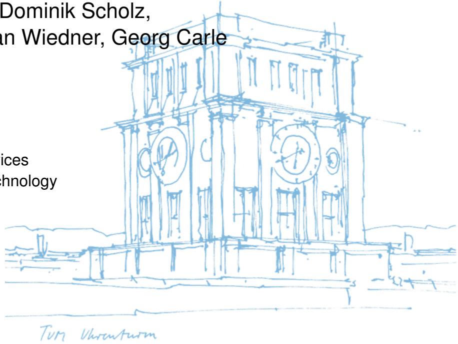
Chair of Network Architectures and Services
School of Computation, Information, and Technology
Technical University of Munich

High-Performance Packet Processing Experiments

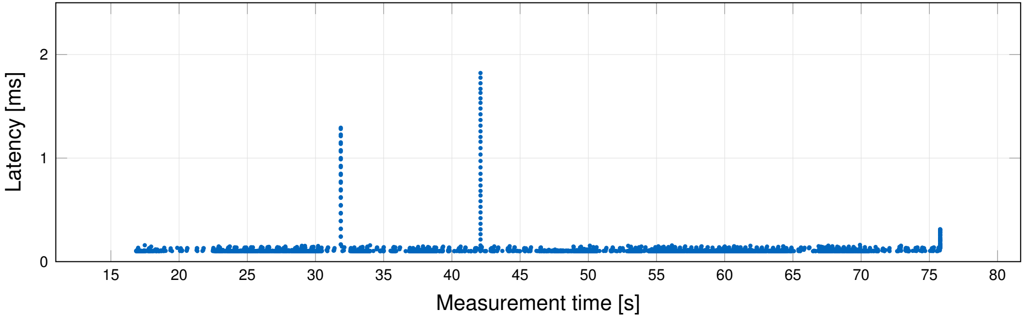
Sebastian Gallenmüller, Johannes Naab, Dominik Scholz,
Henning Stubbe, Manuel Simon, Eric Hauser, Florian Wiedner, Georg Carle

Chair of Network Architectures and Services
School of Computation, Information, and Technology
Technical University of Munich



Motivation

Latency of a Network Function



Suricata forwarder worst-case latencies

- Latency spikes are caused by the OS network stack (happen for any application)

➔ Why should we care about 1 or 2 ms?

Motivation

Why should we care about latency?

IEEE standard	TX Rate [Gbit/s]	Serialization Delay [ns]	Impacted Packets [#/ms]
802.3z	1	672.0	1488
802.3ae	10	67.2	14 880
802.3bm	100	6.7	149 253
802.3bs	400	1.7	588 235
P802.3dj	1600	0.4	2 500 000

1-ms transmission for different Ethernet bandwidths

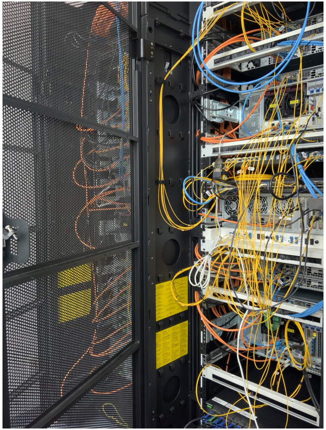
- Using minimum-sized Ethernet packets (64 B incl. FCS) at full line rate
- Impact increases for every new standard
- For 1.6 Tbit/s a 1-ms delay 2.5 M packets are impacted (approx. 150 MB)

→ High-performance packet processing needs to pay attention to delays

Motivation

Main challenges

1. Measurement methodology that can handle the latency
   - How to measure reproducibly?
   - How to measure at high bandwidths?
   - How to measure latency precisely and accurately?
2. Low-latency measurement examples[1]
   - What is causing latency on software packet processing systems?
   - What is the impact of specific components on software packet processing?



High-performance network testbed

[1] [3] S. Gallenmüller, F. Wiedner, J. Naab, and G. Carle. "How Low Can You Go? A Limbo Dance for Low-Latency Network Functions". In: J. Netw. Syst. Manag. 31.1 (2023), p. 20

Measurement Methodology

Reproducible Measurements—The Plain Orchestrating Service (pos)

Our solution to create reproducible research

1. Create a testbed management system
2. Create a well-defined experiment workflow

[2] [2] S. Gallenmüller, D. Scholz, H. Stubbe, and G. Carle. “The pos framework: a methodology and toolchain for reproducible network experiments”. In: CoNEXT. ACM, 2021

Reproducible Measurements—The Plain Orchestrating Service (pos)

Our solution to create reproducible research

1. Create a testbed management system
2. Create a well-defined experiment workflow

Achieving Repeatability

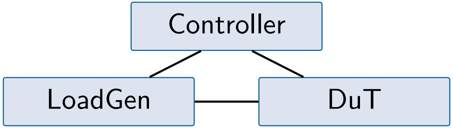
- Automation
- Live images
  - Researchers **must** automate configuration
  - No residual state between reboots

➔ Experiments become repeatable

Achieving Reproducibility

- Providing access to experiment infrastructure
- Other researchers can easily (re-)run experiment

➔ Experiments become reproducible



Minimal pos[2] experiment topology

[2] [2] S. Gallenmüller, D. Scholz, H. Stubbe, and G. Carle. "The pos framework: a methodology and toolchain for reproducible network experiments". In: CoNEXT. ACM, 2021

Reproducible Measurements—pos Experiment Workflow

Setup phase

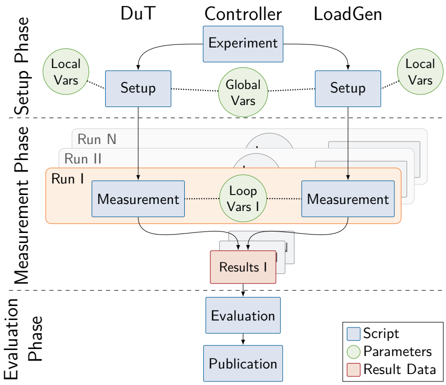
- Controller manages experiment
- Controller configures experiment nodes (DuT, LoadGen)
- Global / local variables (vars) parametrize setup

Measurement phase

- Repeated execution of measurement script
- Loop variables parameterize each measurement run
  - e.g., different packet rates
  - data of each run is connected to a specific set of loop vars

Evaluation phase

- Collected results / loop vars used for experiment evaluation
- Automated experiment release (git repository, website)



pos workflow

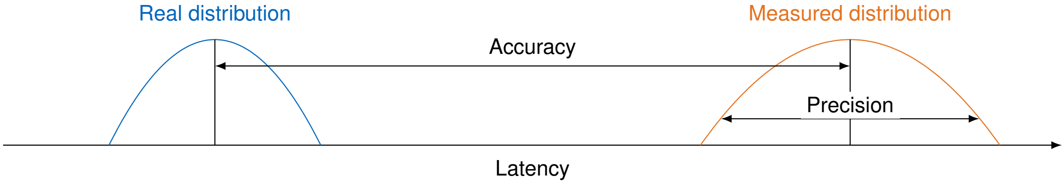
High-Performance Measurement and Generation of Traffic—MoonGen

MoonGen[3] features

- Software packet generator
  - Easy to adapt (via Lua scripting language)
  - High-performance (up to 100 Gbit/s or 100 million packets per second)
- Important features for measurements with high bandwidths
  1. Precise rate control
     - Traffic patterns can have a significant impact on measurement results
     - MoonGen allows to precisely control traffic patterns (via software and with hardware support)
  2. Timestamping
     - NICs typically offer precise clocks for PTP (Precise Time Protocol)
     - MoonGen uses these clocks for hardware timestamping

[3] [1] P. Emmerich et al. “MoonGen: A Scriptable High-Speed Packet Generator”. In: ACM IMC, Tokyo, Japan, 2015

Measuring Latency—High-Accuracy and High-Precision Timestamping



Accuracy vs. precision

- Accuracy: *"closeness of agreement between a test result and the accepted reference value"*
- Precision: *"closeness of agreement between independent test results"*
- Accuracy can be improved if timestamps are taken early in the processing path
- Precision can be improved if measurements are not impacted by jitter, e.g., caused by interrupts

➔ Hardware timestamping on NIC (high accuracy) not impacted by interrupts (high precision)

Measuring Latency—High-Accuracy and High-Precision Timestamping

Three-node setup

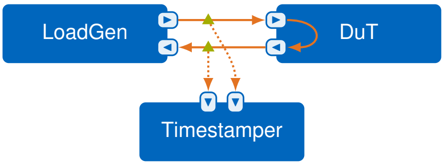
LoadGen

- Flexible software packet generator (MoonGen)
- Bandwidth: Up to 100 Gbit/s or 100 Mpkts/s
- High-precision and high-accuracy generation

Device under Test (DuT)

- Device under test processes packets
- Forwards packets back to LoadGen
- LoadGen analyzes traffic (generated vs. received)

Measuring Latency—High-Accuracy and High-Precision Timestamping



Three-node setup

LoadGen

- Flexible software packet generator (MoonGen)
- Bandwidth: Up to 100 Gbit/s or 100 Mpkts/s
- High-precision and high-accuracy generation

Device under Test (DuT)

- Device under test processes packets
- Forwards packets back to LoadGen
- LoadGen analyzes traffic (generated vs. received)

Timestamper

- LoadGen cannot timestamp all sent packets in hardware (only approx. 1000 pkts/s)
- Specific Intel NICs (e.g., E810) can timestamp all received packets in hardware
- Use passive optical splitters to convert entire traffic to *received* traffic

Low-latency Measurements

Low-Latency Software Stack Design

Problems & Solutions

Reasons for latency impairment

- Interrupt-based IO
  - Linux NAPI
- CPU features
  - Dynamic scheduling of processes onto CPU cores
  - Virtual cores (SMT/Hyperthreading)
  - Energy-saving mechanisms
  - Dynamic cache allocation
- Expensive VM IO

Low-Latency Software Stack Design

Problems & Solutions

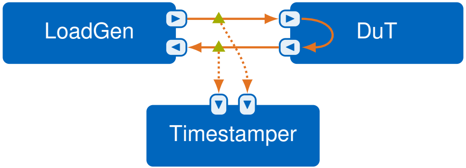
Reasons for latency impairment

- Interrupt-based IO
  - Linux NAPI
- CPU features
  - Dynamic scheduling of processes onto CPU cores
  - Virtual cores (SMT/Hyperthreading)
  - Energy-saving mechanisms
  - Dynamic cache allocation
- Expensive VM IO

Improving latency performance

- Polling-based IO
  - DPDK
- CPU features
  - Statically allocate CPU cores for processes
  - Disable SMT/Hyperthreading
  - Disable energy-saving mechanisms
  - Static cache allocation (Intel CAT)
- NIC acceleration (SR-IOV)

Setup



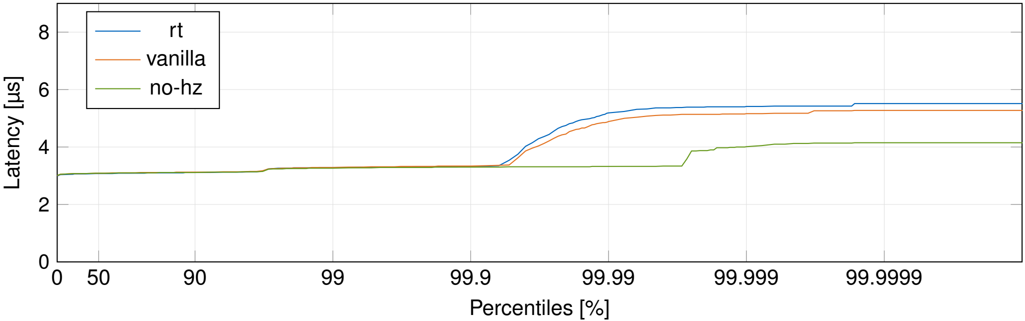
Three-node setup

- Loadgen runs a packet generator (MoonGen) creating UDP packets
- Device under Test (DuT) runs a forwarding application
  - Investigation of different scenarios by modifying the DuT
  - DuT runs a forwarder in different investigated scenarios
- Timestamper records DuT ingress/egress traffic (passive optical TAPs)
  - Hardware-timestamping of entire network traffic (timer resolution 12.5 ns)
- Hardware: Xeon D-1518 (Quad-core, 2.20 GHz), NIC: X557 (10G)
- Traffic: UDP, constant bit rate

Impact of the Linux Kernel

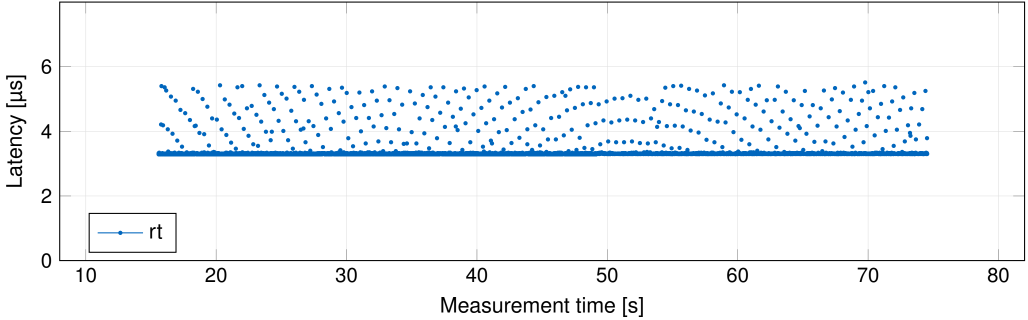
- Linux kernel is offered in different variants
- Two-specific versions of the Linux kernel are optimized to deliver a predictable latency:
  - Realtime kernel
    - Specific kernel patches to deliver consistent latency
  - No-HZ or tickless kernel
    - Disables regular interrupts of the Linux kernel (so-called "tick")
    - Kernel uses the tick to perform housekeeping taks via interrupts (e.g., scheduling)
- The following measurements investigate three different Linux kernels for the DuT
  - rt (realtime) Linux kernel
  - vanilla (unmodified) Linux kernel
  - no-hz (tickless) Linux kernel
- The following measurements use a DPDK-based forwarder

Impact of the Linux Kernel



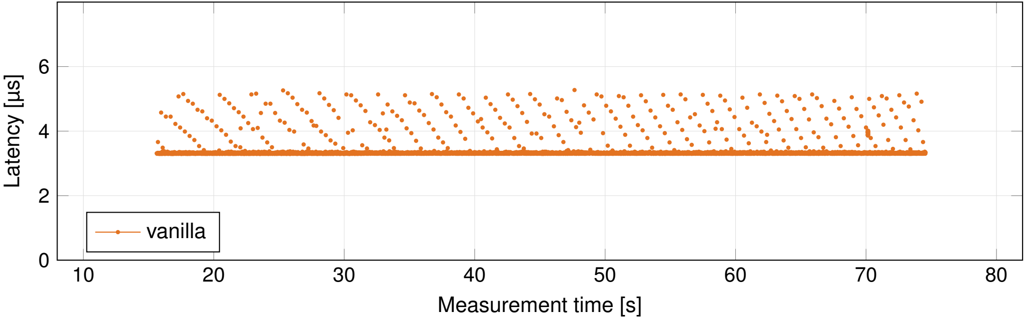
- No measurable differences for percentiles below 99.9
- Stable latency (below 6 µs) is possible for software forwarding even for high percentiles:
  - Similar behavior between realtime and vanilla kernel
  - Lower latency for tickless kernel

Impact of the Linux Kernel



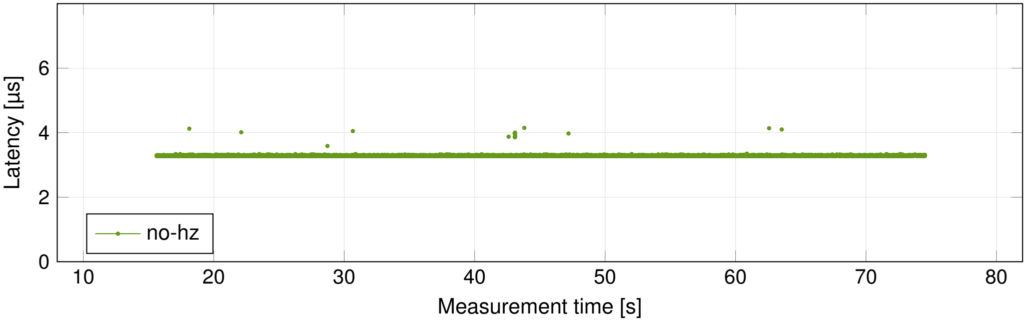
- Two different possibilities for realtime kernel
  - High chance to have stable latency of approx. 3 µs
  - Low chance to be processed during interrupt ("tick") resulting in higher latency of up to 6 µs

Impact of the Linux Kernel



- Two different possibilities for realtime kernel or vanilla kernel:
  - High chance to have stable latency of approx. 3 µs
  - Low chance to be processed during interrupt ("tick") resulting in higher latency of up to 6 µs

Impact of the Linux Kernel

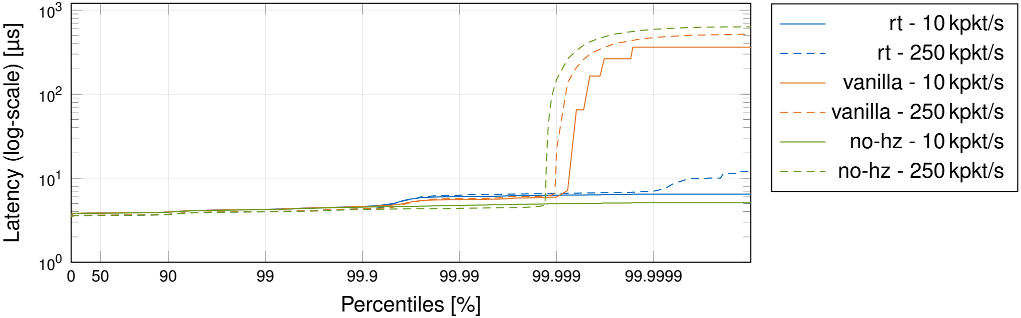


- Two different possibilities for realtime kernel or vanilla kernel:
  - High chance to have stable latency of approx. 3 µs
  - Low chance to be processed during interrupt ("tick") resulting in higher latency of up to 6 µs
- More stable behavior for tickless kernel without interrupts

Impact of the Linux Kernel

- Impact of Linux on other packet processing applications
- The following measurements investigate three different Linux kernels for the DuT
  - vanilla (unmodified) Linux kernel
  - rt (realtime) Linux kernel
  - no-hz (tickless) Linux kernel
- DuT: Suricata an intrusion prevention system using a DPDK-based network stack

Impact of the Linux Kernel

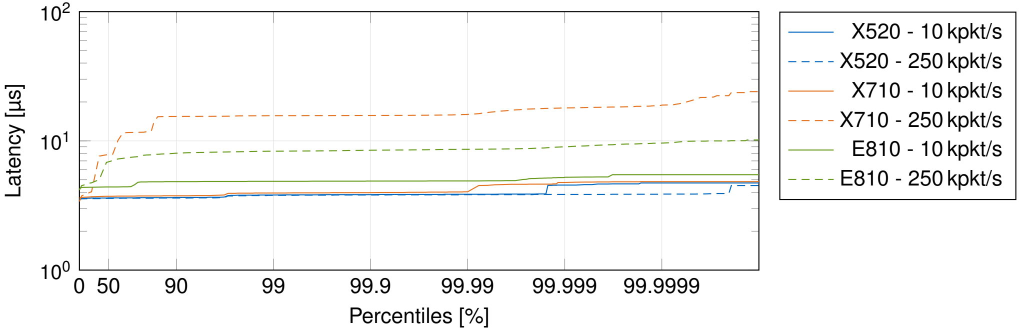


- Significant difference between previous measurement:
  - Vanilla and tickless show similar latency behavior
  - Realtime kernel shows consistently lower performance for high percentiles
- Reason:
  - Tickless kernel only works for single-thread application (otherwise it falls back to vanilla behavior)
  - Realtime kernel offers more consistent performance for multithreaded applications such as Suricata

Impact of the NIC

- Different generations of (Intel) NICs are currently available:
  - X500 generation (up to 10 Gbit/s, released in 2009)
  - X700 generation (up to 40 Gbit/s, released in 2014)
  - E800 generation (up to 100 Gbit/s, released in 2020)
- The following measurements use a DPDK-based forwarder with rates between 10 and 250 kpkt/s

Impact of the NIC

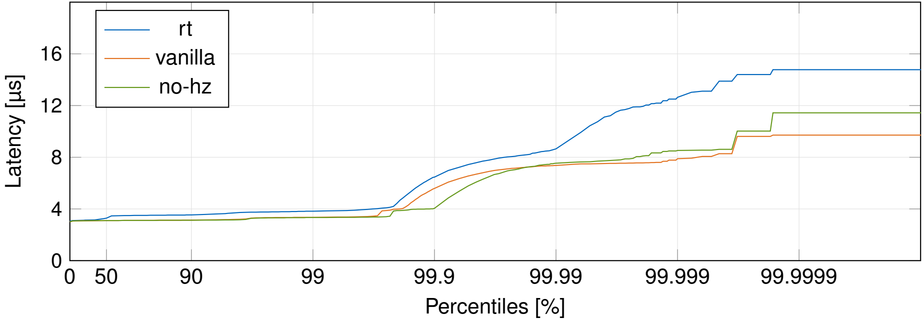


- X500 rather simple architecture, most of the features implemented in hardware, most stable latency
- X700 more complex architecture (more like a switch architecture than NIC), significant latency increase
- E800 complex architecture, more stable latency than previous generation

Impact of Virtualization

- Impact of virtualization on latency
- The following measurements use a DPDK-based forwarder
- SR-IOV is used for a hardware-accelerated network IO of VMs (based on X557 NIC)
- Comparison of vanilla, realtime, and tickless kernel

Impact of Virtualization



- Similar latency performance vanilla and tickless kernel
- Realtime kernel performs slightly worse for high percentiles
- In general, latency in VMs can be close to bare-metal deployments

Conclusion

Measurement methodology

- Measurement methodology is highly relevant to perform effective measurements (especially for latency)
- Hardware support is required for latency measurements

Conclusion

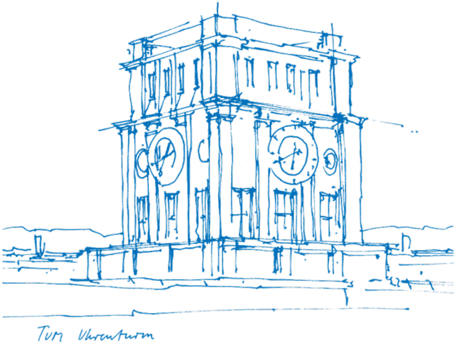
Measurement methodology

- Measurement methodology is highly relevant to perform effective measurements (especially for latency)
- Hardware support is required for latency measurements

Low-latency experiments

- Linux kernel relevant for latency (even if OS stack is not used)
- No clear recommendation which kernel is best, highly depends on the specific scenario:
  - realtime kernel offered the lowest latency for multi-threaded applications
  - tickless kernel offered lowest latency for single-threaded applications
  - vanilla kernel performed best for our VM scenario
- Choice of NIC controller impacts latency
- In our scenario, we observed that older NICs with a simpler architecture offered the best latency

Thank you for listening.



Bibliography

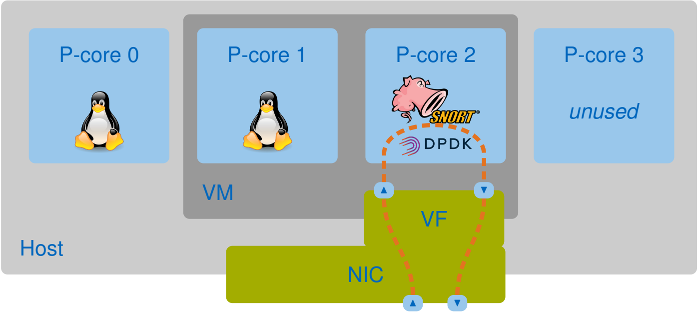
[1] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle. “MoonGen: A Scriptable High-Speed Packet Generator”. In: ACM IMC, Tokyo, Japan, 2015.

[2] S. Gallenmüller, D. Scholz, H. Stubbe, and G. Carle. “The pos framework: a methodology and toolchain for reproducible network experiments”. In: CoNEXT. ACM, 2021.

[3] S. Gallenmüller, F. Wiedner, J. Naab, and G. Carle. “How Low Can You Go? A Limbo Dance for Low-Latency Network Functions”. In: J. Netw. Syst. Manag. 31.1 (2023), p. 20.

Backup

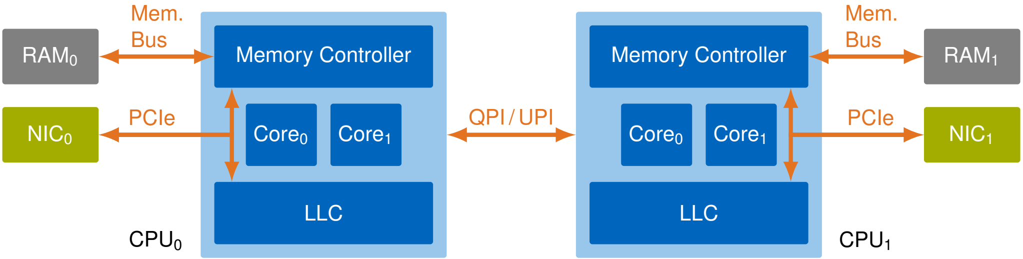
Design



Example setup on a 4-core CPU

- Static pinning: Host OS ➔ p(hysical)-core 0, VM OS ➔ p-core 1, App ➔ p-core 2
- P-core 2 is isolated from scheduling from Host OS & VM OS
- SR-IOV splits NIC into Virtual Functions (VF), one VF exclusively bound to p-core 2

Hardware Architecture



Typical high-level hardware architecture

Typical resources available for packet processing

- Ethernet: 10 Gbit/s to 100 Gbit/s
- PCIe: 32 Gbit/s to 125 Gbit/s (8$\times$ PCIe 2.0 / 4.0)
- Memory bus: 51 Gbit/s to 205 Gbit/s (DDR3-800 / DDR4-3200)
- QPI/UPI: 77 Gbit/s to 166 Gbit/s
- CPU: 2.0 GHz/core to 4.0 GHz/core