NetBricks: Taking the V out of NFV

Authors: Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, Soctt Shenker

Presented By: 孙晓鹏

2019.5.17

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

What will cover

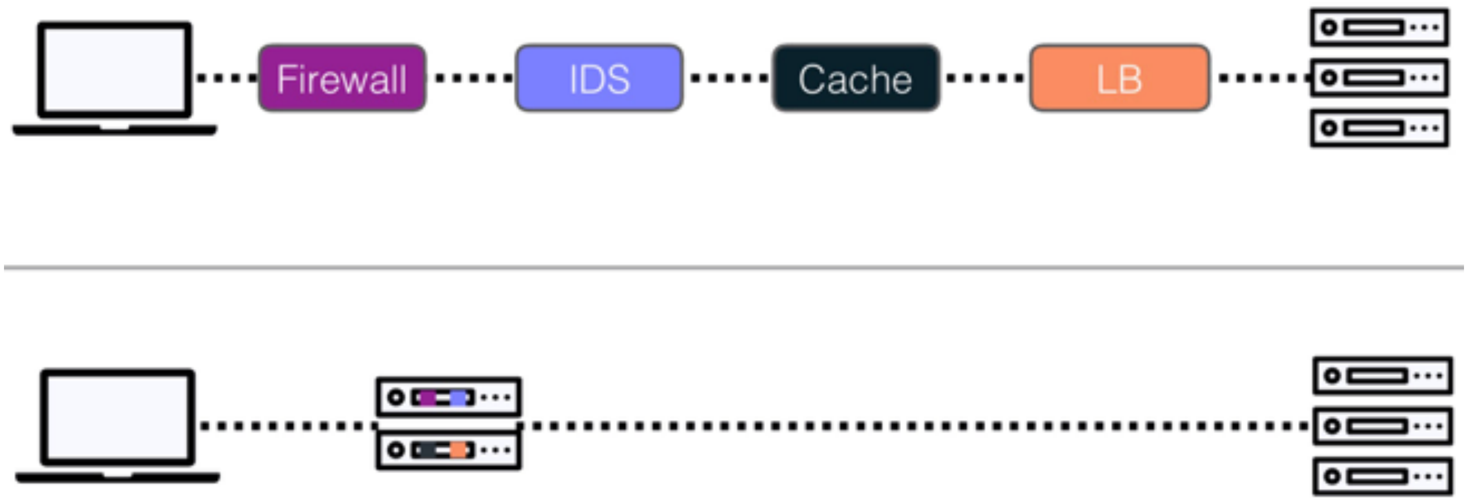
- Network functions and their virtualization
- Challenges in virtualization
- Methods invented to overcome these challenges
- NetBricks:
    - Design & Implementation
    - Evaluation

Contents

Contents

What is NFV?

- Replacing dedicated hardware with software on servers
- Aim: transform network architecture
- No new hardware needed



Why NFV?

- Simplifies adding new functionality: Deploy new software
- Simplifies developing new functionality: Write software vs design hardware
- Reuse management tools form other domains
- Reduce cost

Why NFV unpopular?

- NFV requirements:
  - Performance: latencies & throughput
  - Efficiency: maximize number of NFs on single machine
  - Chaining: each packet needs to be processed by sequence of NFs

- Current tools for building NFs fall short of these requirements:
  - State-of-the-art for NFV is much more primitive
  - VMs/containers incur substantial performance overheads

Contents

Background

- State-of-the-art for NFV is 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)

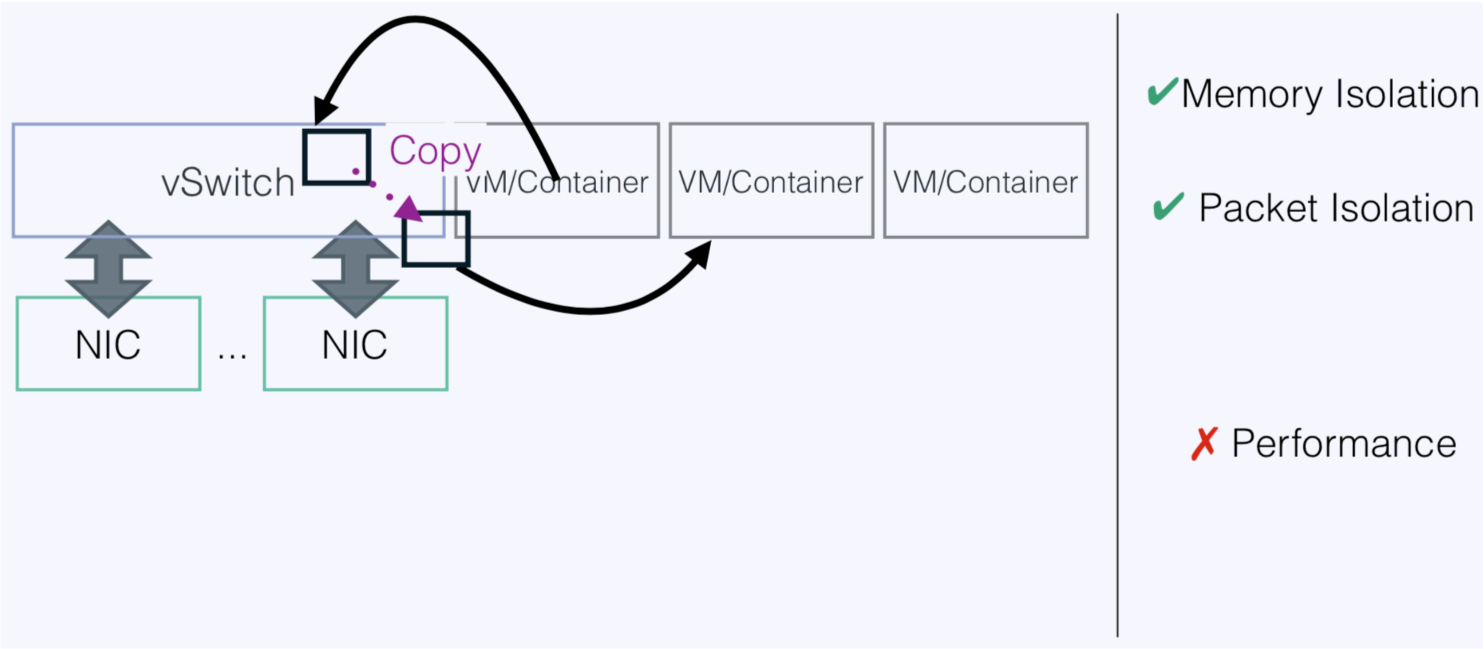
Building NFs

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

Running NFs

- Isolation between NFs is critical
  - Memory isolation
  - Packet isolation
  - Performance isolation
- Current deployment relay on VMs for isolation
- VMs incur substantial overheads

Current Solution



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

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)
  - 3x when using containers
  - 7x when using VMs
- For chained NFs
  - 7x when using containers
  - 11x when using VMs

Contents

Design

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

Abstractions

Packet Processing

- Parse/Deparse — Header
- Transform — UDF
- Filter — UDF

Control Flow

- Group By — UDF
- Shuffle — UDF
- Merge

Byte Stream

- Window — UDF
- Packetize — UDF

State

- Bounded Consistency

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 structure, UDF	Allows packets meeting some criteria (as defined by UDF) to be dropped

Execution environment

- Memory isolation:
  - VM based isolation incurs heavy penalties
  - NetBricks uses software isolation instead
  - Previous work - 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)

Execution environment

- Packet Isolation:
  - Usually achieved by copying
  - Zero Copy Soft Isolation (ZCSI)
    - Unique Types - NO simultaneous access to same data from 2 threads
    - Verification at compile time, to avoid runtime overheads

Execution environment

- Placement and Scheduling:
  - NetBricks runs serval NFs
  - 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

Contents

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Example NF: Maglev

- Maglev: Load balancer from Google
- NetBricks implementation: 105 lines, 2 hours of time
- Comparable performance to optimized code

```
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.

Contents

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Setup

- Dual-socket servers equipped with Intel Xeon E5-2660 CPUs
- Each with 10 cores
- Intel XL710QDA2 40Gb NIC
- 2Virtual Switches
- OpenVSwitch with DPDK
- SoftNIC (new virtual switch optimized for NFV use cases)

Overhead for checking array bounds

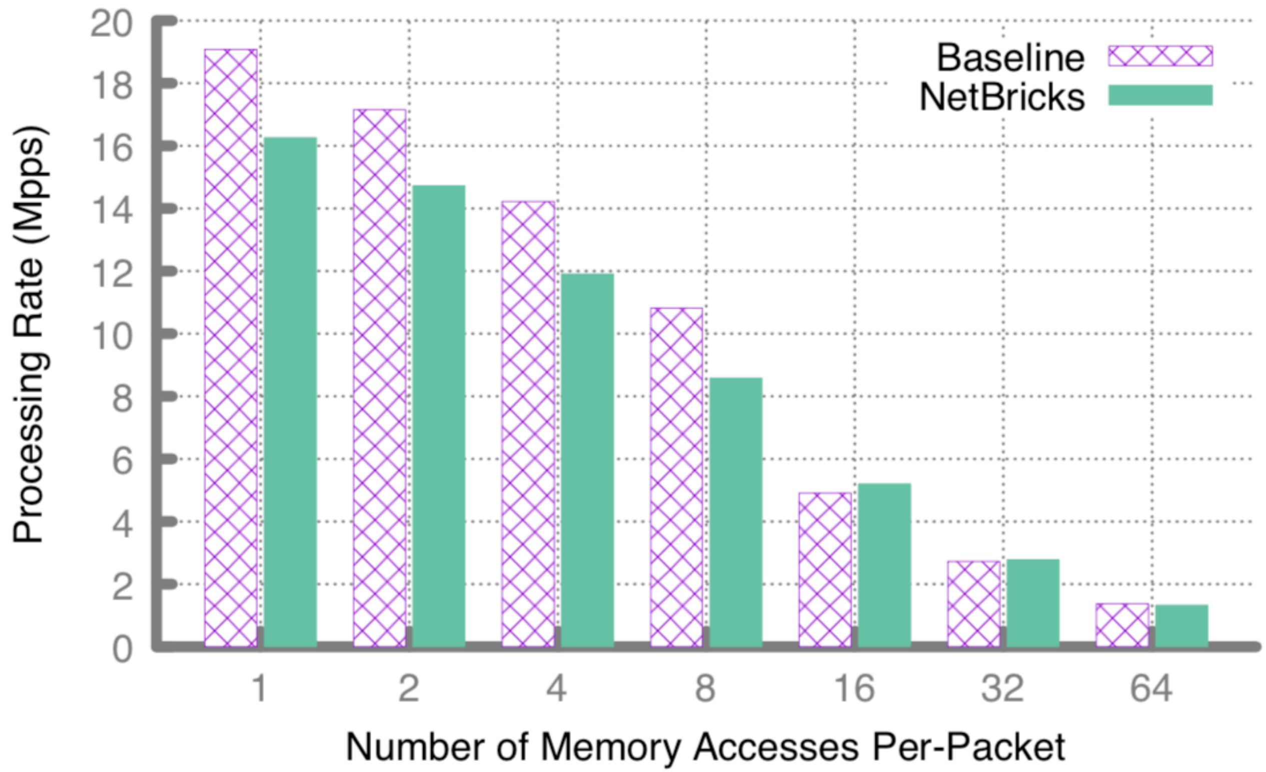


Figure 1: Throughput achieved by a NetBricks NF and an NF written in C using DPDK as the number of memory accesses in a large array grows.

Cost of isolation

- Single NF

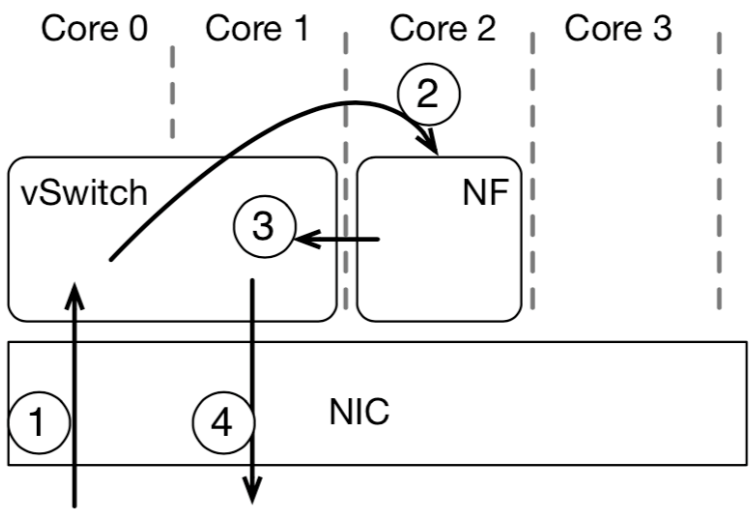


Figure 2: Setup for evaluating single NF performance for VMs and containers.

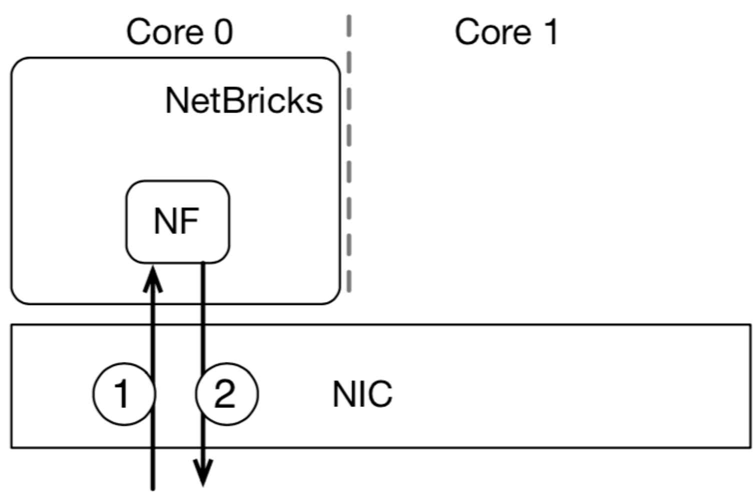


Figure 3: Setup for evaluating single NF performance using NetBricks.

Cost of isolation

- Single NF

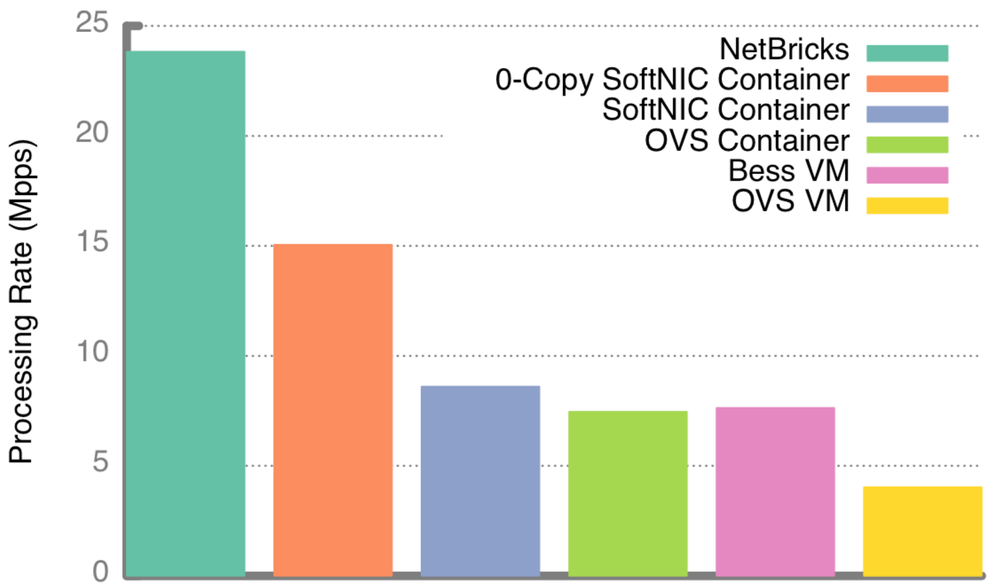


Figure 4: Throughput achieved using a single NF running under different isolation environments.

Cost of isolation

- NF Chains

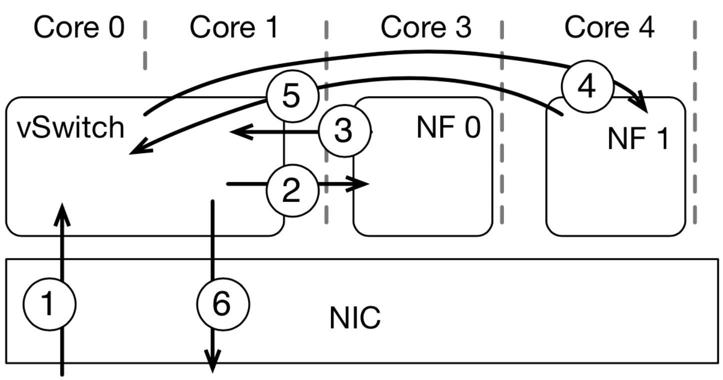


Figure 5: Setup for evaluating the performance for a chain of NFs, isolated using VMs or Containers.

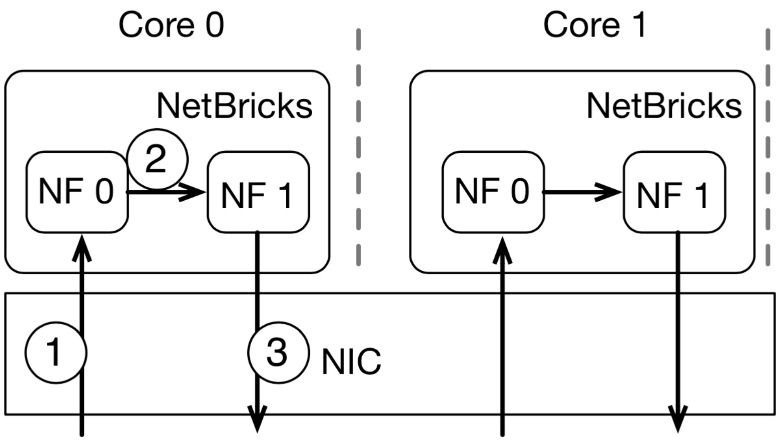


Figure 6: Setup for evaluating the performance for a chaining of NFs, running under NetBricks.

Cost of isolation

- NF Chains

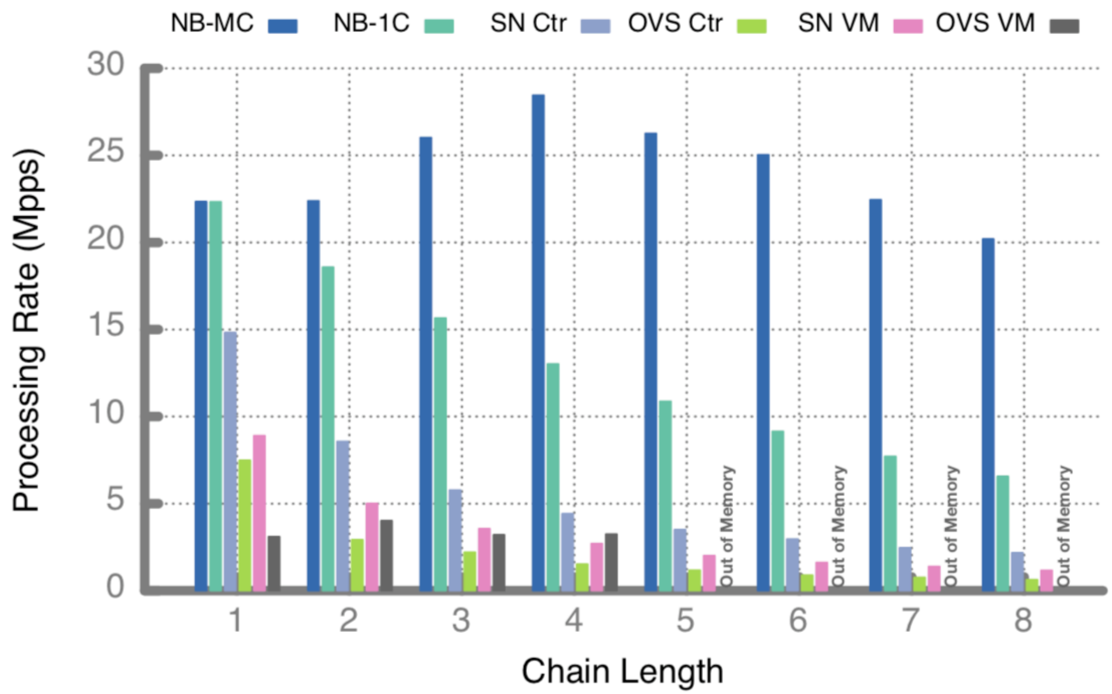


Figure 7: Throughput with increasing chain length when using 64B packets. In this figure NB-MC represents NetBricks with multiple cores, NB-1C represents NetBricks with 1 core.

Effect of Increasing NF complexity

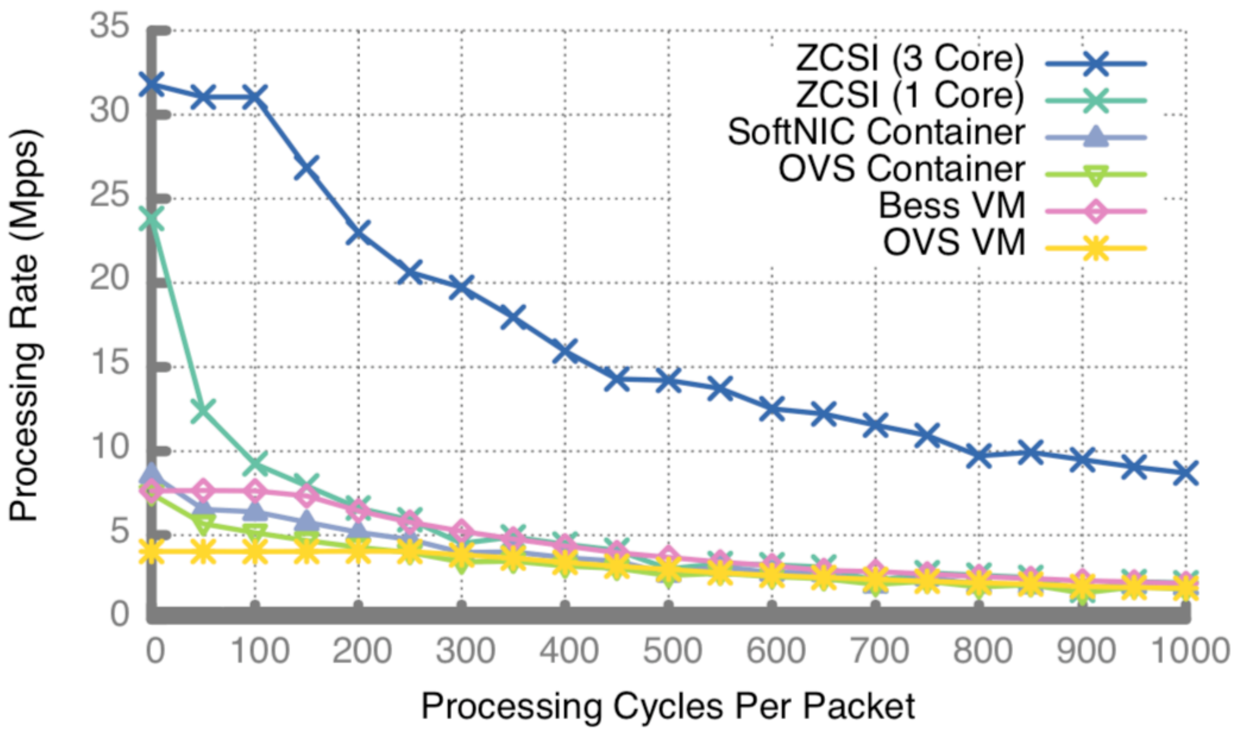


Figure 9: Throughput for a single NF with increasing number of cycles per-packet using different isolation techniques.

Contents

Conclusion

- High-level programming brings convenience
  - Abstract operators + UDF simplify development
- Software isolation is necessary for high performance NFV
  - Type checking + bound checking + unique types

Related work

- ClickOS (NSDI' 14)
- E2: a framework for NFV applications (SOSP' 15)
- Rollback-Recovery for Middleboxes (SIGCOMM' 15)
- NFP: Enabling Network Function Parallelism in NFV (SIGCOMM' 17)
- Adaptive Interference-Aware VNF Placement for Service-Customized 5G Network Slices (INFOCOM' 19)

Thank you !

Backup

NetBricks Runtime Architecture