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
Announcement

- Thursday, Feb.6, last lecture before exam
- If you have any questions please send an e-mail to acn@net.in.tum.de
- If we do not receive any questions by tomorrow 12am there will be no lecture on Thursday
- Deregister for the endterm exam if you are sure that you will not participate
- Registration for retake exam does not depend on the registration for the endterm
P4 Benchmarking & Wireless Measurements

P4 Benchmark

Motivation

Related Work - Whippersnapper

P4 NetFPGA Benchmark

Result Summary

Conclusion

Reproducible Wireless Measurements

Motivation

Repeatability, Replicability, and Reproducibility

Investigation of Wireless Channel Behavior

Bibliography
P4 Benchmarking & Wireless Measurements

P4 Benchmark

Motivation

Related Work - Whippersnapper

P4 NetFPGA Benchmark

Result Summary

Conclusion

Reproducible Wireless Measurements

Bibliography
Motivation

Benchmarks

- RFC 2544 – Benchmarking Methodology for Network Interconnect Devices
- RFC 3511 – Benchmarking Methodology for Firewall Performance
- RFC 8239 – Data Center Benchmarking Methodology
- ...
- Should we really do another benchmark?
Whippersnapper

- paper and P4\textsubscript{14} benchmark suite by Dang et al. [1] published in 2017
- synthetic benchmark (artificial traffic/P4 programs)
- discusses possible comparison parameters
  - target independent – P4 language features
  - target dependent – CPU, NPU, FPGA, ASIC
- compares benchmark of P4\textsubscript{14} programs
  - P4 programs tailored to measure a specific P4 feature are executed
  - traffic is sent to P4 device and performance is measured
## Target Independent Comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parsing</td>
<td>#Packet headers, #Packet fields, #Branches in parse graph</td>
</tr>
<tr>
<td>Processing</td>
<td>#Tables (no dependencies), Depth of pipeline, Checksum on/off, Size of tables</td>
</tr>
<tr>
<td>State Accesses</td>
<td>#Writes to different register, #Writes to same register, #Reads to different register, #Reads to same register</td>
</tr>
<tr>
<td>Packet Modification</td>
<td>#Header adds, #Header removes</td>
</tr>
<tr>
<td>Action Complexity</td>
<td>#Field writes, #Arithmetic expressions, #Boolean expressions</td>
</tr>
</tbody>
</table>
### Target Dependent Comparison

<table>
<thead>
<tr>
<th>Target</th>
<th>Metric(s)</th>
<th>Parameter</th>
<th>Measurement Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Latency, Throughput</td>
<td>Changing working set of flows quickly</td>
<td>Runtime</td>
</tr>
<tr>
<td></td>
<td>Latency, Throughput</td>
<td>Changing size of flows</td>
<td>Runtime</td>
</tr>
<tr>
<td></td>
<td>Latency, Throughput</td>
<td>Read, modify, update same register</td>
<td>Runtime</td>
</tr>
<tr>
<td>NPU</td>
<td>Latency, Throughput</td>
<td>Changing working set of flows quickly</td>
<td>Runtime</td>
</tr>
<tr>
<td></td>
<td>Latency, Throughput</td>
<td>Changing size of flows</td>
<td>Runtime</td>
</tr>
<tr>
<td></td>
<td>Latency, Throughput</td>
<td>Read, modify, update same register</td>
<td>Runtime</td>
</tr>
<tr>
<td>FPGA</td>
<td>Does it fit?</td>
<td>#Tables</td>
<td>Compile-time</td>
</tr>
<tr>
<td></td>
<td>Does it fit?</td>
<td>Table size</td>
<td>Compile-time</td>
</tr>
<tr>
<td>ASIC</td>
<td>Does it fit?</td>
<td>#Tables</td>
<td>Compile-time</td>
</tr>
<tr>
<td></td>
<td>Does it fit?</td>
<td>Table size</td>
<td>Compile-time</td>
</tr>
<tr>
<td></td>
<td>Does it fit?</td>
<td>#Depth of dependency in expression</td>
<td>Compile-time</td>
</tr>
</tbody>
</table>
Related Work - Whippersnapper

Results

Target Independent Results

<table>
<thead>
<tr>
<th>Target</th>
<th>Parse single header</th>
<th>Write single field</th>
<th>Single table application</th>
</tr>
</thead>
<tbody>
<tr>
<td>bmv2</td>
<td>11.2 ms</td>
<td>11.1 ms</td>
<td>14.4 ms</td>
</tr>
<tr>
<td>PISCES</td>
<td>5.2 µs</td>
<td>5.5 µs</td>
<td>4.8 µs</td>
</tr>
<tr>
<td>P4FPGA</td>
<td>0.3 µs</td>
<td>0.4 µs</td>
<td>0.4 µs</td>
</tr>
</tbody>
</table>

• Basic two-server setup:
  • server 1: load generator
  • server 2: device under test (DuT, running l2 forwarder)
  • direct connection via 10 G Ethernet

• Three different targets as DuT:
  • bmw2: P4 software switch
  • PISCES: P4 software switch, based on Open vSwitch and DPDK
  • P4FPGA: FPGA-based P4 compiler

Figure 1: Basic two-server setup
Related Work - Whippersnapper

Current state

- code available github.com
- no update since May 2017
- no P4\textsubscript{16} support
- relying on slow packet generator of scapy (not fit for 10G Ethernet)

How to solve this situation?

1. find some student
2. let him read wippersnapper paper
3. collaboration for 6 months . . .

The following slides are based on a Master's Thesis by Henning Stubbe.
What to benchmark?

- **Parser:**
  - header stack height
  - header fields
  - header field size
  - number of next header

- **Packet Modification:**
  - header removal
  - header insertion
  - header field modification

- **Processing:** table application

- **State Access:** read/write
P4 NetFPGA Benchmark
P4-Target: NetFPGA SUME and P4-NetFPGA

SUME

- FPGA-based PCIe card; 10 Gbit/s SFP+ Ethernet
- developed by NetFPGA project [2]
- designed as prototyping platform

P4-NetFPGA

- synthesize P4 programs for SUME
- P4_{16} architecture: SimpleSumeSwitch
- partial P4_{16} language support, e.g., no return statement

Figure 2: Top down image of SUME
P4 NetFPGA Benchmark
SimpleSumeSwitch Model

- internal switch architecture of P4 NetFPGA
- very basic model, e.g., no traffic manager or other complex components

Figure 3: Information exchanged in SimpleSumeSwitch
P4 NetFPGA Benchmark

Setup

- benchmarks conducted in Baltikum testbed [3]
- adapted previous work: PlutosKerberos [4]
  - framework: generate and benchmark P4\textsubscript{16}
  - uses MoonGen, pos
  - added support for SUME
- measurements run on SUME host
  - simulate program & synthesize bitstream
  - program & benchmark SUME

**Figure 4:** Connections between measurement host and SUME
P4 NetFPGA Benchmark

header eth_t { bit <48> d; bit <48> s; bit <16> type; }

struct headers { eth_t eth; }
struct meta { bit <8> unused; }
struct digest { bit <80> unused; }

parser parse(packet_in p, out headers h, out meta m, out digest d, inout sume_metadata_t s) {
    state start {
        p.extract(h.eth);
        transition select(h.eth.type) {
            default: accept;
        }
    }
}

control pipeline(...) {
    ...
}
control deparse(...) {
    ...
}

SimpleSumeSwitch(parse(), pipeline(), deparse()) main;

Sample program parsing Ethernet and forwarding packets (reference)
Sample program with Ethernet and h000_t header (parser-hsh1)

h000_t contains only a single 16 bit field

```
P4 NetFPGA Benchmark

header  eth_t  {  bit <48> d;  bit <48> s;  bit <16> type;  }
header  h000_t  {  bit <16> f000;  }

struct  headers  {  eth_t  eth;  h000_t  h000;  }
struct  meta  {  bit <8>  unused;  }
struct  digest  {  bit <80>  unused;  }

parser  parse(packet_in  p,  out  headers  h,  out  meta  m,
              out  digest  d,  inout  sume_metadata_t  s)  {

    state  start  {
        p.extract(h.eth);
        transition  select(h.eth.type)  {
            16w0x0800:  parse_h000;
            default:  accept;
        }
    }
    state  parse_h000  {
        p.extract(h.h000);
        transition  accept;
    }
}

control  pipeline(...){ ... } 
control  deparse(...){ ... } 

SimpleSumeSwitch(parse(),  pipeline(),  deparse())  main;
```
Figure 5: Header Stack Height: throughput maxima approx. 9999.8 Mbit/s
Figure 6: Header Stack Height: Latency histogram for different runs
Figure 6: Header Stack Height: Latency histogram for different runs
Figure 6: Header Stack Height: Latency histogram for different runs
P4 NetFPGA Benchmark

Header Stack Height: FPGA Usage

Reported FPGA element usage

<table>
<thead>
<tr>
<th>P4 program</th>
<th>LUT</th>
<th>BRAM</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>18 kbit</td>
<td>36 kbit</td>
</tr>
<tr>
<td>reference</td>
<td>17.1 % (74 179)</td>
<td>1.5 % (43)</td>
<td>26.5 % (390)</td>
</tr>
<tr>
<td>parser-hsh1</td>
<td>18.7 % (81 073)</td>
<td>1.7 % (51)</td>
<td>27.2 % (400)</td>
</tr>
<tr>
<td>parser-hsh2</td>
<td>20.1 % (86 935)</td>
<td>1.8 % (53)</td>
<td>28.0 % (411)</td>
</tr>
<tr>
<td>parser-hsh3</td>
<td>21.4 % (92 804)</td>
<td>1.7 % (51)</td>
<td>29.0 % (426)</td>
</tr>
</tbody>
</table>

Figure 7: Header Stack Height: Usage of different FPGA element types

FPGA element usage increase

- LUT: $\approx 1.4 \%$
- BRAM: $\leq 0.1 \%$ and $\approx 0.8 \%$
- Register: $\approx 0.8 \%$
<table>
<thead>
<tr>
<th>P4 feature</th>
<th>Throughput Decrease</th>
<th>Latency</th>
<th>LUT</th>
<th>BRAM 18 kbit</th>
<th>BRAM 36 kbit</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>parser-hsh</td>
<td>≤ 0.1 Mbit/s</td>
<td>150 ns</td>
<td>1.4 %</td>
<td>≤ 0.1 %</td>
<td>0.8 %</td>
<td>0.8 %</td>
</tr>
<tr>
<td>parser-hf</td>
<td>≤ 0.1 Mbit/s</td>
<td>≤ 10 ns</td>
<td>≤ 0.1 %</td>
<td>≤ 0.1 %</td>
<td>≤ 0.1 %</td>
<td>≤ 0.1 %</td>
</tr>
<tr>
<td>parser-hfs</td>
<td>≤ 0.1 Mbit/s</td>
<td>≤ 10 ns</td>
<td>≤ 0.1 %</td>
<td>≤ 0.1 %</td>
<td>≤ 0.1 %</td>
<td>≤ 0.1 %</td>
</tr>
<tr>
<td>parser-bf</td>
<td>≤ 0.1 Mbit/s</td>
<td>150 ns</td>
<td>1.4 %</td>
<td>≤ 0.1 %</td>
<td>0.9 %</td>
<td>0.9 %</td>
</tr>
<tr>
<td>add-header</td>
<td>20 Mbit/s</td>
<td>150 ns</td>
<td>1.3 %</td>
<td>0.2 %</td>
<td>0.8 %</td>
<td>0.8 %</td>
</tr>
<tr>
<td>add-header-f</td>
<td>≤ 0.1 Mbit/s</td>
<td>≤ 10 ns</td>
<td>≤ 0.1 %</td>
<td>≤ 0.1 %</td>
<td>≤ 0.1 %</td>
<td>0.2 %</td>
</tr>
<tr>
<td>add-header-fs</td>
<td>≤ 0.1 Mbit/s</td>
<td>≤ 10 ns</td>
<td>≤ 0.1 %</td>
<td>≤ 0.1 %</td>
<td>≤ 0.1 %</td>
<td>≤ 0.1 %</td>
</tr>
<tr>
<td>rm-header</td>
<td>≤ 0.1 Mbit/s</td>
<td>≤ 10 ns</td>
<td>≤ 0.1 %</td>
<td>≤ 0.1 %</td>
<td>≤ 0.1 %</td>
<td>≤ 0.1 %</td>
</tr>
<tr>
<td>set-field</td>
<td>≤ 0.1 Mbit/s</td>
<td>≤ 10 ns</td>
<td>≤ 0.1 %</td>
<td>≤ 0.1 %</td>
<td>≤ 0.1 %</td>
<td>≤ 0.1 %</td>
</tr>
<tr>
<td>tables</td>
<td>≤ 0.1 Mbit/s</td>
<td>30 ns</td>
<td>0.4 %</td>
<td>≤ 0.1 %</td>
<td>0.3 %</td>
<td>0.5 %</td>
</tr>
<tr>
<td>read-state</td>
<td>≤ 0.1 Mbit/s</td>
<td>125 ns</td>
<td>0.7 %</td>
<td>≤ 0.1 %</td>
<td>1.1 %</td>
<td>1.1 %</td>
</tr>
<tr>
<td>write-state</td>
<td>≤ 0.1 Mbit/s</td>
<td>125 ns</td>
<td>0.7 %</td>
<td>≤ 0.1 %</td>
<td>1.1 %</td>
<td>1.1 %</td>
</tr>
</tbody>
</table>

Figure 8: Observed result changes by P4 feature tested; unless otherwise stated, increase per feature occurrence.
## Result Summary

<table>
<thead>
<tr>
<th>P4 feature</th>
<th>Throughput Decrease</th>
<th>Latency</th>
<th>LUT (18\text{kbit})</th>
<th>BRAM (36\text{kbit})</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>parser-hsh</td>
<td>(\leq 0.1\text{Mbit/s})</td>
<td>150 ns</td>
<td>1.4 %</td>
<td>(\leq 0.1%)</td>
<td>0.8 %</td>
</tr>
</tbody>
</table>

*Figure 8: Observed result changes by P4 feature tested; unless otherwise stated, increase per feature occurrence.*
### Result Summary

<table>
<thead>
<tr>
<th>P4 feature</th>
<th>Throughput Decrease</th>
<th>Latency</th>
<th>LUT 18 kbit</th>
<th>BRAM 36 kbit</th>
<th>Registers 0.8 %</th>
</tr>
</thead>
<tbody>
<tr>
<td>add-header</td>
<td>20 Mbit/s</td>
<td>150 ns</td>
<td>1.3 %</td>
<td>0.2 %</td>
<td>0.8 %</td>
</tr>
</tbody>
</table>

**Figure 8:** Observed result changes by P4 feature tested; unless otherwise stated, increase per feature occurrence.
## Result Summary

<table>
<thead>
<tr>
<th>P4 feature</th>
<th>Throughput Decrease</th>
<th>Latency</th>
<th>LUT</th>
<th>BRAM</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>parser-hsh</td>
<td>≤ 0.1 Mbit/s</td>
<td>150 ns</td>
<td>0.7%</td>
<td>≤ 0.1%</td>
<td>1.1 %</td>
</tr>
<tr>
<td>parser-hf</td>
<td>≤ 0.1 Mbit/s</td>
<td>≤ 10 ns</td>
<td>≤ 0.1%</td>
<td>≤ 0.1%</td>
<td>≤ 0.1%</td>
</tr>
<tr>
<td>parser-hfs</td>
<td>≤ 0.1 Mbit/s</td>
<td>≤ 10 ns</td>
<td>≤ 0.1%</td>
<td>≤ 0.1%</td>
<td>≤ 0.1%</td>
</tr>
<tr>
<td>parser-bf</td>
<td>≤ 0.1 Mbit/s</td>
<td>150 ns</td>
<td>0.7%</td>
<td>≤ 0.1%</td>
<td>0.9 %</td>
</tr>
<tr>
<td>add-header</td>
<td>≤ 20 Mbit/s</td>
<td>150 ns</td>
<td>1.3%</td>
<td>≤ 0.2%</td>
<td>0.3%</td>
</tr>
<tr>
<td>add-header-f</td>
<td>≤ 0.1 Mbit/s</td>
<td>≤ 10 ns</td>
<td>≤ 0.1%</td>
<td>≤ 0.1%</td>
<td>≤ 0.1%</td>
</tr>
<tr>
<td>add-header-fs</td>
<td>≤ 0.1 Mbit/s</td>
<td>≤ 10 ns</td>
<td>≤ 0.1%</td>
<td>≤ 0.1%</td>
<td>≤ 0.1%</td>
</tr>
<tr>
<td>rm-header</td>
<td>≤ 0.1 Mbit/s</td>
<td>≤ 10 ns</td>
<td>≤ 0.1%</td>
<td>≤ 0.1%</td>
<td>≤ 0.1%</td>
</tr>
<tr>
<td>set-field</td>
<td>≤ 0.1 Mbit/s</td>
<td>150 ns</td>
<td>0.7%</td>
<td>≤ 0.1%</td>
<td>1.1 %</td>
</tr>
<tr>
<td>tables</td>
<td>≤ 0.1 Mbit/s</td>
<td>30 ns</td>
<td>0.4%</td>
<td>≤ 0.3%</td>
<td>0.5 %</td>
</tr>
<tr>
<td>read-state</td>
<td>≤ 0.1 Mbit/s</td>
<td>125 ns</td>
<td>0.7%</td>
<td>≤ 0.1%</td>
<td>1.1 %</td>
</tr>
<tr>
<td>write-state</td>
<td>≤ 0.1 Mbit/s</td>
<td>125 ns</td>
<td>0.7%</td>
<td>≤ 0.1%</td>
<td>1.1 %</td>
</tr>
</tbody>
</table>

*Figure 8: Observed result changes by P4 feature tested; unless otherwise stated, increase per feature occurrence*
P4-NetFPGA

- P4 on FPGA — ambitious project
- still in development
  - incomplete language support / imposes further restrictions
  - build process not deterministic
  - ...

NetFPGA

- “works as advertised”: 10 Gbit/s, predictable latency
  - manually optimized P4 [5]: throughput: 5 Gbit/s to 30 Gbit/s; latency: approx. 1 µs to 2 µs
- results suggest: larger throughput feasible for FPGA-based network hardware
Challenges for this master’s thesis

- our chair provided the hardware (server, NetFPGA)
- 1st challenge:
  - getting the P4 NetFPGA toolchain running
  - getting the correct licenses from Xilinx
  - small, very new project → little documentation available
  - rebooting after flashing new firmware without cutting the power
- 2nd challenge:
  - successfully compiling P4 programs
  - synthesizing programs can take up to several hours
  - we solved this by using several servers in parallel (Baltikum testbed)
  - compiled program does not always work
- 3rd challenge:
  - distilling knowledge from observations
  - trends are sometimes more important than hard numbers
  - trying to find out the generalized behavior for P4 systems
P4 Benchmarking & Wireless Measurements

P4 Benchmark

Reproducible Wireless Measurements

Motivation
Repeatability, Replicability, and Reproducibility
Investigation of Wireless Channel Behavior

Bibliography
Motivation

Networked Control System

Figure 9: Networked control system

- **control systems** manage or regulate systems via control loops (e.g., balancing robots)
- **control loops** constantly measure a plant’s state and calculate control signals to be executed by the plant
- **networked control systems** involve networked communication in between plant and controller
  - wired networks: predictable performance and latency, specialized network hardware or protocols (e.g. Industrial Ethernet)
  - wireless networks: unpredictable due to interference or reflections
Why should we use wireless networks for control processes?

- Wired networks cannot be used in all cases
  - high costs for wiring
  - too widespread networks for wired transmission
  - mobile plants (robots, flying drones)
- Wireless systems are cost-efficient and support large and mobile networks

But can we use them for control processes?

- Active area of research
- We want to provide measurements and models for wired channel behavior
- Therefore, we need dependable, trustworthy measurements for wireless channels
- However, our measurement results have high variance because of interference or reflections
Motivation

Short Recap: Reproducibility

Classification according to ACM

1. **Repeatability**: Same people use same setup to repeat results.
2. **Replicability**: Different people use same setup to replicate results. We provide “artifacts” (scripts, data, documentation) so any other teams can easily replicate our work.
3. **Reproducibility**: Different people use different setup to reproduce results. We provide a detailed documentation of our approach so other teams can reproduce our work without using our artifacts.

Due to the high sensitivity towards interference wireless measurements have not even **repeatable** results
Repeatability, Replicability, and Reproducibility

Setup

Repeatable wireless measurements

- fully controllable wireless environment through shielded boxes
- wireless signal is transmitted via shielded coax cables between both boxes
- our shielding works for 2.4 GHz, 5.0 GHz for testing different IEEE 802.11 standards
- environment can be influenced artificially, e.g., attenuators between the boxes
- making the wireless environment repeatable

Figure 10: Setup for reproducible wireless measurements
Repeatability, Replicability, and Reproducibility
IEEE 802.11 in two Boxes

Figure 11: Testbed for reproducible wireless measurements
Making entire experiments **repeatable**

- making the software repeatable
- create repeatable configuration
- use live images and automate entire experiment workflow
  - system developed at our chair: plain orchestrating service (pos)
Repeatability, Replicability, and Reproducibility

Experiment process in pos

1. deployment of Linux live images and experiment scripts
2. configuration of DuT & LoadGen
3. execution of experiment
4. collection of experiment results
5. evaluation of experiment results
Repeatability, Replicability, and Reproducibility

Benefits of pos

• reuse of testbed infrastructure for different experiments
  • works also for repeatable wired experiments
  • used for different setups (VM setups, NetFPGA measurements, etc.)

• configuration scripts enforce consistent configuration

• repeatable experiments through automation

• replicable experiments by granting access to our testbed
Investigation of Wireless Channel Behavior
Bachelor’s Thesis

Topic

- characterize wireless channel through a series of measurements
- focus on latency and reliability
- development of a measurement suite to characterize the channel on different layers of the network stack

The following slides are based on a Bachelor’s Thesis by Eric Hauser.
Investigation of Wireless Channel Behavior

- measure connection on different layers
- end-to-end on layer 7 to determine overall quality of the control process
- lower layer measurements to determine potential sources of observed behavior
Investigation of Wireless Channel Behavior
Stack measurements

Figure 14: Delays between all stages of the transmission
- Partial Reliable Real-Time Protocol (PRRT)
- developed by University of Saarland and Friedrich-Alexander-Universität Erlangen-Nürnberg (see nt.uni-saarland.de)
- control systems require regular updates preferably with constant delivery time:
  - PRRT guarantees that packets arrive only at controller only after a specified target delay
  - grace period before transmit delay is also configurable
  - drop packets if guarantees cannot be met
### Investigation of Wireless Channel Behavior

#### PRRT Measurements

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet-to-packet time ($l$)</td>
<td>1 ms</td>
<td>10 ms</td>
<td>5</td>
</tr>
<tr>
<td>Payload size ($\ell$)</td>
<td>20 B</td>
<td>1400 B</td>
<td>5</td>
</tr>
<tr>
<td>PRRT target delay</td>
<td>1 ms</td>
<td>10 ms</td>
<td>5</td>
</tr>
<tr>
<td>PRRT receive window</td>
<td>0.1 ms</td>
<td>2 ms</td>
<td>6</td>
</tr>
</tbody>
</table>

*Figure 15: Parameters of the delay measurement set*

- $5 \times 5 \times 5 \times 6 = 750$ measurements
- Goal: deduct answers from this pool of measurements
Investigation of Wireless Channel Behavior
PRRT - Measurement Results

- sampling time of 1 ms, target delay 10 ms, receive window 0.48 ms, payload size 20 B
- log scale (!) on y-axis
- target delay (upper red line) and receive window (distance between both red lines)
- packet loss on application layer \( (m_6 - m_1) \) caused by PRRT due high delay on lower layers
Investigation of Wireless Channel Behavior
Influence of Target Delay & Receive Window

- sampling time higher than 1 ms, all packet sizes
- variable target delay and receive windows
- target delay should be over 1 ms and receive window below 0.1 ms to achieve reasonable packet loss rates
Investigation of Wireless Channel Behavior
Influence of Packet Size

- sampling time higher than 1 ms
- black boxplots: 0.48 ms receive window and 1.0 ms target delay
- red boxplots: 0.48 ms receive window and 3.25 ms target delay
- packet size has only minor impact (relevant only for situations where loss is already higher)
Challenges for this bachelor’s thesis

- our chair provided the hardware (servers, WiFi boxes) and measurement software
- 1st challenge:
  - getting the testbed running
  - setup was created and built together with other students and advisors
  - setup tool chain to create plots in an automated manner
- 2nd challenge:
  - getting PRRT running
  - close collaboration with protocol developers who updated and fixed their protocol
- 3rd challenge:
  - distilling knowledge from observations over 750 single measurements, each containing several plots
  - trends are sometimes more important than hard numbers
  - trying to find out the generalized behavior for wireless systems
P4 Benchmarking & Wireless Measurements

P4 Benchmark

Reproducible Wireless Measurements

Bibliography


