EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

# Automated Test Bench for High-Performance Network Equipment

**Benjamin Steinert**[1,2], Gabriel Paradzik[1,2], Michael Menth[1]
[1]Chair of Communication Networks, [2]Zentrum für Datenverarbeitung
University of Tübingen

bwNET

04.04.2025 - 4. GI/ITG KuVS Fachgespräch "Network Softwarization" (online)

# Motivation

- ► New devices require thorough testing and evaluation before deployment
  - ▪ Do data sheet specs hold?
  - ▪ Not all important information may be stated on data sheets (e.g., control plane performance)
  - ▪ Need for traffic load at high rates (≥100 Gbit/s)

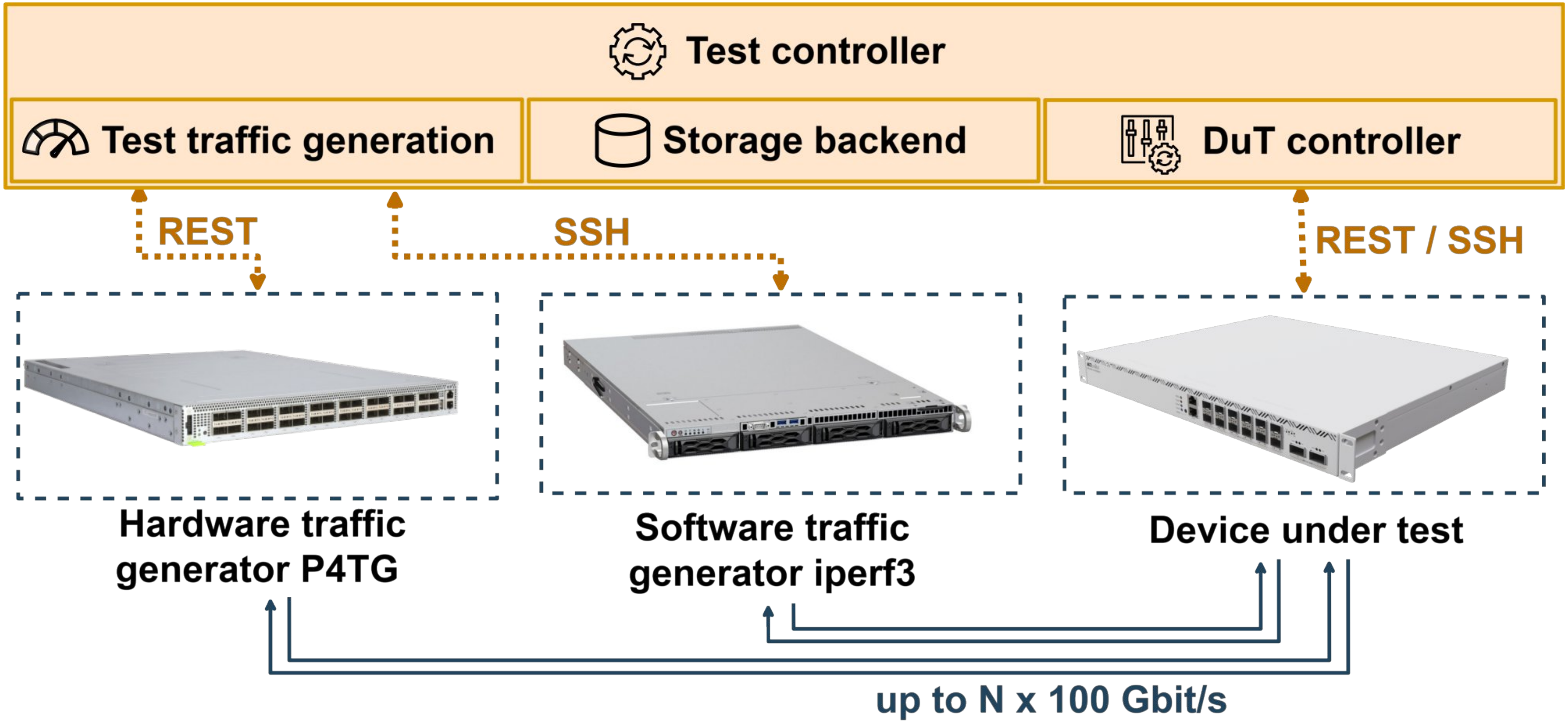
- ► Commercial Ethernet test systems are available, but limited in flexibility, and may be very expensive

- ► Manual testing is error-prone and time-consuming

➡ Leverage automation interfaces and central orchestration for automated benchmarking

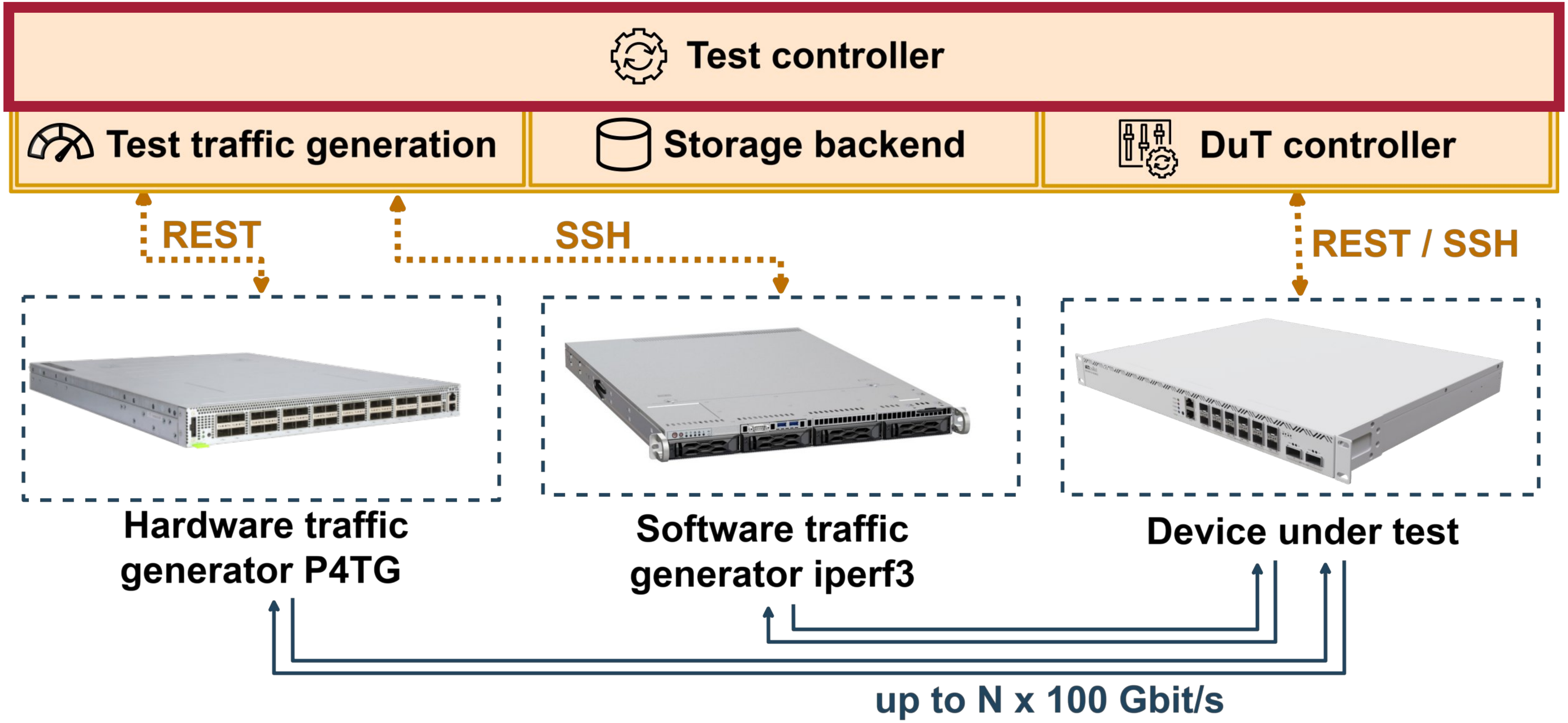
# TEST BENCH ARCHITECTURE AND COMPONENTS

# Test Bench Architecture and Components



EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

# Test Bench Architecture and Components



Test controller

Test traffic generation

Storage backend

DuT controller

REST

SSH

REST / SSH

Hardware traffic generator P4TG

Software traffic generator iperf3

Device under test

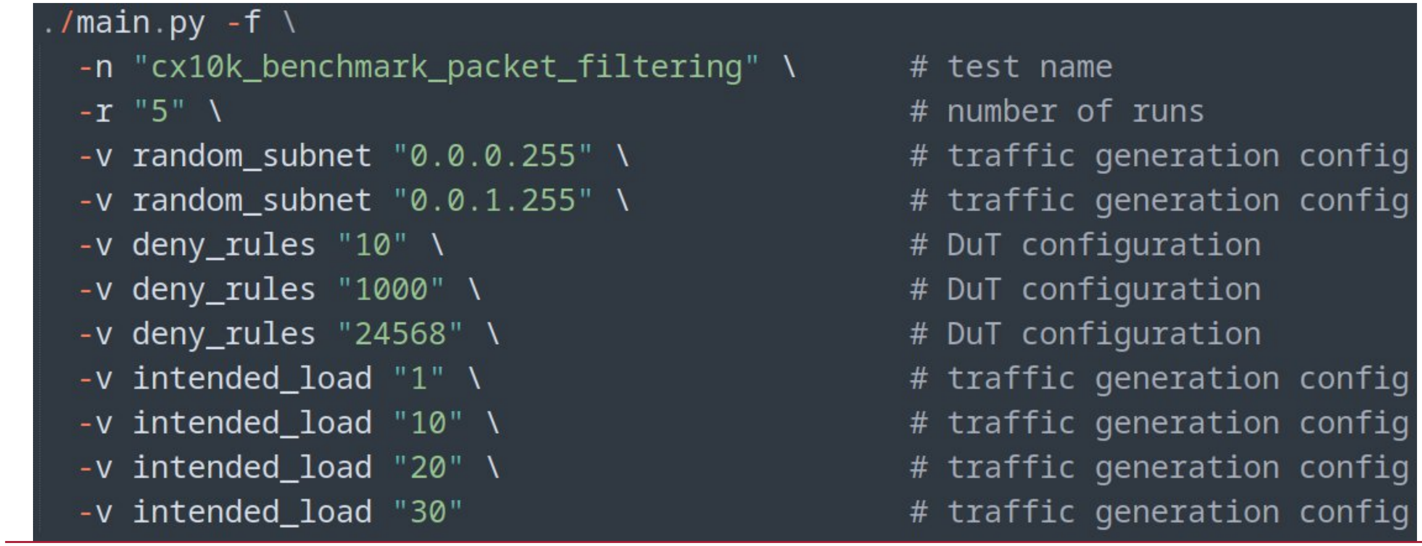
up to N x 100 Gbit/s

# Test Controller

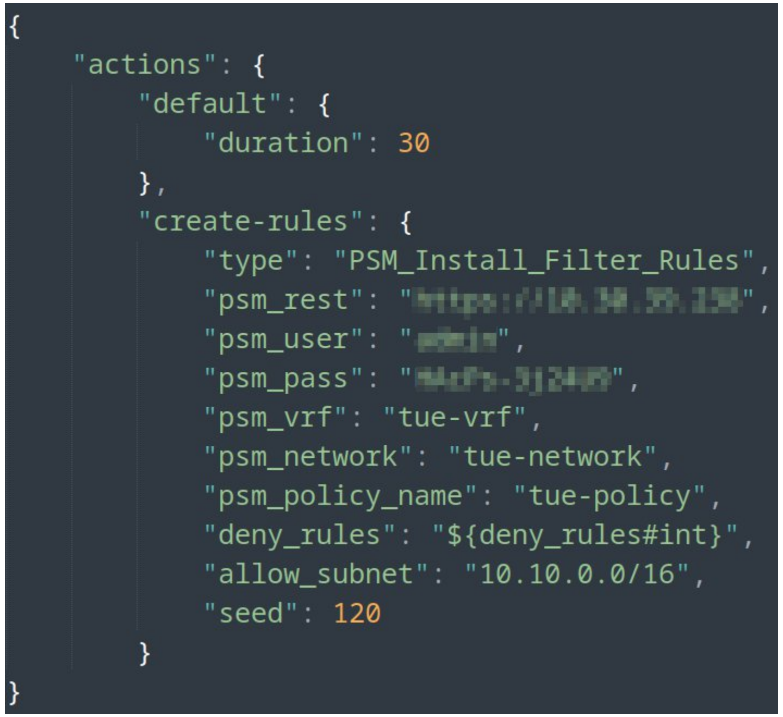
- ▶ Written in Python

- ▶ Orchestrates the different modules for automated experiment runs
  - Automated test traffic generation with configurable traffic generators (e.g., P4TG, iperf3)
  - Automated storage of results with configurable storage backend (e.g., MongoDB)
  - Automated device configuration / metric monitoring with DuT Controller (vendor-specific)

- ▶ Declarative experiment configuration



```
./main.py -f \
  -n "cx10k_benchmark_packet_filtering" \      # test name
  -r "5" \                                     # number of runs
  -v random_subnet "0.0.0.255" \               # traffic generation config
  -v random_subnet "0.0.1.255" \               # traffic generation config
  -v deny_rules "10" \                         # DuT configuration
  -v deny_rules "1000" \                       # DuT configuration
  -v deny_rules "24568" \                      # DuT configuration
  -v intended_load "1" \                       # traffic generation config
  -v intended_load "10" \                      # traffic generation config
  -v intended_load "20" \                      # traffic generation config
  -v intended_load "30"                        # traffic generation config
```



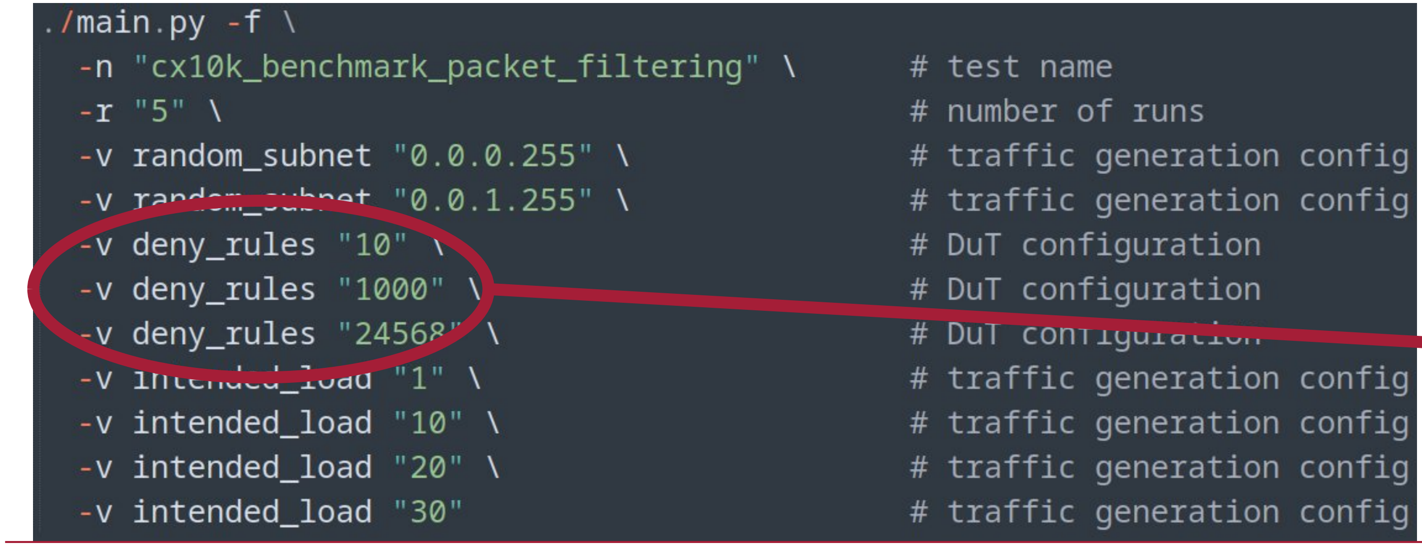
```
{
    "actions": {
        "default": {
            "duration": 30
        },
        "create-rules": {
            "type": "PSM_Install_Filter_Rules",
            "psm_rest": "[redacted]",
            "psm_user": "[redacted]",
            "psm_pass": "[redacted]",
            "psm_vrf": "tue-vrf",
            "psm_network": "tue-network",
            "psm_policy_name": "tue-policy",
            "deny_rules": "${deny_rules#int}",
            "allow_subnet": "10.10.0.0/16",
            "seed": 120
        }
}
```

# Test Controller

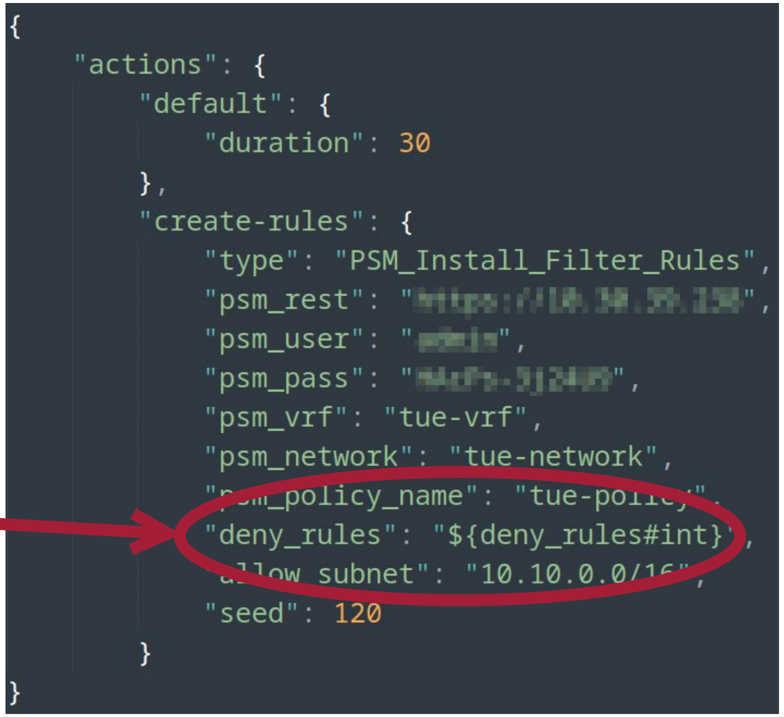
- Written in Python

- Orchestrates the different modules for automated experiment runs
  - Automated test traffic generation with configurable traffic generators (e.g., P4TG, iperf3)
  - Automated storage of results with configurable storage backend (e.g., MongoDB)
  - Automated device configuration / metric monitoring with DuT Controller (vendor-specific)

- Declarative experiment configuration

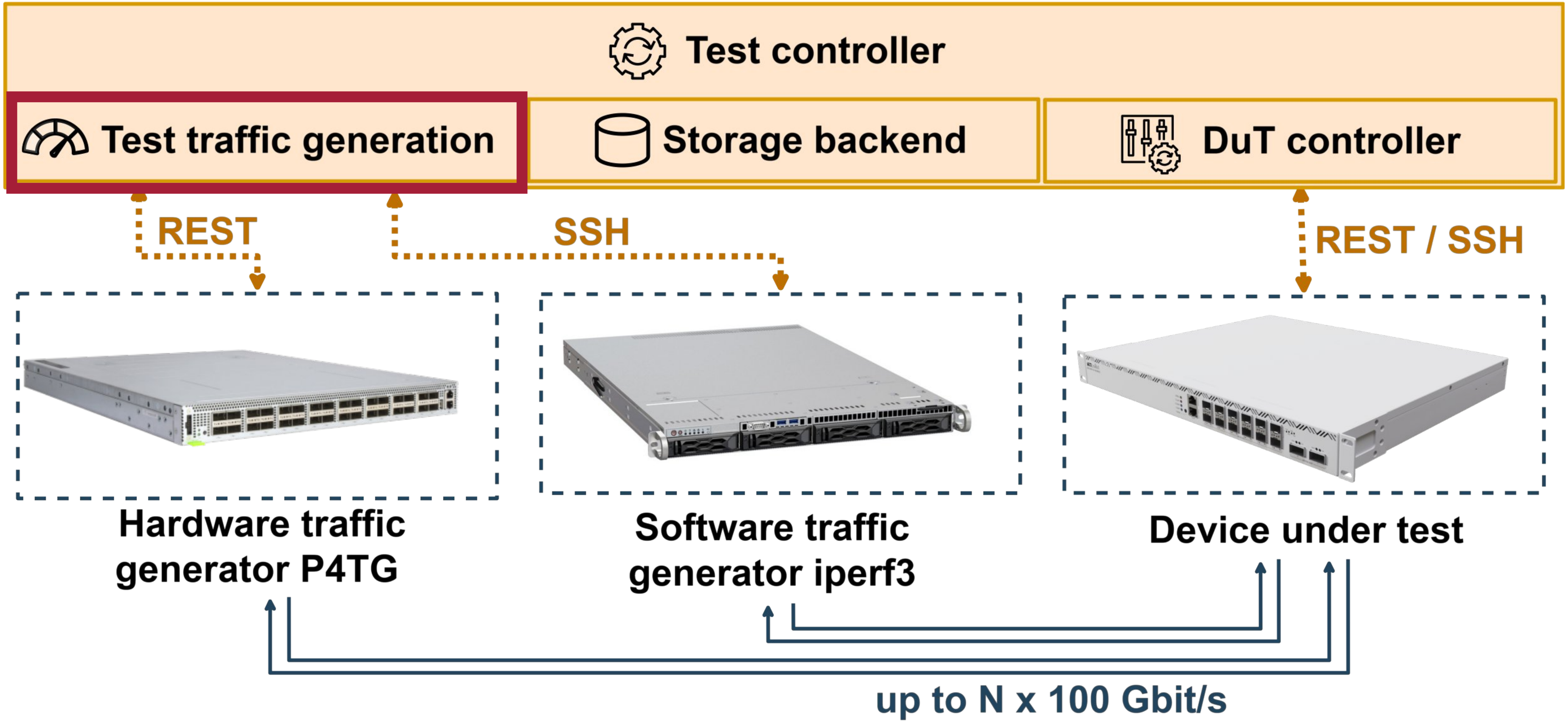


```
./main.py -f \
  -n "cx10k_benchmark_packet_filtering" \      # test name
  -r "5" \                                     # number of runs
  -v random_subnet "0.0.0.255" \               # traffic generation config
  -v random_subnet "0.0.1.255" \               # traffic generation config
  -v deny_rules "10" \                         # DuT configuration
  -v deny_rules "1000" \                       # DuT configuration
  -v deny_rules "24568" \                      # DuT configuration
  -v intended_load "1" \                       # traffic generation config
  -v intended_load "10" \                      # traffic generation config
  -v intended_load "20" \                      # traffic generation config
  -v intended_load "30"                        # traffic generation config
```



```
{
    "actions": {
        "default": {
            "duration": 30
        },
        "create-rules": {
            "type": "PSM_Install_Filter_Rules",
            "psm_rest": "[redacted]",
            "psm_user": "[redacted]",
            "psm_pass": "[redacted]",
            "psm_vrf": "tue-vrf",
            "psm_network": "tue-network",
            "psm_policy_name": "tue-policy",
            "deny_rules": "${deny_rules#int}",
            "allow_subnet": "10.10.0.0/16",
            "seed": 120
        }
}
```

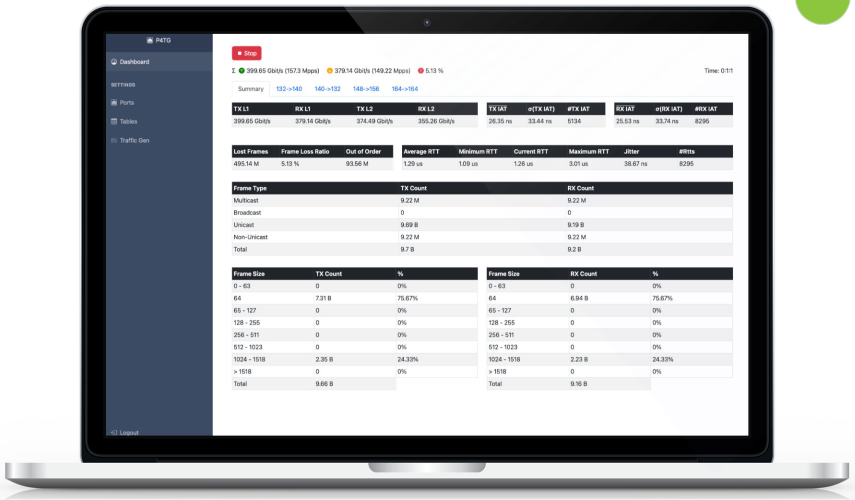
# Test Bench Architecture and Components



# Test Traffic Generation

- P4TG
  - Runs on Edgecore Wedge 100-32X
  - Controlled via REST API of P4TG Controller
  - Offers traffic generation at high data rates
    (up to 100 Gbit/s per port; up to 400 Gbit/s per port for Tofino 2)
  - Low-cost hardware TG
  - Customizable for individual needs
  - Open source: https://github.com/uni-tue-kn/P4TG

- Iperf3
  - Runs on separate server / VM
  - Controlled via SSH



Σ 399.65 Gbit/s (157.3 Mpps) 379.14 Gbit/s (149.22 Mpps) 5.13 %

Time: 0:1:1

| TX L1 | RX L1 | TX L2 | RX L2 |
|---|---|---|---|
| 399.65 Gbit/s | 379.14 Gbit/s | 374.49 Gbit/s | 355.26 Gbit/s |

| $\overline{\text{TX IAT}}$ | σ(TX IAT) | #TX IAT |
|---|---|---|
| 26.35 ns | 33.44 ns | 5134 |

| $\overline{\text{RX IAT}}$ | σ(RX IAT) | #RX IAT |
|---|---|---|
| 25.53 ns | 33.74 ns | 8295 |

| Lost Frames | Frame Loss Ratio | Out of Order |
|---|---|---|
| 495.14 M | 5.13 % | 93.56 M |

| Average RTT | Minimum RTT | Current RTT | Maximum RTT | Jitter | #Rtts |
|---|---|---|---|---|---|
| 1.29 us | 1.09 us | 1.26 us | 3.01 us | 38.67 ns | 8295 |

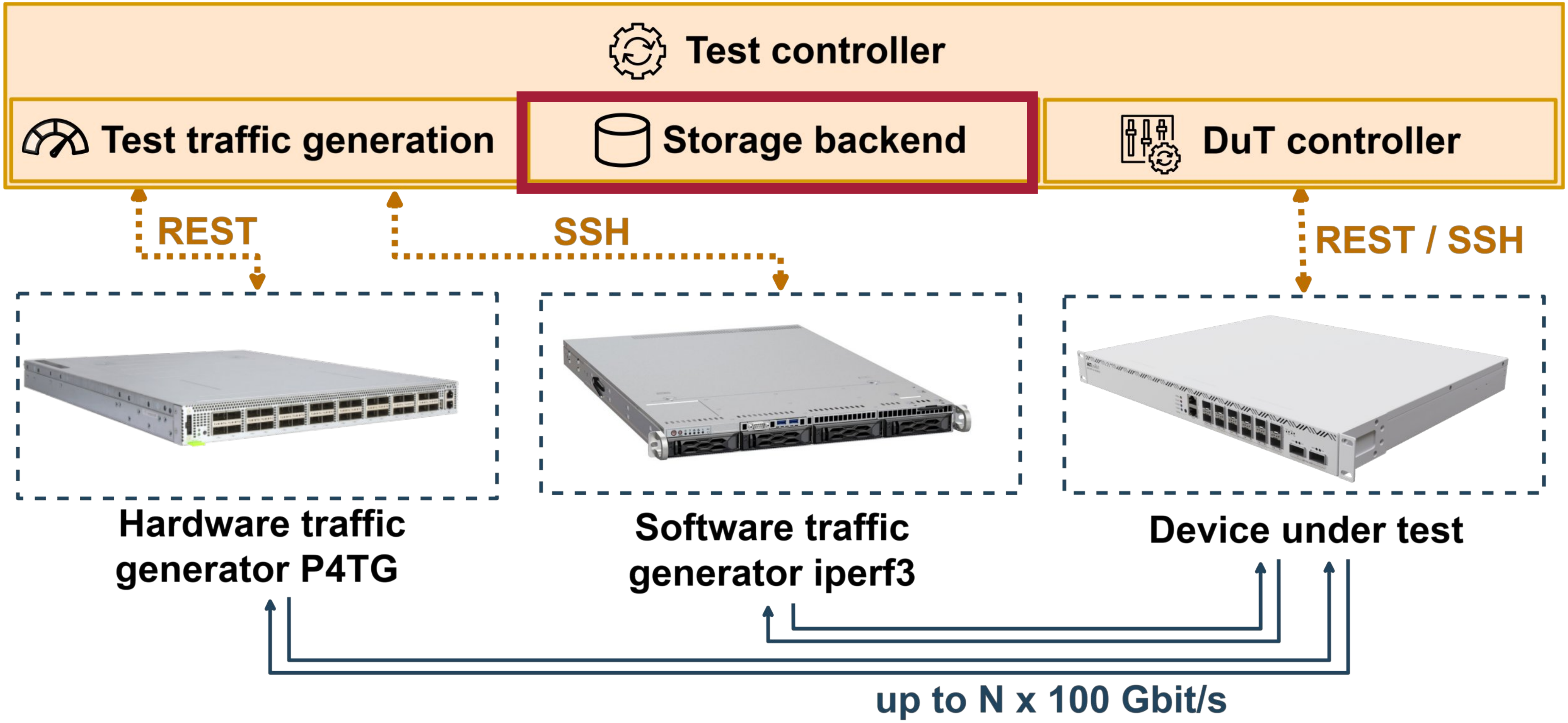
| Frame Type | TX Count | RX Count |
|---|---|---|
| Multicast | 9.22 M | 9.22 M |
| Broadcast | 0 | 0 |
| Unicast | 9.69 B | 9.19 B |
| Non-Unicast | 9.22 M | 9.22 M |
| Total | 9.7 B | 9.2 B |

| Frame Size | TX Count | % |
|---|---|---|
| 0 - 63 | 0 | 0% |
| 64 | 7.31 B | 75.67% |
| 65 - 127 | 0 | 0% |
| 128 - 255 | 0 | 0% |
| 256 - 511 | 0 | 0% |
| 512 - 1023 | 0 | 0% |
| 1024 - 1518 | 2.35 B | 24.33% |
| > 1518 | 0 | 0% |
| Total | 9.66 B | |

| Frame Size | RX Count | % |
|---|---|---|
| 0 - 63 | 0 | 0% |
| 64 | 6.94 B | 75.67% |
| 65 - 127 | 0 | 0% |
| 128 - 255 | 0 | 0% |
| 256 - 511 | 0 | 0% |
| 512 - 1023 | 0 | 0% |
| 1024 - 1518 | 2.23 B | 24.33% |
| > 1518 | 0 | 0% |
| Total | 9.16 B | |

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

# Test Bench Architecture and Components

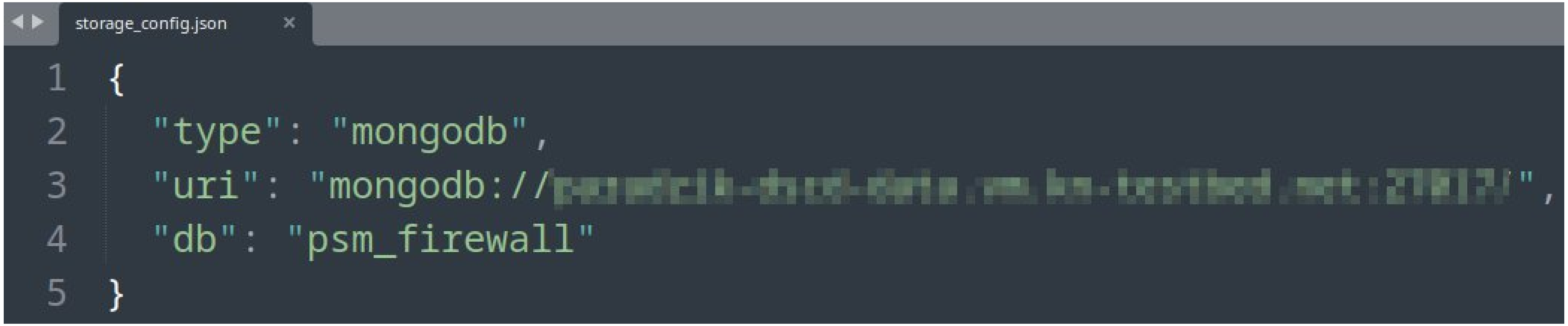


# Storage Backend

- Definition of abstract interface class “`StorageAdapter`“
  - Defines abstract methods like “`read`“, “`write`“, etc.

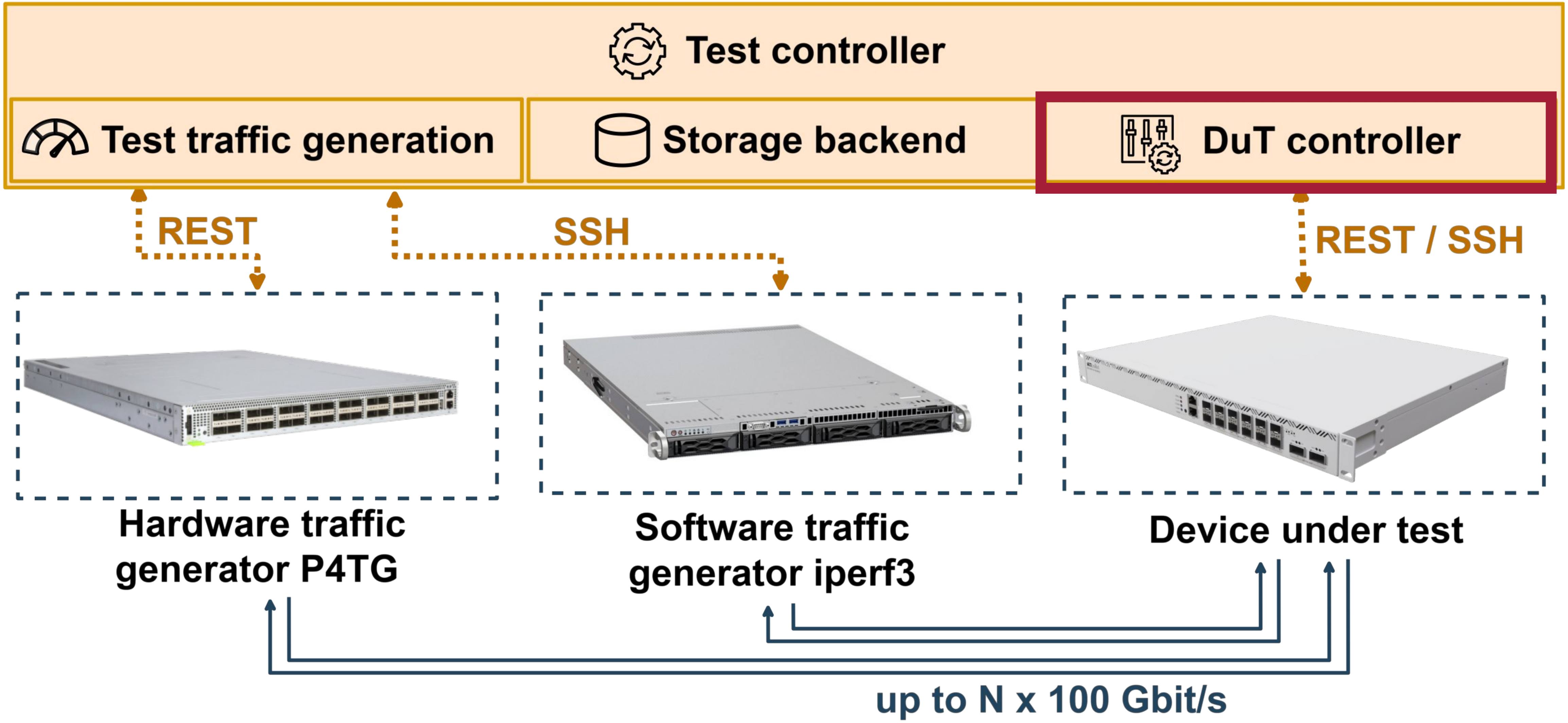
- Interface class is implemented for specific database backend, e.g. “`MongoDBStorageAdapter`“
  - Implements abstract methods

- Desired storage backend can then be configured in test controller



```
{
    "type": "mongodb",
    "uri": "mongodb://[redacted]",
    "db": "psm_firewall"
}
```

# Test Bench Architecture and Components



Test controller

Test traffic generation

Storage backend

DuT controller

REST

SSH

REST / SSH

Hardware traffic generator P4TG

Software traffic generator iperf3

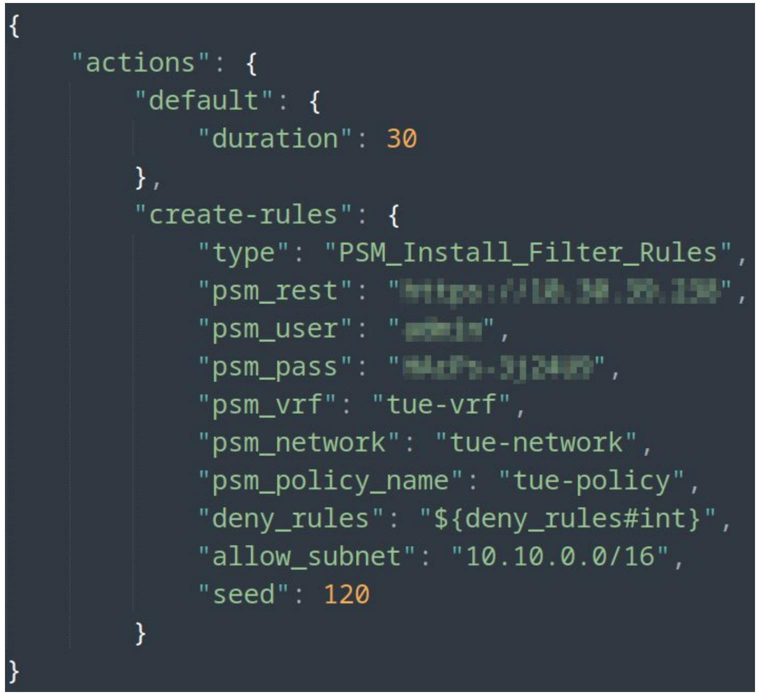
Device under test

up to N x 100 Gbit/s

# DuT Controller

- ► Implements vendor-specific automation interfaces of the DuT, e.g., REST API
  - ▪ Initiates connection with automation interface incl. authentication
  - ▪ Implements desired configuration tasks, e.g., “`PSM_Install_Filter_Rules`“

- ► Possible to add new vendor-specific modules
  - ▪ Implemented for MikroTik RouterOS, Aruba AOS-CX, AMD/Pensando PSM

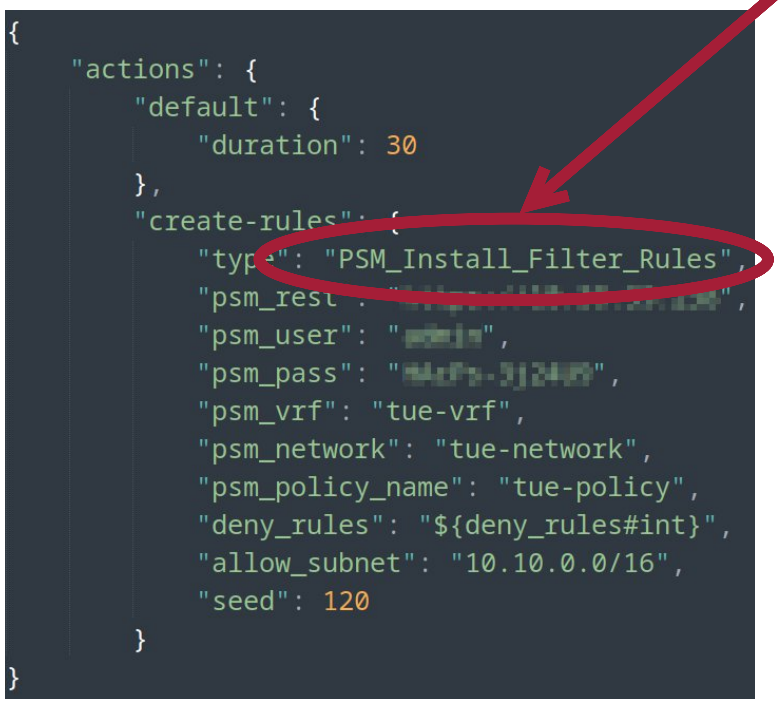


```
{
    "actions": {
        "default": {
            "duration": 30
        },
        "create-rules": {
            "type": "PSM_Install_Filter_Rules",
            "psm_rest": "[redacted]",
            "psm_user": "[redacted]",
            "psm_pass": "[redacted]",
            "psm_vrf": "tue-vrf",
            "psm_network": "tue-network",
            "psm_policy_name": "tue-policy",
            "deny_rules": "${deny_rules#int}",
            "allow_subnet": "10.10.0.0/16",
            "seed": 120
        }
}
```

# DuT Controller

- Implements vendor-specific automation interfaces of the DuT, e.g., REST API
  - Initiates connection with automation interface incl. authentication
  - Implements desired configuration tasks, e.g. “`PSM_Install_Filter_Rules`“

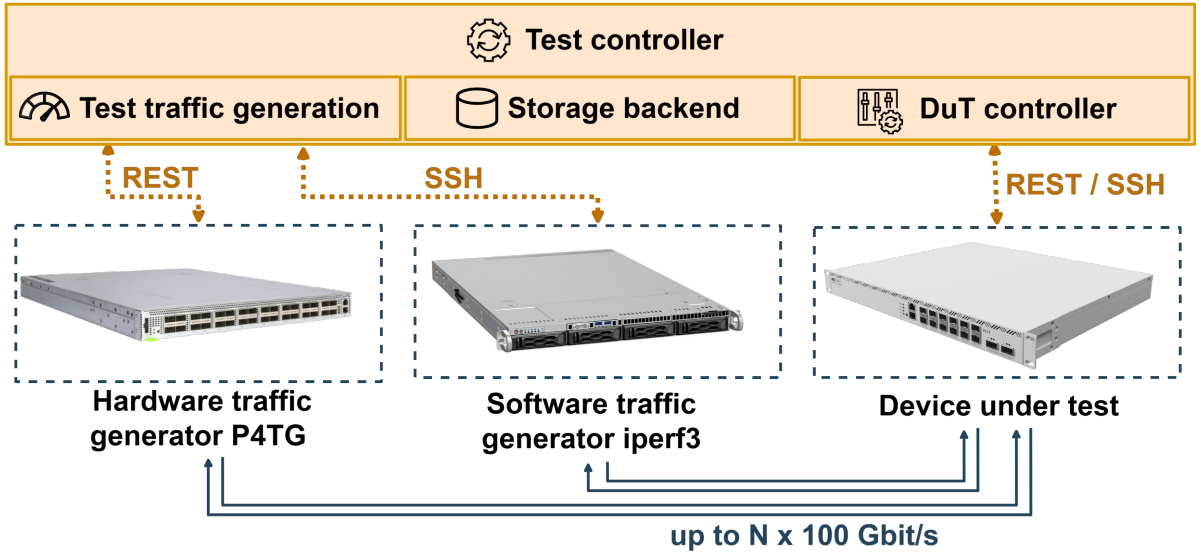
- Possible to add new vendor-specific modules
  - Implemented for MikroTik RouterOS, Aruba AOS-CX, AMD/Pensando PSM



```
{
    "actions": {
        "default": {
            "duration": 30
        },
        "create-rules": {
            "type": "PSM_Install_Filter_Rules",
            "psm_rest": "[redacted]",
            "psm_user": "[redacted]",
            "psm_pass": "[redacted]",
            "psm_vrf": "tue-vrf",
            "psm_network": "tue-network",
            "psm_policy_name": "tue-policy",
            "deny_rules": "${deny_rules#int}",
            "allow_subnet": "10.10.0.0/16",
            "seed": 120
        }
}
```

# Workflow

1. Manually set test parameters in test controller & start test
2. Automated configuration of the DuT
3. Automated configuration of the traffic generator(s)
4. Automated experiment runs, gather data of interest
5. Store gathered data in database



# CASE STUDY: HARDWARE EVALUATION -
# L4 PACKET FILTERING ON ARUBA CX 10000

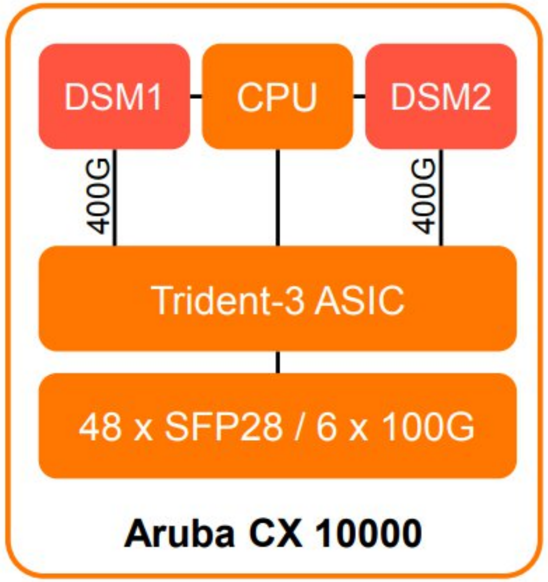
# Evaluated Device – Aruba CX 10000

- Port density: 48x SFP28 25 Gbit/s + 6x QSFP 100 Gbit/s

- Two different built-in ASICs
  - AMD/Pensando P4 DSM ASIC for 100G L4 packet filtering
  - Broadcom Trident-3 for switching & routing

- Two different software for configuration of ASICs
  - AOS-CX software for control of Broadcom chip
  - Pensando Stateful Manager (PSM) software for control of AMD/Pensando chip
  - Different automation interfaces are exposed

aruba + PENSANDO



DSM1

CPU

DSM2

400G

400G

Trident-3 ASIC

48 x SFP28 / 6 x 100G

Aruba CX 10000

# Data Plane Performance – IMIX

- Methodology
  - P4TG is used for test traffic generation of IMIX traffic
  - IMIX: 12% 64 B packets, 54% 512 B packets, 34% 1518 B packets
  - Install different numbers of filter rules (0, 10, 100, 1000, 10000, 24568)
- Result: 100 Gbit/s without drops possible, independent of number of installed filter rules

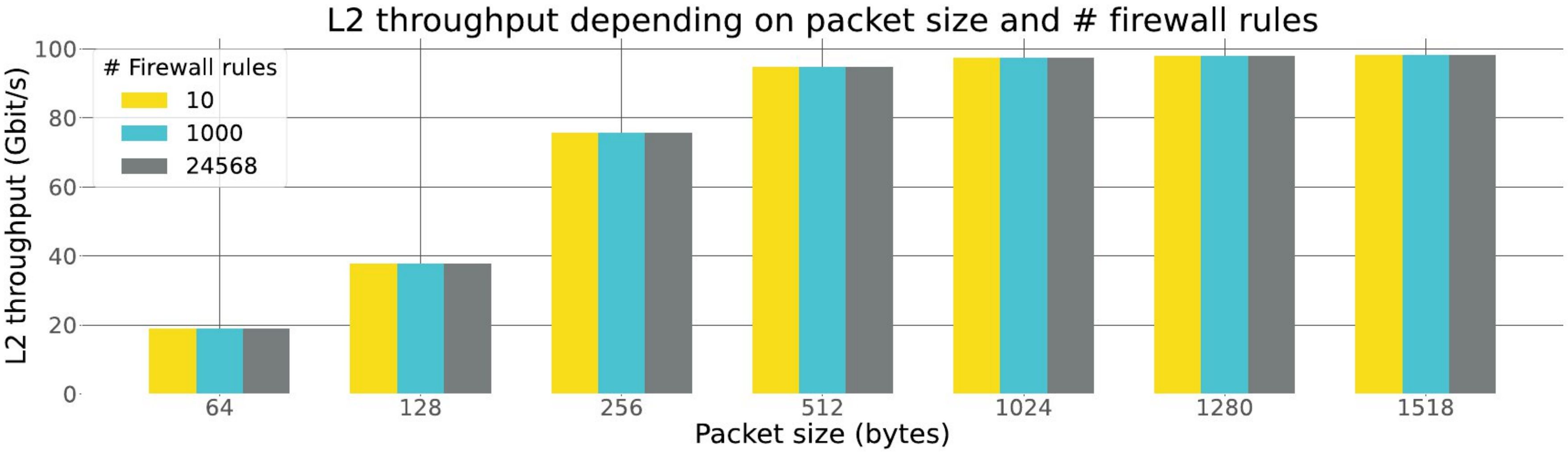
# Data Plane Performance – CBR

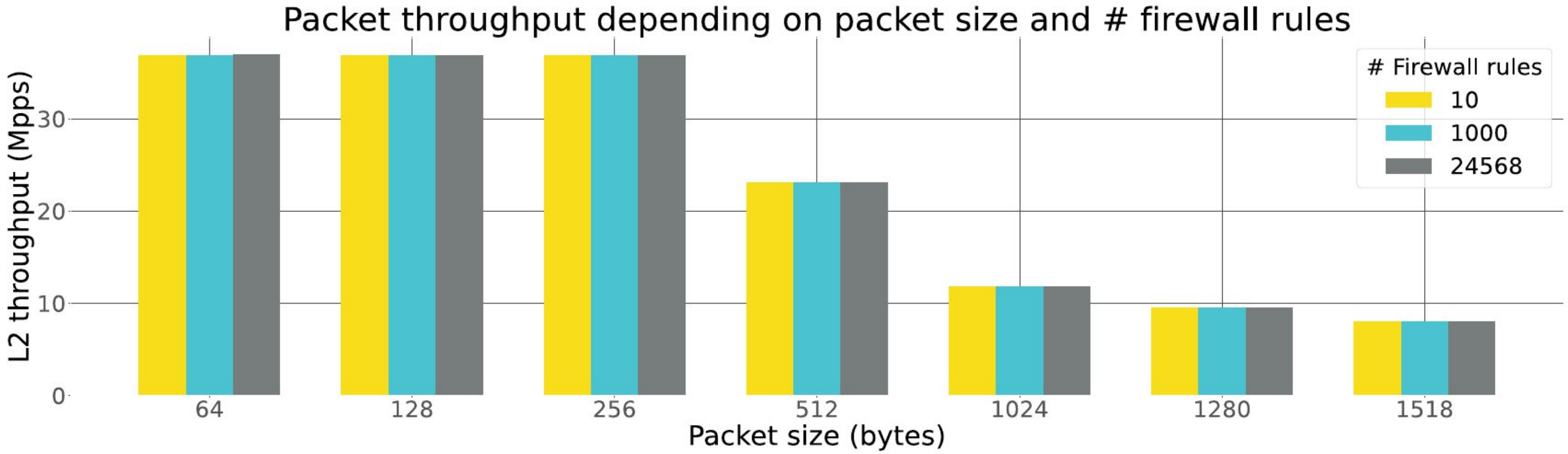
- Methodology
  - P4TG is used for test traffic generation of CBR traffic
  - CBR traffic of one packet size (64 B, 128 B, 256 B, 512 B, 1024 B, 1280 B, 1518 B)
  - Install different numbers of filter rules (0, 10, 100, 1000, 10000, 24568)

- Result
  - Up to ~40 Mpps for small packets <= 256 B
  - Up to ~100 Gbit/s for larger packets >= 512 B

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

# Data Plane Performance – CBR

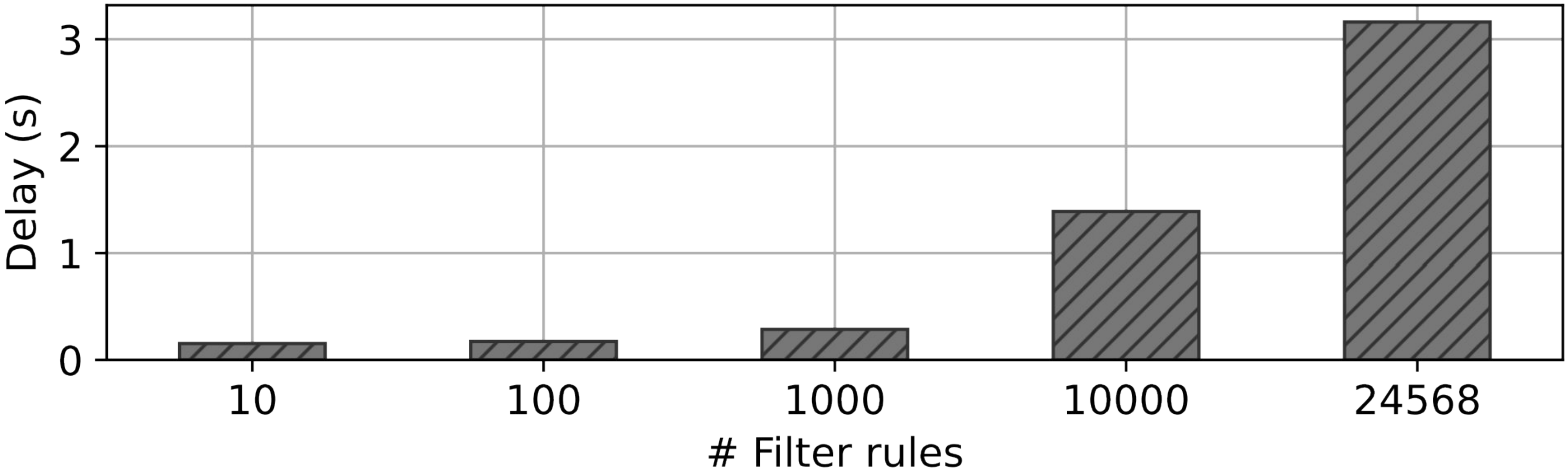

L2 throughput depending on packet size and # firewall rules


Packet throughput depending on packet size and # firewall rules

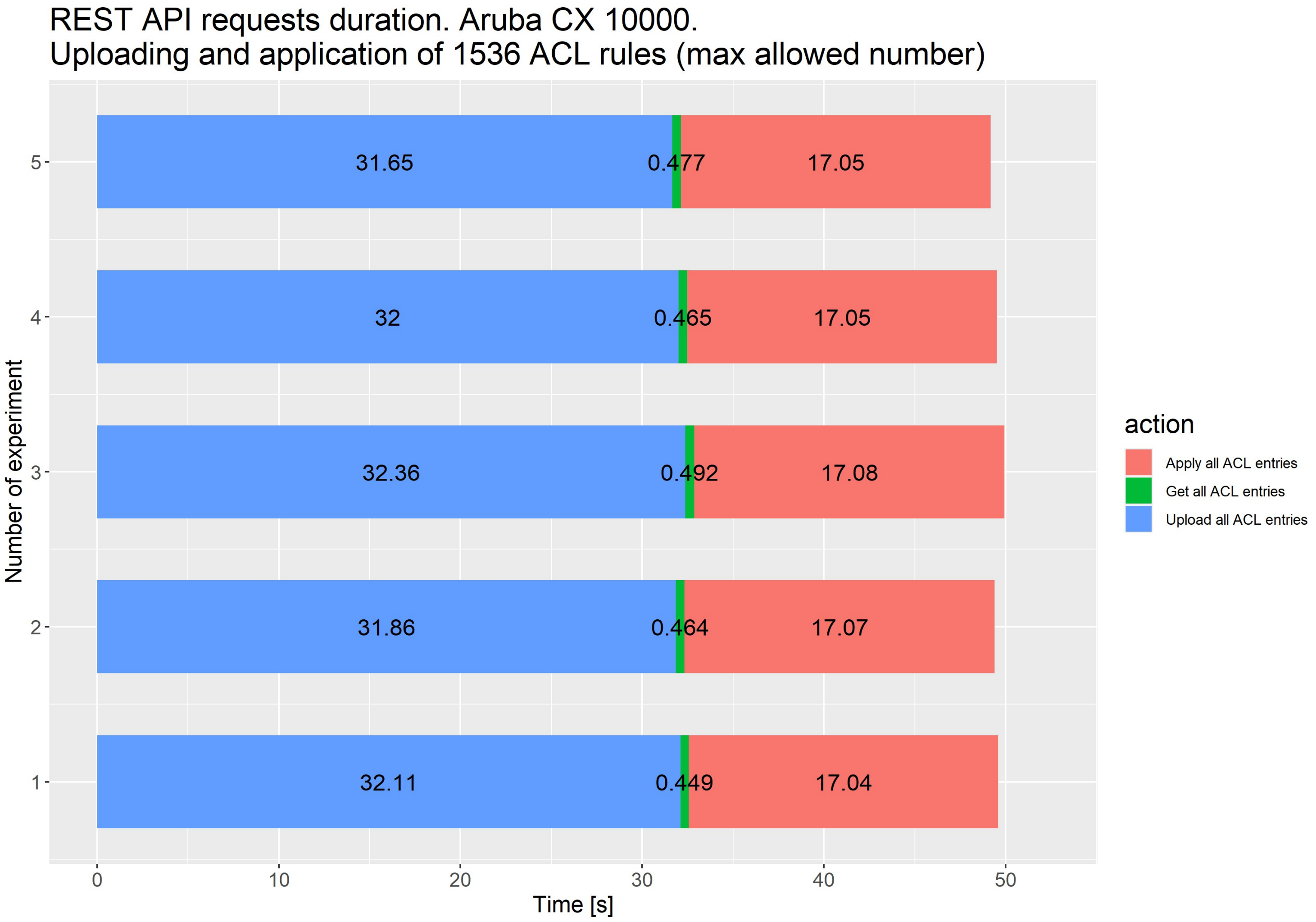
# Control Plane Performance – PSM REST API

- Automated installation of filter rules via DuT controller / PSM REST API

➡ How long is the installation time for those rules to be active the data plane?



# Control Plane Performance – AOS-CX REST API



# Conclusion

- Automated test bench provides an automated, flexible, reproducible framework for benchmarking high-performance network devices
  - Fully automated test traffic generation with P4TG and iperf3
  - Fully automated DuT configuration & metric gathering with DuT controller
  - Modular and extensible architecture – new device modules, traffic generators, or storage backends may be added
  - Declarative definition of test parameters

- Case study demonstrates the feasibility of the approach
  - Exemplary hardware evaluation of high-performance L4 packet filtering feature
  - P4-programmable ASICs built into COTS devices show promising results

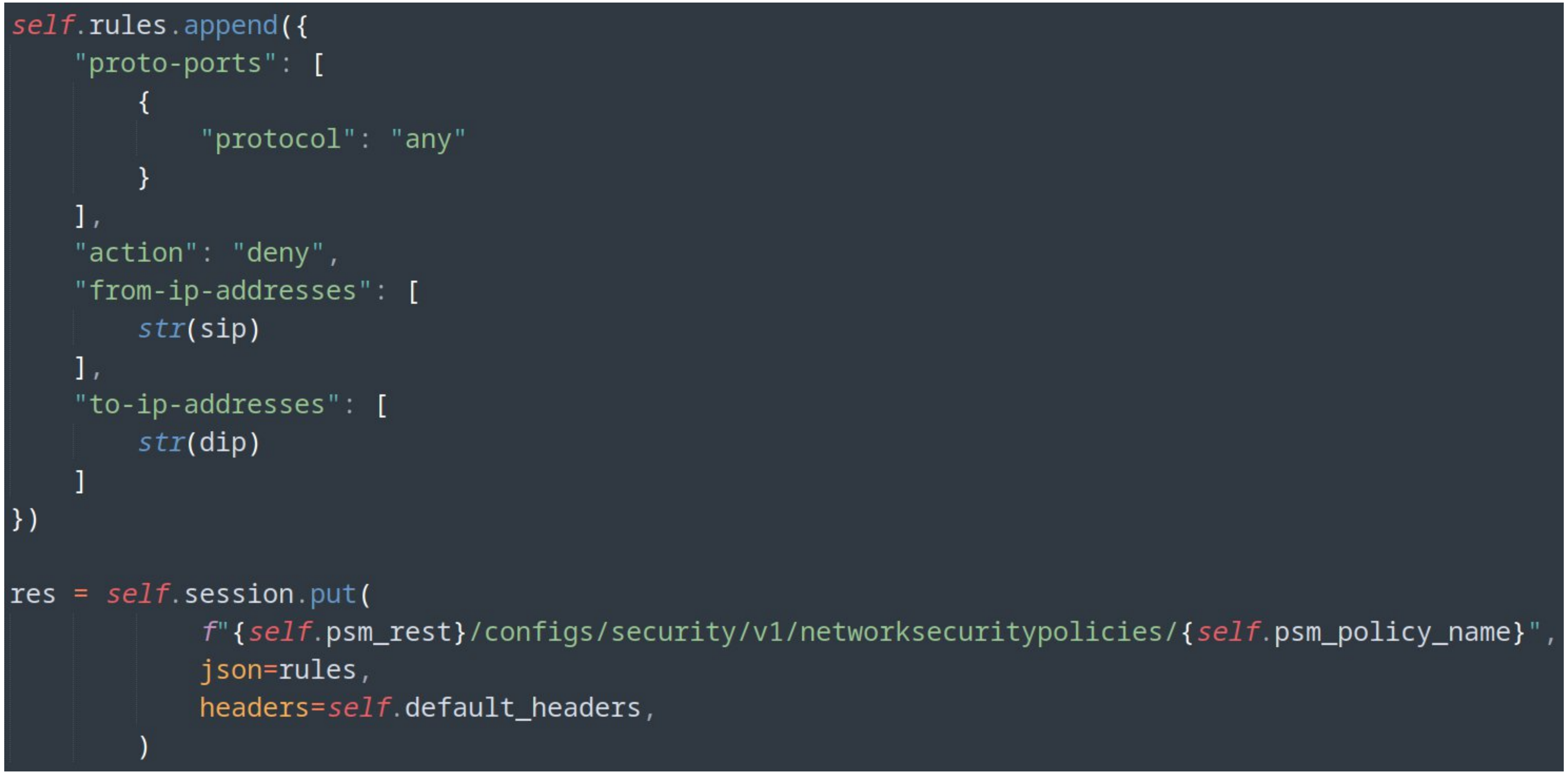
**Benjamin Steinert**[1,2], Gabriel Paradzik[1,2], Michael Menth[1]
[1]Chair of Communication Networks, [2]Zentrum für Datenverarbeitung
University of Tübingen

benjamin.steinert@uni-tuebingen.de

# THANK YOU!

# QUESTIONS?

This work was supported
by the bwNET2.0 project
funded by the MWK



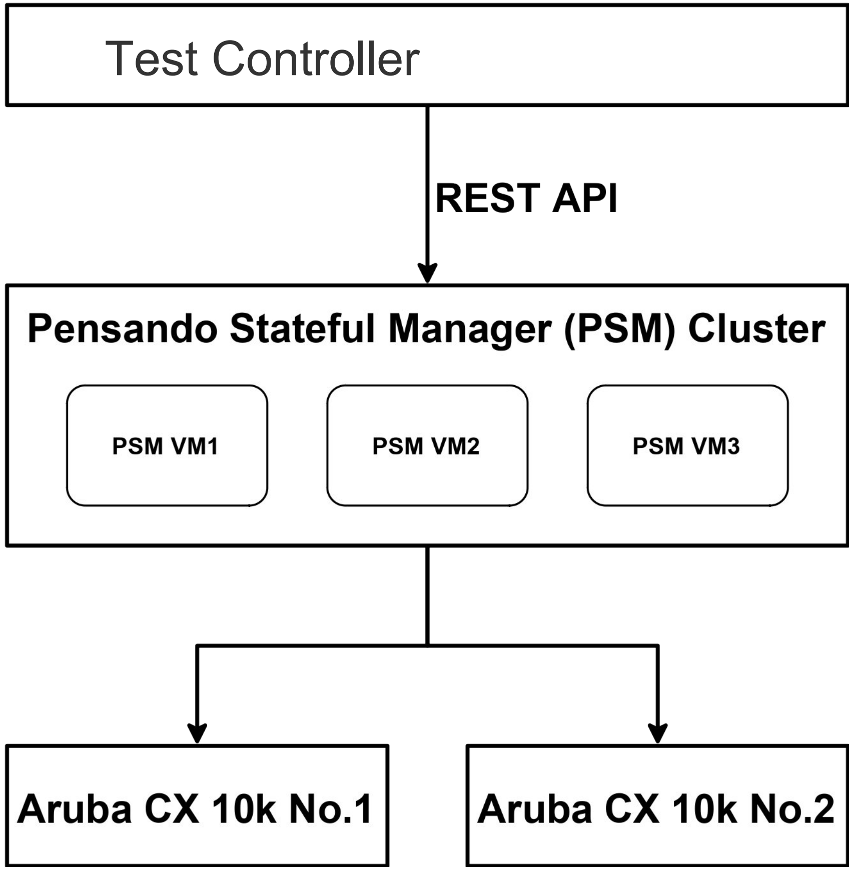
```
self.rules.append({
    "proto-ports": [
        {
            "protocol": "any"
        }
    ],
    "action": "deny",
    "from-ip-addresses": [
        str(sip)
    ],
    "to-ip-addresses": [
        str(dip)
    ]
})

res = self.session.put(
            f"{self.psm_rest}/configs/security/v1/networksecuritypolicies/{self.psm_policy_name}",
            json=rules,
            headers=self.default_headers,
        )
```

# Control Plane Performance - PSM

- Redundant setup
  - 2x Aruba CX 10k
  - 3-Node PSM Cluster
- Automated installation of filter rules via DuT controller / PSM REST API

➡ How long is the installation time for those rules to be active on both devices?


Test Controller
REST API
Pensando Stateful Manager (PSM) Cluster
PSM VM1
PSM VM2
PSM VM3
Aruba CX 10k No.1
Aruba CX 10k No.2