Hardware Offloading for Connection Tracking

Amanda Nee, Andreas Kinnunen, Emilia Blidborg
Johan Edman, Raman Salih, Vignesh Purushotham Srinivas

Introduction

- Network traffic is increasing and is expected to continue to increase [3].

Introduction

- Network traffic is increasing and is expected to continue to increase [3].
- Old specialized hardware is inflexible and costly.

Introduction

- Network traffic is increasing and is expected to continue to increase [3].
- Old specialized hardware is inflexible and costly.
- Moore's law is dead, long live Moore's law!

Introduction

- Network traffic is increasing and is expected to continue to increase [3].
- Old specialized hardware is inflexible and costly.
- Moore's law is dead, long live Moore's law!
- Network Function Virtualization (NFV) is a popular alternative to expensive specialised hardware and uses commodity hardware.

Introduction

- Network traffic is increasing and is expected to continue to increase [3].
- Old specialized hardware is inflexible and costly.
- Moore's law is dead, long live Moore's law!
- Network Function Virtualization (NFV) is a popular alternative to expensive specialised hardware and uses commodity hardware.
- Software offloading can increase the efficiency of CPU usage.

Introduction

- Network traffic is increasing and is expected to continue to increase [3].
- Old specialized hardware is inflexible and costly.
- Moore's law is dead, long live Moore's law!
- Network Function Virtualization (NFV) is a popular alternative to expensive specialised hardware and uses commodity hardware.
- Software offloading can increase the efficiency of CPU usage.
- Hardware offloading is an even more promising when it comes further increasing the hardware efficiency.

Goals

The primary objective of this project is to evaluate the effectiveness of hardware offloading for connection tracking.

Goals

The primary objective of this project is to evaluate the effectiveness of hardware offloading for connection tracking.

- Develop a 100Gbps Traffic Generator.

Goals

The primary objective of this project is to evaluate the effectiveness of hardware offloading for connection tracking.

- Develop a 100Gbps Traffic Generator.
- Implement a basic forwarder to derive baseline performance.

Goals

The primary objective of this project is to evaluate the effectiveness of hardware offloading for connection tracking.

- Develop a 100Gbps Traffic Generator.
- Implement a basic forwarder to derive baseline performance.
- Implementing the load balancer with hardware offloading capabilities.

Goals

The primary objective of this project is to evaluate the effectiveness of hardware offloading for connection tracking.

- Develop a 100Gbps Traffic Generator.
- Implement a basic forwarder to derive baseline performance.
- Implementing the load balancer with hardware offloading capabilities.
- Study the benefits, disadvantages and limitations of offloading.

Goals

The primary objective of this project is to evaluate the effectiveness of hardware offloading for connection tracking.

- Develop a 100Gbps Traffic Generator.
- Implement a basic forwarder to derive baseline performance.
- Implementing the load balancer with hardware offloading capabilities.
- Study the benefits, disadvantages and limitations of offloading.
- Develop and benchmark techniques to optimize the offloading.

FastClick and DPDK

FastClick and DPDK

FastClick

- FastClick [2] is a high-speed userspace packet processing framework that evolved from Click Modular Router [4].

FastClick and DPDK

FastClick

- FastClick [2] is a high-speed userspace packet processing framework that evolved from Click Modular Router [4].
- FastClick allows NFV development by allowing users to compose graphs of elements.

FastClick and DPDK

FastClick

- FastClick [2] is a high-speed userspace packet processing framework that evolved from Click Modular Router [4].
- FastClick allows NFV development by allowing users to compose graphs of elements.

DPDK

- The Data Plane Development Kit (DPDK) is an open-source software project that provides a set of data plane libraries and NIC drivers for packet processing in userspace.

FastClick and DPDK

FastClick

- FastClick [2] is a high-speed userspace packet processing framework that evolved from Click Modular Router [4].
- FastClick allows NFV development by allowing users to compose graphs of elements.

DPDK

- The Data Plane Development Kit (DPDK) is an open-source software project that provides a set of data plane libraries and NIC drivers for packet processing in userspace.
- DPDK uses Poll Mode Driver(PMD) to bypass the the standard networking stack to accelerate the packet processing.

Load Balancing

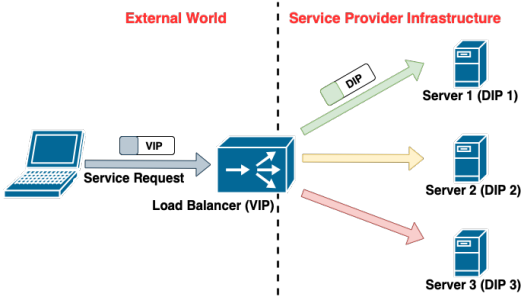
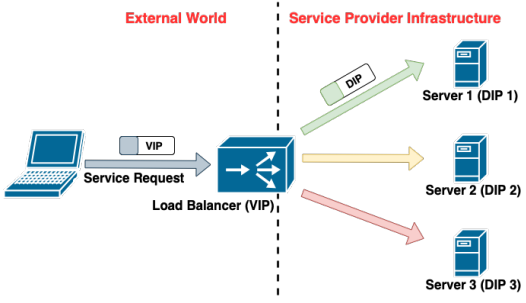


Figure 1: Load Balancing

- Load balancer is a crucial network component in today's internet infrastructure.

Load Balancing



- Load balancer is a crucial network component in today's internet infrastructure.
- A load balancer provides translations from virtual IPs(VIPs) to direct IPs(DIPs)

Figure 1: Load Balancing

Load Balancing

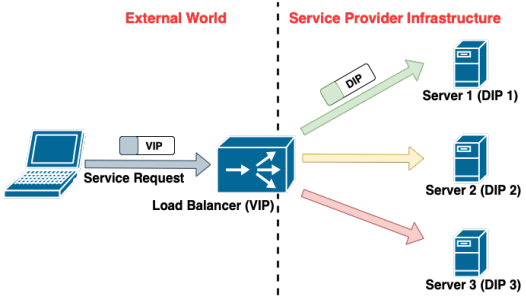


Figure 1: Load Balancing

- Load balancer is a crucial network component in today's internet infrastructure.
- A load balancer provides translations from virtual IPs(VIPs) to direct IPs(DIPs)
- Can become a potential performance bottleneck

Traffic Generator

That is capable of:

- Saturating the link with 100Gbps traffic

Traffic Generator

That is capable of:

- Saturating the link with 100Gbps traffic
- Replaying real traffic traces, CAIDA [1].

Traffic Generator

That is capable of:

- Saturating the link with 100Gbps traffic
- Replaying real traffic traces, CAIDA [1].
- Controlling different parameter such as packet length, number of flows, etc

Traffic Generator

That is capable of:

- Saturating the link with 100Gbps traffic
- Replaying real traffic traces, CAIDA [1].
- Controlling different parameter such as packet length, number of flows, etc

Design:

- Multiple cores replaying traffic simultaneous

Traffic Generator

That is capable of:

- Saturating the link with 100Gbps traffic
- Replaying real traffic traces, CAIDA [1].
- Controlling different parameter such as packet length, number of flows, etc

Design:

- Multiple cores replaying traffic simultaneous
- Use Mindumps instead of pcap traces.

Traffic Generator

That is capable of:

- Saturating the link with 100Gbps traffic
- Replaying real traffic traces, CAIDA [1].
- Controlling different parameter such as packet length, number of flows, etc

Design:

- Multiple cores replaying traffic simultaneous
- Use Mindumps instead of pcap traces.
- Sink - To collect the statistics

Traffic Generator

That is capable of:

- Saturating the link with 100Gbps traffic
- Replaying real traffic traces, CAIDA [1].
- Controlling different parameter such as packet length, number of flows, etc

Design:

- Multiple cores replaying traffic simultaneous
- Use Mindumps instead of pcap traces.
- Sink - To collect the statistics

Testbed Basic Forwarder

- 100 Gbps Mellanox ConnectX-5 cards

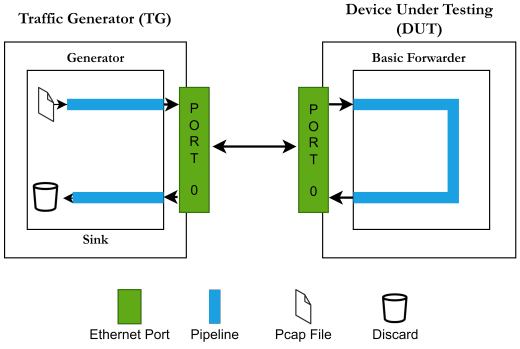


Figure 2: Hardware topology Basic Forwarder

Testbed Basic Forwarder

- 100 Gbps Mellanox ConnectX-5 cards

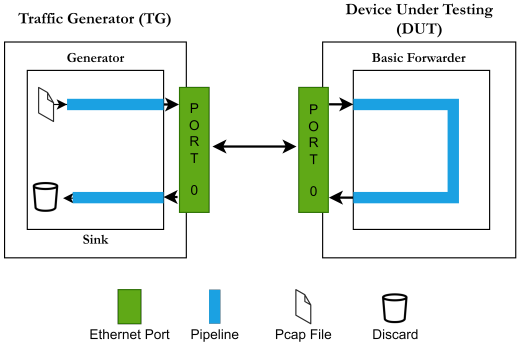


Figure 2: Hardware topology Basic Forwarder

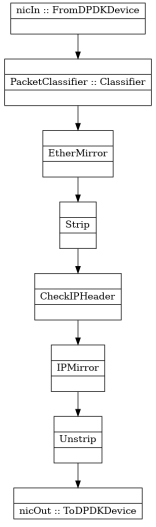


Figure 3: Internal structure Basic Forwarder

Testbed Load Balancer

- 100 Gbps Mellanox ConnectX-5 cards

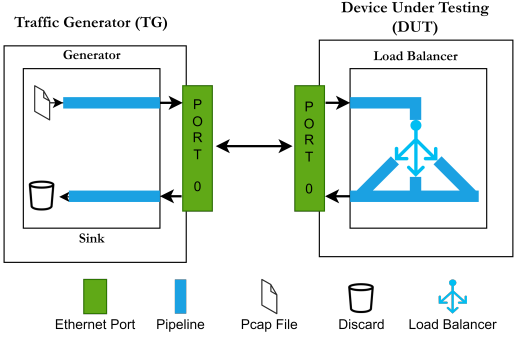


Figure 4: Hardware topology Load Balancer

Testbed Load Balancer

- 100 Gbps Mellanox ConnectX-5 cards

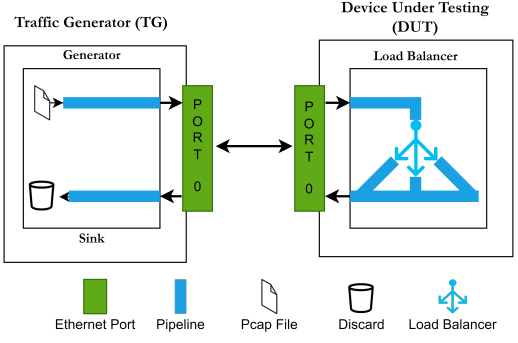


Figure 4: Hardware topology Load Balancer

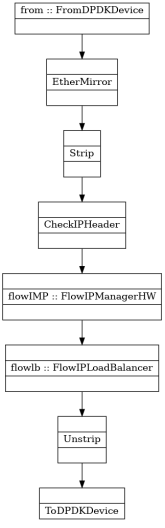


Figure 5: Internal structure Load Balancer

FlowIPManagerHW element

```
if not MARKED then
    FLOW = hashtable_lookup(PACKET)
    if FLOW < 0 then
        FLOW = hashtable_add(PACKET)
        FCB = fcb_get(FLOW)
    else
        FCB = fcb_get(FLOW)
    if OFFLOAD then
        switch OFFLOAD_TYPE do
            case SIZE
                stats_update(PACKET_LENGTH)
                if STATS_BYTECOUNT > THRESHOLD then
                    insert_flow(PACKET, FCB)
            case NAIVE
                insert_flow(PACKET, FCB)
else
    offload_fcb_get(MARKED)
```

Testing

Core Count	Hash Table Size	Flow Count	Packet Size
1	2E6	2E6	1500 bytes
2	4E6	8E6	
4	8E6		
8	16E6		

Table 1: Tested input variables

Testing

Core Count	Hash Table Size	Flow Count	Packet Size
1	2E6	2E6	1500 bytes
2	4E6	8E6	
4	8E6		
8	16E6		

Table 1: Tested input variables

Metrics of interest:

- Throughput
- Latency
- Packetloss

Results 2M Hash Table Entries 2M Flows

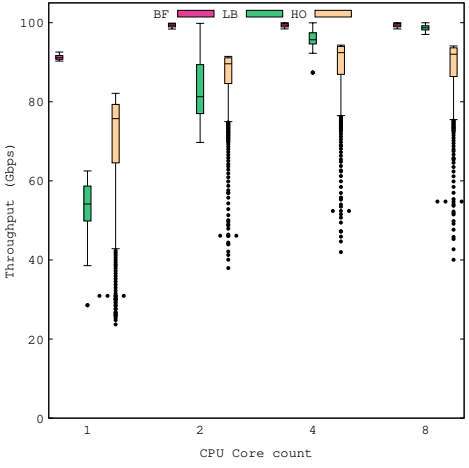


Figure 6: Run 1

Results 2M Hash Table Entries 2M Flows

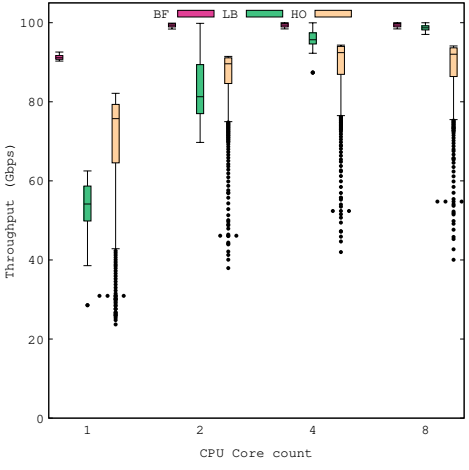


Figure 6: Run 1

- Basic Forwarder in red give baseline performance of packet mirroring

Results 2M Hash Table Entries 2M Flows

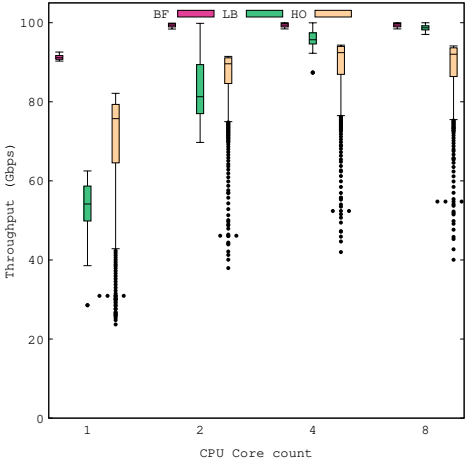


Figure 6: Run 1

- Basic Forwarder in red give baseline performance of packet mirroring
- LoadBalancer in green shows full software performance

Results 2M Hash Table Entries 2M Flows

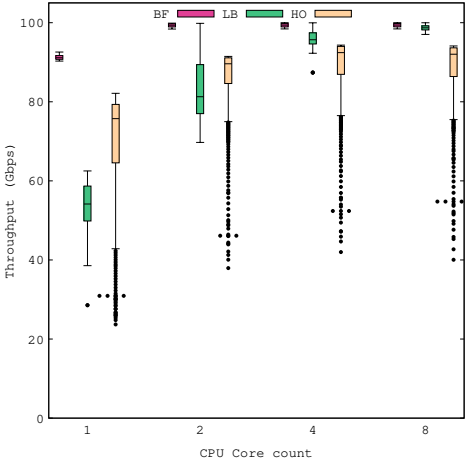


Figure 6: Run 1

- Basic Forwarder in red give baseline performance of packet mirroring
- LoadBalancer in green shows full software performance
- LoadBalancer with Hardware Offloading shows performance when letting the hardware classify some flows

Results 2M Hash Table Entries 2M Flows

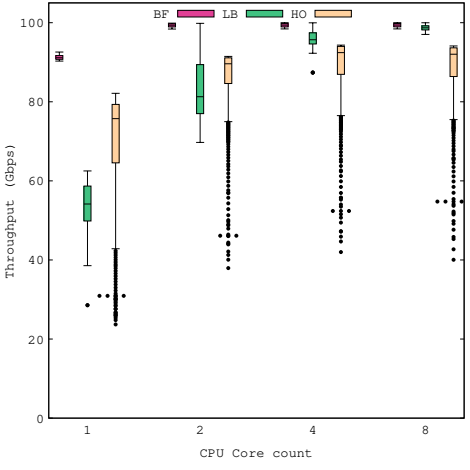


Figure 6: Run 1

- Basic Forwarder in red give baseline performance of packet mirroring
- LoadBalancer in green shows full software performance
- LoadBalancer with Hardware Offloading shows performance when letting the hardware classify some flows
- First run with insertions

Results 2M Hash Table Entries 2M Flows

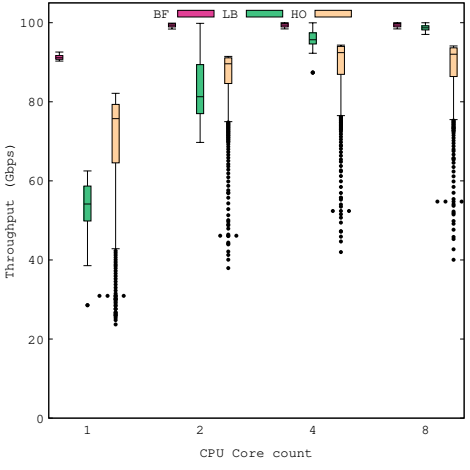


Figure 6: Run 1

- Basic Forwarder in red give baseline performance of packet mirroring
- LoadBalancer in green shows full software performance
- LoadBalancer with Hardware Offloading shows performance when letting the hardware classify some flows
- First run with insertions
- 1 million flows inserted

Results 2M Hash Table Entries 2M Flows

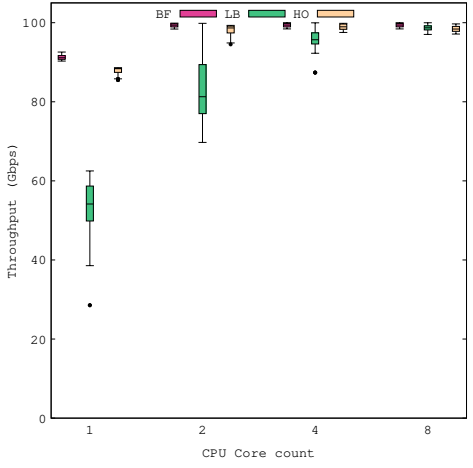


Figure 7: Run 2

Results 2M Hash Table Entries 2M Flows

- Second run to avoid insertions

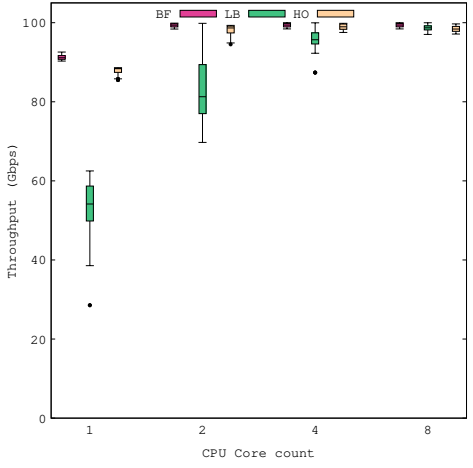


Figure 7: Run 2

Results 2M Hash Table Entries 2M Flows

- Second run to avoid insertions
- We can see the gains in throughput from avoiding looking up the flow in a hash table

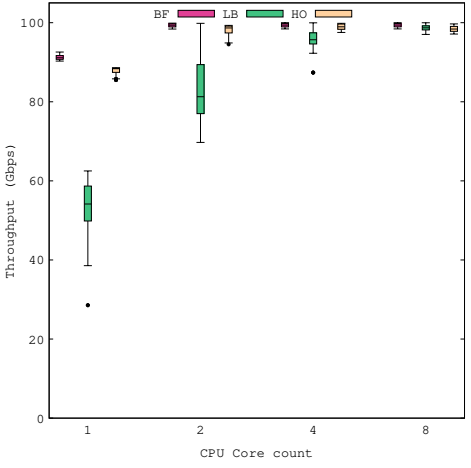


Figure 7: Run 2

Results 2M Hash Table Entries 2M Flows

- Second run to avoid insertions
- We can see the gains in throughput from avoiding looking up the flow in a hash table
- Around half of the flows are offloaded to hardware for classification

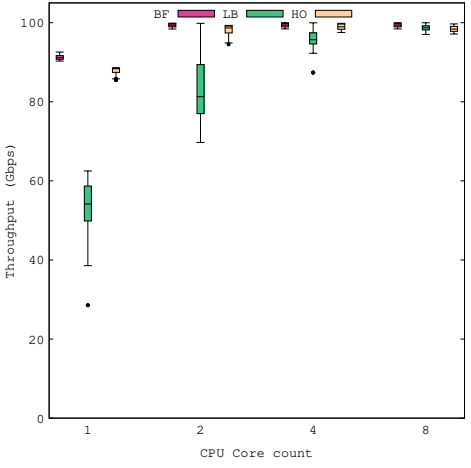


Figure 7: Run 2

Results 2M Hash Table Entries 2M Flows

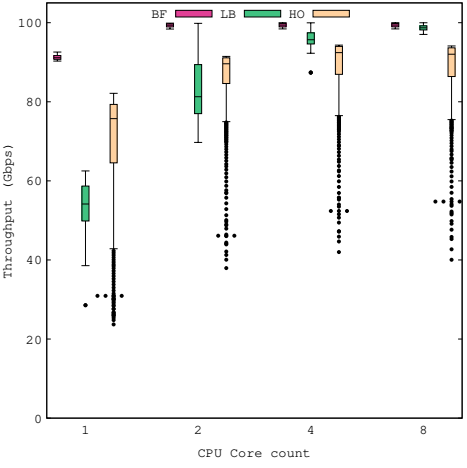


Figure 6: Run 1

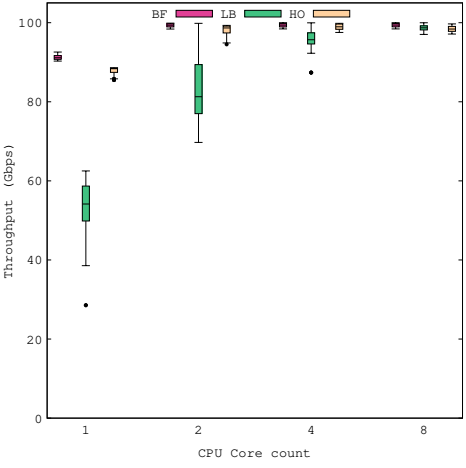


Figure 7: Run 2

Results optimally offloading 2M Hash Table Entries 2M Flows

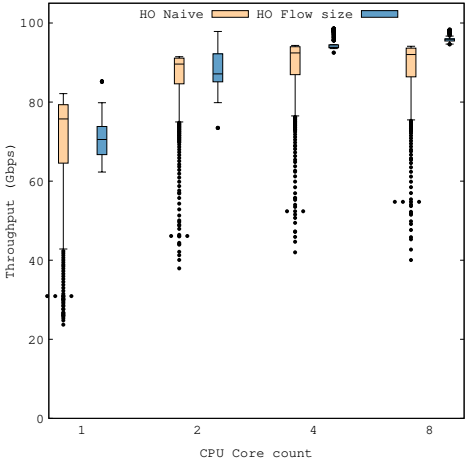


Figure 8: Run 1

Results optimally offloading 2M Hash Table Entries 2M Flows

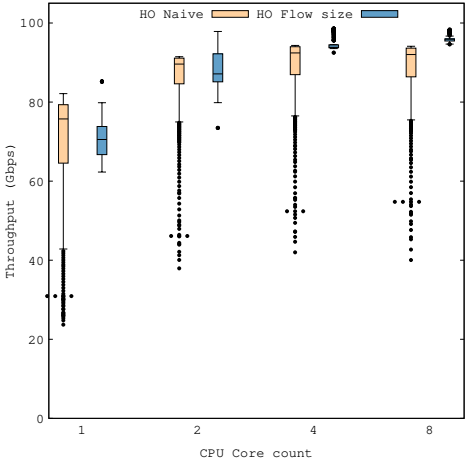


Figure 8: Run 1

- Selectively choosing which flows to offload in blue

Results optimally offloading 2M Hash Table Entries 2M Flows

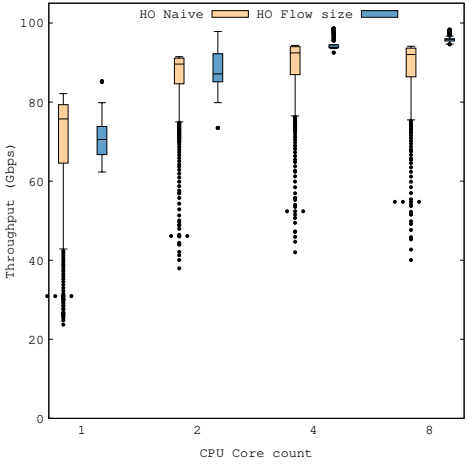


Figure 8: Run 1

- Selectively choosing which flows to offload in blue
- Not as limited while inserting flows

Results optimally offloading 2M Hash Table Entries 2M Flows

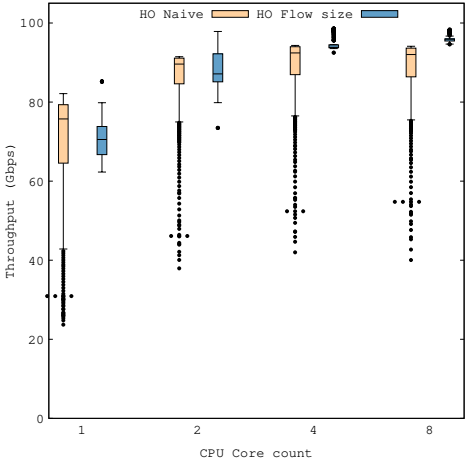


Figure 8: Run 1

- Selectively choosing which flows to offload in blue
- Not as limited while inserting flows
- Fewer outliers

Results optimally offloading 2M Hash Table Entries 2M Flows

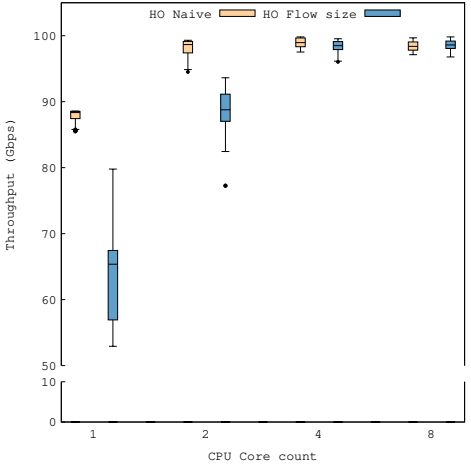


Figure 9: Run 2

Results optimally offloading 2M Hash Table Entries 2M Flows

- After insertions performance seem better for the naive solution

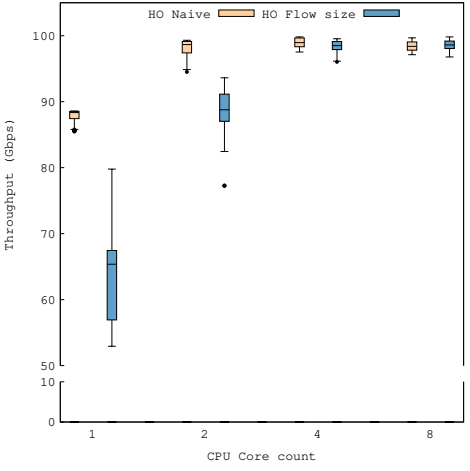


Figure 9: Run 2

Results optimally offloading 2M Hash Table Entries 2M Flows

- After insertions performance seem better for the naive solution
- Perhaps not 1 million flows have exceeded the threshold for the selective variant?

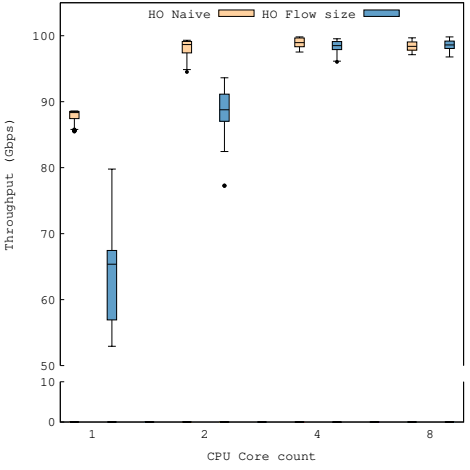


Figure 9: Run 2

Results optimally offloading 2M Hash Table Entries 2M Flows

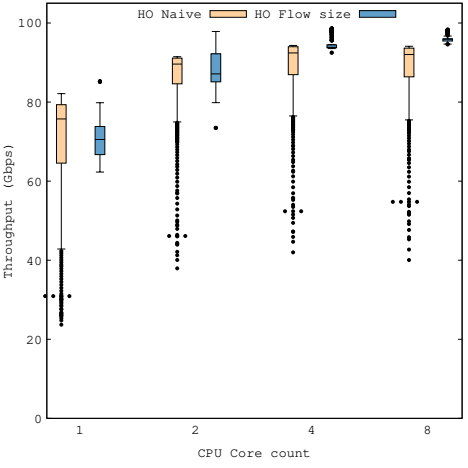


Figure 8: Run 1

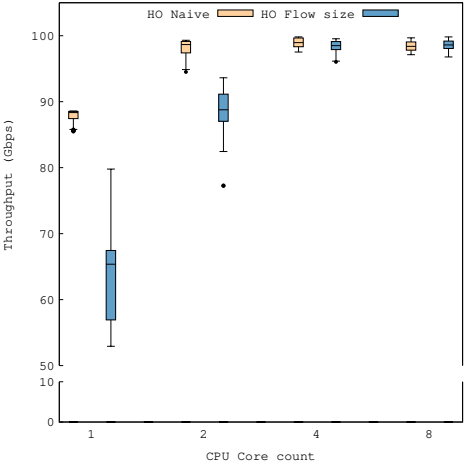


Figure 9: Run 2

Results optimally offloading 16M Hash Table Entries 8M Flows

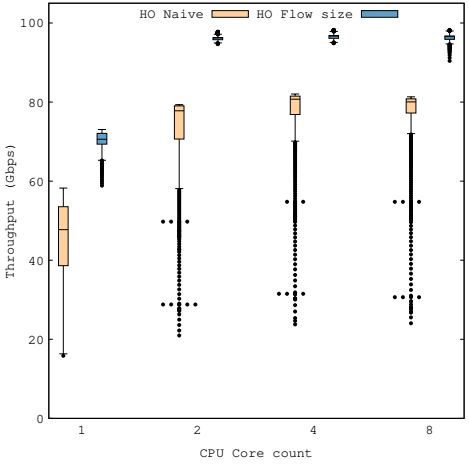


Figure 10: Run 1

Results optimally offloading 16M Hash Table Entries 8M Flows

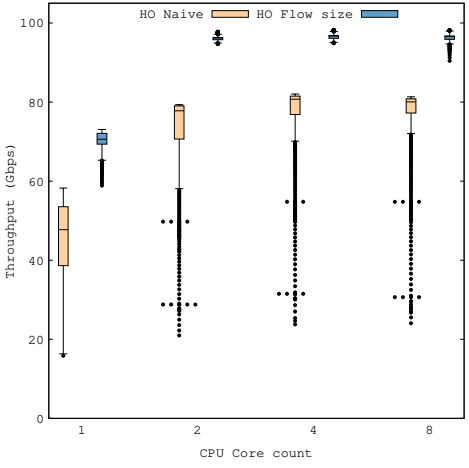


Figure 10: Run 1

- Amount of flows increased

Results optimally offloading 16M Hash Table Entries 8M Flows

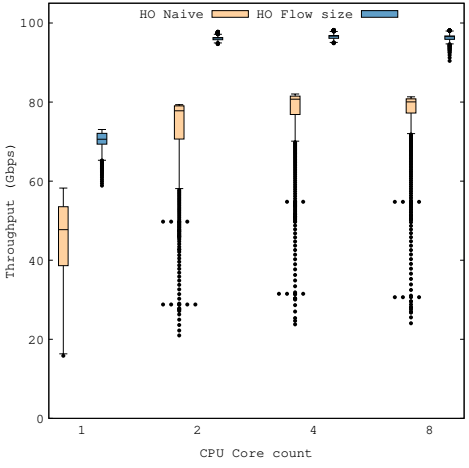


Figure 10: Run 1

- Amount of flows increased
- Selective variant runs away in throughput during insertion

Results optimally offloading 16M Hash Table Entries 8M Flows

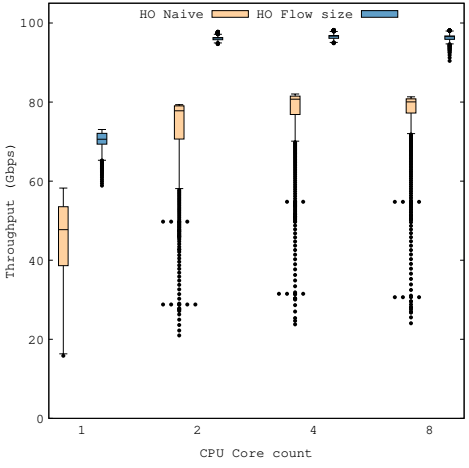


Figure 10: Run 1

- Amount of flows increased
- Selective variant runs away in throughput during insertion
- More outliers than previously

Results optimally offloading 16M Hash Table Entries 8M Flows

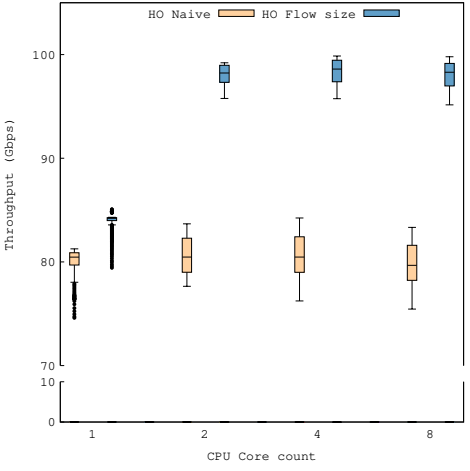


Figure 11: Run 2

Results optimally offloading 16M Hash Table Entries 8M Flows

- The previously seen increase in throughput holds up the second run

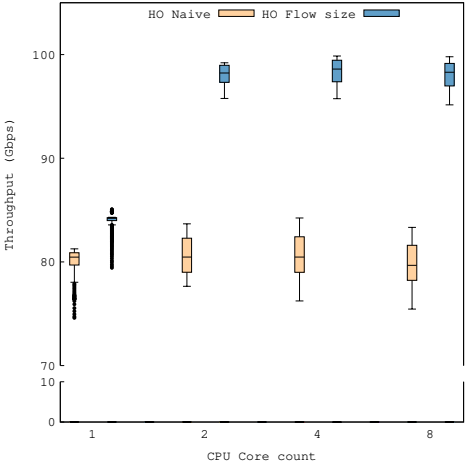


Figure 11: Run 2

Results optimally offloading 16M Hash Table Entries 8M Flows

- The previously seen increase in throughput holds up the second run
- The naive solution don't seem as performant as before when the amount of flows increase

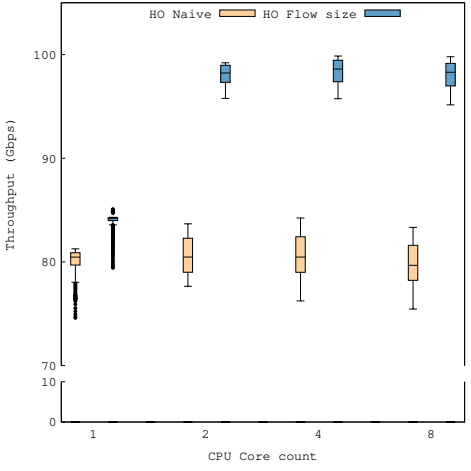


Figure 11: Run 2

Results optimally offloading 16M Hash Table Entries 8M Flows

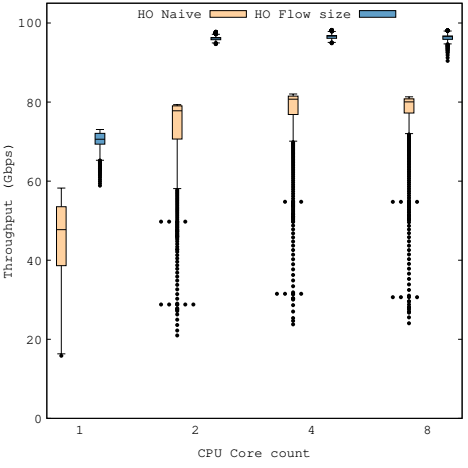


Figure 10: Run 1

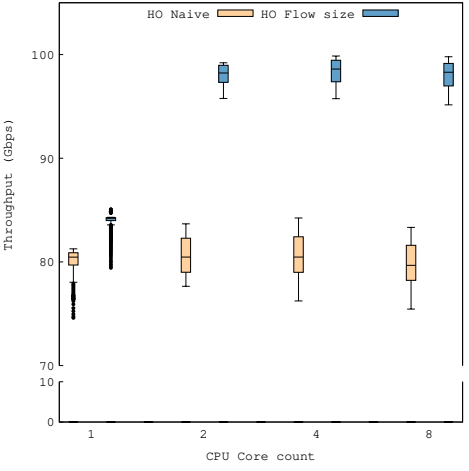


Figure 11: Run 2

Conclusion

Conclusion

- Hardware Offloading allow for higher packet throughput while using the same amount of clock cycles

Conclusion

- Hardware Offloading allow for higher packet throughput while using the same amount of clock cycles
- This comes at the expense of the need to insert the offloaded flows into hardware for classification

Conclusion

- Hardware Offloading allow for higher packet throughput while using the same amount of clock cycles
- This comes at the expense of the need to insert the offloaded flows into hardware for classification
- This cost can be somewhat deterred by being selective with which flows to insert

References

[1] *Anonymized Internet Traces 2018*. https://catalog.caida.org/details/dataset/passive_2018_pcap. Accessed: 2022-1-5.

[2] Tom Barbette, Cyril Soldani, and Laurent Mathy. "Fast userspace packet processing". In: *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. 2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS). Oakland, CA, USA: IEEE, May 2015, pp. 5–16. ISBN: 978-1-4673-6633-5. DOI: 10.1109/ANCS.2015.7110116. URL: http://ieeexplore.ieee.org/document/7110116/ (visited on 11/09/2021).

[3] Cisco. *Cisco Annual Internet Report - Cisco Annual Internet Report (2018–2023) White Paper*. Cisco. URL: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html (visited on 12/21/2021).

[4] Robert Morris et al. "The Click modular router". In: *Proceedings of the seventeenth ACM symposium on Operating systems principles - SOSP '99*. the seventeenth ACM symposium. Charleston, South Carolina, United States: ACM Press, 1999, pp. 217–231. ISBN: 978-1-58113-140-6. DOI: 10.1145/319151.319166. URL: http://portal.acm.org/citation.cfm?doid=319151.319166 (visited on 11/09/2021).

Thanks for listening

Questions?