Advanced Computer Networking (ACN)

IN2097 – WiSe 2025–2026

Prof. Dr.-Ing. Georg Carle, Sebastian Gallenmüller

Christian Dietze, Marcel Kempf, Lorenz Lehle

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

QoS Measurements

Motivation

Low-latency software stack design

Measurements

Conclusion

Reproducibility

QoS Measurements

Motivation

Low-latency software stack design

Measurements

Conclusion

Reproducibility

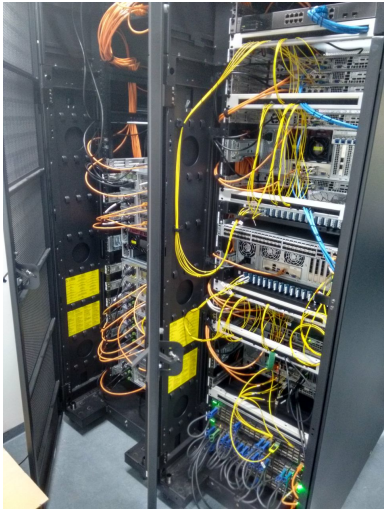
Motivation

Challenge: managing complexity

- Complex hardware architectures (many/multicore)
- Complex software architectures (VMs, container, bare-metal)
- Complex switch & router architectures (SW, FPGA, ASIC)

Selected research activities

- High-performance & high-precision measurement tools
  - Software packet generator MoonGen
  - NetFPGA & optical TAPs
- Reproducible network research
  - Testbed for reproducible network experiments
  - Reproducible Wi-Fi experiments
- Programmable hardware
  - P4-programmable switches (64 $\times$ 100G Ethernet)
  - Programmable network cards
- Performance benchmarking & modeling of networks



High-performance network testbed

Motivation
5G Communication Services

5th generation of mobile networks offering three communication services:

1. eMBB (enhanced Mobile Broadband): high bandwidth service
2. mMTC (massive Machine Type Communications): high number of network nodes (cf. Internet of Things), low power requirements
3. URLLC (Ultra-Reliable Low-Latency Communications): typical use cases: industrial applications, e.g. control processes or self-driving cars

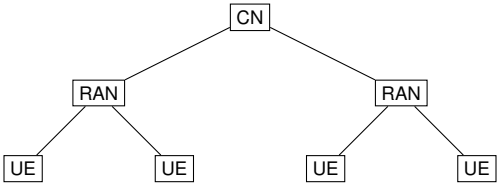
From a QoS perspective URLLC is the most challenging service.

Motivation

5G Ultra-Reliable Low-Latency Communication (URLLC) according to ITU [1]

- ultra reliable: 99.999% success probability
- low latency: 1 ms one-way end-to-end latency in the RAN

Motivation
High-Level Overview: 5G Network Architecture



- UE (user equipment): devices (smartphones) connected to the mobile network
- RAN (radio access network): consists of radio network and fixed network, connection between UE and CN
- CN (core network): core network, connects multiple RANs and other networks (Internet)

The measurements presented investigate network functions located in the fixed network of the RAN.

Motivation
Acknowledgements

Joint collaboration

- Johannes Naab and Georg Carle, TUM
- Iris Adam, Nokia Bell Labs

Motivation
Latency of a VNF chain

VNF scenario

- Virtualized environment (Linux, kvm)
- NF (Snort 3 forwarder, no filtering)
- Latency is measured between ingress and egress of NF

Motivation
Latency of a VNF chain

VNF scenario

- Virtualized environment (Linux, kvm)
- NF (Snort 3 forwarder, no filtering)
- Latency is measured between ingress and egress of NF

Results

percentiles	50th	99th	99.9th	99.99th	99.999th
Snort 3 forwarder	69 µs	88 µs	107 µs	1.7 ms	2.5 ms

➔ 99.99th percentile already violates URLLC

Latency of a VNF chain: Zoom in

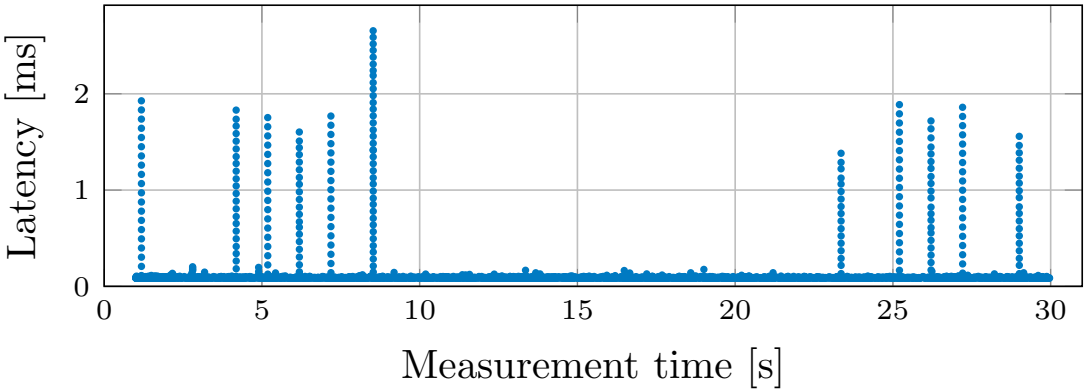


Figure 1: Snort 3 forwarder worst-case latencies

→ **URLLC violations happen irregularly over the entire measurement**

QoS Measurements

Motivation

Low-latency software stack design

Measurements

Conclusion

Reproducibility

Low-latency software stack design
Problems & solutions

Reasons for VNF performance impairment

- Interupt-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

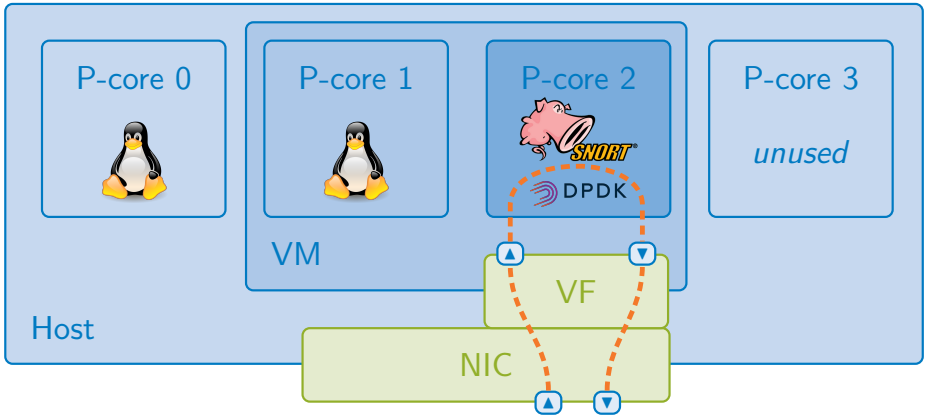
Reasons for VNF performance impairment

- Interupt-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

Fixing VNF 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)

Low-latency software stack design
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

QoS Measurements

Motivation

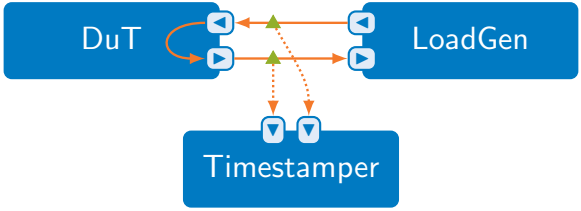
Low-latency software stack design

Measurements

Conclusion

Reproducibility

Measurements
Setup



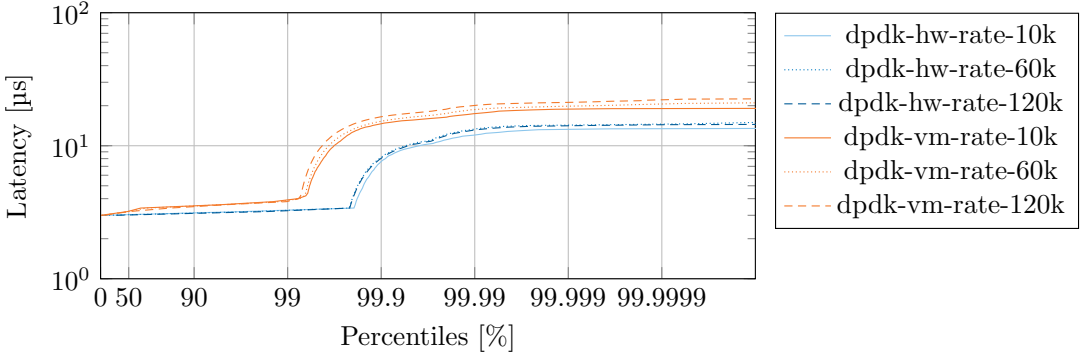
Measurement setup

- Loadgen runs a packet generator (MoonGen) creating UDP packets
- Device under Test (DuT) runs a forwarding application
- Timestamper records DuT ingress/egress traffic (passive optical TAPs)
  - Hardware-timestamping of entire network traffic (timer resolution 12.5 ns)
  - Determine worst-case latencies to check for URLLC delay violations
- Hardware: Xeon D-1518 (Quad-core, 2.20 GHz), NIC: X557 (10G)
- Traffic: UDP resembling DNS (dst port 53), constant bit rate

Measurements
Measuring forwarding latency

- 3 different forwarders:
  - DPDK-l2fwd: minimal forwarder to determine IO overhead
  - Snort-fwd & DPDK: minimal Snort 3 forwarder to determine Snort overhead
  - Snort-filter & DPDK: forwarder with filter rules to determine cost of rule application
- 2 differrent deployments:
  - Bare-metal deployment (HW): determine base-level performance
  - Virtual machine (VM): determine costs of virtualization

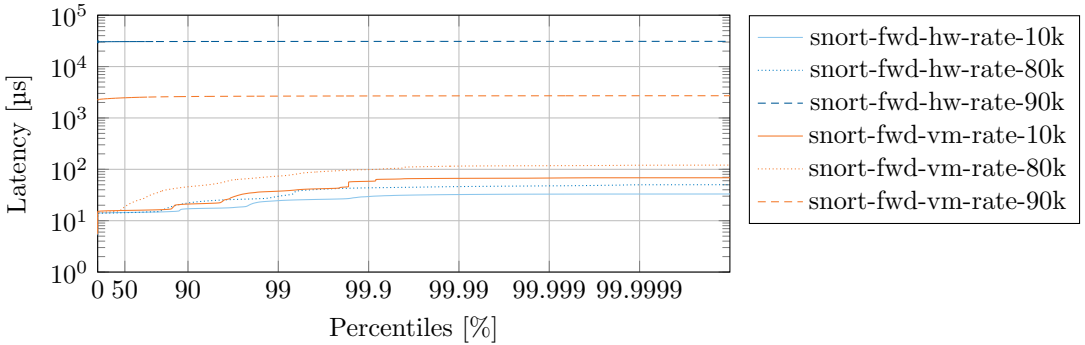
Measurements
DPDK-l2fwd



- **Differences between packet rates:** Almost no difference, low latency, no overload
- **Differences between HW and VM:** roughly 4 µs difference, no overload

➔ Latency below 1 ms

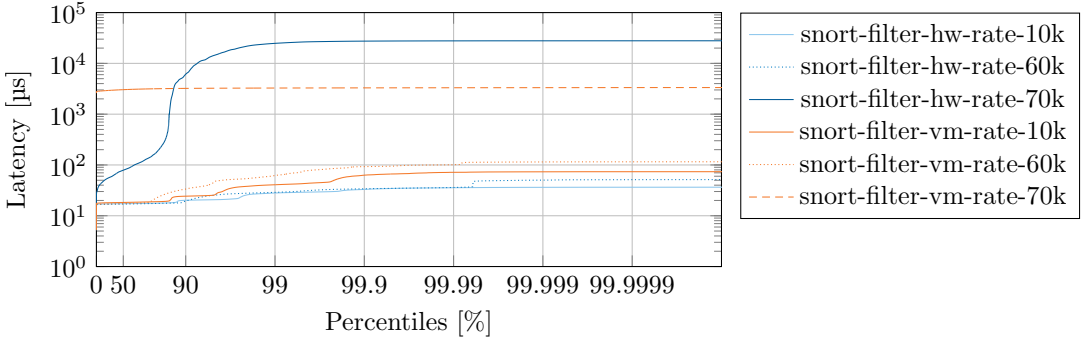
Measurements
Snort forwarder (on DPDK)



- **Differences between packet rates:** Overload between 80 and 90 kpkt/s
- **Overload for HW and VM:** 30 ms and 3 ms
- Only minor differences for non-overloaded scenarios between HW and VM (up to 50 µs)

➔ Latency below 1 ms without overload

Measurements
Snort filter (on DPDK)



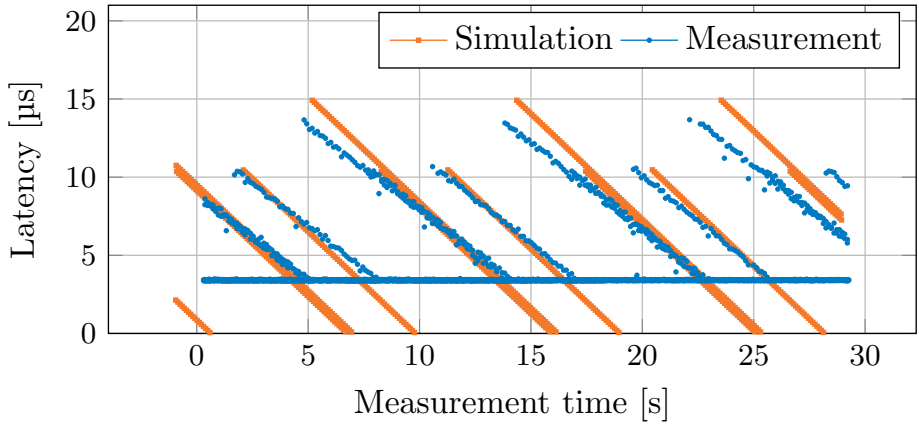
- **Differences between packet rates:** Overload between 60 and 70 kpkt/s
- **Overload for HW and VM:** 30 ms and 3 ms
- Only minor differences for non-overloaded scenarios between HW and VM (up to 50 µs)

→ Latency below 1 ms without overload

Measurements
Measuring tail latencies

- DPDK-l2fwd: most stable and therefore cleanest measurement of IO
- Investigation:
  - When and why does the latency increase?

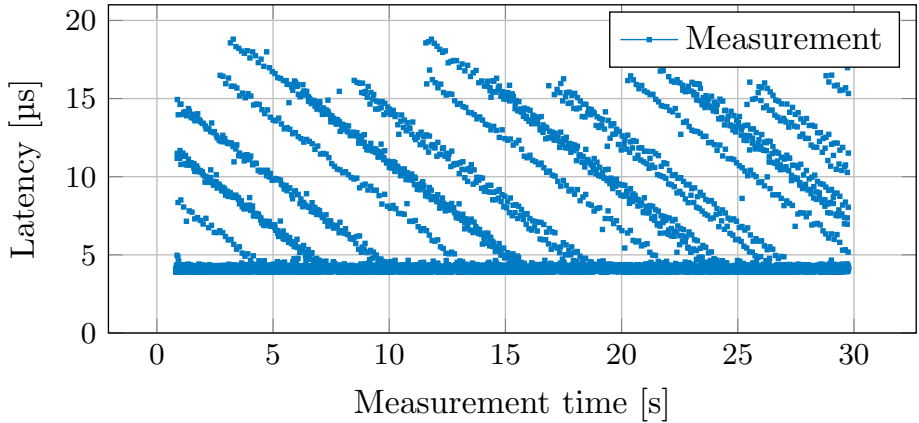
Measuring tail latencies (HW)



- Delay happens regularily in certain patterns
- Explanation: CBR traffic tries to *sample* OS interrupts (undersampling)

→ OS interrupts of different length (10 and 13 µs) are responsible for tail latencies

Measuring tail latencies (VM)

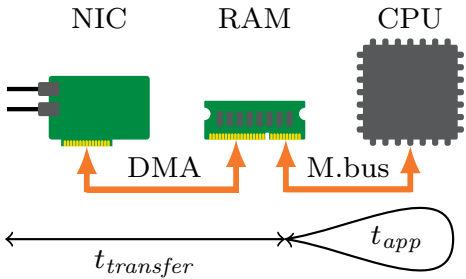


- 2 × interrupts (Host OS & VM OS)
- Interrupts take longer, approx. 5 µs

Measurements
Overload prediction

Prediction

- Latency under system overload violates URLLC QoS
- ➔ Predict system capacity to help preventing overload scenarios
- ➔ CPU is the typically the limiting factor for packet processing [2]



Sources of packet delay in a software packet processing system

- We measure the end-to-end delay ($t_{end2end}$) per packet:
  - Transfer delay ($t_{transfer}$), including propagation, serialization, system busses
  - Application delay (t_{app})
- Transfer delay ($t_{transfer}$) is not overloaded (at investigated packet rates)
- CPU (t_{app}) limiting factor

➔ How to determine t_{app}?

Measurements

How to determine t_{app}?

- $t_{end2end}$: measurements
- DPDK-l2fwd, the basic forwarder, has almost no processing delay
- ➔ Use DPDK-l2fwd measurement as approximation for $t_{transfer}$
- ➔ Calculate t_{app} for Snort scenarios

Interrupt processing

- CPU time needs to be spent on interrupt processing
- ➔ Calculate time spent on interrupts (d_Σ) from our measured interrupt rates and interrupt execution times

Maximum packet rate per second (R_{max}):

$$R_{max} = \frac{1\,\text{s} - d_\Sigma}{t_{app}}$$

Measurements

Table 1: Calculated CPU times and maximum rate

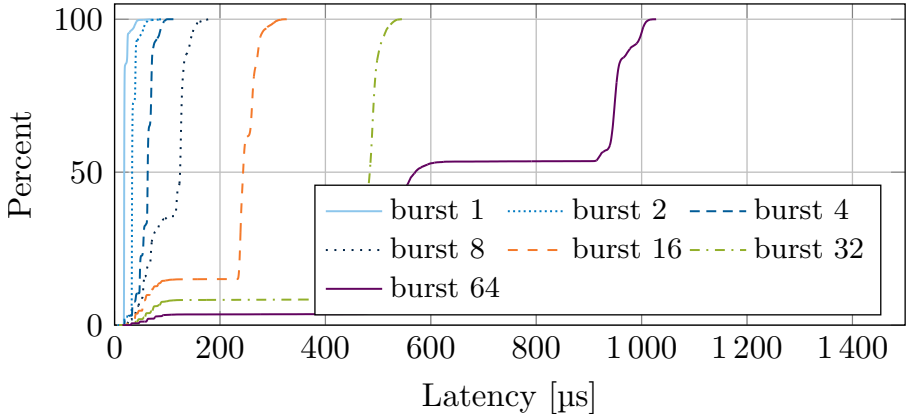
		DPDK-l2fwd $2t_{transfer}$ [µs]	Med. latency $t_{end2end}$ [µs]	CPUtime t_{app} [µs]	Interrupt time d_{Σ} [µs]	Max. Rate R_{max} [kpkts/s]
Snort-fwd	HW	3.1	14.5	11.4	2949.9	87.4
	VM	3.3	15.9	12.6	8891.6	78.7
Snort-filter	HW	3.1	17.4	14.3	2949.9	69.7
	VM	3.3	18.4	15.1	8891.6	65.6

Model

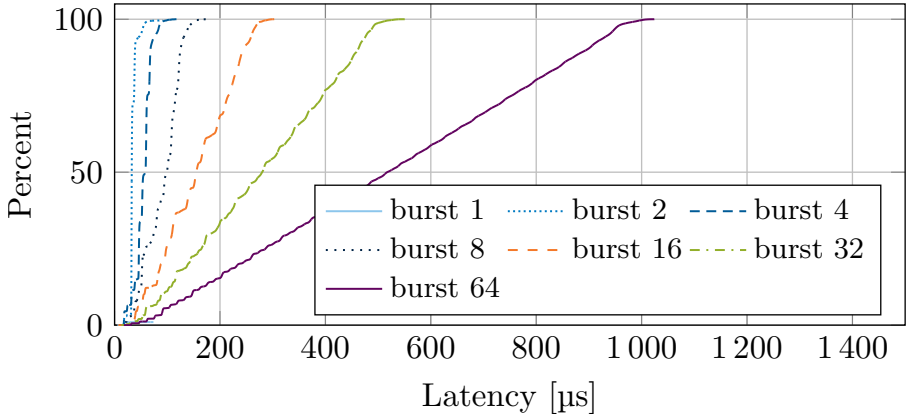
- Conservative approximation (real R_{max} slightly higher)
- Required input for model:
  - Median latency (basic scenario)
  - Median latency (investigated scenario)
  - Interrupt processing time

Measurements
Impact of bursts

- Bursts of packets impact processing performance
  - Burst: sequence of packets with no gap in between
- Application processes batches of packets
  - Batch: sequence of packets an application accepts and processes internally
  - Maximum allowed batch size by default 32
  - Smaller batches are allowed
- Investigated parameters:
  - Snort-filter
  - Packet rate: 10 000 packets per second
  - Burst sizes: 1 (all previous measurements), 2, 4, 8, 16, 32, 64
  - Batch sizes: 4-batch, 32-batch (default)



➔ Burst sizes $\geq$ 16 in combination with blocking behavior of 32-batch increase median latency signficantly

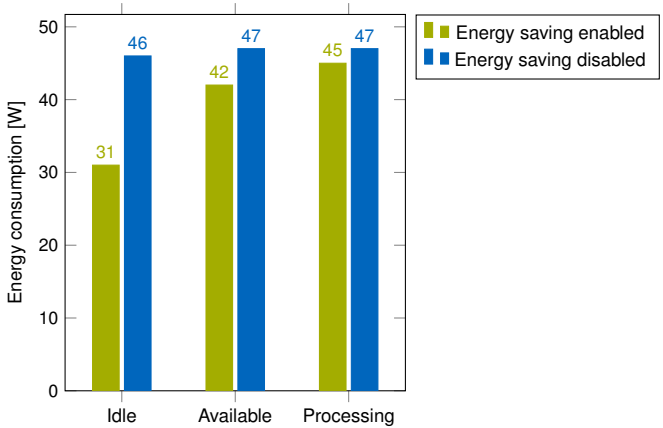


➔ Burst sizes $\geq$ 16 in combination with minimum batch size of 4 lowers median latency significantly

Measurements
Energy consumption

- CPU energy-saving features were disabled to avoid delays
- Energy consumption of whole server measured in three scenarios:
  - Idle: application not started, no packet transfer
  - Available: application started, no packet transfer
  - Processing: application started forwarding packets

Measurements
Energy consumption



- For idle systems: high influence, up to 50%
- Under high load: Little influence, up to 4%

QoS Measurements

Motivation

Low-latency software stack design

Measurements

Conclusion

Reproducibility

Conclusion

- 1 ms latency goal includes the entire RAN

➔ even stricter limits for network functions, e.g. 350 µs [3]

- 99.999th percentile 114.7 µs for Snort-filter

➔ URLLC is possible on off-the-shelf hardware if done correctly

- Limitations:
  - Large bursts may violate latency goals
  - Energy consumption may raise up to 50%
  - Static allocation of CPU cores disallows CPU sharing between VMs
- Full paper available: https://arxiv.org/pdf/1909.08397.pdf

QoS Measurements

Motivation

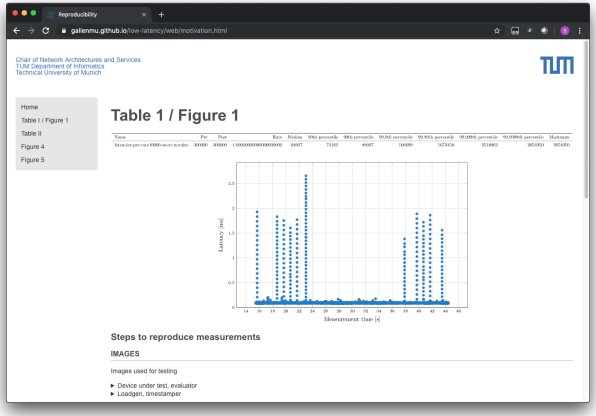
Low-latency software stack design

Measurements

Conclusion

Reproducibility

Reproducibility
Reproduce our measurements



- Measurement tools/experiment scripts: https://gallenmu.github.io/low-latency
- Measurements data: https://github.com/gallenmu/low-latency

Reproducibility

[1] ITU, Report ITU-R M.2410-0 (11/2017) Minimum requirements related to technical performance for IMT-2020 radio interface(s), https://www.itu.int/dms_pub/itu-r/opb/rep/R-REP-M.2410-2017-PDF-E.pdf, Accessed: 2019-05-30.

[2] S. Gallenmüller, P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle, "Comparison of Frameworks for High-Performance Packet IO," in ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS 2015), 2015.

[3] Z. Xiang, F. Gabriel, E. Urbano, G. T. Nguyen, M. Reisslein, and F. H. Fitzek, "Reducing Latency in Virtual Machines: Enabling Tactile Internet for Human-Machine Co-Working," IEEE Journal on Selected Areas in Communications, vol. 37, no. 5, pp. 1098–1116, 2019.