Advanced Computer Networking (ACN)

IN2097 – WiSe 2019-2020

Prof. Dr.-Ing. Georg Carle

Sebastian Gallenmüller, Max Helm,
Benedikt Jaeger, Patrick Sattler, Johannes Zirngibl

Chair of Network Architectures and Services
Department of Informatics
Technical University of Munich
Chapter 10: QoS Measurements

Motivation

Low-latency software stack design

Measurements

Conclusion

Reproducibility

Backup Slides
Chapter 10: QoS Measurements

Motivation

Low-latency software stack design

Measurements

Conclusion

Reproducibility

Backup Slides
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 × 100G Ethernet)
  - Programmable network cards
- Performance benchmarking & modeling of networks

High-performance network testbed
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.
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

<table>
<thead>
<tr>
<th>percentiles</th>
<th>50th</th>
<th>99th</th>
<th>99.9th</th>
<th>99.99th</th>
<th>99.999th</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snort 3 forwarder</td>
<td>69 µs</td>
<td>88 µs</td>
<td>107 µs</td>
<td>1.7 ms</td>
<td>2.5 ms</td>
</tr>
</tbody>
</table>

→ 99.99th percentile already violates URLLC
Figure 1: Snort 3 forwarder worst-case latencies

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

Low-latency software stack design

Measurements

Conclusion

Reproducibility

Backup Slides
Reasons for VNF performance 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 VNF performance 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

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(ysical)-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
Chapter 10: QoS Measurements

Motivation

Low-latency software stack design

Measurements

Conclusion

Reproducibility

Backup Slides
Measurements

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 different deployments:
  • Bare-metal deployment (HW): determine base-level performance
  • Virtual machine (VM): determine costs of virtualization
• **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
• **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
• **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-I2fwd: most stable and therefore cleanest measurement of IO
- Investigation:
  - When and why does the latency increase?
Measurements
Measuring tail latencies (HW)

• Delay happens regularly 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
Measurements
Measuring tail latencies (VM)

\[ 2 \times \text{interrupts (Host OS & VM OS)} \]
\[ \text{Interrupts take longer, approx. } 5\mu s \]
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 ($t_{end\text{2end}}$) 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 \( (d_\Sigma) \) 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

Figure 2: Calculated CPU times and maximum rate

<table>
<thead>
<tr>
<th></th>
<th>DPDK-l2fwd</th>
<th>Med. latency</th>
<th>CPUtime</th>
<th>Interrupt time</th>
<th>Max. Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$2t_{transfer}$</td>
<td>$t_{end2end}$</td>
<td>$t_{app}$</td>
<td>$\Sigma d$</td>
<td>$R_{max}$</td>
</tr>
<tr>
<td>Snort-fwd</td>
<td>HW</td>
<td>3.1</td>
<td>14.5</td>
<td>11.4</td>
<td>2949.9</td>
</tr>
<tr>
<td></td>
<td>VM</td>
<td>3.3</td>
<td>15.9</td>
<td>12.6</td>
<td>8891.6</td>
</tr>
<tr>
<td>Snort-filter</td>
<td>HW</td>
<td>3.1</td>
<td>17.4</td>
<td>14.3</td>
<td>2949.9</td>
</tr>
<tr>
<td></td>
<td>VM</td>
<td>3.3</td>
<td>18.4</td>
<td>15.1</td>
<td>8891.6</td>
</tr>
</tbody>
</table>

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)
Measurements

Impact of bursts: 32-batch processing

Burst sizes $\geq 16$ in combination with blocking behavior of 32-batch increase median latency significantly
Measurements
Impact of bursts: 4-batch processing

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%
Chapter 10: QoS Measurements

Motivation

Low-latency software stack design

Measurements

Conclusion

Reproducibility

Backup Slides
• 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
Chapter 10: QoS Measurements

Motivation

Low-latency software stack design

Measurements

Conclusion

Reproducibility

Backup Slides
Reproducibility
Reproduce our measurements

- Measurement tools/experiment scripts: https://gallenmu.github.io/low-latency
- Measurements data: https://github.com/gallenmu/low-latency
Chapter 10: QoS Measurements


Chapter 10: QoS Measurements

Motivation

Low-latency software stack design

Measurements

Conclusion

Reproducibility

Backup Slides
Latencies for bare-metal forwarders

<table>
<thead>
<tr>
<th>Mode</th>
<th>Rate</th>
<th>Loss</th>
<th>n-th percentiles [µs]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[kpkts/s]</td>
<td>[%]</td>
<td>50</td>
</tr>
<tr>
<td><strong>DPDK-l2fwd</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW</td>
<td>10</td>
<td>-</td>
<td>3.1</td>
</tr>
<tr>
<td>HW</td>
<td>60</td>
<td>-</td>
<td>3.1</td>
</tr>
<tr>
<td>HW</td>
<td>120</td>
<td>-</td>
<td>3.1</td>
</tr>
<tr>
<td><strong>Snort-fwd &amp; DPDK</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW</td>
<td>10</td>
<td>0.1</td>
<td>14.4</td>
</tr>
<tr>
<td>HW</td>
<td>80</td>
<td>3.3</td>
<td>30 609.5</td>
</tr>
<tr>
<td><strong>Snort-filter &amp; DPDK</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW</td>
<td>10</td>
<td>0.0</td>
<td>17.1</td>
</tr>
<tr>
<td>HW</td>
<td>60</td>
<td>0.0</td>
<td>79.0</td>
</tr>
</tbody>
</table>
## Latencies for bare-metal forwarders

<table>
<thead>
<tr>
<th>Mode</th>
<th>Rate</th>
<th>Loss</th>
<th>n-th percentiles [µs]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[kpkts/s]</td>
<td>[%]</td>
<td>50</td>
</tr>
<tr>
<td><strong>DPDK-l2fwd</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW 10</td>
<td>10</td>
<td>-</td>
<td>3.1</td>
</tr>
<tr>
<td>HW 60</td>
<td>60</td>
<td>-</td>
<td>3.1</td>
</tr>
<tr>
<td>HW 120</td>
<td>120</td>
<td>-</td>
<td>3.1</td>
</tr>
<tr>
<td><strong>Snort-fwd &amp; DPDK</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW 10</td>
<td>80</td>
<td>0.1</td>
<td>14.4</td>
</tr>
<tr>
<td>HW 10</td>
<td>90</td>
<td>3.3</td>
<td>30 609.5</td>
</tr>
<tr>
<td><strong>Snort-filter &amp; DPDK</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW 10</td>
<td>60</td>
<td>0.0</td>
<td>17.1</td>
</tr>
<tr>
<td>HW 70</td>
<td>70</td>
<td>0.0</td>
<td>79.0</td>
</tr>
</tbody>
</table>

- **Overload:** delay ≥ 30 ms
Latencies for bare-metal forwarders

<table>
<thead>
<tr>
<th>Mode</th>
<th>Rate</th>
<th>Loss</th>
<th>50</th>
<th>99</th>
<th>99.9</th>
<th>99.99</th>
<th>99.999</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[kpkts/s]</td>
<td>[%]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DPDK-l2fwd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW 10</td>
<td>10</td>
<td>-</td>
<td>3.1</td>
<td>3.4</td>
<td>7.7</td>
<td>12.2</td>
<td>13.4</td>
</tr>
<tr>
<td>HW 60</td>
<td>60</td>
<td>-</td>
<td>3.1</td>
<td>3.3</td>
<td>8.3</td>
<td>13.5</td>
<td>14.4</td>
</tr>
<tr>
<td>HW 120</td>
<td>120</td>
<td>-</td>
<td>3.1</td>
<td>3.3</td>
<td>8.1</td>
<td>13.2</td>
<td>14.3</td>
</tr>
<tr>
<td>Snort-fwd &amp; DPDK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW 10</td>
<td>10</td>
<td>-</td>
<td>14.5</td>
<td>24.7</td>
<td>29.7</td>
<td>32.4</td>
<td>33.1</td>
</tr>
<tr>
<td>HW 80</td>
<td>80</td>
<td>0.1</td>
<td>14.4</td>
<td>29.9</td>
<td>43.7</td>
<td>46.2</td>
<td>47.7</td>
</tr>
<tr>
<td>HW 90</td>
<td>90</td>
<td>3.3</td>
<td>30 609.5</td>
<td>30 834.8</td>
<td>30 882.7</td>
<td>30 915.3</td>
<td>30 936.1</td>
</tr>
<tr>
<td>Snort-filter &amp; DPDK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW 10</td>
<td>10</td>
<td>-</td>
<td>17.4</td>
<td>28.2</td>
<td>33.1</td>
<td>35.8</td>
<td>36.4</td>
</tr>
<tr>
<td>HW 60</td>
<td>60</td>
<td>0.0</td>
<td>17.1</td>
<td>29.0</td>
<td>34.1</td>
<td>36.1</td>
<td>50.4</td>
</tr>
<tr>
<td>HW 70</td>
<td>70</td>
<td>0.0</td>
<td>79.0</td>
<td>24 897.2</td>
<td>27 521.2</td>
<td>27 847.0</td>
<td>27 947.1</td>
</tr>
</tbody>
</table>

- Overload: delay $\geq 30$ ms
- 99.999-th percentile delay: 50.4 $\mu$s

.URLLLC processing on HW is possible, as long as system is not overloaded.
## Latencies for VM forwarders

<table>
<thead>
<tr>
<th>Mode</th>
<th>Rate</th>
<th>Loss</th>
<th>50</th>
<th>99</th>
<th>99.9</th>
<th>99.99</th>
<th>99.999</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPDK-l2fwd</td>
<td>VM</td>
<td>10</td>
<td>3.3</td>
<td>4.0</td>
<td>14.7</td>
<td>17.6</td>
<td>19.0</td>
</tr>
<tr>
<td></td>
<td>VM</td>
<td>60</td>
<td>3.3</td>
<td>4.0</td>
<td>15.4</td>
<td>18.9</td>
<td>19.9</td>
</tr>
<tr>
<td></td>
<td>VM</td>
<td>120</td>
<td>3.3</td>
<td>3.9</td>
<td>16.6</td>
<td>20.2</td>
<td>21.3</td>
</tr>
<tr>
<td>Snort-fwd &amp; DPDK</td>
<td>VM</td>
<td>10</td>
<td>15.9</td>
<td>37.6</td>
<td>58.6</td>
<td>66.8</td>
<td>68.0</td>
</tr>
<tr>
<td></td>
<td>VM</td>
<td>80</td>
<td>18.8</td>
<td>73.9</td>
<td>98.8</td>
<td>115.6</td>
<td>117.7</td>
</tr>
<tr>
<td></td>
<td>VM</td>
<td>90</td>
<td>2469.6</td>
<td>2657.9</td>
<td>2679.8</td>
<td>2692.2</td>
<td>2700.6</td>
</tr>
<tr>
<td>Snort-filter &amp; DPDK</td>
<td>VM</td>
<td>10</td>
<td>18.4</td>
<td>40.9</td>
<td>63.1</td>
<td>71.8</td>
<td>73.4</td>
</tr>
<tr>
<td></td>
<td>VM</td>
<td>60</td>
<td>17.5</td>
<td>62.2</td>
<td>92.9</td>
<td>101.1</td>
<td>114.7</td>
</tr>
<tr>
<td></td>
<td>VM</td>
<td>70</td>
<td>3036.9</td>
<td>3270.2</td>
<td>3294.4</td>
<td>3313.1</td>
<td>3326.8</td>
</tr>
</tbody>
</table>

- **Overload:** delay $\geq 3$ ms
- **99.999th percentile delay:** 114.7 µs (+100% compared to HW): URLLC processing in VMs is possible, as long as the system is not overloaded.
Latencies for VM forwarders

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DPDK-l2fwd VM 10</td>
<td>10</td>
<td>-</td>
<td>3.3</td>
<td>4.0</td>
<td>14.7</td>
<td>17.6</td>
<td>19.0</td>
</tr>
<tr>
<td>VM 60</td>
<td>3.3</td>
<td>4.0</td>
<td>15.4</td>
<td>18.9</td>
<td>19.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VM 120</td>
<td>3.3</td>
<td>3.9</td>
<td>16.6</td>
<td>20.2</td>
<td>21.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Snort-fwd &amp; DPDK VM 10</td>
<td>15.9</td>
<td>37.6</td>
<td>58.6</td>
<td>66.8</td>
<td>68.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VM 80</td>
<td>0.1</td>
<td>18.8</td>
<td>73.9</td>
<td>98.8</td>
<td>115.6</td>
<td>117.7</td>
<td></td>
</tr>
<tr>
<td>VM 70</td>
<td>3.0</td>
<td>3036.9</td>
<td>3270.2</td>
<td>3294.4</td>
<td>3313.1</td>
<td>3326.8</td>
<td></td>
</tr>
<tr>
<td>Snort-filter &amp; DPDK VM</td>
<td>10</td>
<td>-</td>
<td>18.4</td>
<td>40.9</td>
<td>63.1</td>
<td>71.8</td>
<td>73.4</td>
</tr>
<tr>
<td>VM 60</td>
<td>0.1</td>
<td>17.5</td>
<td>62.2</td>
<td>92.9</td>
<td>101.1</td>
<td>114.7</td>
<td></td>
</tr>
<tr>
<td>VM 70</td>
<td>3.0</td>
<td>3036.9</td>
<td>3270.2</td>
<td>3294.4</td>
<td>3313.1</td>
<td>3326.8</td>
<td></td>
</tr>
</tbody>
</table>

- Overload: delay ≥ 3 ms
### Latencies for VM forwarders

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DPDK-l2fwd</td>
<td>VM 10</td>
<td>-</td>
<td>3.3</td>
<td>4.0</td>
<td>14.7</td>
<td>17.6</td>
<td>19.0</td>
</tr>
<tr>
<td></td>
<td>VM 60</td>
<td>-</td>
<td>3.3</td>
<td>4.0</td>
<td>15.4</td>
<td>18.9</td>
<td>19.9</td>
</tr>
<tr>
<td></td>
<td>VM 120</td>
<td>-</td>
<td>3.3</td>
<td>3.9</td>
<td>16.6</td>
<td>20.2</td>
<td>21.3</td>
</tr>
<tr>
<td>Snort-fwd &amp; DPDK</td>
<td>VM 10</td>
<td>-</td>
<td>15.9</td>
<td>37.6</td>
<td>58.6</td>
<td>66.8</td>
<td>68.0</td>
</tr>
<tr>
<td></td>
<td>VM 80</td>
<td>0.1</td>
<td>18.8</td>
<td>73.9</td>
<td>98.8</td>
<td>115.6</td>
<td>117.7</td>
</tr>
<tr>
<td></td>
<td>VM 90</td>
<td>7.1</td>
<td>2469.6</td>
<td>2657.9</td>
<td>2679.8</td>
<td>2692.2</td>
<td>2700.6</td>
</tr>
<tr>
<td>Snort-filter &amp; DPDK</td>
<td>VM 10</td>
<td>-</td>
<td>18.4</td>
<td>40.9</td>
<td>63.1</td>
<td>71.8</td>
<td>73.4</td>
</tr>
<tr>
<td></td>
<td>VM 60</td>
<td>0.1</td>
<td>17.5</td>
<td>62.2</td>
<td>92.9</td>
<td>101.1</td>
<td>114.7</td>
</tr>
<tr>
<td></td>
<td>VM 70</td>
<td>3.0</td>
<td>3036.9</td>
<td>3270.2</td>
<td>3294.4</td>
<td>3313.1</td>
<td>3326.8</td>
</tr>
</tbody>
</table>

- **Overload**: delay ≥ 3 ms
- **99.999th percentile delay**: 114.7 µs (+100% compared to HW)

→ URLLC processing in VMs is possible, as long as system is not overloaded