SmartSwitch – High Availability Overview

Riff Jiang

Outline

- What is HA?
- HA scope and ENI pair placement
- Network setup for HA and traffic forwarding
- Control plane overview and ENI programming model
- HA state machine and operations

HA for ENI

What is HA (High Availability)?

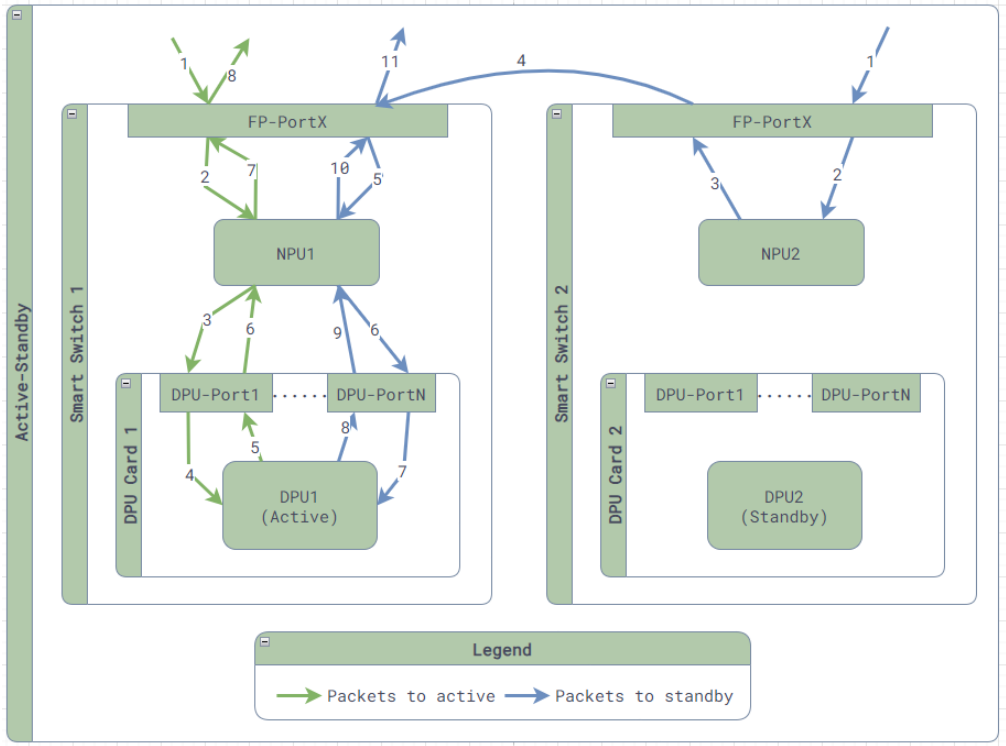
- A single switch / DPU goes down or when a network failure happens, it will not:
  - Kill all the traffic for an ENI (vNIC).
  - Causing all existing flows to be dropped.
- Goals: https://github.com/sonic-net/DASH/blob/main/documentation/high-avail/high-availability-and-scale.md.
  - 0 downtime on planned switchover.
  - <2 sec downtime on unplanned failover to standalone setup for each ENI.
  - Ability to resume connections in the event of both planned and unplanned failover.
  - After both planned and unplanned failover and recovery, the flow on all DPUs will be aligned.
  - …

So, what is HA?

- **Flow HA:** Each ENI will be backed up by 2 DPUs, so flows can be replicated between them, and won't be dropped when one DPU/Switch is having problem.
- **Data path HA:** Handles network failures and reduce the chance of packet drops.

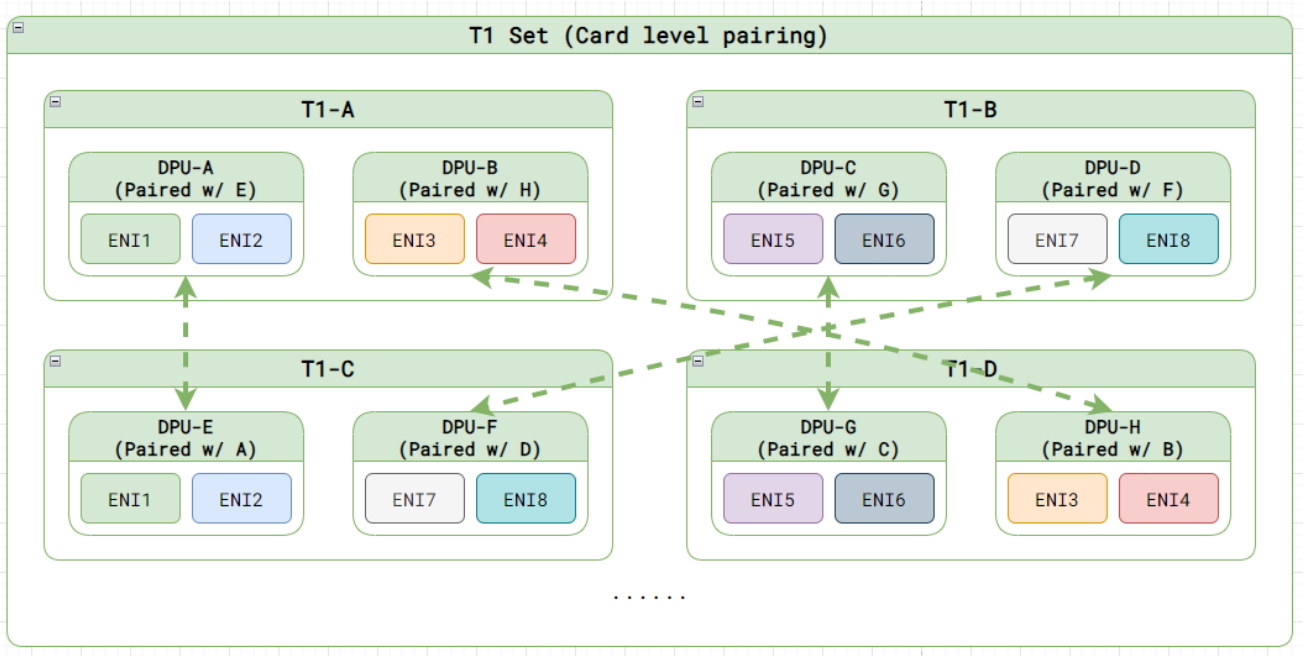
ENI Flow HA – Active/Standby

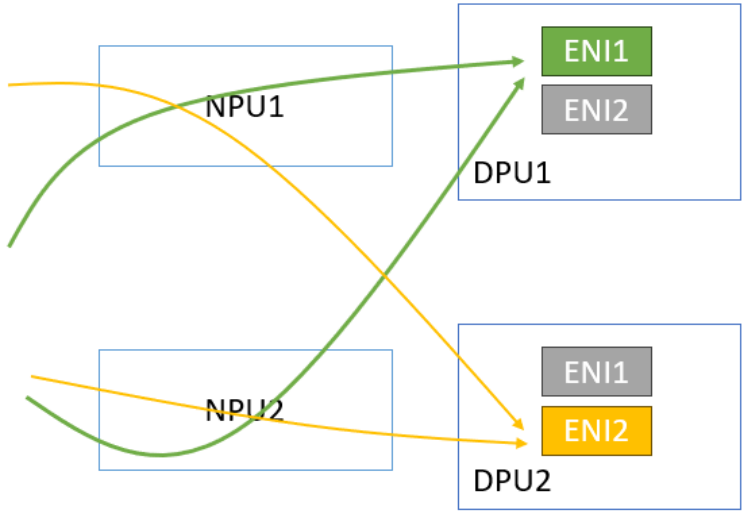
- Each ENI must be placed in 2 DPUs from 2 different switch forming a HA pair.
- 1 DPU serves as active and making flow decisions, while the other DPU serves as standby, acting as a flow storage, only accepting flows replicated from active.
- In steady state, all flows will be inline sync'ed from active to standby. When a DPU rejoins the HA pair, we bulk sync the flows.
- When the active DPU runs into issues, we will failover the active, make the standby the new active, and switch over the traffic to avoid further impact.



ENIs in SmartSwitch

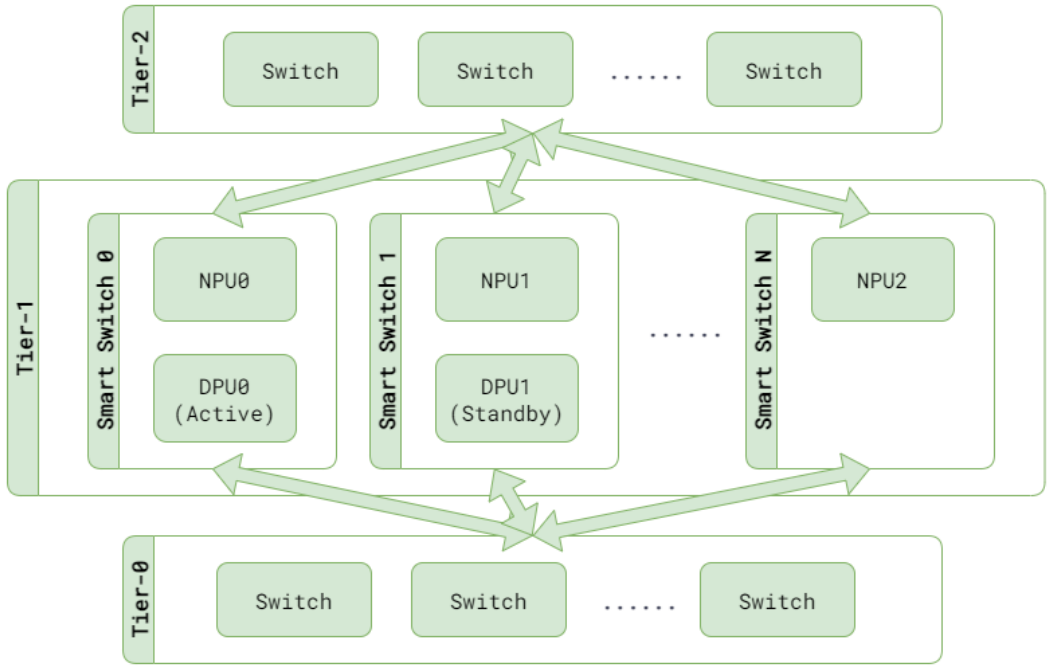
- Card-level ENI pairing
- ENI-level Active/Standby





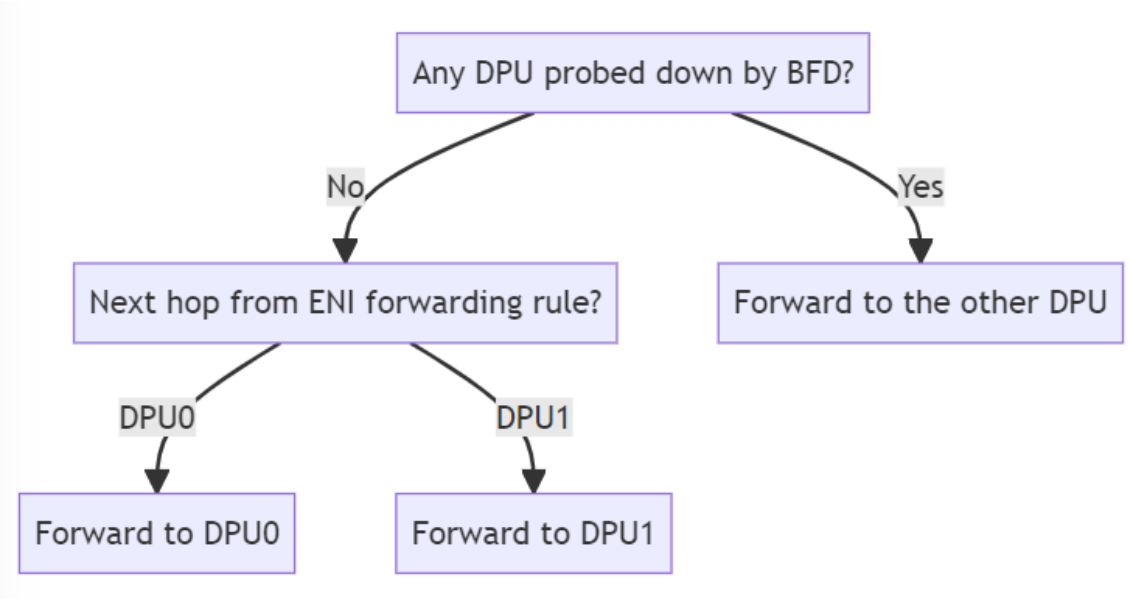
Data Path HA

- All T1s advertise the same VIPs.
- When a T1 receives a packet for an ENI, it forward the packet directly to the active DPU using a VxLan tunnel.
- Handles single switch failure and no more waiting for BGP reconcile.



Controlling Traffic Forwarding

- Card-level NPU-to-DPU probing
  - NPU sending BFD probe to DPU to check if DPU is alive or not.
  - When probed down, NPU will stop forwarding all traffic to this DPU.
  - When probed up, NPU will respect the ENI-level traffic control.
- ENI-level NPU-to-DPU traffic control
  - Explicitly setting up the next hop to the current active side.



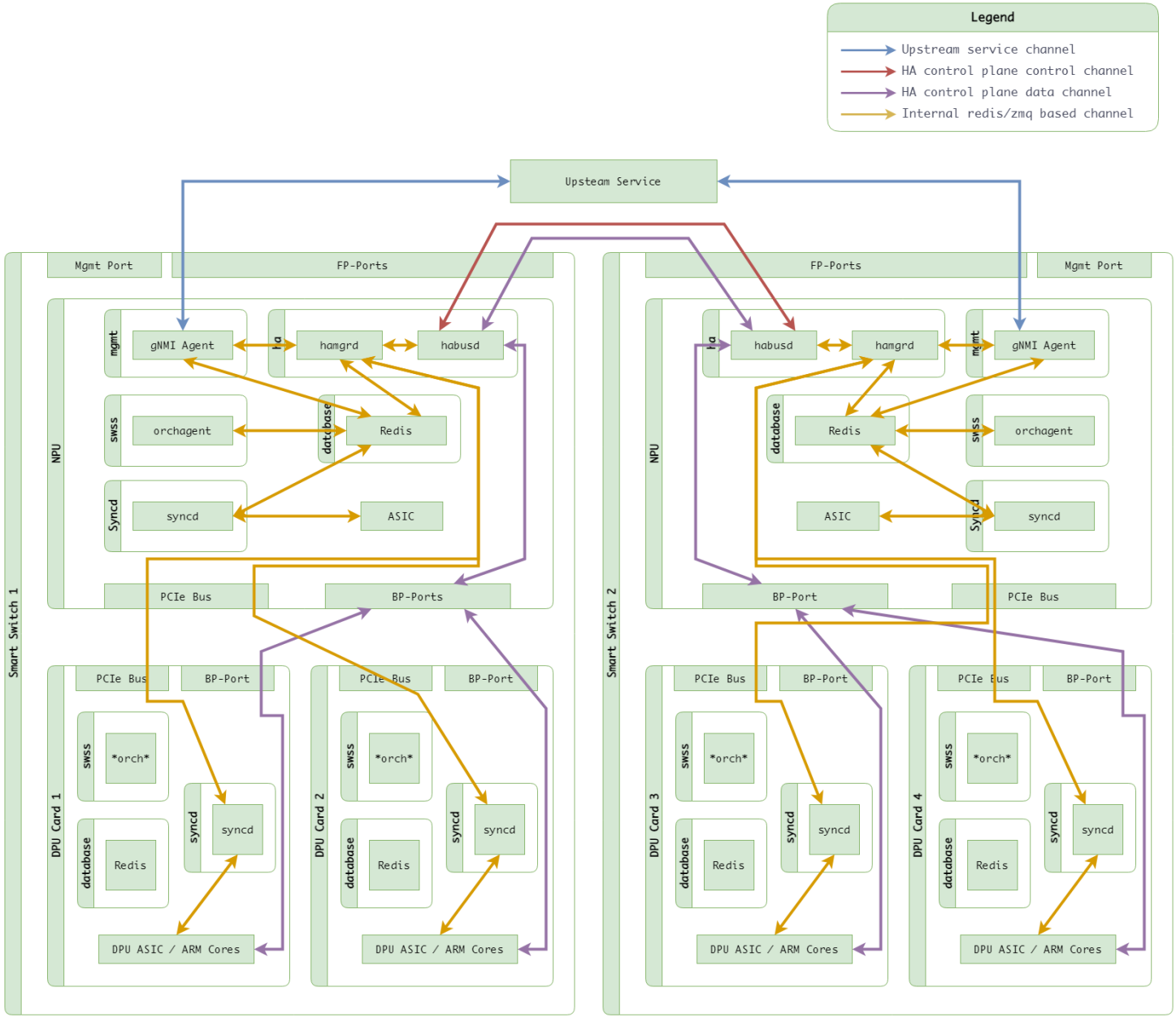
HA Control Plane Overview

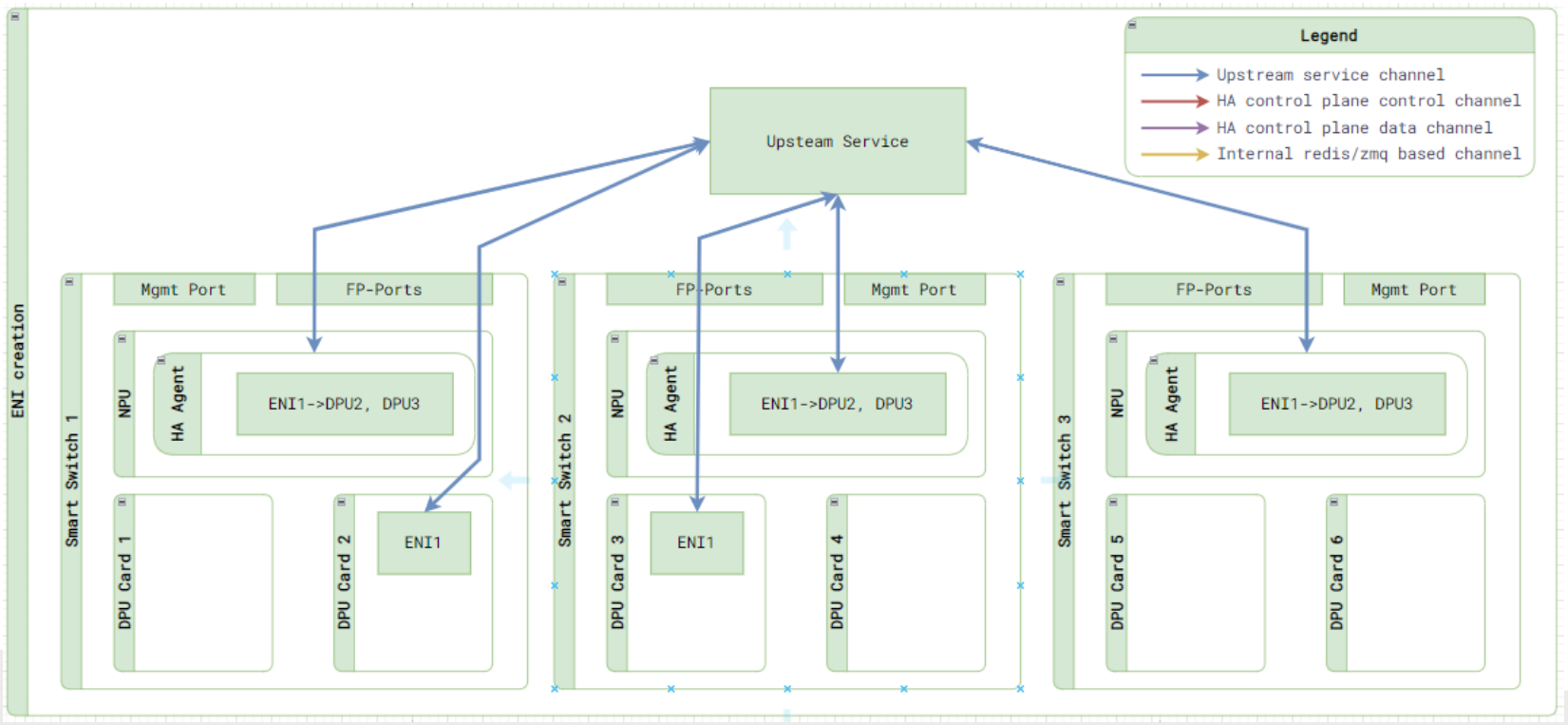
Control Plane Overview

- Upstream Service (SDN controller):
  - Decide ENI placement and pairing.
  - Decide preferred active ENI placement.
  - Decide desired HA states under planned events, such as planned switchover, planned shutdown for upgrades, evening traffic and etc.
  - Triggering HA operations for manual live site mitigations.
- SmartSwitch:
  - Drive the HA state machine transitions.
  - Report every ENI HA state change and reasons, so upstream service knows what is happening and can make decisions for planned events.
  - Handle HA related requests from upstream service.
  - Monitor and handle unplanned events, and trigger defined mitigations, such as driving to standalone setup.

HA communication channels

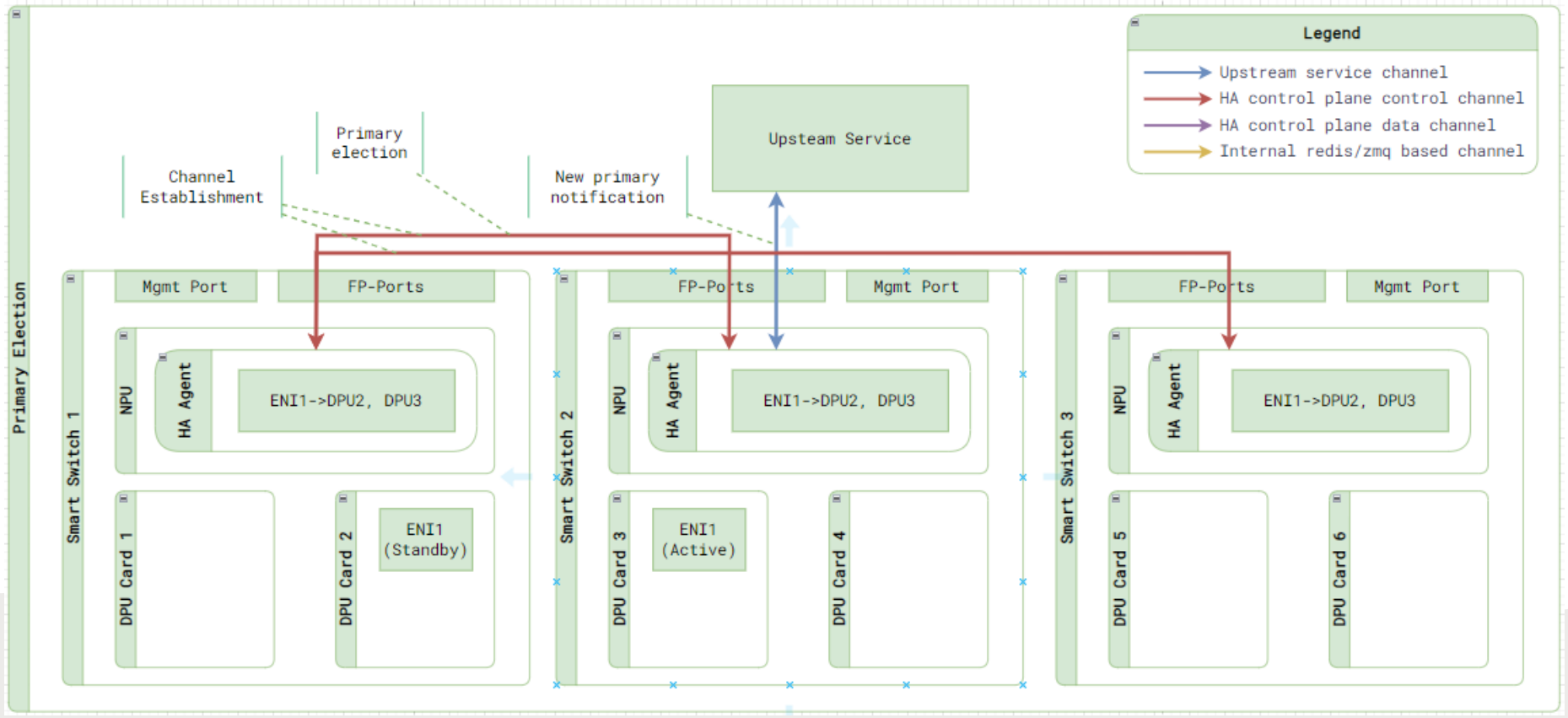
- Upstream Service Channel
  - Goal state programming
- HA Control Plane Channels (gRPC)
  - HA Control Plane Control Channel
    - Driving HA state transitions
    - Updating ENI traffic forwarding rules
    - Guaranteed to work within a bounded time
  - HA Control Plane Data Channel
    - Bulk sync
- Data Plane Channel (Tunnel)
  - Inline sync





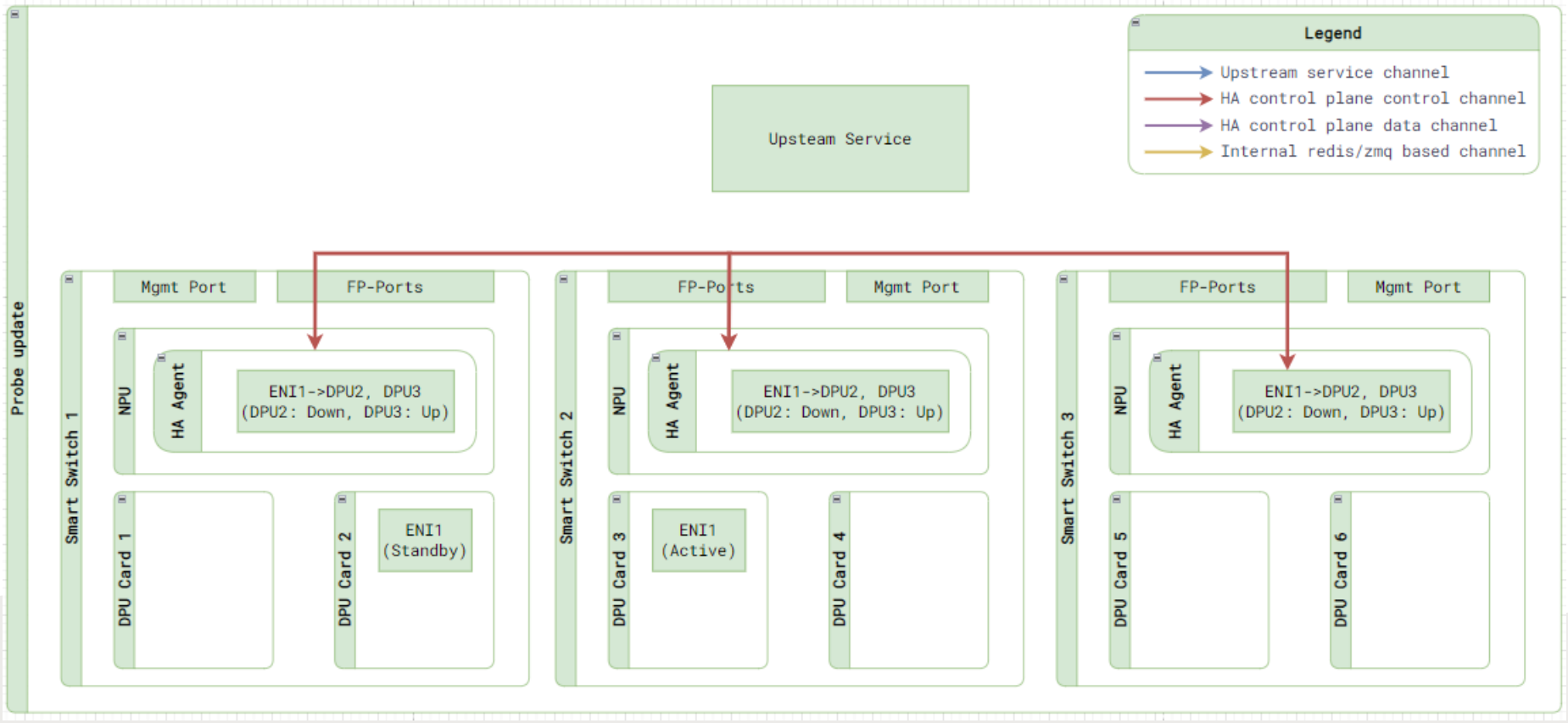
ENI Creation (Step 1)

- Upstream service will first decide where to put the new ENI and form the HA set.
- Upstream service calls northbound interface and programs the following things on each SmartSwitch independently
  - Create the ENI on selected DPUs with its peer information, so we can form a HA set.
  - Program traffic forwarding rules to all the switches that will receive the traffic.



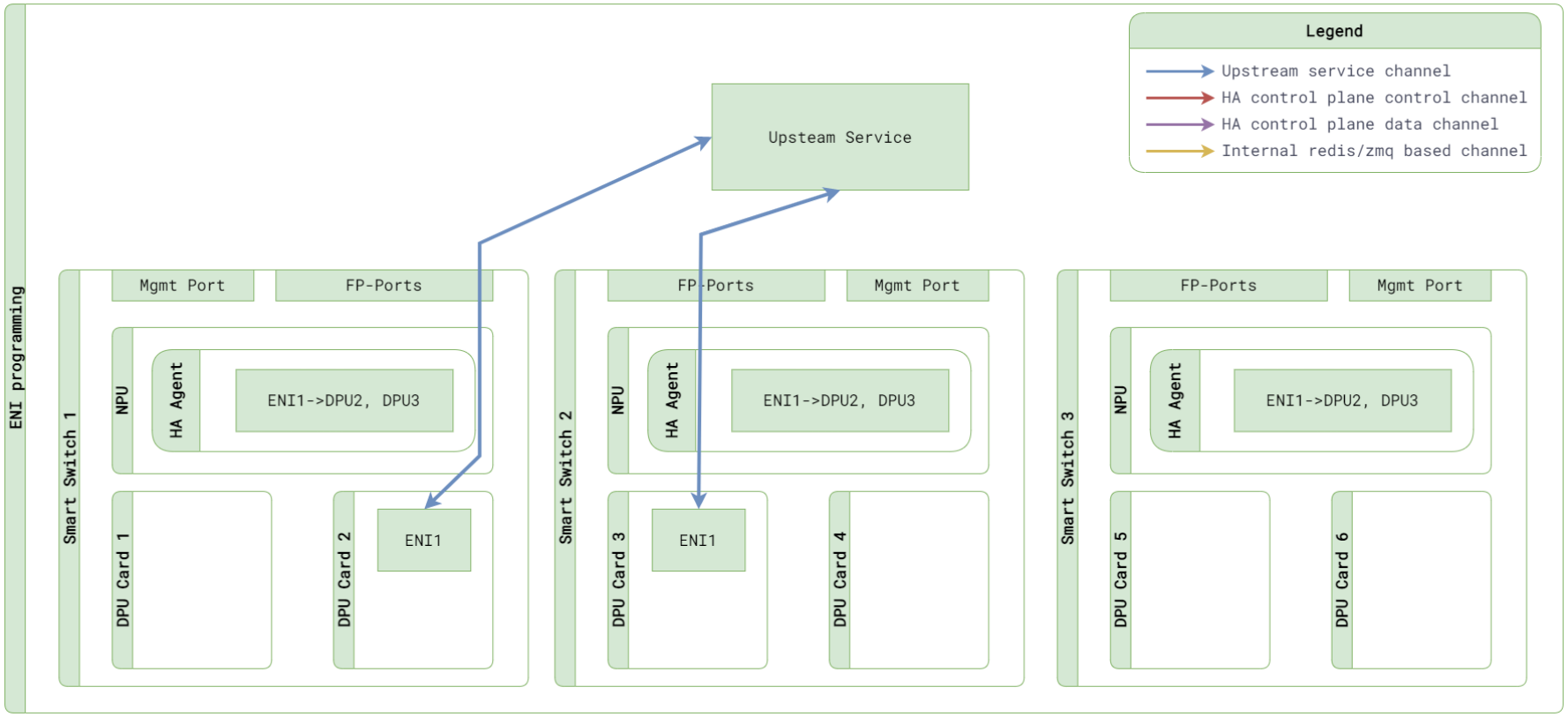
ENI Creation (Step 2)

- Once programming is finished, the 2 ENIs will start forming the HA pair with:
  - Control plane channels created
  - New active elected with the instruction that is specified by upstream service.
- Once the new primary is elected, SmartSwitch will notify the upstream service that the primary is selected.



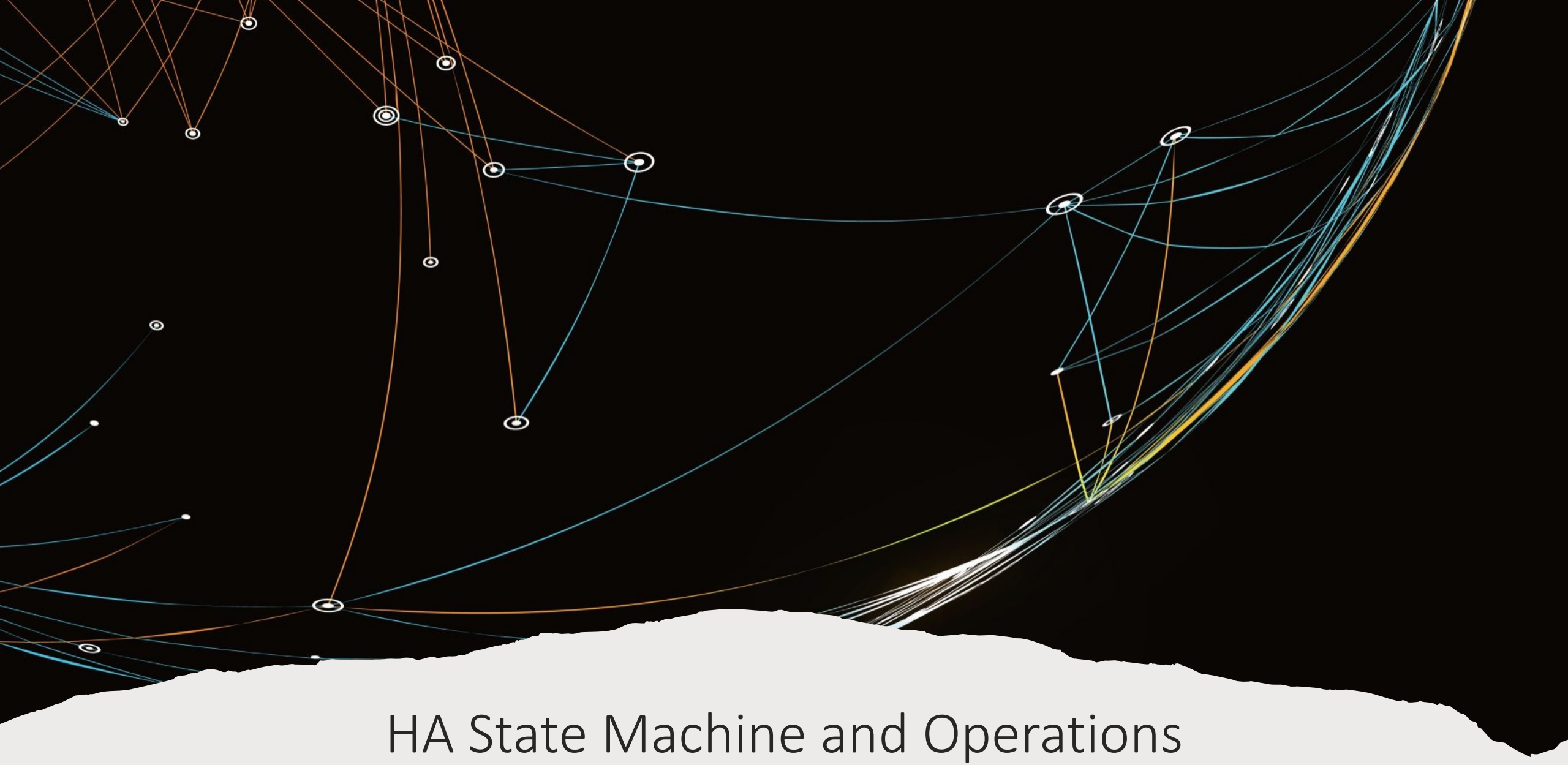
ENI Creation (Step 3)

- The primary election process will also update the probe state of all ENIs, making sure the traffic is forwarded to the right DPU.



ENI Programming

- For all future goal state update, upstream service will program both ENI independently.



HA State Machine and Operations

Planned Events vs Unplanned Events

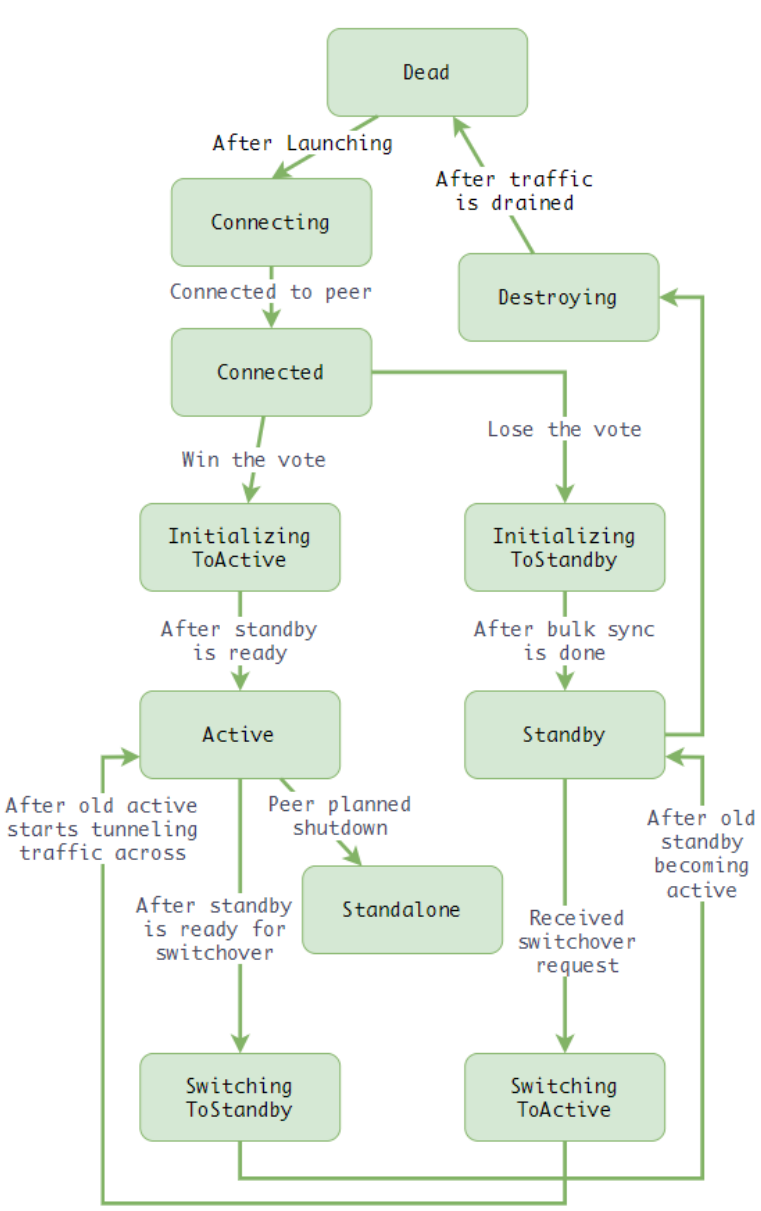
- Planned events:
  - ENI launch, Switchover, Shutdown, Migration, etc.
  - Goal: (Close to) 0 down time. Avoid bulk sync / flow merge as long as we can.
  - Initiated from and approved by upstream service.
- Unplanned events:
  - Network failure, DPU/NPU/PCIe failure.
  - Goal: <2s failover to standalone setup.
  - Initiated and driven from SmartSwitch w/ predefined config from upstream service.

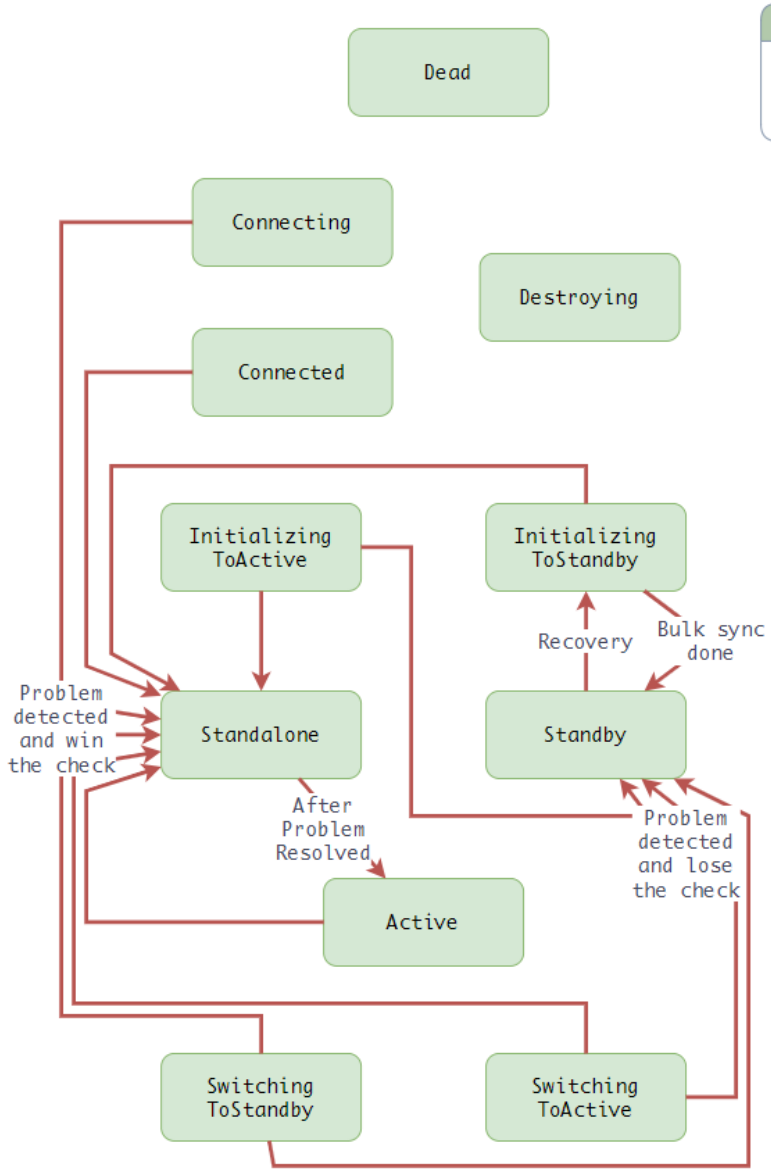
HA State Machine

- Key states: Dead, Active, Standby, Standalone
- Other transition states can be found in appendix.

State	Definition	Receive Traffic from NPU?	Make decision?	Handling old flow?	Respond flow sync?	Init flow sync?	Init Bulk sync?
Dead	HA participant is just getting created, and not connected yet.	No	Drop	Drop	No	No	No
Active	Connected to pair and act as decision maker.	Yes	Yes	Yes	No	Yes	Yes
Standby	Connected to pair and act as backup flow store.	No	Tunneled to pair	Tunneled to pair	Yes	No	No
Standalone	Heartbeat to pair is lost. Acting like a standalone setup.	Yes	Yes	Yes	Yes	No	No

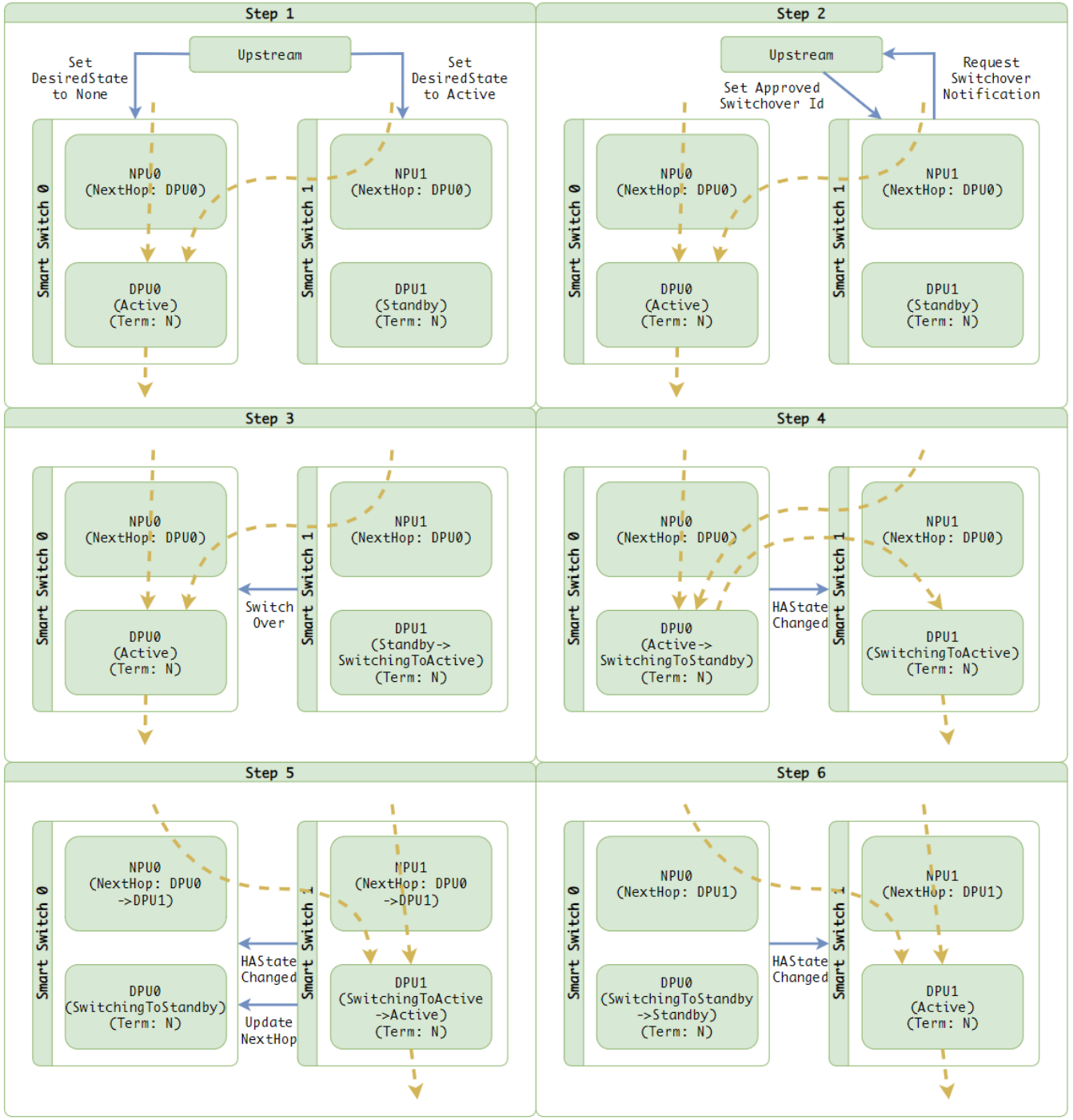
HA State Transition

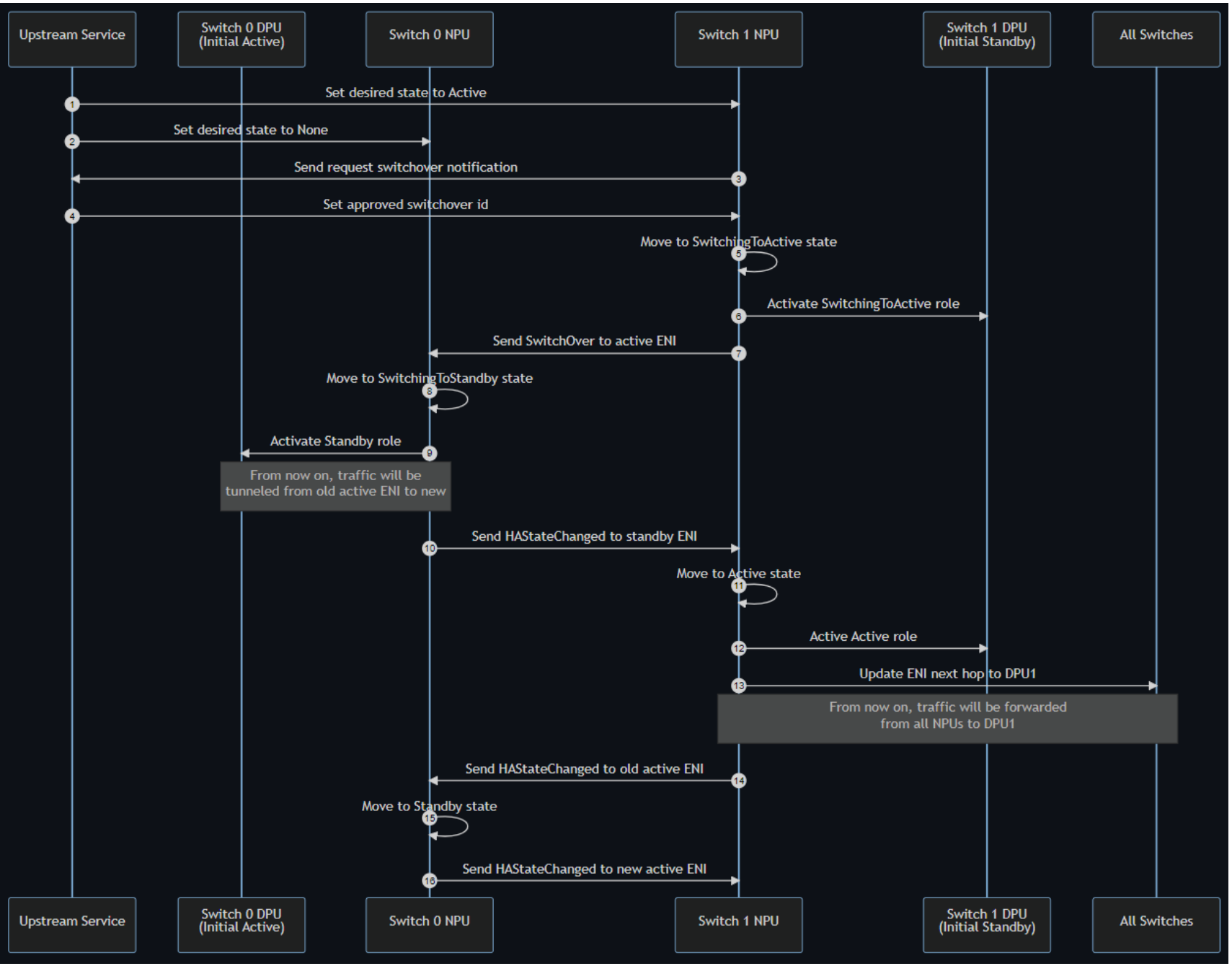




Planned Switchover

- Upstream initiates the operation by setting the desired state.
- SmartSwitch requests upstream for switchover, in which upstream ensures the state is in steady state and policy on both side matches.
- Once approved, SmartSwitch drives the state transition and update the ENI-level traffic control to desired state.
- Only 1 flow decider at all time and bulk sync is not needed.

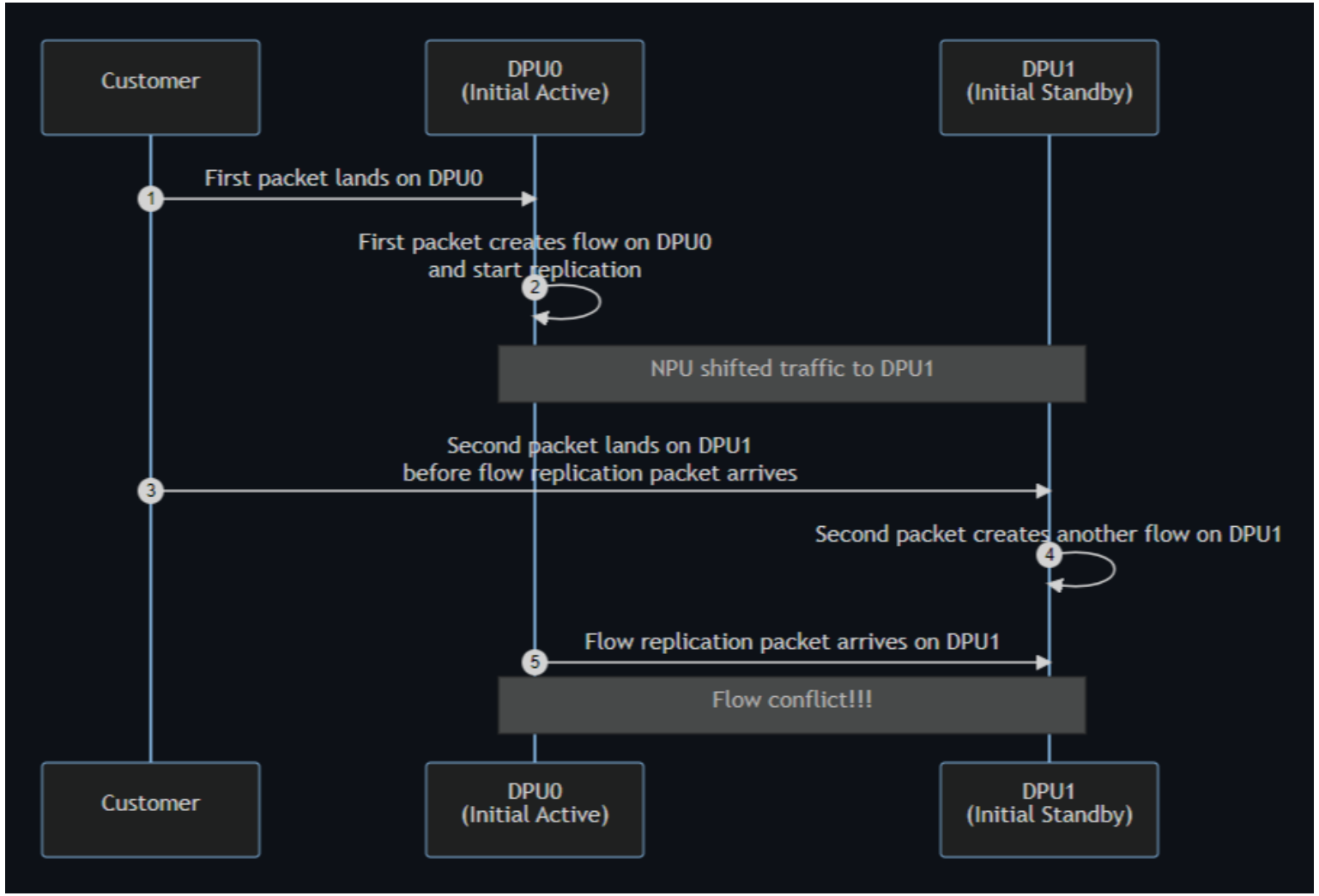




Planned Switchover – Seq. Diagram

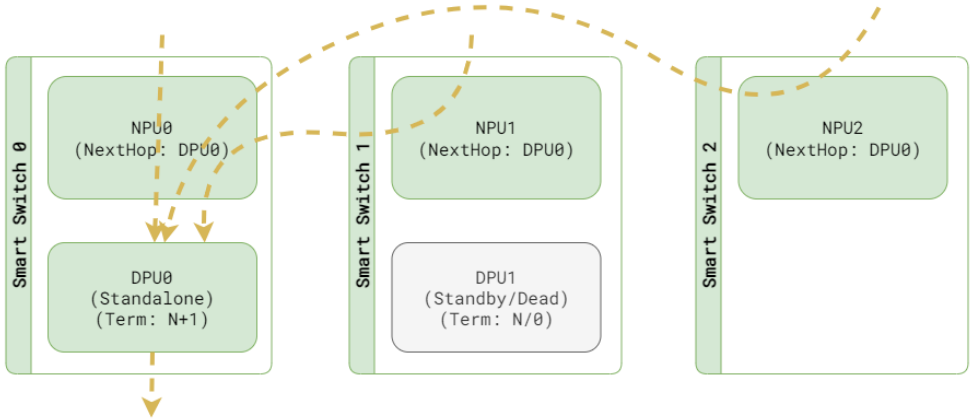
Packet racing during switchover

- Flow replication delay could cause flow being created on both sides.
- Dropping the flow replication packet to avoid flow being created on old active.
- Upstream service will ensure 2 DPUs running the same policy on switchover request approval.



Unplanned events vs Standalone setup

- Network interruption will be unavoidable.
- Best Effort: Reduce the chance of dropping packet by stopping flow replication. No solution is going to be guaranteed to be perfect.
- “Standalone-Standby/Dead” pair to avoid 2 deciders / Brain split.
- Card-level standalone setup - All ENIs will failover together. This is due to bulk sync limitation today, but we are going towards ENI level.
- Unplanned events are monitored. If any unplanned events last long, fire alerts.



Step 1 - Triggers

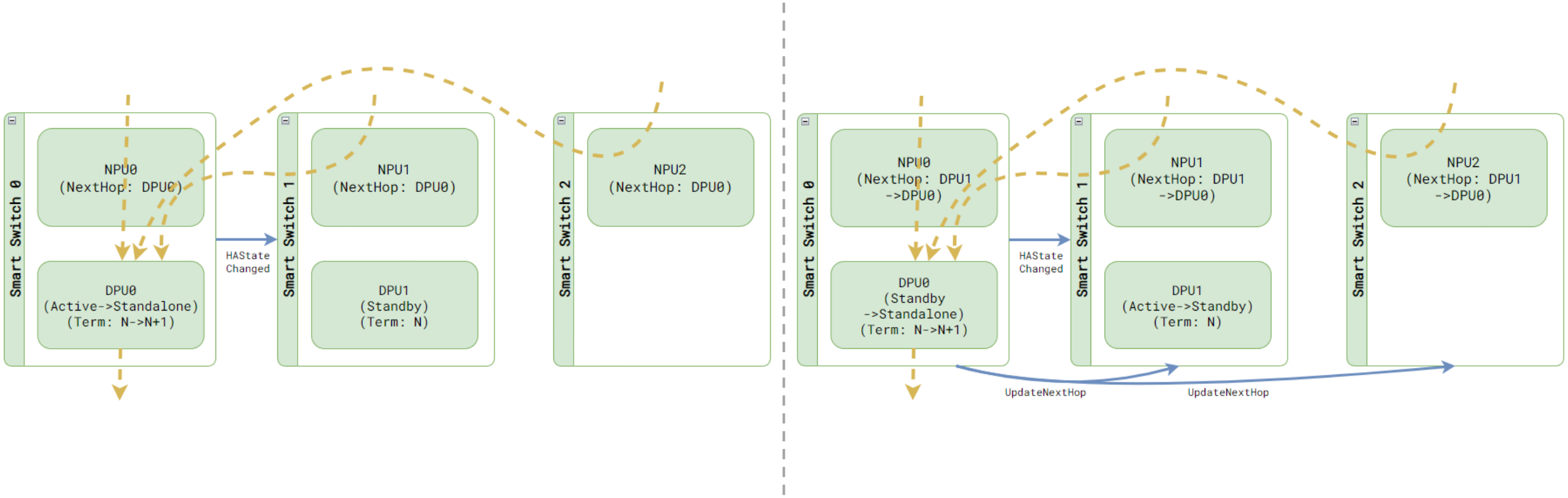
Problem	Trigger	Resolve signal
Peer shutdown	Planned shutdown request	HAStateUpdate with Connected state
Peer DPU lost	Peer lost SAI notification (Full DPU-to-DPU liveness probe failure)	Peer connected SAI notification (DPU-to-DPU liveness probe succeeded again)
Peer DPU dead	HAStateChanged with dead peer	HAStateChanged with non-dead peer
High data plane packet drop rate	ENI-level data plane counters	ENI-level data plane counters
Manual Pinned	ENI-level DPU isolation	Isolation removed
Card pinned to standalone	Card pinned to standalone	Pinning removed

Data Plane Channel and DPU-to-DPU liveness probe

- Data plane channel
  - Used for inline sync and etc.
  - Source port is calculated based on the hash of the (inner most) packet.
- DPU-to-DPU liveness probe
  - Used for detecting data plane channel failure.
  - Rotate the source port to handle gray failures.
  - Initiated when calling the create HA set SAI API.
  - Probe happens on DPU level and counters are logged on ENI level.
- It doesn’t include the ENI-level pipeline failure detection
  - The ENI pipeline policies are programmed based on each SDN scenario.
  - Each scenario should design its own tests/probe, which is beyond the scope of HA.

Step 2 - Vote

- Merge the signals from ENI level to card level first
- Things to check:
  - Already in standalone state
  - Peer DPU health signals
    - Covers DPU/Switch hardware failures, e.g., Peer DPU lost/dead.
  - Send request to pair to start voting
    - Standalone state of the pair
    - Check if any DPU is manual pinned by DRI. If both pinned, cancel vote and alert.
    - Use preferred standalone DPU programmed from upstream (Covers data path failures).
- ENI-level standalone voting is in omitted here. We can find it in full design doc.

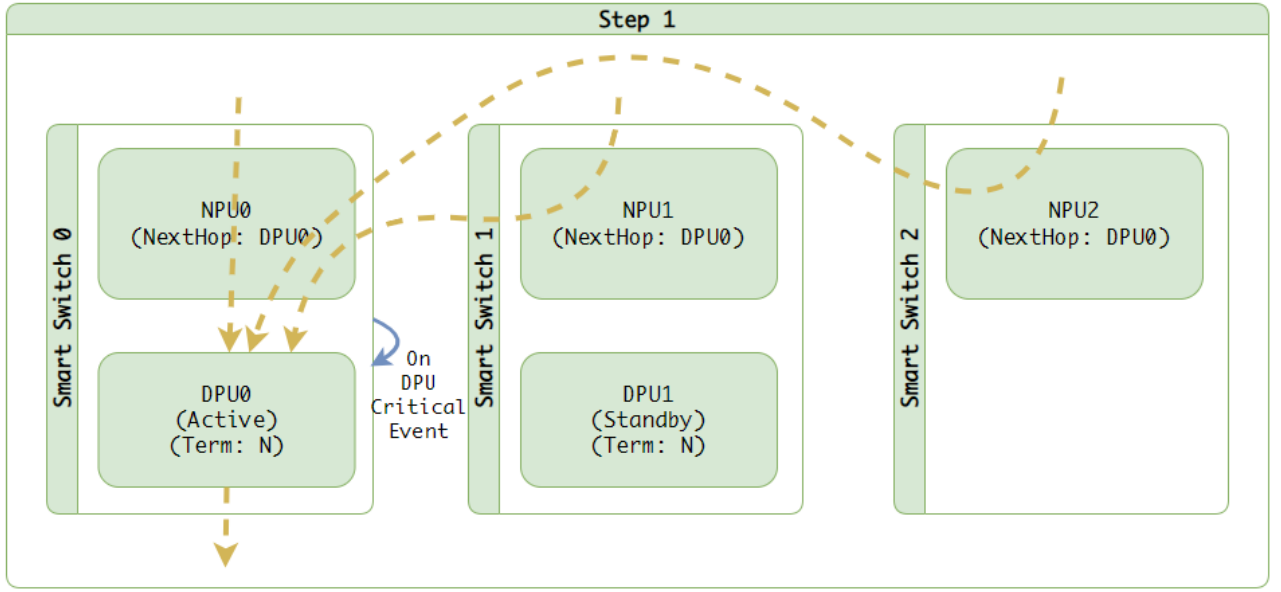


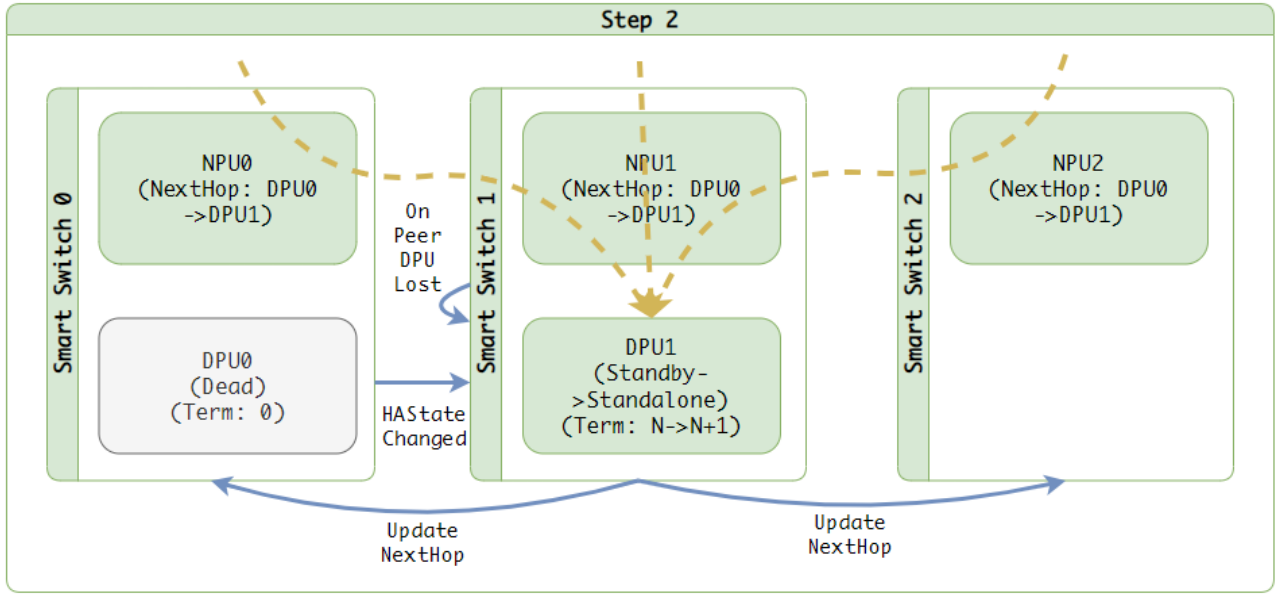
Step 3 – Drive into standalone state

Once vote is done, each ENI's state will be updated and start to drive on its own.

Example – Peer DPU is down

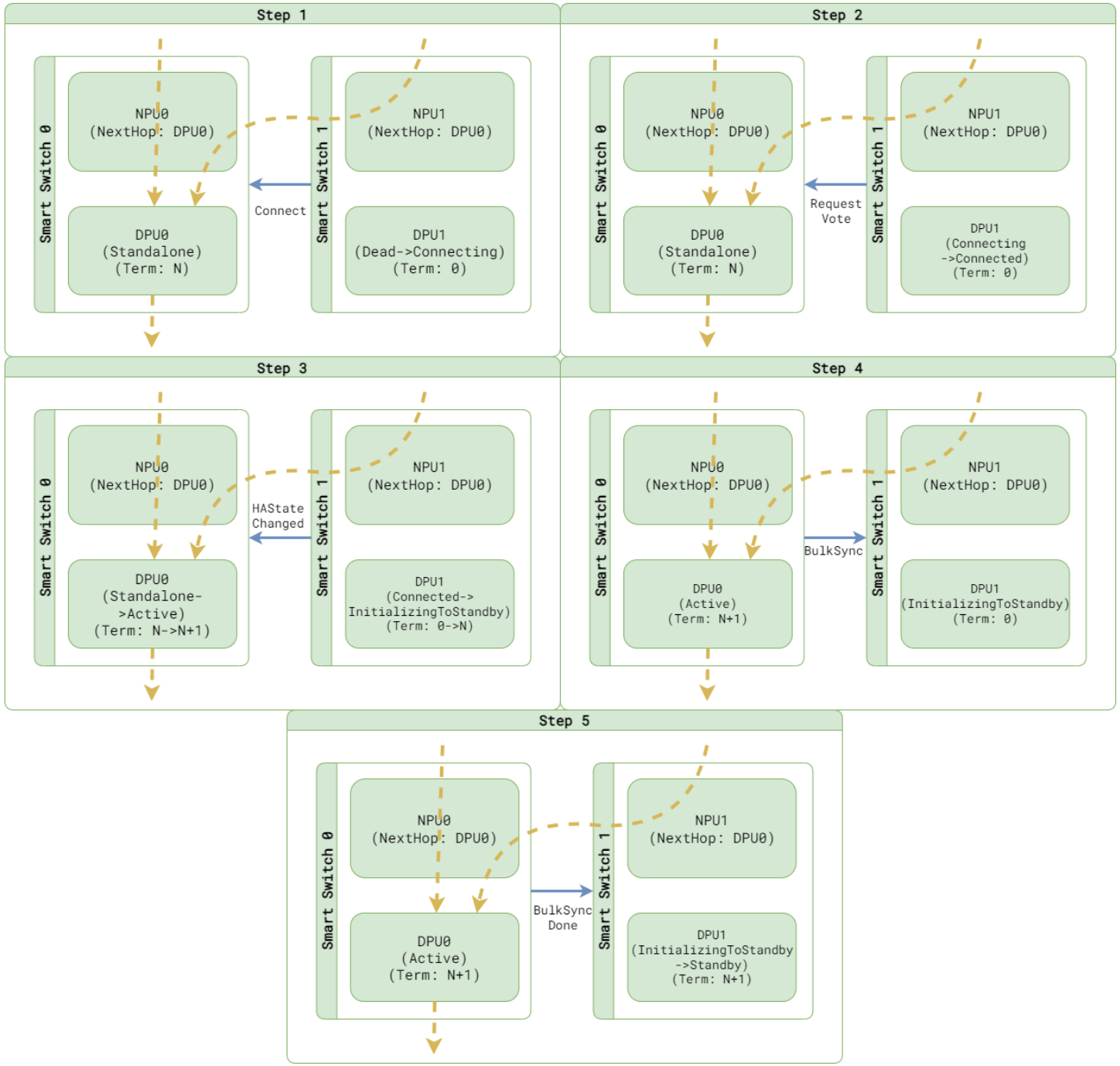
- Using “Peer Lost” SAI notification or “Peed dead” update as trigger.
- Once detected, we will start voting. This will cause the good side to go into standalone state, which stops flow replication.
- Due to card level standalone, other ENIs in standby state will also be moved to standalone state.





Bulk sync is used for bring the node to ready again

Recover from standalone setup



Term Tracking and Primary Election

- Term:
  - Find the DPU that lives the longest and knows the most on the historical flow changes.
  - While a DPU can make flow decision, the term increases whenever the inline sync channel is enabled or disabled.
- Primary election: Whichever with higher term wins.
  - It also need to consider other cases to make the algorithm practical, such as ENI not found, Pinned to standalone, ENI is shutting down.
  - https://github.com/r12f/DASH/blob/user/r12f/ha/documentation/high-avail/smartswitch-ha-hld-proposal.md#83-primary-election

Flow lifetime management

- Active rules all, standby always follows.
  - Inline sync – Connection creation, close or re-simulated.
  - Batched sync – Flow aged out, bulk sync TCP seq number or flow deletion via SAI APIs.
  - Bulk sync – When a node rejoins the HA set.

Bulk sync

- Perfect sync
  - Whenever a node becomes active, we increase the flow version (color).
  - Sync all flows that has lower flow version than current.
- Range sync (ENI level only)
  - Idea is only sync the flow that is not sync’ed
    - ImpactStart = Whenever inline flow sync channel breaks
    - ImpactEnd = Whenever inline flow sync channel reestablished
  - While a DPU can make flow decision, whenever the inline sync channel is enabled or disabled, we increases the flow version.
  - When inline sync channel is disabled, track all existing flow deletions.
  - Sync only the flows that has the flow version when inline sync channel is disabled.

Having problem talking to upstream?

- Policy will become stale gradually.
- HA will continue to function and handle unplanned events.
- As long as upstream can talk to one side of DPU, we can use pin the reachable side to standalone to make the policy update working again.
- If both sides are down, then policy will not be updated. Everything will freeze as it is. (No planned operations and no policy updates).

DASH / SAI APIs

- HA session API
  - https://github.com/r12f/SONiC/blob/user/r12f/ha/doc/smart-switch/high-availability/smart-switch-ha-hld.md#1541-ha-session-apis
- Flow API
  - https://github.com/zhixiongniu/SONiC/blob/master/doc/dash/dash-flow/DASH_FLOW_SAI.md

Flow API Definition

- A general Flow API for services
- Flow Entry
  - Flow Key (match): e.g., 5 Tuple
  - Value (action)
    - Attributes: for incremental operation
    - Protobuf : for full operation
- Flow Table
  - A table consists multiple flow entries

Key: 1.1.1.1:1000, 2.2.2.2:80, TCP	Value: ACCEPT, Rewrite

Flow Entry Example

ENI: 8888

Key: 1.1.1.1:1000, 2.2.2.2:80, TCP	Value: ACCEPT, Rewrite
Key: 1.1.1.2:1000, 2.2.2.2:80, TCP	Value: REJCTED
Key: 1.1.1.3:1000, 2.2.2.2:80, TCP	Value:ACCEPT

…

Flow Table Example

Flow API Definition: Operation APIs

- Operation API
  - Table
    - Create
    - Remove
    - …
  - Entry
    - Create
    - Remove
    - Set
    - Get
    - Bulk_Get
    - …

API	Description
sai_dash_flow_create_table	Create a new flow table
sai_dash_flow_remove_table	Remove a flow table
sai_dash_flow_get_table_attribute	Obtain the attributes of a flow table
sai_dash_flow_create_entry	Add single new entry to a certain flow table
sai_dash_flow_bulk_create_entry	Add multiple entries to a certain flow table
sai_dash_flow_remove_entry	Remove single entry in a certain flow table
sai_dash_flow_bulk_remove_entry	Remove multiple entries in a certain flow table
sai_dash_flow_set_entry	Set single entry in a certain flow table
sai_dash_flow_get_entry	Get single entry in a certain flow table
sai_dash_flow_bulk_get_entry_callback	Request the vendor interate the flow_table and call the callback function

Flow APIs

Flow API Definition: Table Create

- API
  - sai_dash_flow_create_table(flow_table_id, attr_count, attr_list)
- Allow Reduced Key
  - Key bitmap: 3 Tuple, 5 Tuple
- Allow multiple table
  - Per ENI

Flow table attribute definiation

```
typedef enum _sai_flow_table_attr_t {
    /* Start of attributes */
    SAI_FLOW_TABLE_ATTR_START,

    /* ENI for the flow table */
    SAI_FLOW_TABLE_ATTR_ENI,

    /* Size attribute of the flow table */
    SAI_FLOW_TABLE_ATTR_SIZE,

    /* Expiry time attribute for entries in the flow table */
    SAI_FLOW_TABLE_ATTR_EXPIRE_TIME,

    /* Behavior attribute to track TCP states */
    SAI_FLOW_TABLE_ATTR_BEHAVIOR_TRACK_TCP_STATES,

    /* Reset a TCP connection if it is illegal */
    SAI_FLOW_TABLE_ATTR_TCP_RESET,

    /* Version of the flow table */
    SAI_FLOW_TABLE_ATTR_VERSION,

    /* Flag to specify how to use the key for the flow table */
    SAI_FLOW_TABLE_ATTR_KEY_FLAG,

    /* End of attributes */
    SAI_FLOW_TABLE_ATTR_END

} _sai_flow_table_attr_t;
```

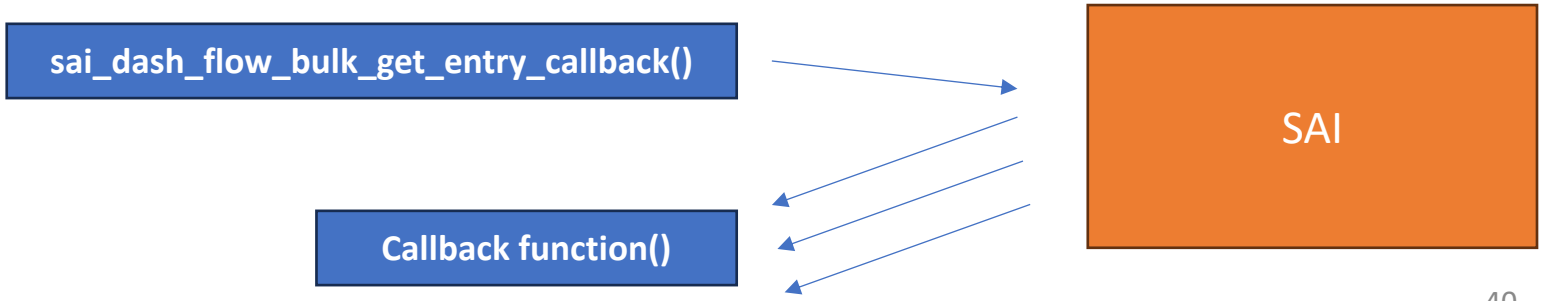
Flow API Definition: Flow Entry Create

- API
  - sai_dash_flow_create_entry(flow_table_id, flow_key, attr_count, attr_list)
- Allow Single/Bi-directional flow-entry
  - Bi-Directional: The SAI creates two entries and links them.
  - Single Directional: The SAI creates one entry; the user creates and links another.
- How to link them?
  - Set the reverse dictionary key in the attributes.
- Bulk transfer
  - Both Single and Bi-directional entries, along with their relationships, should be included in the bulk transfer.

```
typedef enum _sai_flow_state_metadata_attr_t {
    /* Starting marker for attributes */
    SAI_FLOW_ATTR_START,

    /* Version of the policy for this flow. Type: [uint32_t] */
    SAI_FLOW_STATE_ATTR_VERSION,

    /* Metadata of the policy results in protobuf format. Type: [sai_protobuf_t] */
    SAI_FLOW_STATE_ATTR_METADATA_PROTOBUF,

    /* Indicates if the flow entry is bi-directional. Type: [sai_uint8_t] Default: True (1) */
    SAI_FLOW_STATE_ATTR_BIDIRECTIONAL,

    /* Direction of the flow entry. Type: [sai_uint8_t] */
    SAI_FLOW_STATE_ATTR_DIRECTION,

    /* For single directional entries, this represents the key of the reverse direction. Type:
[sai_dash_flow_key_t] */
    SAI_FLOW_STATE_ATTR_REVERSE_DIRECTION_KEY,

    /* Result of the policy. Type: [sai_dash_policy_result_t] */
    SAI_FLOW_METADATA_ATTR_POLICY_RESULT,

    /* Destination Protocol Address. Type: [sai_ip_address_t] */
    SAI_FLOW_METADATA_ATTR_DEST_PA,

    /* ID for metering class. Type: [sai_uint64_t] */
    SAI_FLOW_METADATA_ATTR_METERING_CLASS,

    /* Information required for rewriting. Type: [sai_dash_rewrite_info_t] */
    SAI_FLOW_METADATA_ATTR_REWRITE_INFO,

    /* Vendor-specific metadata details. Type: [sai_u8_list_t] */
    SAI_FLOW_METADATA_ATTR_VENDOR_METADATA,

    /* Ending marker for attributes */
    SAI_FLOW_METADATA_ATTR_END

} sai_flow_metadata_attr_t;
```

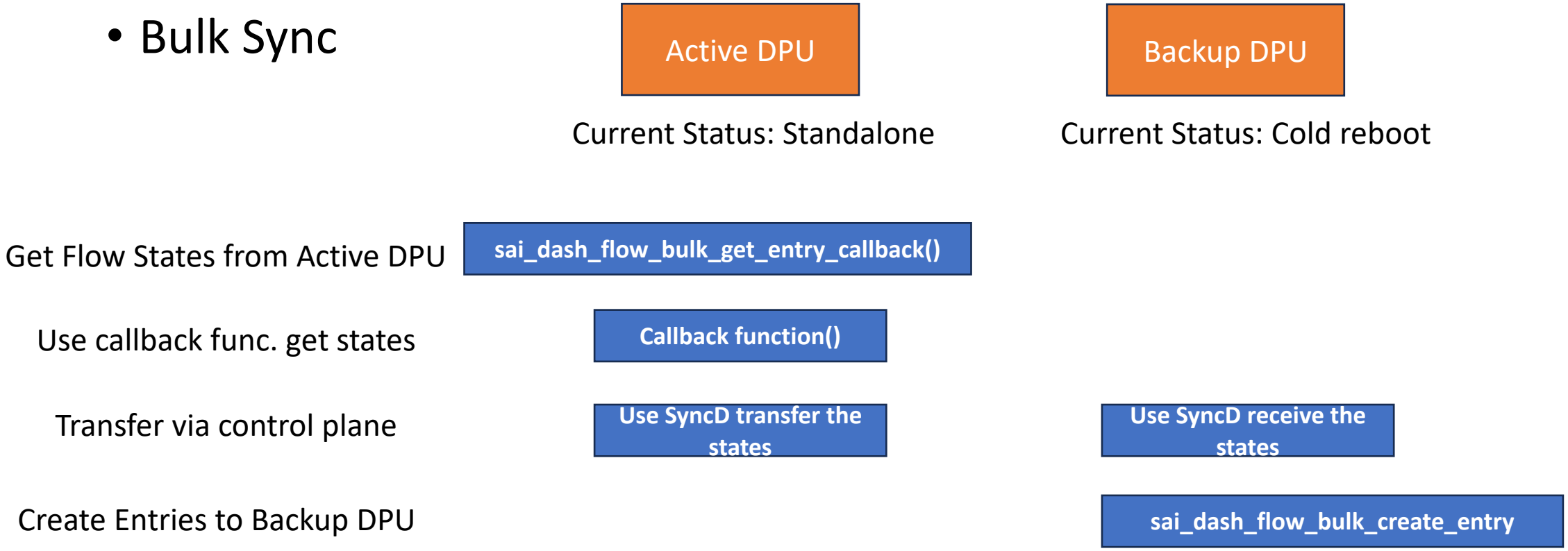
Event-driven Flow Entry Get

- Event-driven flow get entry
  - Why?
    - The flow table is very large (might exceed 10M entries) and always changing.
  - Definition
    - **sai_dash_flow_bulk_get_entry_callback**(flow_table_id, flow_key, query_type, attr_count, attr_list, **callback_function**, timeout, finish)
    - **callback_function**(flow_table_id, flow_key[], attr_count[], attr_list[])
  - Finish condition
    - Flow Version
    - Timeout



Bulk Sync with Flow Query API

- Bulk Sync



Flow Query API

- Bulk transfer in HA
  - Version-based
    - Collect all flows with old versions and transfer them to the backup DPU.
    - To implement the API, it needs one round table for-loop.
- Debug
  - Snapshot of flow table for a period
  - Timeout can be end-condition

- **sai_dash_flow_bulk_get_entry_callback**(flow_table_id, flow_key, query_type, attr_count, attr_list, **callback_function**, timeout, finish)
- **callback_function**(flow_table_id, flow_key[], attr_count[], attr_list[])

Bulk Get API

```
SAI_FLOW_TABLE_ENTRY_QUERY_VERSION_LESS_THAN,
SAI_FLOW_TABLE_ENTRY_QUERY_VERSION_LESS_THAN_OR_EQUAL_TO,
SAI_FLOW_TABLE_ENTRY_QUERY_VERSION_EQUAL_TO,
SAI_FLOW_TABLE_ENTRY_QUERY_VERSION_GREATER_THAN,
SAI_FLOW_TABLE_ENTRY_QUERY_VERSION_GREATER_THAN_OR_EQUAL_TO
```

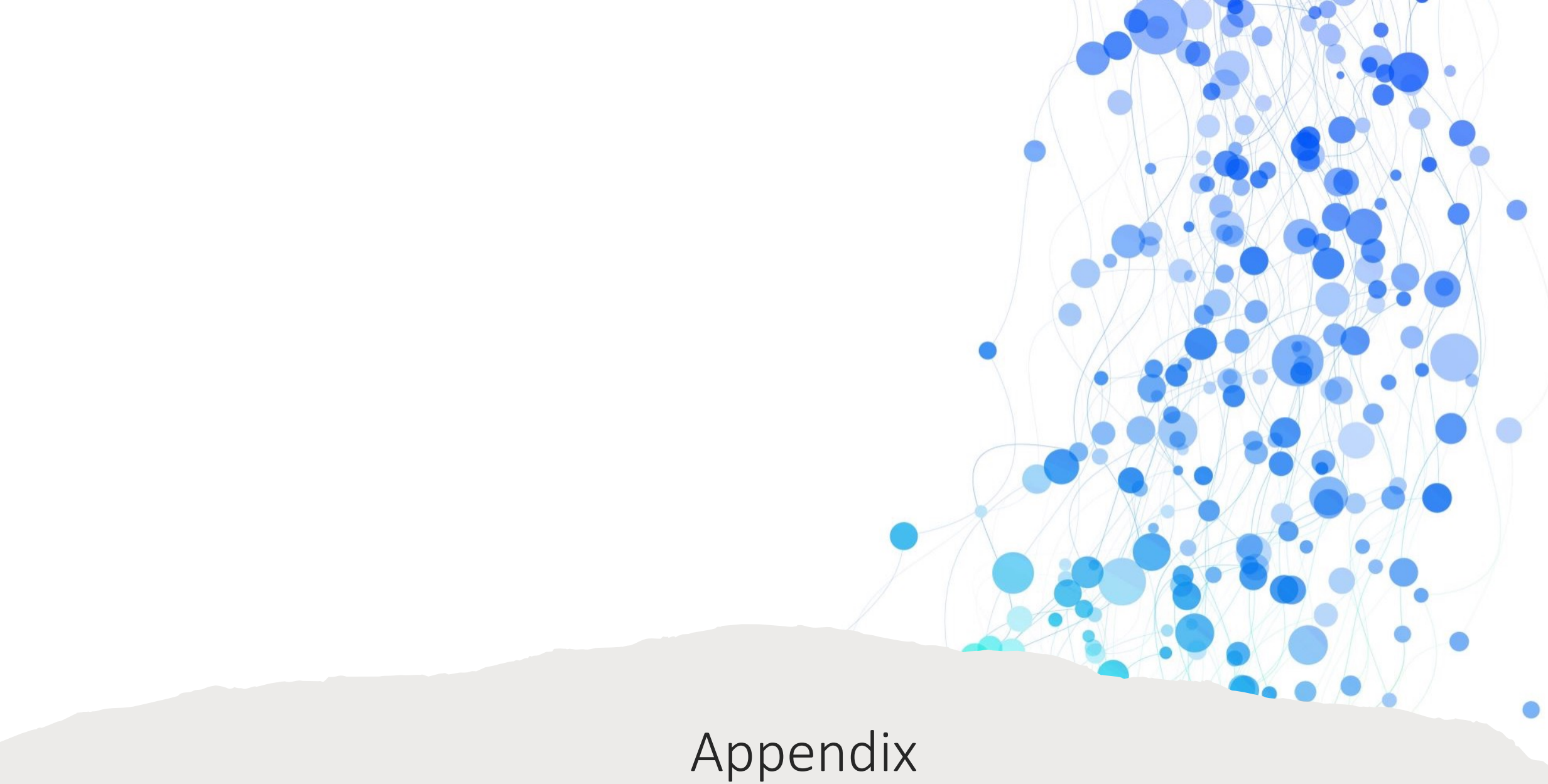
Flow Query: Version Part

Detailed Doc Discussion

- Flow-based API
  - https://github.com/zhixiongniu/SONiC/blob/master/doc/dash/dash-flow/DASH_FLOW_SAI.md

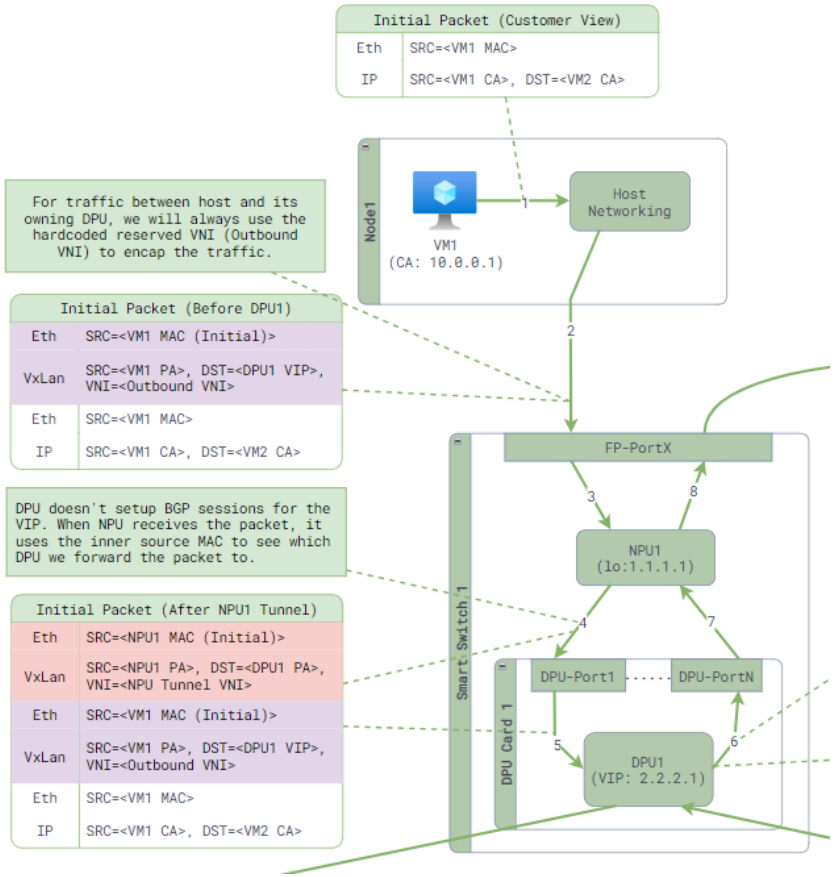
References

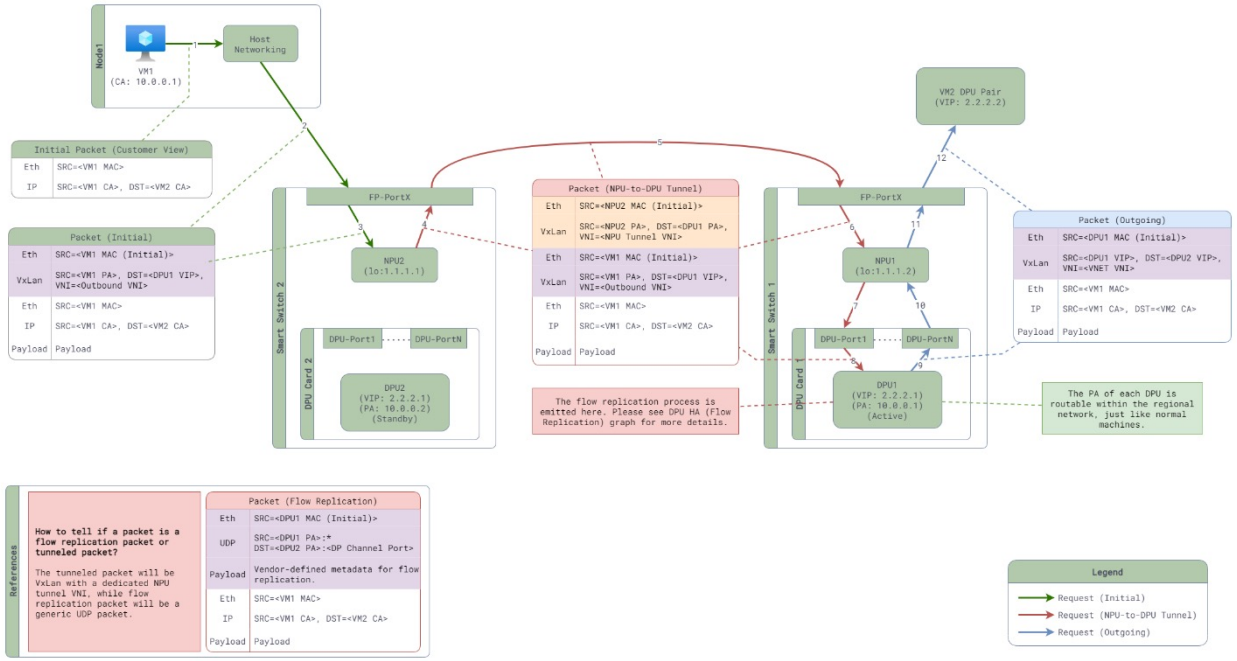
- SmartSwitch HA High Level Design doc: https://github.com/r12f/SONiC/blob/user/r12f/ha/doc/smart-switch/high-availability/smart-switch-ha-hld.md.
- DASH High Level Design doc: https://github.com/sonic-net/DASH/blob/main/documentation/general/dash-high-level-design.md.
- DASH scale design: https://github.com/sonic-net/DASH/blob/main/documentation/general/dash-sonic-hld.md.



Appendix

NPU-to-DPU traffic tunneling





State	Definition	Receive traffic from NPU?	Make decision?	Handling old flow?	Respond flow sync?	Init flow sync?	Init Bulk sync?
Dead	HA participant is just getting created, and not connected yet.	No	Drop	Drop	No	No	No
Connecting	Connecting to its peer.	No	Drop	Drop	No	No	No
Connected	Connected to its peer, but starting primary election to join the HA set.	No	Drop	Drop	No	No	No
InitializingToActive	Connected to pair for the first time, voted to be active.	No	Drop	Drop	No	No	No
InitializingToStandby	Connected to pair for the first time, voted to be standby.	No	Tunneled to pair	Tunneled to pair	Yes	No	No
Destroying	Preparing to be destroyed. Waiting for existing traffic to drain out.	No	Tunneled to pair	Tunneled to pair	No	No	No
Active	Connected to pair and act as decision maker.	Yes	Yes	Yes	No	Yes	Yes
Standby	Connected to pair and act as backup flow store.	No	Tunneled to pair	Tunneled to pair	Yes	No	No
Standalone	Heartbeat to pair is lost. Acting like a standalone setup.	Yes	Yes	Yes	Yes	No	No
SwitchingToActive	Connected and preparing to switch over to active.	No	Yes	Yes	Yes	Yes	No
SwitchingToStandby	Connected, leaving active state and preparing to become standby.	Yes	Tunneled to pair	Tunneled to pair	Yes	No	No