OpenNF High Availability: Implementation

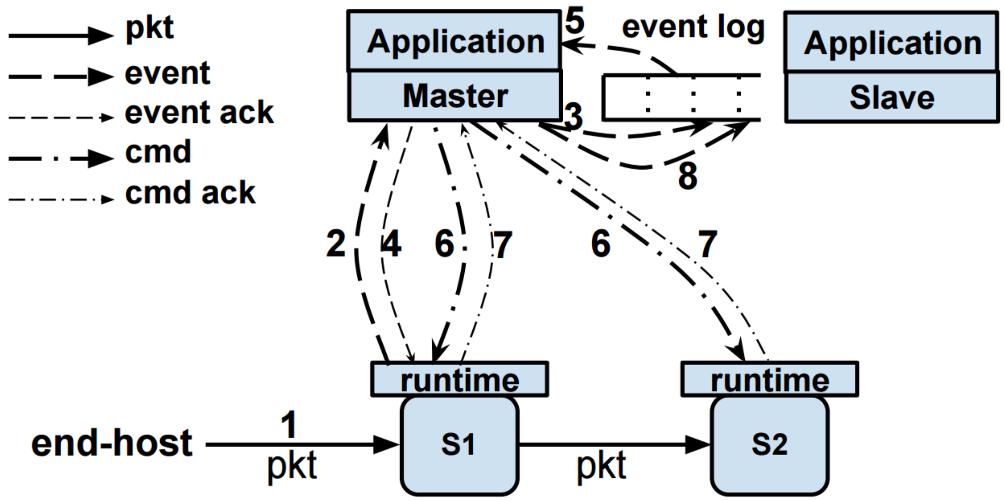
1. Background + Motivation

a. OpenNF is a control-plane architecture built on top of SDN.
b. What services does OpenNF provide over SDN?
   i. SDN networks only provide communication between controllers and switches (using OpenFlow).
   ii. **OpenNF Contribution**: OpenNF adds a control plane architecture between controllers and middleboxes (NFs).
   iii. **OpenNF Contribution:** OpenNF adds mechanisms to perform state transfer operation between middleboxes.
c. Existing approaches to SDN High Availability typically deal with:
   i. Failures in the SDN controller-switch control plane
      1. OpenNF adds NFs into the SDN control plane
      2. **Implementation Goal:** OpenNF HA must account for failures at all three points in the environment: controller, network and NFs.
   ii. General NF failures
      1. In OpenNF, NFs are part of the control plane now, and we have new mechanisms that allow state transfer operations between different NFs.
      2. **Implementation Goal:** OpenNF HA needs to address a unique special-case failure: *failure in the middle of a state transfer operation.*

2. Existing High Availability approaches to SDN

a. **Ravana: High Availability for SDN Controllers**
   i. Guarantees controller availability by adding a replica controller (in active:active configuration)
   ii. Leverages OpenFlow 1.3 functionality allowing a switch to connect to multiple controllers (master and slave)
   iii. All events (ie. a `linkdown` event) sent between switch and controller are wrapped up in a *transaction* abstraction.
      1. All transactions are buffered at the switch until they are ACKed by the controller
      2. All transactions are saved in a shared memory log between the master + slave controllers.
      3. Two stage replication ensures 1) transactions are reliably replicated, 2) transactions are acknowledged.
   iv. Extends OpenFlow to guarantee messages are delivered at least once under failures.

v. Ravana system diagram:



b. **Pico-Replication: High Availability for SDN Middleboxes**

  i. Provides *flow-level replication* of middlebox state

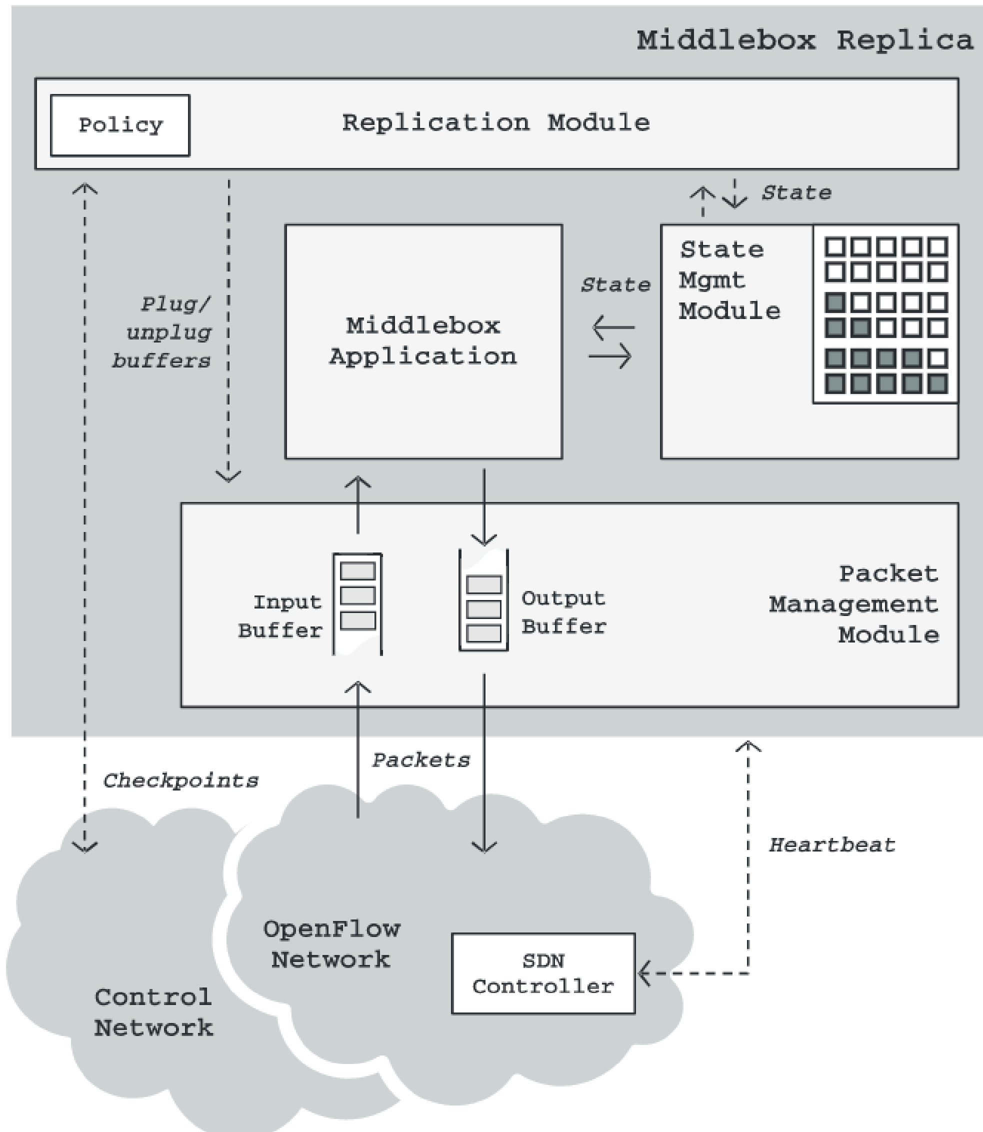
  ii. Checkpoints individual flows and copies them to replica (backup) NFs

    1. Ingress packets for a flow are suspended/buffered
    2. Flow is checkpointed + replicated
    3. Buffered packets are processed

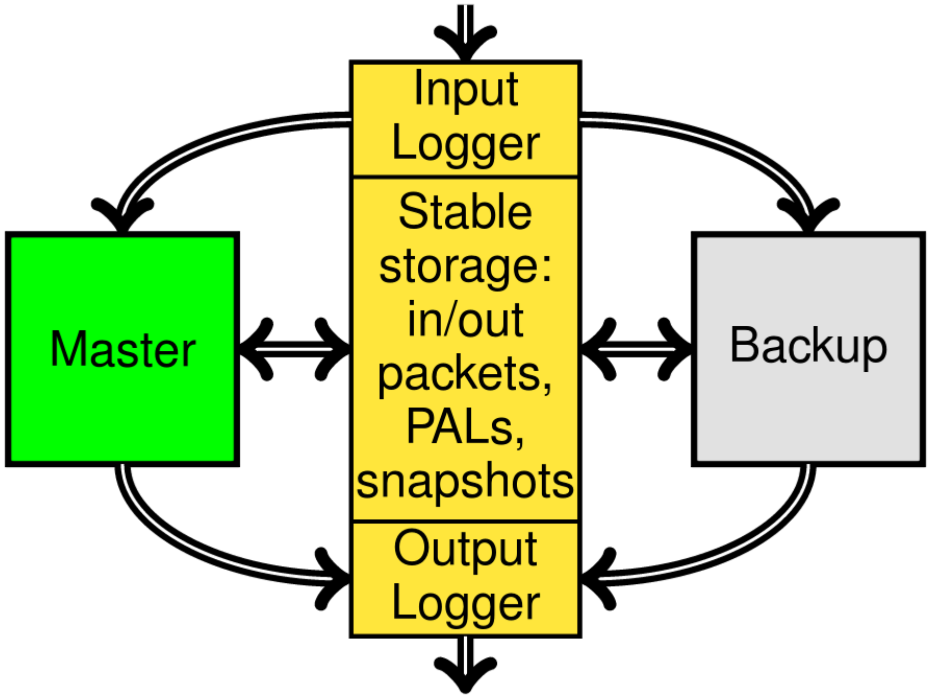
  iii. Organizes flows into *replication groups* to ease the compute overhead. These replication groups are scheduled to the same replica backups.

  iv. SDN controller detects failures and reroutes flows to their backup replicas

  v. Pico-Replication system diagram:



c. **Rollback Recovery: High Availability for SDN Middleboxes**
   i. Does not use active NF replicas!
   ii. Instead, uses a combination of NF checkpoints + packet logging mechanisms to record incoming traffic.
      1. These are both saved to a backup in persistent storage.
      2. On failure, RR retrieves the last checkpoint and replays all logged incoming traffic.
   iii. Output traffic is also queued. It is released once RR can guarantee that incoming packets can be replayed deterministically (in a multi-threaded, non-deterministic NF environment).
   iv. Rollback Recovery system diagram:



3. **High Availability challenges in OpenNF**
    a. OpenNF has three main points of failure which a HA implementation must consider:
        i. **Controller**
            1. Ravana can deal with common-case controller failures.
            2. If a controller fails during an OpenNF state transfer operation, we need to ensure the move is either completely correctly, or rolled back correctly.
        ii. **Middleboxes (NFs)**
            1. Pico-Replication and Rollback Recovery both provide mechanisms for handling common-case NF failures.
                a. Which approach do we use?
            2. For OpenNF HA, we need to add mechanisms to ensure correctness of state transfer operations when a NF fails.
        iii. **Network**
            1. We have no control over network failures, but we still need to reason about them.
            2. Network failures will cause dropped sockets on both controller and NFs. Endpoints need to determine whether a dropped socket indicates an endpoint failure or network failure.
    b. Detecting failures
        i. Existing works on SDN HA only consider fail-stop errors.

1. **Assumption:** We only need to reason about fail-stop (crash) errors, not Byzantine faults.

ii. How does a network component detect failures in other components?

1. **Assumption:** A fail-stop error will take a network component offline. So we can detect failures by a dropped socket connection.

c. Ensuring correctness of state transfer operations in spite of failure

i. An OpenNF state transfer involves the transmission of many state/event messages. These need to get delivered correctly and in order.

4. OpenNF Controller HA Implementation

a. We can implement Ravana to guarantee that a replica controller is always available.

i. Maintains controller + switch consistency in face of failures

ii. Ravana is application independent, should be plug & play with OpenNF

iii. **Problem:** Ravana does not provide services to NF control plane

b. **NF control plane failures:** if a controller fails, we need to ensure OpenNF application state is correct on the replica (slave) controller

i. We can do this by copying/rebuilding Ravana's mechanisms in the OpenNF application code

ii. Ravana uses OpenFlow to communicate between switch and controller runtime.

1. **Problem:** NFs don't speak OpenFlow.

iii. **Solution:** We need to extend OpenNF's state/event channels to provide some additional features:

1. NFs need extra state/event channels to communicate with multiple controllers (master + replica)
2. **Research problem:** Controllers need to store incoming OpenNF state/event messages in some manner of shared storage. What storage do we use?
3. NFs will need to buffer outgoing messages and not erase them until they are ACKed by controller

5. OpenNF Middlebox HA Implementation

a. **Research problem:** Two different ways to recover.

i. Do we take a rollback approach?

ii. Or do we find a way to continue the move using replicated instances?

b. **Research problem:** If we take a Pico-Replication approach, state gets backed up on a different NF.

i. Can we eliminate state transfers entirely at this point? Is it sufficient to simply update the switch forwarding table and reroute flows to the backup?

6. Reasoning about network availability in OpenNF

a. When a controller sees a dropped NF socket connection, how can we tell if this is due to network failure or endpoint failure?
    i. If we use a replication approach to 5.a), controller will be aware of a backup NF. If this backup NF is still online, we can assume failure is in the master NF.
    ii. **Research question:** If we use a rollback technique for recovery for 5.a) (and hence only one active NF) how can a controller differentiate between network failure and NF failure?
b. When an NF sees a dropped controller socket connection, how can we tell if this is due to network failure or endpoint failure?
    i. All NFs connect to two controllers. So we could reason about failure by seeing in only one controller goes offline (controller failure) or both go offline (network failure).
    ii. However there are some corner cases where this approach fails.
    iii. **Research question:** How can an NF differentiate between network failure and controller failure?
c. **Research question:** Once we've identified the source of a network failure, how do we react to it?
    i. Just try reconnecting repeatedly?