Ambient Intelligence for Immersive Networked Systems Lab
Institute of Information Security
National Tsing Hua University

CUCKOOGUARD: A Memory-Efficient SYN Flood Defense Architecture for SmartNICs

Presentation for Master's Thesis Defense by

Tristan Döring

Wednesday 11th June, 2025

Motivation – DDoS Attacks

NTHU

Common Cyberthreat: **DDoS** (**D**istributed **D**enial **o**f **S**ervice) attacks

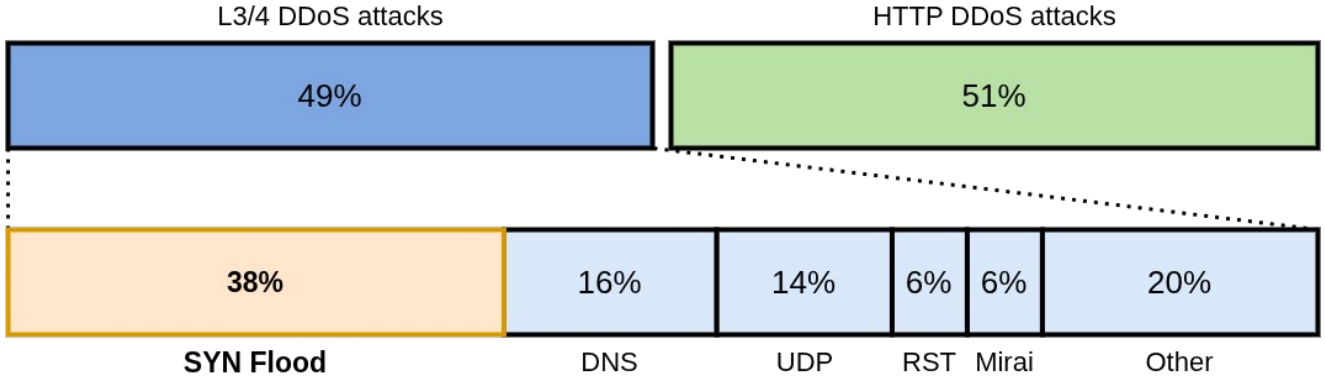
- Taiwan ranks #3 as attack target worldwide
- 2022 Nancy Pelocy visits Taiwan -> Ministry of Foreign Affairs website received over 8.5 million access requests within one minute

  ➥ Denial of Service: server overloaded - website goes offline

- DDoS Attacks
  - Target governments
  - Target companies
  - Are expensive (e.g. 300,000 USD/ hour)

Motivation – SYN Flood Attacks

SYN Floods are the most common network layer attack in 2024



(based on statistics collected by Cloudflare, a major network security company)

Goal

NTHU

Develop an effective SYN Flood Defense

What does “effective” mean?

fast successful protection *cheap*

Agenda

NTHU

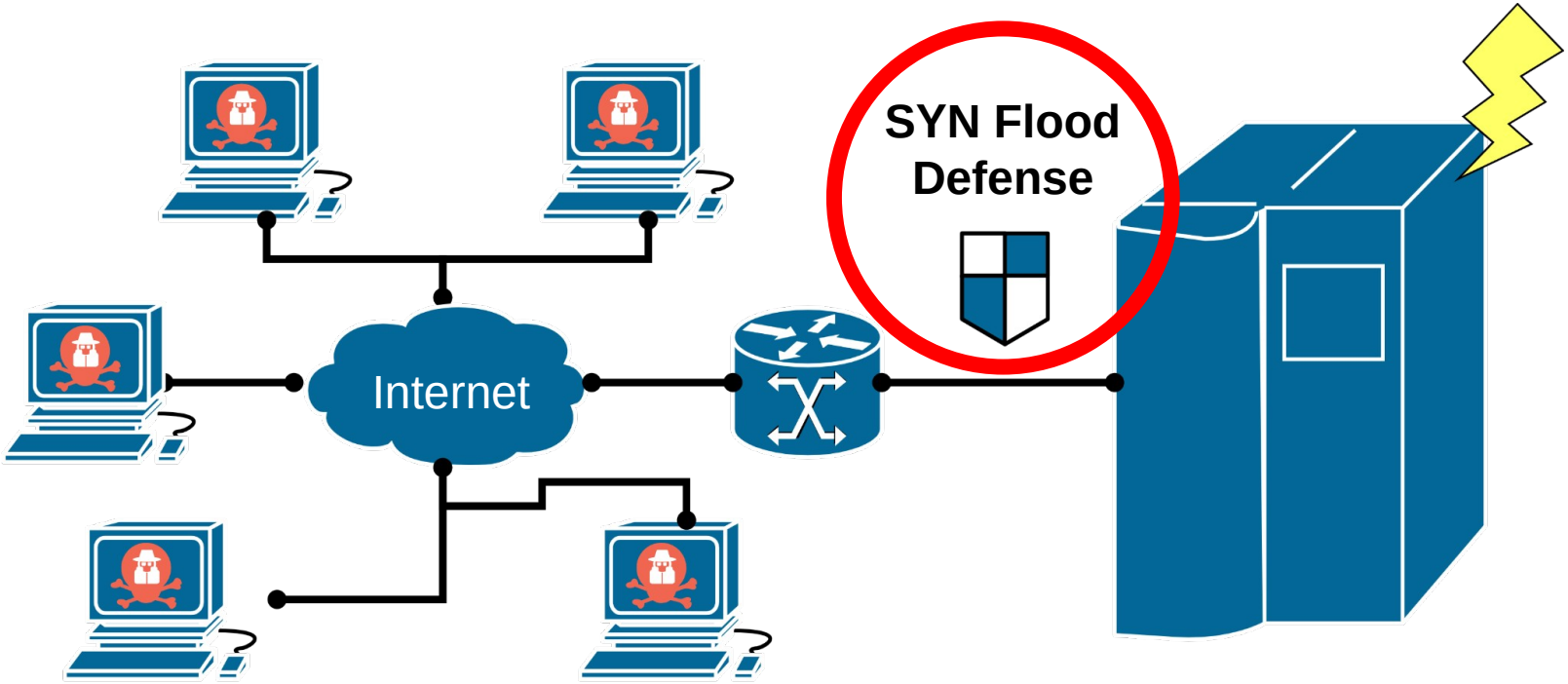
1. High-Performance Network Functions deployed in modern datacenters
2. General SYN Flood Defense Architecture Requirements
3. How to protect against SYN Floods
    1. Technical Background
    2. Related Work
4. The CUCKOOGUARD Architecture
    1. Overview
    2. Cuckoo Filters for flow filtering
5. CUCKOOGUARD Experimental Evalutation
6. Conclusion
7. Future Work

Main Contributions

NTHU

- **Improved CPU Offload**: Decreasing server-side CPU overhead by 79% and more
- A Novel **Memory-Efficient** and **Precise TCP-Connection Tracking** Scheme using Cuckoo filters

Background – Network View



Background – Hardware Accelerated Network Functions

Programmable Hardware

SmartNICs

Programmable Switches

Programming Paradigms (discussed later)

P4

eBPF

DPDK
DATA PLANE DEVELOPMENT KIT

Background – FPGA-based SmartNICs

E.g. XILINX Alveo U25 SmartNIC

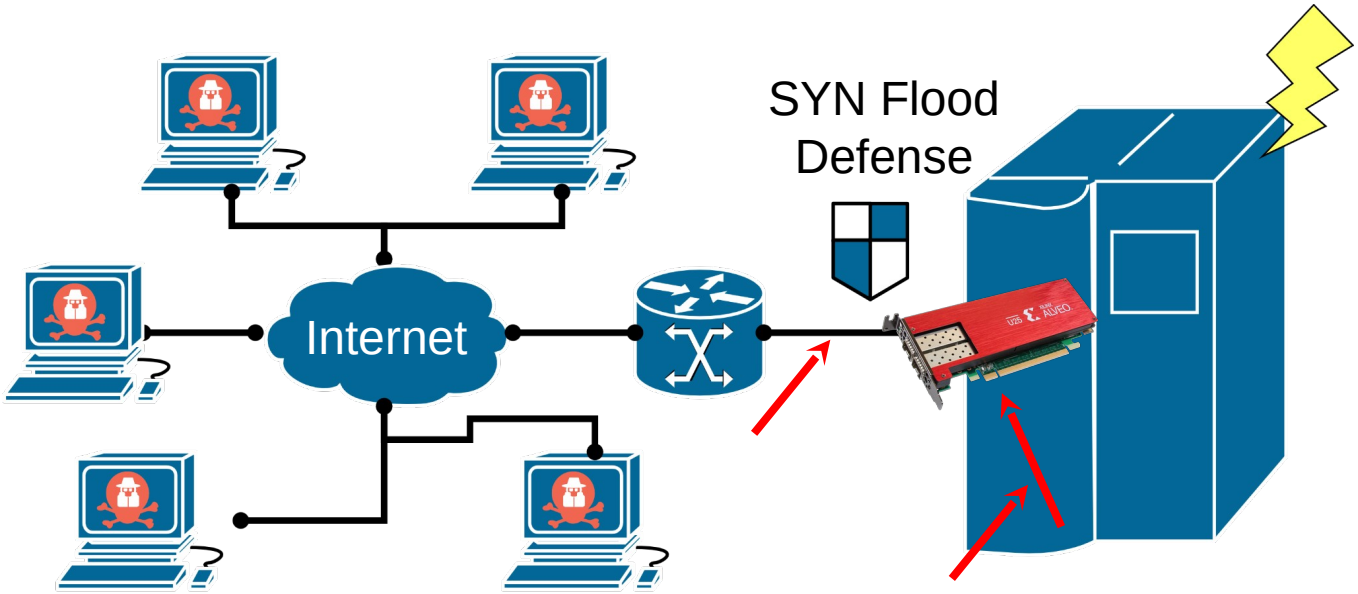
Advantages (compared to software):

- Line-rate performance (e.g. 25 Gbit/s)
- Low latency
- Full programmability
- Often include hardware accelerations

Limitations:

- Limited on-chip memory (e.g. ~8.8 MB)
- On-board RAM is too slow for line-rate per-packet access
- Multiple network functions must share resources

CUCKOOGUARD Architecture Requirements



Requirements

R1 – MEM

R2 – LAT

R3 – THR

R4 – TRP

R5 – OFF

R1 — R5 (MEM)

CUCKOOGUARD Architecture Requirements

NTHU

R1 — Low Memory Consumption (MEM)

R2 — Low Latency (LAT)

R3 — High Throughput (THR)

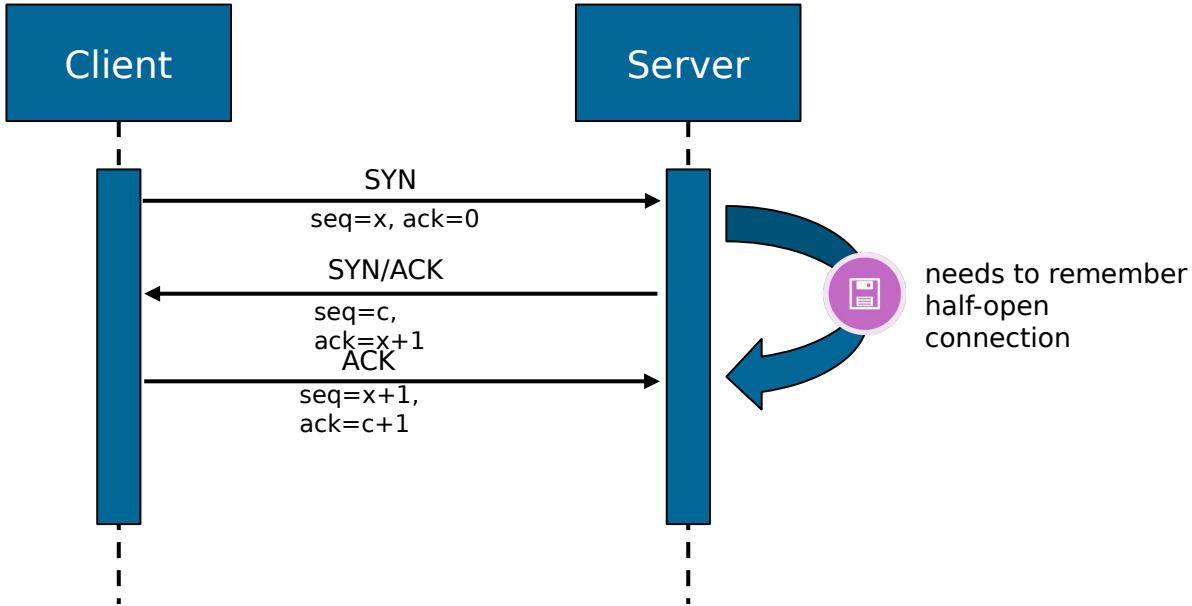
R4 — Transparency (TRP)

R5 — Server Offloading (OFF)

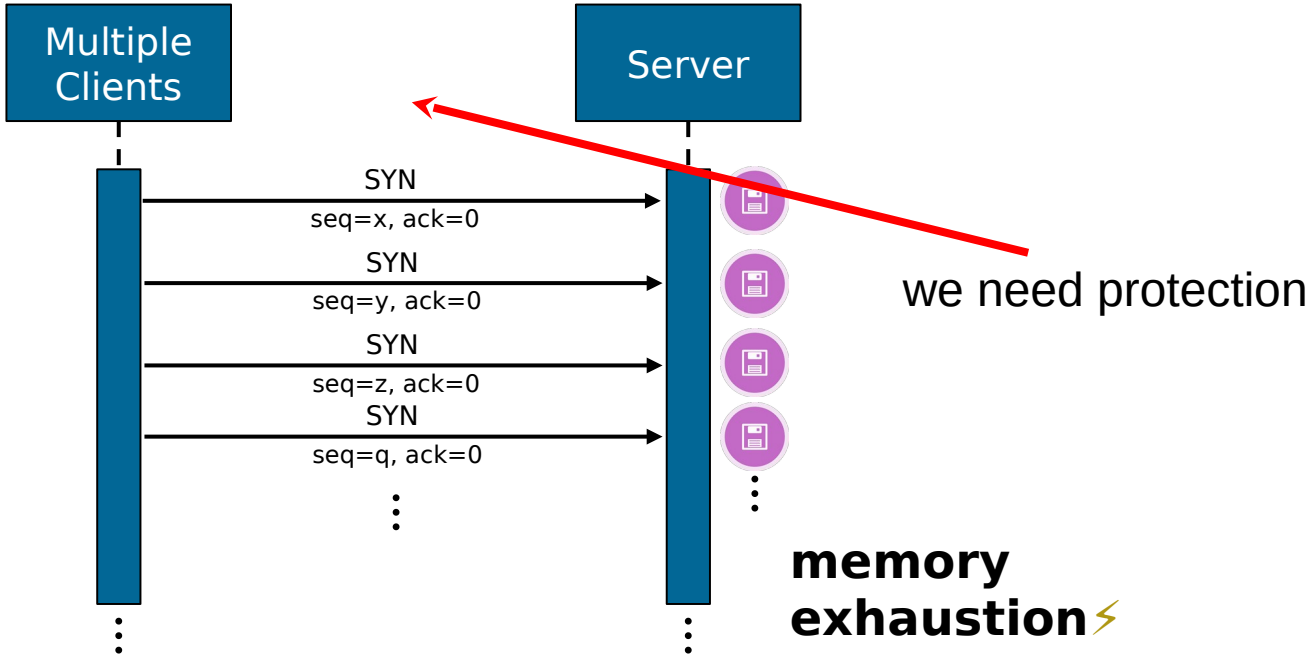
R6 — High Churn Tolerance (CHN)

R7 — Connection Stability and Long-Term Robustness (STA)

TCP Handshake Basics



SYN Flood Attack Surface



SYN Flood Protection in Programmable Data Planes

NTHU

3 Architectural Approaches

Machine Learning

- Computationally expensive
- Slow reaction time
- Classification errors

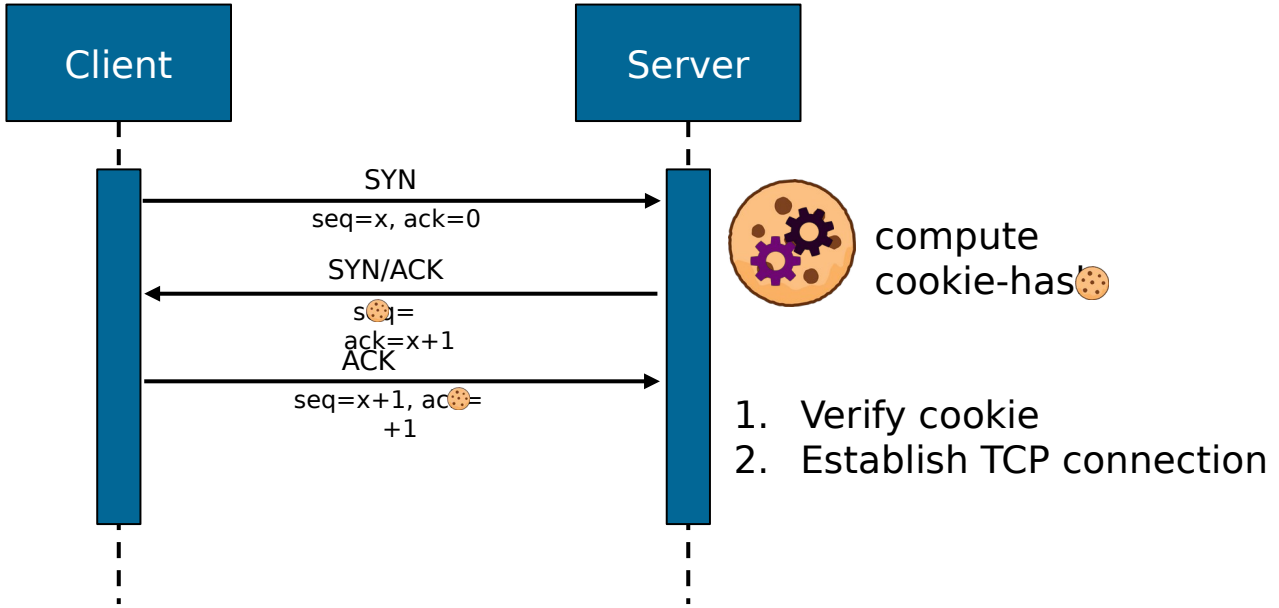
Rule-based

- Rigid assumptions
- High-memory overheads
- Arbitrary thresholds
- Limited scalability

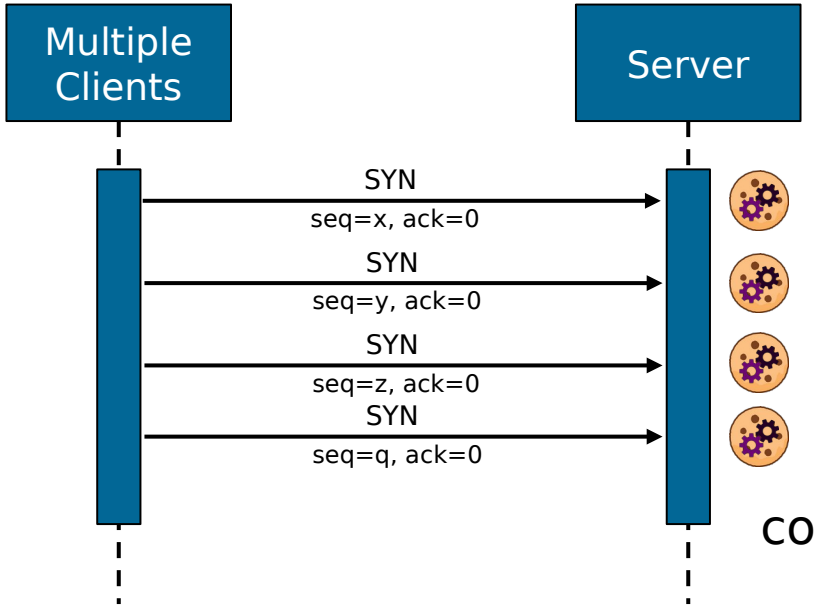
SYN-Cookies

- Precise
- Scalable
- Preserve memory
- Protocol transparent

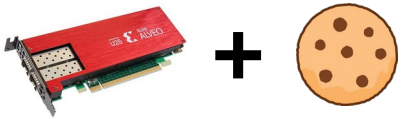
SYN-Cookie Basics



SYN-Cookies Attack Scenario



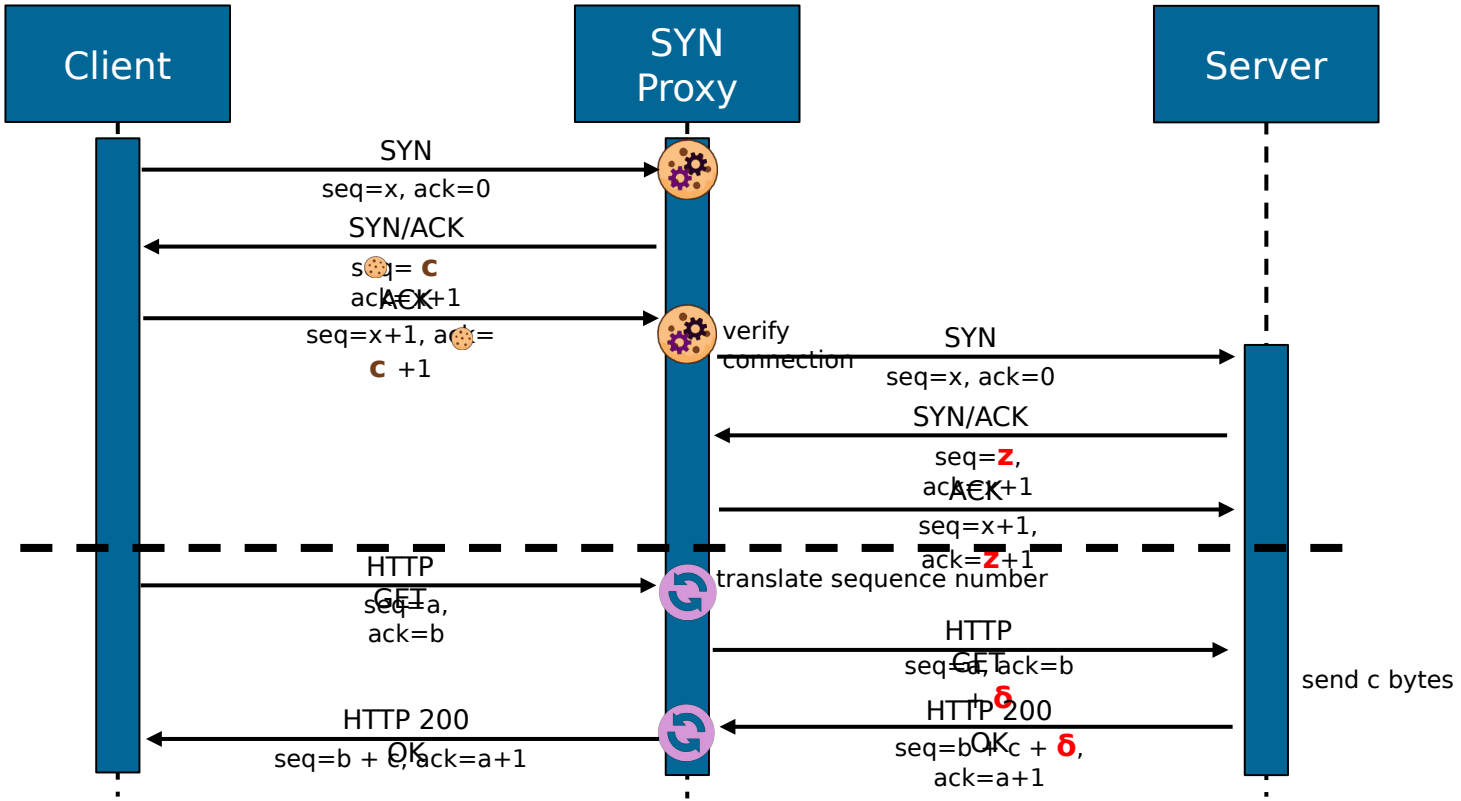
Idea:



Offload SYN-Cookies from the server

computational exhaustion⚡

SYN(-Cookie) Proxy



Problem Analysis

SYN-Cookie Proxy is very promising!

Let's put it into the programmable data plane (SmartNIC)

Deploying SYN-Cookie Proxy on SmartNICs is too memory expensive

E.g. ~9.9 MB needed vs. 8.8 MB available (579,600 flows)

- Other network functions need memory resources
- SYN Flood Defense often only secondary
  - Too expensive for server operators for one function only

Good:

1. SmartNIC is
2. Only verified

Challenges:

3. Sequence number translation for every regular TCP packet
4. SmartNIC needs to keep connection state for every connection
   - Connection identifier + sequence number difference δ

CUCKOOGUARD Outperforms State-of-the-art Solutions

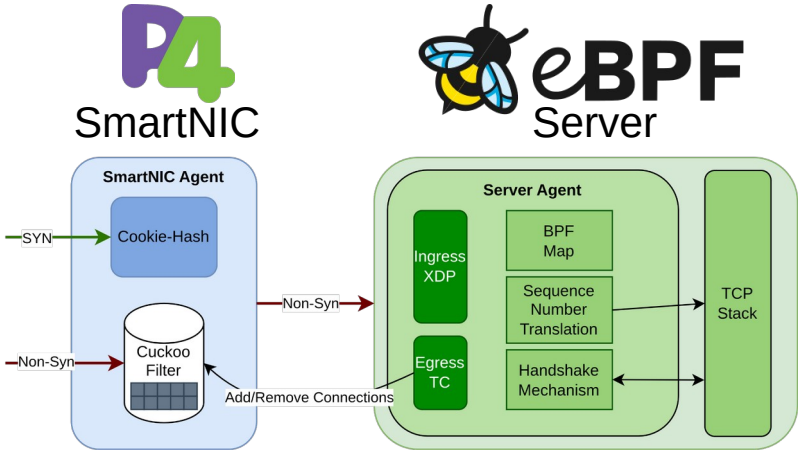
NTHU

Approach	R1 – MEM	R2 – LAT	R3 – THR	R4 – TRP	R5 – OFF
Machine Learning	✕	✓	✓	✕	◦
Rule-based	✕	✓	✓	✕	✓
SYN-Cookie Proxy[1]	✕	✓	✓	✓	✓
SMARTCOOKIE[2]	◦	✓	✓	✓	✕
CUCKOOGUARD	✓	✓	✓	✓	✓

✓: fully satisfies, ◦: partially satisfies, ✕: does not satisfy.

[1]Scholz et al., *SYN Flood Defense*, EuroP4 '20
[2]Yoo et al., *SmartCookie*, USENIX Sec. '24

CUCKOOGUARD Architecture

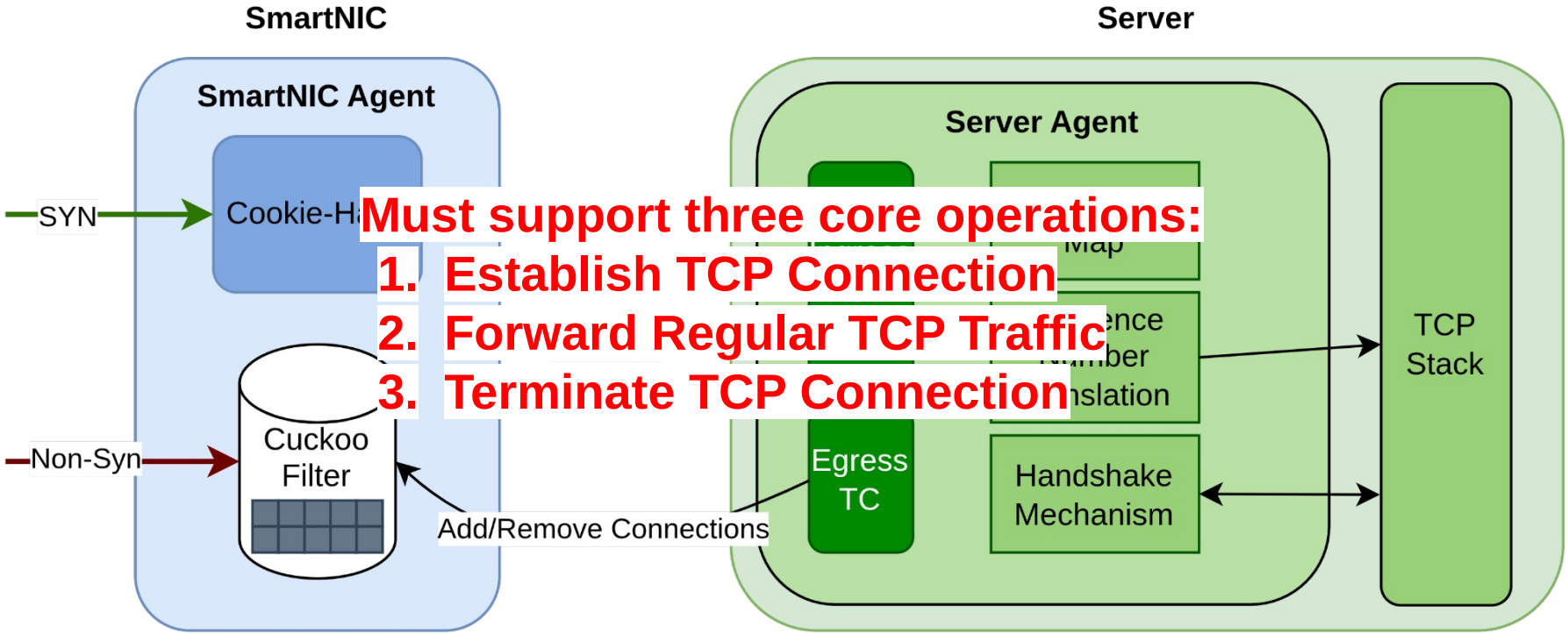


Main Ideas:

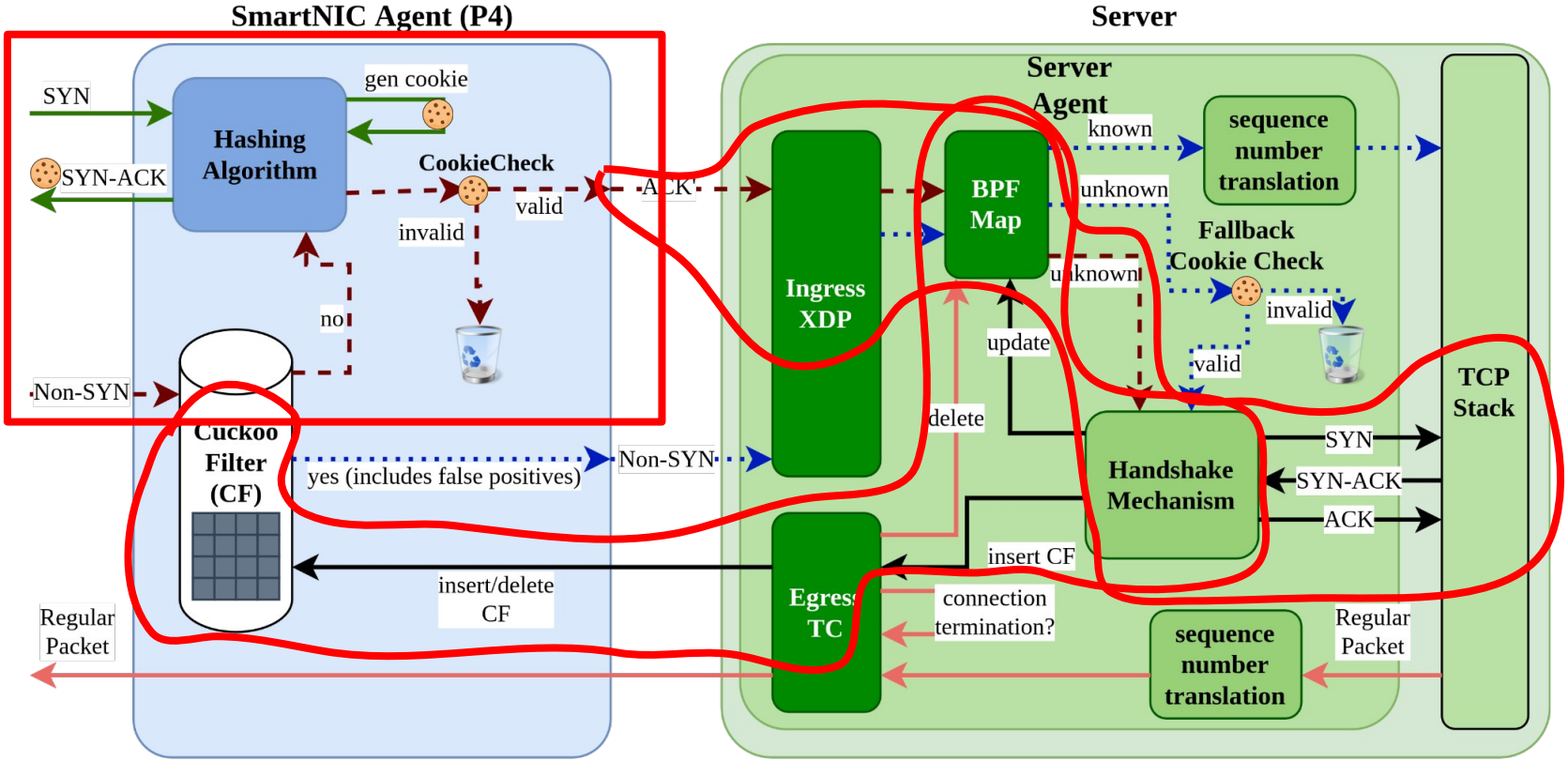
- Split-Proxy approach (originally from baseline: SMARTCOOKIE[2])
  1. **Offload SYN-Cookie computation to SmartNIC**
  2. **Offload connection state to the servers kernel**
- Memory-efficient line-rate packet filtering in the SmartNIC
- **Major Innovation**: Deploy a Cuckoo filter for connection tracking

[2]Yoo et al., *SmartCookie*, USENIX Sec. '24

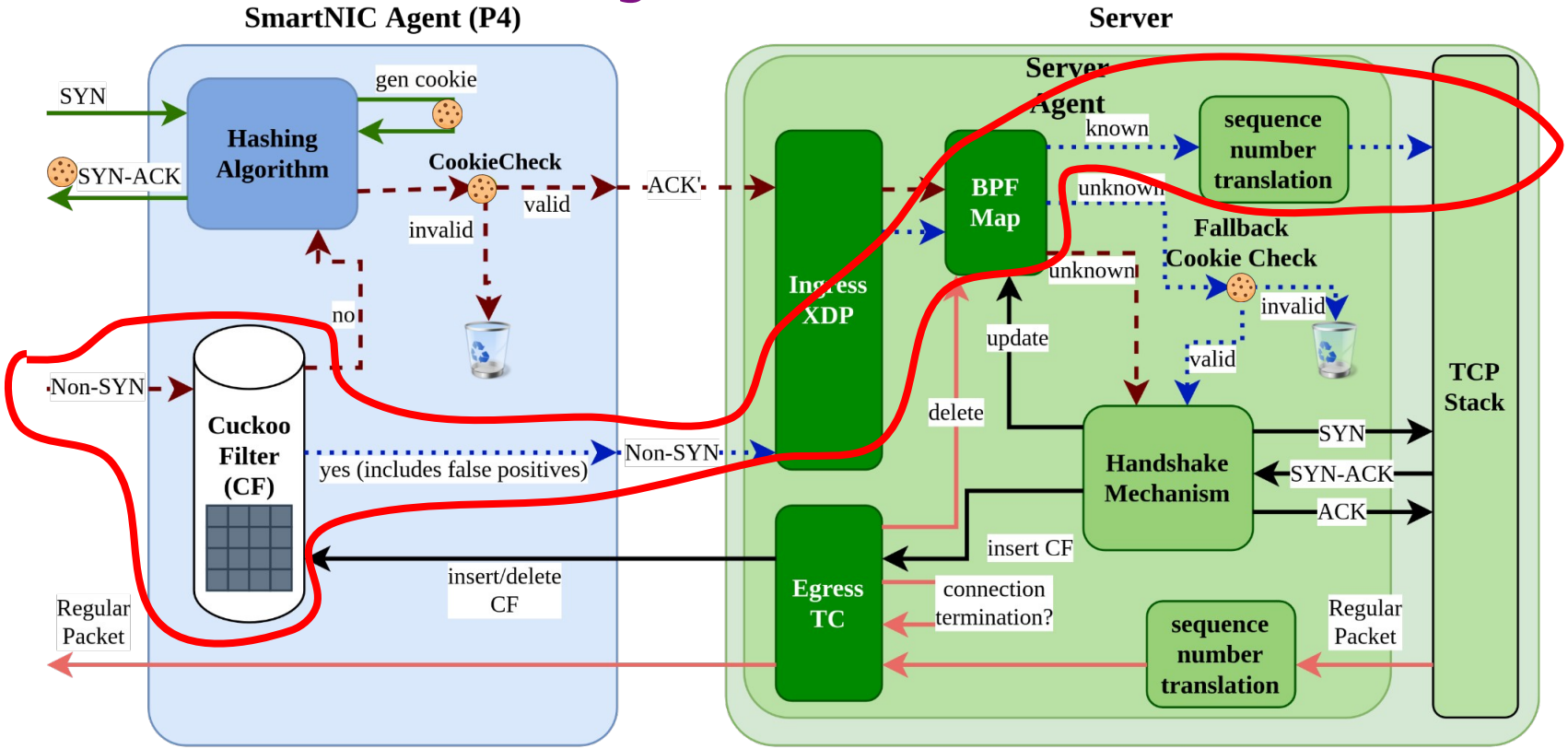
CUCKOOGUARD Architecture



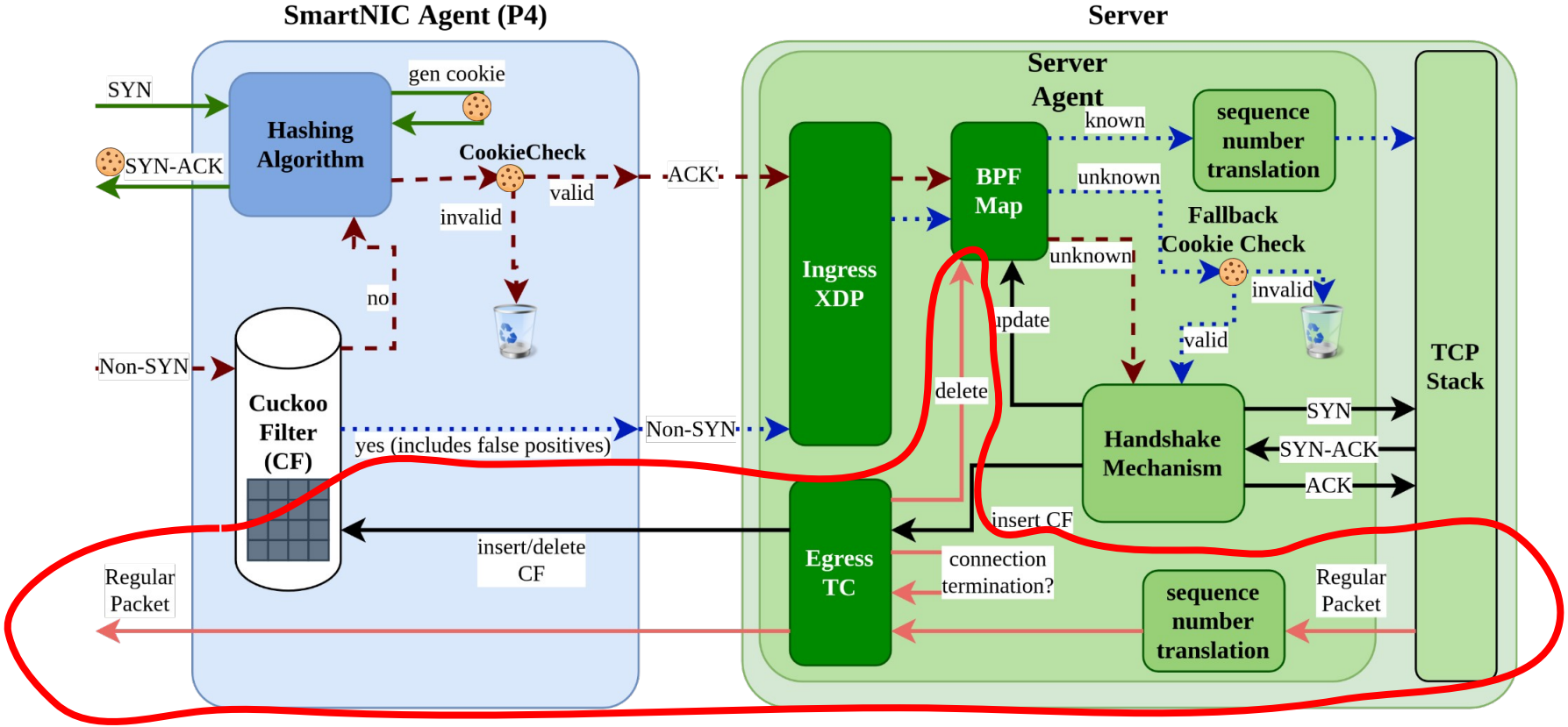
CUCKOOGUARD – TCP Client Connection



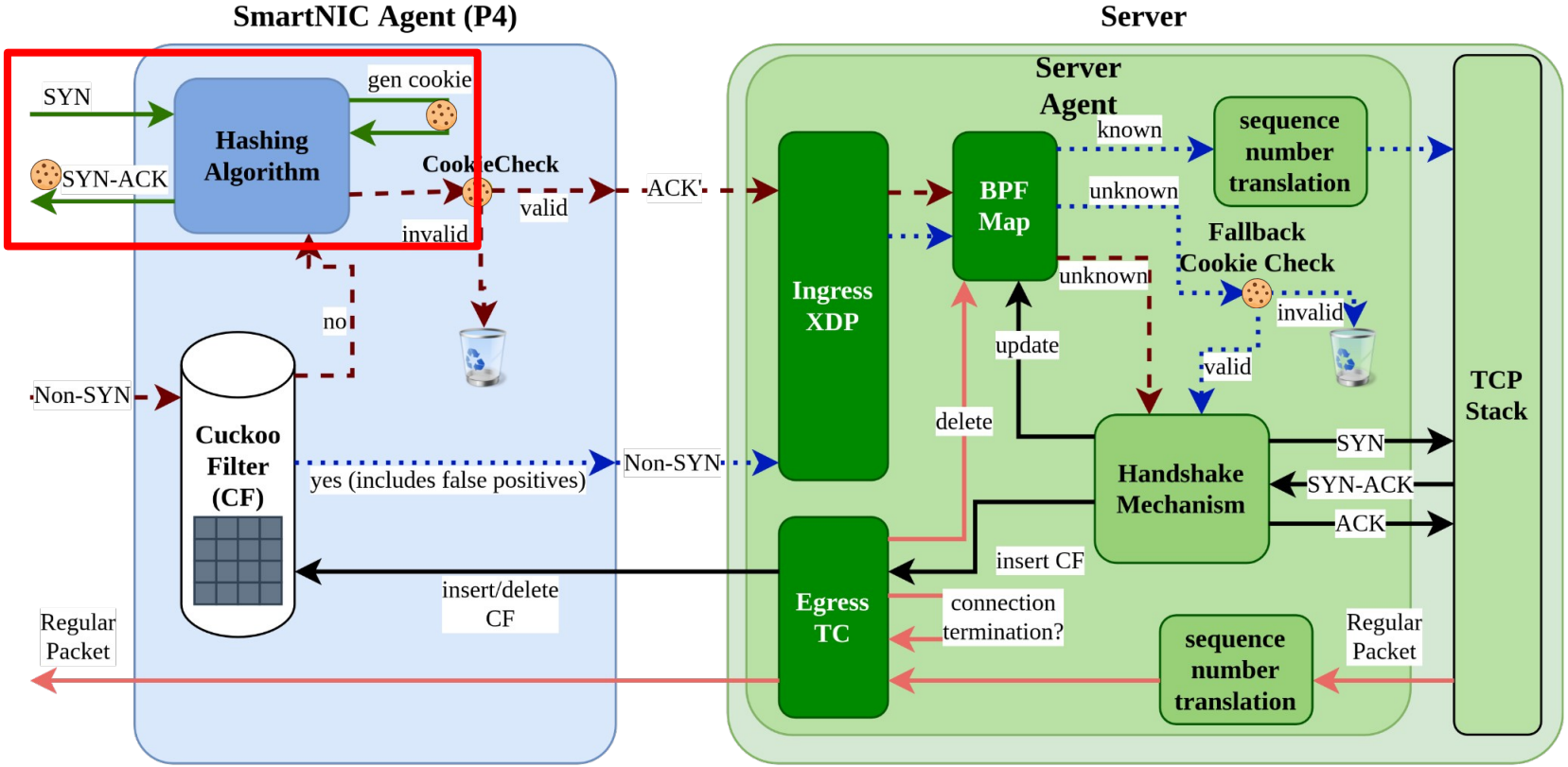
CUCKOOGUARD– Regular TCP Traffic



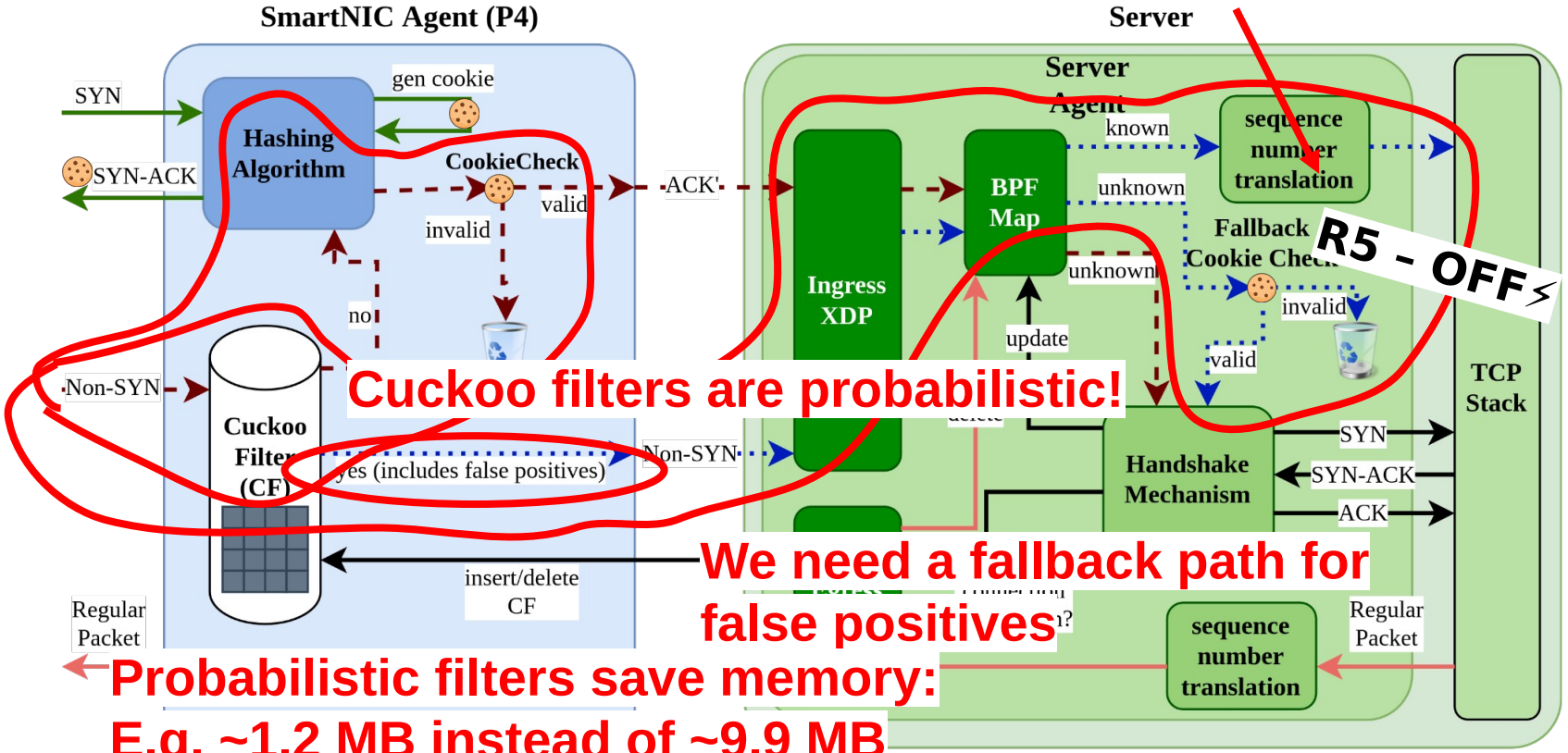
CUCKOOGUARD – TCP Connection Termination



CUCKOOGUARD – SYN Flood Attack Resistance



CUCKOOGUARD – ACK Flood Attack Resistance



CUCKOOGUARD – An Argument for Cuckoo Filters

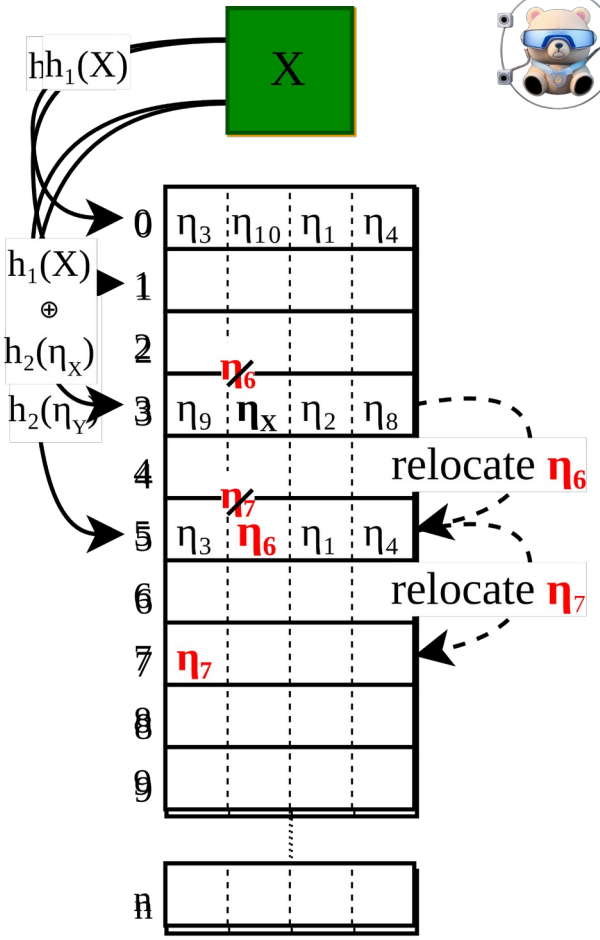
NTHU

Filter Type	Insertion	Query	Deletion
Bloom Filter	O(k)	O(k)	**Not supported**
Cuckoo Filter	**O(1) amortized**	O(1)	O(1)

k = number of hash functions

- Cuckoo filters support explicit entry removal
  - We can utilize this to remove terminated TCP connections precisely
  - **Memory-efficient**
- Bloom filters need to employ workarounds
  - Time-decaying (keep two filters, alternate between deleting one of them)
  - **Memory-hungry**

Cuckoo Filter Data Structure



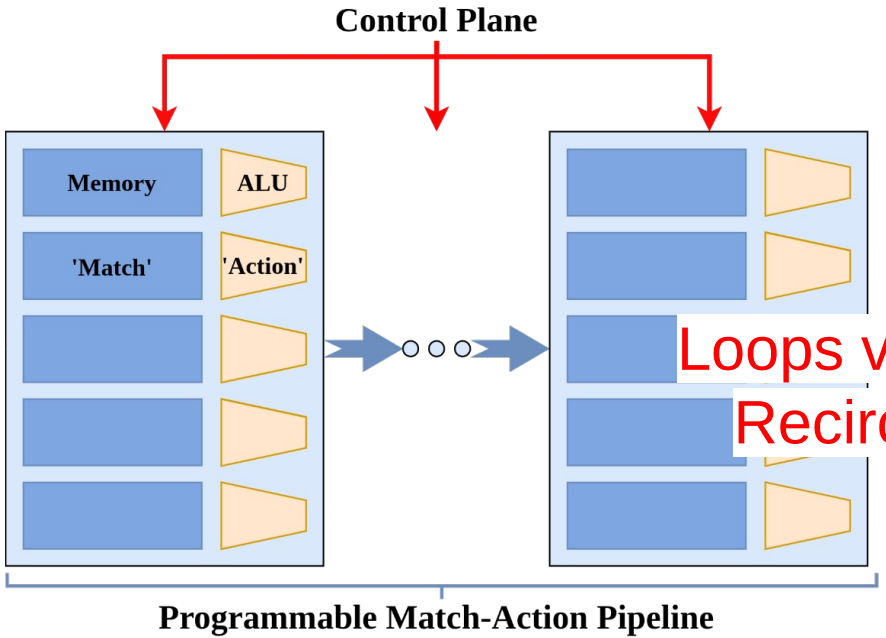
General Idea[3]:

1. Connection tuple ***X*** hashed to fingerprint ***η*** of length ***f***
2. ***i_1, i_2*** indices determined by Cuckoo Hashing

- False Positives happened if connection tuples ***X*** and ***Z*** are hashed to the same fingerprint ***η***
- Increasing ***f*** decreases false positives

[3]B. Fan et al. *Cuckoo Filter*. CoNEXT '14.

Match-Action Pipeline Programming



Advantages:

- High-performance on hardware
- Predictable latency

Loops via Packet Recirculation

- No loops (for, while, ...)
- ...

Evaluation – Experimental Setup

NTHU

Linux-based Implementation Setup:

- SmartNIC Agent emulated via BMv2 software running P4 program
- Server Agent deployed in kernel as eBPF program

Goals:

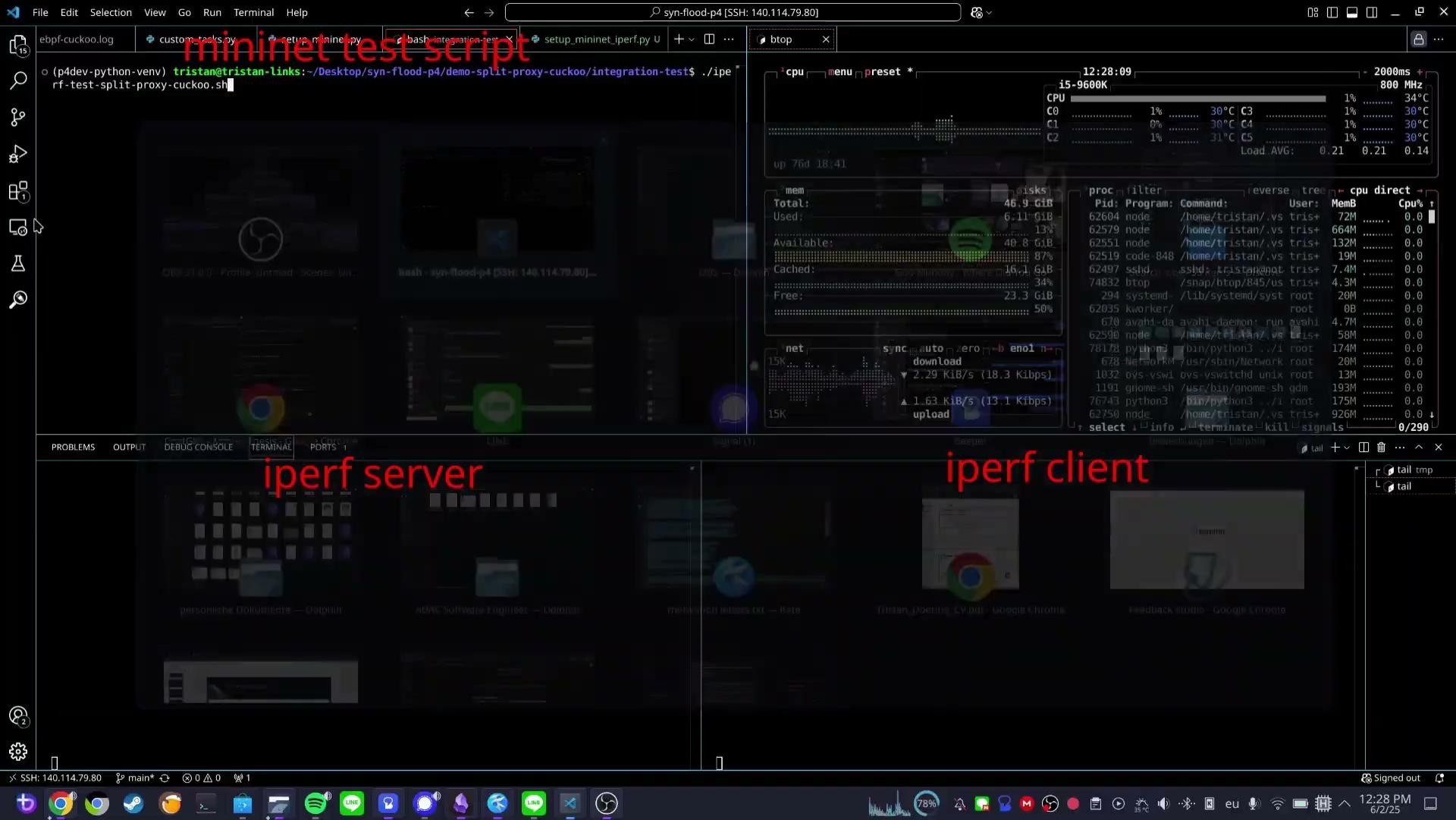
1. Validate Architecture
2. Evaluate Cuckoo filter insertion overhead
3. Bloom filter vs. Cuckoo filter
4. CUCKOOGUARD vs. related work (SMARTCOOKIE)

Validate Architecture: System Demonstation using iPerf

NTHU

Setup:

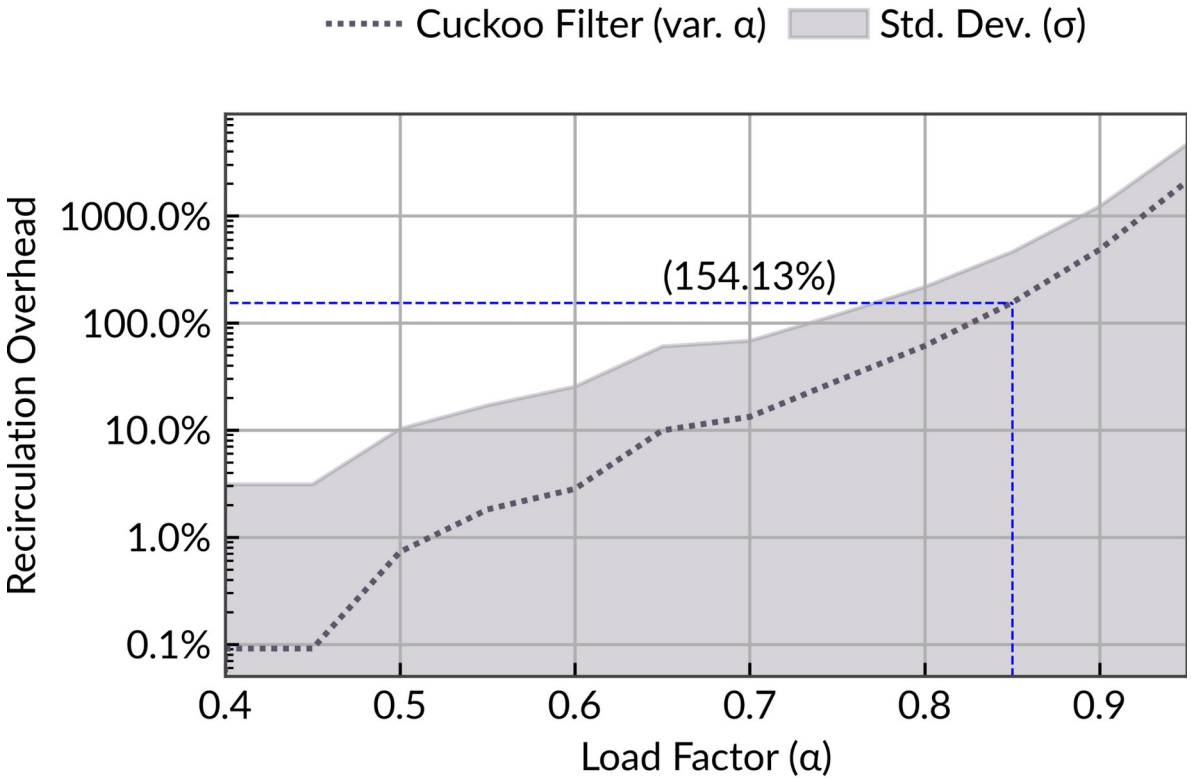
- Virtualized network using mininet
- iPerf throughput test: 10 parallel TCP connections


mininet test script
(p4dev-python-venv) tristan@tristan-links:~/Desktop/syn-flood-p4/demo-split-proxy-cuckoo/integration-test$ ./ipe
rf-test-split-proxy-cuckoo.sh
btop
cpu menu preset * 12:28:09 2000ms
i5-9600K 800 MHz
CPU 1% 34°C
C0 1% 30°C C3 1% 30°C
C1 0% 30°C C4 1% 30°C
C2 1% 31°C C5 1% 30°C
Load AVG: 0.21 0.21 0.14
up 76d 18:41
mem disks
Total: 46.9 GiB
Used: 6.11 GiB 13%
Available: 40.8 GiB 87%
Cached: 16.1 GiB 34%
Free: 23.3 GiB 50%
net sync auto zero enol
download
▼ 2.29 KiB/s (18.3 Kibps)
▲ 1.63 KiB/s (13.1 Kibps)
upload
proc filter reverse tree cpu direct
Pid: Program: Command: User: MemB Cpu%
62604 node /home/tristan/.vs tris+ 72M 0.0
62579 node /home/tristan/.vs tris+ 664M 0.0
62551 node /home/tristan/.vs tris+ 132M 0.0
62519 code-848 /home/tristan/.vs tris+ 19M 0.0
62497 sshd: sshd: tristan@pt tris+ 7.4M 0.0
74832 btop /snap/btop/845/us tris+ 4.3M 0.0
294 systemd- /lib/systemd/syst root 20M 0.0
62035 kworker/ root 0B 0.0
670 avahi-da avahi-daemon: run avahi 4.7M 0.0
62590 node /home/tristan/.vs tris+ 58M 0.0
78178 python3 /bin/python3 ../i root 174M 0.0
678 NetworkM /usr/sbin/Network root 20M 0.0
1032 ovs-vswi ovs-vswitchd unix root 13M 0.0
1191 gnome-sh /usr/bin/gnome-sh gdm 193M 0.0
76743 python3 /bin/python3 ../i root 175M 0.0
62750 node /home/tristan/.vs tris+ 926M 0.0
select info terminate kill signals 0/290
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
tail
tail tmp
tail
iperf server
iperf client
SSH: 140.114.79.80 main* 0 0 1
Signed out
12:28 PM
6/2/25

Cuckoo Filter Insertion Overhead Manageable

Experiment Setup

- Fix memory budget
- Vary load factor **α**
- Test inserting new connections

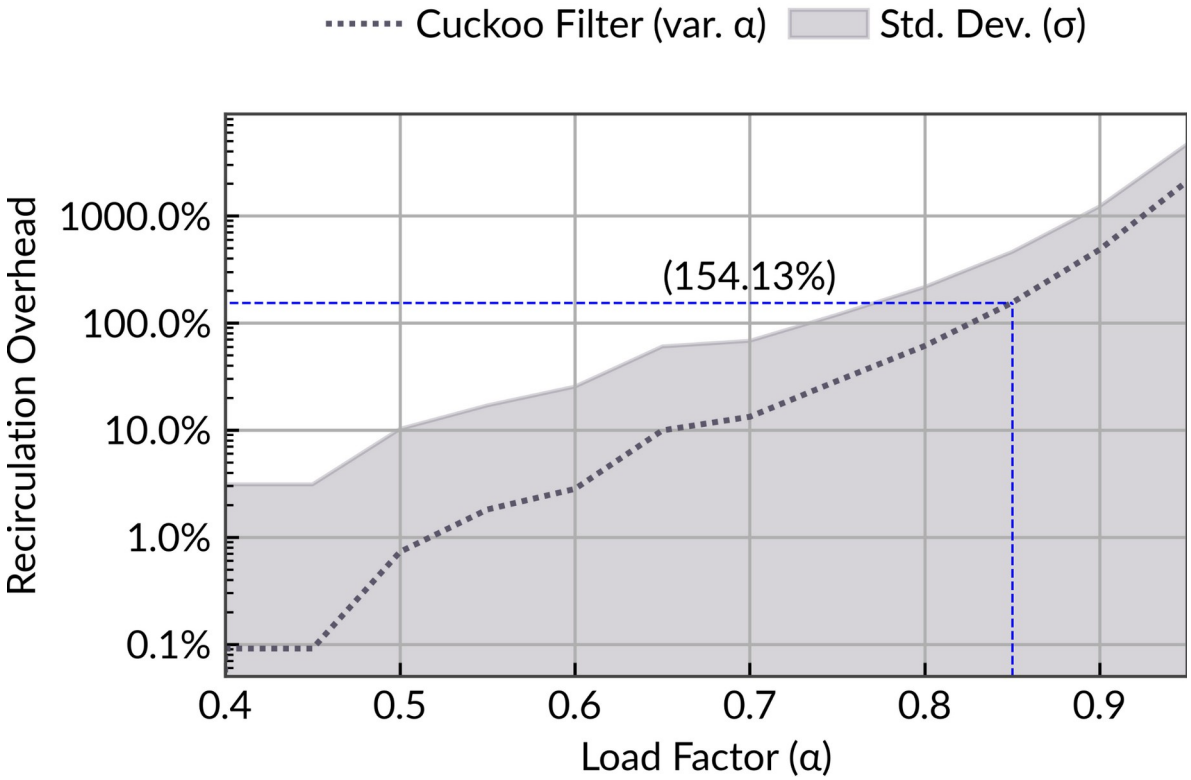


Cuckoo Filter Insertion Overhead Manageable

Is Cuckoo filter insertion too slow?

No, because

- **Only once per connection**
- neglictible at ***α = 0.85***



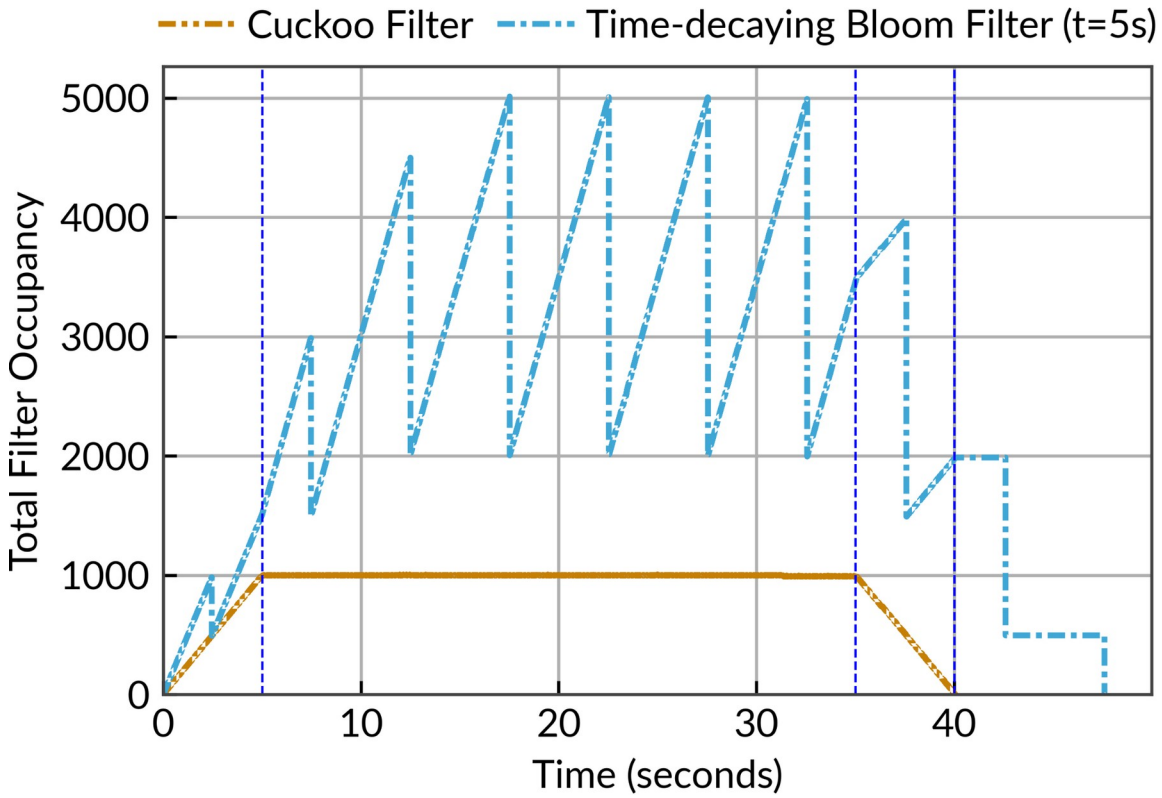
Cuckoo Filter enables Precise Connection Tracking

Experiment Setup

- 200 new TCP connections per second
- Each lasts for 5s

After 35s:

- No new connections

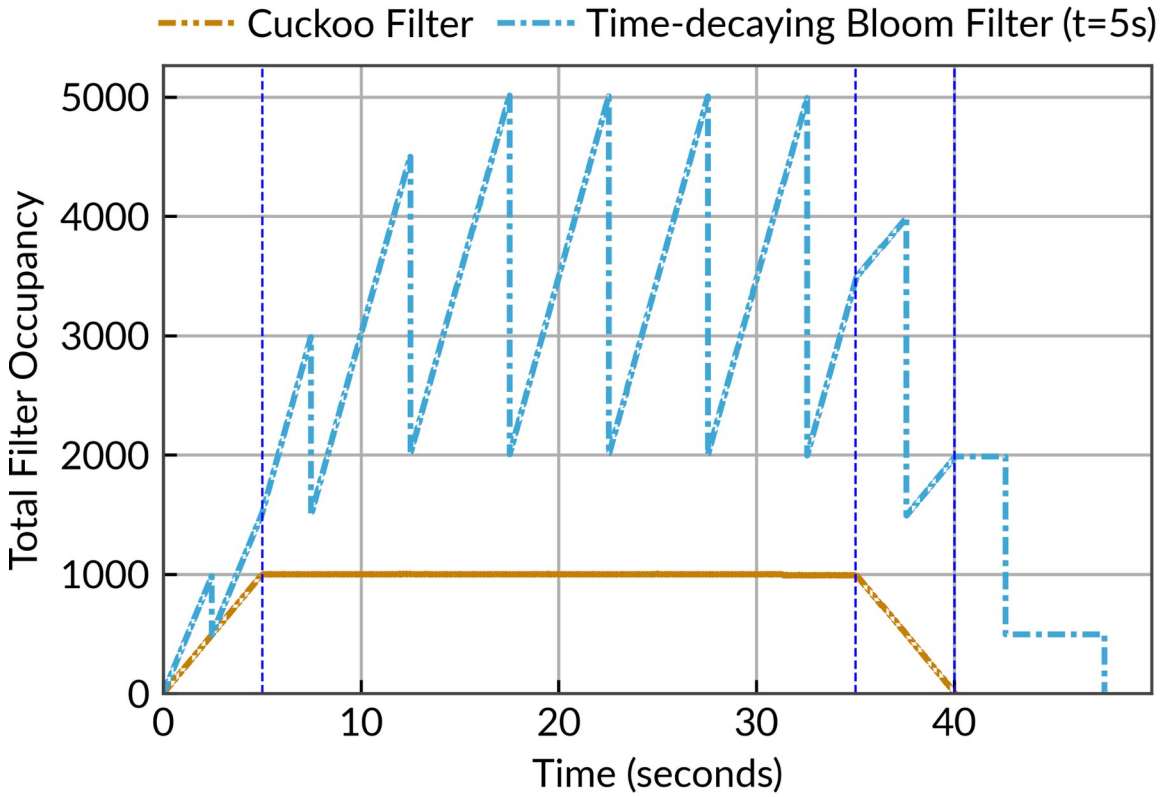


Cuckoo Filter enables Precise Connection Tracking

NTHU

Result

- Bloom filters display **6x overhead**
- Cuckoo filters avoid tracking overhead
- Improved memory-efficiency



Filter Precision: Bloom Filter vs. Cuckoo Filter

NTHU

Experiment Setup

- Vary available memory per connection
- Compare false positive rate at max. capacity (worst case)
- Low false positive rate important for **R5 – OFF**

Does not support connection removal

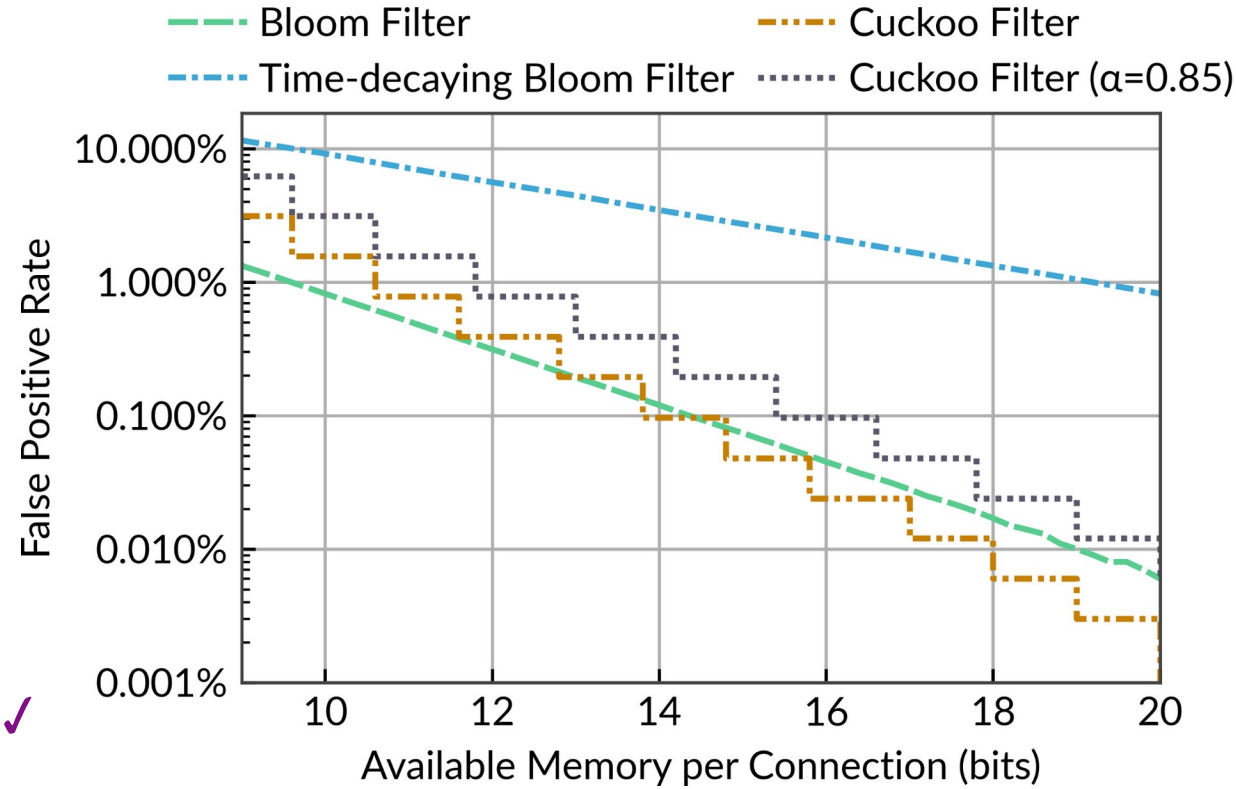
Optimized for low insertion overhead

Filter Precision: Bloom Filter vs. Cuckoo Filter

Result

- Cuckoo filter ~100x more precise (at 20 bits)
- Server offloading improved by ~100x

R5 – OFF ✓



Evaluation – CUCKOOGUARD vs SMARTCOOKIE

Experiment Setup

- Both track 579,6
  concurrent TCP
  connections
- 786 KB of mem
- ACK flood attac
  consisting of 10
  packets

Results

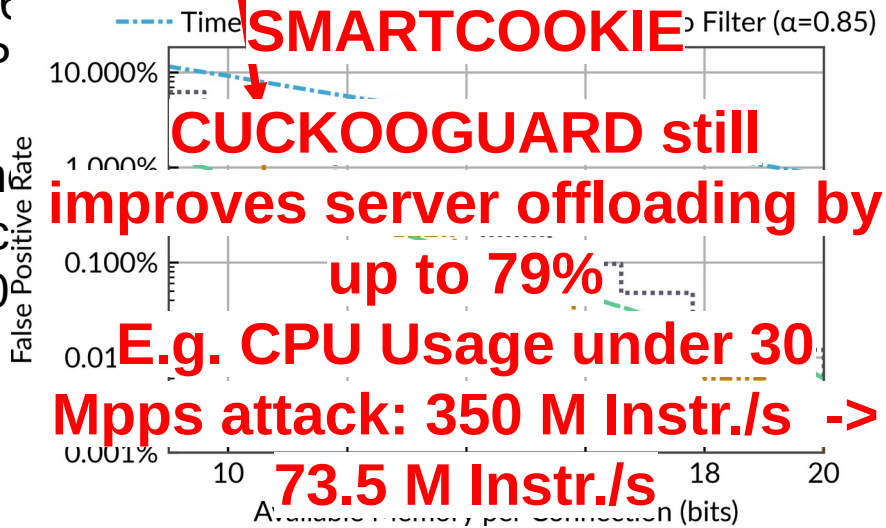
This scenario highly favors the Bloom filters used in SMARTCOOKIE

CUCKOOGUARD still improves server offloading by up to 79%

E.g. CPU Usage under 30 Mpps attack: 350 M Instr./s -> 73.5 M Instr./s

56% false positive

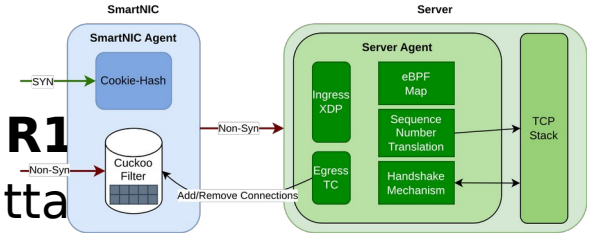
5% false positive



Conclusion

Goal: Develop an effective SYN Flood Defense

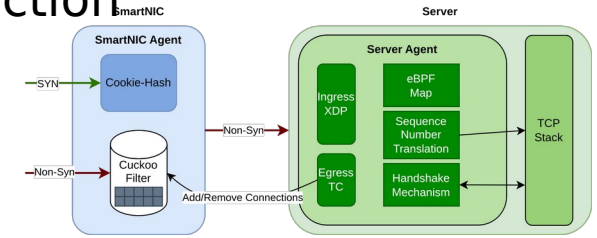
Achievements:



- Architecture that achieves all requirements (**R1**
- Protect any TCP server from SYN/ACK flood atta
- Utilize Programmable Hardware -> high performance
- Improved protection compared to state of the art by at least 79%
- Foundational work for the use of Cuckoo filters in programmable data planes
- Implementations can be used for real deployments

Future Work

- Evaluate Latency/ Performance using real hardware
- Integration into multi-functional network function architectures
- Further optimization of Cuckoo filters by implementing semi-sorting (algorithmic complexity vs. memory-efficiency)
- Analyze impact of TCP options



NTHU

Thank you for listening

Thanks for the support with the

Paper Submission to IEEE Globecom 2025
CISS

Bibliography - Images

NTHU

- Intel Tofino 2 image: GazetteByte – "Intel combines optics to its Tofino 2 switch chip" https://www.gazettabyte.com/home/2020/3/19/intel-combines-optics-to-its-tofino-2-switch-chip.html
- P4 logo: P4.org – https://p4.org/wp-content/uploads/2024/06/p4-logo.svg
- eBPF logo: GitHub – https://github.com/goranvasic/svg-logos/blob/main/ebpf-dark-logo.svg
- DPDK logo: DPDK Project – https://fast.dpdk.org/download/logos/DPDK_logo_horizontal_tag.svg
- Xilinx U25N SmartNIC image: Xilinx (archived) – https://web.archive.org/web/20200511172852im_/https://www.xilinx.com/content/dam/xilinx/imgs/kits/alveo-u25-hero-of-card.jpg

Bibliography

NTHU

[1] D. Scholz, S. Gallenmüller, H. Stubbe, and G. Carle.

SYN Flood Defense in Programmable Data Planes.

Proc. of the 3rd P4 Workshop in Europe (EuroP4 '20), pp. 13–20, 2020.

[2] S. Yoo, X. Chen, and J. Rexford.

SmartCookie: Blocking Large-Scale SYN Floods with a Split-Proxy Defense on Programmable Data Planes.

Proc. of the 33rd USENIX Security Symposium (USENIX Security '24), pp. 217–234, Aug. 2024.

[3] B. Fan, D. G. Andersen, M. Kaminsky, and M. Mitzenmacher.

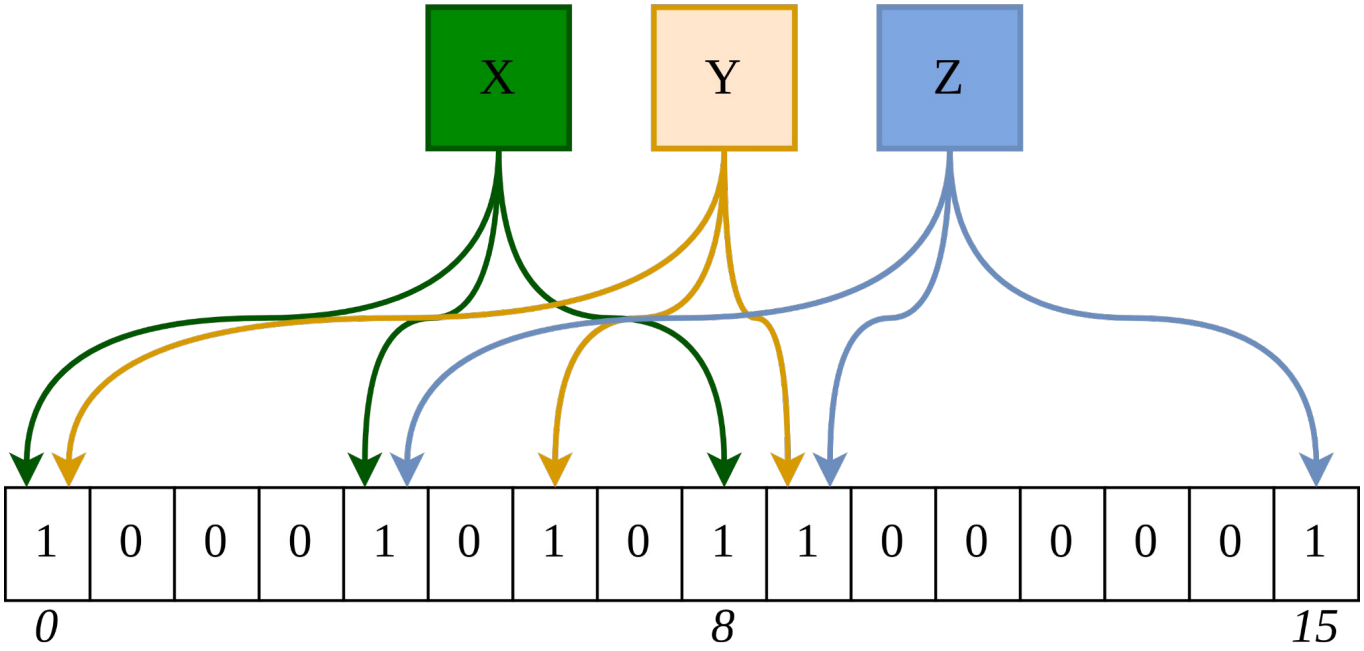
Cuckoo filter: Practically better than Bloom.

Proc. of the 10th ACM CoNEXT, pp. 75–88, 2014.

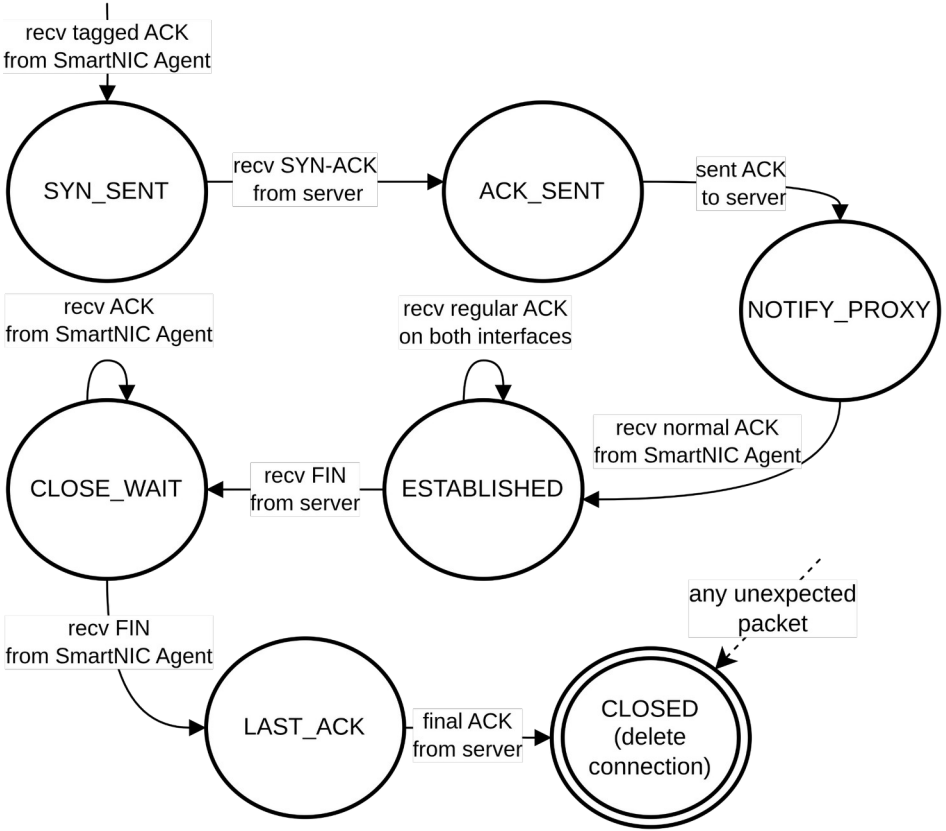
NTHU

Backup Slides

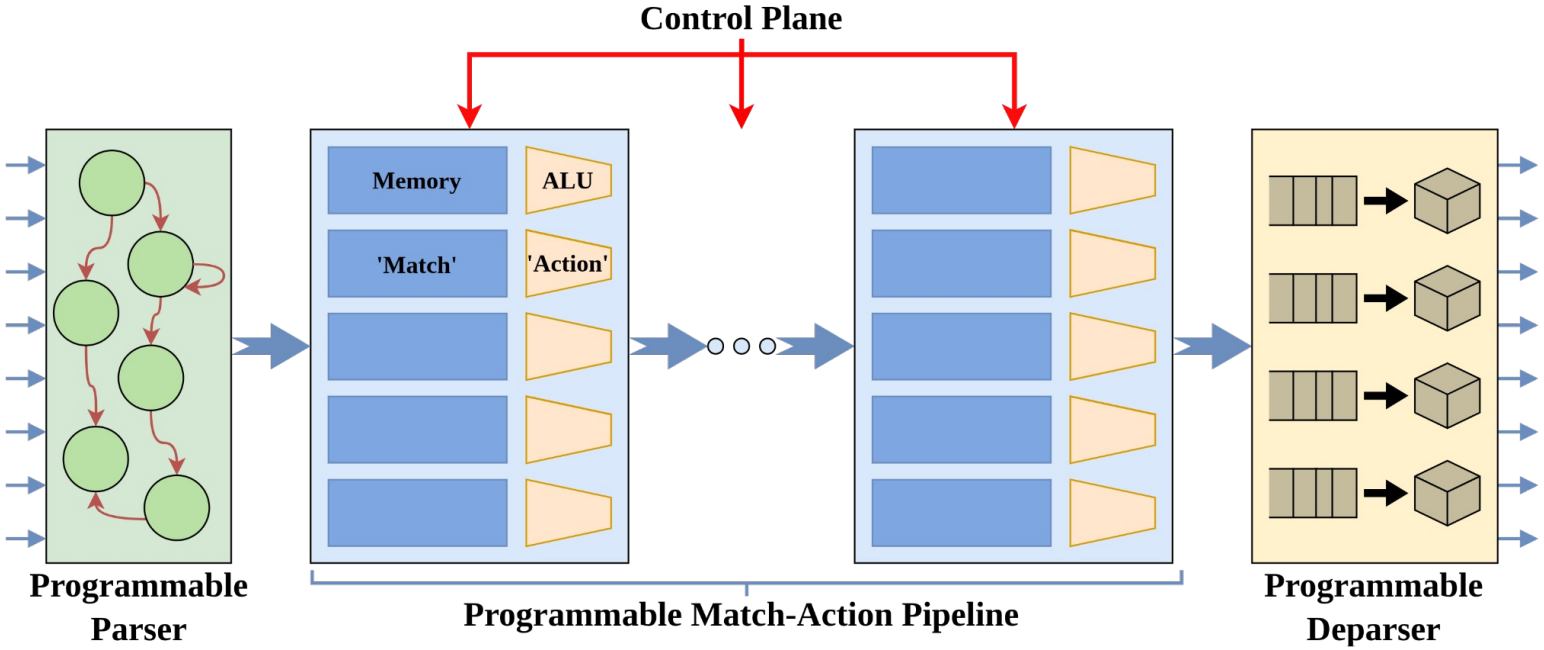
Background – Bloom Filter



Implementation - Server Agent: Connection State Machine



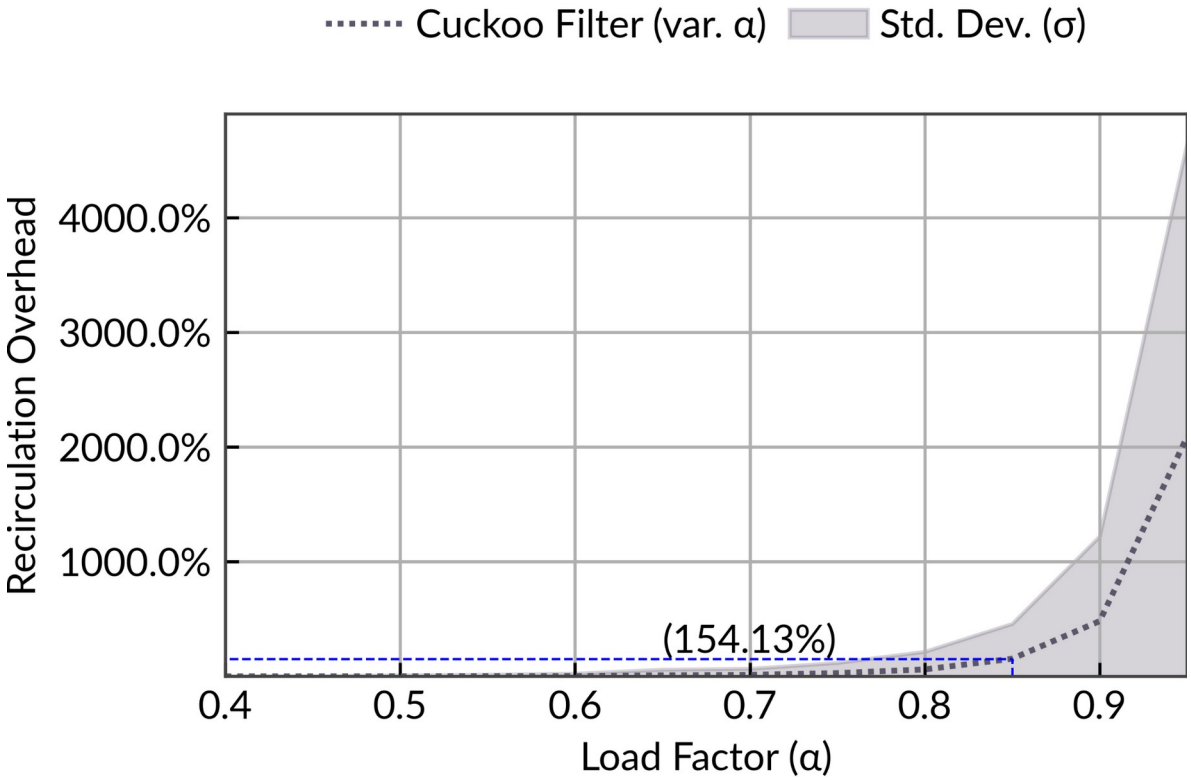
– Match Action Pipeline Programming



Cuckoo Filter Insertion Linear Scale

Experiment Setup

- Fix memory budget
- Vary load factor **α**
- Test inserting new connections



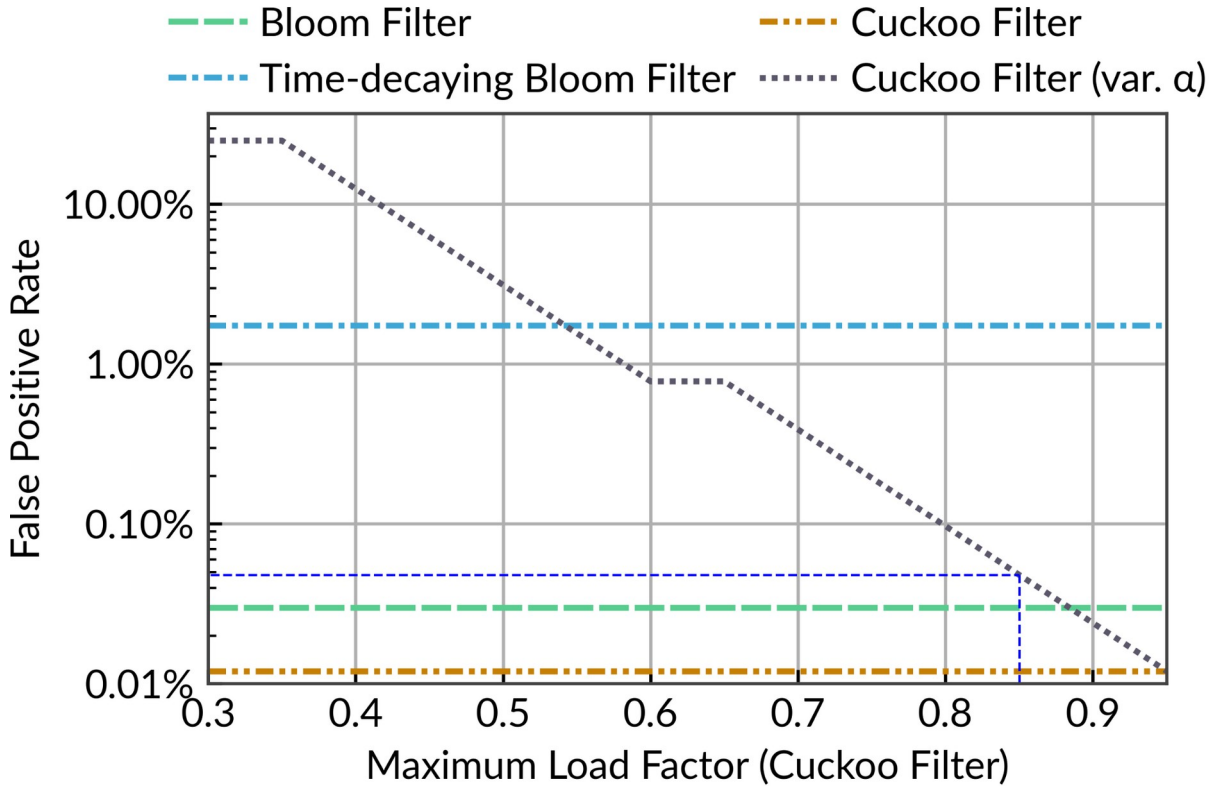
Evaluation – Bloom Filter vs. Cuckoo Filter

Setup

- fixed memory per connection (16.85 bits)

Result

- precision at α=0.85 only slightly decreased



Memory Usage Example Cuckoo Filter vs. Perfect Filter

NTHU

Scenario: 579,600 flows

Data Structure	Memory Usage	False Positive Rate
Connection Table	~9,853 KB ~ 9.9 MB	0%
Cuckoo Filter	~ 1,236 KB ~ 1.2 MB	0.01%