Split / Merge
System Support for Elastic Execution in Virtual Middleboxes

Shriram RAJAGOPALAN IBM Research & UBC
Dan WILLIAMS IBM Research
Hani JAMJOOM IBM Research
Andrew WARFIELD UBC
The Problem
Elastic Applications Need Elastic Middleboxes

Software Defined Network

IDS/IPS  Firewall  LB  VPN  Accelerator

Flows

Elastic Application Tier

Edge Switch

Clients
Hotspots Cannot be Alleviated Quickly

When M1 is overloaded, provision more middleboxes to serve new flows
Scaling Inefficiencies Lead to Poor Utilization

In the worst case, active flows in each instance delay scale-in
The Insight
Flow State is Naturally Partitioned
Enabling Elasticity in Virtual Middleboxes

Dynamic partitioning of flow states among “replicas” enables elastic execution.
Understanding the State Inside a Middlebox

Input

( Flows )

Output

Application Logic
Understanding the State Inside a Middlebox

![Diagram of Middlebox VM with Application Logic and Flow Table](image)

- **Application Logic**
- **Flow Table**
  - **Key**: 5-tuple
  - **Value**: [Flow State]

Partitionable among replicas
Understanding the State Inside a Middlebox

Flow Table

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-tuple</td>
<td>[Flow State]</td>
</tr>
</tbody>
</table>

Threshold counters | Non-critical statistics

Partitionable among replicas

May be shared among replicas (coherent)
Understanding the State Inside a Middlebox

- **Input (Flows)**: Flows are input into the Middlebox.
- **Application Logic**:
  - Caches
  - Other processes
  - ... (other components)
  - Threshold counters
  - Non-critical statistics
- **Flow Table**:
  - Key: 5-tuple
  - Value: [Flow State]
- **Output**: The output flows are processed through the Middlebox's logic and state.

- **Partitionable among replicas**:
  - 5-tuple (Flow State)

- **Internal to a replica (ephemeral)**:
  - Caches, Other processes, Threshold counters

- **May be shared among replicas (coherent)**:
  - Non-critical statistics

- **Other processes**:
  - May be shared among replicas (coherent)

---

12
Our Contribution
Split/Merge: A State-Centric Approach to Elasticity

Virtual Network Interface

VM

Ephemeral

Coherent

Partitionable (Flow States)
Split/Merge: A State-Centric Approach to Elasticity

Split

Replica 1

Replica 2

Replica 3

Virtual Network Interface

VM

Ephemeral

Coherent

Partitionable
(Flow States)
Split/Merge: A State-Centric Approach to Elasticity

**Split**

- **Virtual Network Interface**
- **Ephemeral**
- **Coherent**
- **Partitionable (Flow States)**

**Unchanged Interfaces**

**Split**

- **Replica 1**
- **Replica 2**
- **Replica 3**

**Merge**

- **Replica 1**
- **Replica 2+3**

Coherency is maintained
Implementation
FreeFlow

- A VMM based runtime that provides Split/Merge abstraction to applications

- Developers modify application code to annotate flow state

- FreeFlow takes care of the rest!
FreeFlow: A Split/Merge Implementation

Replica 1

VM

Ephemeral

Coherent

Partitionable (Flow States)

Replica 2

VM

VMM

Traffic to Middlebox

Flow 1
Flow 2

1

2
FreeFlow: A Split/Merge Implementation

- Need to manage application state

Diagram:

- Replica 1
  - VM
  - FreeFlow Library
  - Flow 1
  - VMM

- Replica 2
  - VM
  - FreeFlow Library
  - Flow 2
  - VMM

Traffic to Middlebox

Flow 1
Flow 2
FreeFlow: A Split/Merge Implementation

- Need to manage application state
- Need to ensure flows are routed to the correct replica
FreeFlow: A Split/Merge Implementation

- Need to manage application state
- Need to ensure flows are routed to the correct replica
- Need to decide when to split or merge a replica
Annotating State using FreeFlow API

Partitioned State API

```python
create_flow(flow_key, size)
delete_flow(flow_key)
flow_state get_flow(flow_key)
put_flow(flow_key)

flow_timer(flow_key, timeout, callback)
```

Coherent State API

```python
create_shared(key, size, cb)
delete_shared(key)

state get_shared(key, flags)  // synch | pull | local
put_shared(key, flags)  // synch | push | local
```
Forwarding Flows Correctly using OpenFlow

1.1.1.1 (de:ad:be:ef:ca:fe)

Flow Table

1.1.1.1 (de:ad:be:ef:ca:fe)

Flow Table

OpenFlow Switch

Virtual Switch

OpenFlow Table

<table>
<thead>
<tr>
<th>&lt;a&gt;</th>
<th>port 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OpenFlow Table

<table>
<thead>
<tr>
<th>&lt;b&gt;</th>
<th>port 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OpenFlow Table

<table>
<thead>
<tr>
<th>&lt;c&gt;</th>
<th>port 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OpenFlow Table

<table>
<thead>
<tr>
<th>&lt;b&gt;</th>
<th>port 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Virtual Switch

Middlebox Replica 1

Port 1

Port 1

Port 1

Port 1
Flow Migration

Migrating <b> from replica 2 to replica 1

1.1.1.1 (de:ad:be:ef:ca:fe)

Flow Table

Middlebox Replica 1

Flow Table

Middlebox Replica 2

OpenFlow Table

Virtual Switch

OpenFlow Table

Virtual Switch

Migrating <b> from replica 2 to replica 1
Flow Migration

Suspend flow & buffer packets

1.1.1.1 (de:ad:be:ef:ca:fe)

OpenFlow Table

<table>
<thead>
<tr>
<th>&lt;a&gt;</th>
<th>port 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;b&gt;</td>
<td>controller</td>
</tr>
<tr>
<td>&lt;c&gt;</td>
<td>port 2</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Virtual Switch

Port 1

Middlebox Replica 1

Flow Table

Port 1

1.1.1.1 (de:ad:be:ef:ca:fe)

OpenFlow Table

<table>
<thead>
<tr>
<th>&lt;c&gt;</th>
<th>port 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Virtual Switch

Port 1

Middlebox Replica 2

Flow Table

Port 1

Port 2
Flow Migration

Move flow state to target

1.1.1.1 (de:ad:be:ef:ca:fe)

OpenFlow Table

<table>
<thead>
<tr>
<th>&lt;a&gt;</th>
<th>port 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Virtual Switch

OpenFlow Table

<table>
<thead>
<tr>
<th>&lt;a&gt;</th>
<th>port 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

OpenFlow Switch

<table>
<thead>
<tr>
<th>&lt;b&gt;</th>
<th>controller</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;c&gt;</td>
<td>port 2</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Middlebox Replica 1

<table>
<thead>
<tr>
<th>&lt;a&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Flow Table

Middlebox Replica 2

<table>
<thead>
<tr>
<th>&lt;c&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

1.1.1.1 (de:ad:be:ef:ca:fe)

Move flow state to target
Flow Migration

Release buffer & resume flow

OpenFlow Table

<table>
<thead>
<tr>
<th>&lt;a&gt;</th>
<th>port 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;b&gt;</td>
<td>port 1</td>
</tr>
<tr>
<td>&lt;c&gt;</td>
<td>port 2</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Virtual Switch

OpenFlow Switch

Flow Table

1.1.1.1 (de:ad:be:ef:ca:fe)

Middlebox Replica 1

1.1.1.1 (de:ad:be:ef:ca:fe)

Middlebox Replica 2

1.1.1.1 (de:ad:be:ef:ca:fe)
Managing Coherent State

create_shared(key, size, cb)
delete_shared(key)

state get_shared(key, flags)  // synch | pull | local
put_shared(key, flags)        // synch | push | local
Managing Coherent State

Strong Consistency

```c
create_shared("foo", 4, NULL)
while (1)
    process_packet()
    p_foo = get_shared("foo", synch)
    val = (*p_foo)++
    put_shared("foo", synch)
    if (val > threshold)
        bar()
```

Distributed lock for every update

Middlebox applications rarely need strong consistency!
Managing Coherent State

Eventual Consistency

```c
create_shared("foo", 4, merge_fn)
while (1)
    process_packet()
    p_foo = get_shared("foo", local)
    val = (*p_foo)++
    put_shared("foo", local)
    if (val > threshold)
        bar()
        put_shared("foo", push)
```
Evaluation
Evaluation Overview

- Eliminating hotspots during scale-out
- Fast and efficient scale-in
- Split/Merge Bro during a load burst
Hotspots Cannot be Alleviated Quickly

When M1 is overloaded, provision more middleboxes to serve new flows
Eliminating Hotspots by Shedding Load

![Graph showing max latency over time with and without FreeFlow]

- **w/ FreeFlow**
  - Max Latency (ms)
  - Time (s)

- **w/o FreeFlow**
  - Max Latency (ms)
  - Time (s)
Eliminating Hotspots by Shedding Load
Scaling-In a Deployment: Best Case Scenario

Apps

Virtual Middleboxes

M1  M2  M3  M4
Scaling-In a Deployment: Best Case Scenario
Scaling-In: Best Case Scenario

![Graph showing Avg System Utilization (%) over Time (s) with a best case scenario indicated]

- **Avg System Utilization (%)**
- **Time (s)**
Scaling-In using *kill*: Worst Case Scenario
Scaling-In using *kill*: Worst Case Scenario
Scaling-In using *kill*: Worst Case Scenario

In the worst case, active flows in each instance delay scale-in
Scaling-In using *kill*: Slow & Inefficient
Scaling-In using *merge* : Worst Case Scenario

![Diagram showing the concept of scaling-in using merge for worst case scenario.](image)
Scaling-In using *merge*: Worst Case Scenario
Scaling-In using *merge*: Worst Case Scenario
Scaling-In using *merge*: Worst Case Scenario

FreeFlow consolidates state and flows from multiple replicas into one.
Scaling-In using *merge*: Fast & Efficient

![Graph showing system utilization over time with best and worst cases, including scenarios with and without FreeFlow. The graph highlights the efficiency and speed of the merging process.]
Splitting & Merging Bro IDS

- Ported the Event Engine to FreeFlow
- Support for UDP, TCP/HTTP protocols
- SQL Injection Detection plugin
Handling a Load Burst

![Graph showing load burst and attacks detected over time]

Attacks Detected (%) vs. Time (s)

Web Servers

Load Burst

SQL Injection Attacks

Bro
Without enough capacity to handle the load burst, the system performance degrades severely.
Handling a Load Burst: Pre-Scaled

Two instances are provisioned apriori, enough to handle a load burst, if any
Handling a Load Burst: Pre-Scaled

Load burst has no impact on system performance, as there is enough capacity to handle the load.
Handling a Load Burst: Split/Merge

One replica handles the load well, before the load burst
Handling a Load Burst: Split/Merge

When load burst starts, the Orchestrator splits the replica and rebalances the load.
Handling a Load Burst: Split/Merge

With the load rebalanced, performance returns to normal
When system utilization drops after the load burst, the Orchestrator *merges* the two replicas.
Handling a Load Burst: Split/Merge

- Unscaled
- Pre-scaled
- Dynamically scaled with FreeFlow

Web Servers

FreeFlow

Load Burst

Attacks Detected (%)

Time (s)

SQL Injection Attacks
Flow Migration Overhead - TCP

Throughput (Mbps/s) vs Time (s)

- Baseline
- FreeFlow

Flow Migration
Flow Migration Overhead - UDP

Packet Latency (ms)

Throughput (Mbits) [50ms window]

Time (s)