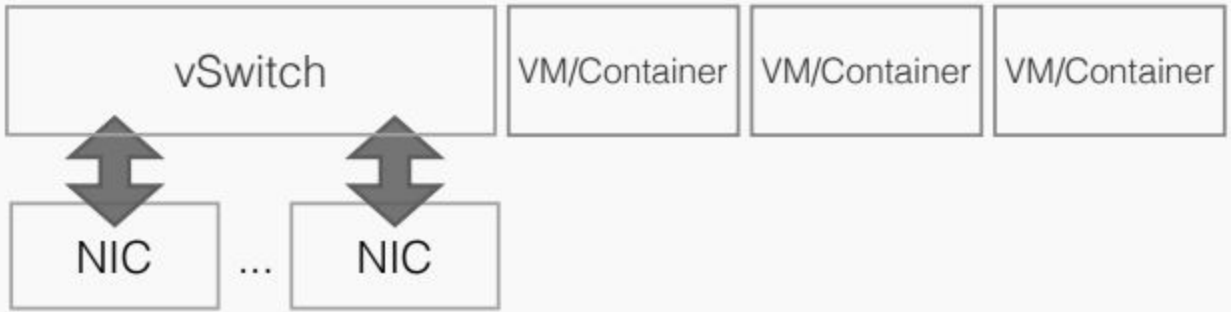
Penalties

- Comparison between
  - Single process running a dedicated NIC
  - Same functionality on a container
  - Same functionality on a VM

Penalties

- For single NF (processing smallest packets - 64B)
  - Per core throughput decreases by up to
    - 3x when using containers
    - 7x when using VMs
- Chained NFs
  - Containers are up to 7x slower
  - VMs are up to 11x slower

Current Solution

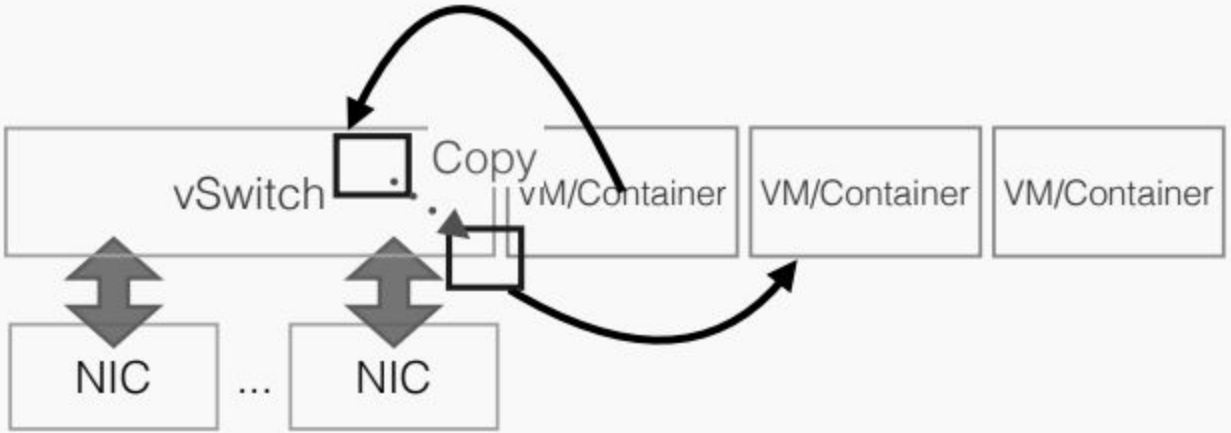


✔Memory Isolation

Packet Isolation

Performance

Current Solution



✔Memory Isolation

✔ Packet Isolation

✗ Performance

Reason

- During network I/O packets must cross a h/w memory isolation boundary
- This needs a context switch/syscall, which incurs significant overheads

Introduction
Background & Motivation
Design
Implementation
Evaluation

NetBricks: Novelty

- Instead of providing developers with many complex and non customizable modules, NetBricks focuses on a core set with well known semantics and highly optimized implementations
- Allows User Defined Functions for customizations
- Avoid overheads by relying on compile time and runtime checks for memory isolation in software
- Memory and packet isolation with **NO** overheads (ZCSI)
- Tradeoff between performance and flexibility

Design

- Programming Abstractions
  - Packet & Bytestream Processing
  - Control Flow
  - State Abstraction
  - Event Scheduling
- Execution environment
  - Isolation
  - Placement & Scheduling

Abstractions

Packet Processing
Parse/Deparse
Transform
Filter

Control Flow
Group By
Shuffle
Merge

Byte Stream
Window
Packetize

State
Bounded Consistency

Packet Processing Abstractions

- Netbricks uses its own packet structure
  - Stack of headers
  - Payload
  - Reference to any per packet metadata
- UDFs operating on a packet are provided with the packet structure
- UDFs can access the last parsed header, along payload and associated metadata

Packet Processing Abstractions

Operation	Input	Process/Output
Parse	Header type and packet structure	Parses the payload using header type and pushes resulting headers onto stack, removes the header bytes from payload
Deparse	-	Pops bottom most header back to payload
Transform	Packet structure, UDF	Modifies header/payload as per UDF
Filter	Packet, UDF	Allows packets meeting some criteria (as defined by UDF) to be dropped

Bytestream Processing Abstractions

Operation	Input	Process/Output
Window	Window size, Timeout, sliding increment, stream UDF	Waits till timeout or window size packets collected. Operates on available bytes. Responsible for receiving, reordering and buffering packets to reconstruct TCP stream.
Packetize	Packet structure	Given header stack and byte array, converts data into packets with appropriate headers attached

Control Flow Abstractions

- Necessary for branching and merging
- Branching is needed for implementing conditionals and multicore processing
- NFs need to minimize cross core access to avoid synchronization costs
- NetBricks provides partition mechanisms (port, destination address, or again, UDFs)
- Allows chaining of NFs

Control Flow Abstractions

Operation	Input	Process/Output
Group By	Packet, No. of target groups, packet based UDF	Branch control flow within NF or across NF chains. The UDF function returns the ID of the group that the packet will go to
Shuffle	Packet	Similar to Group By, except that #target groups is based on #active cores. Shuffle outputs are processed on other cores
Merge	Packets from different branches	A single group of packets

State Abstractions (for data serializability)

- Across cores, cache coherence and synchronization incur excess cost
- NFs are programmed to partition state & minimize cross core access
- NetBricks provides state abstractions that partition data across cores
- Inter-core access have following options:
  - Not allowed
  - Only reads allowed (possibly with certain conditions)
  - Serializable multi write multi read access through synchronization

Scheduled Event Abstractions

- Means to run arbitrary UDFs at given times or periodically
- Helps in implementing, for ex, NFs with monitoring functionality

Runtime Isolation

- Need??
- VM based isolation incurs heavy penalties for simple NFs
- NetBricks uses software isolation instead
- Previous research - safe languages with type checks, and runtimes can provide memory isolation equivalent to that provided by MMU
- NetBricks uses Rust (type checking) & LLVM (runtime env)

Previous research

Property	Implication
Disallow pointer arithmetic	No arbitrary pointers to (isolated) memory
References by allocation or function call	No arbitrary reference to (isolated) memory
Checking bounds on array access	Prevents stray memory access
Disallow access to null objects	Prevents applications from using undefined behavior to access (isolated) memory
Safe and compatible type casts	No unwanted memory access

Zero Copy Soft Isolation

- NFV requires that an NF cannot modify a packet once it has been sent
- *Packet isolation* - usually achieved by copying (performance overhead)
- ***Unique Types*** - NO simultaneous access to same data from 2 threads
- Verification at compile time, to avoid runtime overheads
- NetBricks designed so that only single NF has access to packet

NetBricks: Cornerstones

- Provides memory and packet isolation
- Multiple NFs can now share a core
  - Switches between NFs through function calls
  - Function calls (few cycles) vs context switches (1 µs)
- Reduce memory and cache pressure
  - ZCSI - no need to copy packets

Placement and Scheduling

- NetBricks runs several NFs (several parallel directed graphs)
- NetBricks must decide at compile time what core is to be used to run each NF chain
- NetBricks must make scheduling decisions about which packet to process next
- Currently using run-to-completion scheduling
- Currently using round robin scheduling for deciding event scheduling

Introduction
Background & Motivation
Design
Implementation
Evaluation

Example: Decrementing TTL

```
pub fn ttl_nf<T: 'static + NbNode>(input: T)
                          -> CompositionNode {
  input.parse::<MacHeader>()
       .parse::<IpHeader>()
       .transform(box |pkt| {
            let ttl = pkt.hdr().ttl() - 1;
            pkt.mut_hdr().set_ttl(ttl);
       })
       .filter(box |pkt| {
            pkt.hdr().ttl() != 0
       })
       .compose()
}
```

Listing 1: NetBricks NF that decrements TTL, dropping packets with TTL=0.

```
// cfg is configuration including
// the set of ports to use.
let ctx = NetbricksContext::from_cfg(cfg);
ctx.queues.map(|p| ttl_nf(p).send(p));
```

Listing 2: Operator code for using the NF in Listing 1

Example: Maglev

- Packet processing and forwarding part
- Unsynchronized cache

```
pub fn maglev_nf<T: 'static + NbNode>(
                input: T
                backends: &[str],
                ctx: nb_ctx,
                lut_size: usize)
                -> Vec<CompositionNode> {
  let backend_ct = backends.len();
  let lookup_table =
      Maglev::new_lut(ctx,
            backends,
            lut_size);
  let mut flow_cache =
    BoundedConsistencyMap::<usize, usize>::new();

  let groups =
    input.shuffle(BuiltInShuffle::flow)
         .parse::<MacHeader>()
         .group_by(backend_ct, ctx,
             box move |pkt| {
               let hash =
                 ipv4_flow_hash(pkt, 0);
               let backend_group =
                  flow_cache.entry(hash)
                  .or_insert_with(|| {
                 lookup_table.lookup(hash)});
               backend_group
     });
    groups.iter().map(|g| g.compose()).collect()
}
```

Listing 3: Maglev [9] implemented in NetBricks.

Implementation of Abstractions

- Packet Processing abstractions are lazy.

  Eg. parse nodes do not perform computation until a transform, filter, group by.

- Abstractions process batches of packets for high-performance.

Notes

- Operators running Netbricks chain NFs using same language used for writing NF.
- This provides many optimization opportunities.
- Builds on Rust and uses LLVM as runtime.

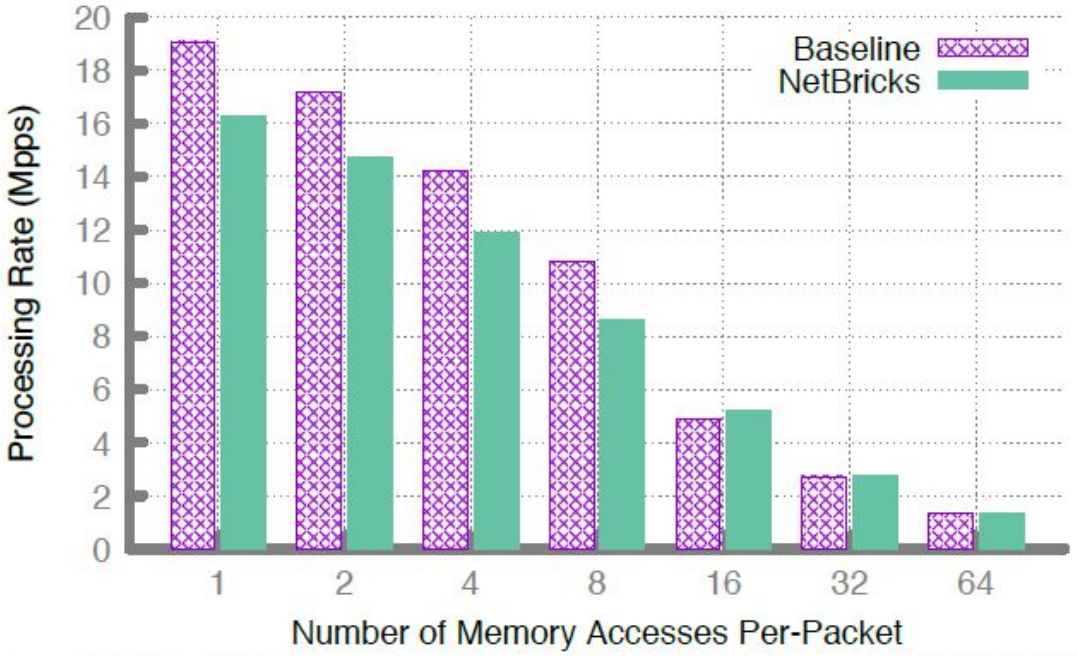
Introduction
Background & Motivation
Design
Implementation
Evaluation

Setup

- Testbed of dual-socket servers equipped with Intel Xeon E5-2660 CPUs
- Each with 10 cores
- Intel XL710 QDA2 40Gb NIC
- 2 Virtual Switches-
  - OpenVSwitch with DPDK.
  - SoftNIC (new virtual switch optimized for NFV use cases)

Overhead for checking array bounds

- Due to use of a safe language.
- Impact of cache misses
- LPM lookup table



Cost of Isolation: Single NF

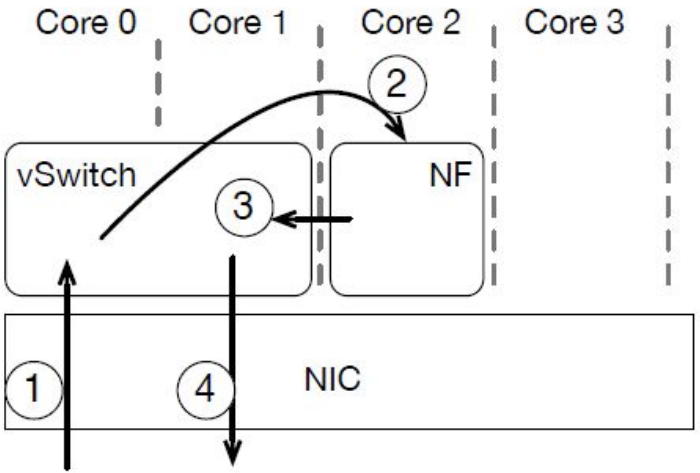


Figure 2: Setup for evaluating single NF performance for VMs and containers.

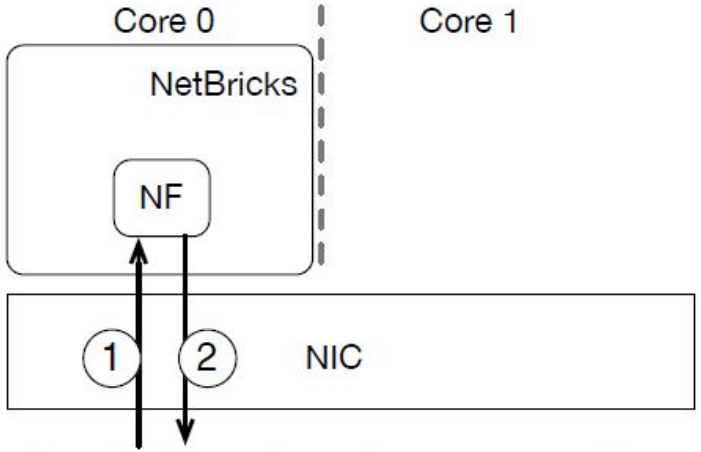


Figure 3: Setup for evaluating single NF performance using NetBricks.

- Only 15% increase in per-packet processing time between 64B and 1500B for cases that involve copying

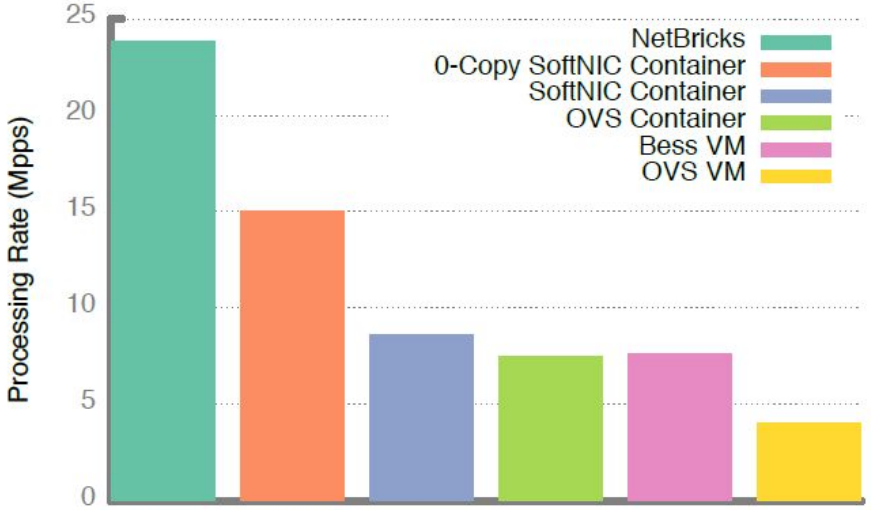


Figure 4: Throughput achieved using a single NF running under different isolation environments.

Cost of Isolation: NF Chains

- Each packet handled by a chain of NF's

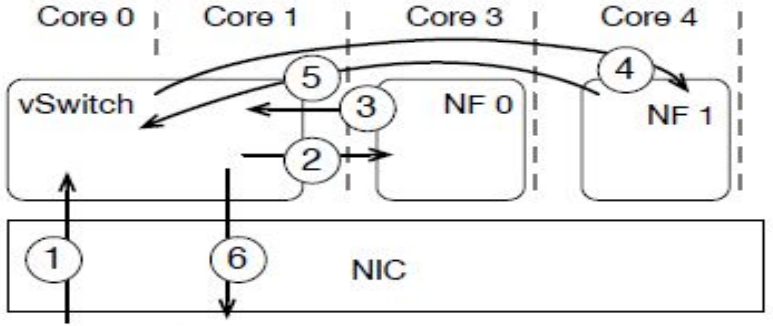


Figure 5: Setup for evaluating the performance for a chain of NFs, isolated using VMs or Containers.

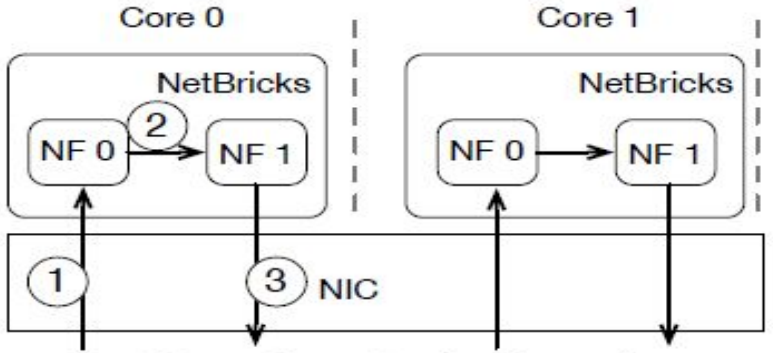


Figure 6: Setup for evaluating the performance for a chaining of NFs, running under NetBricks.

- Netbricks is run under 2 configurations-> single core and multiple cores.
- Trend for Netbricks in multicore configuration!!!

  I/O becomes more expensive as more cores access the same NIC.

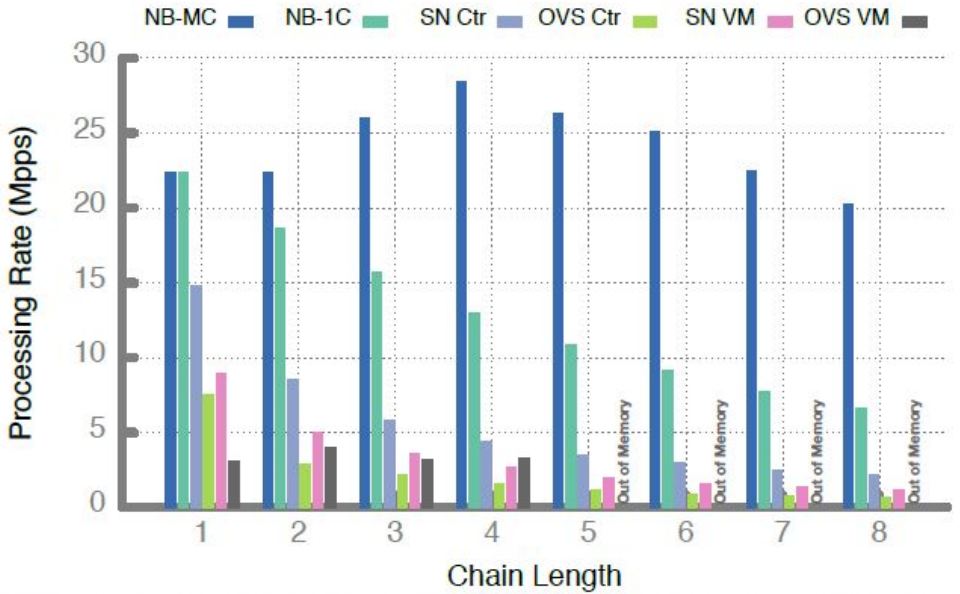


Figure 7: Throughput with increasing chain length when using 64B packets. In this figure NB-MC represents NetBricks with multiple cores, NB-1C represents NetBricks with 1 core.

Effect of Increasing NF complexity

As we increase NF complexity, packet processing time starts to be the dominant factor

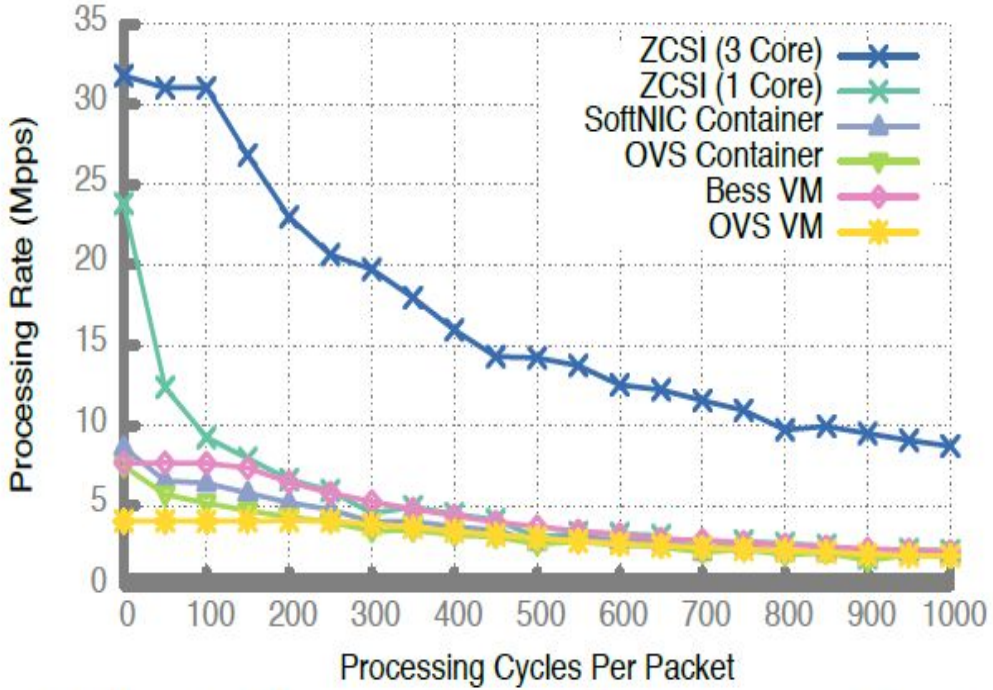


Figure 9: Throughput for a single NF with increasing number of cycles per-packet using different isolation techniques.