Cluster Based Distributed State Management of Network Function

Supervised By :
Dr. Rezwana Reaz

Presented By :
Fardin Hossain (201705038)
Nishat Farhana Purbasha (201705067)
Saif Ahmed Khan (201705110)

Outline

- Preliminaries
- Motivation
- Related Works
- Problem Definition
- Methodology
- Experiments and Results
- Contributions and Future Works

Preliminaries

Network Function

Functionalities of Network Functions:

- Routing
- Filtering
- Inspecting
- Security Features
- Optimizing Network Performance

Examples of Network Functions:

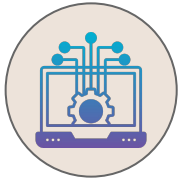
- Network Address Translation - NAT
- Firewall
- Load Balancer
- Intrusion Detection System - IDS

Issues with Hardware Network Function

- Higher Cost
- Limited Scalability
- Single Point of Failure
- Maintenance Difficulty

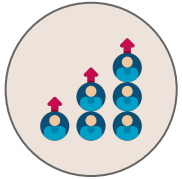
What is the solution?

Network Function Virtualization

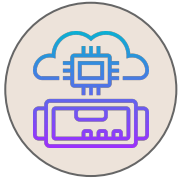


Implement software-based network functions, replacing hardware-based ones.

Use the same hardware for different purposes.



New instances of NF can be created without any new hardware cost.



Decouple the virtualized network functions from underlying hardware.

Software Defined Networking (SDN)

Uses software-based controllers or application programming interfaces (APIs) to communicate with underlying hardware infrastructure.

Controls the routing of data packets through a software-based and centralized server.

Better security in many ways, thanks to greater visibility and the ability to define secure pathways.

Definition of Flow and States

Source IP	Source Port	Destination IP	Destination Port	Protocol
192.168.176.7	5000	173.16.1.2	8000	UDP
192.168.176.8	5002	173.16.1.3	8000	UDP
192.168.176.9	5100	173.16.1.3	7521	UDP

Definition of Flow and States

Source IP	Source Port	Destination IP	Destination Port	Protocol
192.168.176.7	5000	173.16.1.2	8000	UDP
192.168.176.8	5002	173.16.1.3	8000	UDP
192.168.176.9	5100	173.16.1.3	7521	UDP

Flow : A 5-tuple structure describing a connection

Definition of Flow and States

Source IP	Source Port	Destination IP	Destination Port	Protocol
192.168.176.7	5000	173.16.1.2	8000	UDP
192.168.176.8	5002	173.16.1.3	8000	UDP
192.168.176.9	5100	173.16.1.3	7521	UDP

Flow : A 5-tuple structure describing a connection

Definition of Flow and States

Source IP	Source Port	Destination IP	Destination Port	Protocol
192.168.176.7	5000	173.16.1.2	8000	UDP
192.168.176.8	5002	173.16.1.3	8000	UDP
192.168.176.9	5100	173.16.1.3	7521	UDP

Flow : A 5-tuple structure describing a connection

States :

Definition of Flow and States

Source IP	Source Port	Destination IP	Destination Port	Protocol
192.168.176.7	5000	173.16.1.2	8000	UDP
192.168.176.8	5002	173.16.1.3	8000	UDP
192.168.176.9	5100	173.16.1.3	7521	UDP

Flow : A 5-tuple structure describing a connection

States :

1. **Per-flow States** : States associated with a particular flow

Definition of Flow and States

Source IP	Source Port	Destination IP	Destination Port	Protocol
192.168.176.7	5000	173.16.1.2	8000	UDP
192.168.176.8	5002	173.16.1.3	8000	UDP
192.168.176.9	5100	173.16.1.3	7521	UDP

Flow : A 5-tuple structure describing a connection

States :

1. **Per-flow States** : States associated with a particular flow
2. **Global States** : States associated with multiple flows

State Management

State Redistribution

Global State Update

Consistency Requirements

State Management

State Redistribution

Global State Update

Consistency Requirements

State Redistribution

State Migration ✓	Migration Avoidance

• State transferred between NFs
• **Challenge:**
  • Stall during migration
  • Handle packet drop
  • Reduce migration overhead
	• State accessed remotely
	• **Challenge:**
	  • Remote access latency
	  • Central trusted storage
	  • Prone to single point failure

State Management

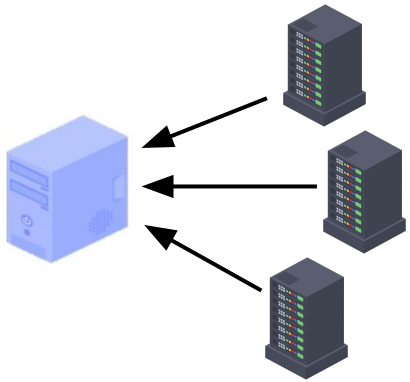
State Redistribution

Global State Update

Consistency Requirements

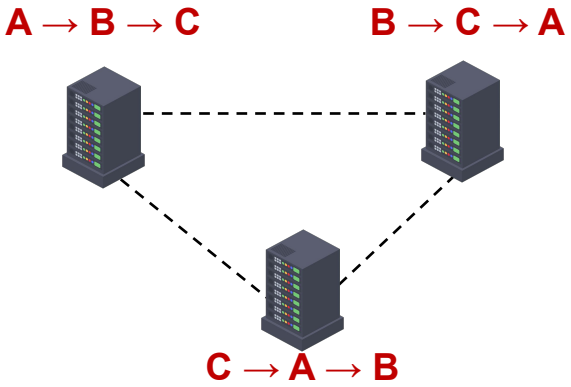
Global State Update

Centralized



- **Challenge:** Central dependency

Distributed ✓



- **Challenge:** State update order

State Management

State Redistribution

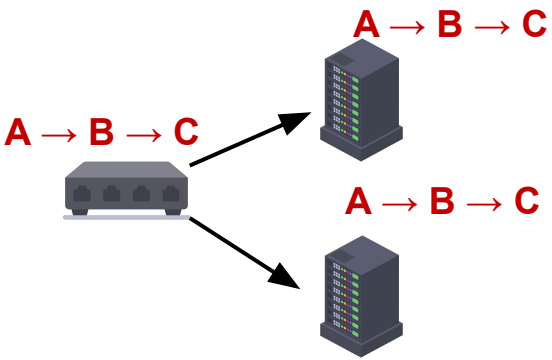
Global State Update

Consistency Requirements

Consistency Requirements

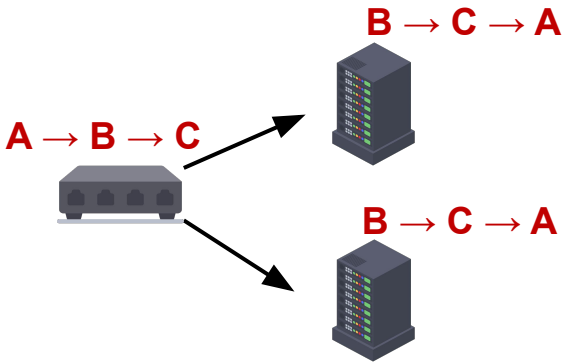
Strict

- Update order across NFs and switch is consistent



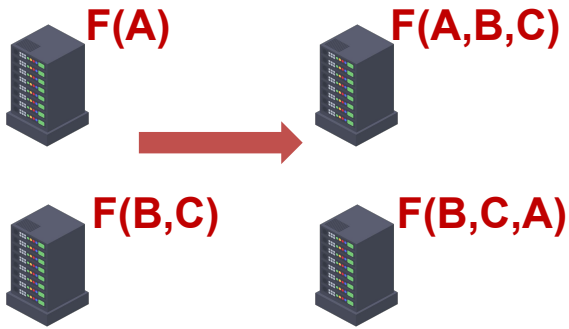
Strong ✓

- Update order across NFs is consistent



Eventual ✓

- States are commutative. F(A,B,C) = F(B,C,A)



Motivation

Motivation

Faster Processing

Reduced Overhead

Motivation

Faster Processing

Reduced Overhead

Faster Processing

What happens in a traditional NF?

Faster Processing

What happens in a traditional NF?

- Every NF performs packet processing of **all flows** serially in a **single thread**.
- Multiple flows cannot be processed in parallel.
- Time consuming.

Faster Processing

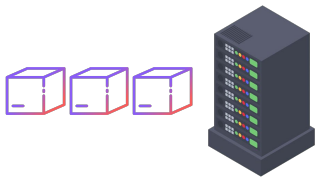
What happens in a traditional NF?

- Every NF performs packet processing of **all flows** serially in a **single thread**.
- Multiple flows cannot be processed in parallel.
- Time consuming.

Faster Processing

What happens in a traditional NF?

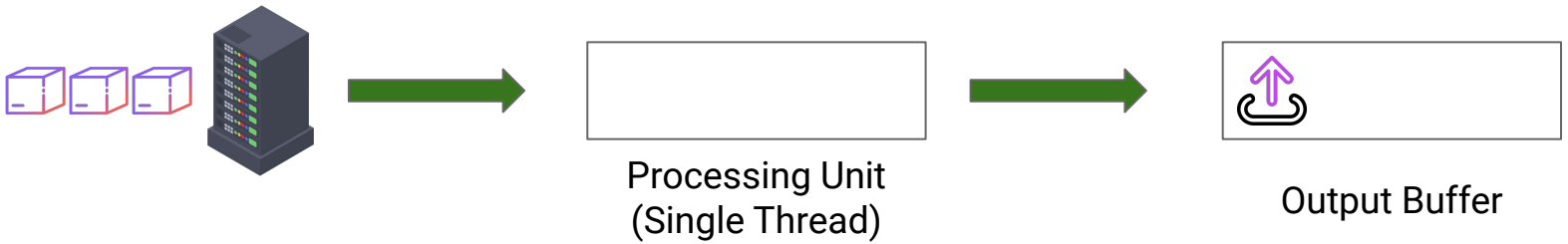
- Every NF performs packet processing of **all flows** serially in a **single thread**.
- Multiple flows cannot be processed in parallel.
- Time consuming.



Faster Processing

What happens in a traditional NF?

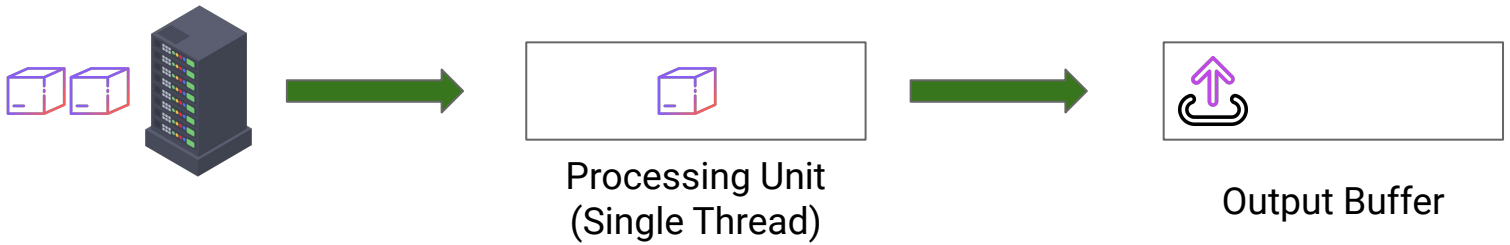
- Every NF performs packet processing of **all flows** serially in a **single thread**.
- Multiple flows cannot be processed in parallel.
- Time consuming.



Faster Processing

What happens in a traditional NF?

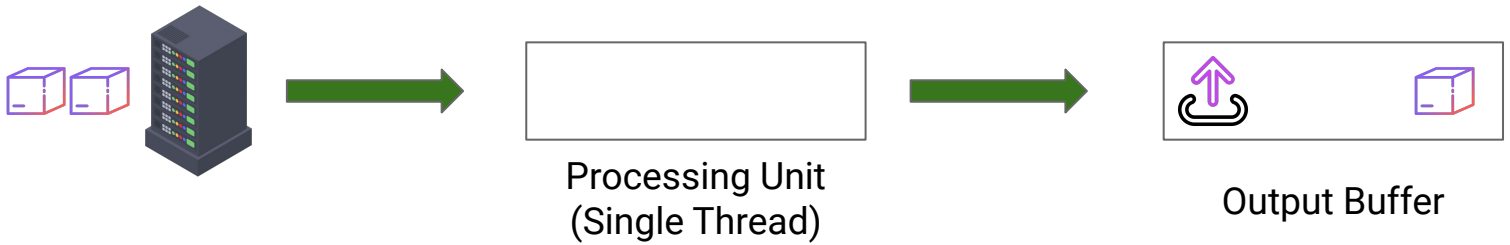
- Every NF performs packet processing of **all flows** serially in a **single thread**.
- Multiple flows cannot be processed in parallel.
- Time consuming.



Faster Processing

What happens in a traditional NF?

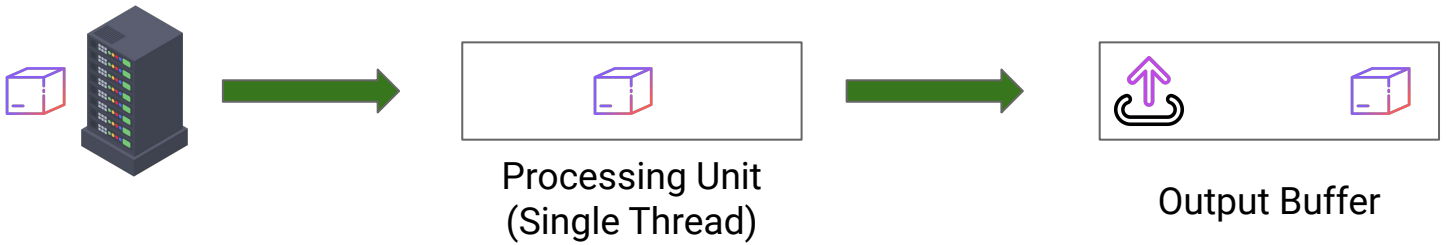
- Every NF performs packet processing of **all flows** serially in a **single thread**.
- Multiple flows cannot be processed in parallel.
- Time consuming.



Faster Processing

What happens in a traditional NF?

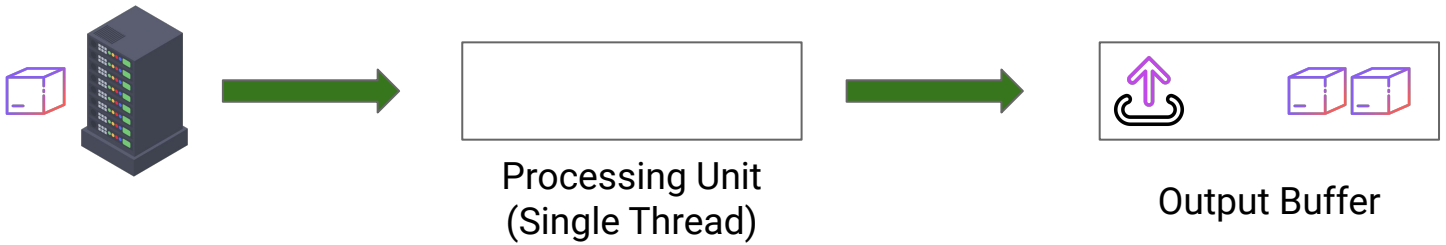
- Every NF performs packet processing of **all flows** serially in a **single thread**.
- Multiple flows cannot be processed in parallel.
- Time consuming.



Faster Processing

What happens in a traditional NF?

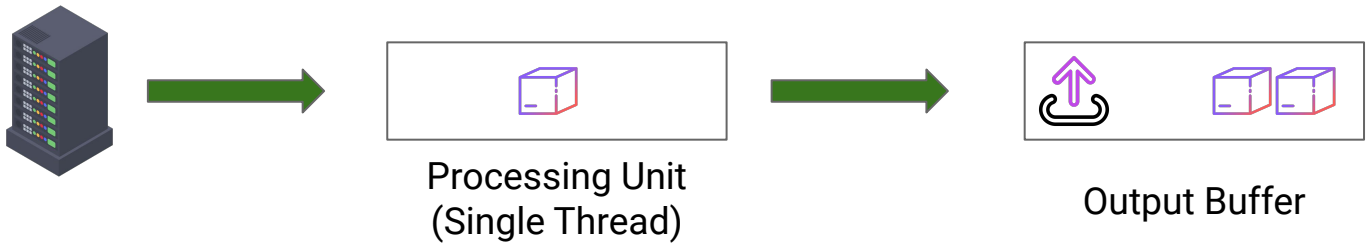
- Every NF performs packet processing of **all flows** serially in a **single thread**.
- Multiple flows cannot be processed in parallel.
- Time consuming.



Faster Processing

What happens in a traditional NF?

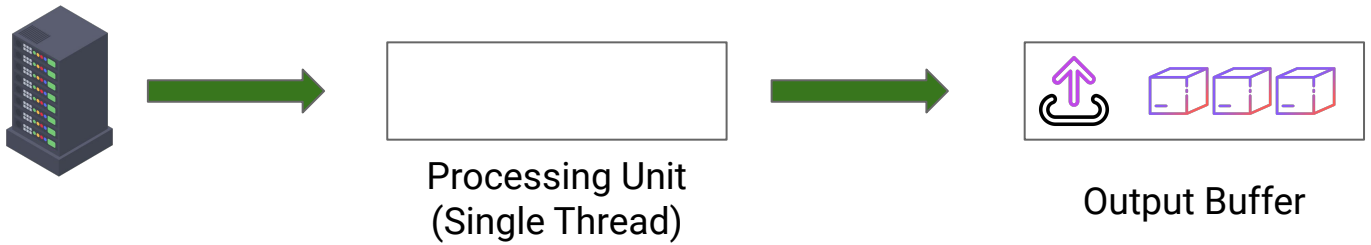
- Every NF performs packet processing of **all flows** serially in a **single thread**.
- Multiple flows cannot be processed in parallel.
- Time consuming.



Faster Processing

What happens in a traditional NF?

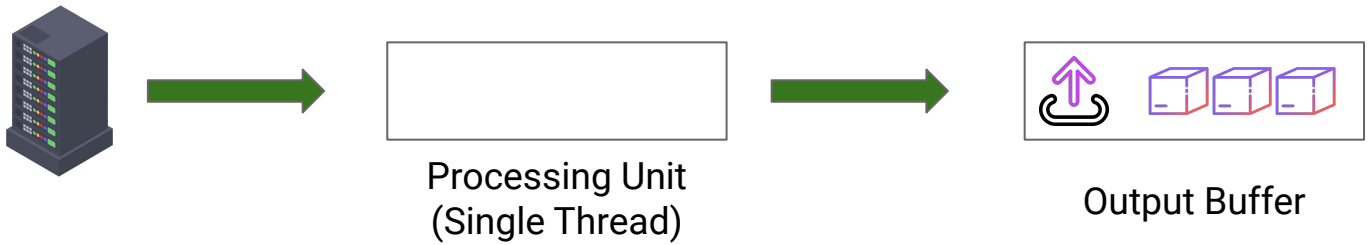
- Every NF performs packet processing of **all flows** serially in a **single thread**.
- Multiple flows cannot be processed in parallel.
- Time consuming.



Faster Processing

What happens in a traditional NF?

- Every NF performs packet processing of **all flows** serially in a **single thread**.
- Multiple flows cannot be processed in parallel.
- Time consuming.



How to ensure faster processing?

Multithreading

Multithreading

- Allows multiple flows to be processed in parallel in different threads.

Multithreading

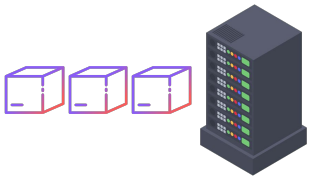
- Allows multiple flows to be processed in parallel in different threads.
- Faster processing and less time to fill output buffer.

Multithreading

- Allows multiple flows to be processed in parallel in different threads.
- Faster processing and less time to fill output buffer.

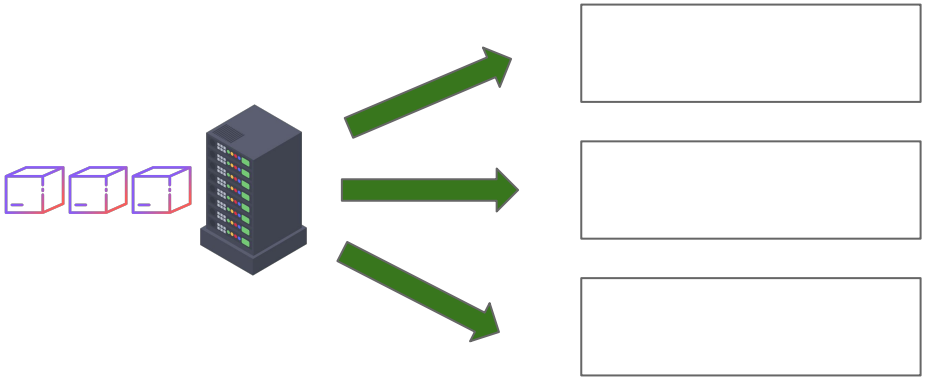
Multithreading

- Allows multiple flows to be processed in parallel in different threads.
- Faster processing and less time to fill output buffer.



Multithreading

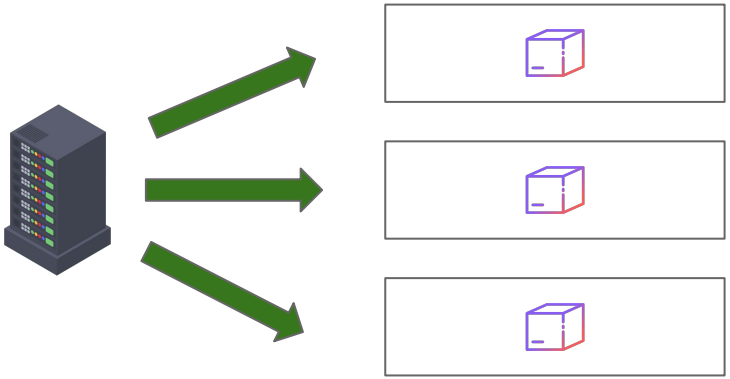
- Allows multiple flows to be processed in parallel in different threads.
- Faster processing and less time to fill output buffer.



Processing Unit
(Single Thread)

Multithreading

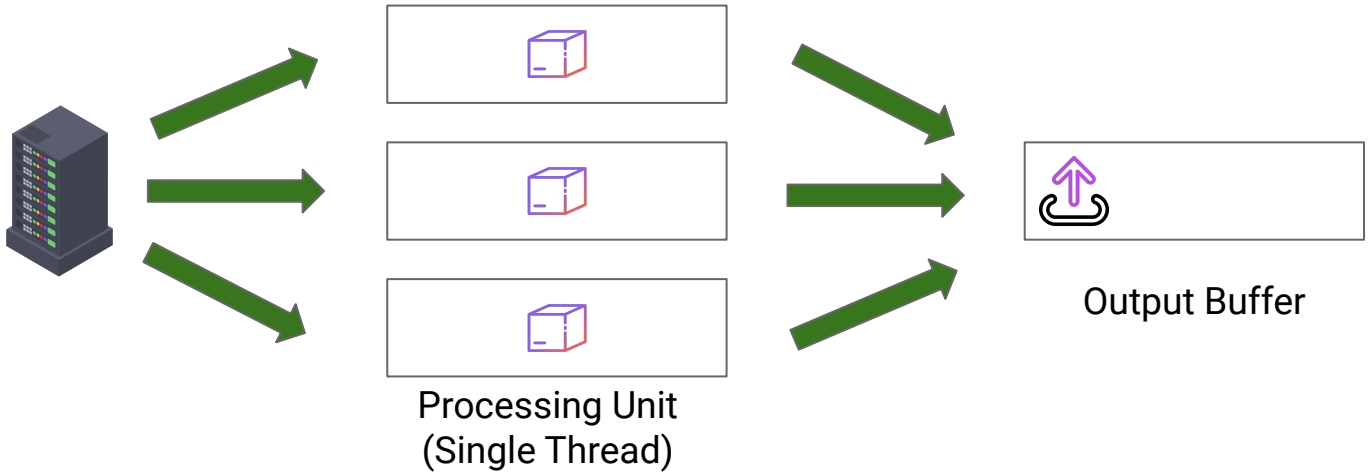
- Allows multiple flows to be processed in parallel in different threads.
- Faster processing and less time to fill output buffer.



Processing Unit
(Single Thread)

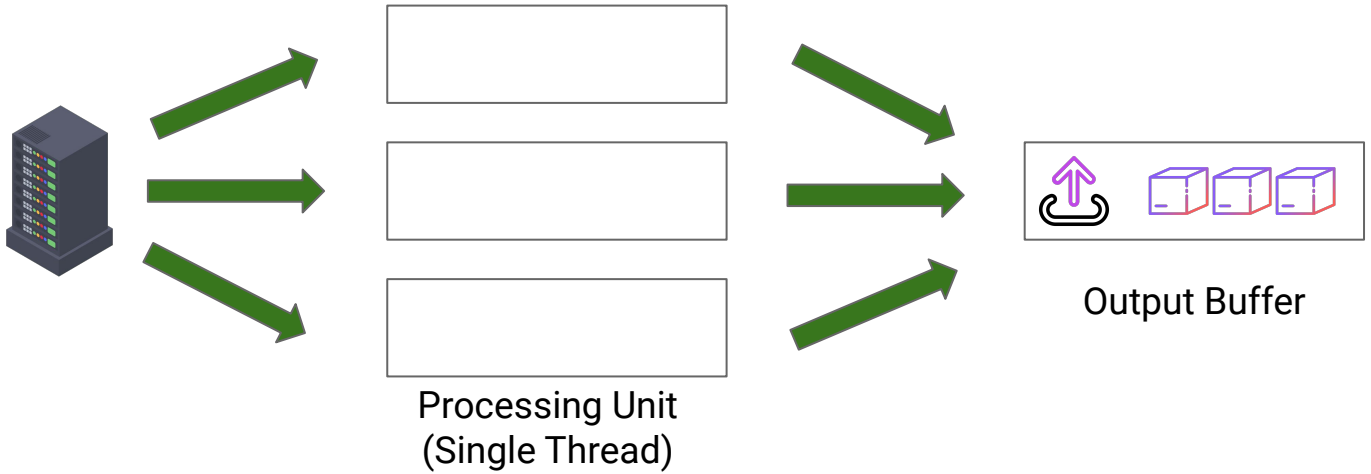
Multithreading

- Allows multiple flows to be processed in parallel in different threads.
- Faster processing and less time to fill output buffer.



Multithreading

- Allows multiple flows to be processed in parallel in different threads.
- Faster processing and less time to fill output buffer.



Motivation

Faster Processing

Reduced Overhead

Reduced Overhead

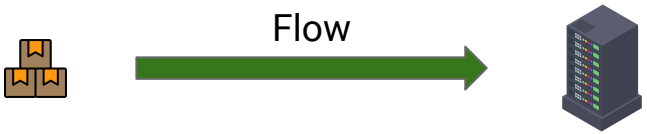
- State is shared only while elastic scaling.

Reduced Overhead

- State is shared only while elastic scaling.
- Need to share all states at once.

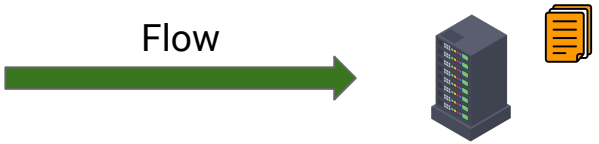
Reduced Overhead

- State is shared only while elastic scaling.
- Need to share all states at once.



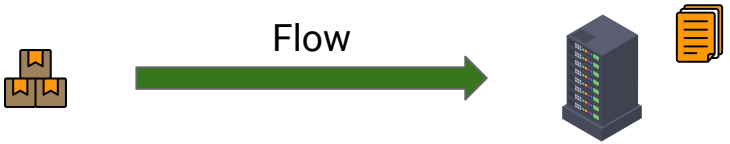
Reduced Overhead

- State is shared only while elastic scaling.
- Need to share all states at once.



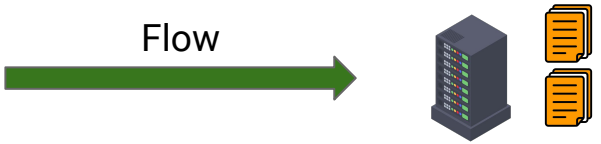
Reduced Overhead

- State is shared only while elastic scaling.
- Need to share all states at once.



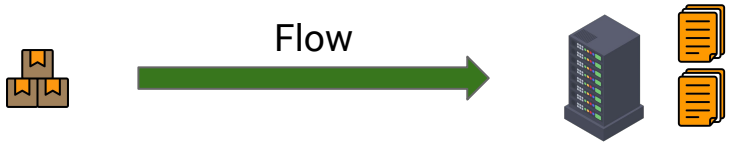
Reduced Overhead

- State is shared only while elastic scaling.
- Need to share all states at once.



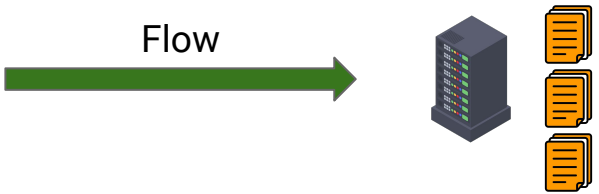
Reduced Overhead

- State is shared only while elastic scaling.
- Need to share all states at once.



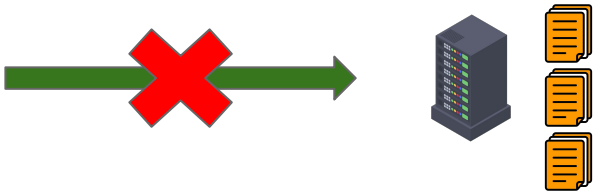
Reduced Overhead

- State is shared only while elastic scaling.
- Need to share all states at once.



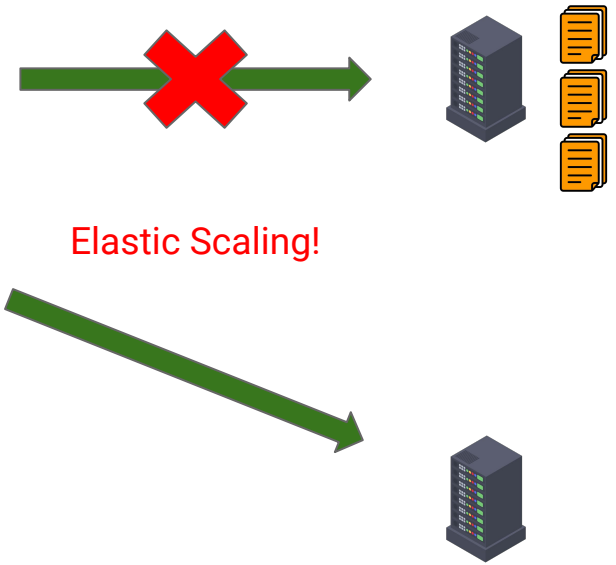
Reduced Overhead

- State is shared only while elastic scaling.
- Need to share all states at once.



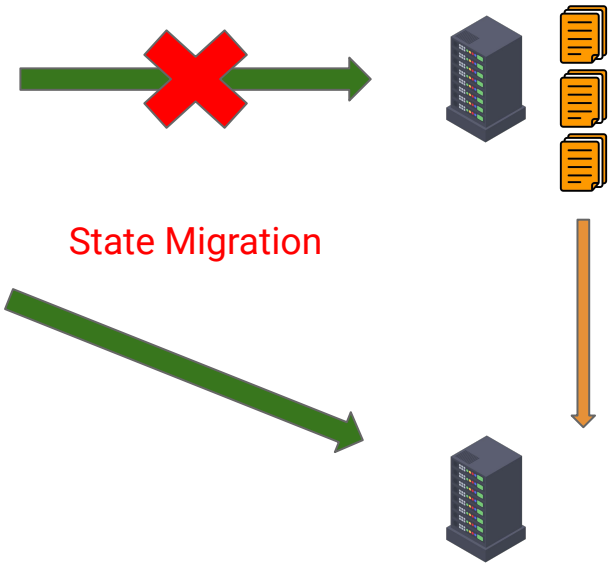
Reduced Overhead

- State is shared only while elastic scaling.
- Need to share all states at once.



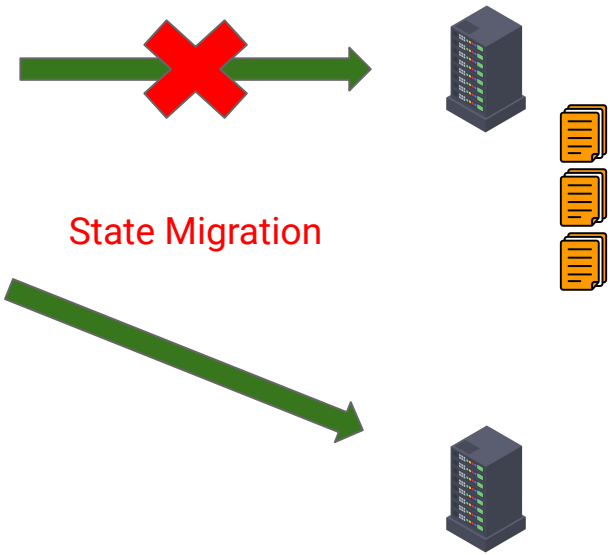
Reduced Overhead

- State is shared only while elastic scaling.
- Need to share all states at once.



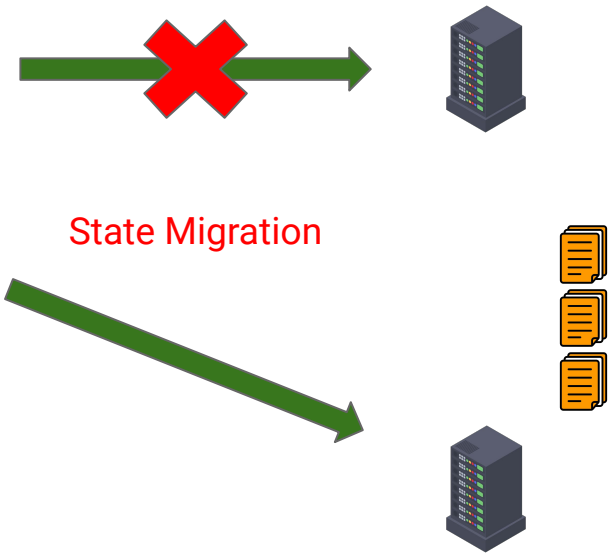
Reduced Overhead

- State is shared only while elastic scaling.
- Need to share all states at once.



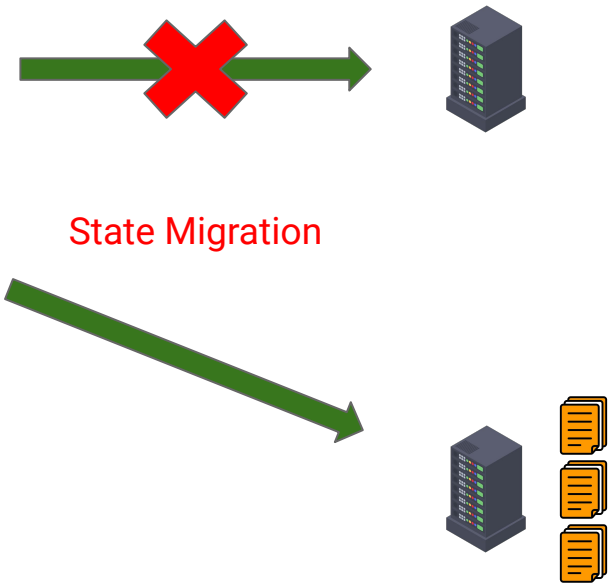
Reduced Overhead

- State is shared only while elastic scaling.
- Need to share all states at once.



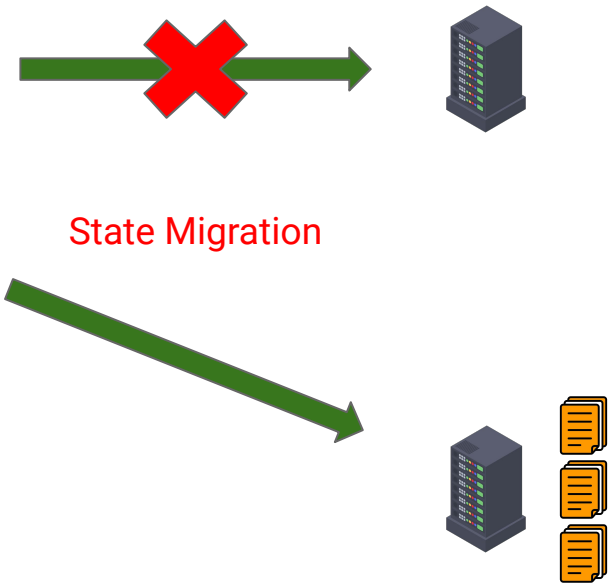
Reduced Overhead

- State is shared only while elastic scaling.
- Need to share all states at once.



Reduced Overhead

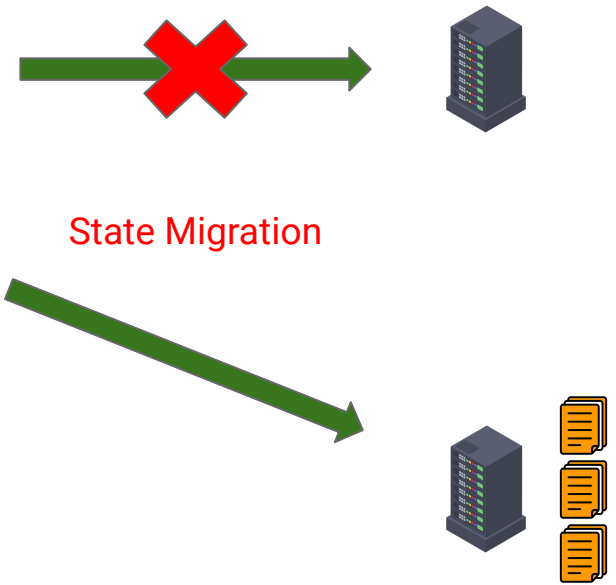
- State is shared only while elastic scaling.
- Need to share all states at once.
- High overhead during state migration.



Reduced Overhead

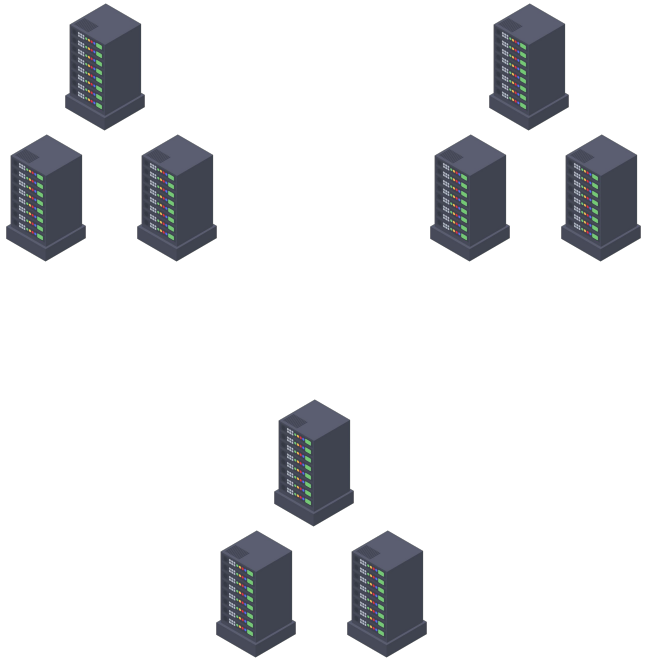
- State is shared only while elastic scaling.
- Need to share all states at once.
- High overhead during state migration.

What could be a better approach?



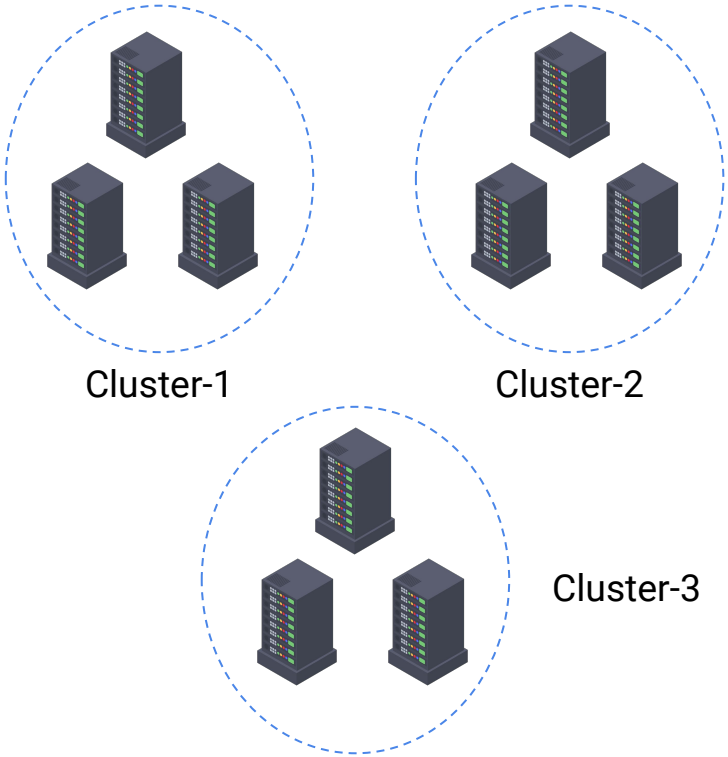
Clustering

- NFs are grouped in clusters.



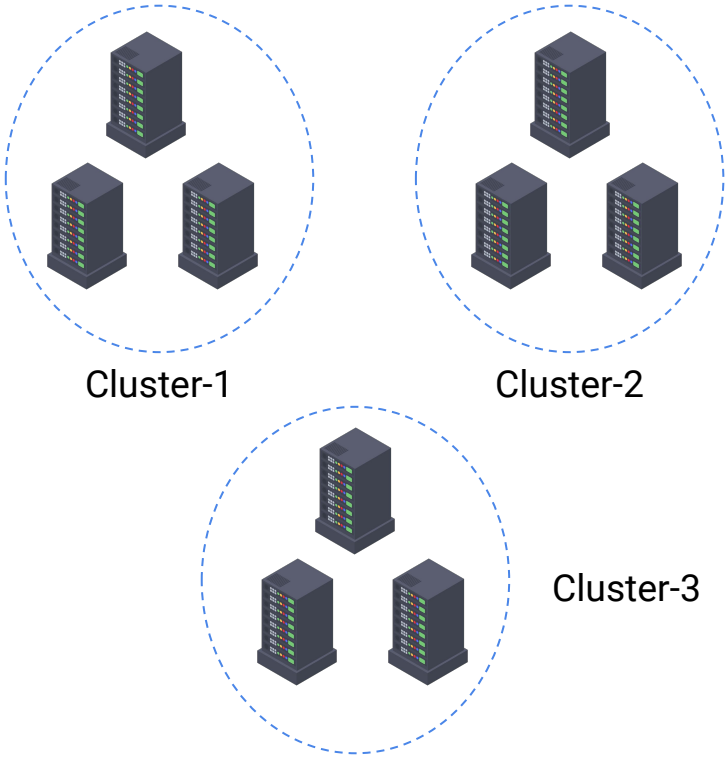
Clustering

- NFs are grouped in clusters.

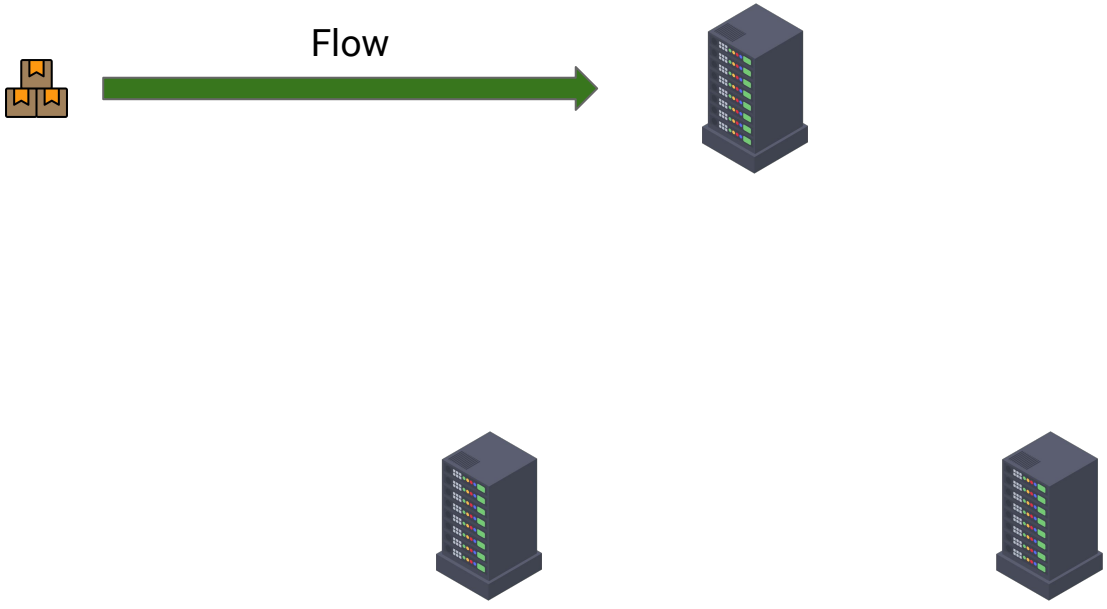


Clustering

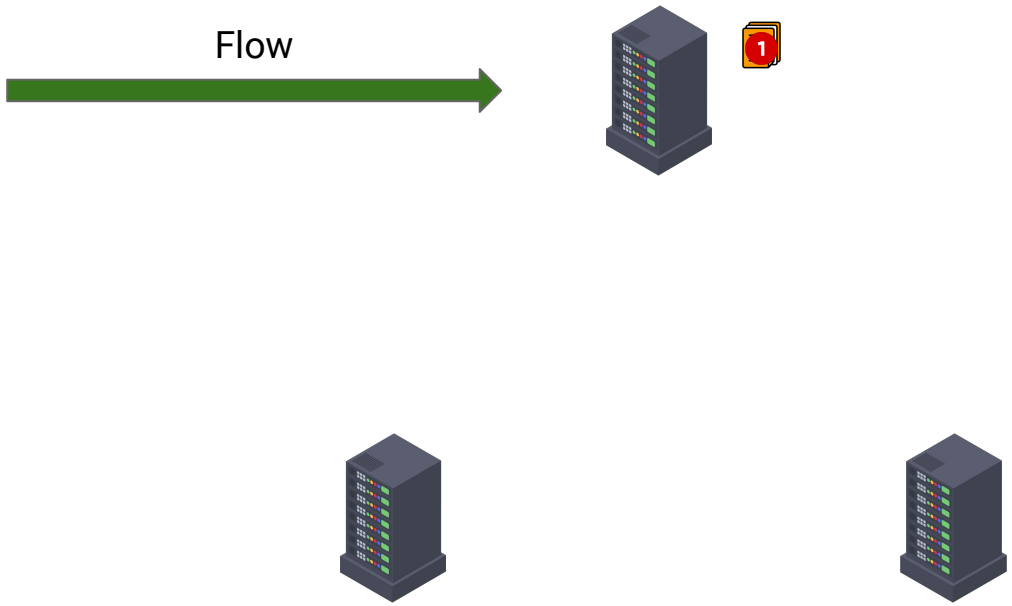
- NFs are grouped in clusters.
- NFs within the same cluster share information.
- State information is updated after each batch processing.



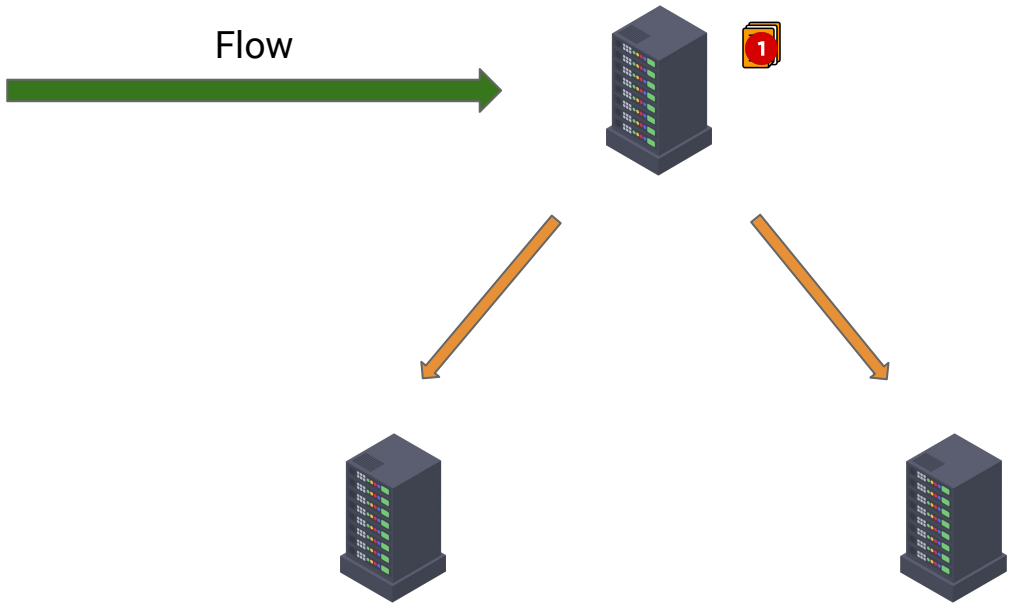
State information update in Clustering



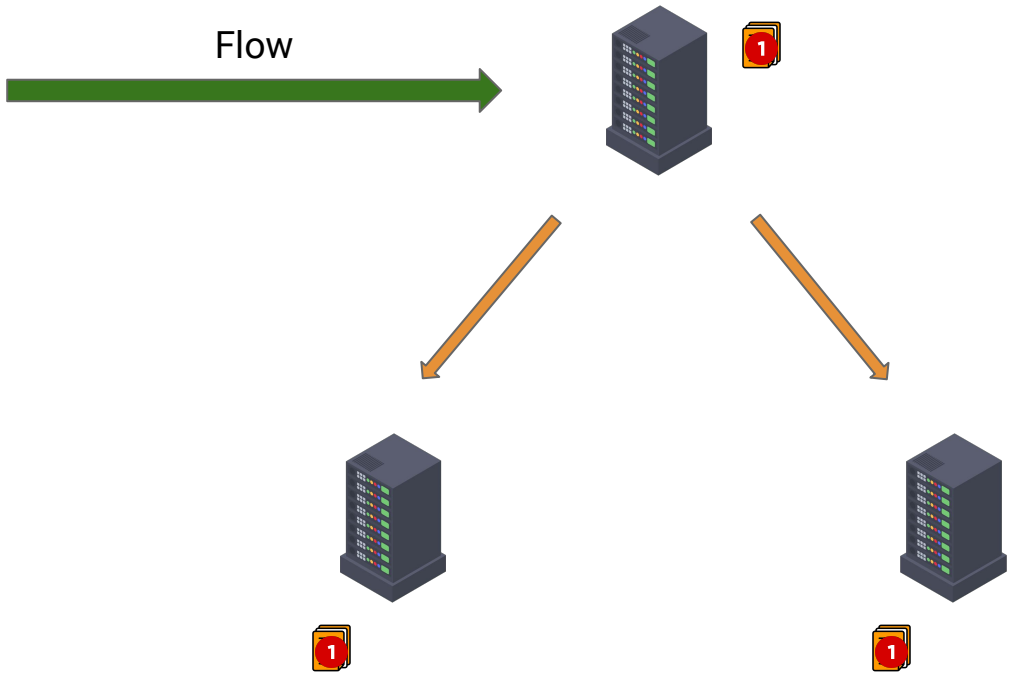
State information update in Clustering



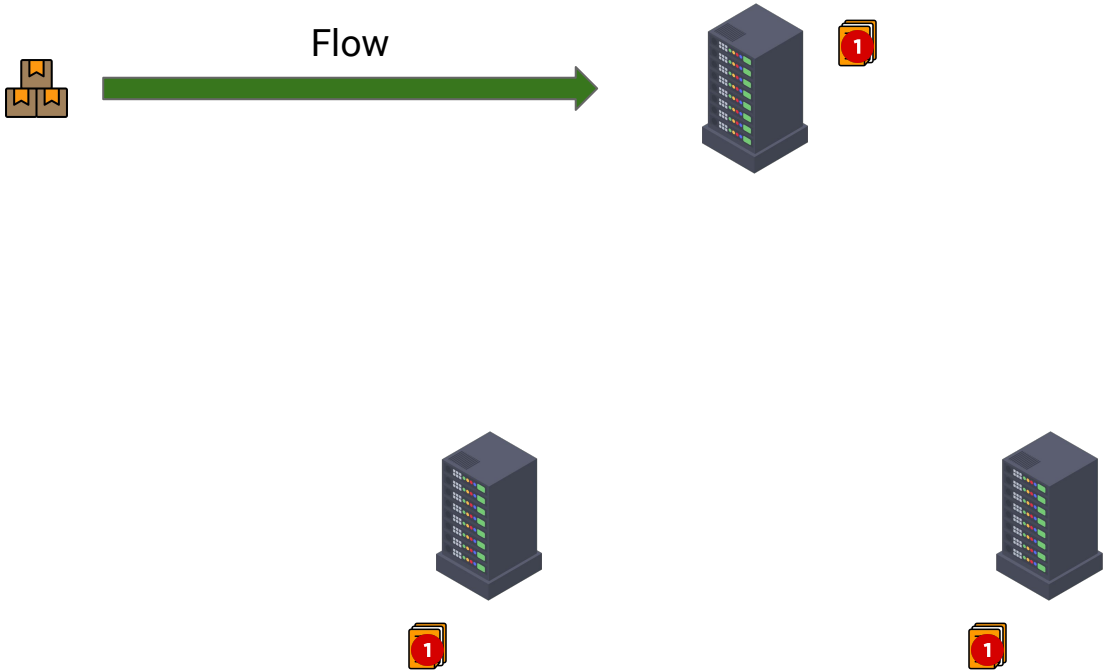
State information update in Clustering



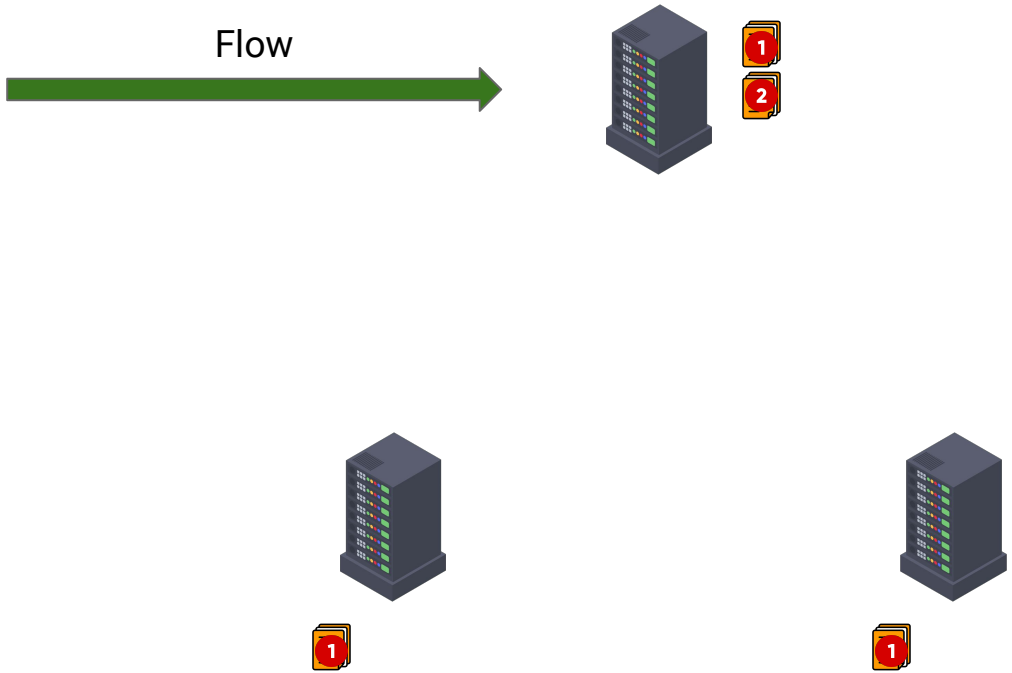
State information update in Clustering



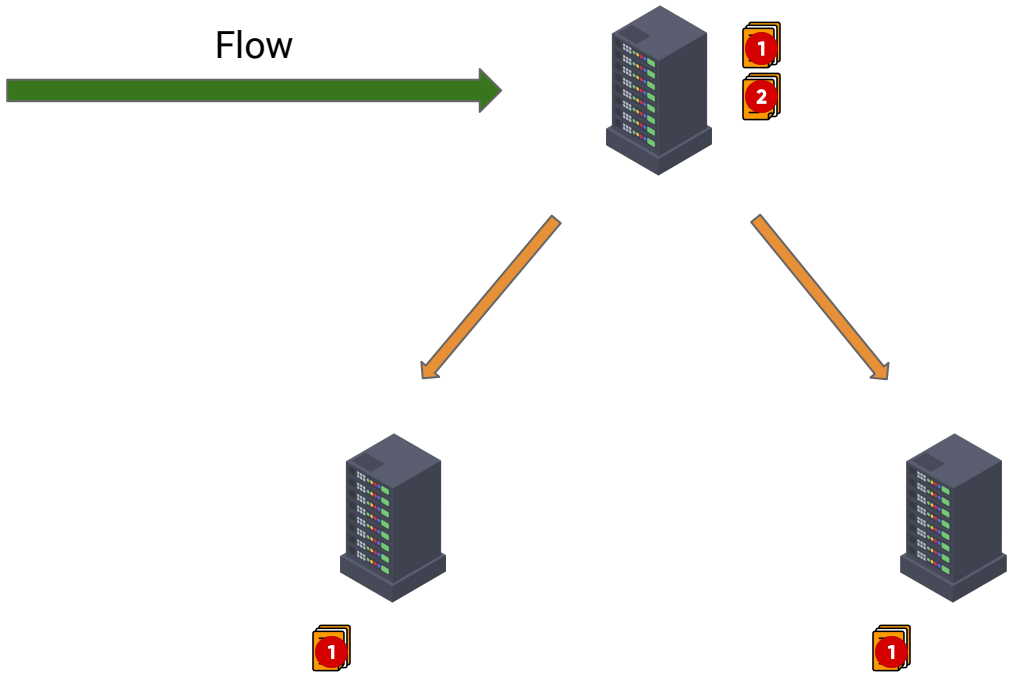
State information update in Clustering



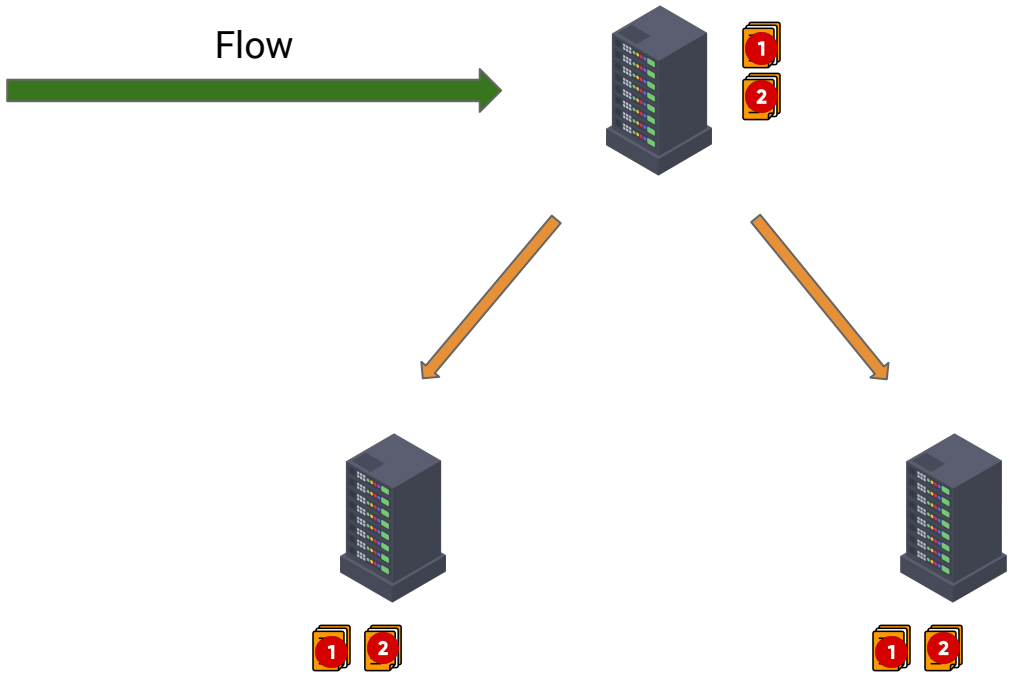
State information update in Clustering



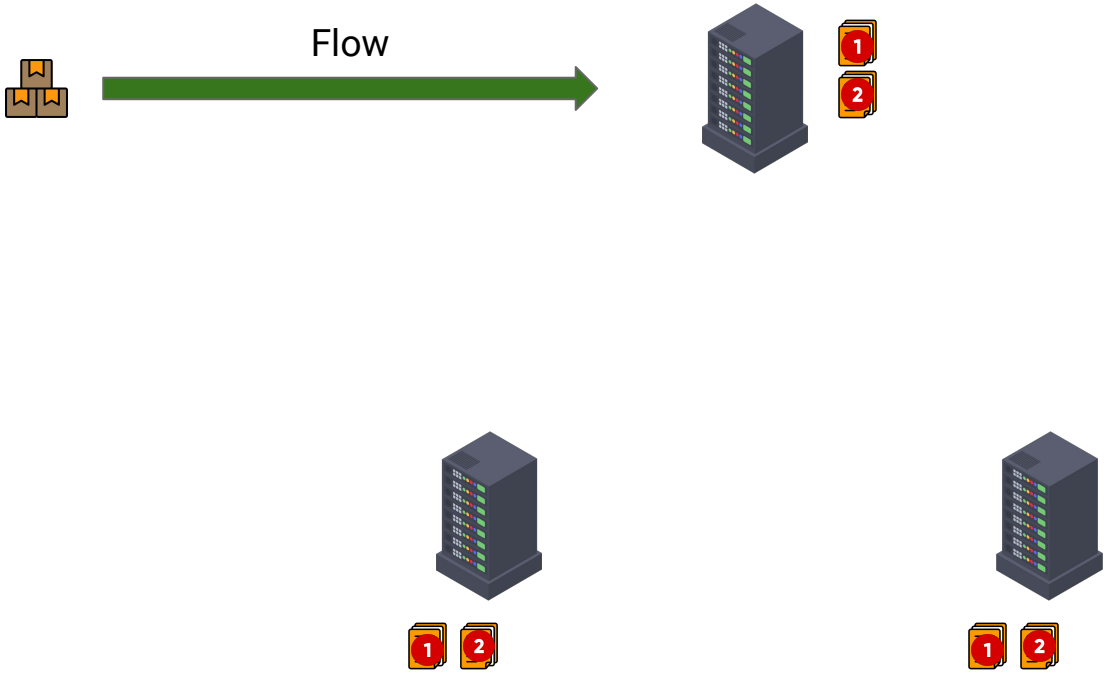
State information update in Clustering



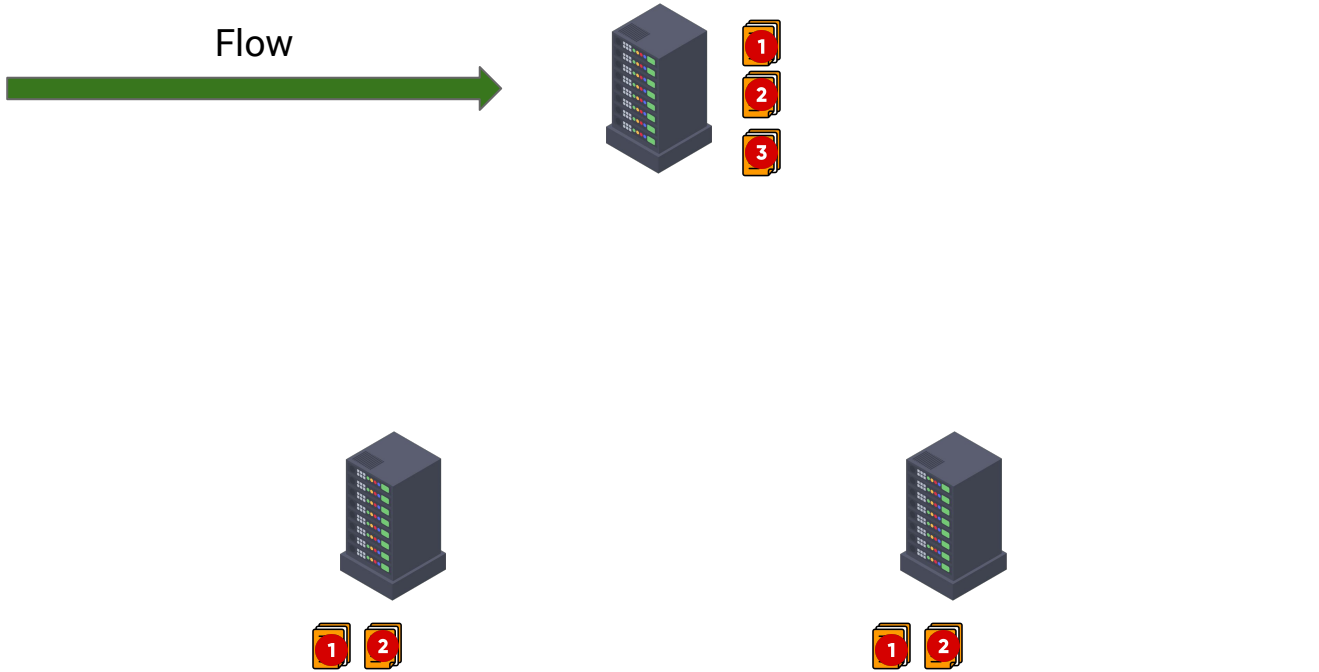
State information update in Clustering



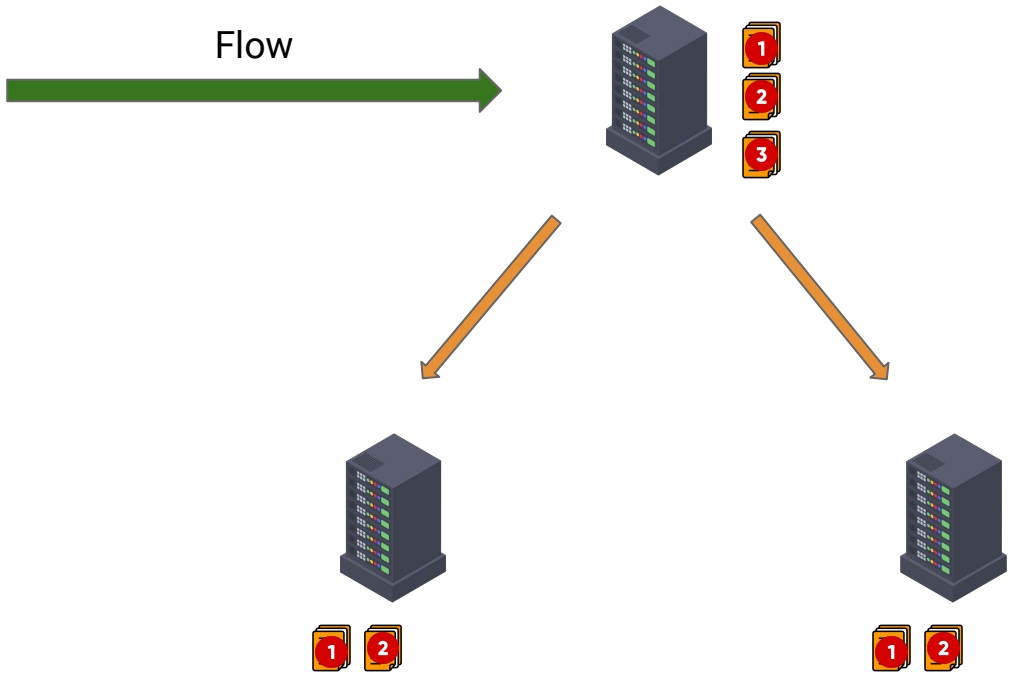
State information update in Clustering



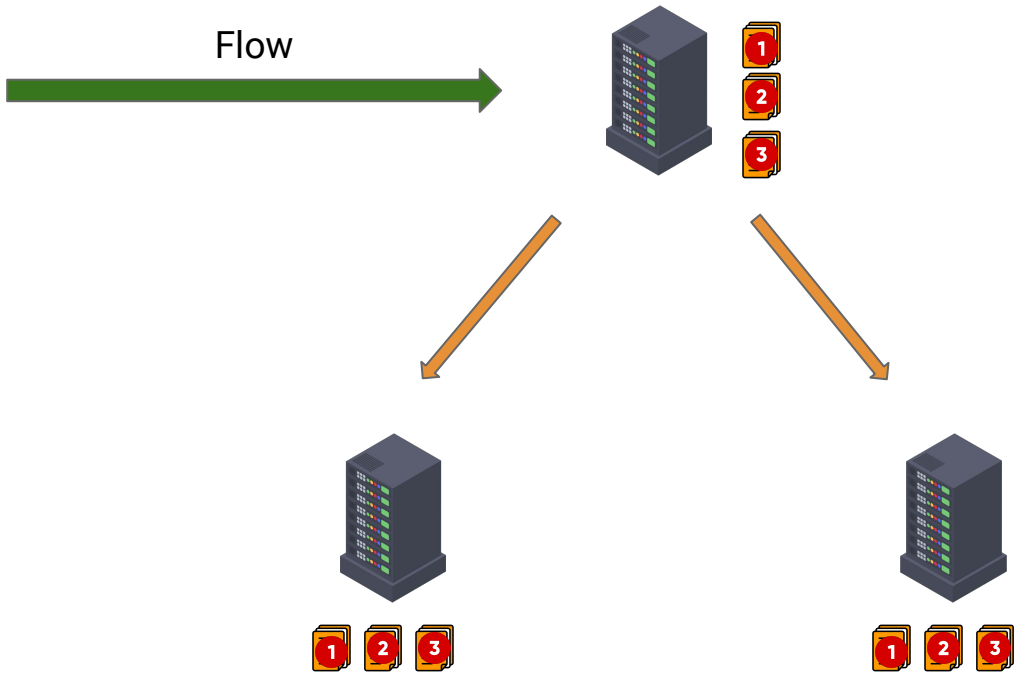
State information update in Clustering



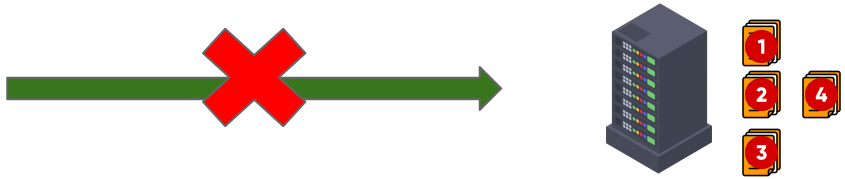
State information update in Clustering



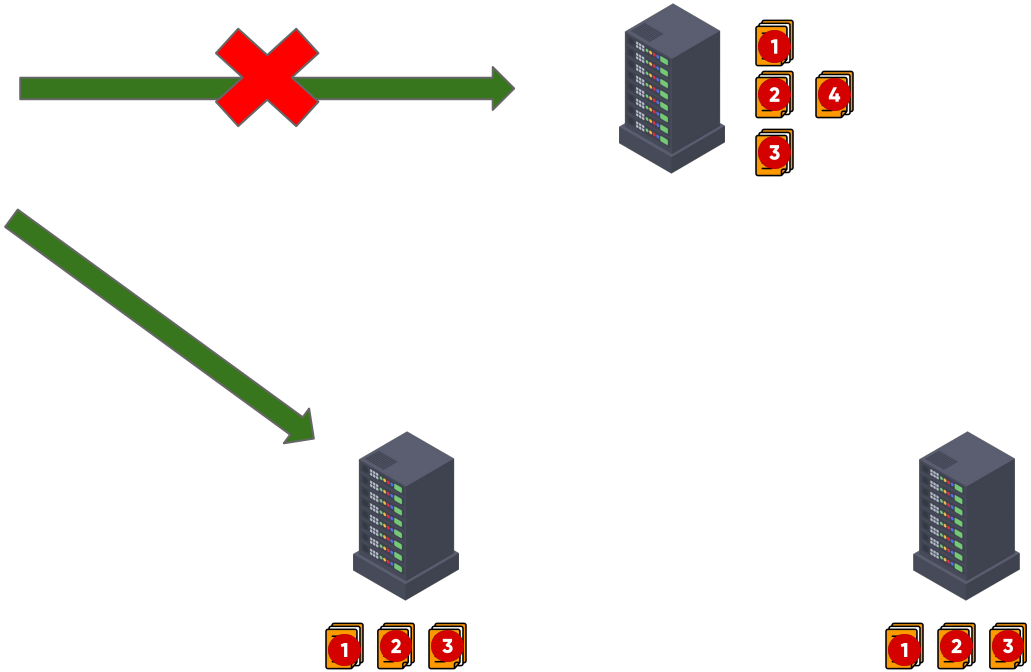
State information update in Clustering



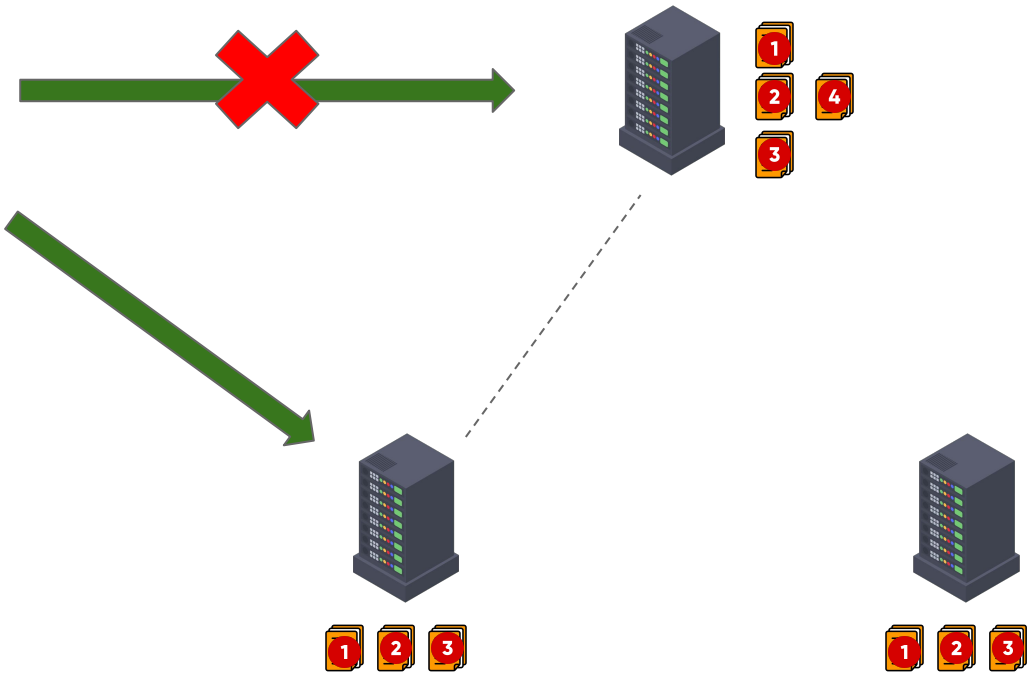
What happens when overloading or NF failure arises?



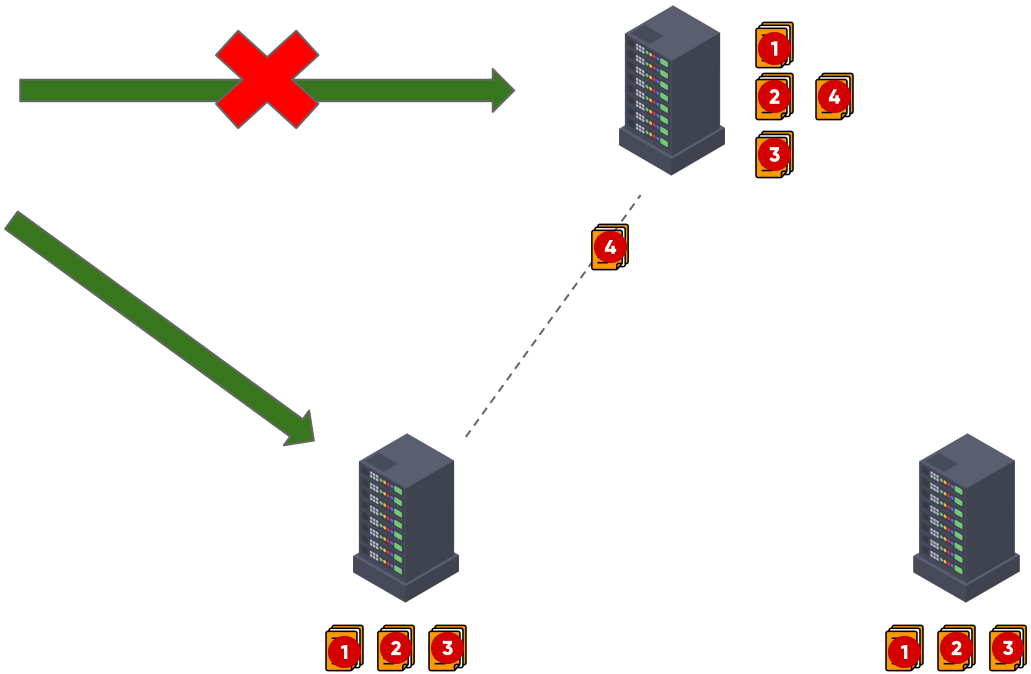
What happens when overloading or NF failure arises?



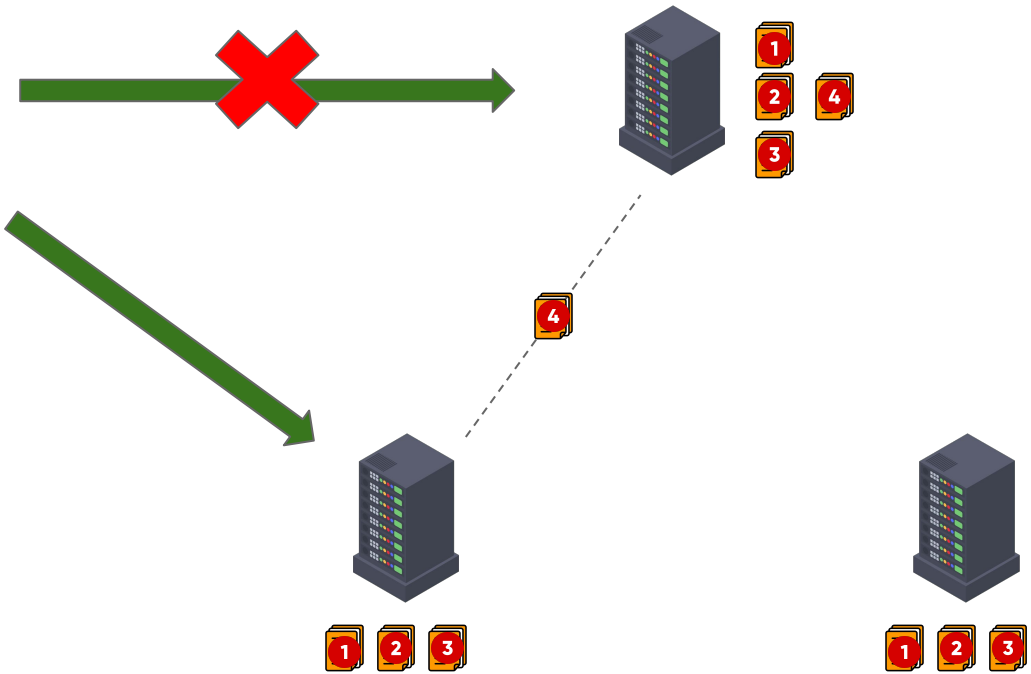
What happens when overloading or NF failure arises?



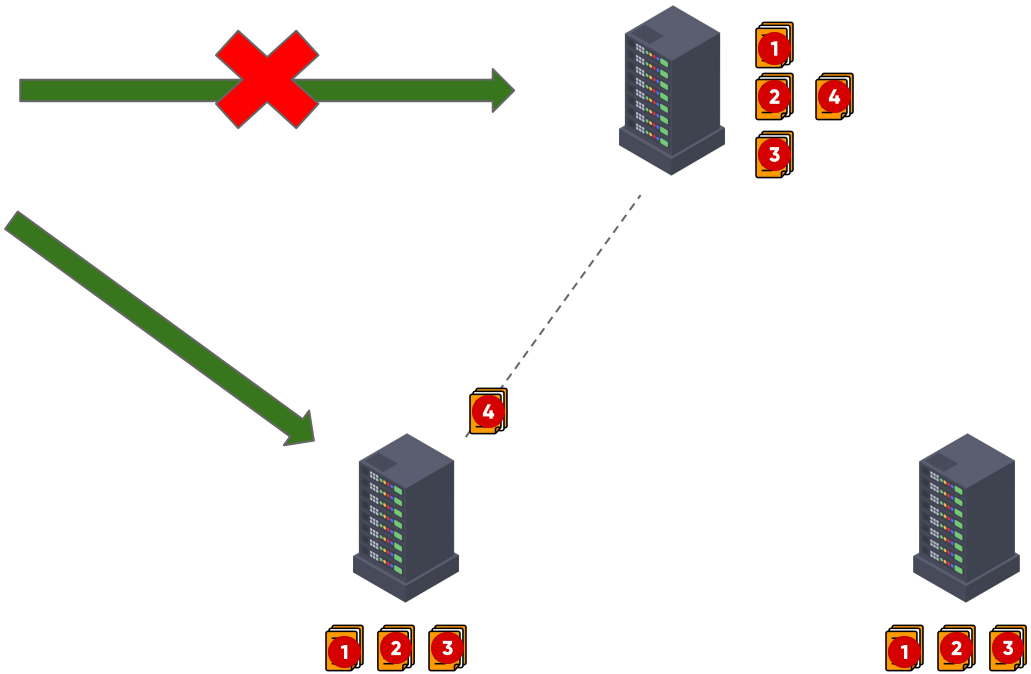
What happens when overloading or NF failure arises?



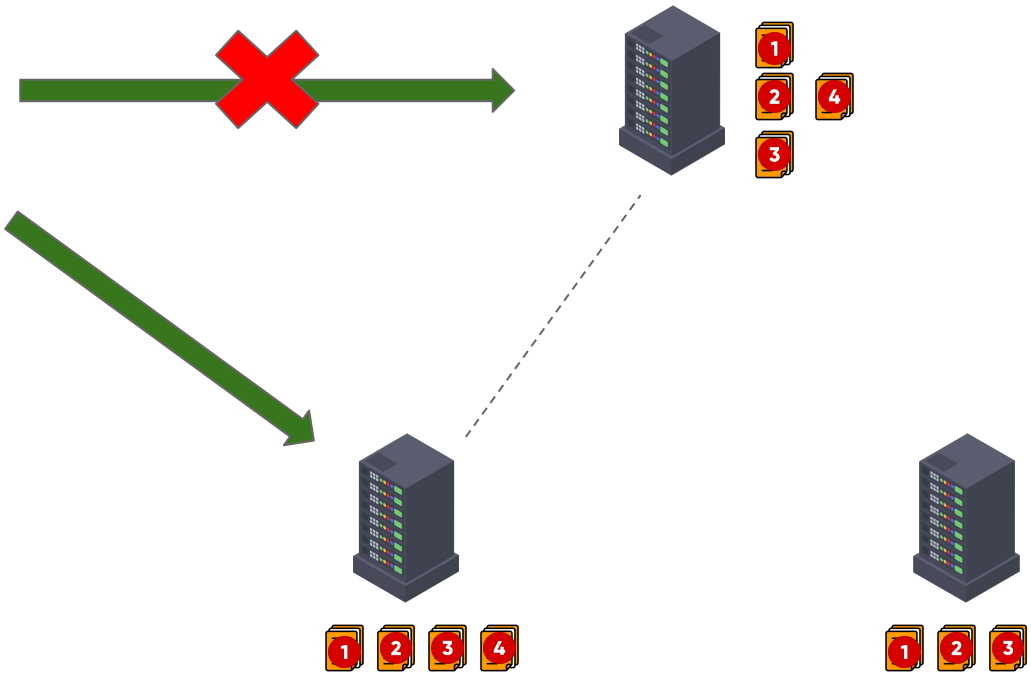
What happens when overloading or NF failure arises?



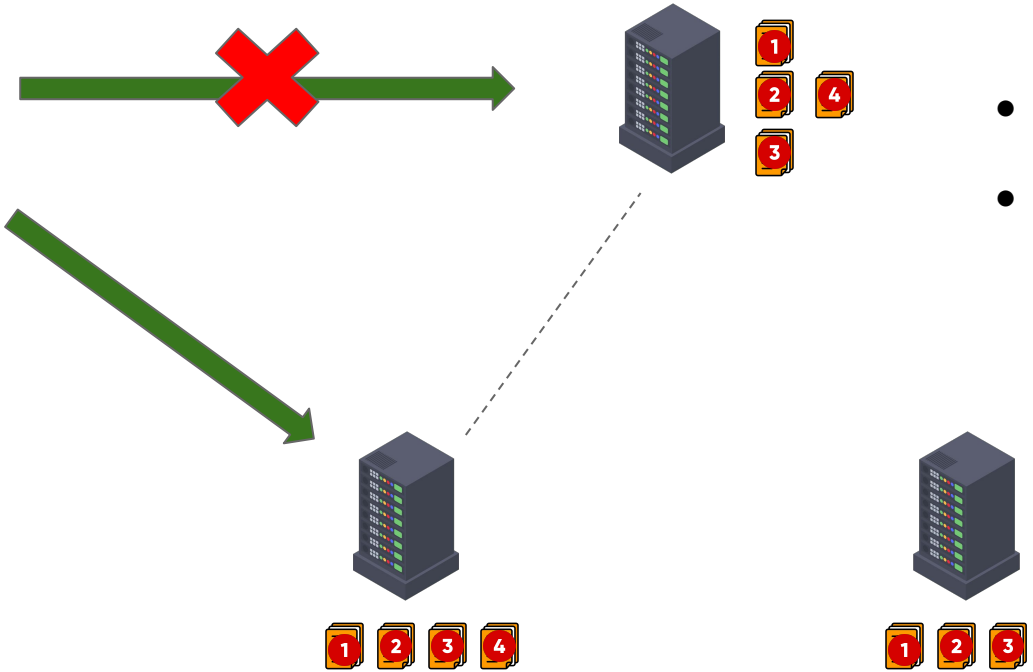
What happens when overloading or NF failure arises?



What happens when overloading or NF failure arises?



What happens when overloading or NF failure arises?



- State information only from last batch is migrated.
- Significantly reduces migration overhead.

Related Works

Previous Works

Elastic State Management

- OpenNF
- StatelessNF

Fault Tolerant State Management

- Pico Replication
- FTMB

Our Work is Based On

- DEFT

OpenNF

Global state update and state migration operations are based on a **central controller**

- NFs send their packets to the controller
- Controller handles packet ordering

Supports both strong and strict consistency for global state update

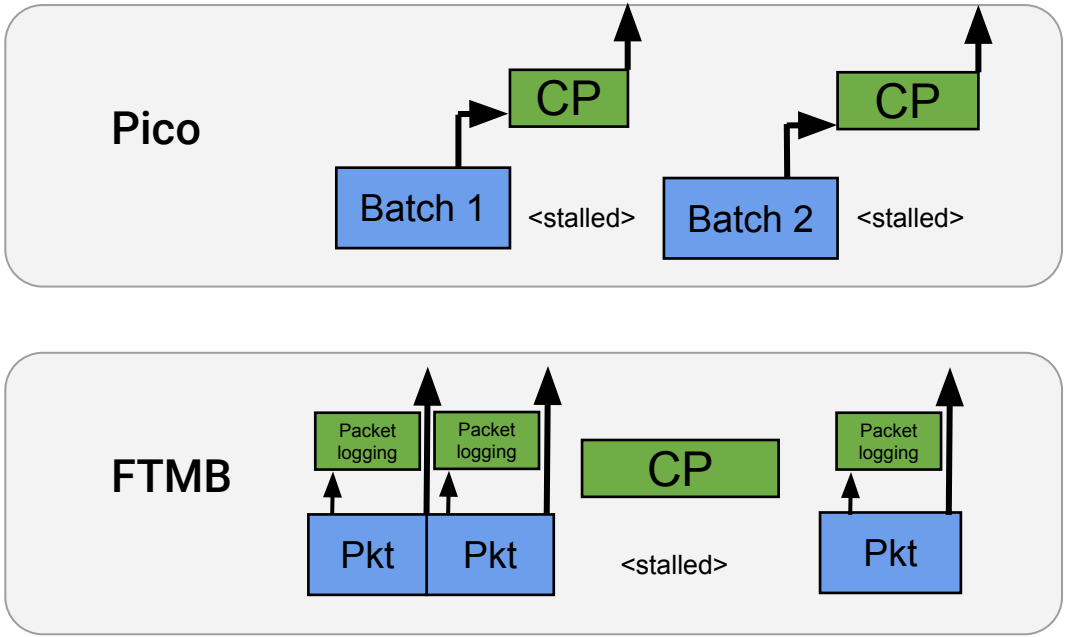
StatelessNF

Packet processing and state storage/access is decoupled, the NFs are stateless

Traffic can be distributed on a per-flow basis or per-packet basis

No state migration required during scaling events

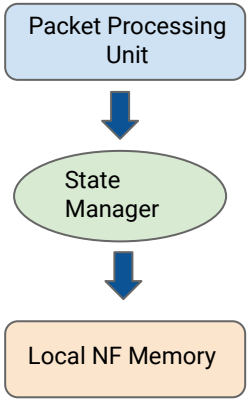
Remotely accessing states introduce additional latency to the system

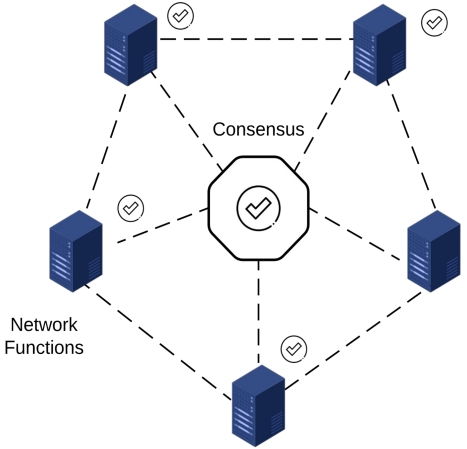


Pico and FTMB

DEFT

Every NF has a **state manager for local states** and **Consensus for Global Updates**

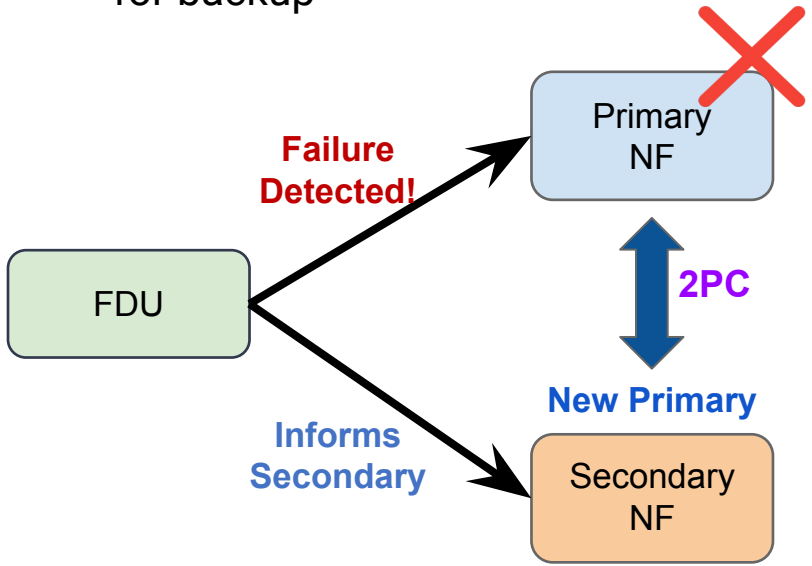




The need for a centralized state storage is no longer necessary

DEFT

Every **Primary NF** has a **Secondary NF** for backup



FDU detects failure and assigns the secondary as the new primary

Problem Definition

Problem Definition

Design a distributed state management system that

Problem Definition

Design a distributed state management system that

Ensures faster processing of packets through multithreading.

Problem Definition

Design a distributed state management system that

Ensures faster processing of packets through multithreading.

Ensures reduced overhead during state migration by implementing clustering.

Problem Definition

Design a distributed state management system that

Ensures faster processing of packets through multithreading.

Ensures reduced overhead during state migration by implementing clustering.

Uses an enhanced cluster-based load balancing algorithm that prevents NF overload, causing reduced latency.

Design Goals

Multithreading Goals

Clustering Goals

Load Balancing Goals

Design Goals

Multithreading Goals

Clustering Goals

Load Balancing Goals

Multithreading Goals

- Packets of different flows will be processed in different threads.
- Multiple flows can be processed in parallel.
- Output buffer will be filled up much faster.
- Overall latency is decreased.

Design Goals

Multithreading Goals

Clustering Goals

Load Balancing Goals

Clustering Goals

- Network function instances will be grouped in clusters.
- After each batch process, an NF instance will share its state info with the other members of the cluster.
- State migration will require only the migration of the information of the latest processed batch.
- State migration overhead will be reduced significantly.

Design Goals

Multithreading Goals

Clustering Goals

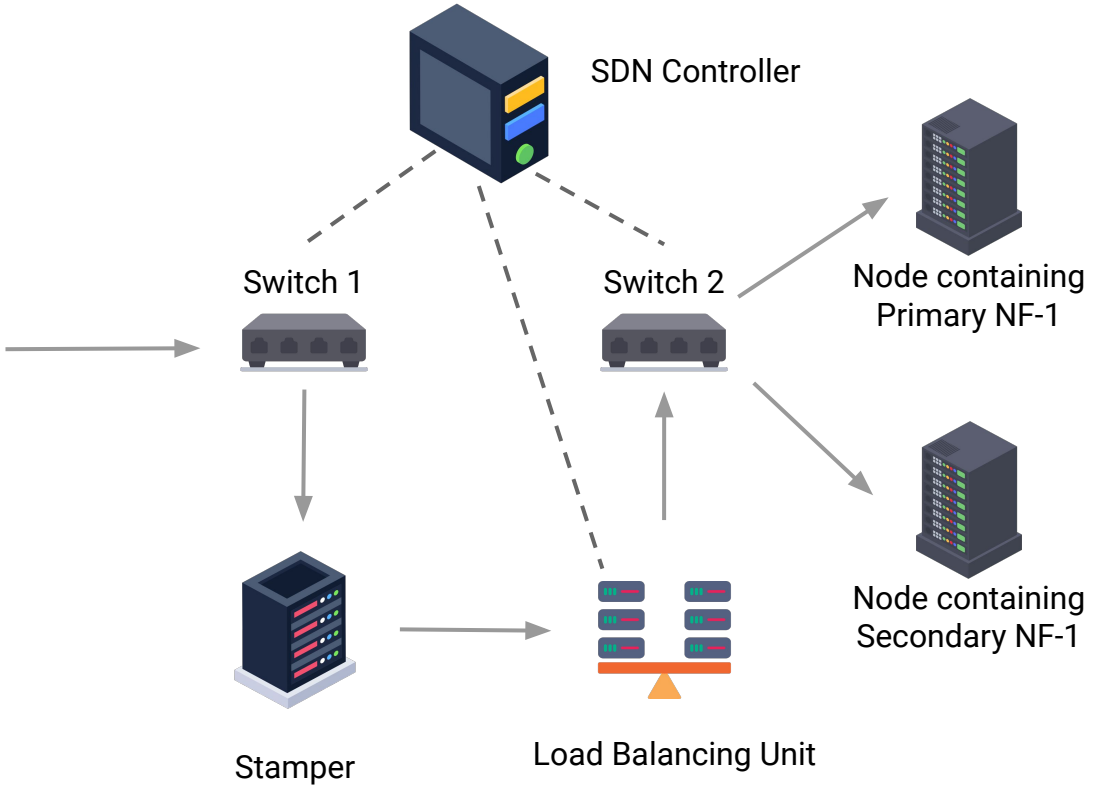
Load Balancing Goals

Load Balancing Goals

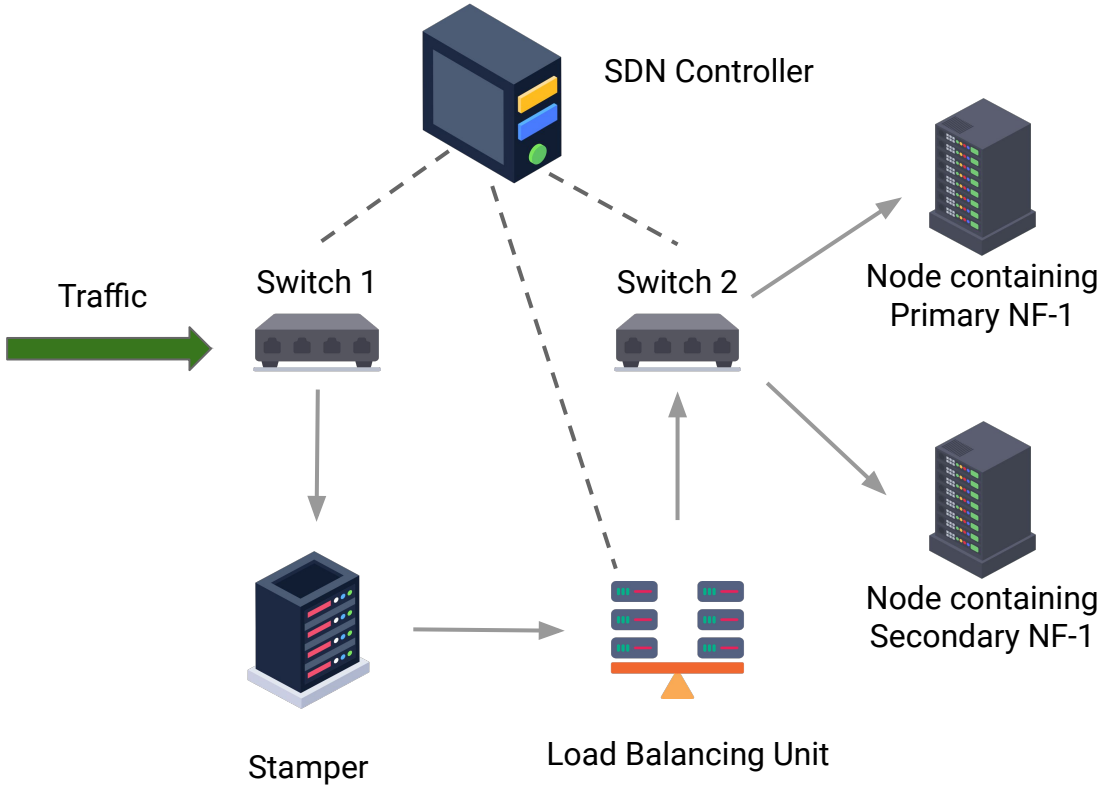
- An enhanced cluster-based load balancing algorithm to prevent NFs from being overloaded.
- Whenever an NF instance is overloaded, a flow is redirected to another NF instance.
- Decreased load will allow the NF to process faster.
- Overall latency is decreased.

Methodology

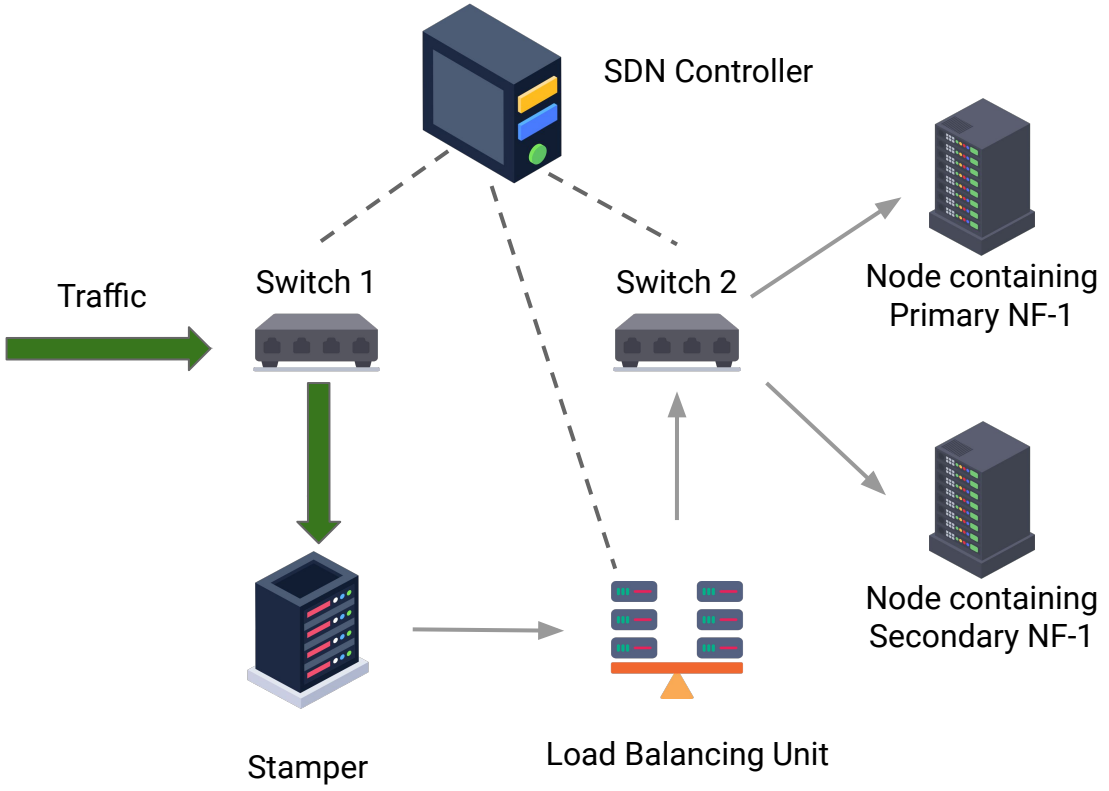
Our Topology



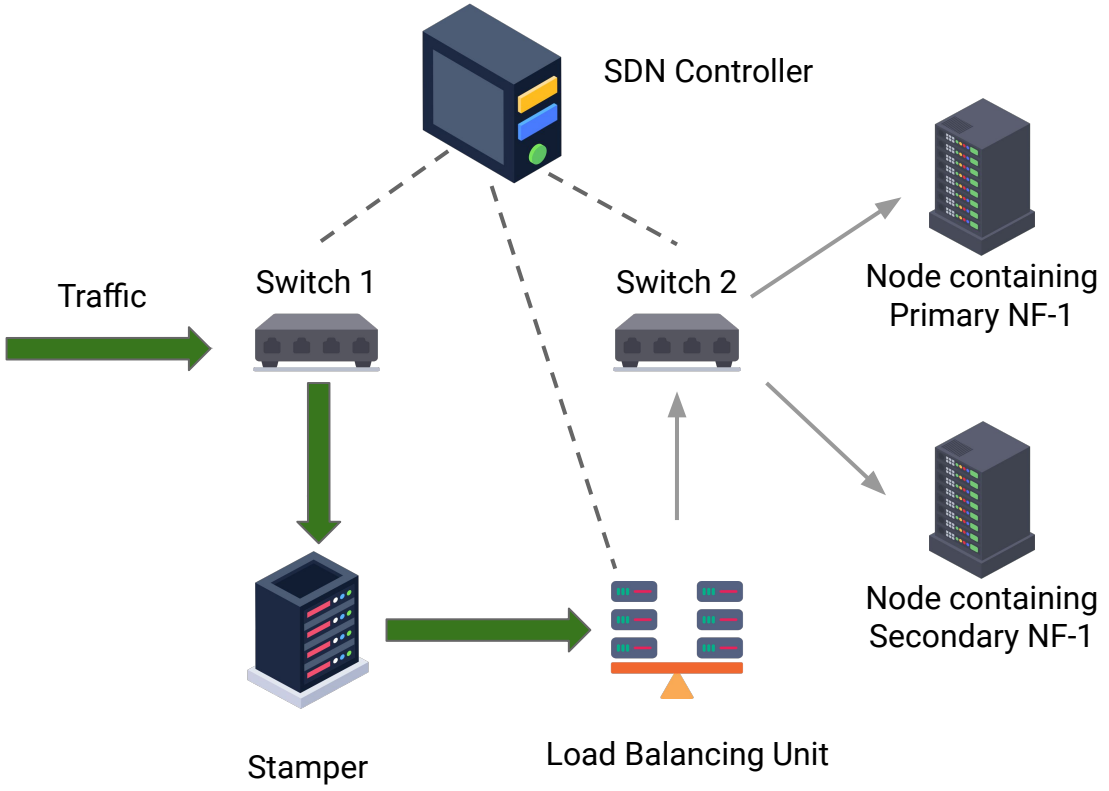
Our Topology



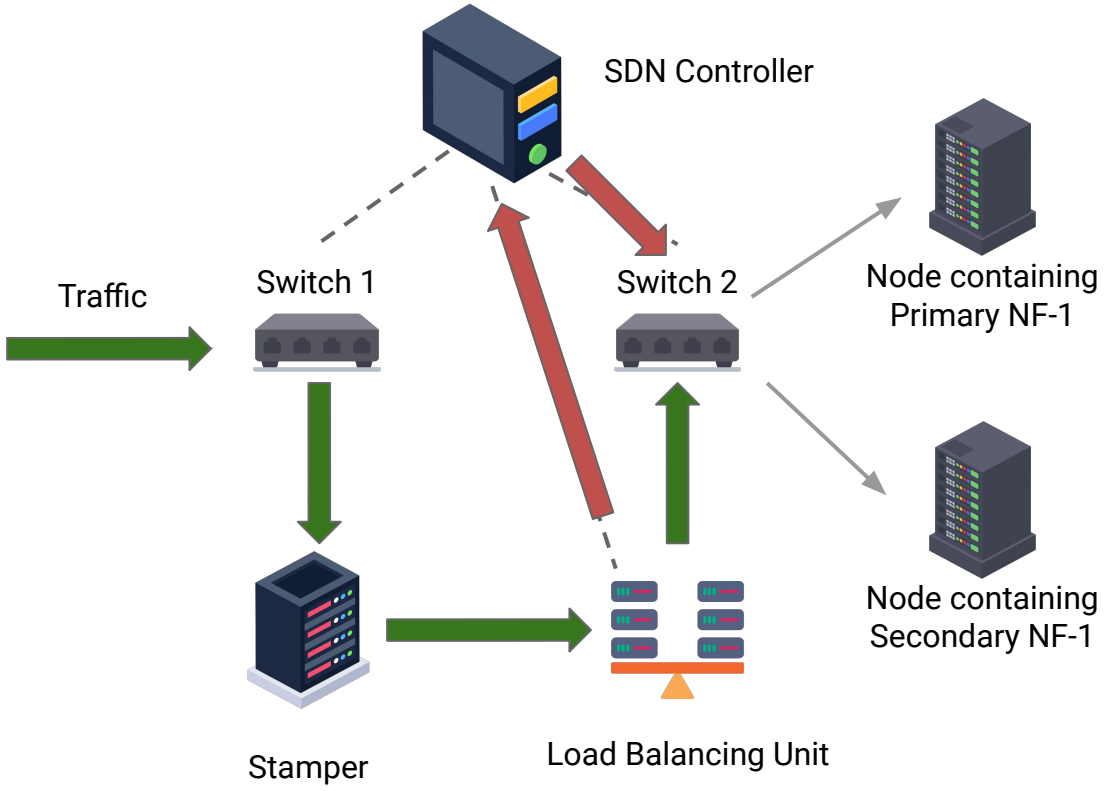
Our Topology



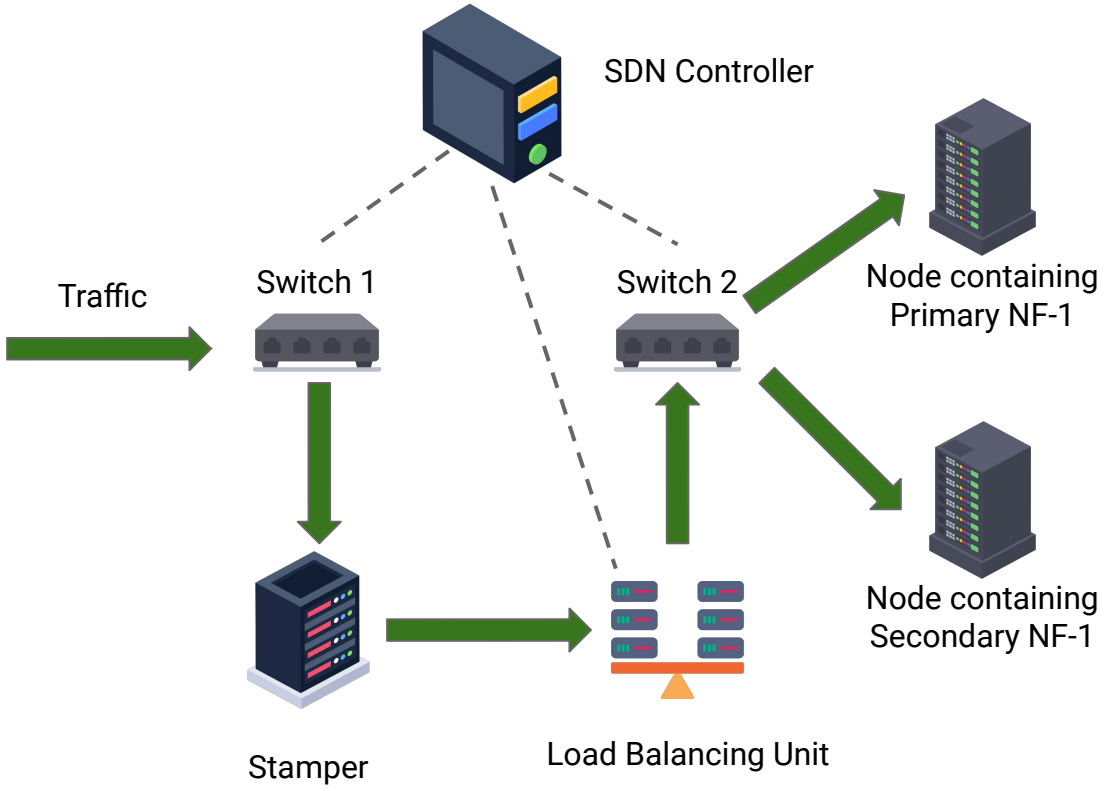
Our Topology



Our Topology

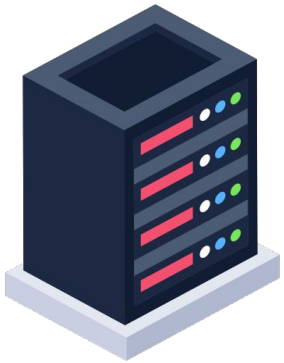


Our Topology



Stamper

- Maintains a counter for packets of each flow
- Stamps the packets with the value of packet_count associated with the respective flow

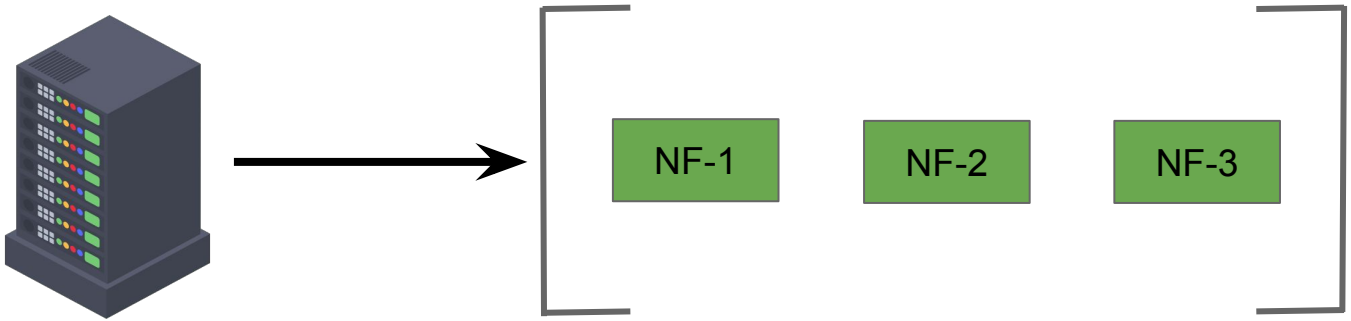


Stamper

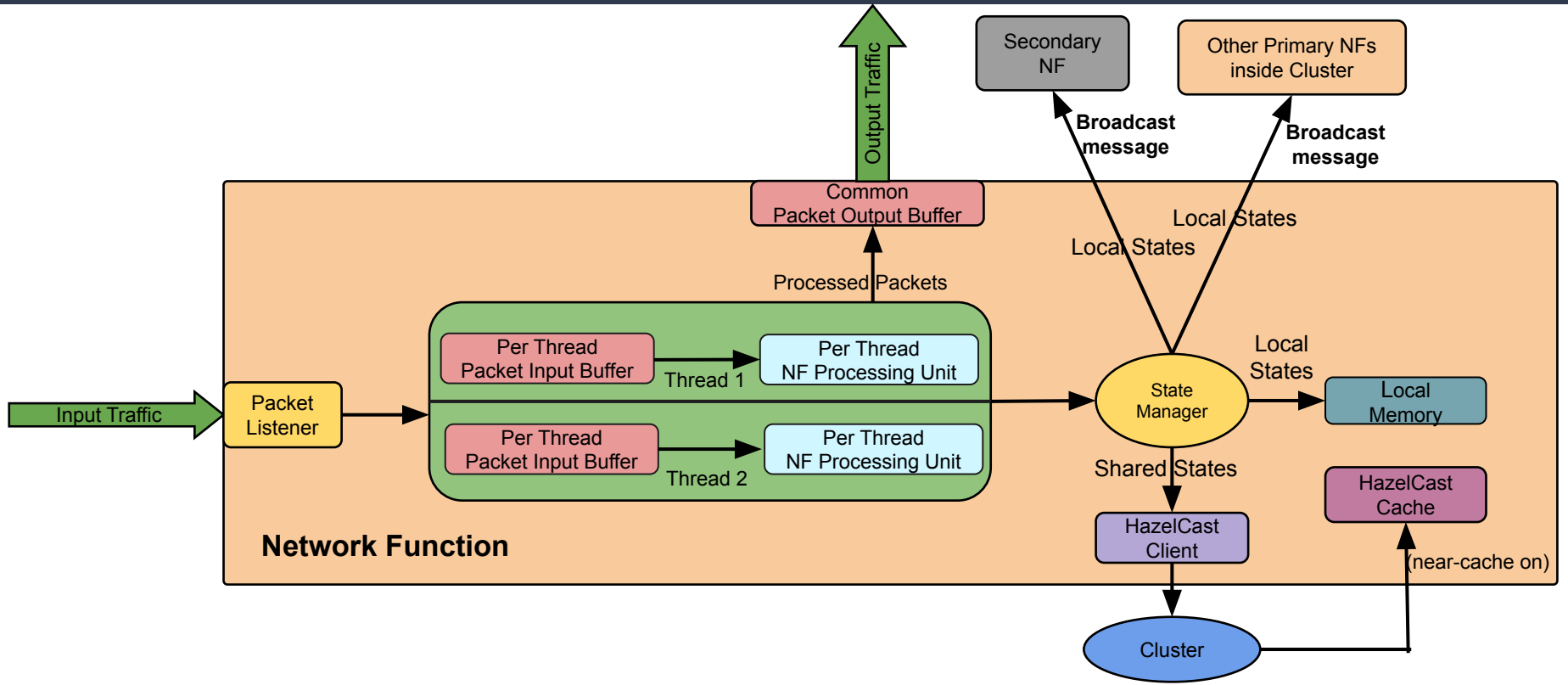
Flow ID	packet_count
src_IP = 173.29.0.4, src_port= 43621, dst_IP = 173.16.0.2, dst_port= 8000, Protocol = UDP	6
src_IP = 173.29.0.4, src_port= 56609, dst_IP = 173.16.0.2, dst_port= 8000, Protocol = UDP	20
src_IP = 173.28.0.8, src_port= 43621, dst_IP = 173.17.0.4, dst_port= 8000, Protocol = UDP	15

Node

- A single node may contain one or more NFs
- All NFs are independent of one another



Architecture



Operations

Normal Operation

Flow Migration Operation

Failover Operation

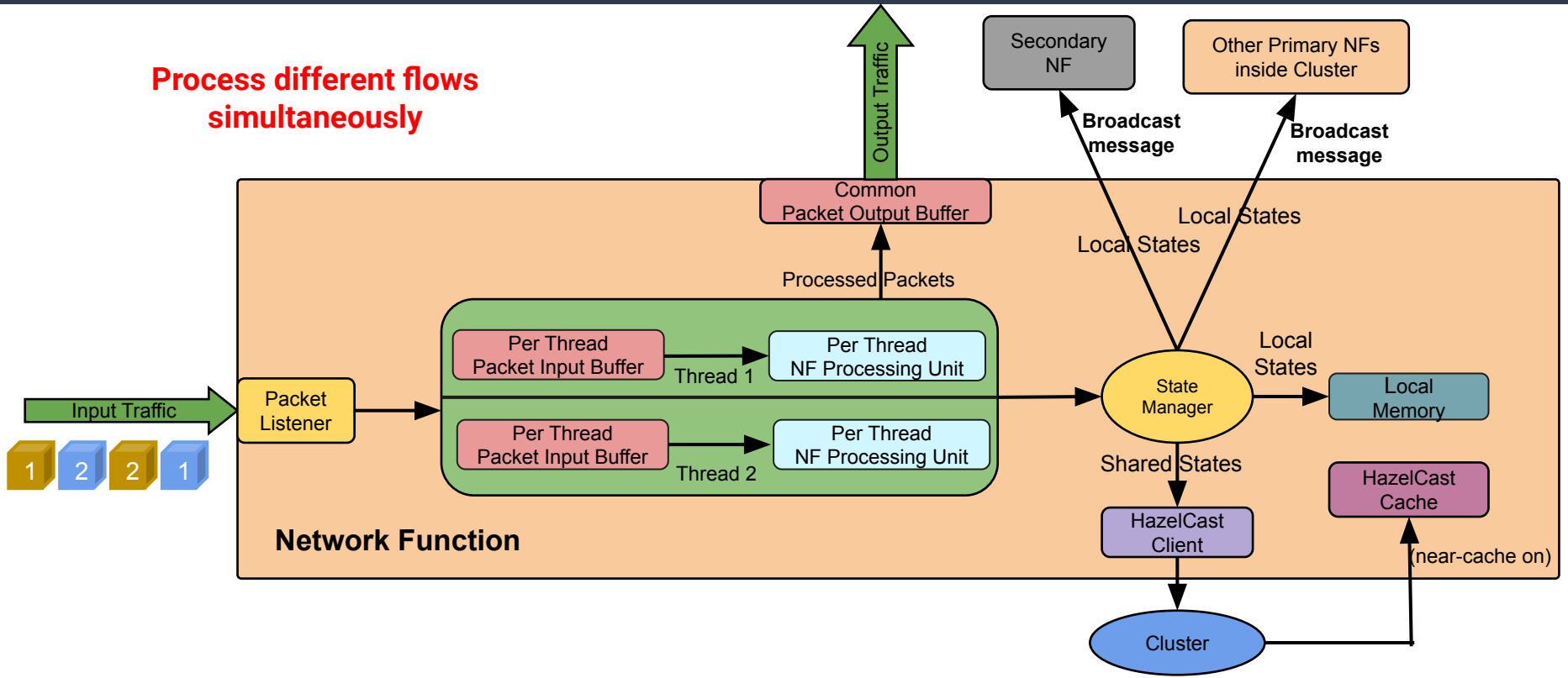
Operations

Normal Operation

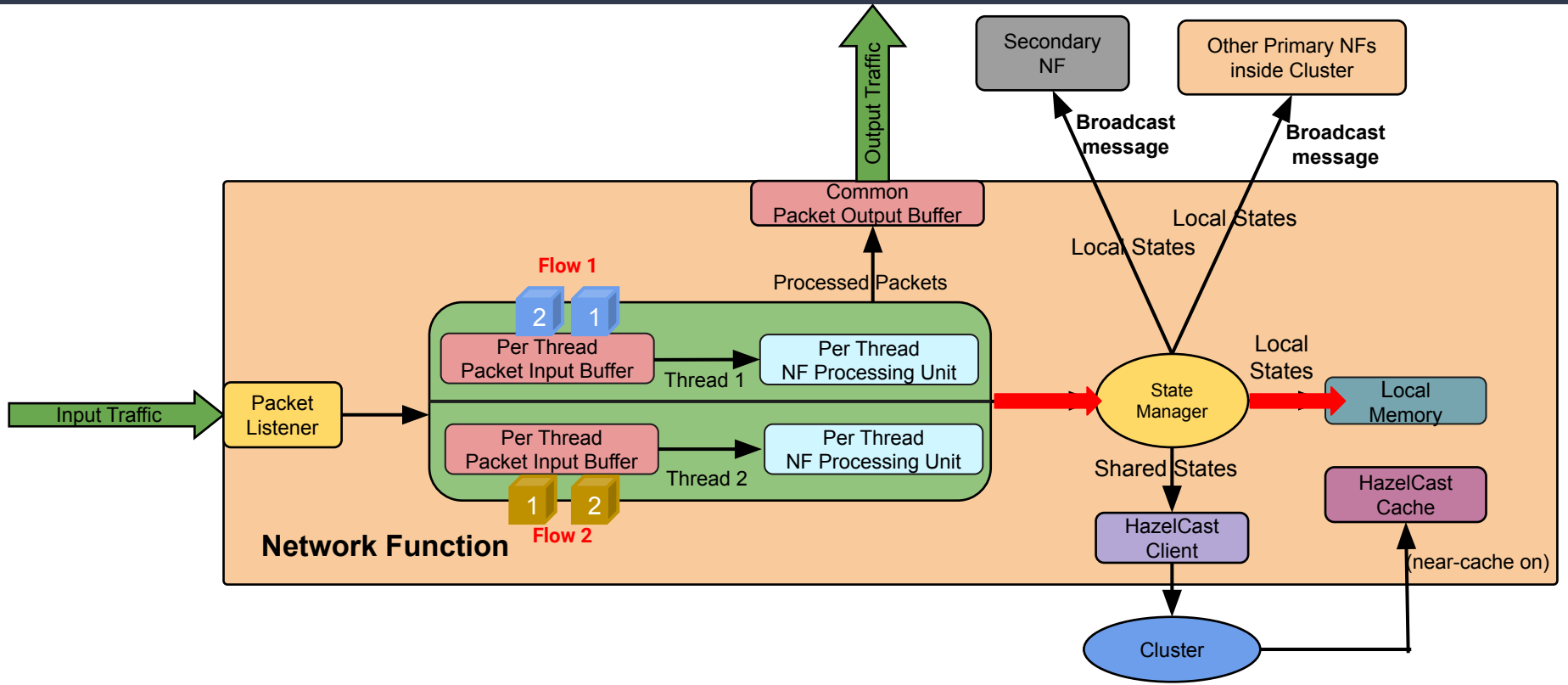
Flow Migration Operation

Failover Operation

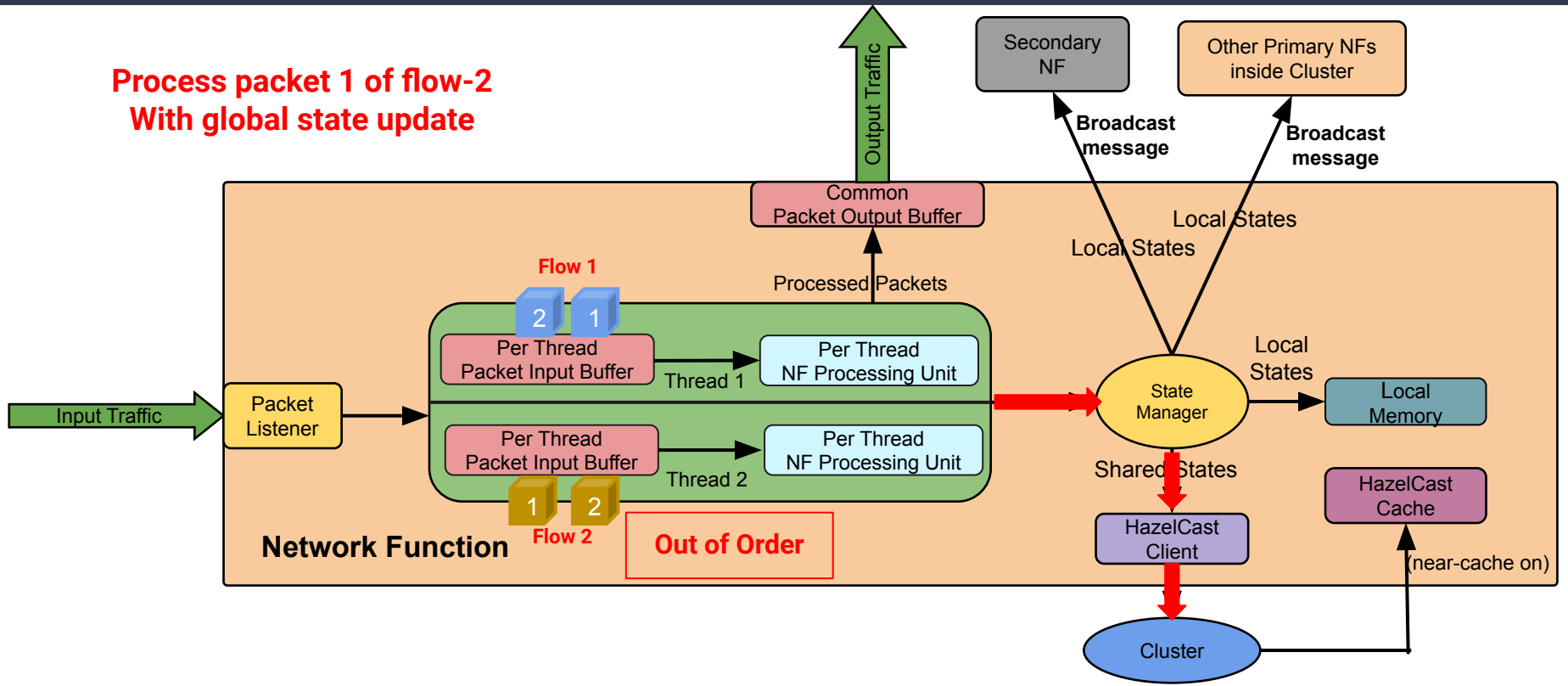
Architecture



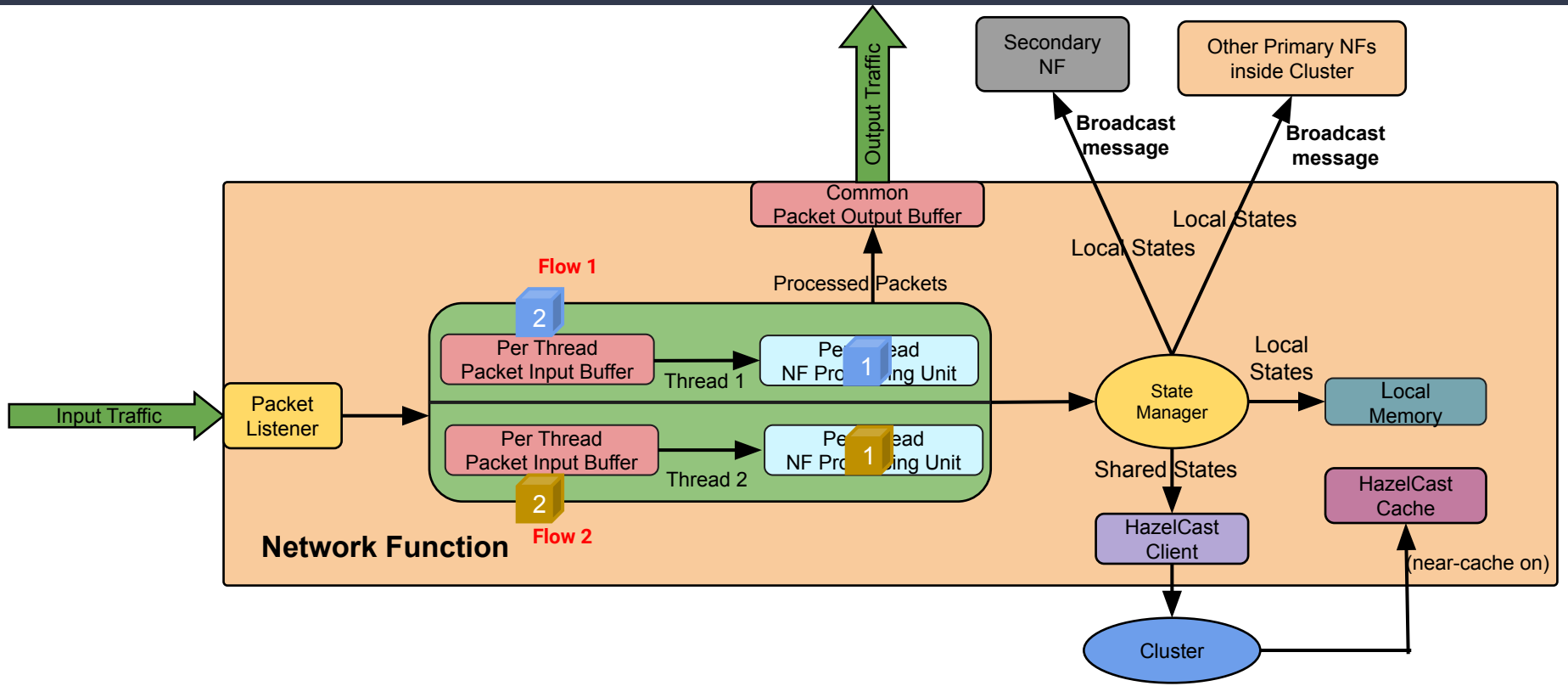
Packet 1 of flow-1 with local state update



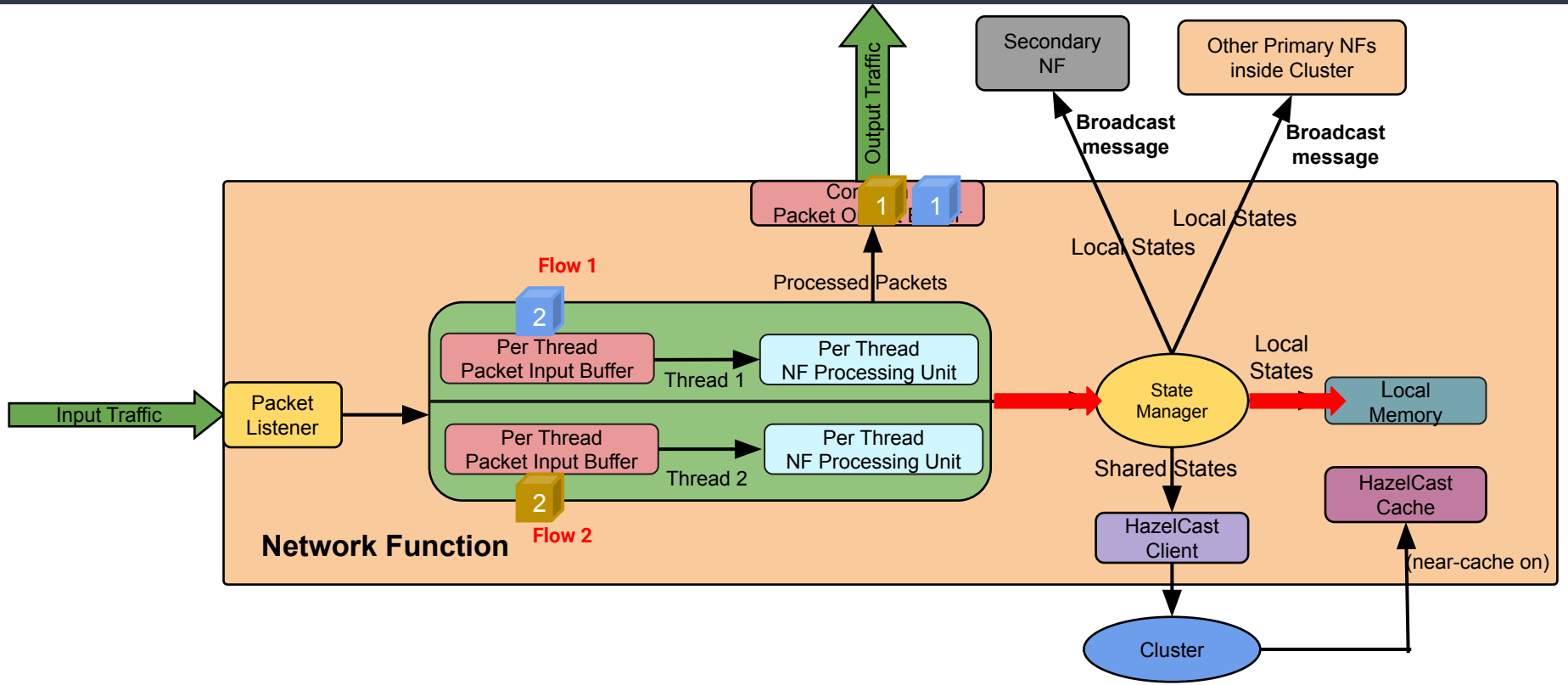
Packet 2 of flow-2 arriving out of order



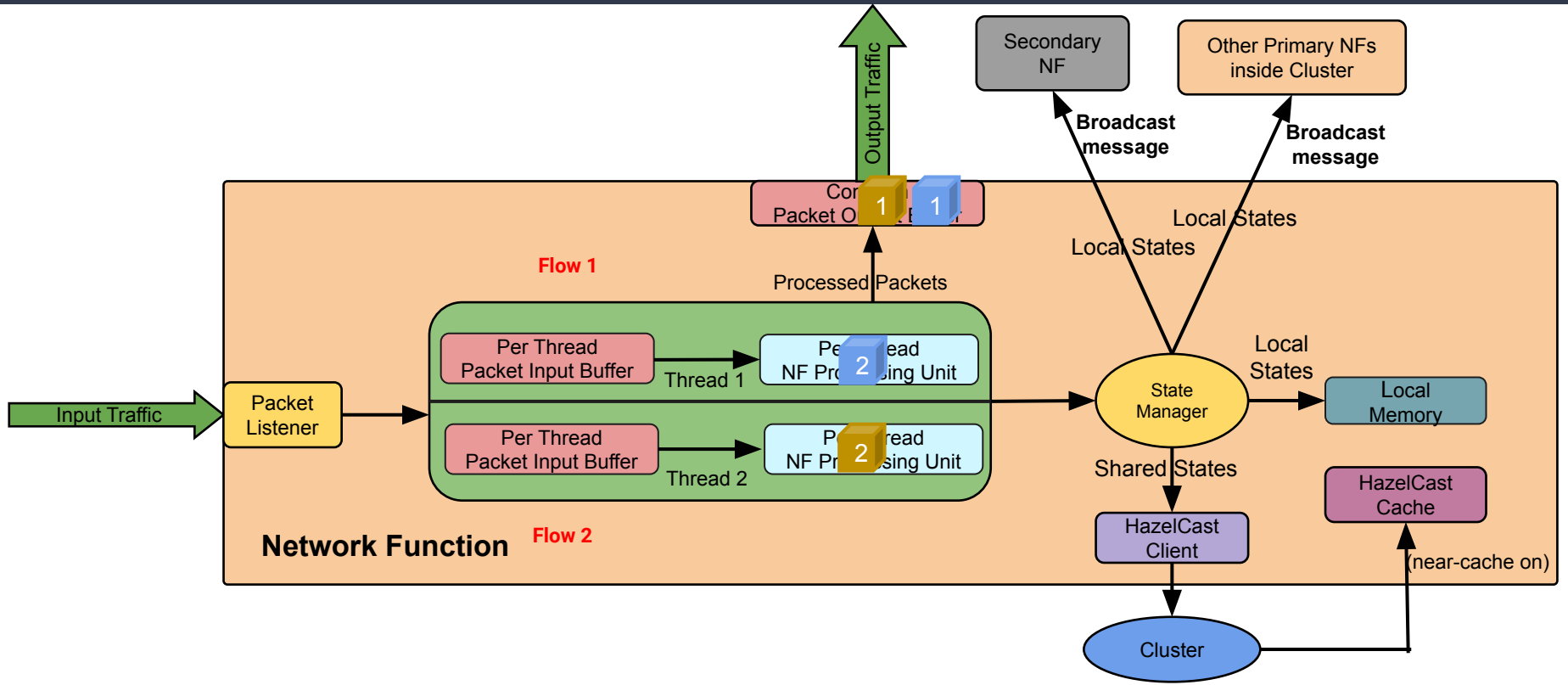
Packet 1 of flow-1 with local state update and packet 1 of flow-2 with global state update



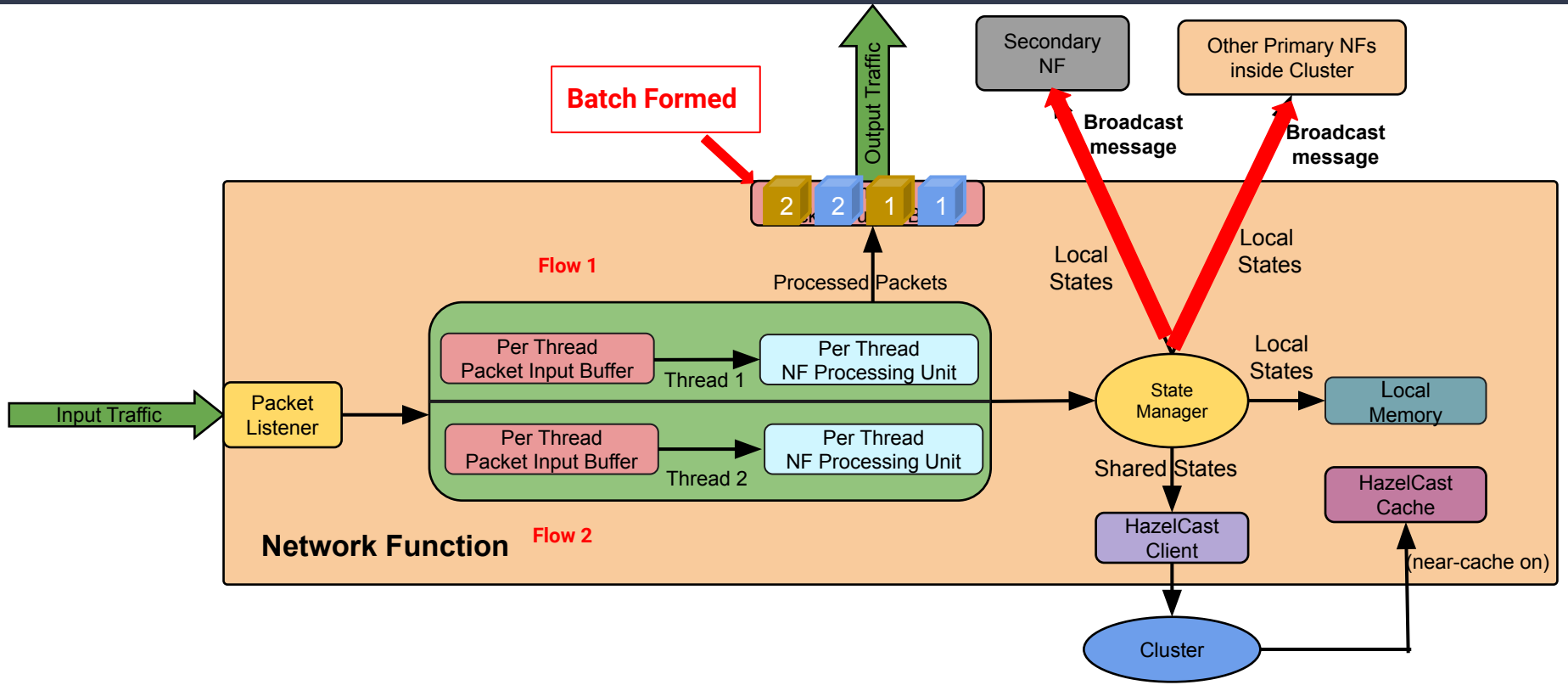
Packet 2 of flow-1 and packet 2 of flow-2 with local state update



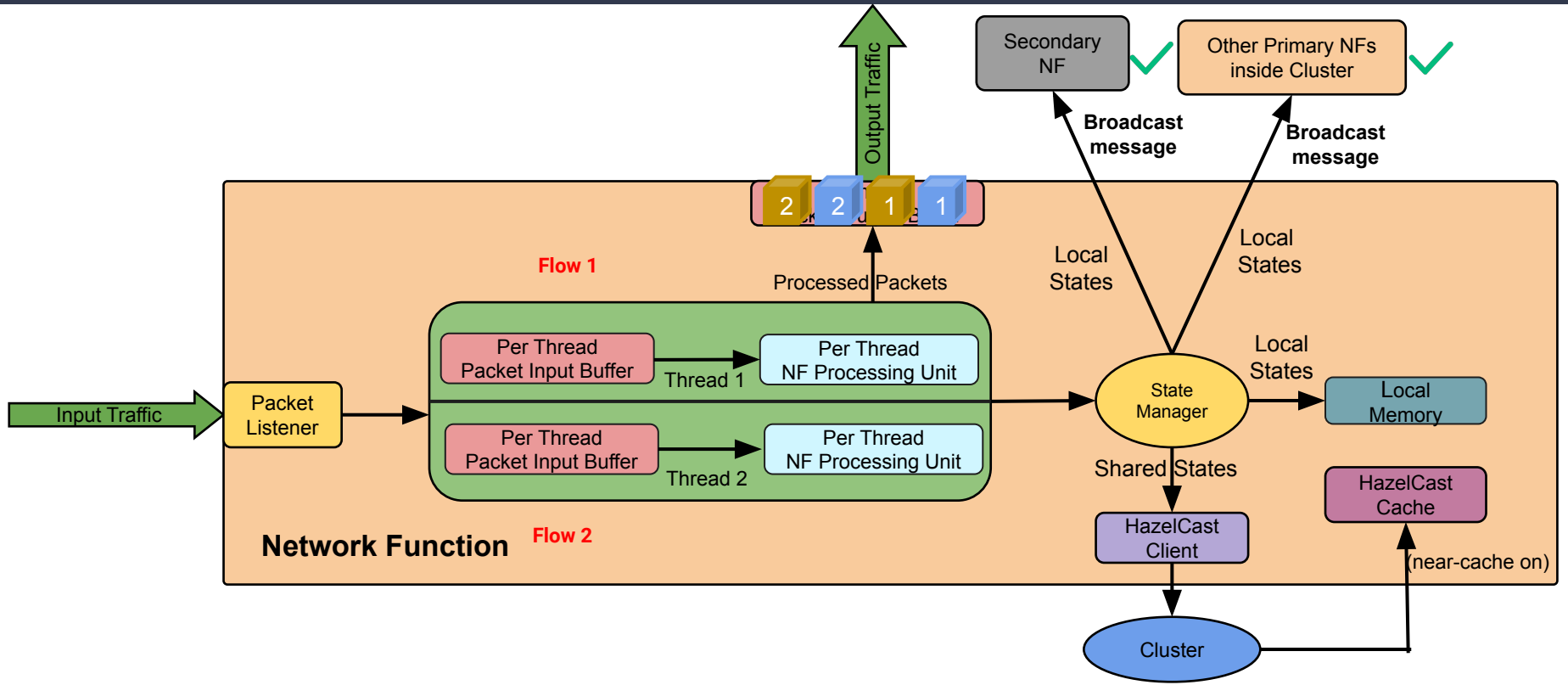
Packet 2 of flow-1 and packet 2 of flow-2 with local state update



Packet Release



Packet Release



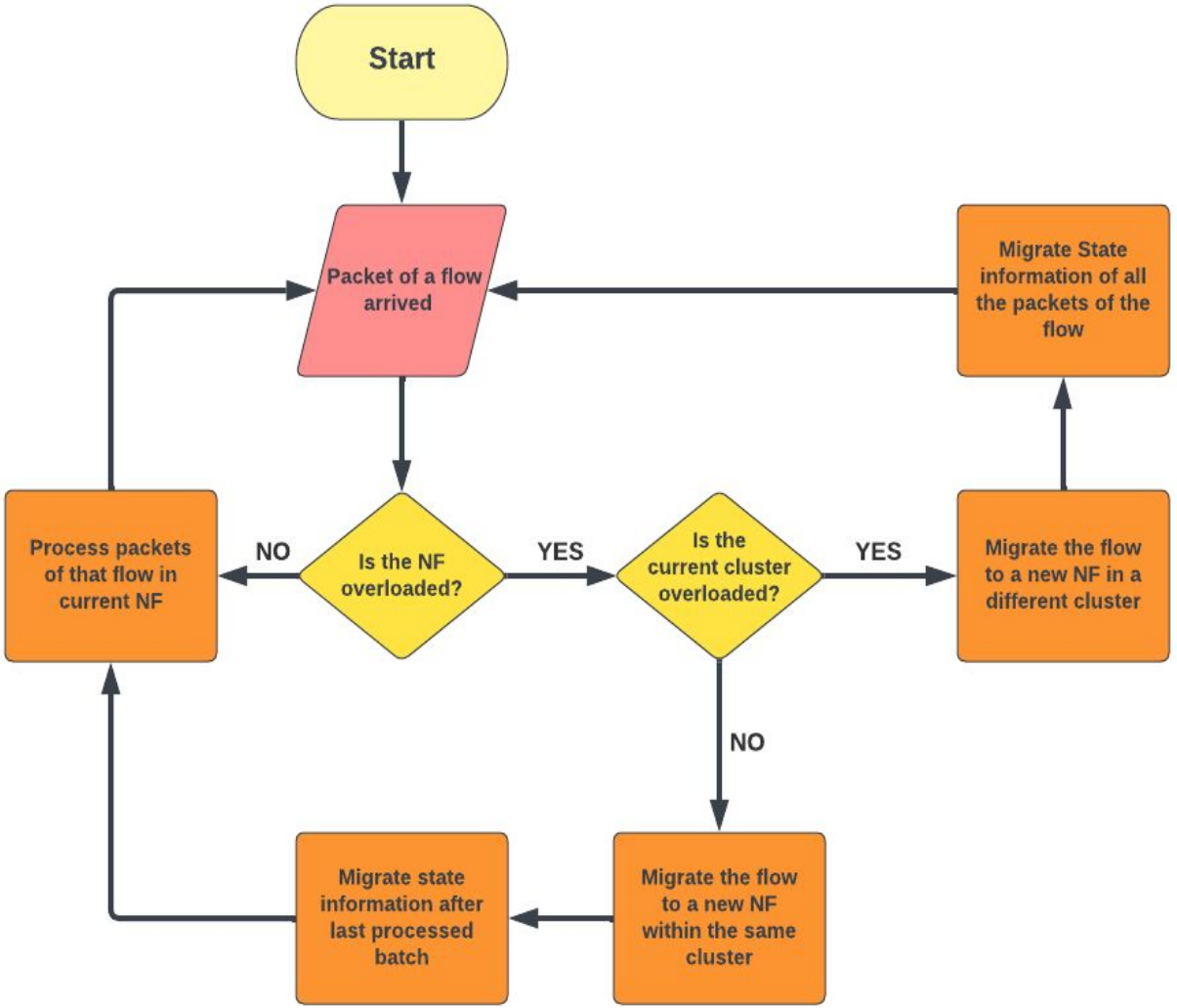
Operations

Normal Operation

Flow Migration Operation

Failover Operation

Load Balancing Technique

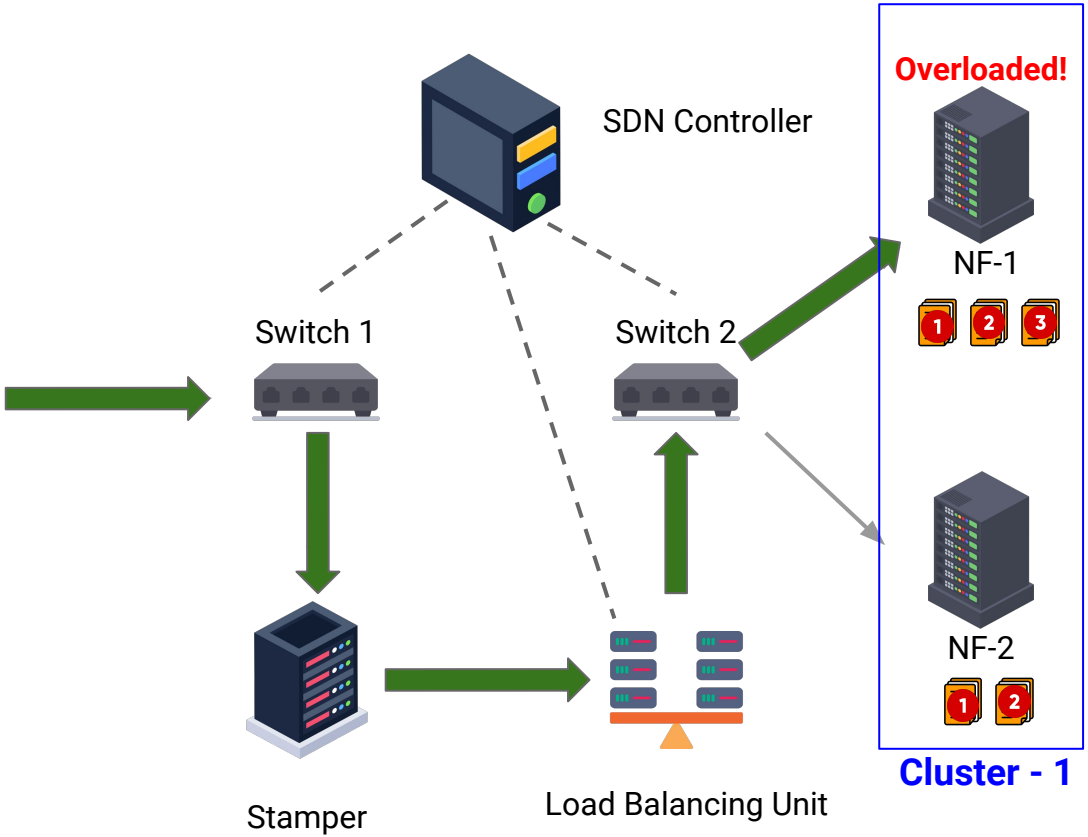


Flow Migration

Case 1

Overloaded NF: NF-1
Selected NF : NF-2 (Inside same cluster)

- NF-1 keeps buffering packets

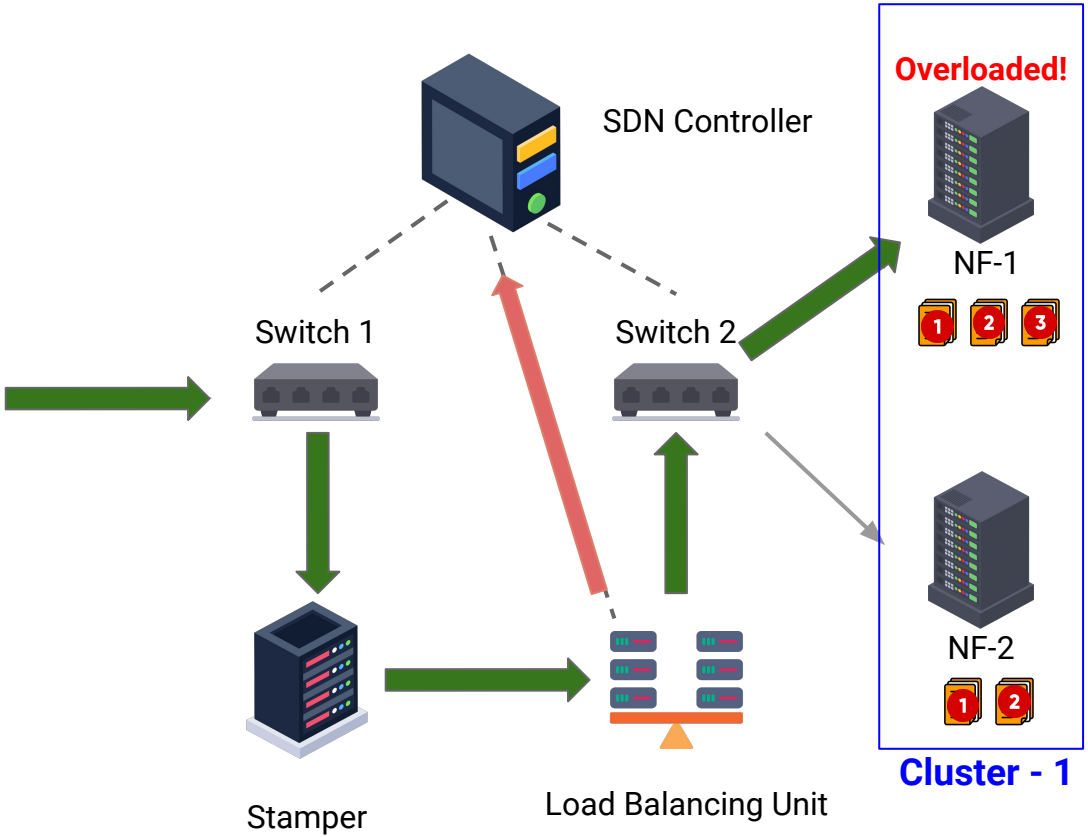


Flow Migration

Case 1

Overloaded NF: NF-1
Selected NF : NF-2 (Inside same cluster)

- NF-1 keeps buffering packets
- Load balancing unit selects existing NF-2

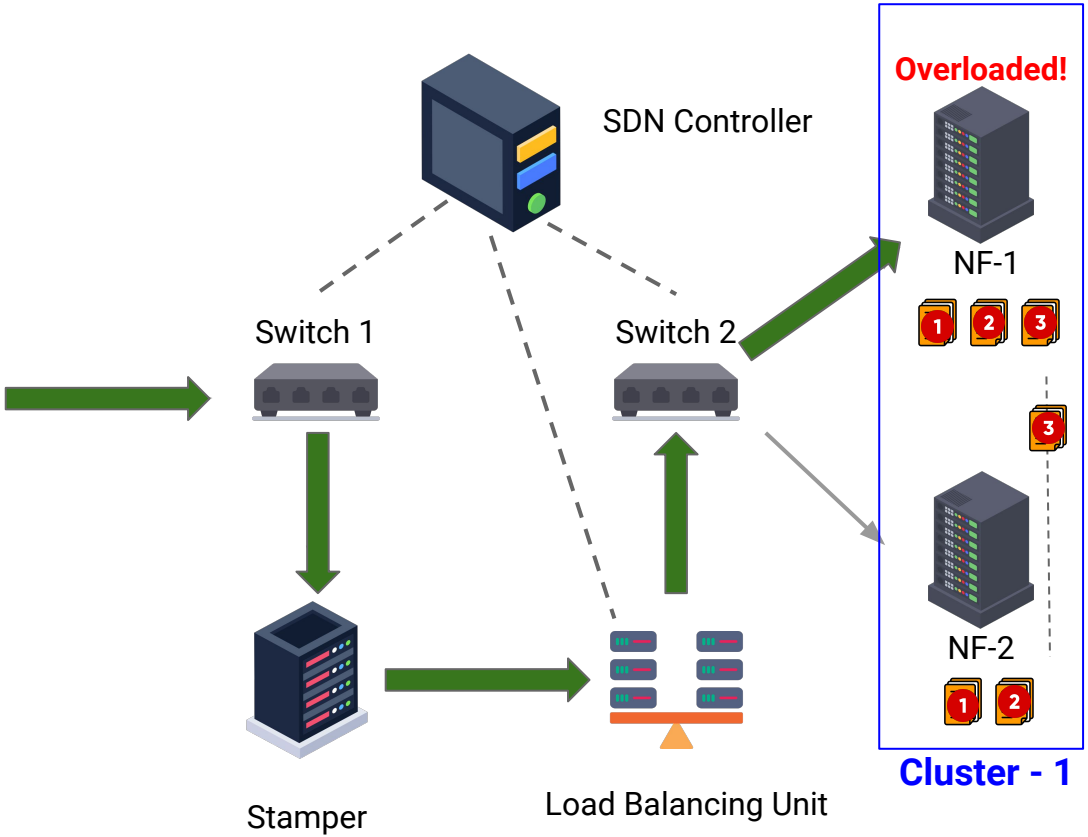


Flow Migration

Case 1

Overloaded NF: NF-1
Selected NF : NF-2 (Inside same cluster)

- NF-1 keeps buffering packets
- Load balancing unit selects existing NF-2
- NF-1 transfers states updated after last batch to NF-2

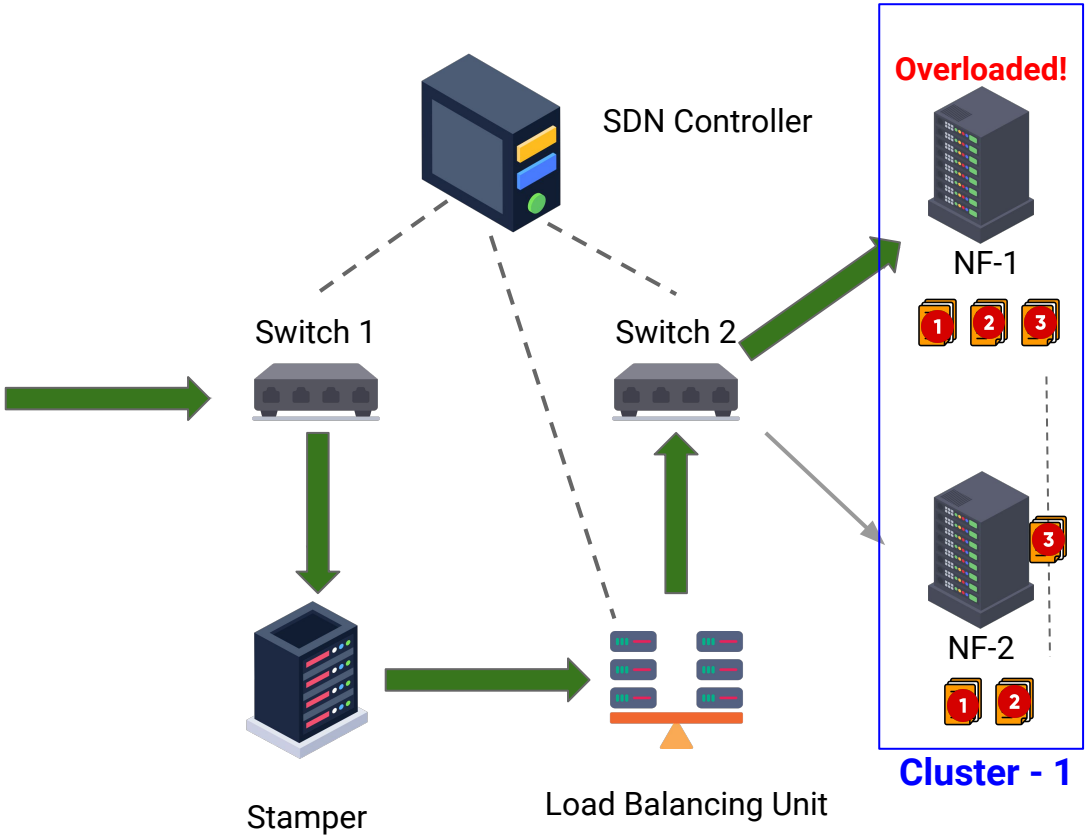


Flow Migration

Case 1

Overloaded NF: NF-1
Selected NF : NF-2 (Inside same cluster)

- NF-1 keeps buffering packets
- Load balancing unit selects existing NF-2
- NF-1 transfers states updated after last batch to NF-2

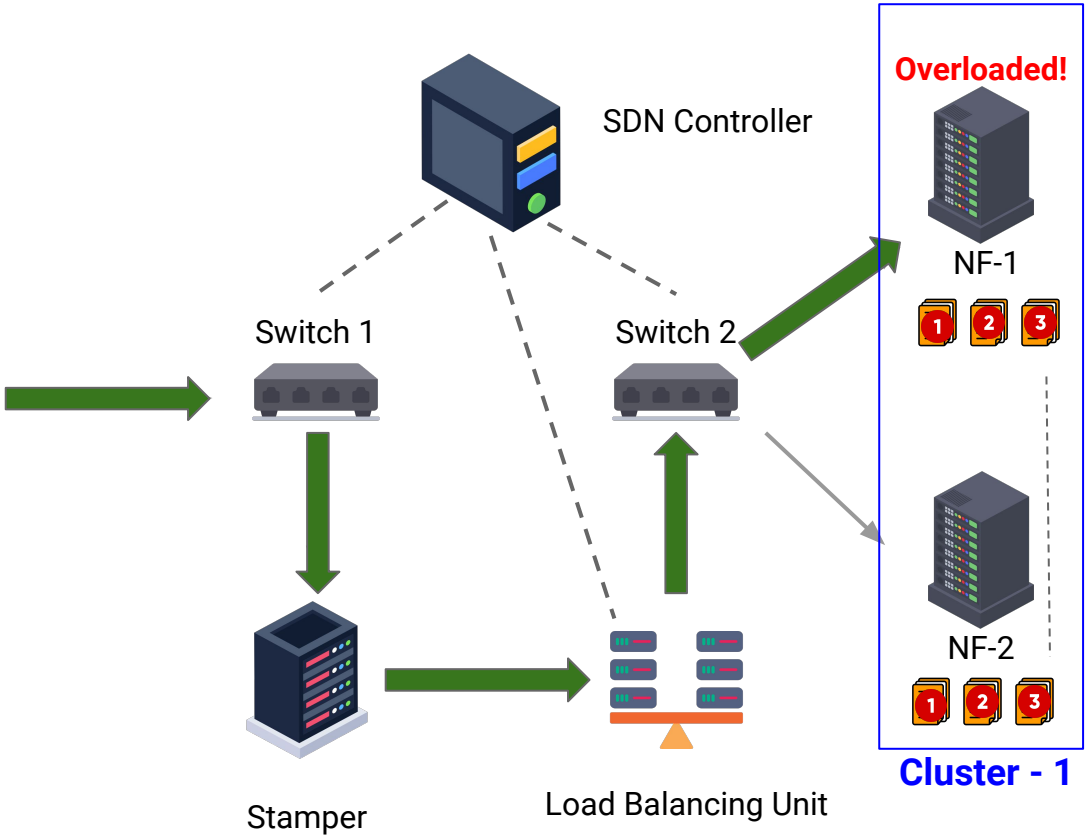


Flow Migration

Case 1

Overloaded NF: NF-1
Selected NF : NF-2 (Inside same cluster)

- NF-1 keeps buffering packets
- Load balancing unit selects existing NF-2
- NF-1 transfers states updated after last batch to NF-2

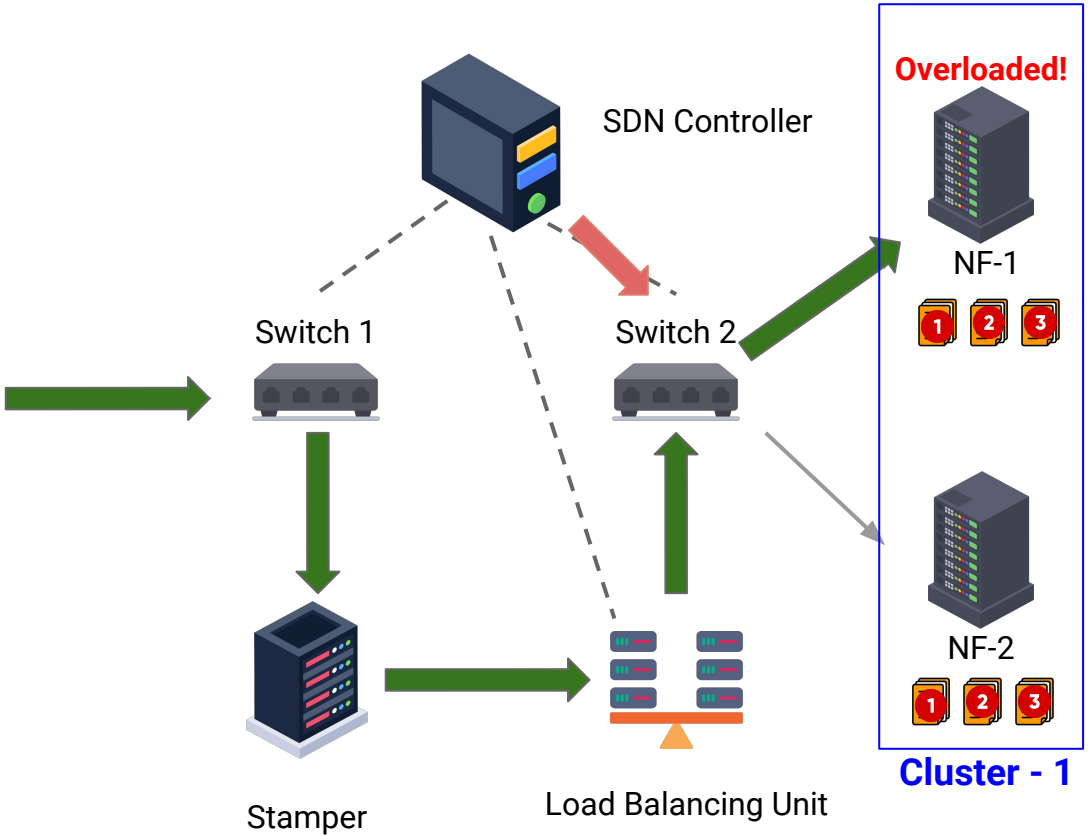


Flow Migration

Case 1

Overloaded NF: NF-1
Selected NF : NF-2 (Inside same cluster)

- NF-1 keeps buffering packets
- Load balancing unit selects existing NF-2
- NF-1 transfers states updated after last batch to NF-2
- SDN controller changes flow rule at switch 2

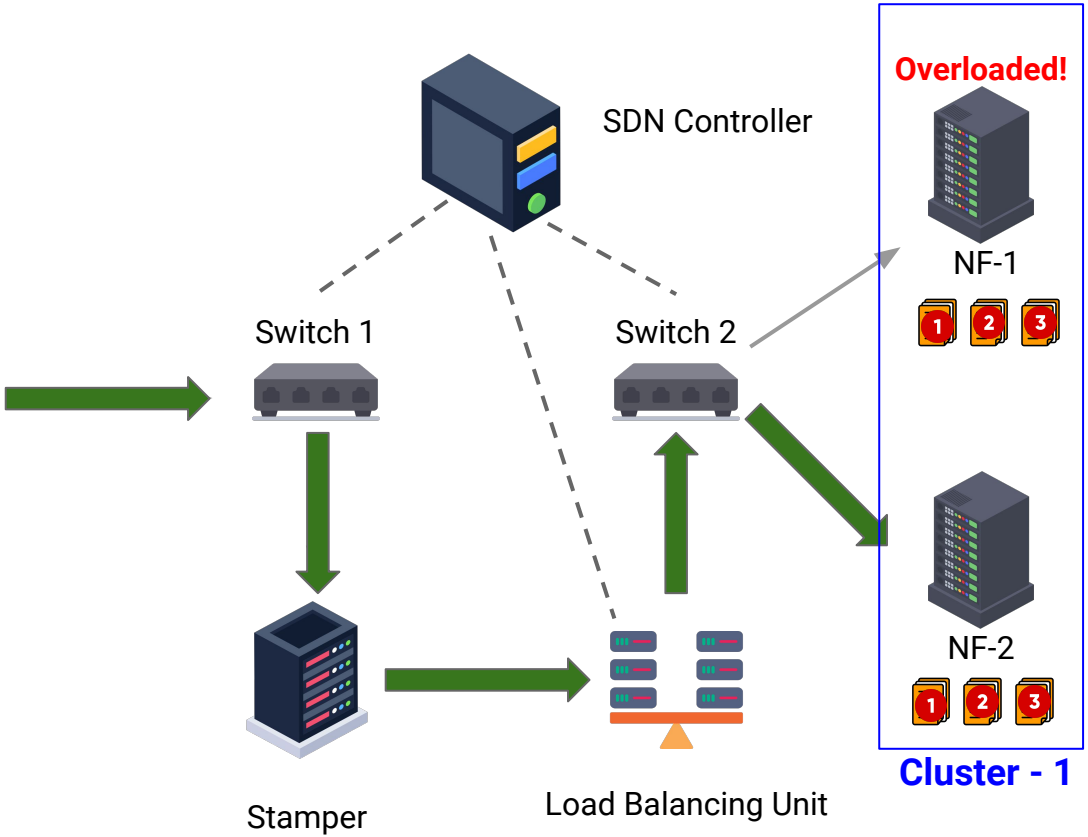


Flow Migration

Case 1

Overloaded NF: NF-1
Selected NF : NF-2 (Inside same cluster)

- NF-1 keeps buffering packets
- Load balancing unit selects existing NF-2
- NF-1 transfers states updated after last batch to NF-2
- SDN controller changes flow rule at switch 2
- NF-1 forwards buffered packets to Destination NF-2

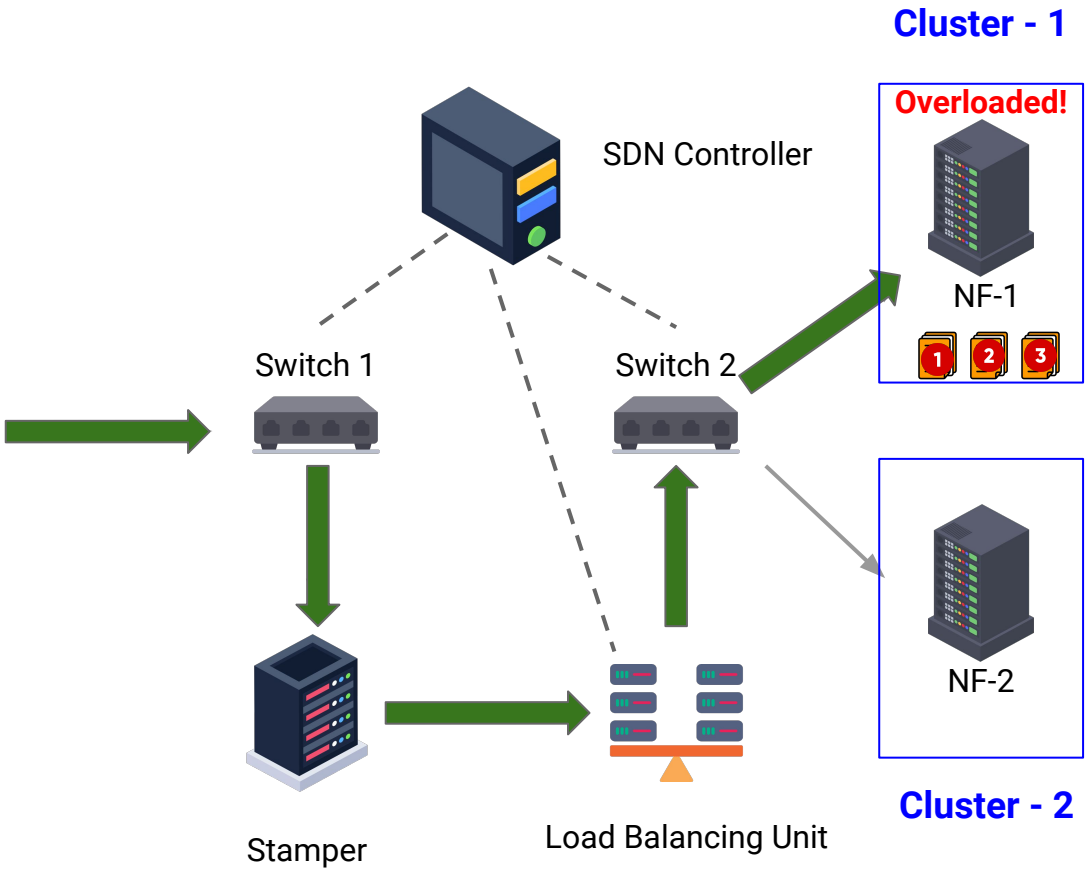


Flow Migration

Case 2

Overloaded NF: NF-1
Selected NF : NF-2 (Another cluster)

- NF-1 keeps buffering packets

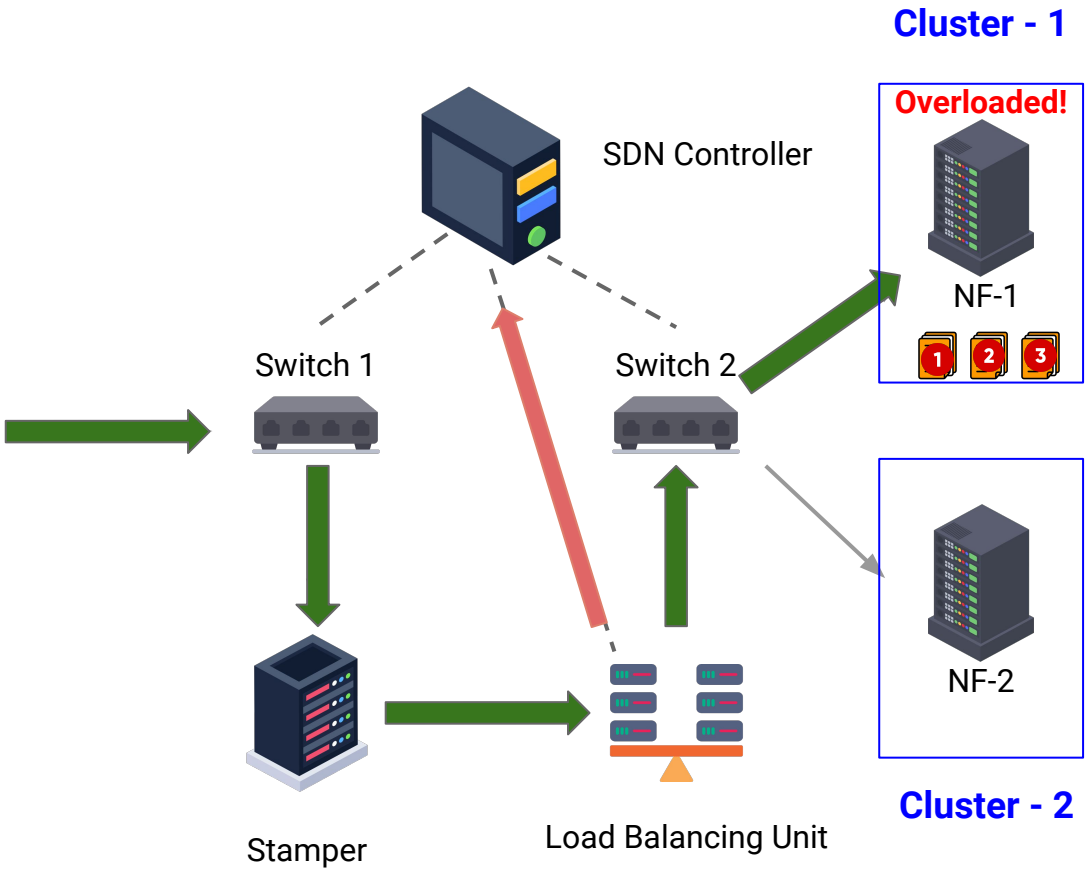


Flow Migration

Case 2

Overloaded NF: NF-1
Selected NF : NF-2 (Another cluster)

- NF-1 keeps buffering packets
- Load balancing unit selects existing NF-2

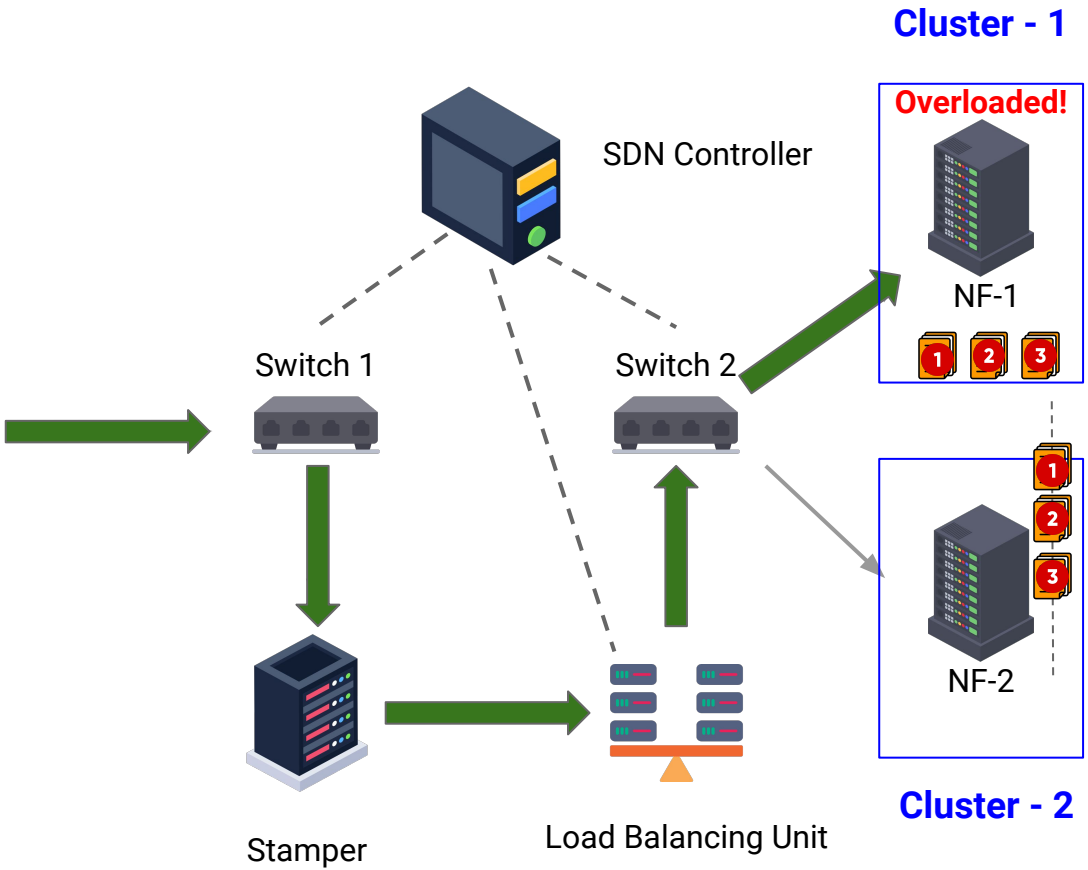


Flow Migration

Case 2

Overloaded NF: NF-1
Selected NF : NF-2 (Another cluster)

- NF-1 keeps buffering packets
- Load balancing unit selects existing NF-2
- NF-1 transfers all state information to NF-2

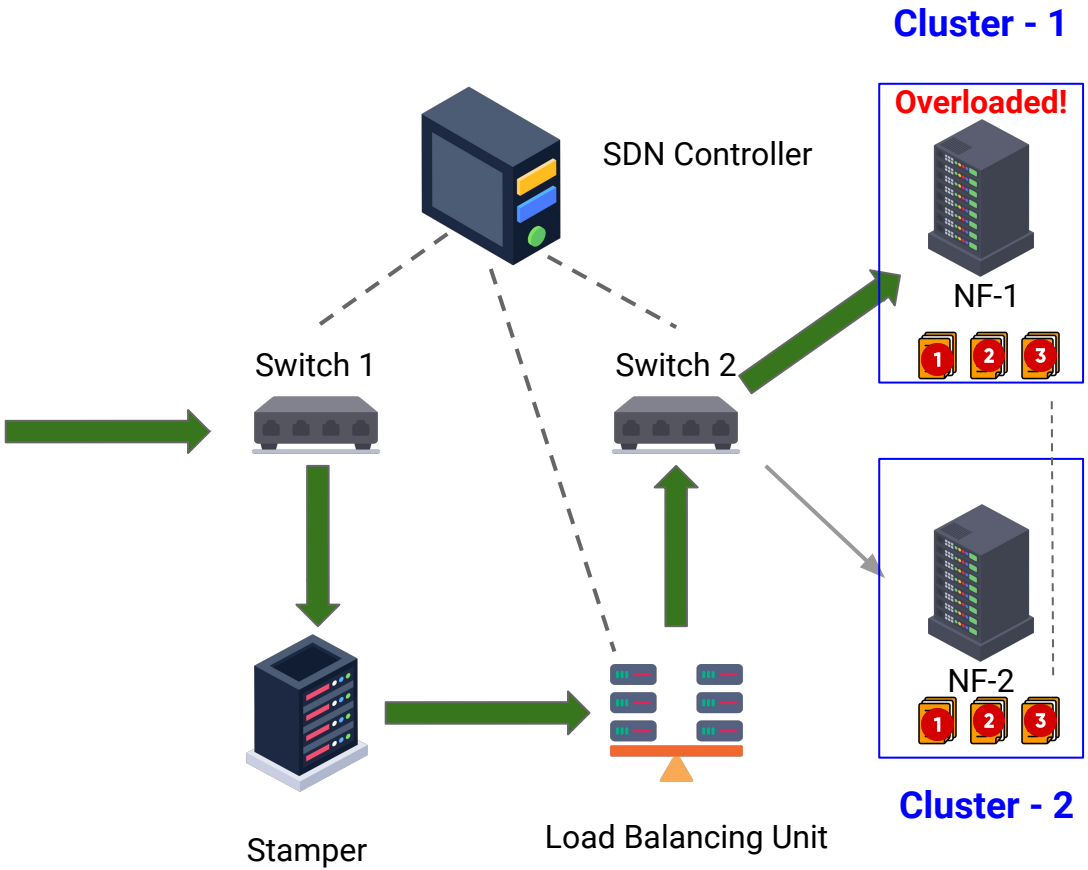


Flow Migration

Case 2

Overloaded NF: NF-1
Selected NF : NF-2 (Another cluster)

- NF-1 keeps buffering packets
- Load balancing unit selects existing NF-2
- NF-1 transfers all state information to NF-2

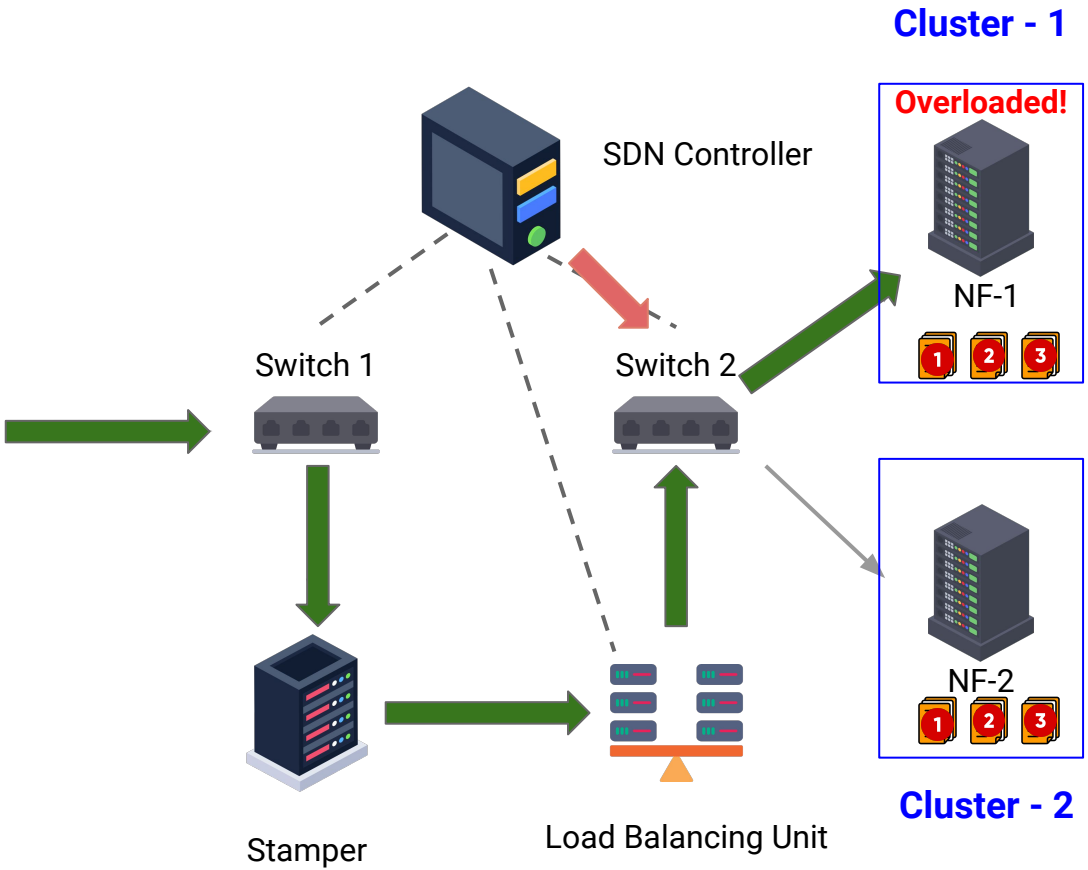


Flow Migration

Case 2

Overloaded NF: NF-1
Selected NF : NF-2 (Another cluster)

- NF-1 keeps buffering packets
- Load balancing unit selects existing NF-2
- NF-1 transfers all state information to NF-2
- SDN controller changes flow rule at switch

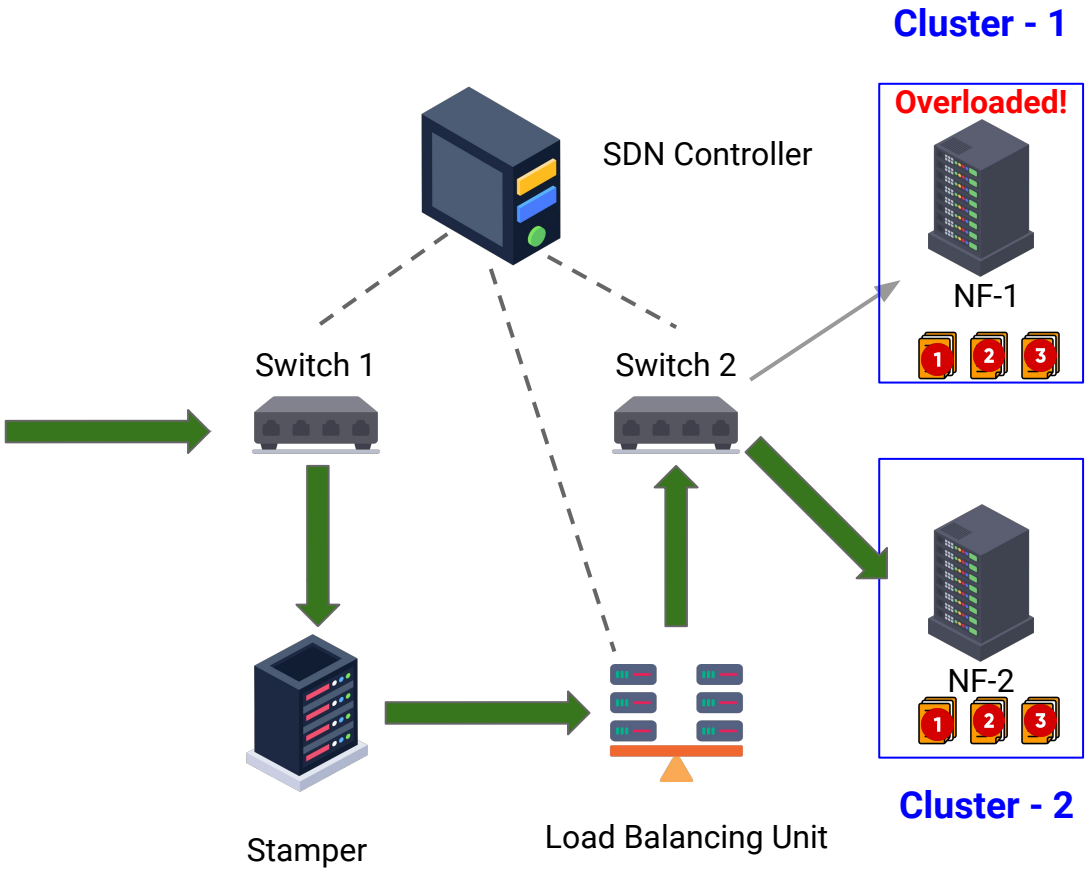


Flow Migration

Case 2

Overloaded NF: NF-1
Selected NF : NF-2 (Another cluster)

- NF-1 keeps buffering packets
- Load balancing unit selects existing NF-2
- NF-1 transfers all state information to NF-2
- SDN controller changes flow rule at switch
- NF-1 forwards buffered packets to Destination NF-2



Operations

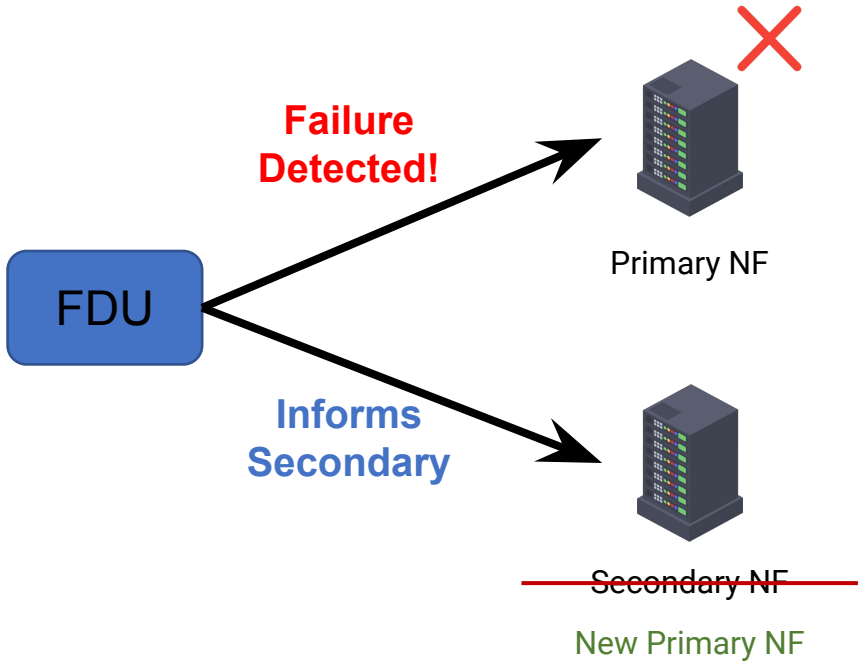
Normal Operation

Flow Migration Operation

Failover Operation

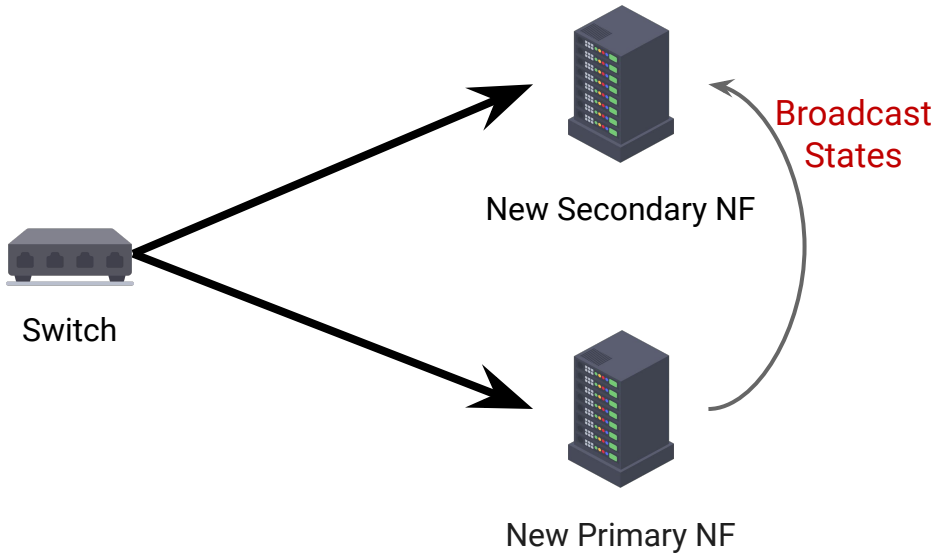
Failover Mechanism

- FDU detects primary NF failure
- Informs secondary about primary failure
- Secondary becomes the new primary



Failover Mechanism

- A new secondary NF is assigned
- The new primary shares its snapshot with the new secondary
- Continues to process packets like normal operation



Experiments and Results

Experimental Setup

- Hardware
  - CPU: Intel(R) Core(TM) i5-8400 CPU @ 2.80GHz, 6 Cores
  - RAM : 24.0 GB
- Software
  - Environment : Docker
  - Consensus Implementation : Hazelcast
- UDP Packets generated using **Packet Sender**

Notations

Parameter	Notation
Number of packets processed	#pkt
Packet Rate	R_{pkt}
Number of Threads	#th
Number of Clusters	#cluster
Total Number of NFs	#NF
Batch Size	S_{batch}

Results of Multithreading

Results of Multithreading

Scenario-1: Latency vs. Number of Threads

Results of Multithreading

Scenario-1: Latency vs. Number of Threads

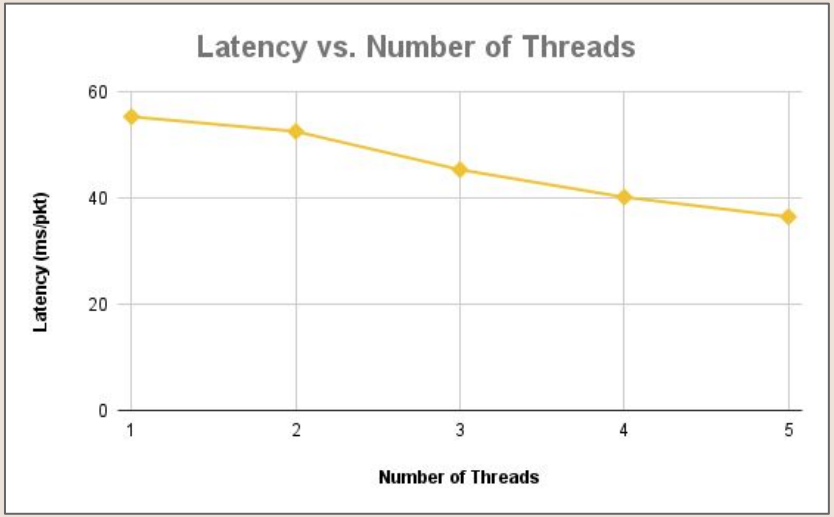
Parameters:

#pkt = 4000	R_{pkt} = 100 pkts/second	S_{batch} = 30	#NF = 1

Results of Multithreading

Scenario-1: Latency vs. Number of Threads

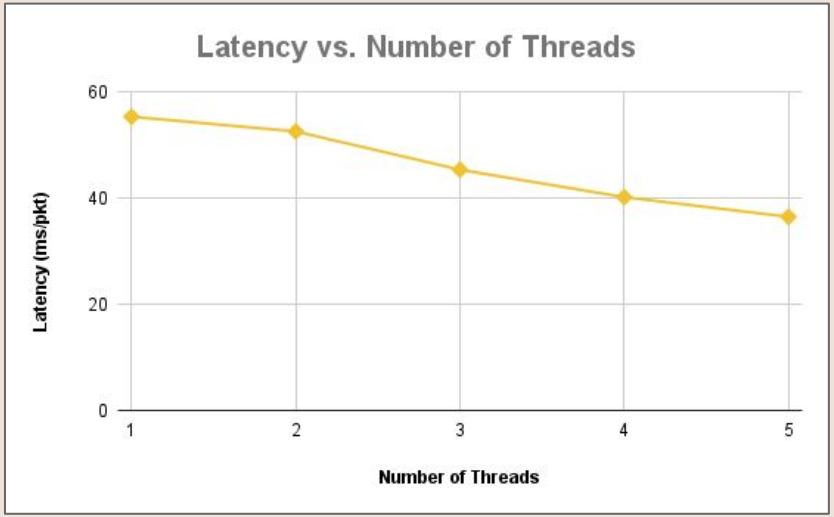
Parameters:	#pkt = 4000	R_{pkt} = 100 pkts/second	S_{batch} = 30	#NF = 1



Results of Multithreading

Scenario-1: Latency vs. Number of Threads

Parameters:	#pkt = 4000	R_{pkt} = 100 pkts/second	S_{batch} = 30	#NF = 1



- Multiple threads enables separate input buffer and packet processing unit for each flow
- Latency decreases as Number of Threads increases

Results of Multithreading

Scenario-2: Throughput vs. Number of Threads

Results of Multithreading

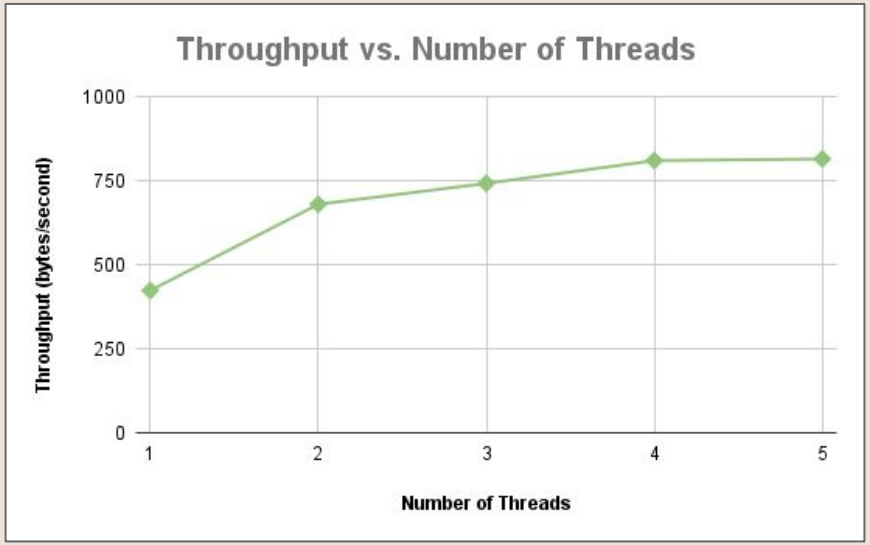
Scenario-2: Throughput vs. Number of Threads

Parameters:	#pkt = 4000	R_{pkt} = 100 pkts/second	S_{batch} = 30	#NF = 1

Results of Multithreading

Scenario-2: Throughput vs. Number of Threads

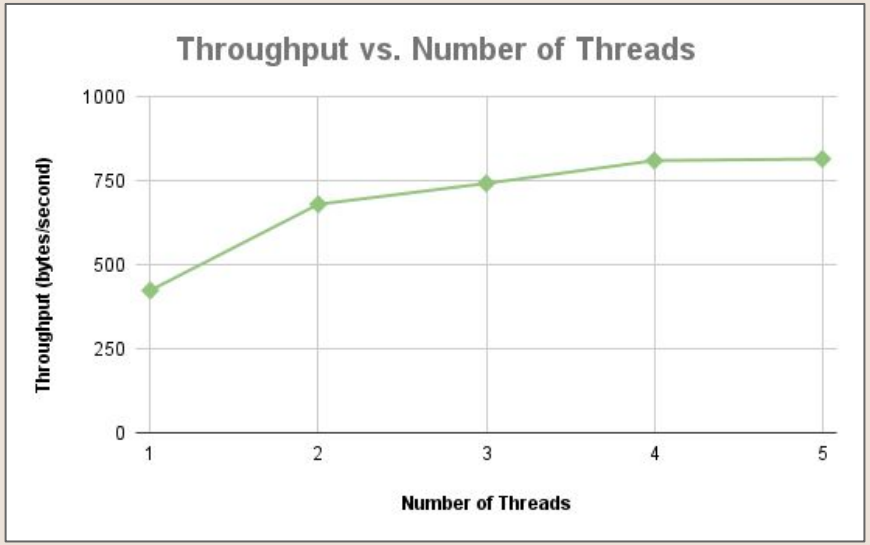
Parameters:	#pkt = 4000	R_{pkt} = 100 pkts/second	S_{batch} = 30	#NF = 1



Results of Multithreading

Scenario-2: Throughput vs. Number of Threads

Parameters:	#pkt = 4000	R_{pkt} = 100 pkts/second	S_{batch} = 30	#NF = 1



- Increased number of threads reduces the dependency between different flows
- So throughput increases as number of threads increases

Effects of Clustering

Effects of Clustering

Scenario-1: Throughput vs. Batch Size (No Clustering vs. Clustering)

Effects of Clustering

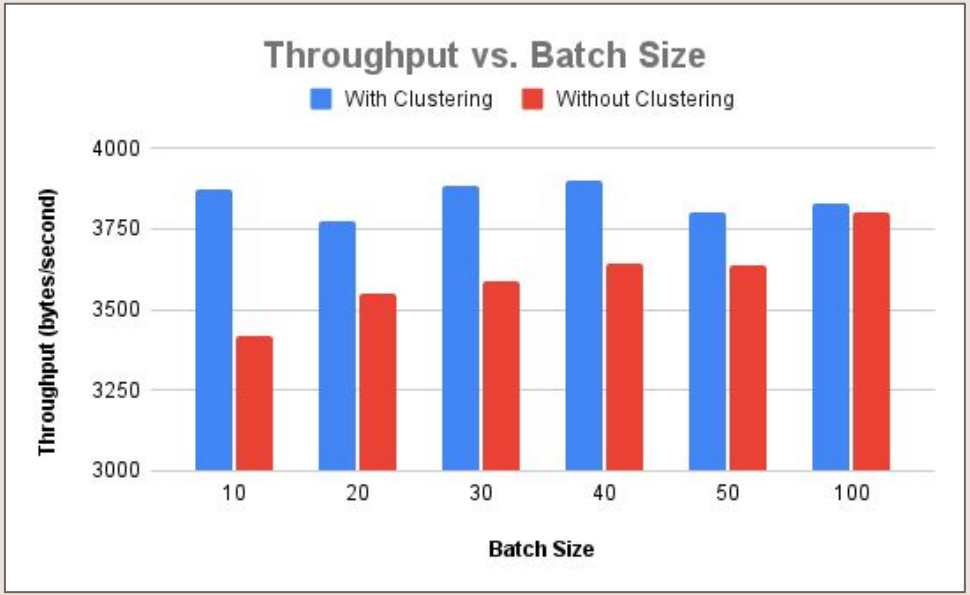
Scenario-1: Throughput vs. Batch Size (No Clustering vs. Clustering)

Parameters:	#pkt = 4000	R_{pkt} = 100 pkts/second	#cluster = 3	#NF = 12

Effects of Clustering

Scenario-1: Throughput vs. Batch Size (No Clustering vs. Clustering)

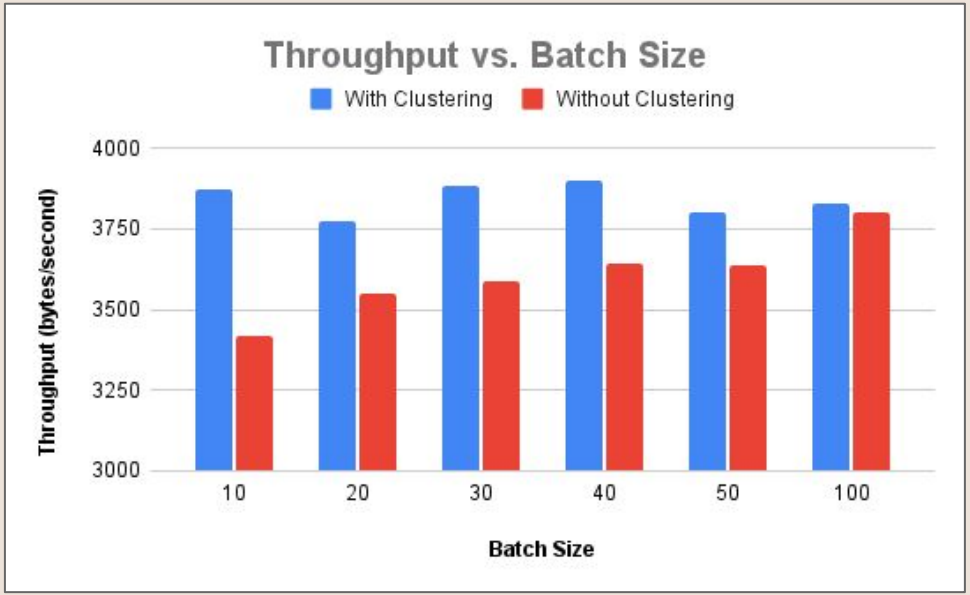
Parameters:	#pkt = 4000	R_{pkt} = 100 pkts/second	#cluster = 3	#NF = 12



Effects of Clustering

Scenario-1: Throughput vs. Batch Size (No Clustering vs. Clustering)

Parameters:	#pkt = 4000	R_{pkt} = 100 pkts/second	#cluster = 3	#NF = 12



- Without clustering, an NF has to send state information to more NFs (in this case, 11 other NFs)
- So the overall processing time is higher than that of clustered system.
- Clustered system has higher throughput than the not clustered variant.

Effects of Clustering

Scenario-2: Throughput vs. Packet Rate (No Clustering vs. Clustering)

Effects of Clustering

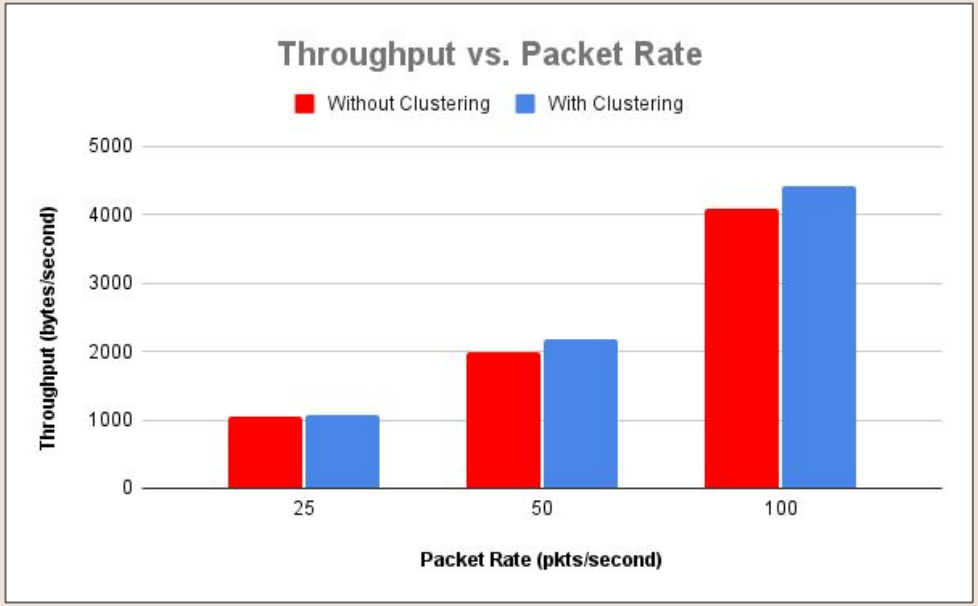
Scenario-2: Throughput vs. Packet Rate (No Clustering vs. Clustering)

Parameters:	#pkt = 4000	S_{batch} = 30	#cluster = 3	#NF = 12

Effects of Clustering

Scenario-2: Throughput vs. Packet Rate (No Clustering vs. Clustering)

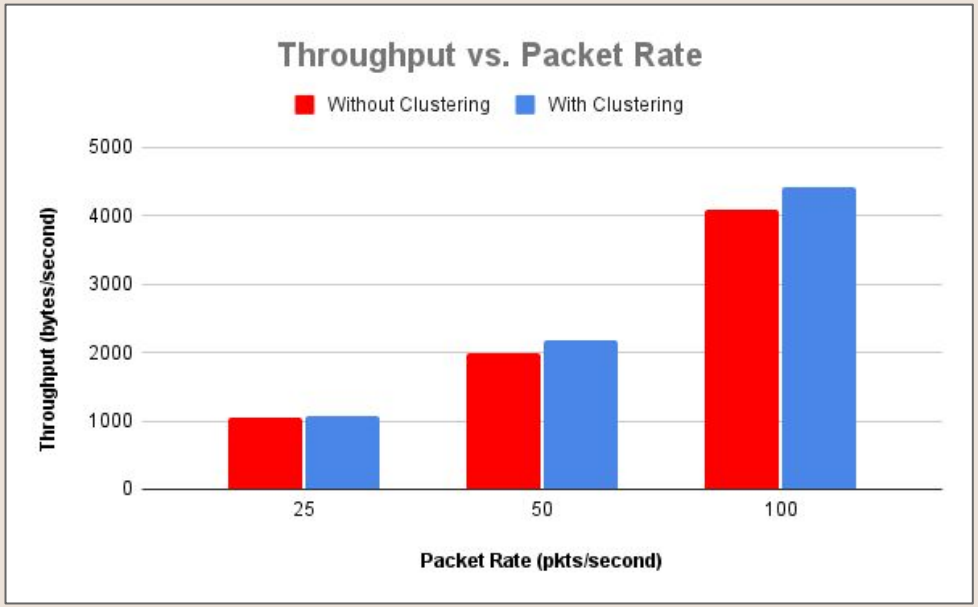
Parameters:	#pkt = 4000	S_{batch} = 30	#cluster = 3	#NF = 12



Effects of Clustering

Scenario-2: Throughput vs. Packet Rate (No Clustering vs. Clustering)

Parameters:	#pkt = 4000	S_{batch} = 30	#cluster = 3	#NF = 12

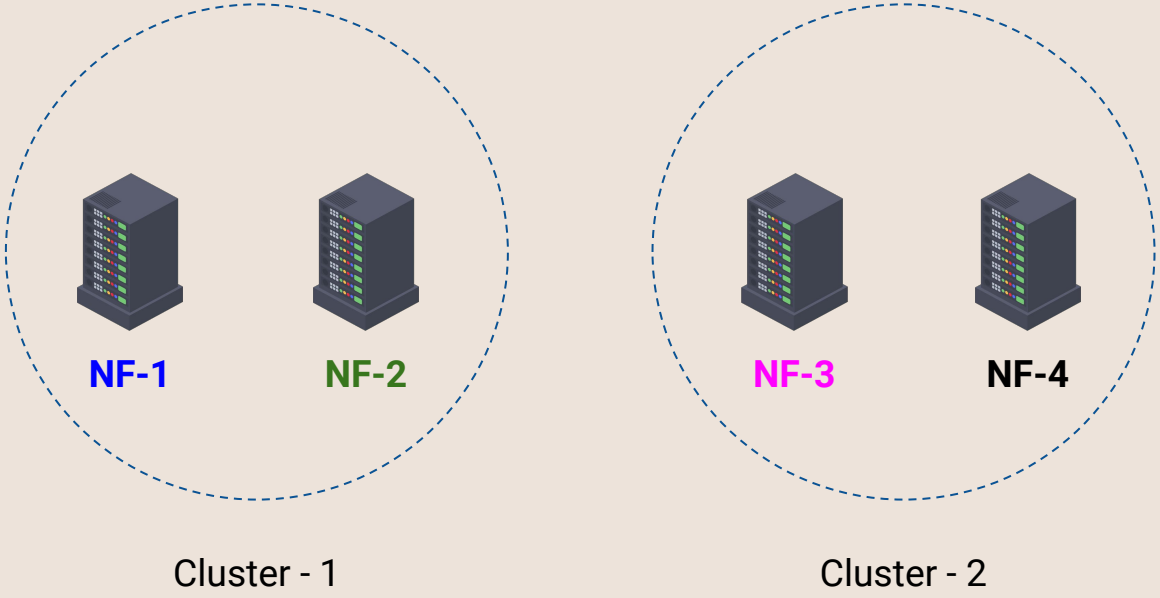


- The clustered system shows higher throughput than that of the non-clustered system.
- The difference is more evident in higher packet rate.

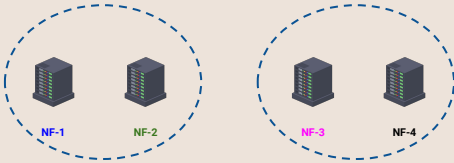
Results of State Migration Overhead in Clustering

Results of State Migration Overhead in Clustering

Testing Scenario:



Results of State Migration Overhead in Clustering

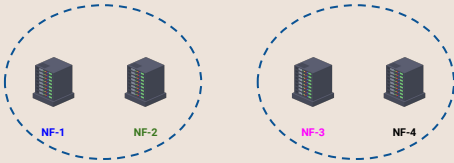


Scenario: Migration Overhead

Parameters:

S_{batch} = 30	R_{pkt} = 50 pkts/second	#cluster = 2	#NF = 4

Results of State Migration Overhead in Clustering



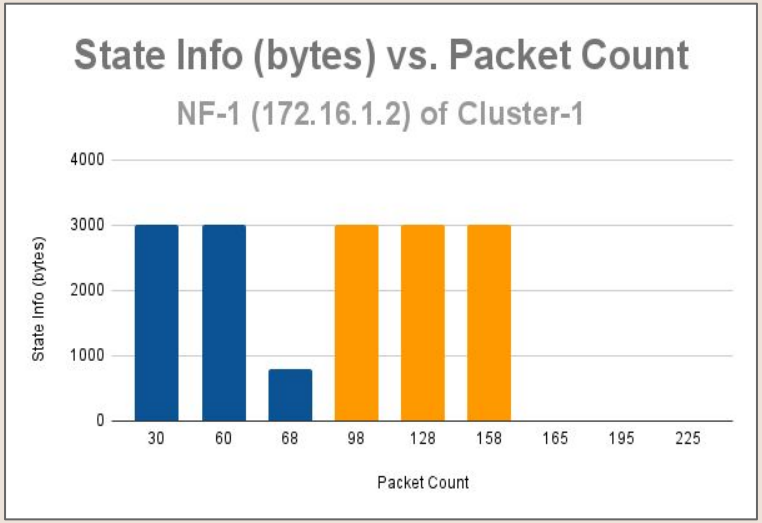
Scenario: Migration Overhead

Parameters:

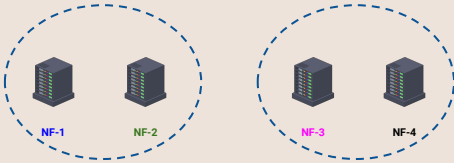
S_{batch} = 30	R_{pkt} = 50 pkts/second	#cluster = 2	#NF = 4

Legends:

	Overhead while sending state information to own cluster members		Overhead while receiving state information from own cluster members		Overhead while receiving state information from an NF of different cluster



Results of State Migration Overhead in Clustering



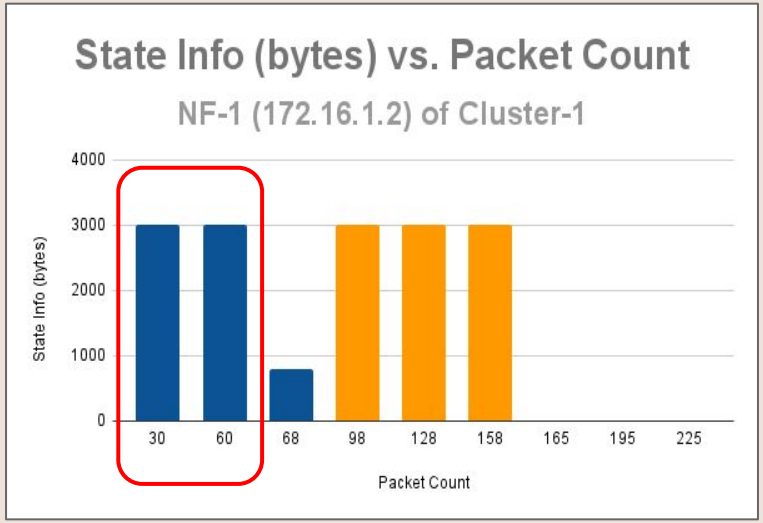
Scenario: Migration Overhead

Parameters:

S_{batch} = 30	R_{pkt} = 50 pkts/second	#cluster = 2	#NF = 4

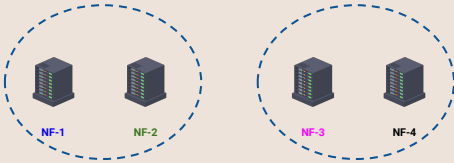
Legends:

	Overhead while sending state information to own cluster members		Overhead while receiving state information from own cluster members		Overhead while receiving state information from an NF of different cluster



- **NF-1** sends state information of **Flow-1** to other members of its own cluster after each batch (30 packets)

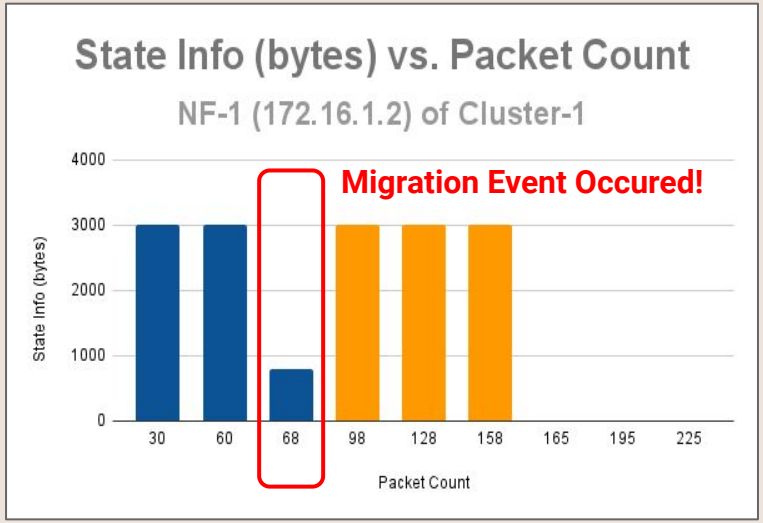
Results of State Migration Overhead in Clustering



Scenario: Migration Overhead

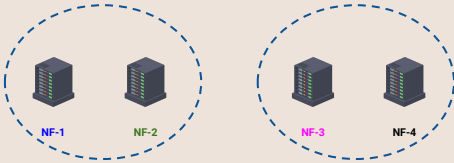
Parameters:	S_{batch} = 30	R_{pkt} = 50 pkts/second	#cluster = 2	#NF = 4

Legends:					
	Overhead while sending state information to own cluster members		Overhead while receiving state information from own cluster members		Overhead while receiving state information from an NF of different cluster



- **NF-1** sends state information of **Flow-1** to other members of its own cluster after each batch (30 packets)
- After **68 packets**, migration scenario occurs. **NF-1** sends state information of only 8 packets to other members of its own cluster.

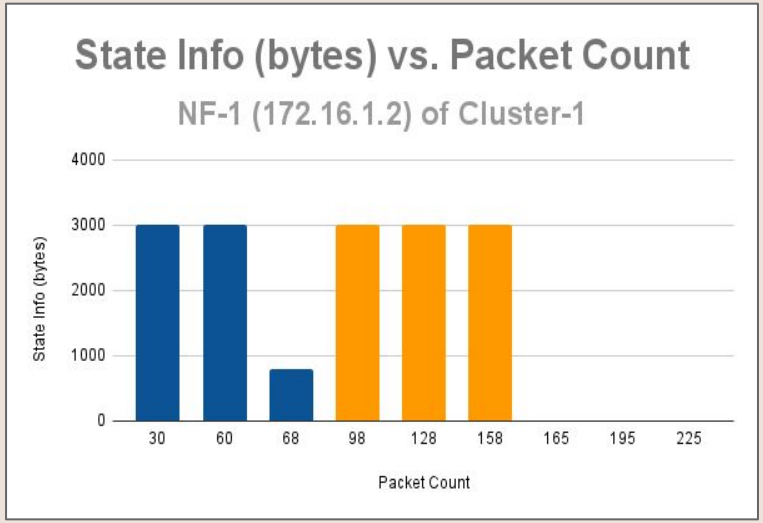
Results of State Migration Overhead in Clustering



Scenario: Migration Overhead

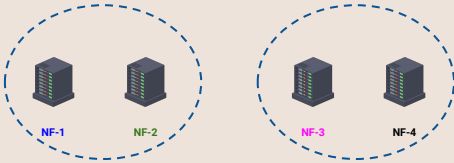
Parameters:	S_{batch} = 30	R_{pkt} = 50 pkts/second	#cluster = 2	#NF = 4

Legends:					
	Overhead while sending state information to own cluster members		Overhead while receiving state information from own cluster members		Overhead while receiving state information from an NF of different cluster



- **NF-1** sends state information of **Flow-1** to other members of its own cluster after each batch (30 packets)
- After **68 packets**, migration scenario occurs. **NF-1** sends state information of only 8 packets to other members of its own cluster.
- **Flow-1** is migrated to and now processed by **NF-2**.

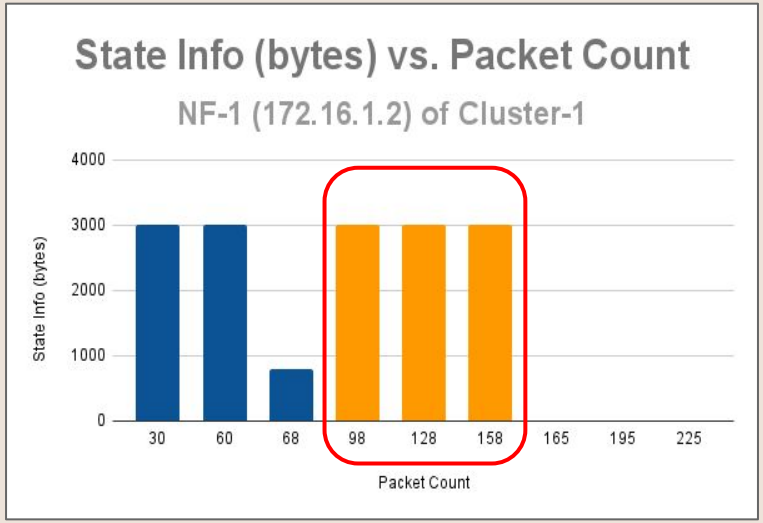
Results of State Migration Overhead in Clustering



Scenario: Migration Overhead

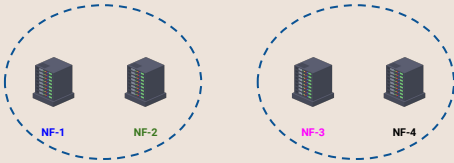
Parameters:	S_{batch} = 30	R_{pkt} = 50 pkts/second	#cluster = 2	#NF = 4

Legends:					
	Overhead while sending state information to own cluster members		Overhead while receiving state information from own cluster members		Overhead while receiving state information from an NF of different cluster



- **NF-1** sends state information of **Flow-1** to other members of its own cluster after each batch (30 packets)
- After **68 packets**, migration scenario occurs. **NF-1** sends state information of only 8 packets to other members of its own cluster.
- **Flow-1** is migrated to and now processed by **NF-2**.
- **NF-1** continues to receive state information of **Flow-1** from **NF-2** after each batch.

Results of State Migration Overhead in Clustering



Scenario: Migration Overhead

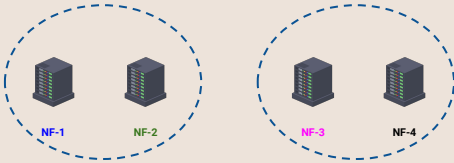
Parameters:

S_{batch} = 30	R_{pkt} = 50 pkts/second	#cluster = 2	#NF = 4

Legends:

	Overhead while sending state information to own cluster members		Overhead while receiving state information from own cluster members		Overhead while receiving state information from an NF of different cluster

Results of State Migration Overhead in Clustering



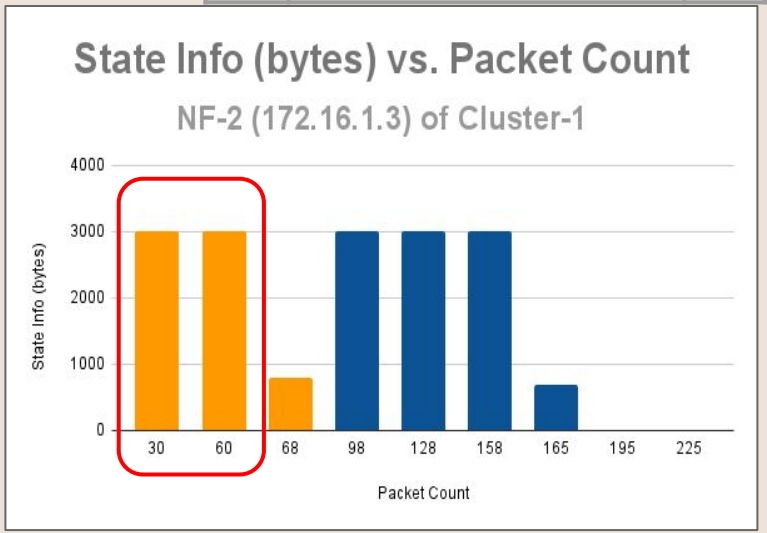
Scenario: Migration Overhead

Parameters:

S_{batch} = 30	R_{pkt} = 50 pkts/second	#cluster = 2	#NF = 4

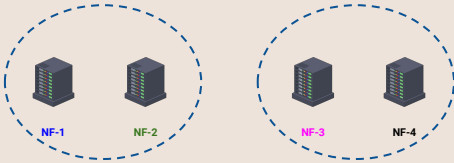
Legends:

	Overhead while sending state information to own cluster members		Overhead while receiving state information from own cluster members		Overhead while receiving state information from an NF of different cluster



- **NF-2** receives state information of **Flow-1** from **NF-1** after each batch processed by **NF-1** (30 packets)

Results of State Migration Overhead in Clustering



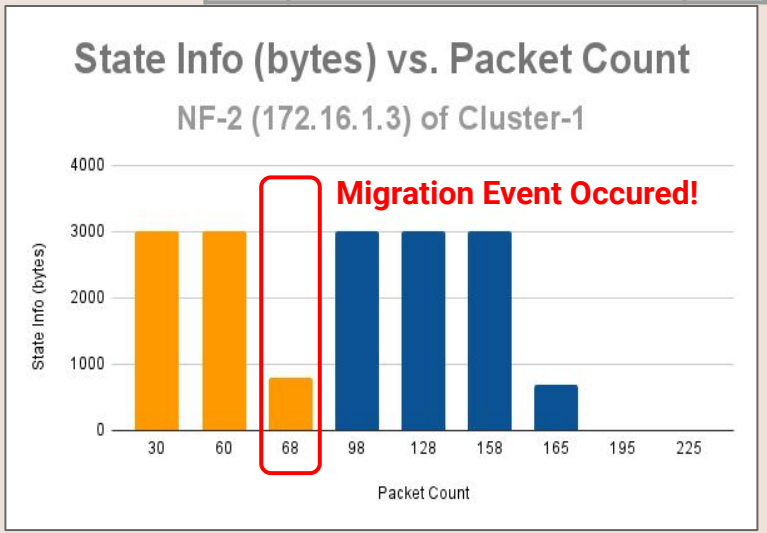
Scenario: Migration Overhead

Parameters:

S_{batch} = 30	R_{pkt} = 50 pkts/second	#cluster = 2	#NF = 4

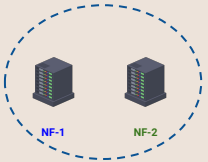
Legends:

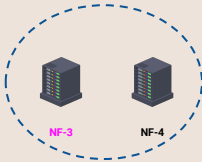
	Overhead while sending state information to own cluster members		Overhead while receiving state information from own cluster members		Overhead while receiving state information from an NF of different cluster



- NF-2 receives state information of Flow-1 from NF-1 after each batch processed by NF-1 (30 packets)
- After NF-1 processes **68 packets**, migration scenario occurs. NF-2 receives state information of only 8 packets from NF-1. Flow-1 is now processed by NF-2.

Results of State Migration Overhead in Clustering





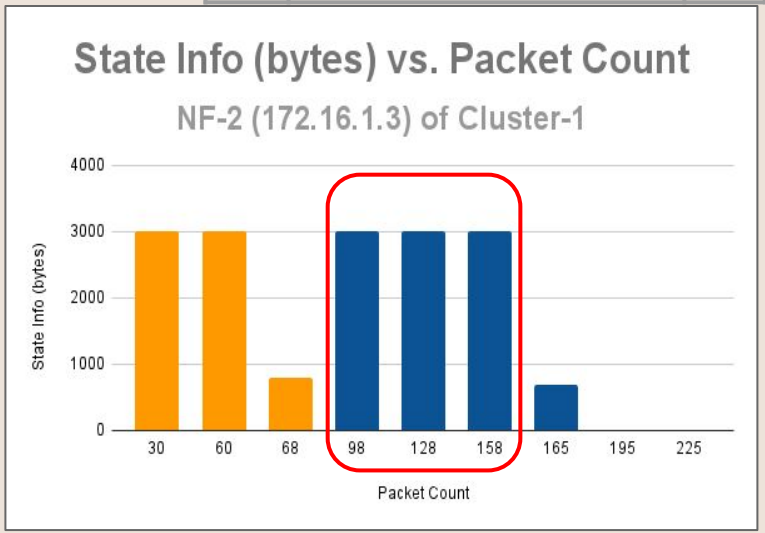
Scenario: Migration Overhead

Parameters:

S_{batch} = 30	R_{pkt} = 50 pkts/second	#cluster = 2	#NF = 4

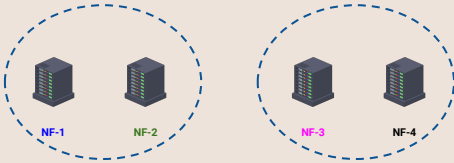
Legends:

	Overhead while sending state information to own cluster members		Overhead while receiving state information from own cluster members		Overhead while receiving state information from an NF of different cluster



- NF-2 receives state information of Flow-1 from NF-1 after each batch processed by NF-1 (30 packets)
- After NF-1 processes **68 packets**, migration scenario occurs. NF-2 receives state information of only 8 packets from NF-1. Flow-1 is now processed by NF-2.
- NF-2 continues to send state information within cluster after each batch size.

Results of State Migration Overhead in Clustering



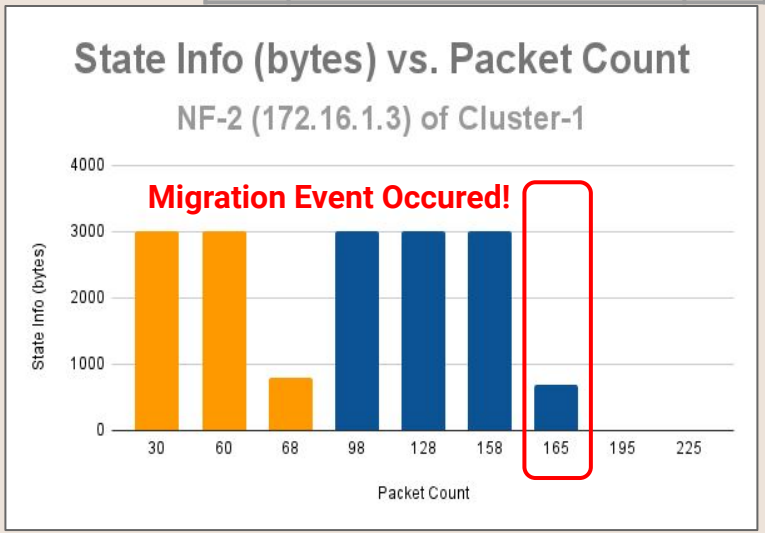
Scenario: Migration Overhead

Parameters:

S_{batch} = 30	R_{pkt} = 50 pkts/second	#cluster = 2	#NF = 4

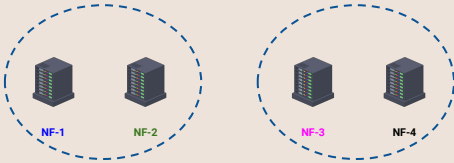
Legends:

	Overhead while sending state information to own cluster members		Overhead while receiving state information from own cluster members		Overhead while receiving state information from an NF of different cluster



- NF-2 receives state information of Flow-1 from NF-1 after each batch processed by NF-1 (30 packets)
- After NF-1 processes **68 packets**, migration scenario occurs. NF-2 receives state information of only 8 packets from NF-1. Flow-1 is now processed by NF-2.
- NF-2 continues to send state information within cluster after each batch size.
- After **165 packets**, inter-cluster migration event occurs. NF-2 send state information of only 7 packets within its cluster. Flow-1 is migrated and now processed by NF-3, which is in a different cluster than NF-1 and NF-2.

Results of State Migration Overhead in Clustering



Scenario: Migration Overhead

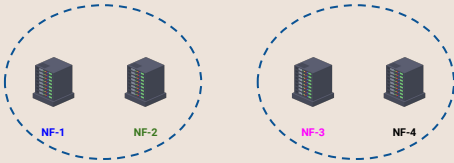
Parameters:

S_{batch} = 30	R_{pkt} = 50 pkts/second	#cluster = 2	#NF = 4

Legends:

	Overhead while sending state information to own cluster members		Overhead while receiving state information from own cluster members		Overhead while receiving state information from an NF of different cluster

Results of State Migration Overhead in Clustering



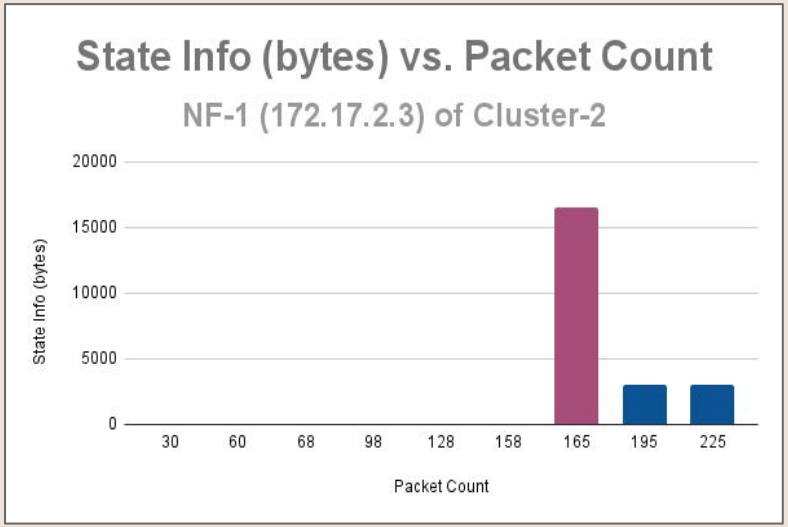
Scenario: Migration Overhead

Parameters:

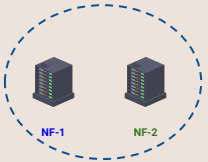
S_{batch} = 30	R_{pkt} = 50 pkts/second	#cluster = 2	#NF = 4

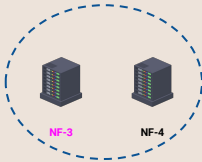
Legends:

	Overhead while sending state information to own cluster members		Overhead while receiving state information from own cluster members		Overhead while receiving state information from an NF of different cluster



Results of State Migration Overhead in Clustering





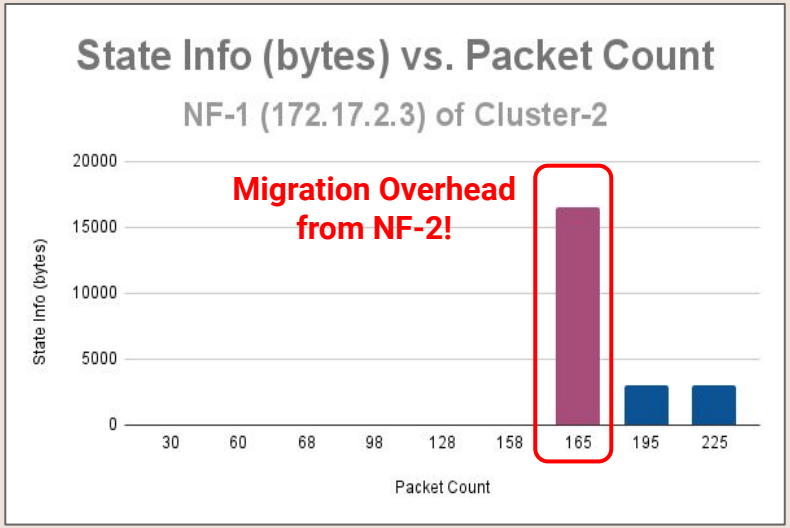
Scenario: Migration Overhead

Parameters:

S_{batch} = 30	R_{pkt} = 50 pkts/second	#cluster = 2	#NF = 4

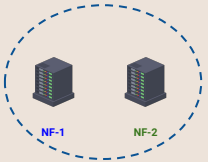
Legends:

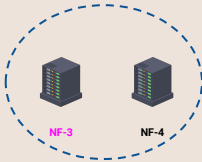
	Overhead while sending state information to own cluster members		Overhead while receiving state information from own cluster members		Overhead while receiving state information from an NF of different cluster



- NF-3 is in a different cluster than NF-1 and NF-2. So when Flow-1 is migrated, NF-3 receives the full state information of all 165 packets processed of Flow-1 from NF-2.

Results of State Migration Overhead in Clustering





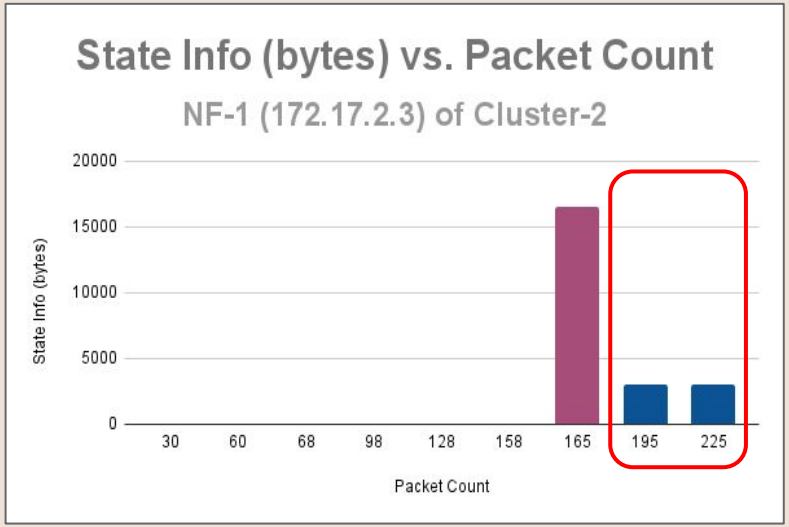
Scenario: Migration Overhead

Parameters:

S_{batch} = 30	R_{pkt} = 50 pkts/second	#cluster = 2	#NF = 4

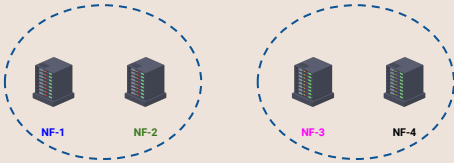
Legends:

	Overhead while sending state information to own cluster members		Overhead while receiving state information from own cluster members		Overhead while receiving state information from an NF of different cluster



- NF-3 is in a different cluster than NF-1 and NF-2. So when Flow-1 is migrated, NF-3 receives the full state information of all 165 packets processed of Flow-1 from NF-2.
- NF-3 continues to send state information to other members of its own cluster after processing every batch (30 packets)

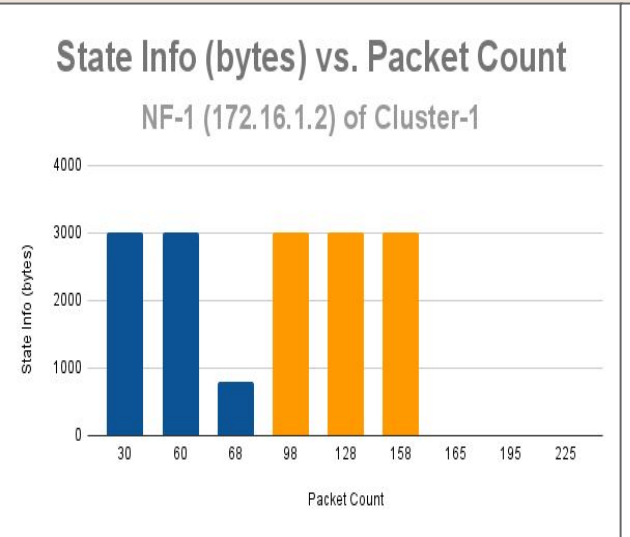
Results of State Migration Overhead in Clustering

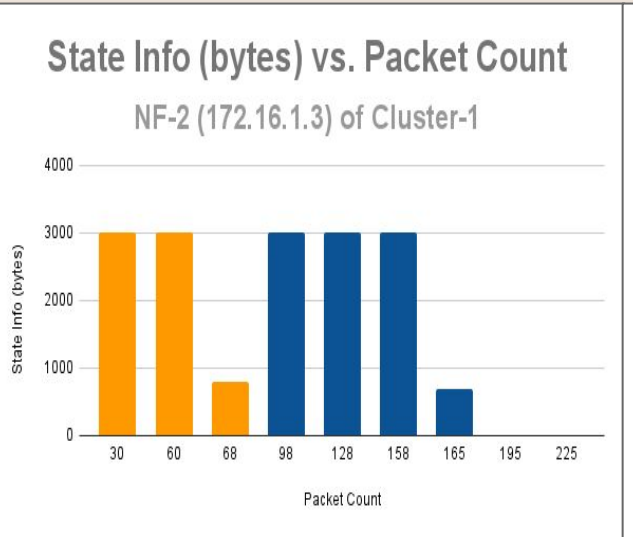


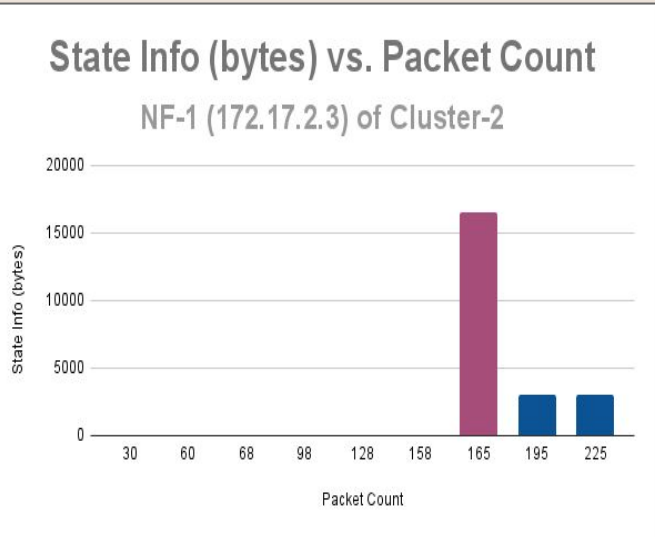
Scenario: Migration Overhead

Parameters:	S_{batch} = 30	R_{pkt} = 50 pkts/second	#cluster = 2	#NF = 4

Legends:			
Overhead while sending state information to own cluster members	Overhead while receiving state information from own cluster members	Overhead while receiving state information from an NF of different cluster







Results of Load Balancing

Results of Load Balancing

Scenario-1: Latency vs. Packet Rate (No Load Balancing vs. Load Balancing)

Results of Load Balancing

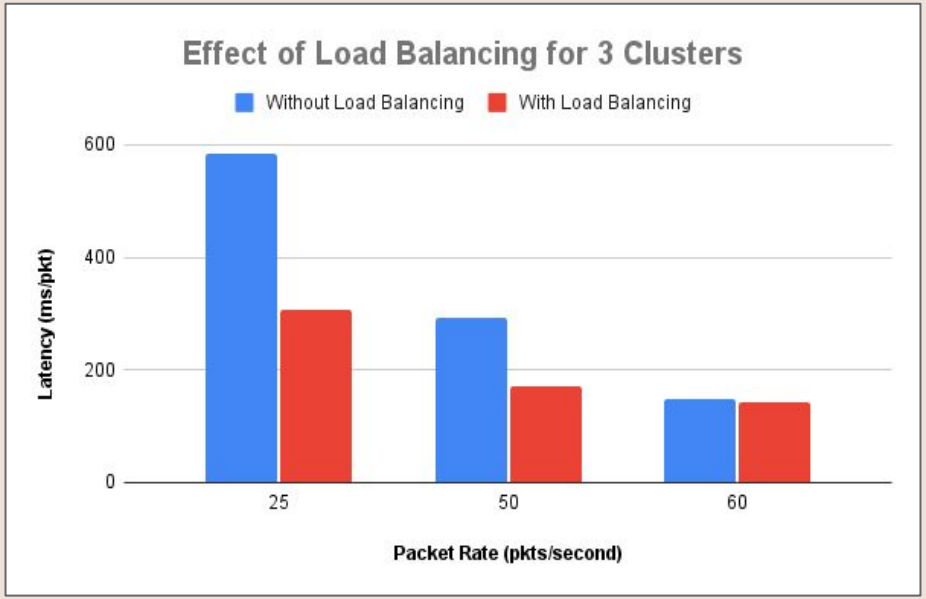
Scenario-1: Latency vs. Packet Rate (No Load Balancing vs. Load Balancing)

Parameters:	#pkt = 4000	S_{batch} = 30	#cluster = 3	#NF = 9

Results of Load Balancing

Scenario-1: Latency vs. Packet Rate (No Load Balancing vs. Load Balancing)

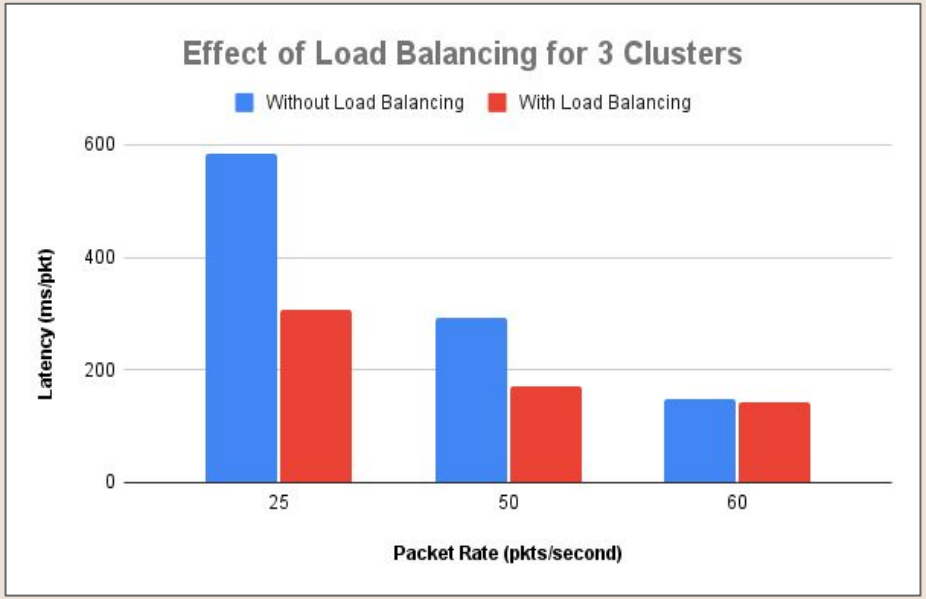
Parameters:	#pkt = 4000	S_{batch} = 30	#cluster = 3	#NF = 9



Results of Load Balancing

Scenario-1: Latency vs. Packet Rate (No Load Balancing vs. Load Balancing)

Parameters:	#pkt = 4000	S_{batch} = 30	#cluster = 3	#NF = 9



- With load balancing, no NF is overloaded for long.
- So NFs can process packets faster.
- A system with load balancing has lower latency than that of a system without any load balancing implemented.

Contribution and Future Works

Our Contributions

- Reduced overhead in state migration
- Applied multithreading in packet processing
- Achieved reduced overall latency by implementing an enhanced cluster-based load balancing technique

Future Work

- Achieve strict consistency of global states in absence of central controller
- Instead of network wide global state update, try cluster-based global states
- Try different load balancing techniques
- Vary different consensus algorithms

References

- A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, “Opennf: Enabling innovation in network function control,” ACM SIGCOMM Computer Communication Review, vol. 44, no. 4, pp. 163–174, 2014
- M. Kablan, A. Alsudais, E. Keller, and F. Le, “Stateless network functions: Breaking the tight coupling of state and processing,” in 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), pp. 97–112, 2017
- S. Rajagopalan, D. Williams, and H. Jamjoom, “Pico replication: A high availability framework for middleboxes,” in Proceedings of the 4th annual Symposium on Cloud Computing, pp. 1–15, 2013
- J. Sherry, P. X. Gao, S. Basu, A. Panda, A. Krishnamurthy, C. Maciocco, M. Manesh, J. Martins, S. Ratnasamy, L. Rizzo, et al., “Rollback-recovery for middleboxes,” in Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, pp. 227–240, 2015.
- M. M. Shahriyar, G. Saha, B. Bhattacharjee, and R. Reaz, “DEFT: Distributed, Elastic,and Fault-tolerant State Management of Network Functions,” IEEE Access Manuscript submitted for publication

Thank You!