Dyssect: Dynamic Scaling of Stateful Network Functions

fabricio.carvalho@ufms.br

Carvalho, F. B.¹, Ferreira, R. A.², Cunha, I.³, Vieira, M. A. M.³, Ramanathan, M. K.⁴

UFMT and UFMS¹  UFMS²  UFMG³  Uber Technologies, Inc.⁴
Introduction

Conventional Enterprise Network

Load Balancer Appliance  NAT Appliance  IDS Appliance  Firewall Appliance

More Expensive
Less Manageable
Introduction

Network Function Virtualization

Load Balancer NF  NAT NF  IDS NF  Firewall NF

Commercial off-the-shelf (COTS) Server

Less More Expensive
More Less Manageable
Introduction

- The vast majority of network functions are *stateful* and may require state updates on a *per-packet* basis;

- Concurrent accesses:
  - Locks?
The vast majority of network functions are stateful and may require state updates on a per-packet basis;

Concurrent accesses:
  ○ Locks?
Introduction

The vast majority of network functions are *stateful* and may require state updates on a \textit{per-packet} basis;

- Concurrent accesses:
  - Locks?
Introduction

Queue length

Incoming packets  →  Dispatching core

Queue length

Processing core 0

Processing core 1

Processing core 2
Introduction

Queue length

Incoming packets

Dispatching core

Queue length

Processing core 0

Processing core 1

Processing core 2
Introduction

Queue length

Incoming packets

Dispatching core

Queue length

Processing core 0

Bottleneck

Processing core 1

Processing core 2
Introduction

Queue length

Incoming packets

Dispatching core

Processing core 0

Bottleneck

Queue length

Processing core 1

Processing core 2
Introduction

- Several research proposals use state *sharding* to avoid the use of locks;
Introduction

- Several research proposals use state *sharding* to avoid the use of locks;
Introduction

Several research proposals use state sharding to avoid the use of locks;
Introduction

Several research proposals use state *sharding* to avoid the use of locks;
Introduction

- A recent effort proposes dynamic reassignments of shards to balance the load across cores;
Introduction

- A recent effort proposes dynamic reassignments of shards to balance the load across cores;
Introduction

- One shard might have multiple large-volume flows;
- Systems cannot allocate more cores to handle the load, as the shard is assigned to a single core.
Introduction

- We evaluate the performance impact of the number of shards in CPU metrics:
  - The throughput drops up to 43.3% comparing 1 vs. 128 shards;

*IPC = Instructions per Cycle
Contributions

Dyssect:

- steers packets to cores;
- moves shards between cores;
- disaggregates of state from network functions;
- avoids frequent shard transfers;
- uses optimization models.
Dyssect
State Management

Packet → H P
Dyssect
State Management

Packet → H P

Hash (H) → Flow Key

→ ... → State Table
Dyssect
State Management

Packet → H P → Insert metadata → M H P

Hash (H) → Flow Key → State Table

Flow Entry
Dyssect
State Management

Packet → H P → Insert metadata → M H P → NF₀

Hash (H) → Flow Key → State Table

Flow Entry → GetState

State Table
Dyssect
State Management

Packet $\rightarrow$ H P $\rightarrow$ Insert metadata $\rightarrow$ M H P $\rightarrow$ NF$_0$ State $\rightarrow$ NF$_1$ State

State Table

Hash (H) $\rightarrow$ Flow Key $\rightarrow$ Flow Entry

GetState

GetState
Dyssect
State Management

Packet → H P → Insert metadata → M H P → NF₀ State → NF₁ State → NF₂ State → Packet

Hash (H) → Flow Key → State Table

Flow Entry → GetState → GetState → GetState
Dyssect

Flow Assignment

- Controller updates RSS table, migrates shards, and defines a subset of flows in a shard to forward to an offloading core;
- Dyssect splits cores into working or offloading cores.
Dyssect
Flow Assignment

Time $T_2$

Incoming packets

Shard 0
Shard 1

Working core 0

Offloading core 0

Working core 1
Dyssect
Correctness Analysis

- **Deadlock freedom**
  - Controller can disable packet processing;
  - Working cores enqueue packet into queues;
  - Offloading cores never blocks during scaling operations.
  - **If there exists an incoming packet p, at a certain moment, p turns into an outgoing packet.**

- **Packet ordering**
  - Controller can reassign shards, offloading cores, or change offload ratio;
  - Auxiliary queues are swapped by the Controller;
  - *Scaling algorithms*;
  - For any pair of packets from the same flow, the first packet of the pair is always processed first.

*Check the formal proofs in our paper.*
Dyssect
Flow Assignment Optimization

Optimization models:

- Long-timescale optimization:
  - minimizes the number of active working and offloading cores.
Dyssect
Flow Assignment Optimization

Optimization models:

- Long-timescale optimization:
  - minimizes the number of active working and offloading cores.
- Short-timescale optimization:
  - minimizes the number of shard migrations and offloading core reassociations.

*Check both optimization models in our paper.*
Dyssect
Flow Assignment Optimization

Optimization models:

- **Long-timescale optimization:**
  - minimizes the number of active working and offloading cores.
- **Short-timescale optimization:**
  - minimizes the number of shard migrations and offloading core reassociations.
- **Constraints:**
  - SLO, core utilization, shard ratio, working and offloading cores relationship.

*Check both optimization models in our paper.*
Evaluation

For evaluation, we use three use cases:

- Use Case I: traffic class prioritization;
Evaluation

For evaluation, we use three use cases:

- Use Case I: traffic class prioritization;
- Use Case II: alternate optimization targets;
Evaluation

For evaluation, we use three use cases:

- Use Case I: traffic class prioritization;
- Use Case II: alternate optimization targets;
- Use Case III: SmartNIC offloading.
Evaluation

Use Case I

- Real trace;
- High and low priority flows;
- Scaling traffic to simulate throughputs from ~2.5 to ~22 Gbps;
- Network functions: NAT and IDS.
Evaluation

Use Case 1

Throughput

Time (s)

Number of Cores

Time (s)
Evaluation

Use Case I

![Graph showing Latency CDF and Shard Migrations](image)

- **Latency CDF**: Comparison of Dyssect (high-priority) and RSS++ (high-priority) with Dyssect (low-priority) and RSS++ (low-priority).
- **Shard Migrations**: Comparison of Dyssect and RSS++.
Evaluation

Use Case II

- We explore Dyssect using a different optimization model:
  - Load balance optimization model (below);
- This model minimizes the quadratic difference between a target value $T$ and the utilization of working and offloading cores.

$$\text{minimize } \sum_{c \in C} (u_c^w - T)^2 + \sum_{k \in C} (u_k^o - T)^2 + \alpha \text{(Eq. 16)}, \quad (21)$$

subject to Equations 2 – 11 and Equations 19 – 20

Check the equation definitions in our paper.
Evaluation

Use Case II

- Synthetic trace (Zipf distribution);
- Load balance optimization model;
- Network functions: NAT and IDS.
Evaluation

Use Case III

- We offload the lookup function to a SmartNIC;
- SmartNIC performs the lookup and inserts the address into the packet metadata;
- Working cores skip the lookup if the metadata already contains an address.
Evaluation
Use Case III

- We use Netronome NFP-4000 2x40 Gbps;
- Synthetic trace (Zipf distribution with $\alpha = 1.1$);
- Measurements of a single core.
Conclusion

- Sharding impacts on the performance of stateful network functions;
- Dyssect disaggregates states from network functions;
- Dyssect employs optimization models;
- Dyssect increases throughput up to 19% and reduces tail latency up to 32% when compared with other load-balancing proposals.

https://github.com/dyssect/dyssect
Thank you!

fabricio.carvalho@ufms.br