

A background graphic for the title section showing a network of blue nodes connected by lines, set against a light blue and grey geometric pattern.

# bwNET2020+

## Innovation im Landeshochschulnetz durch Verzahnung von Betrieb und Forschung

24. Januar 2023

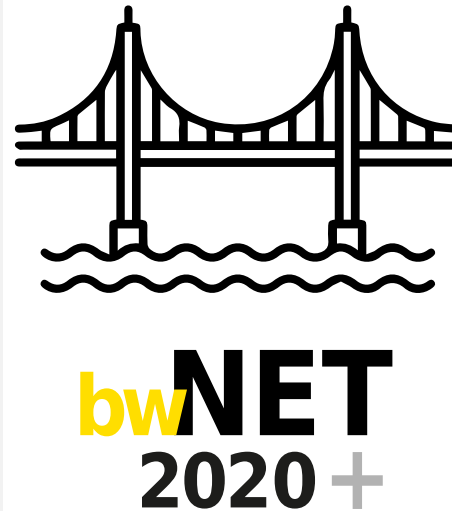
Philipp Wolter

[philipp.wolter@kit.edu](mailto:philipp.wolter@kit.edu)

# Was will man mit bwNET erreichen?

## Betrieb

- Verlässlicher Betrieb hat absoluten Vorrang
- Weiterentwicklung eher inkrementell oder evolutionär
- Geringes Zeitbudget für neue Ideen
- Neueste Ansätze aus Forschung /Industrie nicht immer bekannt



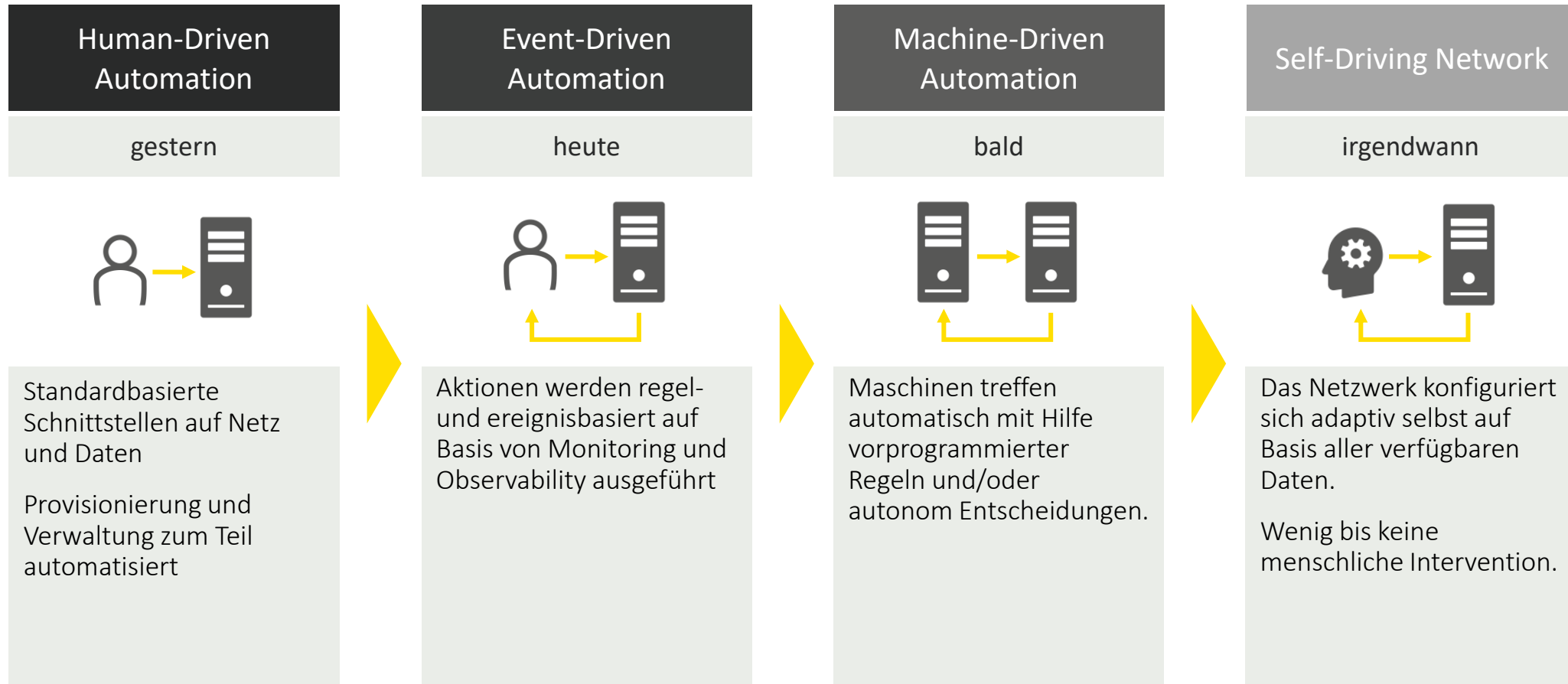
## Forschung

- Fokus auf Innovationen
- Offen für radikale oder disruptive Neuerungen
- Hohes Zeitbudget für neue Ideen
- Probleme des realen Netzbetriebs sind oft weit weg

## Arbeitsfelder und Schlüsseltechnologien

- Sicherheit, Service-Function-Chaining, Monitoring und Hochleistungsdatentransfer
- SFC, P4, SDN, NFV

# Self-Driving Networks





# Fragen? Viel Spaß mit unseren 3 Kurzvorträgen!



# P4TG: 1 Tb/s Verkehrsgenerierung und Analyse für Ethernet/IP Netze

Steffen Lindner, Marco Häberle, Michael Menth





# Motivation

- New protocols & network equipment needs to be tested with realistic traffic rates
- Traffic generators (TGs) used for this purpose
- The top 10 used TGs in the literature are all software-based!
  - iperf2
  - Netperf
  - Moongen
  - ...
- 100+ Gbit/s difficult to generate with software
  - Need hardware acceleration
  - Hardware based TGs very expensive (\$\$\$\$\$)
- Multi-Port (several 100 Gbit/s) testing for business-grade switches/routers not feasible with software TGs



# Idea

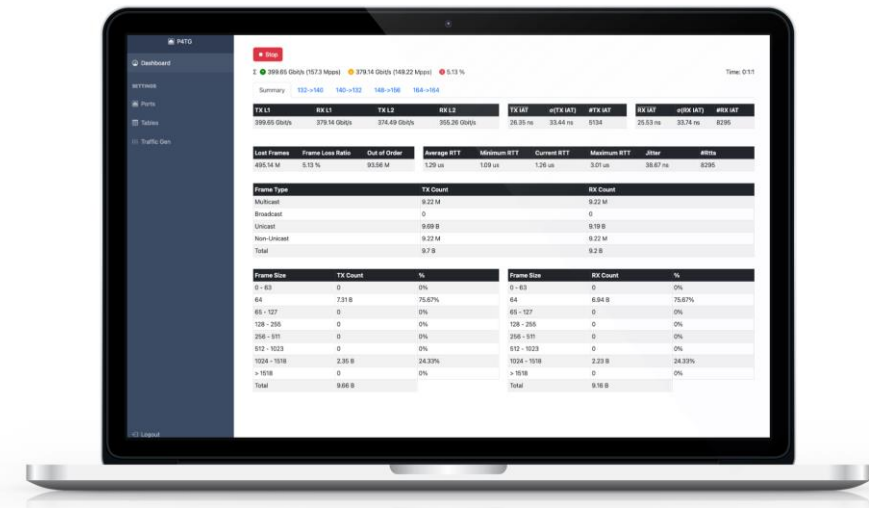
- Traffic generation with P4 and Intel Tofino™ ASIC (< 8.000€)
- Intel Tofino™ offers built-in capabilities for traffic generation
- We implement measuring functions and configuration in P4 + GUI
- Constant bit-rate & poisson traffic



+



=



# Background (I)

- Intel Tofino™ ASIC
  - 3.2 Tbit/s or 6.2 Tbit/s P4 programmable switching ASIC (Gen. 1)
    - Our Edgecore Wedge supports 32x 100 Gbit/s ports
  - 12.8 Tbit/s P4 programmable switching ASIC with 32x 400 Gbit/s ports (Gen. 2)
  - 25.6 Tbit/s P4 programmable switching ASIC with 64x 400 Gbit/s ports (Gen. 3)
- Intel Tofino™ ASIC allows for internal traffic generation
  - Up to 8 different packet (byte) descriptions with periodic timer for packet generation



<https://www.edge-core.com/productsInfo.php?id=335>



# Background (II)

- **P4: Programming protocol-independent packet processors**
  - High-level programming language to describe data planes
  - Target-specific compiler maps P4 program to hardware

```
control MyPipeline(inout headers hdr, inout metadata meta, inout
standard_metadata_t std_meta) {

  /* Declarations region */
  table ipv4_lpm { ... }
  action ipv4_forward(...) { ... }
  ...

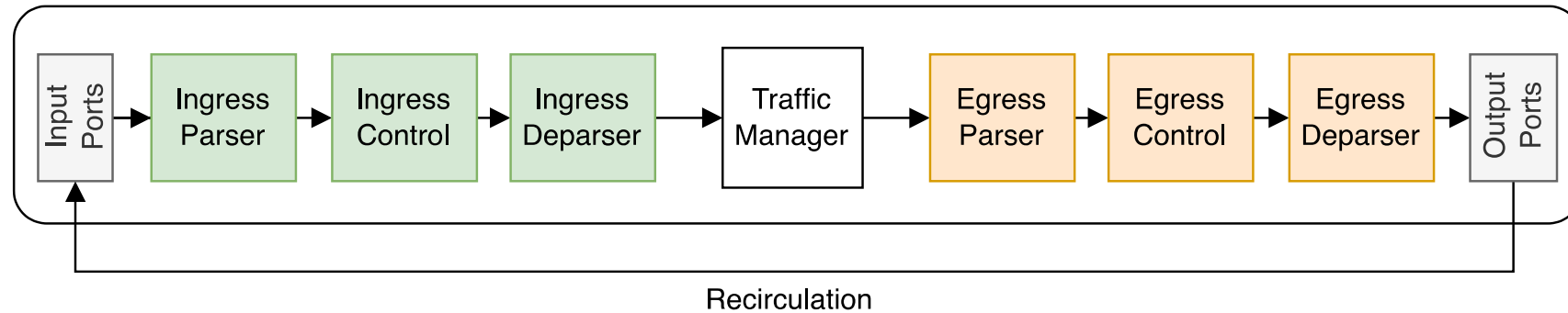
  apply {
    /* Control Flow */
    if(hdr.ipv4.isValid()){
      ipv4_lpm.apply();
    }
  }
}
```

Target-specific P4  
compiler

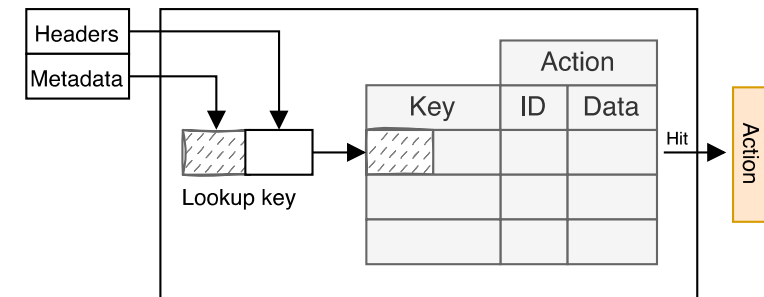
P4 target

- P4 defines low level (packet processing) operations
- ⇒ Fully programmable data plane
  - Limited only by expressiveness and features of P4 (and not by vendor)

# Background (III)



- P4-programmable
  - Ingress/Egress Parser
  - Ingress/Egress Control
  - Ingress/Egress Deparser



Match+action table used in ingress/egress control

# Concept (I)

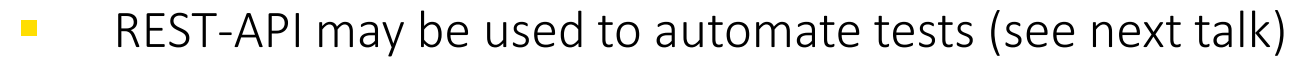
---

- Leverage internal traffic generator for packet generation
- Packet header rewrite (Ethernet & IPv4) for traffic randomization
- Up to 10x 100 Gbit/s traffic generation
  
- Measure several metrics directly in the data plane (P4)
  - L1/L2 TX & RX rates
  - Per stream TX & RX rates
  - TX & RX frame sizes and types (unicast, multicast, broadcast)
  - Packet loss, out of order, round-trip-time (RTT; sampled)
  - TX & RX inter-arrival times (sampled)

# Concept (II)

---

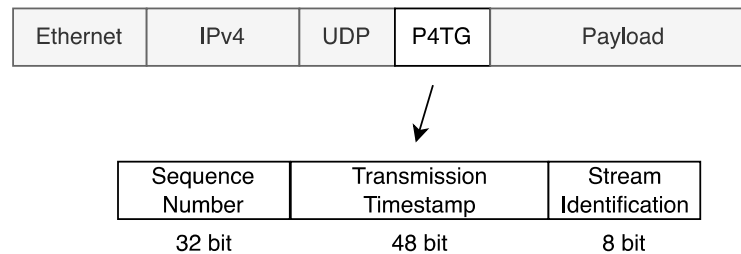
- Measurements
  - 64-bit registers to store total TX & RX bytes per port
    - Hardware timestamps with nanosecond precision for rate calculation
    - Tcpdump timestamp accuracy  $\sim 100\mu\text{s}$
  - 64-bit registers to store # of lost and out-of-order packets
  
- Collected statistics are regularly polled by the control plane
  - Monitoring packets retrieve stored measurements
  - Including hardware timestamps





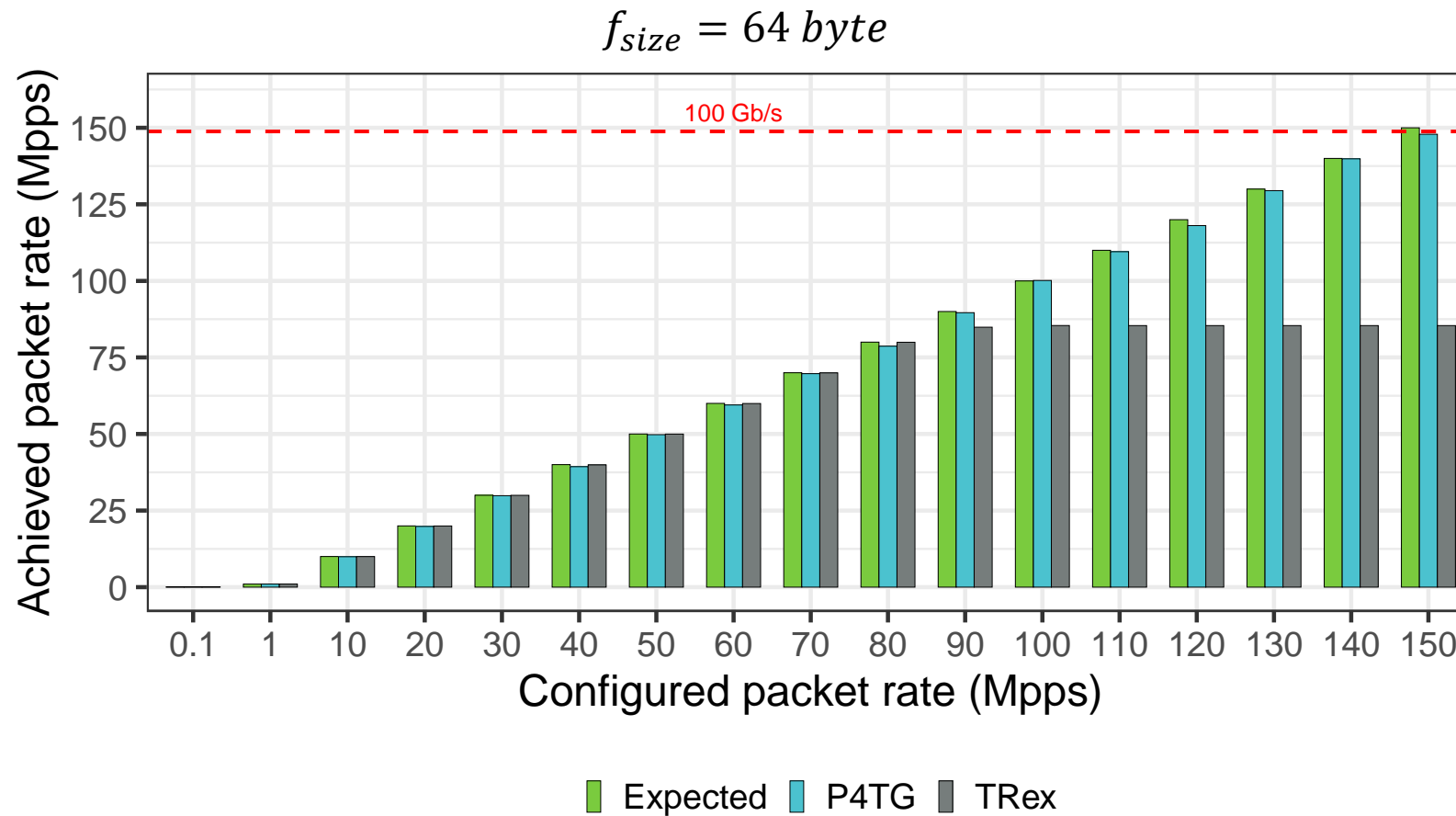
# Traffic Generation (I)

- Generated packets contain Ethernet, IPv4, UDP, P4TG header



- 32-bit sequence number for packet loss & out-of-order detection
- 48-bit timestamp for RTT calculation
- 8-bit stream identification

# Traffic Generation (II)



“Demo”



CBR

Stream-ID	Frame Size	Rate	Mode	Options
1	256 bytes	80 Gbps	Rate Precision	
2	1280 bytes	20 Gbps	Rate Precision	

[+ Add stream](#)

Up to 7 different streams

TX Port	RX Port	Stream 1	Stream 2
1 (132)	2 (140)	<input checked="" type="checkbox"/> ⚙	<input checked="" type="checkbox"/> ⚙
2 (140)	1 (132)	<input checked="" type="checkbox"/> ⚙	<input checked="" type="checkbox"/> ⚙
3 (148)	4 (156)	<input checked="" type="checkbox"/> ⚙	<input checked="" type="checkbox"/> ⚙
4 (156)	3 (148)	<input checked="" type="checkbox"/> ⚙	<input checked="" type="checkbox"/> ⚙
6 (172)	8 (188)	<input checked="" type="checkbox"/> ⚙	<input checked="" type="checkbox"/> ⚙
7 (180)	Select RX Port	<input type="checkbox"/> ⚙	<input type="checkbox"/> ⚙
8 (188)	6 (172)	<input checked="" type="checkbox"/> ⚙	<input checked="" type="checkbox"/> ⚙
9 (56)	10 (48)	<input checked="" type="checkbox"/> ⚙	<input checked="" type="checkbox"/> ⚙
10 (48)	9 (56)	<input checked="" type="checkbox"/> ⚙	<input checked="" type="checkbox"/> ⚙

[✓ Save](#)[✕ Reset](#)



CBR

Stream-ID	Frame Size	Rate	Mode	Options
1	256 bytes	80 Gbps	Rate Precision	
2	1280 bytes	20 Gbps	Rate Precision	

[+ Add stream](#)

TX Port	RX Port	Stream 1	Stream 2
1 (132)	2 (140)	<input checked="" type="checkbox"/> ⚙	<input checked="" type="checkbox"/> ⚙
2 (140)	1 (132)	<input checked="" type="checkbox"/> ⚙	<input checked="" type="checkbox"/> ⚙
3 (148)	4 (156)	<input checked="" type="checkbox"/> ⚙	<input checked="" type="checkbox"/> ⚙
4 (156)	3 (148)	<input checked="" type="checkbox"/> ⚙	<input checked="" type="checkbox"/> ⚙
6 (172)	8 (188)	<input checked="" type="checkbox"/> ⚙	<input checked="" type="checkbox"/> ⚙
7 (180)	Select RX Port	<input type="checkbox"/> ⚙	<input type="checkbox"/> ⚙
8 (188)	6 (172)	<input checked="" type="checkbox"/> ⚙	<input checked="" type="checkbox"/> ⚙
9 (56)	10 (48)	<input checked="" type="checkbox"/> ⚙	<input checked="" type="checkbox"/> ⚙
10 (48)	9 (56)	<input checked="" type="checkbox"/> ⚙	<input checked="" type="checkbox"/> ⚙

☒ Save☒ Reset

Up to 10 different ports

Stream configuration for port





Stop

Σ

799.44 Gbit/s (305 Mpps)

799.44 Gbit/s (305 Mpps)

Time: 0:0:30

Summary132->140140->132148->156156->148172->188188->17256->4848->56

TX L1	RX L1	TX L2	RX L2
799.44 Gbit/s	799.44 Gbit/s	750.64 Gbit/s	750.64 Gbit/s

TX IAT	σ(TX IAT)	#TX IAT	RX IAT	σ(RX IAT)	#RX IAT
35.05 ns	29.89 ns	2841	32.15 ns	26.89 ns	4793

Lost Frames	Frame Loss Ratio	Out of Order
0	0.00 %	0

Average RTT	Minimum RTT	Current RTT	Maximum RTT	Jitter	#Rtts
1.16 us	1.1 us	1.16 us	1.2 us	13.13 ns	4783

Frame Type	TX Count	RX Count
Multicast	914.03 K	913.84 K
Broadcast	0	0
Unicast	9.27 B	9.27 B
Non-Unicast	914.03 K	913.84 K
Total	9.27 B	9.27 B

Frame Size	TX Count	%
0 - 63	0	0%
64	0	0%
65 - 127	0	0%
128 - 255	0	0%
256 - 511	8.69 B	94.96%
512 - 1023	0	0%
1024 - 1518	461.42 M	5.04%
> 1518	0	0%
Total	9.15 B	

Frame Size	RX Count	%
0 - 63	0	0%
64	0	0%
65 - 127	0	0%
128 - 255	0	0%
256 - 511	8.69 B	94.96%
512 - 1023	0	0%
1024 - 1518	461.39 M	5.04%
> 1518	0	0%
Total	9.15 B	














# Conclusion

- P4TG offers traffic generation at high data rates (up to 100 Gbit/s per port)
  - Up to 400 Gbit/s with 2. / 3. Gen. Tofino
- Low-cost hardware TG
- Customizable for individual needs
  - Both data and control plane
-  <https://github.com/uni-tue-kn/P4TG>





# bwNET2020+

## Evaluation von 100G Hardware & redundantes L4 Packet Filtering

Benjamin Steinert (TÜ/ZDV), Gabriel Paradzik (TÜ/ZDV),  
Philipp Wolter (KIT/SCC), Oleksandr Miroshkin (UULM/KIZ)

# Gerätemarkt

---

- Hersteller fangen an programmierbare Chips in ihre Geräte zu bauen
  - AMD/Pensando P4 DSM ASIC in Aruba CX 10000
  - Juniper stellt programmierbares „Trio“ Chipset auf SIGCOMM'22 vor
  - Intel Tofino P4 ASIC (z.B. Edgecore Wedge 100BF-32X)
  - Nvidia Mellanox Spectrum ASIC (z.B. in SN4410)
  - Cisco Silicon One P4 ASIC (z.B. Catalyst 9500)
  
- Mikrotik hat erstmals 100G Gerät rausgebracht (CCR2216)

→ Taugen die Geräte was?

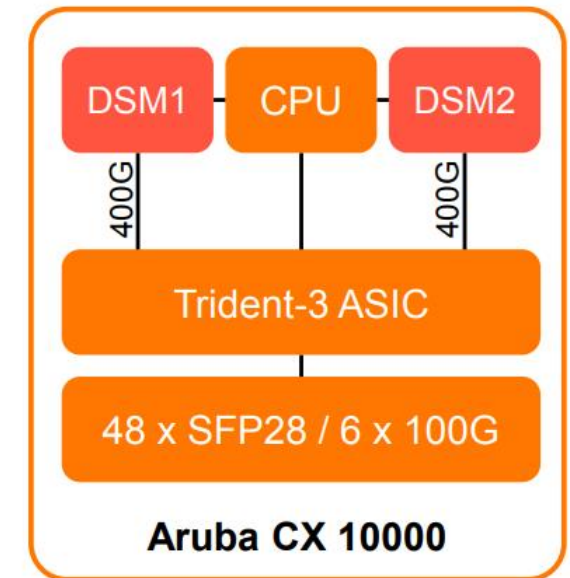
# Was können wir in die Finger bekommen?

---

- Aruba CX 10k mit AMD/Pensando P4 DSM ASIC
    - Verspricht 100G L4 stateful Firewalling
  - Edgecore Wedge 100BF-32X mit Intel Tofino ASIC
    - Komplett programmierbar für 100G Packet Processing inkl. L4 Filtering
    - Géant RARE mit freeRtr Software verfügbar
  - Mikrotik CCR2216
    - Hat bereits in dem einen oder anderen RZ Einzug gefunden
- Start mit Aruba CX 10k – Teststellung des Herstellers

# Aruba CX 10000

- Portdichte: 48x 1/10/25G + 6x 40/100G Ports
- Zwei verschiedene ASICs eingebaut
  - AMD/Pensando P4 DSM ASIC für 100G L4 stateful FW
  - Broadcom Trident 3 for Switching & Routing
- Zwei Softwares zur Steuerung der Chips
  - AOS-CX Software zur Steuerung von Broadcom Chip
  - Pensando Stateful Manager (PSM) Software zur Steuerung von Pensando Chip
  - Getrennte REST APIs für Automatisierungen

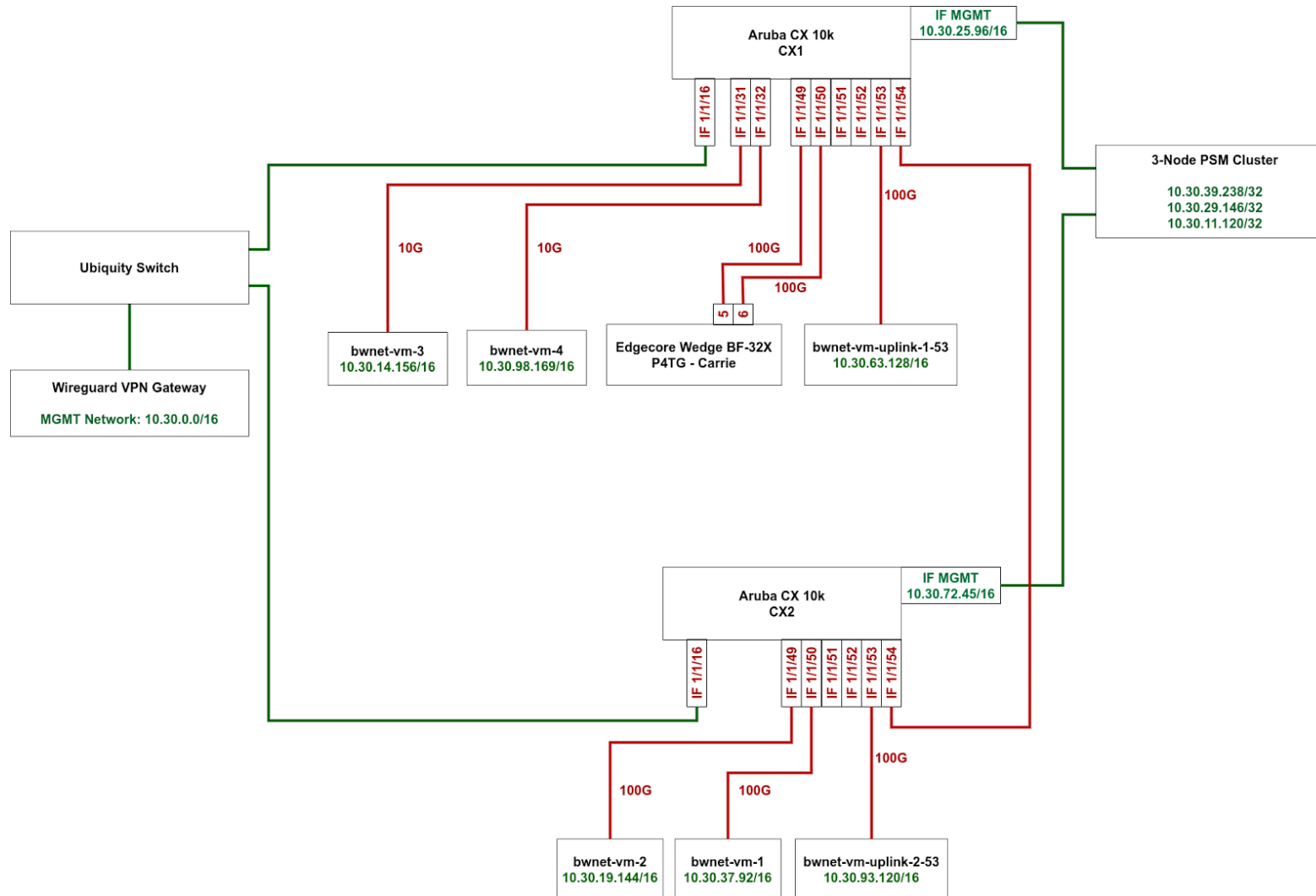


# Evaluation von 100G Hardware

---

- Verschiedene Bereiche angeschaut:
  - Hardware & Software
  - Monitoring
  - Management & Automatisierung
  - Sicherheit
  - ...
  
- Zusammenarbeit verschiedener Standorte inkl. Einbeziehung „Externer“
  - Tübingen, Ulm, Karlsruhe, Stuttgart
  - Hohenheim

# Testbett-Aufbau



# Ausschnitt aus Erfahrungen

---

- Leistungstarker Pensando P4 ASIC mit großem Potenzial
  - 100G Stateful Firewalling, weitere stateful Services sollen kommen
  - Nette Features (PSM Diagnostics Feature, Feingranulares RBAC mit LDAP- und RADIUS- Integration für PSM, ...)
  - PSM REST API intuitiv, gut benutzbar, und ordentlich dokumentiert
  
- Manches fühlt sich nicht 100% rund an
  - 2 Geräte in einem Chassis, dadurch erhöhte Komplexität mit gewisser Lernkurve
  - Flows können *im Moment* **nicht unterbrochen** werden (soll kommen)
  - Kein IPv6 (soll kommen)
  - AOS-CX REST API funktioniert anders als PSM REST API

# Redundantes L4 Packet Filtering

## **Disclaimer**

Gezeigte Ergebnisse werden im  
Nachgang zur Verfügung gestellt.



# Redundantes L4 Packet Filtering bis 100G

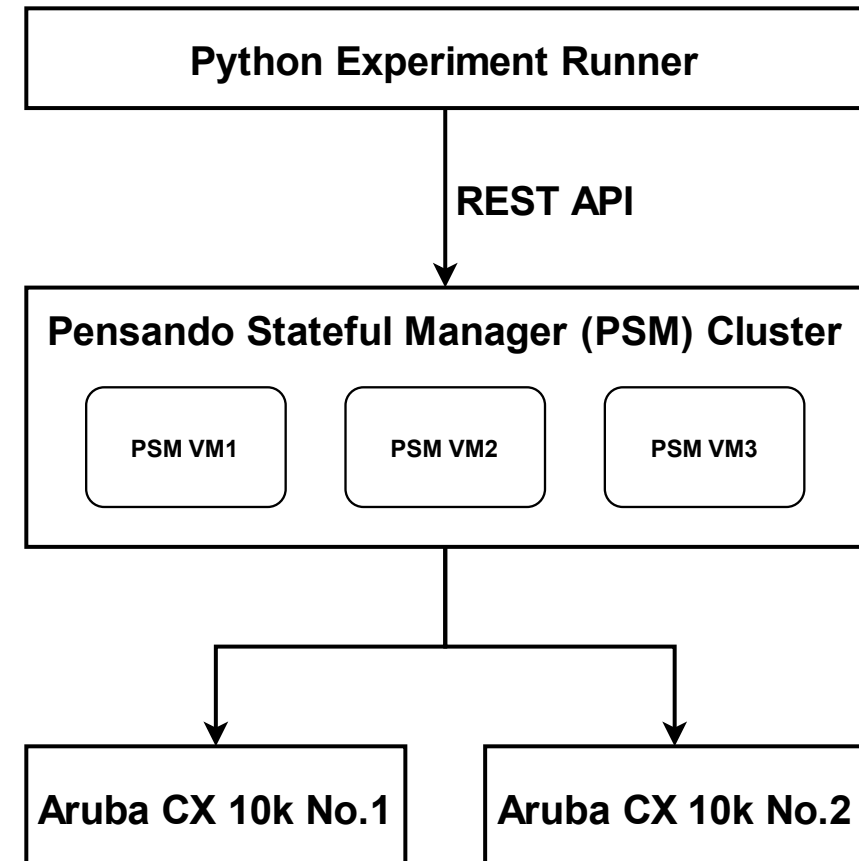
---

- Problem: „Richtige“ FWs teuer & aufwändig im Betrieb
- Lösung: Möglicherweise Einsatz von aktueller Hardware für „Basisschutz“
- Interessante Kernmetriken für FW / Packet Filtering (RFC 3511):
  - Max. Durchsatz & RTT abhängig von Paketgröße & Regelanzahl
  - Max. Durchsatz & RTT bei IMIX abhängig von Regelanzahl
  - Max. Connections per Second (CPS)
  - Max. Anzahl gleichzeitig aktiver Verbindungen
  - Control Plane Performanz – Installationszeit für X Regeln

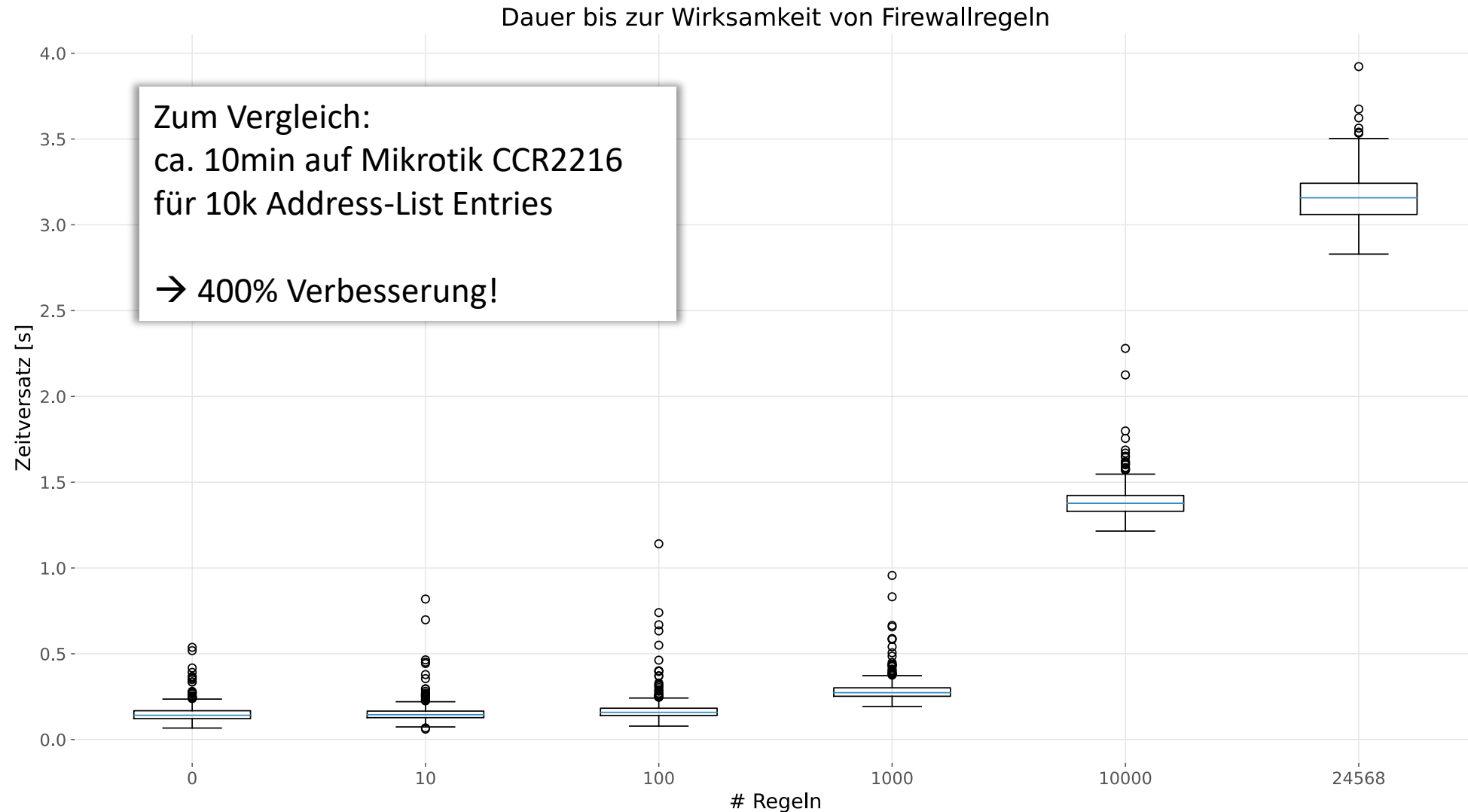
→ Reproduzierbare Messungen mit P4TG!

# Control Plane Performanz

- Redundanz
    - 2x Aruba CX 10k
    - 3-Node PSM Cluster
  - Automatisierte Installation neuer Regeln via PSM REST API
- Wie lange dauert es bis Regeln auf beiden Geräten aktiv sind?



# Control Plane Performanz



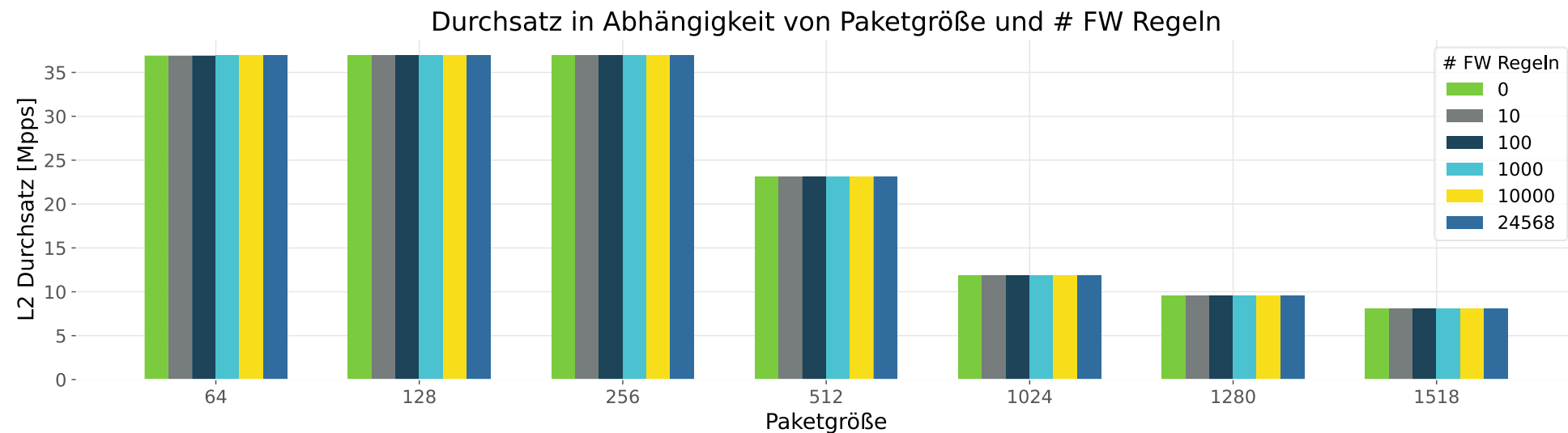
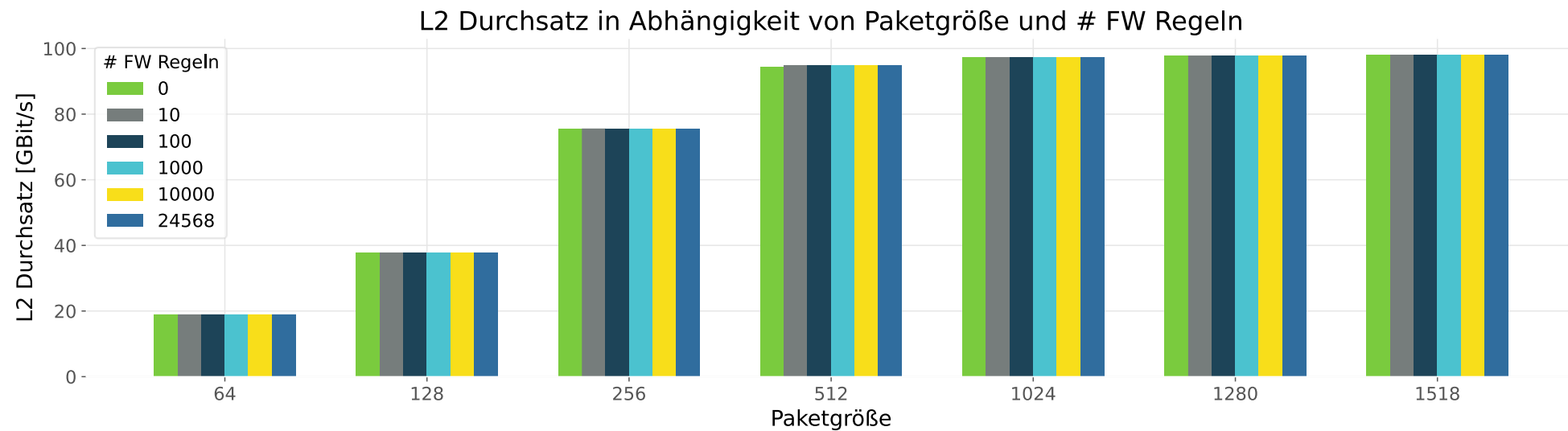
# Durchsatz bei verschiedenen Paketgrößen

---

## Methodik

- Je Messung eine feste Quell- und Ziel-IP
  - IP-Adressen ändern sich zwischen zwei Messungen
- Variiere Paketgrößen und die Anzahl der Firewall Regeln
  - Anzahl Regeln = 0, 10, 100, 1000, 10000, 24568
  - Paketgrößen = 64, 128, 256, 512, 1024, 1280, 1518
- Regeln werden zu Beginn der Messung geladen
- Der jeweilige P4TG-Verkehr besteht aus Paketen fester Größe
- Der angezeigte Durchsatz ist die höchste Bandbreite, welche nicht zu Paketverlusten führt

# Durchsatz bei verschiedenen Paketgrößen



# RTT bei verschiedenen Paketgrößen

---

## Methodik

- Je Messung eine feste Quell- und Ziel-IP
  - IP-Adressen ändern sich zwischen zwei Messungen
- Variiere Paketgrößen und die Anzahl der Firewall Regeln
  - Anzahl Regeln = 0, 10, 100, 1000, 10000, 24568
  - Paketgrößen = 64, 128, 256, 512, 1024, 1280, 1518
- Die RTT wurde während der höchsten Bandbreite, welche keine Paketverluste aufweist, gemessen

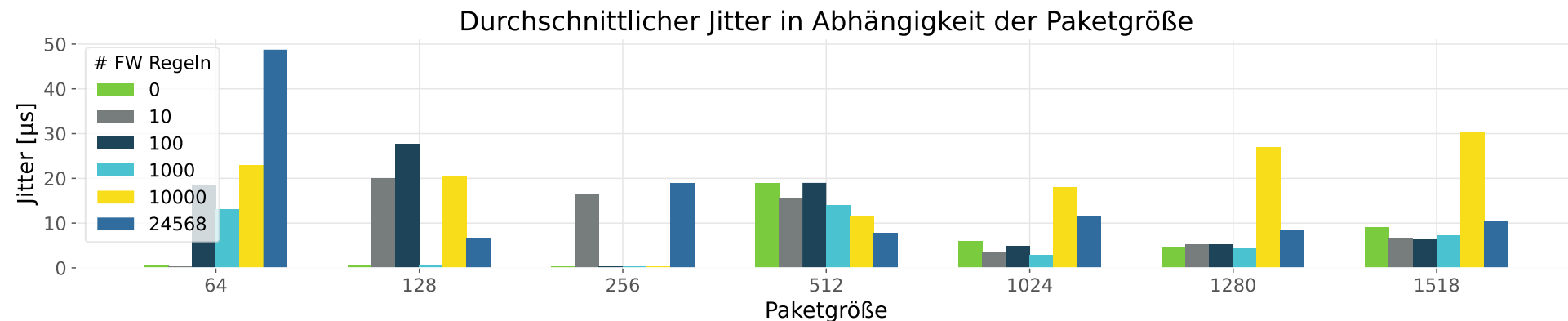
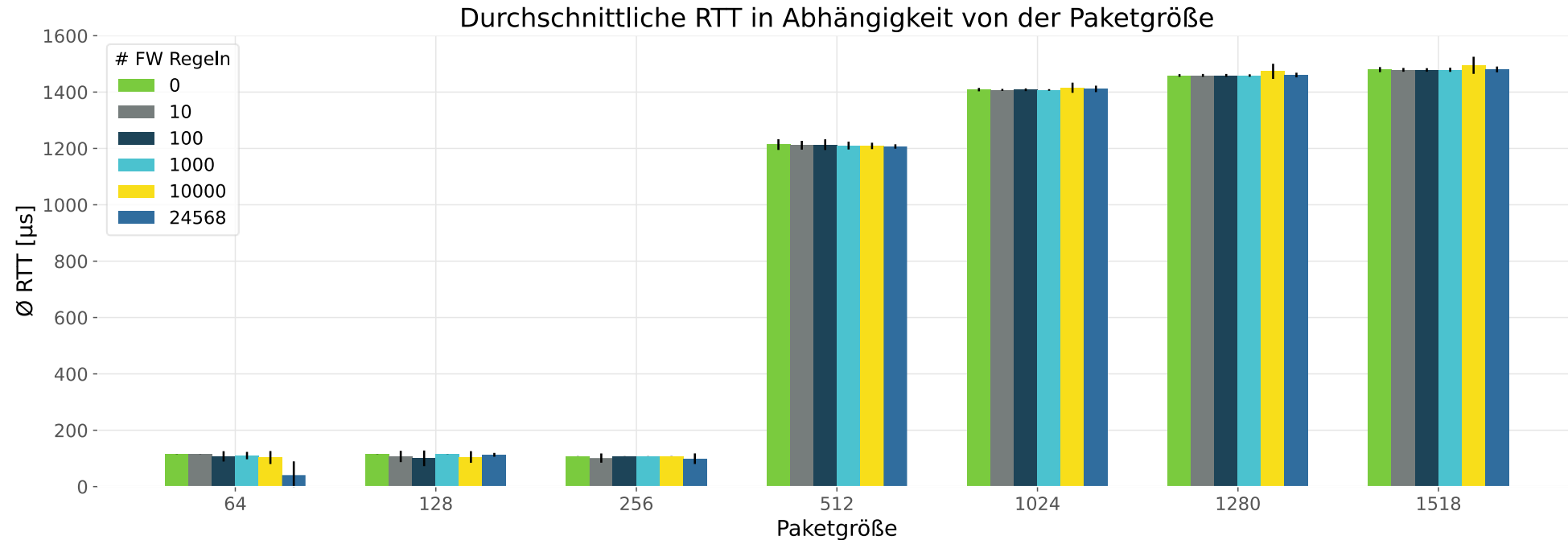
## Resultat

- RTT beträgt ca. 5  $\mu$ s für alle Konfigurationen (Jitter 10 ns)

## Methodik

- Je Messung eine feste Quell- und Ziel-IP
  - IP-Adressen ändern sich zwischen zwei Messungen
- Variiere Paketgrößen und die Anzahl der Firewall Regeln
  - Anzahl Regeln = 0, 10, 100, 1000, 10000, 24568
  - Paketgrößen = 64, 128, 256, 512, 1024, 1280, 1518
- Die RTT wurde während der niedrigsten Bandbreite, welche Paketverluste aufweist, gemessen

# RTT bei verschiedenen Paketgrößen - Überlast





# Durchsatz/RTT bei IMIX

---

## Methodik:

- Je Messung randomisierte Quell- und Ziel-IPs
  - P4TG generiert für jedes Paket zufällige IP-Adressen
- Variiere die Anzahl der Firewall Regeln:
  - Anzahl der Regeln = 0, 10, 100, 1000, 10000, 24568
- P4TG sendet IMIX Verkehr mit einer L1-Rate von 100G
- Die Bandbreite des IMIX Verkehrs besteht aus:
  - 12% 64 B Pakete
  - 54% 512 B Pakete
  - 34% 1518 B Pakete

## Resultat

- Forwarding in Line Rate unabhängig von der Regelanzahl
- RTT ca. 2.5  $\mu$ s (Jitter 15 ns)

# CPS, Sessions, Regeln

---

- Herstellerangaben:
  - Max. 2M gleichzeitig aktive Verbindungen
  - Max. 800K neue Verbindungen pro Sekunde
  - Max. Regelanzahl 1M total, 24K pro Policy

# Ausblick

---

- Messungen mit Mikrotik CCR2216
    - Inkl. Implementierung von Software zum automatisierten Verteilen von Regeln auf mehrere Geräte
  - Messungen mit Intel Tofino
- Report inkl. Vergleich der Geräte folgt

Bei Interesse an zukünftigen Evaluationen einfach bei uns melden!

# Kontakt Daten

---

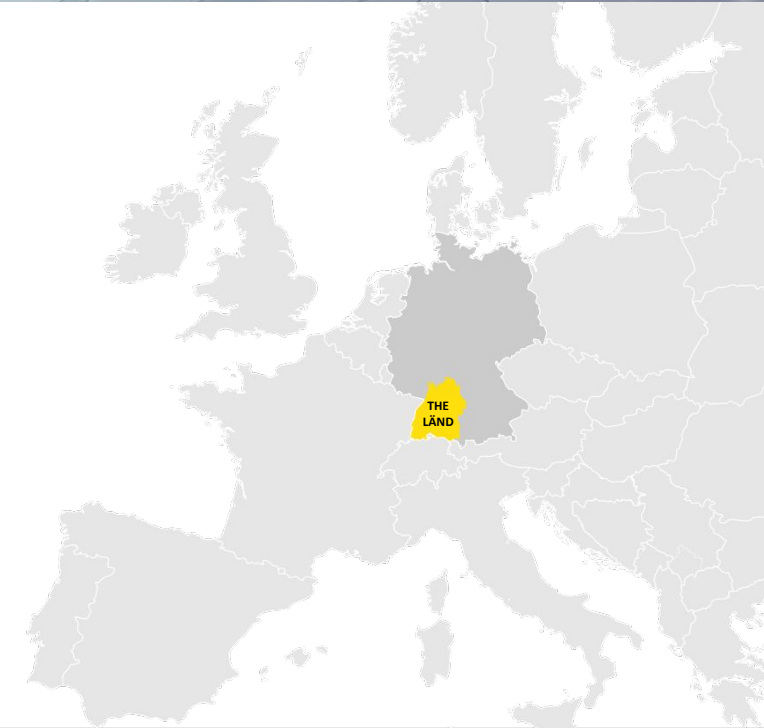
- Philipp Wolter [philipp.wolter@kit.edu](mailto:philipp.wolter@kit.edu)
- Steffen Lindner [steffen.lindner@uni-tuebingen.de](mailto:steffen.lindner@uni-tuebingen.de)
- Benjamin Steinert [benjamin.steinert@uni-tuebingen.de](mailto:benjamin.steinert@uni-tuebingen.de)
- Gabriel Paradzik [gabriel.paradzik@uni-tuebingen.de](mailto:gabriel.paradzik@uni-tuebingen.de)
- Oleksandr Miroshkin [oleksandr.miroshkin@uni-ulm.de](mailto:oleksandr.miroshkin@uni-ulm.de)
- bwNET Projektmanagement [bwnet100-pmo@lists.uni-ulm.de](mailto:bwnet100-pmo@lists.uni-ulm.de)

# BelWü Netflow API

für Unis und Hochschulen

## BelWü Tech Day

24. Januar 2023



# Netflow bei BelWü

---

BelWü nutzt Netflow an Border Interfaces um verschiedene betriebliche Fragen zu beantworten:

- Gibt es neue Peerings die sich positiv auf die Auslastung an einem IX oder Transit Interface auswirken würden?
- Was ist eine geeignete BGP Flowspec Regel um einen laufenden DDoS Angriff zu mitigieren?
- Bei welchem Teilnehmer liegen die Adressen die Traffic zu Botnetzen haben?

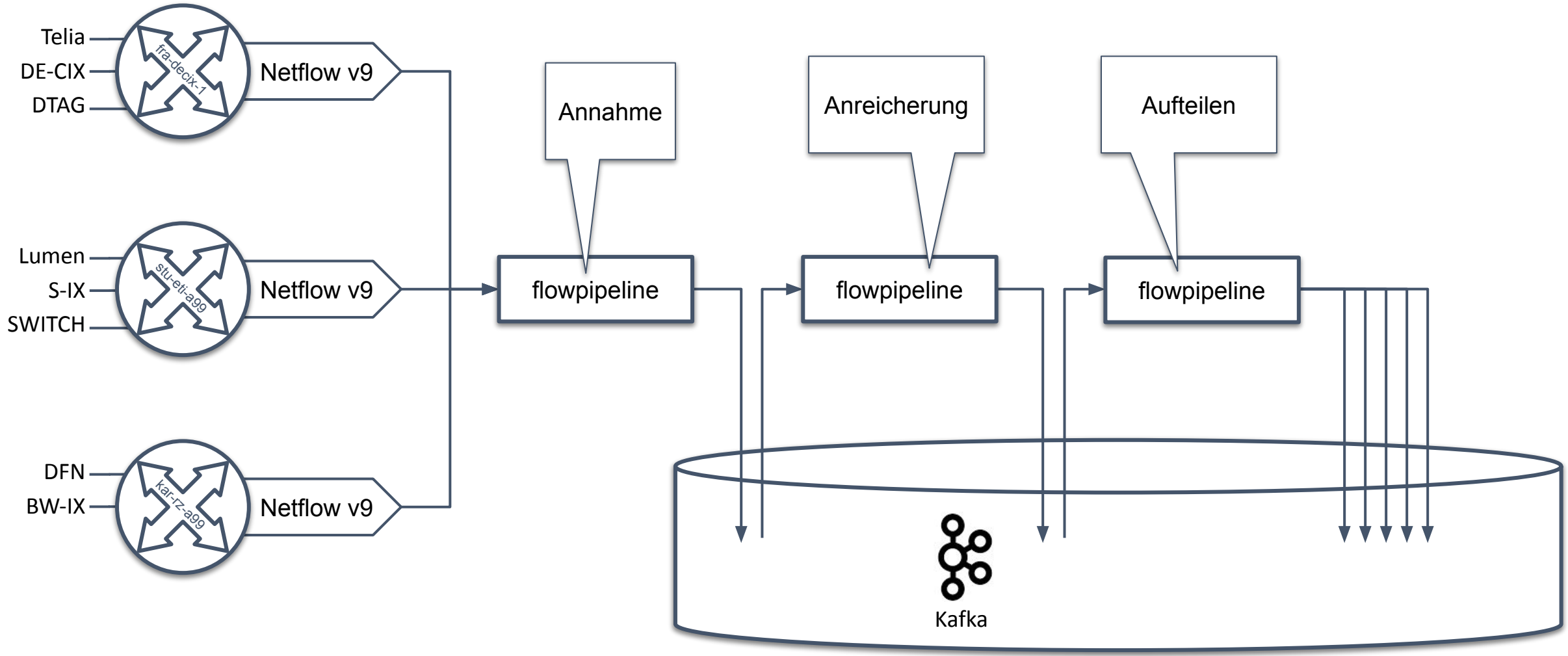
# Aktuelles Setup: flowpipeline

---

- Input von **Netflow v9**, IPFIX, NFv5, sFlow, PCAP, eBPF und **Kafka**
- Zusammenführung von verschiedenen Routern
- Anreicherung mit beliebigen Daten, insbesondere **Kundennummern**
- Kafka als Drehkreuz für alle erfassten Flows:
  - Support für Kunden-spezifische Streams
  - Support für Accounts, ACLs, und Kommunikation via TLS
  - ➔ Zugriff auf Informationen über **eigene** Flows die das BelWü verlassen

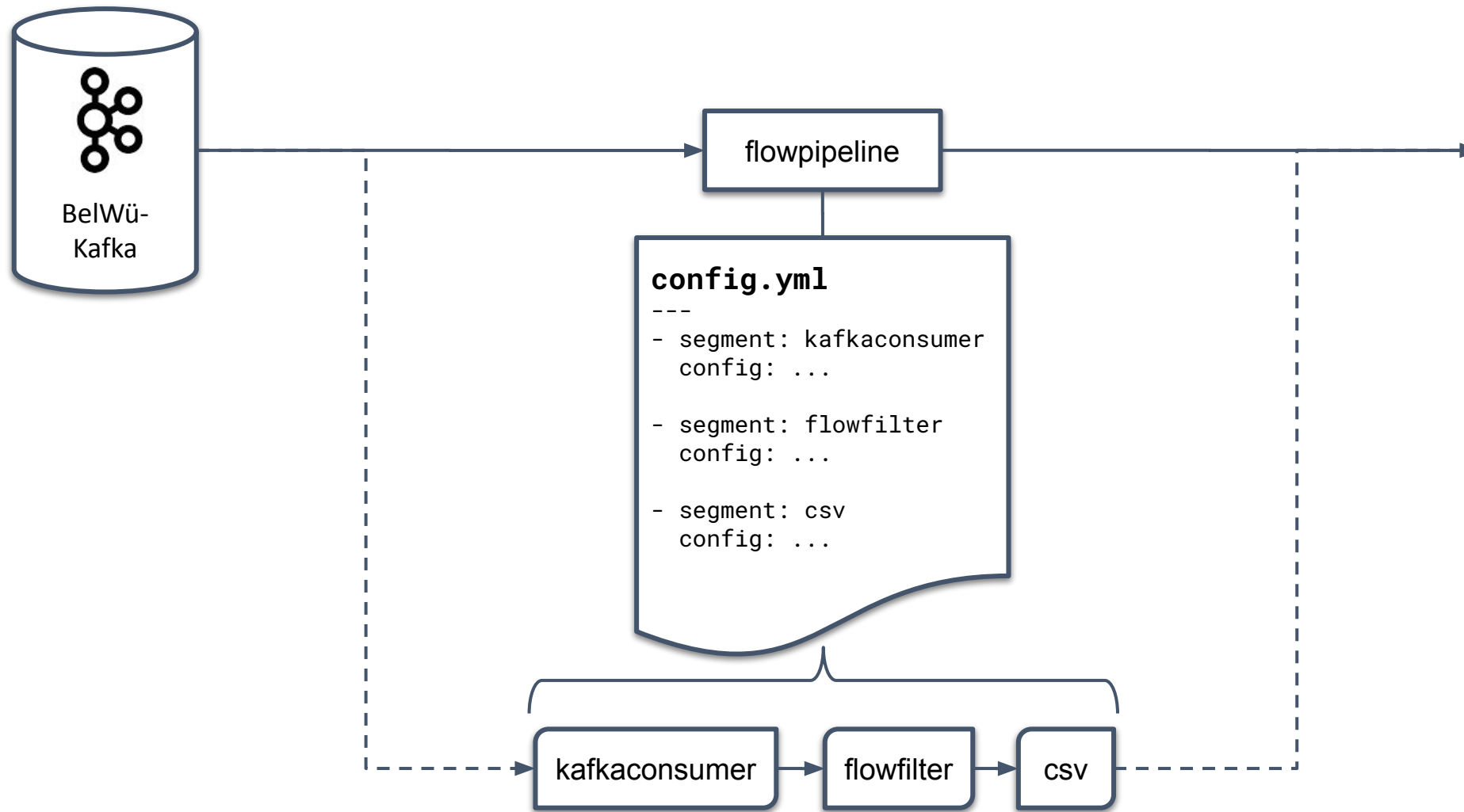
Details zur Software: [Mein Talk bei der DENO14](#) (30min)

# Flow Processing Gesamtübersicht

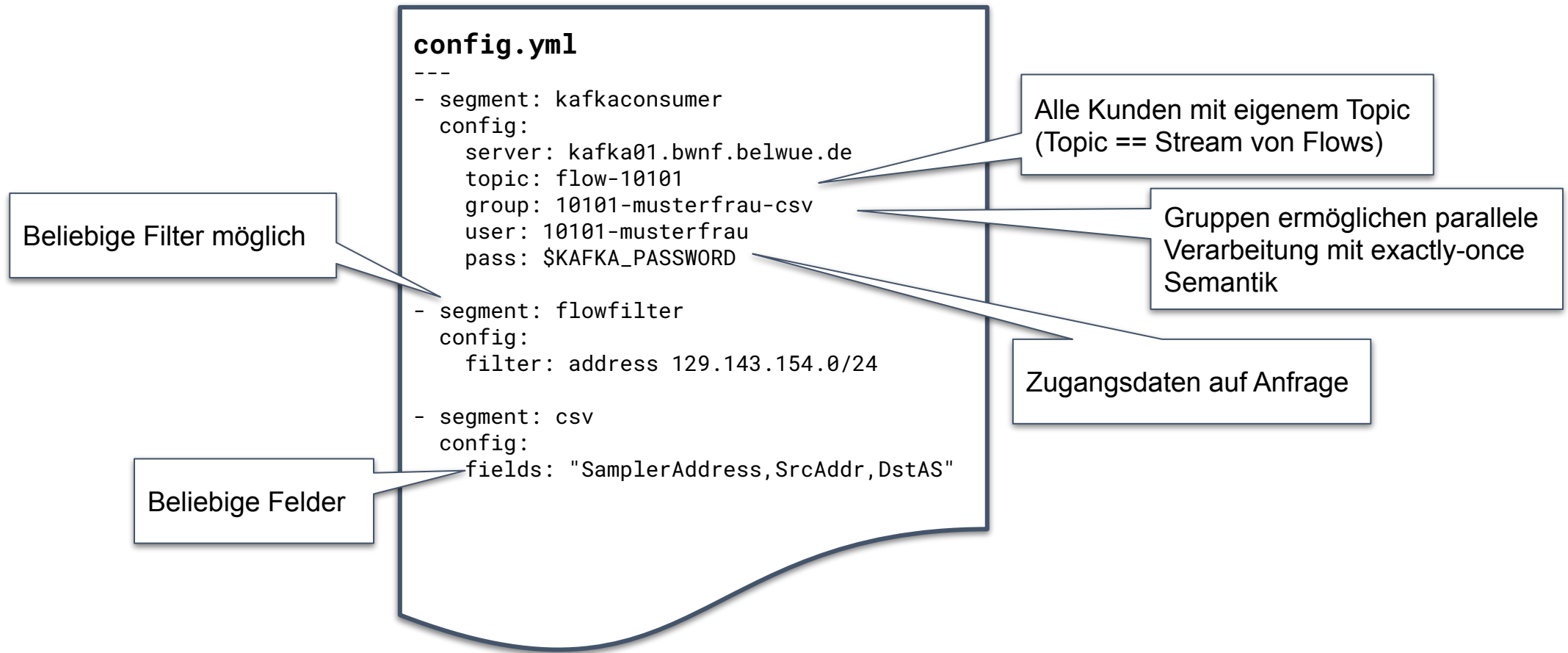




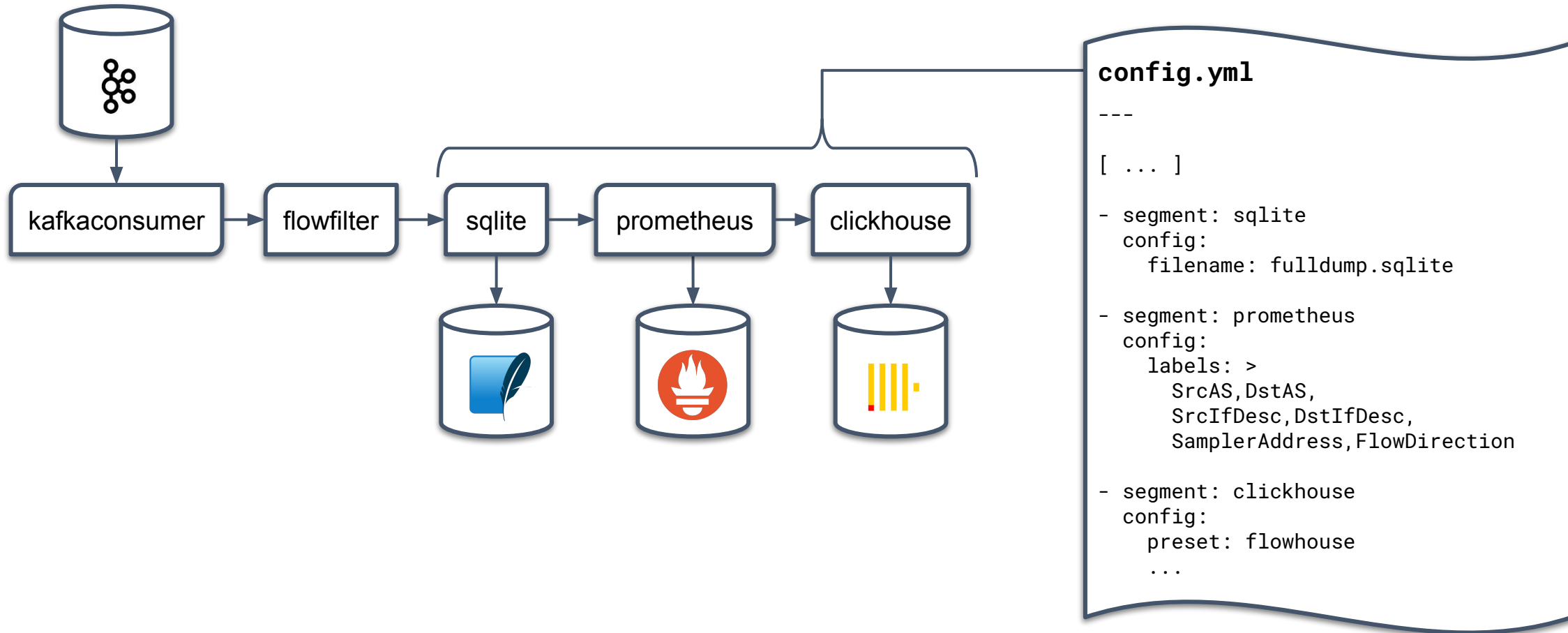
# Flow Processing als BelWü Kunde



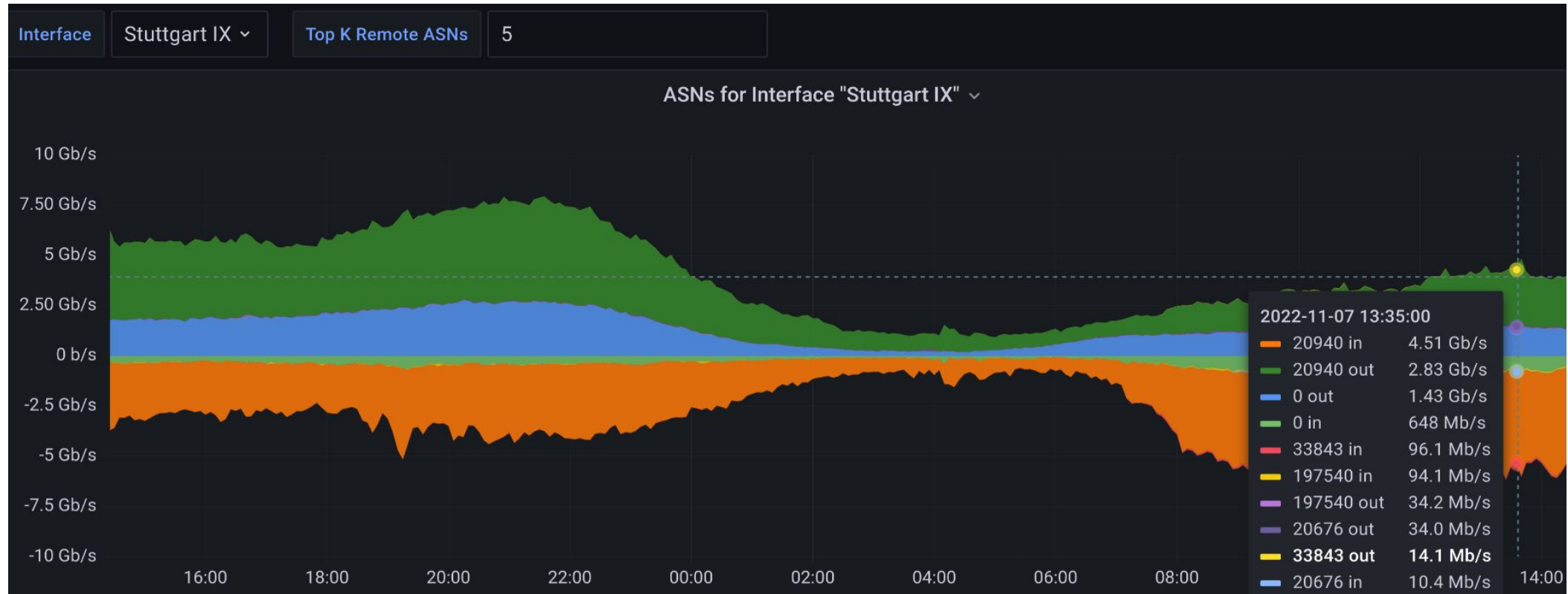
# Flow Processing als BelWü Kunde



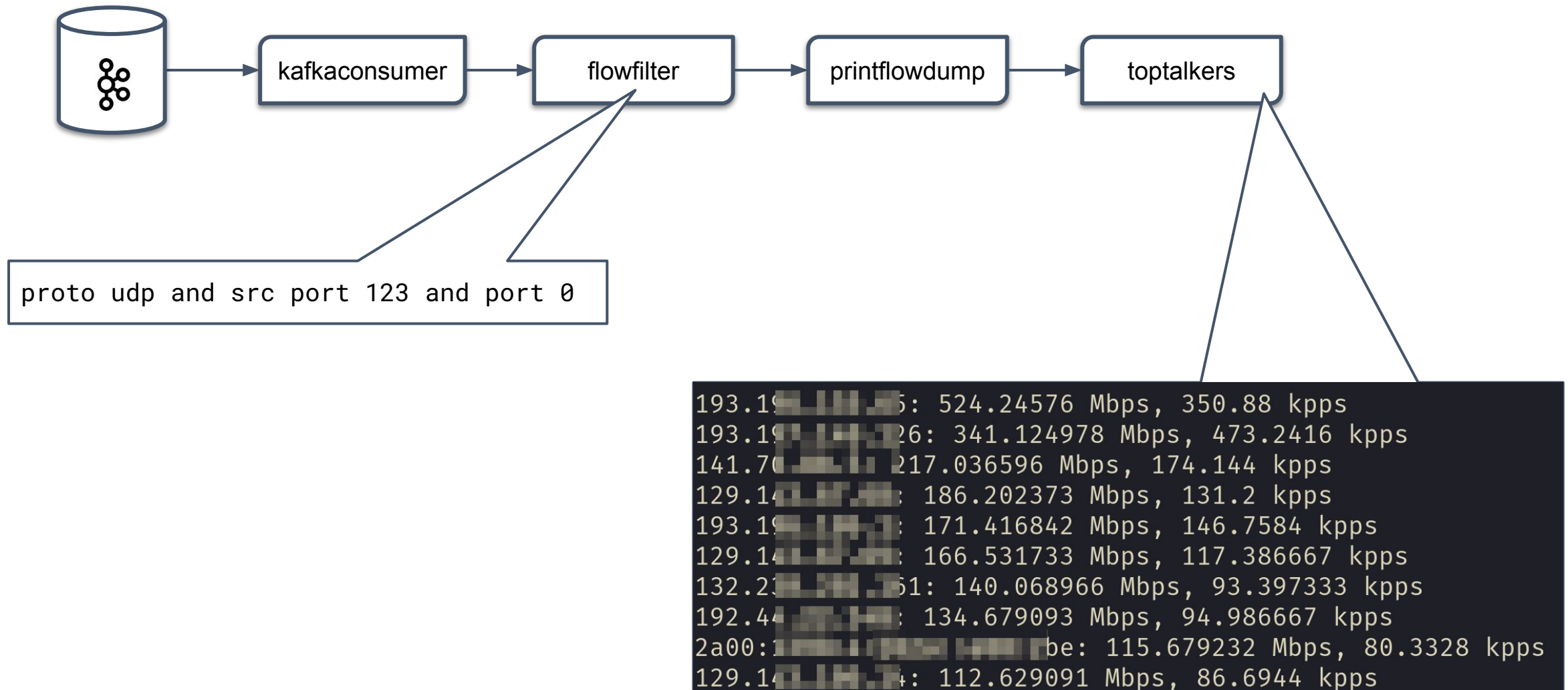
# Zusätzliche Möglichkeiten zum Export



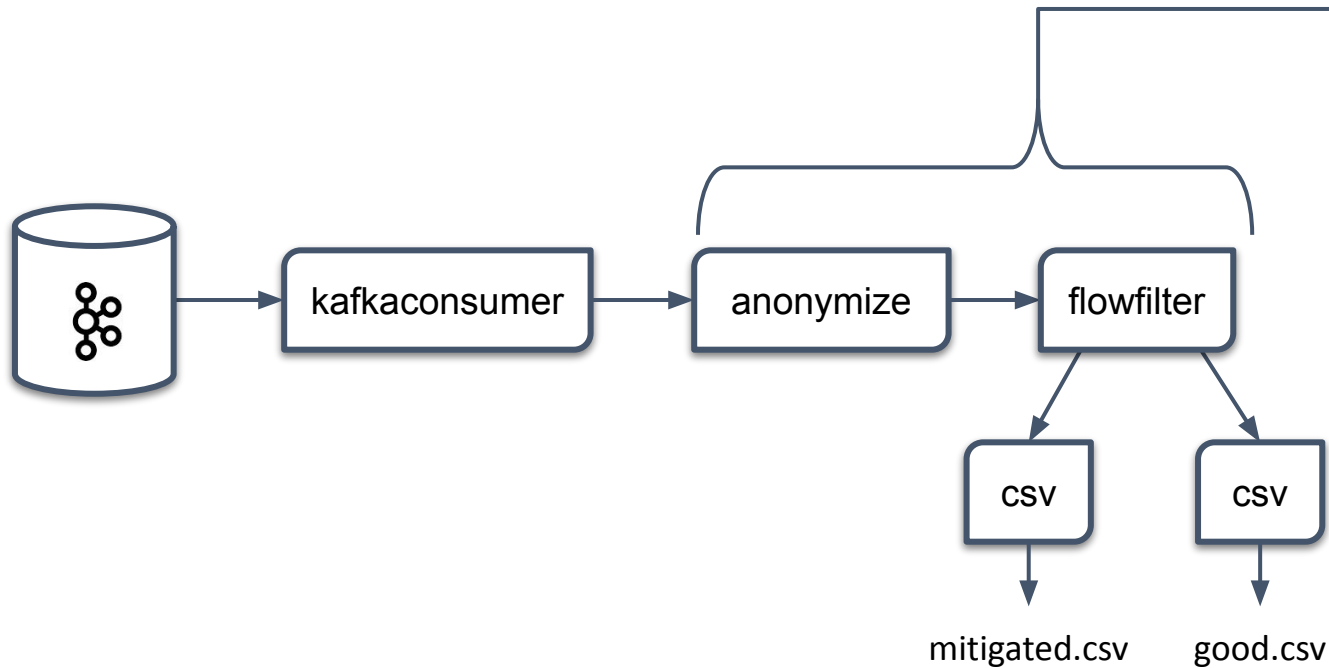
# Beispielsweise: Dashboard auf Prometheus Basis



# Beispiel: Finden des DDoS Empfängers



# Generierung von Forschungsdatensätzen



## config.yml

```

---
[ ... ]

- segment: anonymize
  config:
    key: "qwertyuiop"
    fields: SrcAddr,DstAddr

- segment: branch
  if:
    - segment: flowfilter
      config:
        filter: "... and status dropped"
    then:
      - segment: csv
        config:
          filename: mitigated.csv
    else:
      - segment: csv
        config:
          filename: good.csv
  
```

# Weitere Anwendungen

---

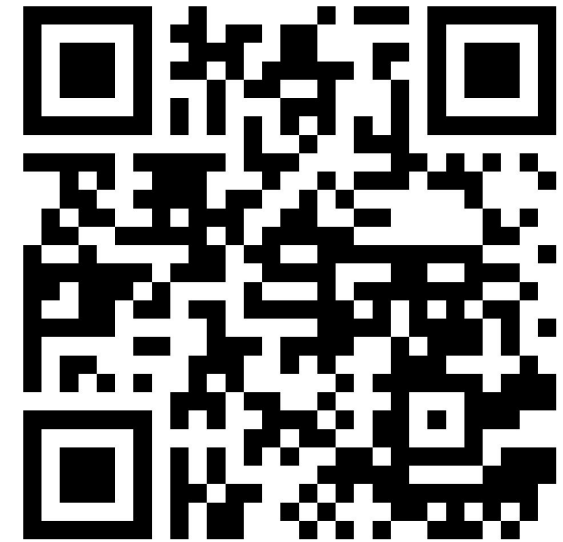
- Einstieg in das Thema Flow Monitoring ohne Hürden
  - Läuft auf dem eigenen Laptop mit Linux/MacOS/Docker
  - Einfache Konfiguration für schnelle Ergebnisse, “tcpdump-style”
  - Support für komplizierte Setups
  - Plugins erlauben zusätzliche Erweiterung in Go
- Ergänzung oder Aufbau eigener Flow Kollektoren
  - Software und Konfiguration wie bei BelWü möglich
  - Flowpipeline als einheitliche Flow Schnittstelle für das NOC

# Danke für eure Aufmerksamkeit!

---

Weitere Fragen?

<https://github.com/bwNetFlow/flowpipeline>

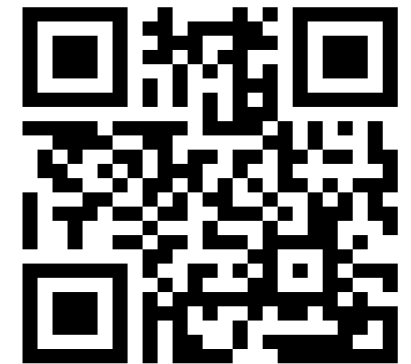


Daniel Nägele – `naegele@belwue.de` – `@debugloop` (on IRC & social)



# Fragen? Vielen Dank für Ihre Aufmerksamkeit!

---



unsere  
Webseite