Line-Rate Cybersecurity: Modern DPI and Encrypted Traffic Fingerprinting at 100 Gbps

Luca Deri
deri@ntop.org

Alfredo Cardigliano
cardigliano@ntop.org

About Us

Luca Deri

- Founder of **ntop**
- Creator of ntopng, nProbe, **nDPI**, ...
- Ph.D. Computer Science, Professor at UniPi
- Regular speaker at networking & security conferences

Alfredo Cardigliano

- Principal Engineer at **ntop**
- Lead developer of **PF_RING**, n2disk, nScrub, ...
- Focus on kernel/user-space integration, line-rate packet processing
- DPDK contributor (SmartNIC drivers)

*github.com/**ntop***

Talk Outline

- Why DPI?
- Cryptographic fingerprinting
- Performance baseline
- Linux Netfilter integration
- 100 Gbps architecture
- Flow offload to hardware
- Open problems & future work
- Conclusions

Why Deep Packet Inspection?

for Monitoring and Security

Traditional Monitoring

- The old world:
  - Port 80 -> HTTP
  - Port 443 -> HTTPS
- The real world today:
  - Netflix, Zoom, Corporate VPN, C&C channels, they all use port 443
  - Web-based SaaS replaced dedicated ports
  - Evasion tools deliberately mimic web traffic

Intrusion Detection Systems

- IDS inspection is traditionally based on signatures and rule-based approaches (byte-based payload analysis)

- They shown their limitations in detection capability, especially when attackers heavily rely on **encryption** to obfuscate communications.

```
alert tcp any any <> any 443 (msg:"APT.Backdoor.MSIL.SUNBURST";
content:"|16 03|"; depth:2; content:"|55 04 03|"; distance:0;
content:"digitalcollege.org"; within:50; sid:77600846; rev:1;)
```

nDPI

- **Open source** (GNU LGPL) Deep Packet Inspection toolkit.
- Supports hundreds of protocols including:
  - P2P (BitTorrent)
  - Messaging (Viber, Whatsapp, Telegram, Facebook)
  - Multimedia (YouTube, Last.gm, iTunes)
  - Conferencing (Skype, Webex, Teams, Meet, Zoom)
  - Streaming (Zattoo, Disney, Netflix)
  - Business (VNC, RDP, Citrix)
  - Gaming

What nDPI Provides

- Detect:
  - Layer-7 Application Protocol, e.g. HTTP, YouTube, BitTorrent, etc.
  - Category, e.g. Social Networks, Gaming, Streaming, etc.
  - Geographical location, e.g. country or continent, with the integration with GeoIP databases
  - Autonomous System
- Extract
  - Layer-7 metadata, e.g. TLS SNI, HTTP User Agent, etc.
- Compute
  - Flow Risks, e.g. TLS with no SNI, Binary/executable transfer, etc.

What is a Protocol in nDPI

- Each protocol is identified as MAJOR.MINOR (usually TRANSPORT.APPLICATION)
  - Example: QUIC.YouTube
- Today most protocols are HTTP/TLS-based.
- nDPI includes support for dynamic protocol detection based on:
  - DNS query name
  - HTTP Host/Server header fields
  - TLS/QUIC SNI (Server Name Indication)

```
{ "netflix.com",   NULL, "netflix" TLD,   "NetFlix", NDPI_PROTOCOL_NETFLIX, NDPI_PROTOCOL_CATEGORY_STREAMING, NDPI_PROTOCOL_FUN },
{ "nflxvideo.net", NULL, "nflxvideo" TLD, "NetFlix", NDPI_PROTOCOL_NETFLIX, NDPI_PROTOCOL_CATEGORY_STREAMING, NDPI_PROTOCOL_FUN },
```

Why (still) DPI?

- Internet traffic is increasingly encrypted (>80%).
- Many traditional clear-text protocols are moving to encryption (e.g. DNS vs DNS-over-HTTPS).
- There is still a lot we can do.
  - Identify the service based on visible information (SNI, CDN, etc.).
  - Analyse encrypted traffic to detect issues, even without accessing the (encrypted) payload.
  - Detect whether a TLS connection is a HTTPS connection, a VPN, or something else.
  - Recognize Malwares, with Cryptographic Fingerprinting and statistical properties, entropy, traffic bins (packet lengths and timings), etc.
  - Associate flow risks to identify communications that are affected by security issues.

Cryptographic Fingerprinting

JA3, JA4, and their structural flaws

ntop

Plain-Text vs Encryption

Plain-Text

- Clear content.
- Signature matching or regex on payload (IDS).
- DPI on (plain) payload.

Encryption

- Handshake with configuration parameters.
  - Cipher suites, EC curves, extensions.
- Tied to OS, TLS library, browser engine.
- ➡ Can be used to identify a client.

JA3 (2017)

- TLS Fingerprinting for TLS/SSL (similar to HASSH for SSH).
- Designed for cybersecurity (malware detection).
- Quickly became the industry standard (shipped in Zeek, Suricata, nDPI, etc.).
- Creates a fingerprint of the handshake (cipher suites, extensions, elliptic curves).
- Problem: JA3 hash is based on the exact order of TLS extensions and cipher suites -> Malwares can circumvent it by randomizing extensions and reordering the cipher suite.

JA4 (2023)

- TLS Fingerprinting designed to overcome the limitations of JA3.
  - Sorting and normalizing cipher suites and extensions.
  - Structured format, more human-readable.
- Problem: there are still structural flaws.
  - Does not account for *Ephemeral* TLS extensions.
  - Malwares (and modern browsers) randomize extension parameters.

nDPI Fingerprint

- Designed to overcome the limitations of JA4 and improve reliability for identifying client applications.
- Ignore ephemeral TLS extensions (*Session Ticket*, *Pre-Shared Key*, *Padding*), thus making it suitable for modern traffic.
- Reduce collisions by including a IP/TCP fingerprint computed on TCP flags, IP TTL bucket, TCP window, TCP options, properly masked.

JA4 vs nDPI Fingerprint

```
ndpiReader -i tls.pcap -K json -k out.json -q
cat out.json | jq | grep ja4
      "ja4": "t13d1516h2_8daaf6152771_d8a2da3f94cd",
      "ja4": "t13d1516h2_8daaf6152771_d8a2da3f94cd",
      "ja4": "t13d1517h2_8daaf6152771_b6f405a00624",
      "ja4": "t13d1517h2_8daaf6152771_b6f405a00624",
      "ja4": "t13d1516h2_8daaf6152771_d8a2da3f94cd",
      "ja4": "t13d1516h2_8daaf6152771_d8a2da3f94cd",
```

```
ndpiReader --cfg "tls,metadata.ja_ignore_ephemeral_tls_extn,1" -i tls.pcap -K json -k out.json -q
cat out.json | jq | grep ja4
      "ja4": "t13d1515h2_8daaf6152771_31ec0a762479",
      "ja4": "t13d1515h2_8daaf6152771_31ec0a762479",
      "ja4": "t13d1515h2_8daaf6152771_31ec0a762479",
      "ja4": "t13d1515h2_8daaf6152771_31ec0a762479",
      "ja4": "t13d1515h2_8daaf6152771_31ec0a762479",
      "ja4": "t13d1515h2_8daaf6152771_31ec0a762479",
```

Performance Baseline

nDPI performance

DPI Lifecycle

- Dissectors are evaluated based on Layer-4 information, starting with the one that will most likely match the flow (e.g. TCP/80 -> HTTP dissector first).

- Flow metadata keep the state for non-matching dissectors in order to skip them in future iterations.

- Analysis lasts until a match is found or after too many attempts. Typical figures (average):

```
DPI Packets (TCP):     301    (5.38 pkts/flow)
DPI Packets (UDP):     182    (1.88 pkts/flow)
DPI Packets (other):   12     (1.00 pkts/flow)
```

nDPI Performance (Per Core)

```
nDPI Memory statistics:
  nDPI Memory (once):      3.69 KB
  Flow Memory (per flow):  1.28 KB
  Total memory allocated:  955.13 MB
  Setup Time:              32 msec
  Packet Processing Time:  756 msec

Traffic statistics:
  Ethernet bytes:        1043493248    (includes ethernet CRC/IFC/trailer)
  Discarded bytes:       1269660
  IP packets:            3000543       of 3295278 packets total
  IP bytes:              971480216     (avg pkt size 294 bytes)
  Unique flows:          527
  TCP Packets:           2567005
  UDP Packets:           411579
  VLAN Packets:          0
  MPLS Packets:          0
  PPPoE Packets:         0
  Fragmented Packets:    0
  Max Packet size:       8981
  Packet Len < 64:       1517845
  Packet Len 64-128:     208516
  Packet Len 128-256:    449421
  Packet Len 256-1024:   510237
  Packet Len 1024-1500:  312214
  Packet Len > 1500:     2310
  nDPI throughput:       3.97 M pps / 10.28 Gb/sec
  Analysis begin:        17/Aug/2010 22:23:10
  Analysis end:          17/Aug/2010 22:49:24
  Traffic throughput:    1.91 K pps / 5.06 Mb/sec
  Traffic duration:      1573.805 sec
  Guessed flow protos:   63
  DPI Packets (TCP):     1848          (9.73 pkts/flow)
  DPI Packets (UDP):     1165          (4.10 pkts/flow)
  DPI Packets (other):   53            (1.00 pkts/flow)
```

Linux Netfilter Integration

NF_QUEUE & conntrack lifecycle

DPI in the Kernel: Design Constraints

- Integrating DPI into the Linux kernel provides application awareness and enables Layer-7 firewalling capabilities.
- However, this should be avoided:
  - DPI is complex, a dissector crash could take down the kernel.
  - Long DPI processing could introduce overhead and latency.
  - Kernel modules are hard to reload safely for protocol updates.

A Decoupled Architecture

- Idea:
  - Use the NF_QUEUE + conntrack infrastructure provided by the kernel
  - Run DPI in user space, enforces the resulting verdict in the kernel
- With this approach:
  - Kernel never touches DPI logic (failure isolation)
  - Standard Netfilter mechanisms do all the work
  - Zero kernel patches required

Classification Lifecycle

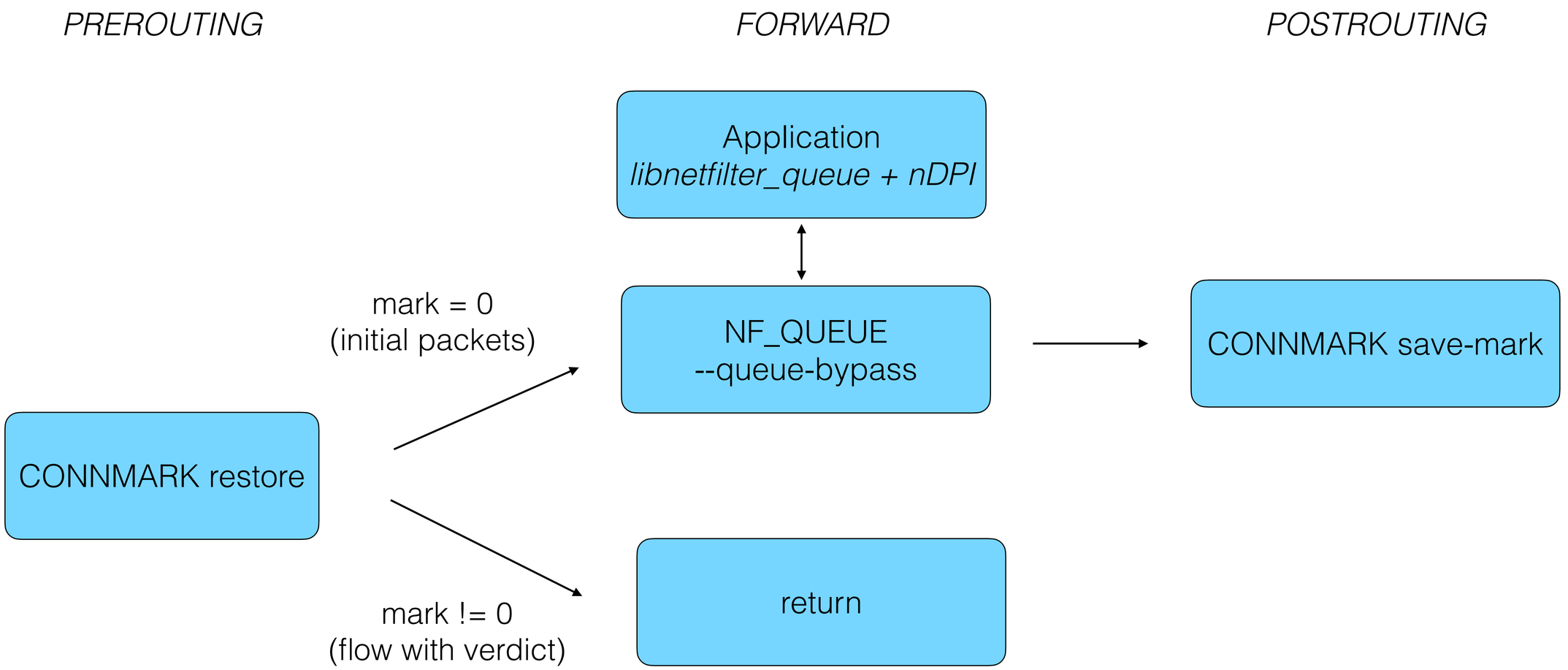
1. Send packets for unclassified flows to user-space.

2. Run Deep Packet Inspection in the application (Slow-Path).

3. Detect the application protocol and compute a verdict according to the policies (mark)

4. Set the marker to the conntrack entry and apply the verdict in kernel space (Fast-Path). Packets are no longer sent to user-space.

NF_QUEUE + conntrack Architecture



Sample Configuration

```
# CT -> Packet: restore verdict/mark from the conntrack
iptables -t mangle -A PREROUTING -j CONNMARK --restore-mark

# Already classified (mark != 0) -> leave the chain, don't go to userspace via NFQUEUE
iptables -t mangle -A FORWARD -m mark ! --mark 0 -j RETURN

# Unclassified (mark == 0) -> hand off to the application (or fail-open if down)
iptables -t mangle -A FORWARD -j NFQUEUE --queue-num 0 --queue-bypass -m physdev \
   --physdev-in $IFACE

# Packet -> CT: save verdict/mark set by the application into the conntrack entry
iptables -t mangle -A POSTROUTING -j CONNMARK --save-mark
```

Properties of this Design

- Isolation: DPI crash doesn't affect kernel
- Stateful: conntrack carries Application ID for the flow lifetime.
- Low overhead: only first N packets are queued for processing, fast-path for the rest.
- Composable: works with existing iptables/nftables rules.

Scaling Out

- Multiple NF_QUEUE instances, multiple application threads
- NFQUEUE --queue-balance hashes on the 5-tuple in the kernel (a flow lands on the same queue/thread for its whole lifetime)
  - no locking, no cross-thread data)

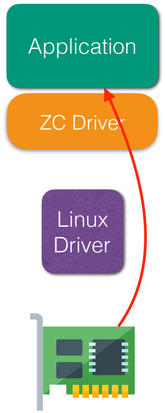
```
iptables -t mangle -A FORWARD -j NFQUEUE --queue-balance 0:3 --queue-bypass \
  -m physdev --physdev-in $IFACE
```

100 Gbps Architecture

PF RING, kernel bypass

Scaling to 100 Gbps [1/2]

- First-order bottleneck: kernel copy overhead
  - Standard Linux network stack: per-packet copy, syscall, IRQ
  - At 100 Gbps / 148 Mpps (64-byte), kernel can't keep up
- Solution: Zero Copy capture (e.g. PF RING ZC or DPDK)
  - NIC DMA (user-space buffer directly)
  - nDPI engine reads from the buffer (zero per-packet copies)
  - Zero kernel involvement on receive path
  - Scaling out: RSS



Scaling to 100 Gbps [2/2]

- Second-order bottleneck: flow table pressure
- Even with kernel bypass, the dominant cost becomes:
  - Processing and maintaining per-flow state for every packet
  - Cost scales with concurrent flows, not just packet rate
  - Cache and TLB pressure on the hash table
- Example: 10 Mpkt/s with 10M active flows = up to 10 M lookups/sec + evictions.
  - This bottleneck is in host memory, not the NIC

Stateful Traffic Processing

- Monitoring applications, both passive (e.g. NetFlow) or inline (e.g. IPS systems), typically have to analyse and maintain the state of each network communication.
- This requires a flow table, an in-memory data structure where the application keeps the status of network communications (flow key and metadata).
- Metadata may include:
  - Simple statistics about the packet stream (number of packets and bytes)
  - Information from application layer protocols (e.g. the HTTP URL or the VoIP caller) extracted by DPI engines

Flow Offload to Hardware

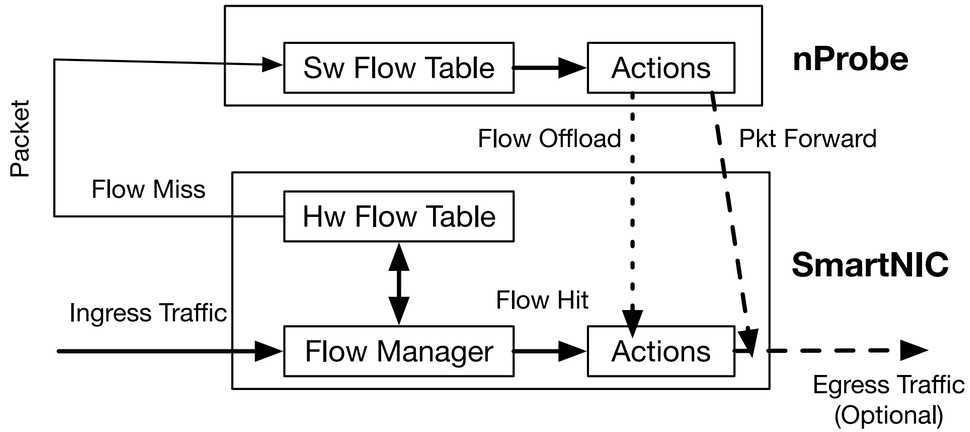
SmartNICs and SuperNICs

ntop

Hardware Flow-Offload Pipeline

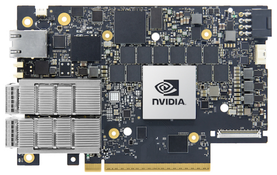
- Concept similar to DPI on NF_QUEUE/conntrack

1. Initial redirection
2. User-space inspection
3. Protocol detection and verdict
4. Flow offload
5. Periodic flow stats retrieval from hardware entries



Target Adapters

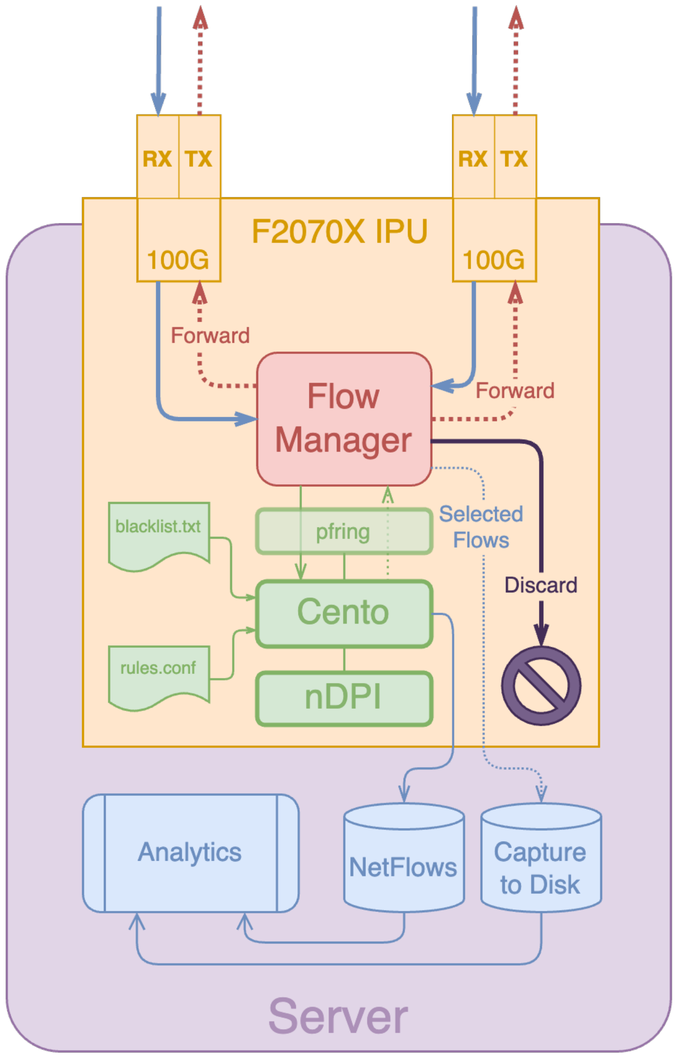
- Napatech NT200 SmartNIC
  - NIC + FPGA engine, constrained programmability.
  - **Native** Flow Offload support (Flow Manager).



- NVIDIA BlueField-3 SuperNIC
  - NIC + general-purpose on-board compute (ARM + accelerators).
  - **Programmable** via DOCA.

Napatech - Flow Manager

- Specialized built-in component implementing flow offload.
- Declared specs:
  - Cache capacity: 140 Million flows
  - Max flow creation rate: 3 Million flows/sec
- Validated with nProbe Cento (100 Gbit NetFlow probe), using the PF_RING API.



Learning Flows

- Flows are programmed by the software, by 5-tuple, when DPI completes.
- Flow (offloaded) metadata include:
  - **Flow ID** generated by the application (used to correlate hardware and software flow table entries).
  - **Action**: discard, forward (to the software or a twin interface when inline).
  - Options for the Flow Manager (e.g. TCP auto unlearn).

Packet Markers

- Packets received from the adapter are marked as:
  - **Miss**: no match / no flow programmed yet.
  - **Hit**: flow match in the adapter.
  - **Unhandled**: unable to match the flow, no hardware pipeline resources.

Hardware Flow Events

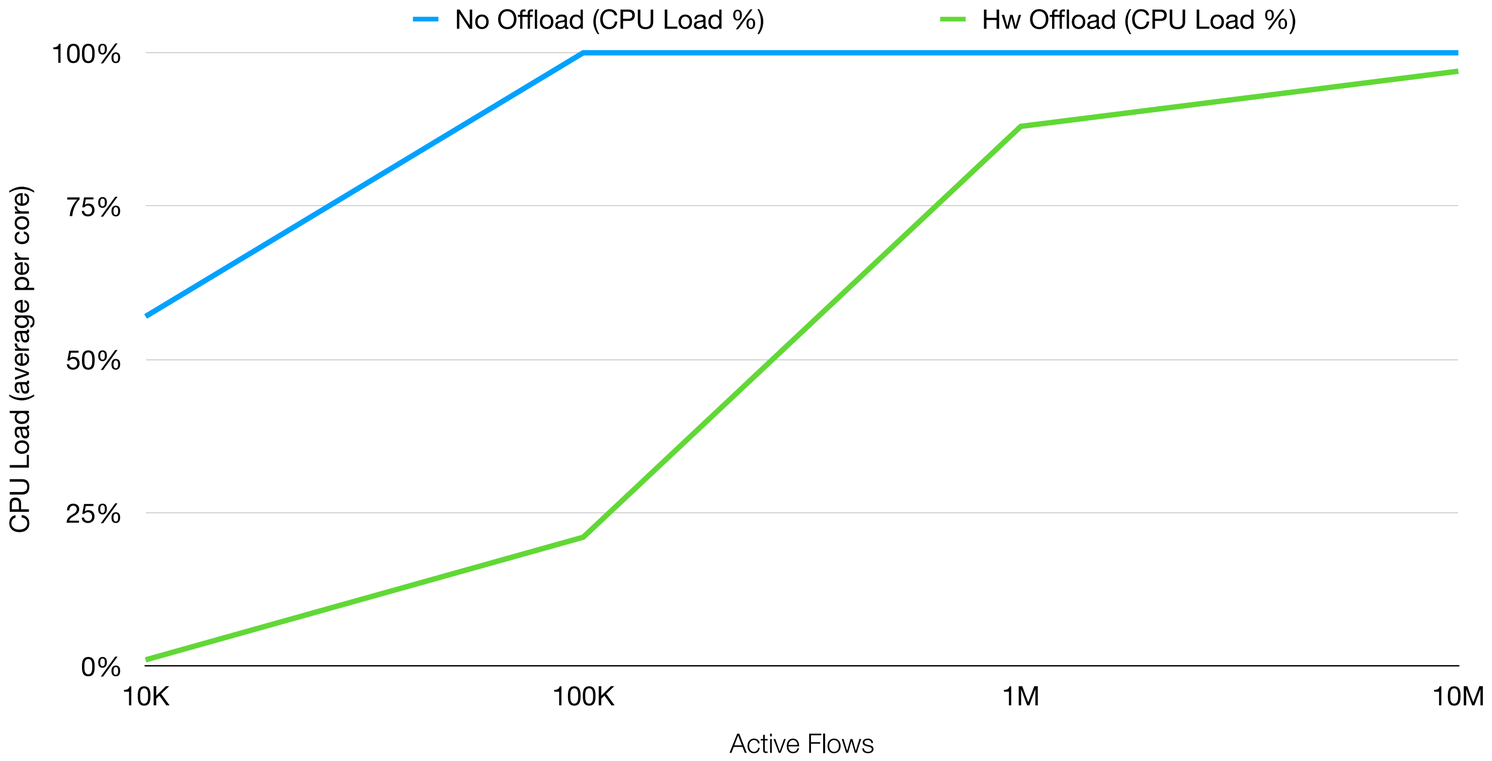
- The adapter delivers flow events, on a dedicated stream, including:
  - Flow unlearned due to **timeout**.
  - Flow unlearned due to **TCP termination**.
  - Flow unlearned due to explicit **application unlearn** command.

Test Results

- Tested on Intel Xeon Gold 6526Y with Napatech NT200A02.
- Traffic load: 89 Mpps with 60-byte packets (60 Gbps).
- CPU load (average per core) and packet loss have been measured, under different flow rates and number of active flows:
  - 10K active flows, 1K new flows/sec.
  - 100K active flows, 10K new flows/sec.
  - 1M active flows, 100K new flows/sec.
  - 10M active flows, 1M new flows/sec.

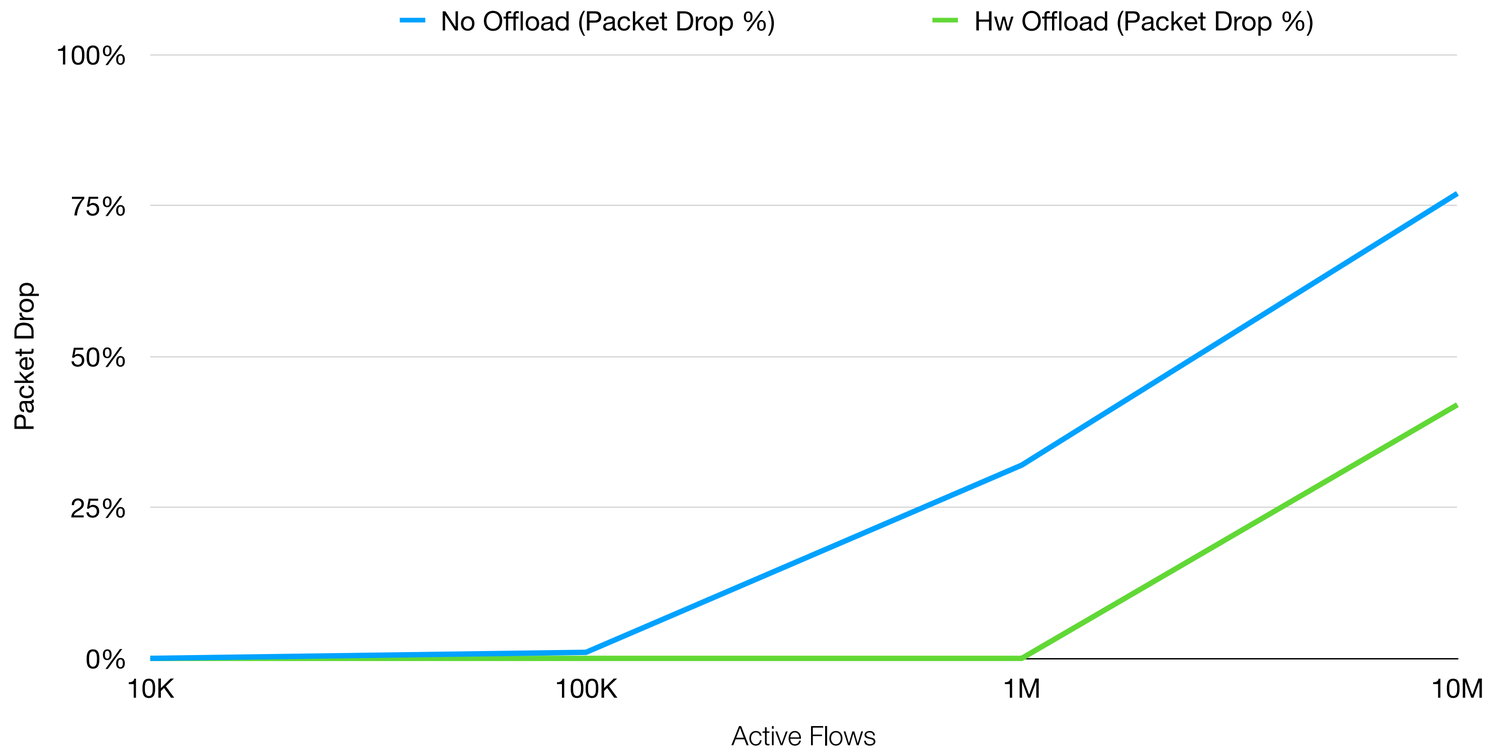
CPU Load (Inline)

- @ 89 Mpps 60 Gbps with DPI on 16 cores (RSS)



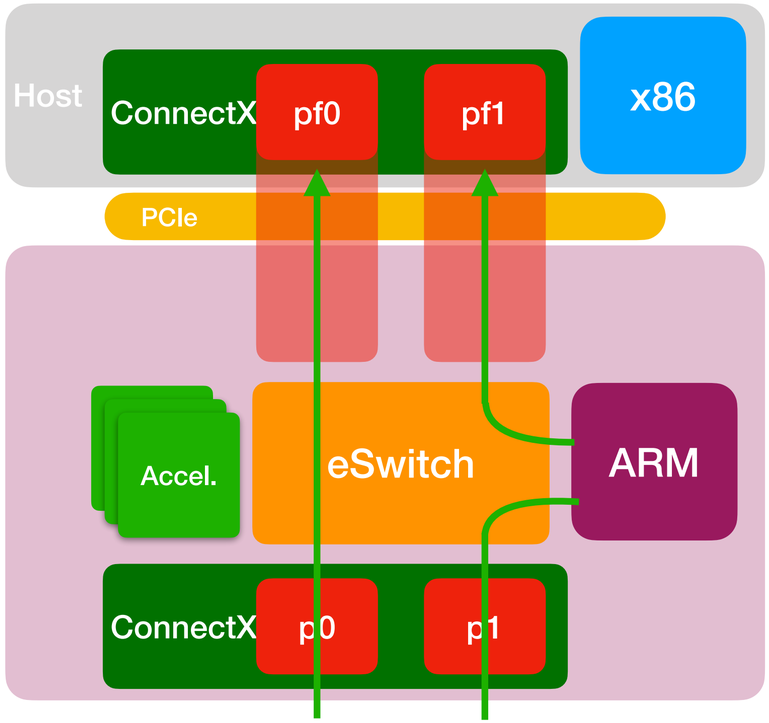
Packet Loss (Inline)

- @ 89 Mpps 60 Gbps with DPI on 16 cores (RSS)

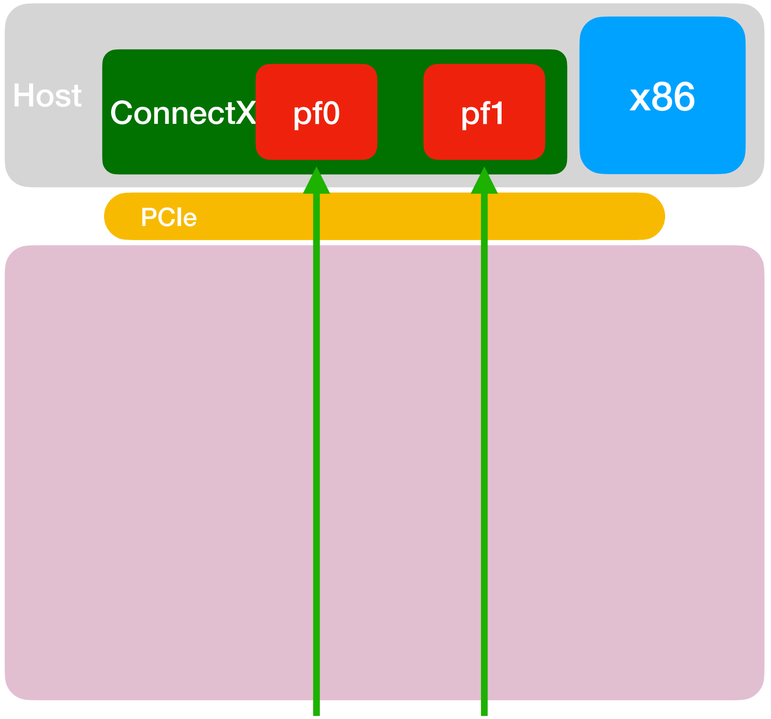


NVIDIA BlueField

- DPU Mode
  - NIC owned and controlled by the embedded ARM subsystem.



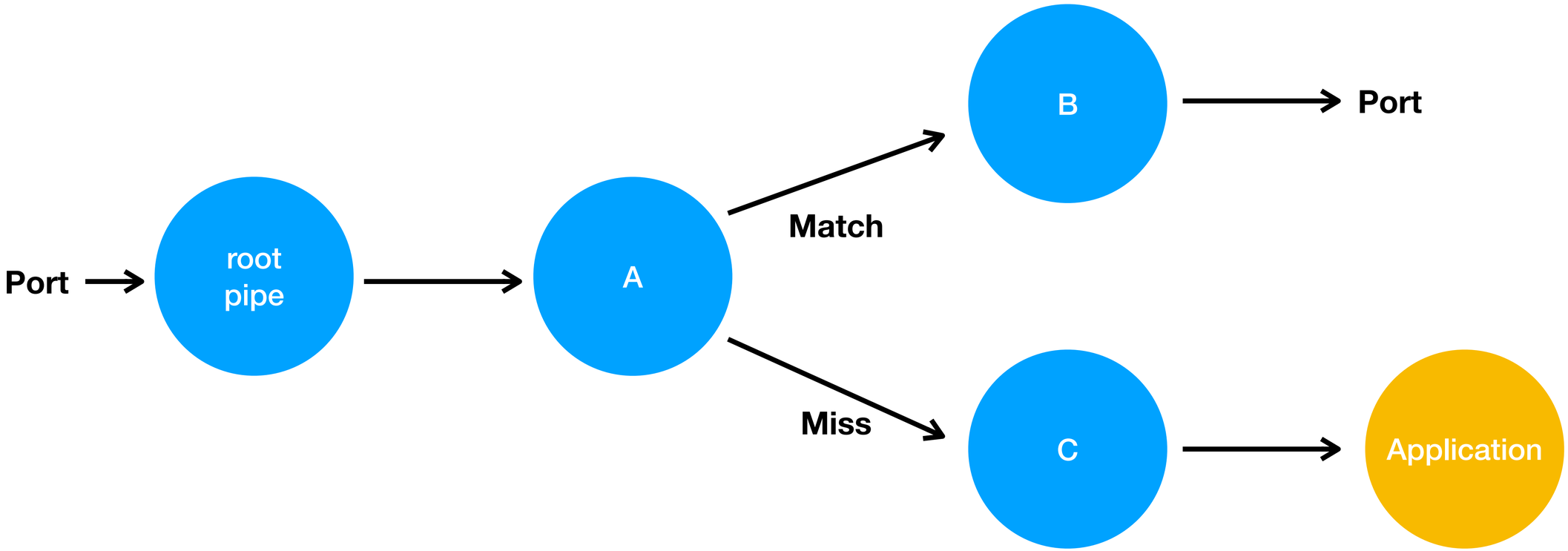
- NIC Mode
  - Behaves exactly like a standard (ConnectX) NIC.



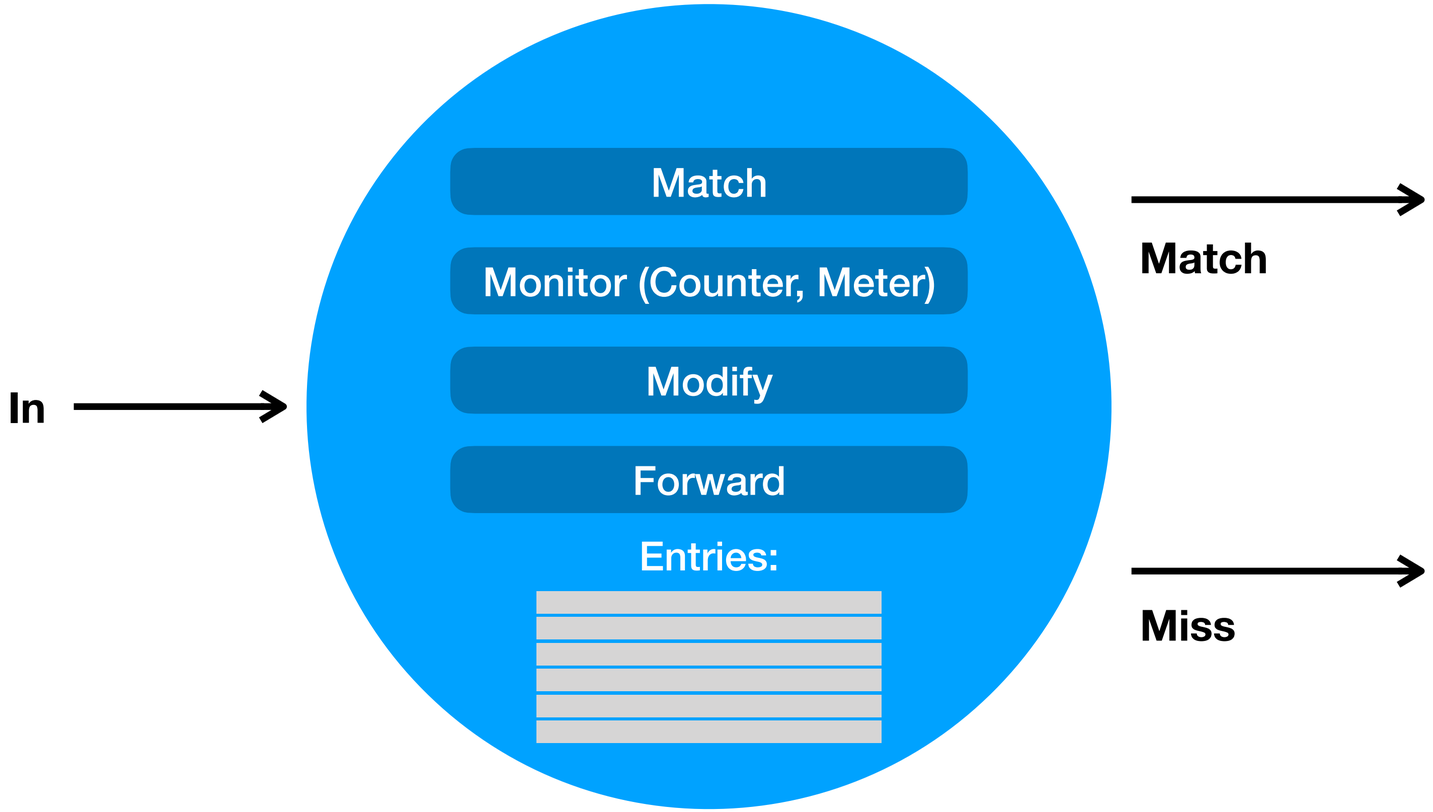
DOCA Flow

- DOCA is an SDK for programming hardware accelerators on NVIDIA adapters.
- Divided into components (*Flow*, *Compress*, etc.)
- DOCA *Flow* is the one required to accelerate packet processing - supported both on NVIDIA BlueField and ConnectX
- APIs for building processing pipelines by creating pipes and chaining them.

Pipeline



Pipe


Match
Monitor (Counter, Meter)
Modify
Forward
Entries:
In
Match
Miss

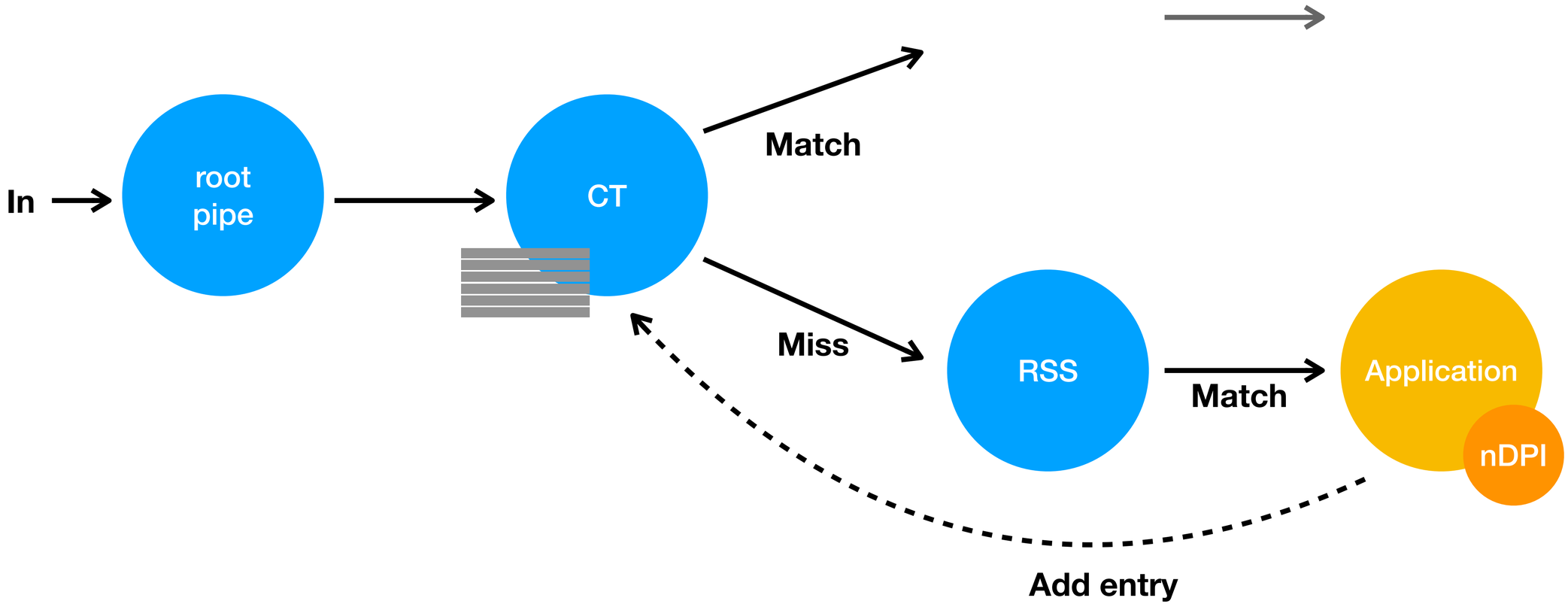
DOCA Flow CT Pipe

- DOCA *Flow CT* (Connection Tracking) is a specialized pipe providing what is required for the flow offload:
  - 5-tuple table to store entries (flows)
  - API to add, remove, update entries
  - Per-entry statistics
  - Flow aging
- Available both on the BlueField DPU and on the host (also on plain ConnectX)

"Kryptonite"

- One-file application implementing Flow Offload using DOCA Flow CT
- Shadow (Software) Flow Table
  - Keep track of offloaded flows
  - Store additional metadata (e.g. DPI)
- Flow export (as text)
- (Optional) Inline traffic forwarding
- Source code: https://github.com/ntop/bluefield-kryptonite

Kryptonite CT-based Pipeline



Test Results

- Tested both on Intel Xeon Gold 6526Y (NIC mode) and on the BlueField-3 ARM (DPU mode).
- Traffic load:
  - 2 Million flows (maximum capacity configurable on BlueField-3 according to our tests).
  - 100 Gbps 200-byte packets (55.8 Mpps) / 40 Gbps 60-byte packets (60 Mpps)

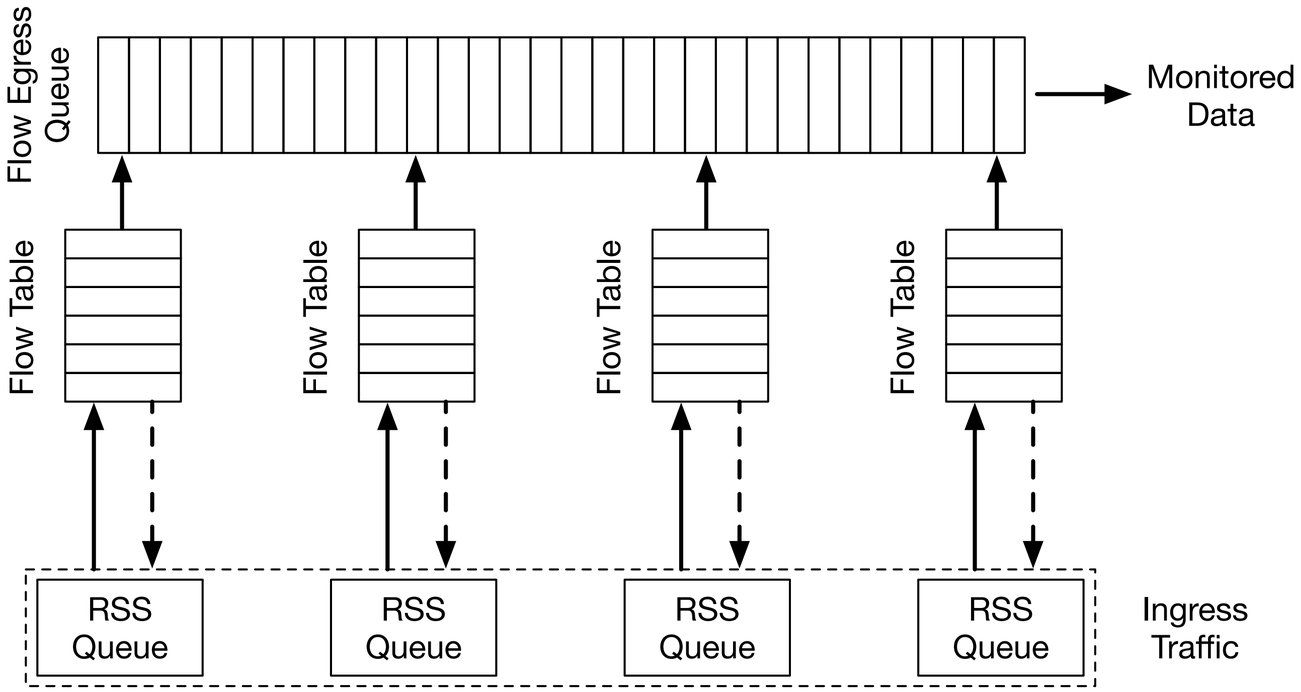
➡ Measured CT performance:

  - Max flow creation rate: 1.5 Million flows/sec on the DPU, 2.7 on the host.
  - Packet loss only occurs (on the Slow Path) when new flows rate exceeds the max creation rate.
  - Full rate (no loss) for traffic handled by the pipeline (Fast Path).

Napatech vs BlueField

- Napatech NT200 FPGA with Flow Manager
  - 140 Million flows
  - 1.5 Million flows/sec single-core / 3 Million flows/sec with multiple streams
  - Selected actions
- BlueField-3 with DOCA Flow CT
  - 2 Million flows (maximum configurable in our tests)
  - 1.3-1.5 M flows/s on DPU / up to 2.7 Million flows/sec on x86 (NIC mode)
  - Programmable actions

Scaling Out


Flow Egress Queue
Monitored Data
Flow Table
Flow Table
Flow Table
Flow Table
RSS Queue
RSS Queue
RSS Queue
RSS Queue
Ingress Traffic

Open Problems & Future Work [1/2]

- Hardware flow-table offload defers the bottleneck but does not eliminate it.
- At high flow-creation rate, host (or DPU) memory becomes the bottleneck (hash table lookups).
- Directions:
  - Mitigate cache/TLB pressure with cache-optimized flow table.
  - Address attack-induced flow-table exhaustion (e.g. DDoS)

Open Problems & Future Work [2/2]

- The netfilter and hardware offload hide traffic (to DPI) once it has been offloaded.
- Emerging protocols introduce additional challenges:
  - TLS 1.3 encrypts the Certificate message.
  - Encrypted Client Hello (ECH) hides SNI and most ClientHello fields, removing the primary signal for fingerprinting and SNI-based filtering.
  - QUIC Connection ID decoupled from the 5-tuple and Connection ID rotation break the flow correlation assumption that flow table offload relies on.

Conclusions

- JA3 and JA4 share a structural flaw: any exact-match fingerprint over mutable fields is sensitive to deliberate randomization. nDPI fingerprint tries to address this.

- NF_QUEUE + conntrack integration: let nDPI classify initial packets of a flow in user space while the kernel enforces verdicts natively, bounding DPI's exposure to kernel-space failure modes and with zero kernel patches.

- Hardware flow offload is empirically validated: Napatech (140M flows, up to 3 Mflows/sec) and BlueField-3 (2M flows tested, up to 2.7 Mflows/sec). Both reduce host CPU load, PCIe and memory bandwidth utilization.

Thank You

Resources

- nDPI: github.com/ntop/nDPI
- PF_RING: github.com/ntop/PF_RING
- Kryptonite: github.com/ntop/bluefield-kryptonite
- ntop blog: ntop.org/blog

Contacts

- deri@ntop.org
- cardigliano@ntop.org