# IN-NETWORK AGGREGATION FOR SHARED MACHINE LEARNING CLUSTERS

## MLSys 2021

DML GROUP MEETING
11.12

# OUTLINE

- THE CASE AND CHALLENGES FOR IN-NETWORK AGGREGATION
- PANAMA OVERVIEW
- AGGREGATION ACCELERATOR DESIGN
- CONGEST CONTROL PROTOCOL
- LOAD-BALANCING MECHANISM
- SIMULATION

# THE CASE FOR IN-NETWORK AGGREGATION

- All-reduce in data-parallel training places significant pressure on the network
- Example:  a training job with 1,000 workers and a 1GB DNN model size can generate around 2PB in 1,000 iterations
- A proposed solution: In-network aggregation
- Two reasons for limited improvement:

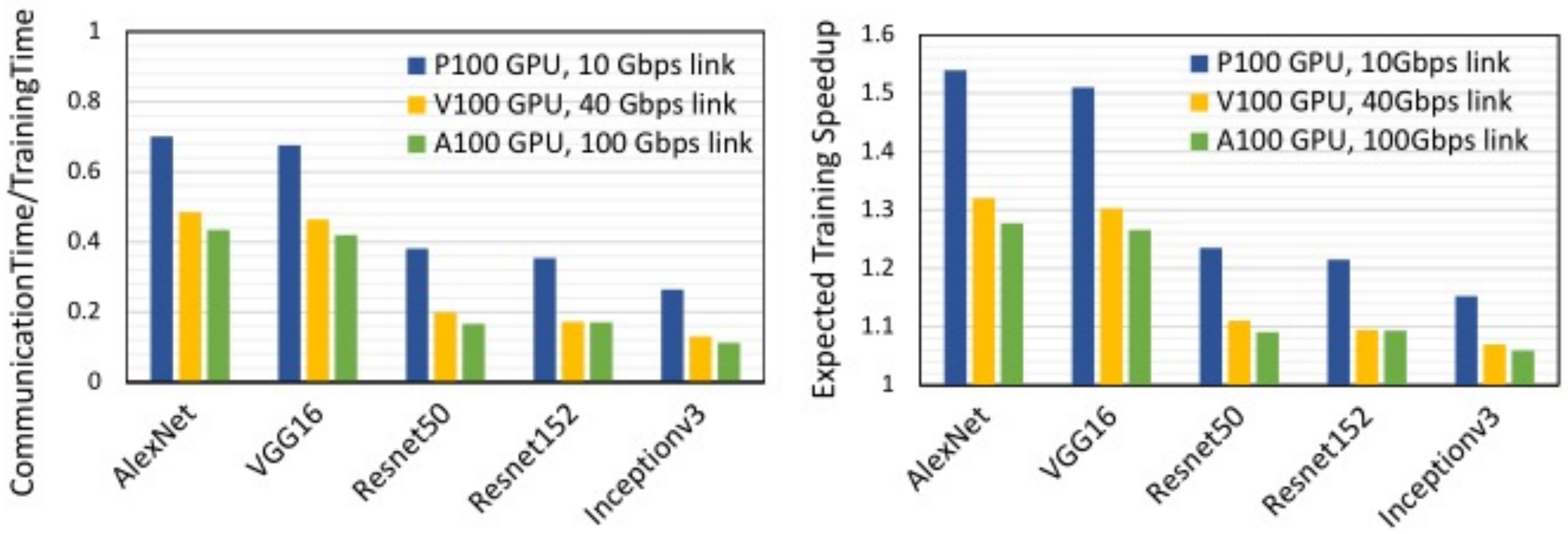
1) Compute time occupies a significant chunk

2) the maximum theoretical improvement is limited (Klenk et al.,2020)

# THE CASE FOR IN-NETWORK AGGREGATION

- The fraction of training time spent on communication reduced as the network becomes faster

- The benefits of reducing communication is less pronounced




(a) Communication ratio (b) Expected speedup

Figure 1: The expected speed-up for a single in-network aggregation job is limited by the ratio of communication time over total training time.

# THE CASE FOR IN-NETWORK AGGREGATION

- Reduce the overall data-parallel traffic injected into the network.

- Free up network bandwidth for non-data-parallel traffic.

- Non-Aggregation flows are not candidates for in-network communication, but benefit from it.

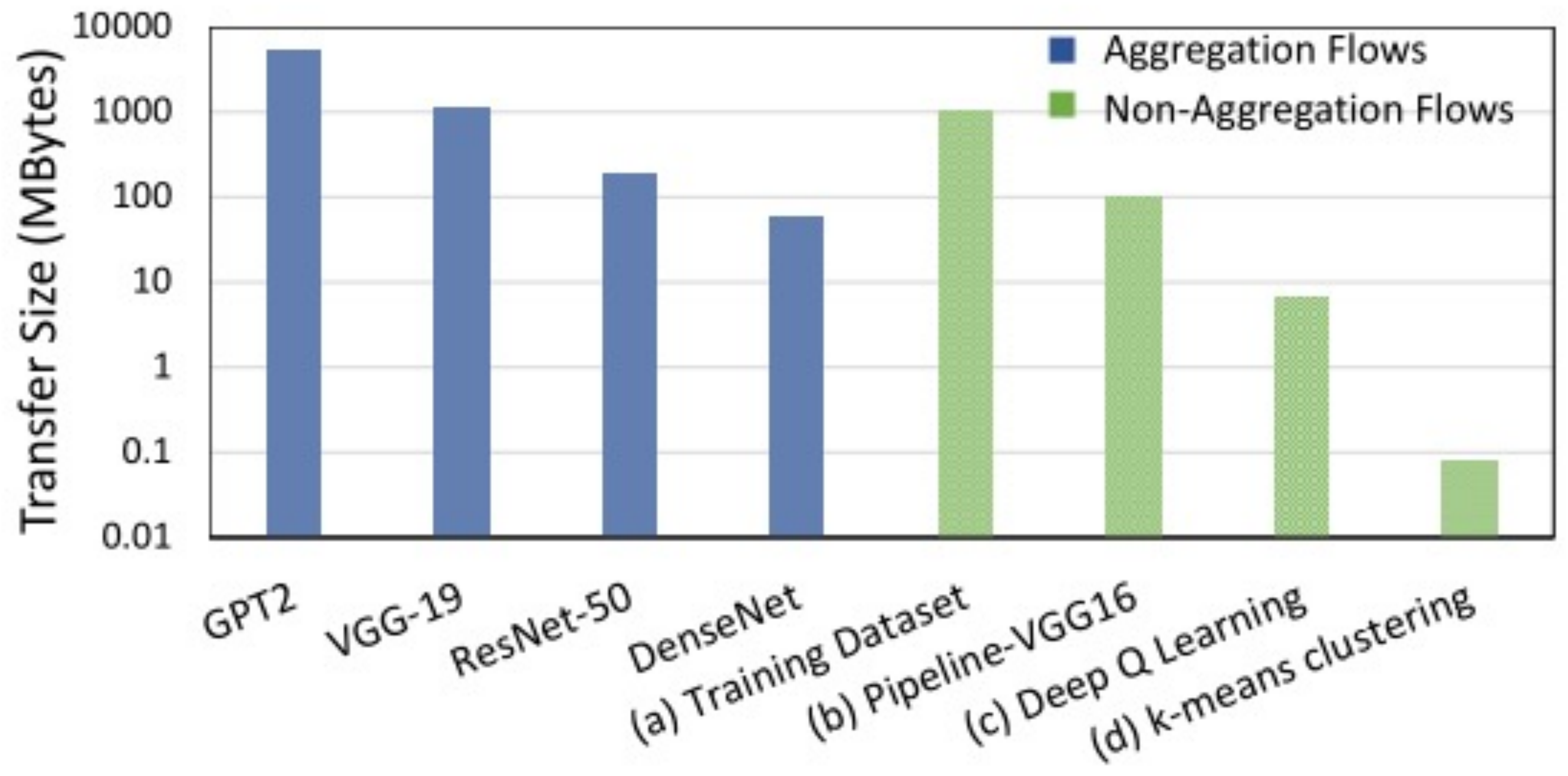



Figure 2: Data transfer sizes in a shared ML cluster.

# PANAMA OVERVIEW

## **P**rogr**A**mmable **N**etwork **A**rchitecture for **M**L **A**pplications

- Key components:

  1) Aggregation Accelerator

  2) Congestion Control Protocol

  3) Load-balancing Mechanism

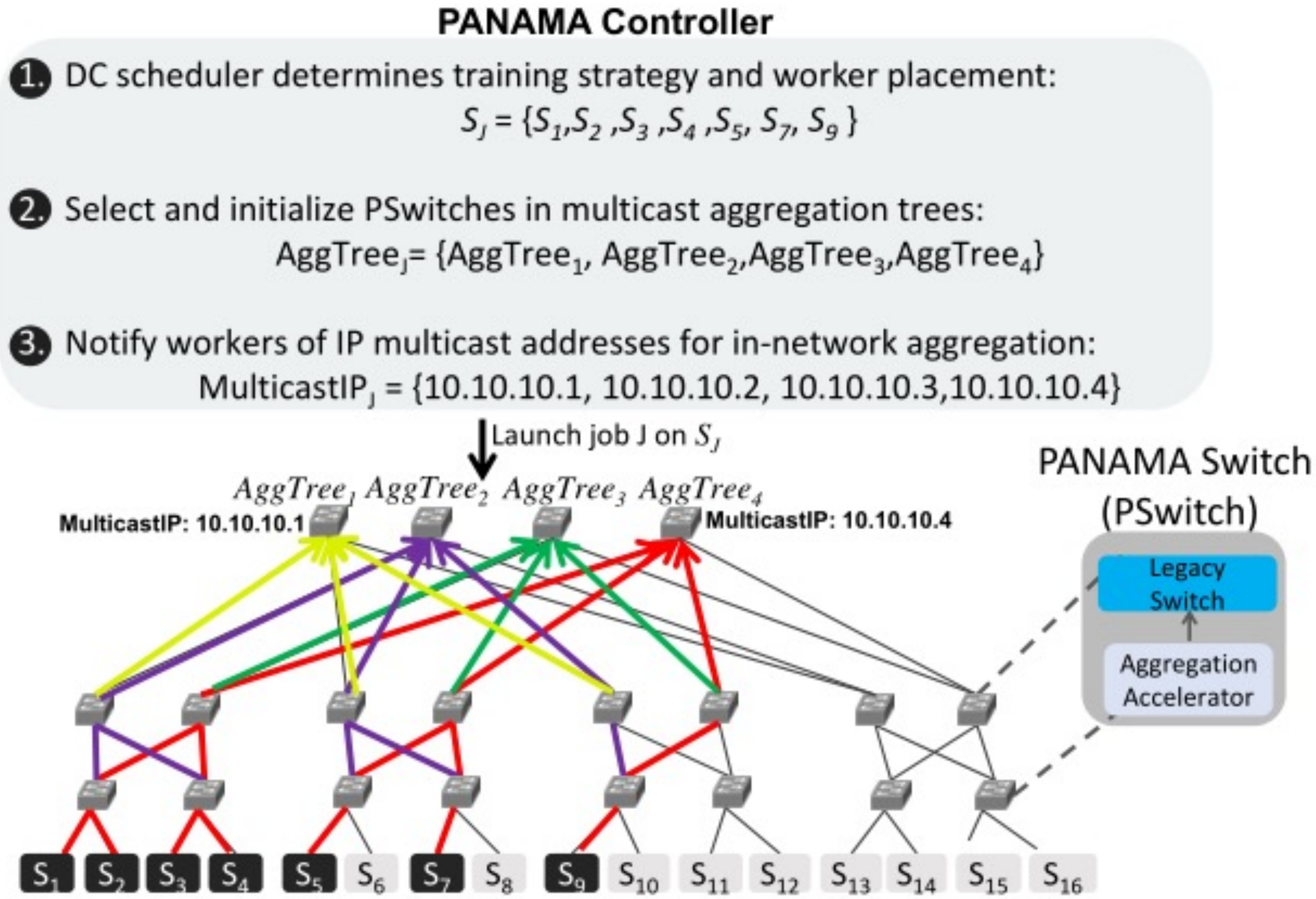
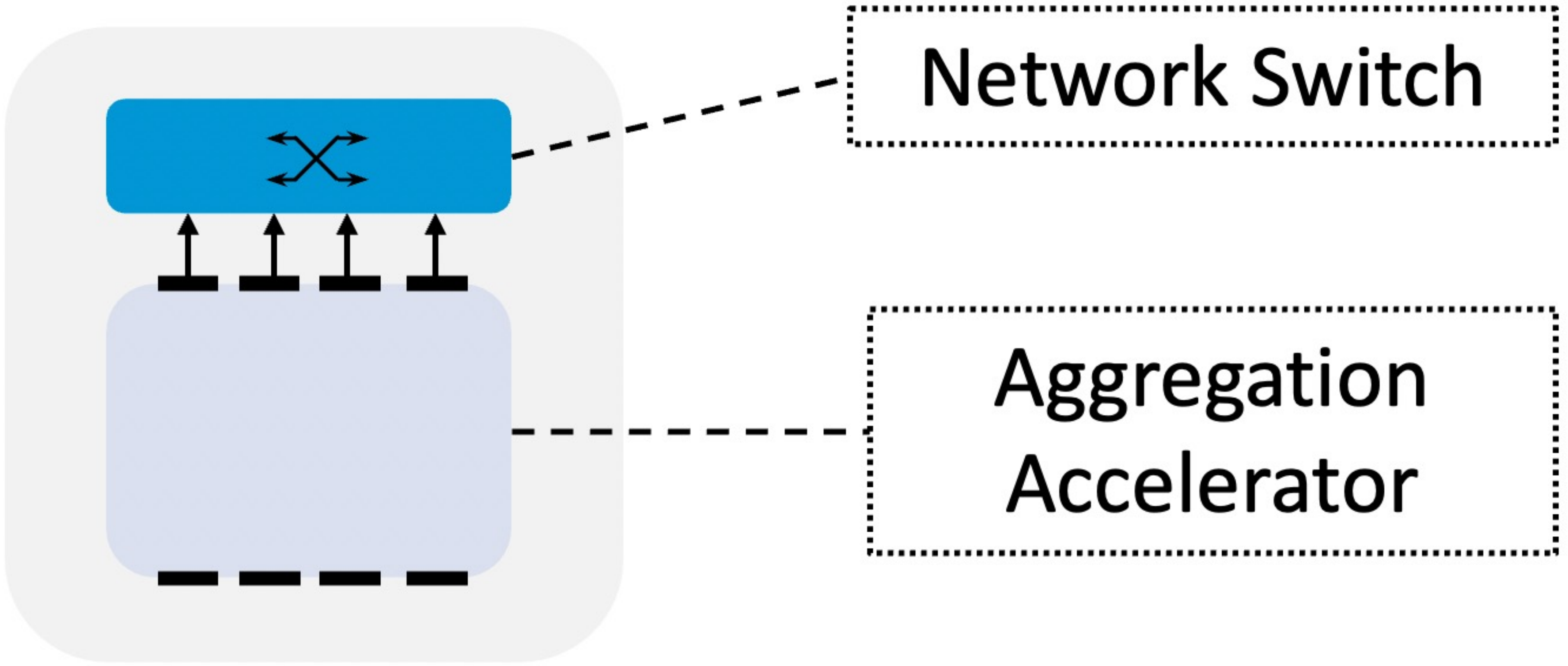



Figure 3: High-level workflow of PANAMA.

# AGGREGATION ACCELERATOR

PANAMA Switch
(PSwitch)




- Routing and switching
- In-network aggregation specialization
  - ➢ Support for floating point computation

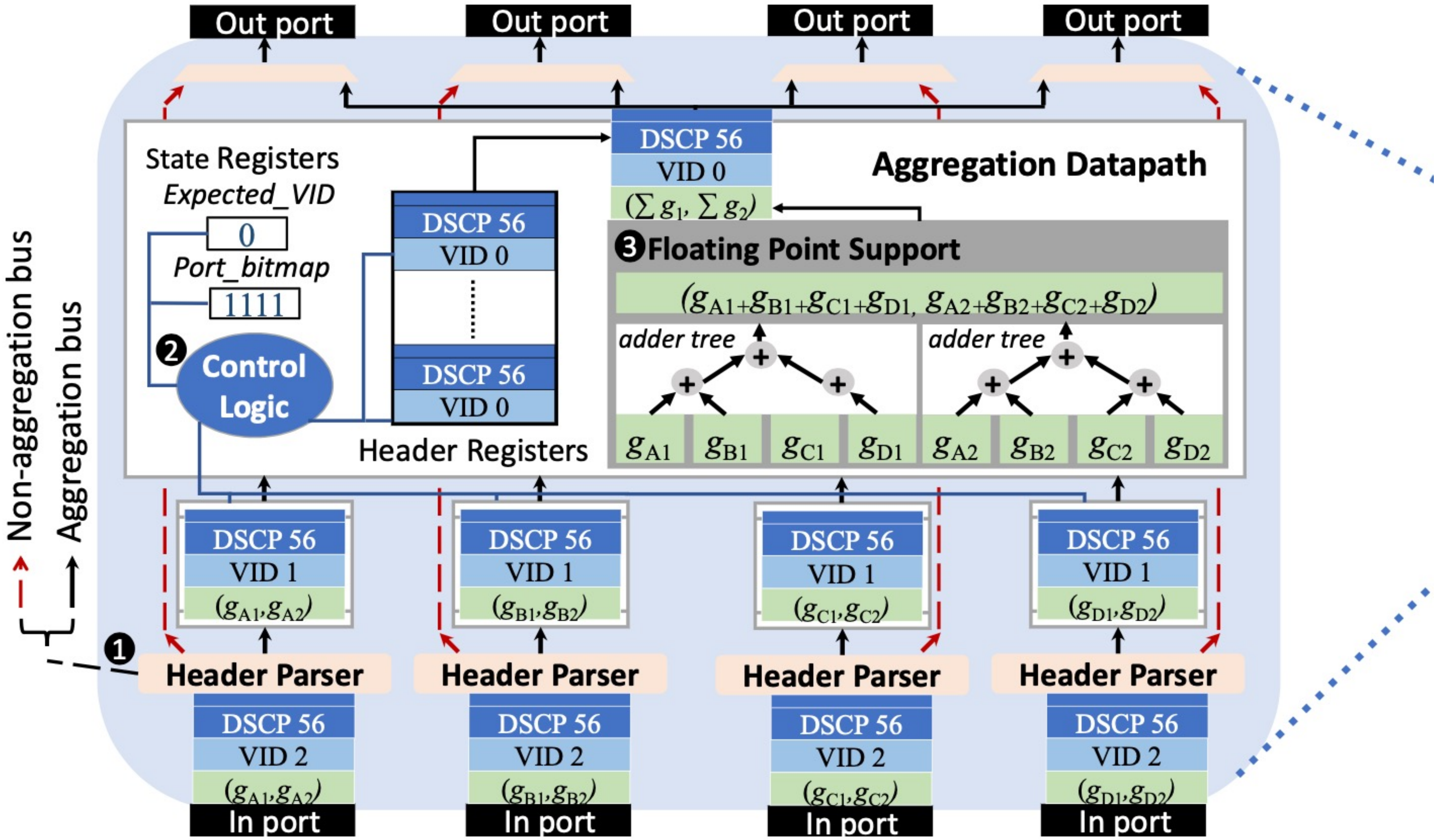
# AGGREGATION ACCELERATOR

- Key components:

  1) Packet header parser

  2) Control logic

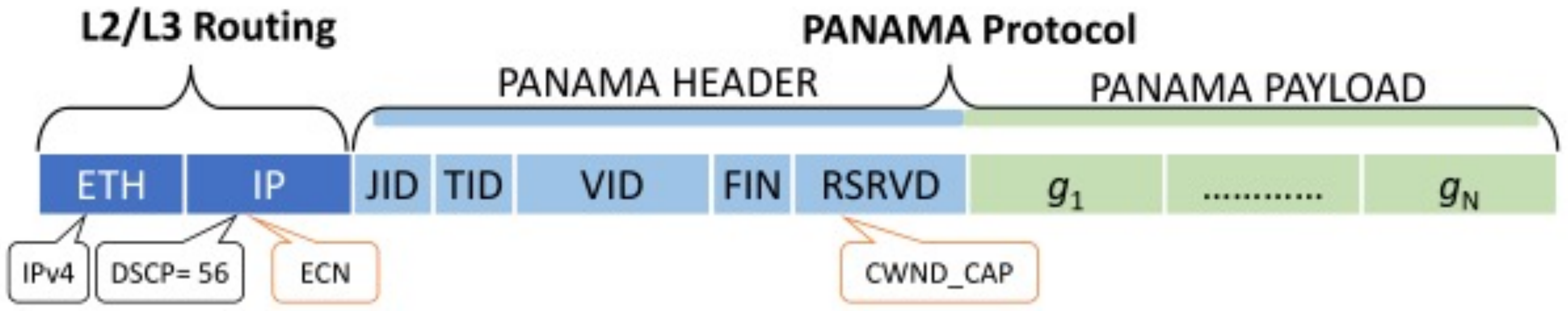
  3) Floating point support




(a) Aggregation accelerator architecture.

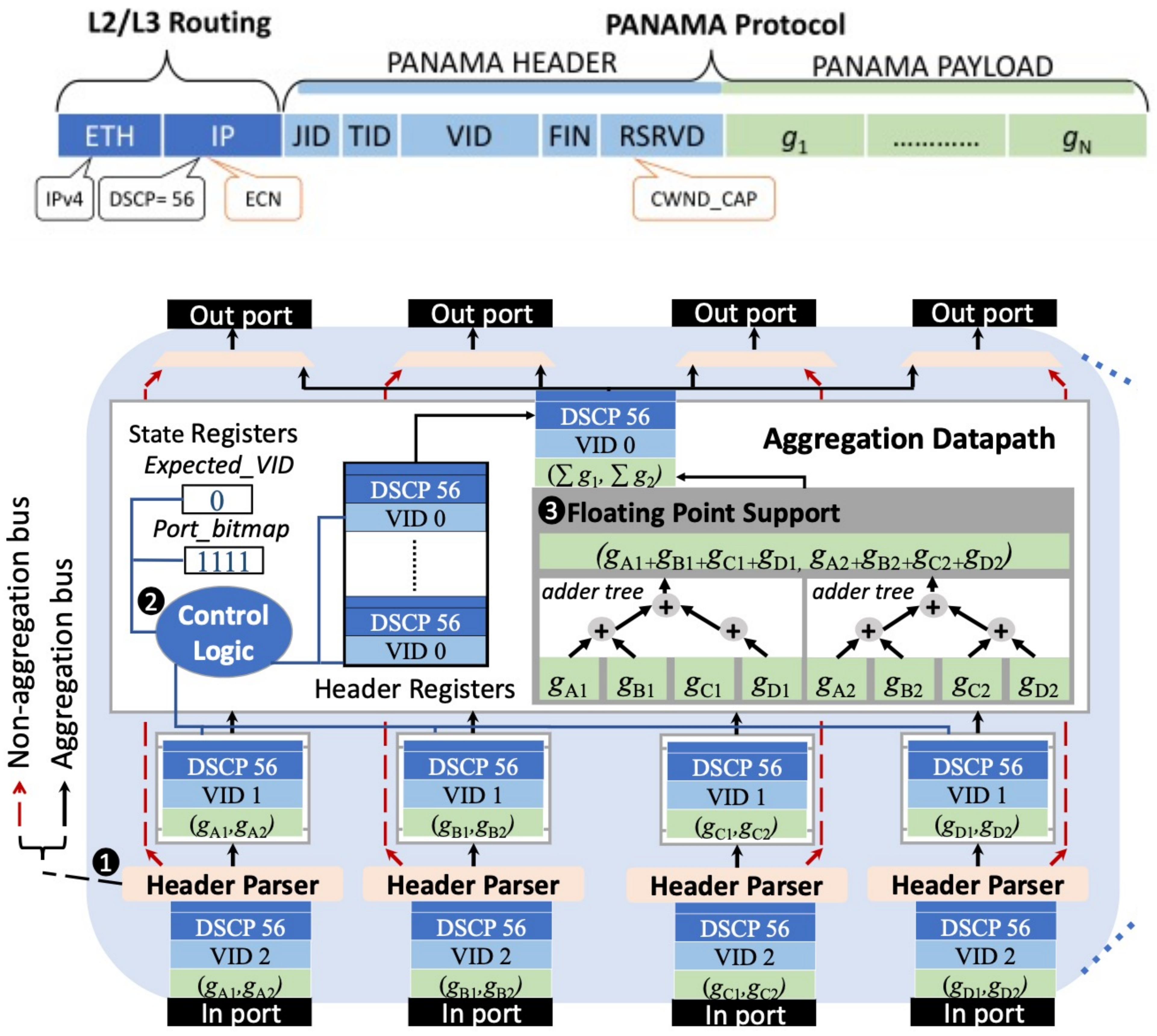
# PACKET HEADER PARSER

- Inspect EtherType and DSCP fields
- IPv4 packets
- DSCP field value equal to 56




# CONTROL LOGIC

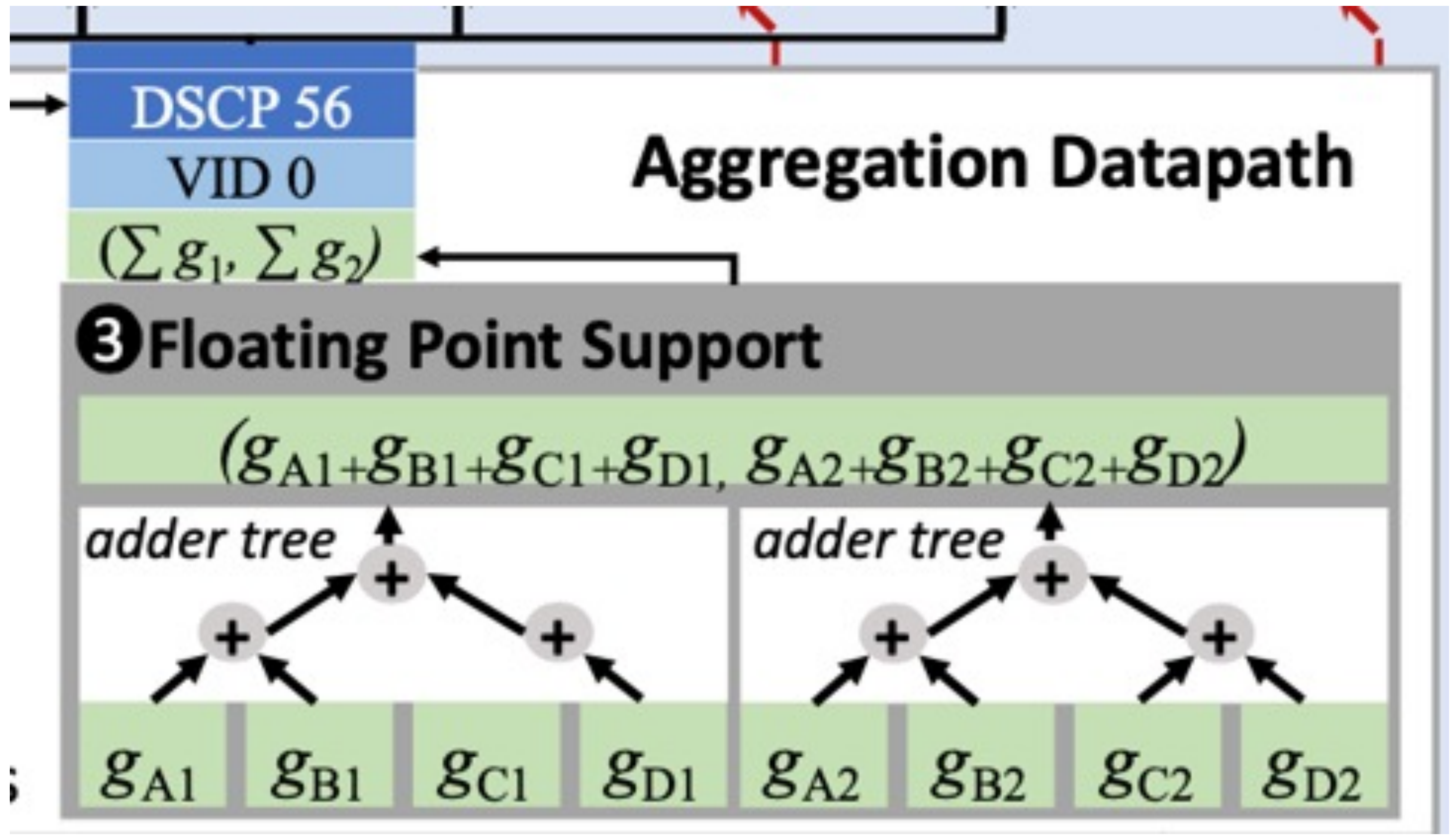
- State Registers: Ports_bitmap and ExpectedVID
- Job ID (JID) and Tree ID (TID) fields are used to identify the Portsbitmap and ExpectedVID
- FIN field are used to notify accelerators that they have sent all aggregation packets



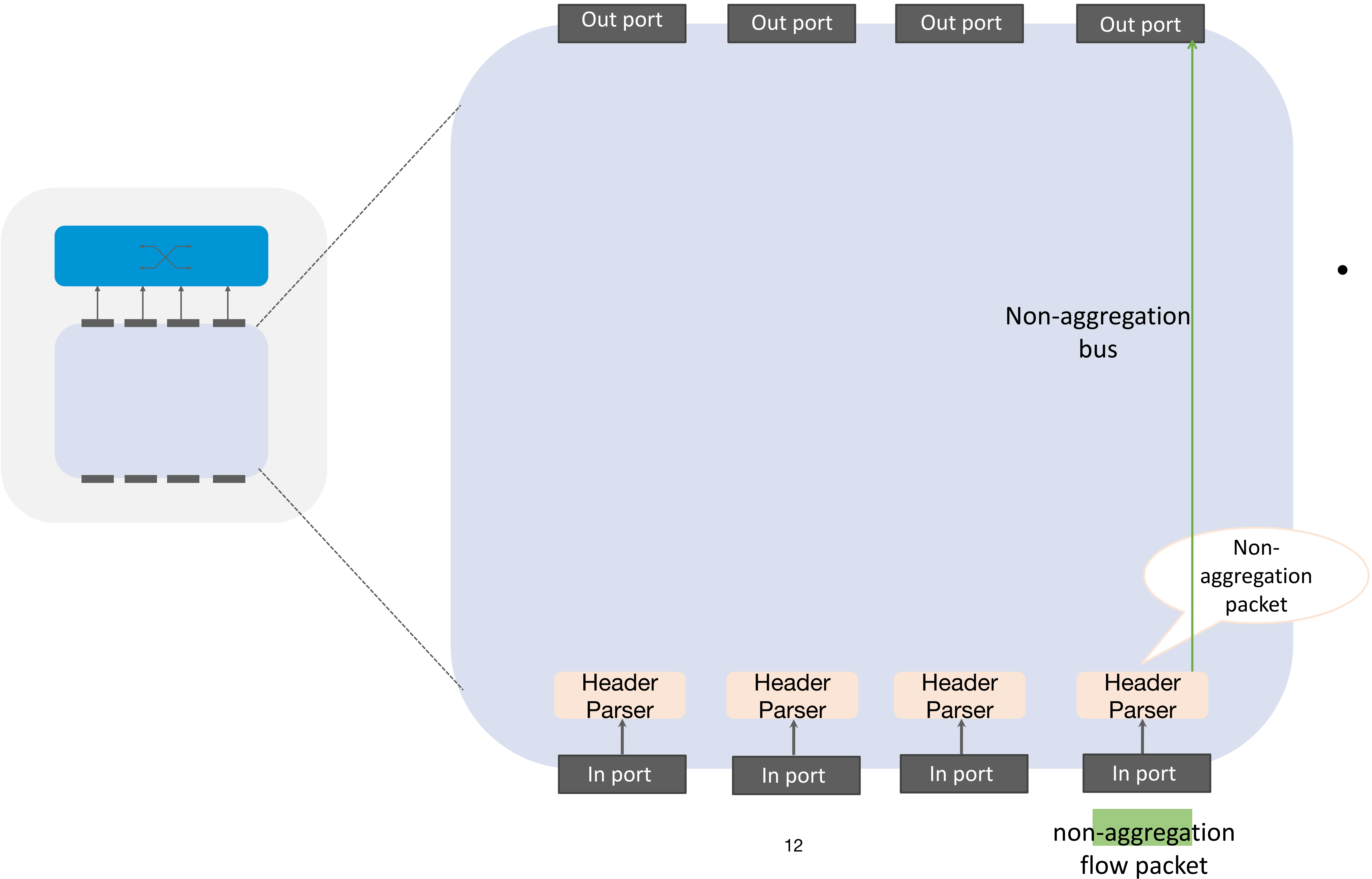

(a) Aggregation accelerator architecture.

# FLOATING POINT SUPPORT

- Aggregation payload streamed from buffer to multiple adder trees.
- Adder trees operate in parallel.
- The number of parallel adder trees is proportional to the number of gradients that can be carried in the data bus.



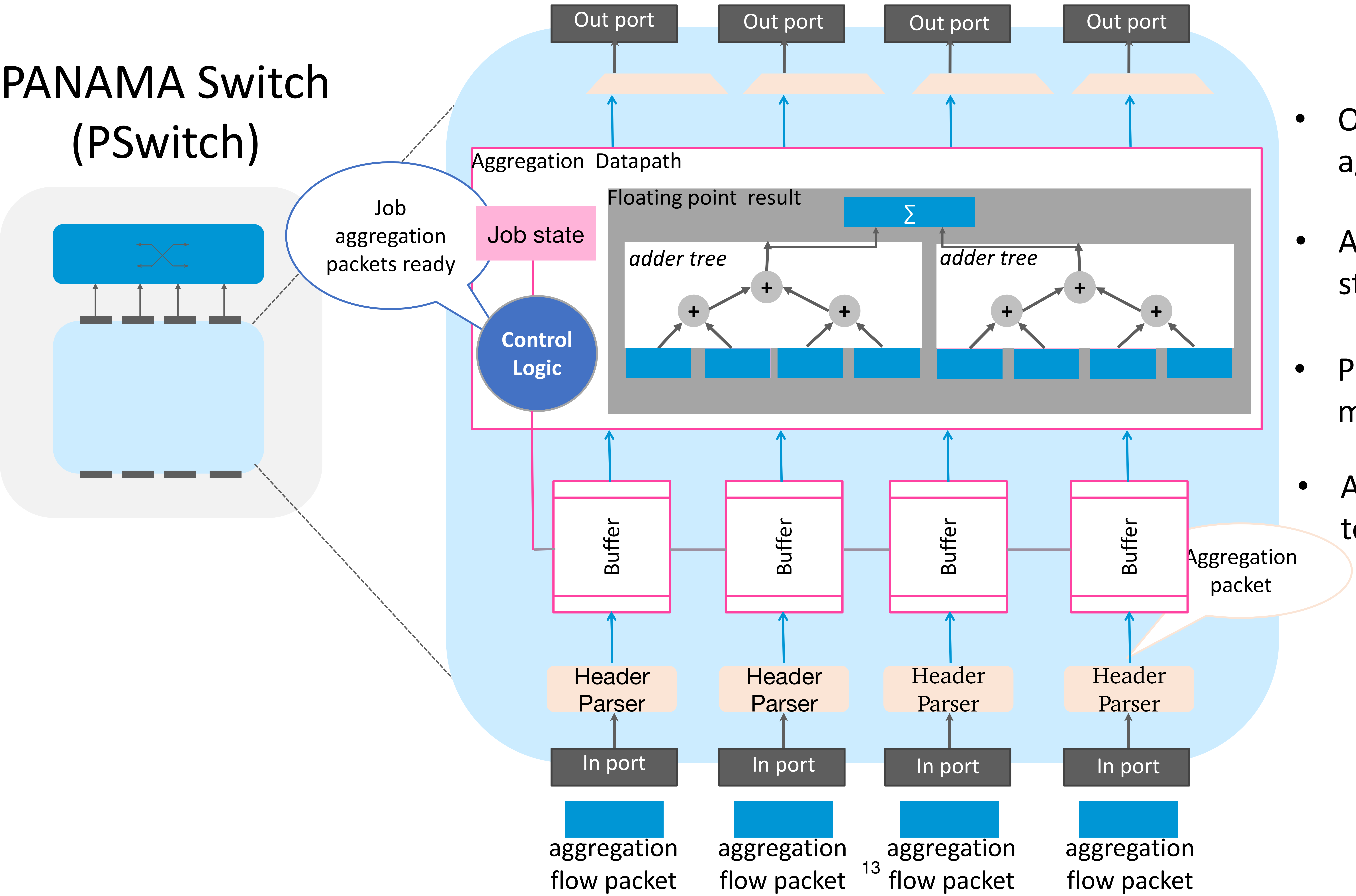
# AGGREGATION ACCELERATOR




- Non-aggregation packets are sent immediately to switch

# AGGREGATION ACCELERATOR

PANAMA Switch (PSwitch)




- On-chip buffers are dedicated to aggregation packets
- Aggregation packets are streamed from buffers
- Parallel additions performed to match required throughput
- Aggregation result packet sent to switch

# CONGESTION CONTROL PROTOCOL

- Requirements

  1) Support for multipoint communication.

  2) Small buffers.

  3) Compatibility with legacy protocols.

  4) Lossless operation.

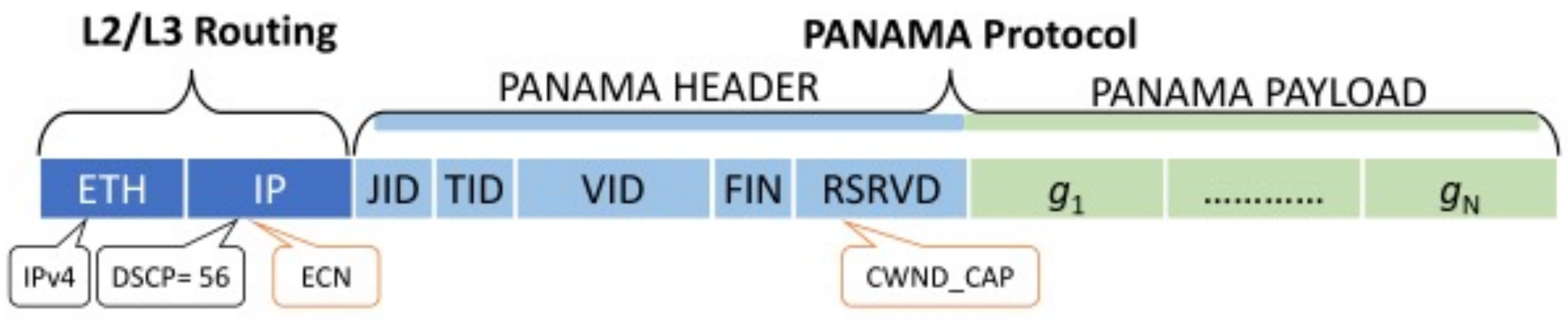
# CONGESTION CONTROL PROTOCOL

- Key components

  1) Implicit acknowledgments

  2) ECN marking

  3) Congestion window capping




**Parameters:** $N$: number of job aggregation trees, *ssthresh*: initial slow start threshold, $g$: weighting factor for fraction of ECN marked result packets, $\alpha$: moving average of ECN marked fraction of packets.

---

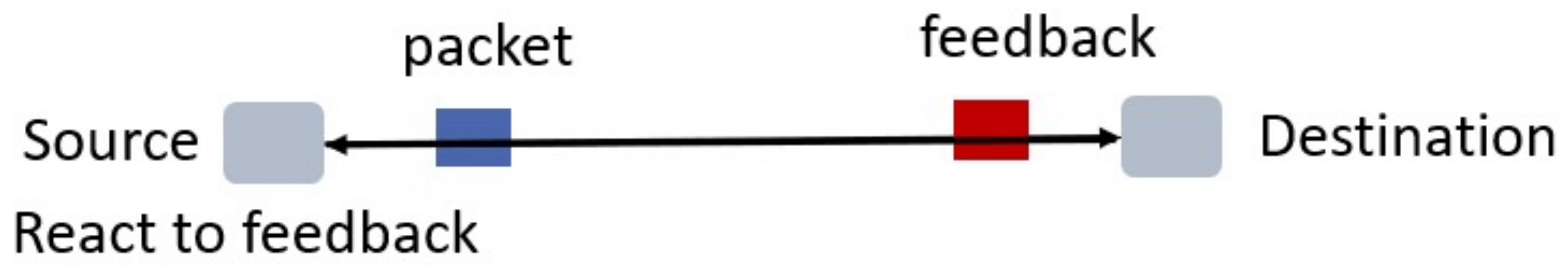
**Initialization** ▷ Independent aggregation tree congestion control
**for** $i = 1 : N$ **do**
  $ssthresh_i \leftarrow 64$
  $\alpha_i \leftarrow 1$
  $cwnd_i \leftarrow 2$
**end for**

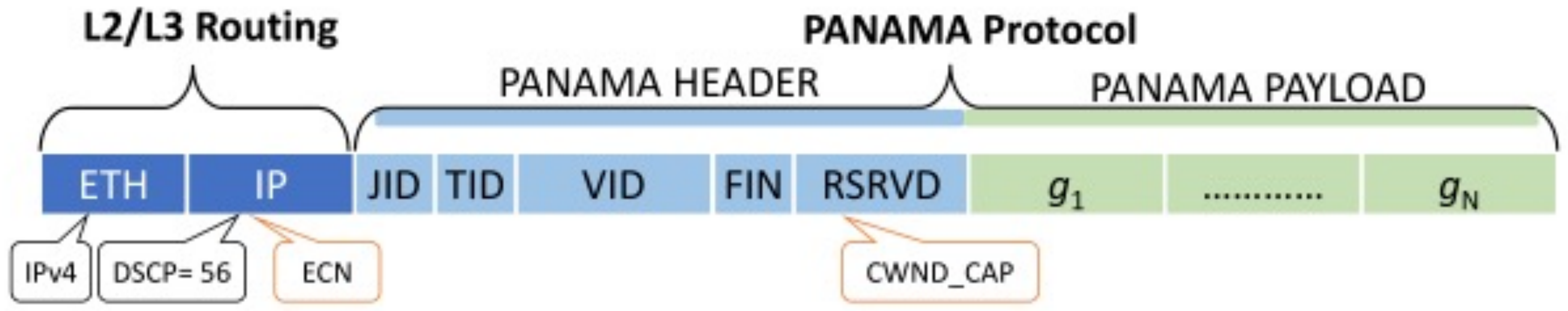
---

**Input:** **Aggregation Result Packet (pkt)** ▷ Implicit acknowledgment
  $i \leftarrow pkt.treeid$
  $rcvd\_agg\_packets_i \leftarrow rcvd\_agg\_packets_i + 1$
  $ecn\_count_i \leftarrow ecn\_count_i + pkt.ecn$ ▷ ECN marking
**if** $rcvd\_agg\_packets_i == cwnd_i$ **then**
  $\alpha_i \leftarrow \alpha_i(1-g) + g \times \frac{ecn\_count_i}{cwnd_i}$
  $rcvd\_agg\_packets_i = 0$
  **if** $ecn\_count_i == 0$ **then** ø Window size increase
    **if** $cwnd_i < ssthresh_i$ **then**
      $cwnd_i \leftarrow 2 \times cwnd_i$
    **else**
      $cwnd_i \leftarrow cwnd_i + 1$
    **end if**
  **else** ø Window size decrease
    $cwnd_i \leftarrow cwnd_i \times (1 - \frac{\alpha_i}{2})$
    $ssthresh_i \leftarrow cwnd_i$
  **end if**
**end if**
**if** $cwnd_i > pkt.cwnd\_cap$ **then**
  $cwnd_i \leftarrow pkt.cwnd\_cap$ ▷ Congestion window capping
**end if**

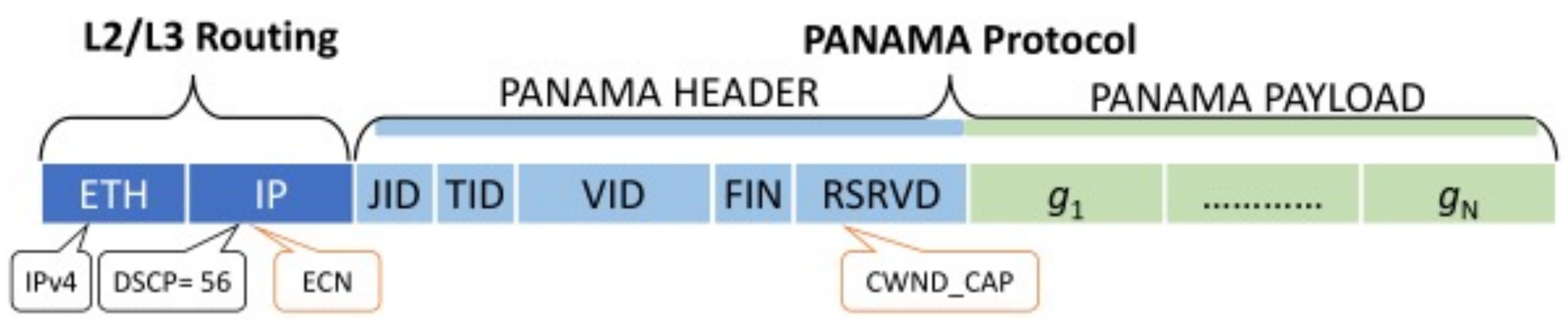
# IMPLICIT ACKNOWLEDGMENTS

- Congestion control protocols assume point-to-point communication between servers
- In-network aggregation requires many-to-many communication between different entities
- Workers treat aggregation result packets as implicit acknowledgement signals to increase or decrease the window size



# ECN MARKING




- Pswitch perform a bitwise OR operation on the ECN field to mirror the ECN bit into the IP header
- Workers adjust the sending rate according to the ECN field
- Ensure the congestion window grows and shrinks in a synchronized fashion across workers in the aggregation tree
- Guarantee compatibility with existing legacy protocols

**Parameters:** $N$: number of job aggregation trees, *ssthresh*: initial slow start threshold, $g$: weighting factor for fraction of ECN marked result packets, $\alpha$: moving average of ECN marked fraction of packets.

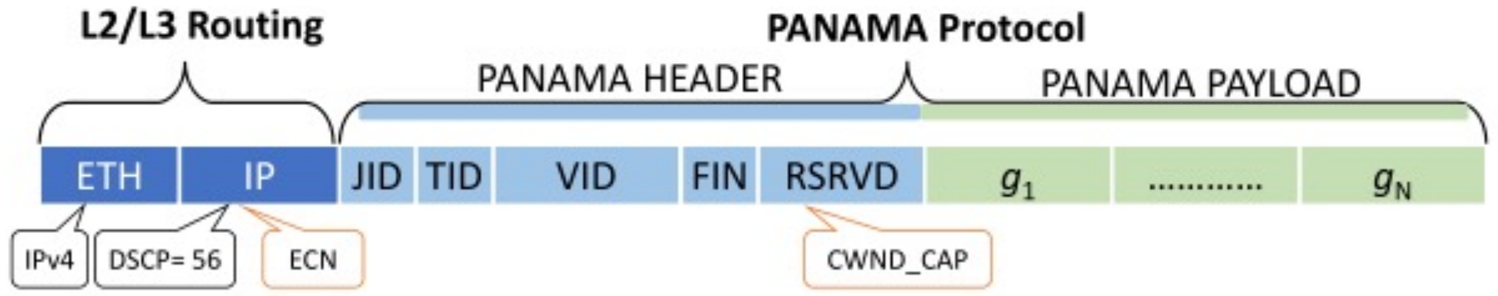
---

**Initialization** ▷ Independent aggregation tree congestion control
**for** $i = 1 : N$ **do**
  $ssthresh_i \leftarrow 64$
  $\alpha_i \leftarrow 1$
  $cwnd_i \leftarrow 2$
**end for**

---

**Input:** **Aggregation Result Packet (pkt)** ▷ Implicit acknowledgment
  $i \leftarrow pkt.treeid$
  $rcvd\_agg\_packets_i \leftarrow rcvd\_agg\_packets_i + 1$
  $ecn\_count_i \leftarrow ecn\_count_i + pkt.ecn$ ▷ ECN marking
**if** $rcvd\_agg\_packets_i == cwnd_i$ **then**
  $\alpha_i \leftarrow \alpha_i(1 - g) + g \times \frac{ecn\_count_i}{cwnd_i}$
  $rcvd\_agg\_packets_i = 0$
  **if** $ecn\_count_i == 0$ **then** ø Window size increase
    **if** $cwnd_i < ssthresh_i$ **then**
      $cwnd_i \leftarrow 2 \times cwnd_i$
    **else**
      $cwnd_i \leftarrow cwnd_i + 1$
    **end if**
  **else** ø Window size decrease
    $cwnd_i \leftarrow cwnd_i \times (1 - \frac{\alpha_i}{2})$
    $ssthresh_i \leftarrow cwnd_i$
  **end if**
**end if**
**if** $cwnd_i > pkt.cwnd\_cap$ **then**
  $cwnd_i \leftarrow pkt.cwnd\_cap$ ▷ Congestion window capping
**end if**

# CONGESTION WINDOW CAPPING




- Reserve 16 bits for cwndcap to capture the minimum available memory to store packets at the accelerators
- Each accelerator calculates its available memory and overwrites cwndcap
- Cwndcap is used as a cap on the maximum number of in-flight packets for each worker

**Parameters:** $N$: number of job aggregation trees, *ssthresh*: initial slow start threshold, $g$: weighting factor for fraction of ECN marked result packets, $\alpha$: moving average of ECN marked fraction of packets.

---

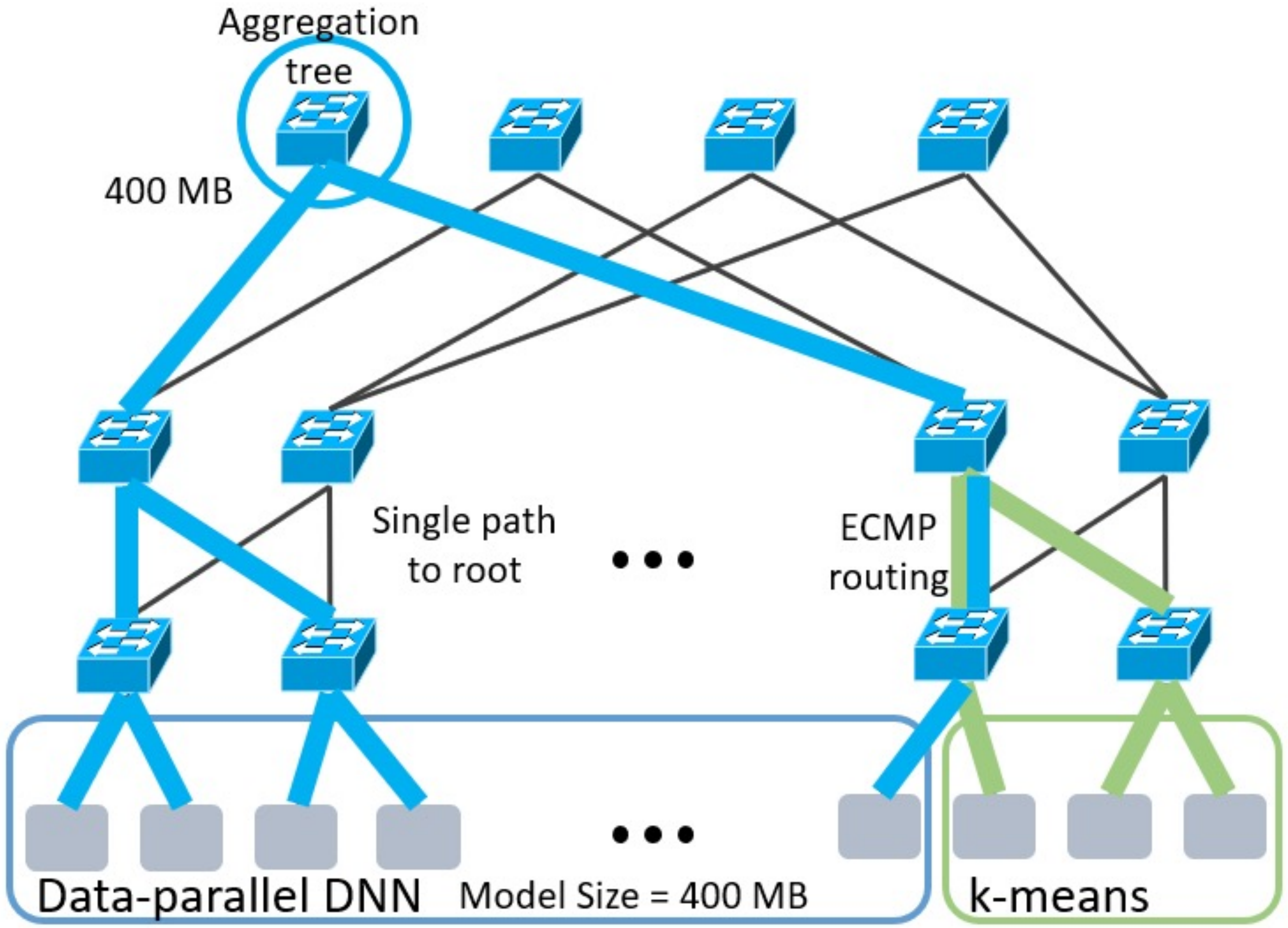
**Initialization** ▷ Independent aggregation tree congestion control
**for** $i = 1 : N$ **do**
  $ssthresh_i \leftarrow 64$
  $\alpha_i \leftarrow 1$
  $cwnd_i \leftarrow 2$
**end for**

---

**Input:** **Aggregation Result Packet (pkt)** ▷ Implicit acknowledgment
  $i \leftarrow pkt.treeid$
  $rcvd\_agg\_packets_i \leftarrow rcvd\_agg\_packets_i + 1$
  $ecn\_count_i \leftarrow ecn\_count_i + pkt.ecn$ ▷ ECN marking
**if** $rcvd\_agg\_packets_i == cwnd_i$ **then**
  $\alpha_i \leftarrow \alpha_i(1-g) + g \times \frac{ecn\_count_i}{cwnd_i}$
  $rcvd\_agg\_packets_i = 0$
  **if** $ecn\_count_i == 0$ **then** ø Window size increase
    **if** $cwnd_i < ssthresh_i$ **then**
      $cwnd_i \leftarrow 2 \times cwnd_i$
    **else**
      $cwnd_i \leftarrow cwnd_i + 1$
    **end if**
  **else** ø Window size decrease
    $cwnd_i \leftarrow cwnd_i \times (1 - \frac{\alpha_i}{2})$
    $ssthresh_i \leftarrow cwnd_i$
  **end if**
**end if**
**if** $cwnd_i > pkt.cwnd\_cap$ **then**
  $cwnd_i \leftarrow pkt.cwnd\_cap$ ▷ Congestion window capping
**end if**

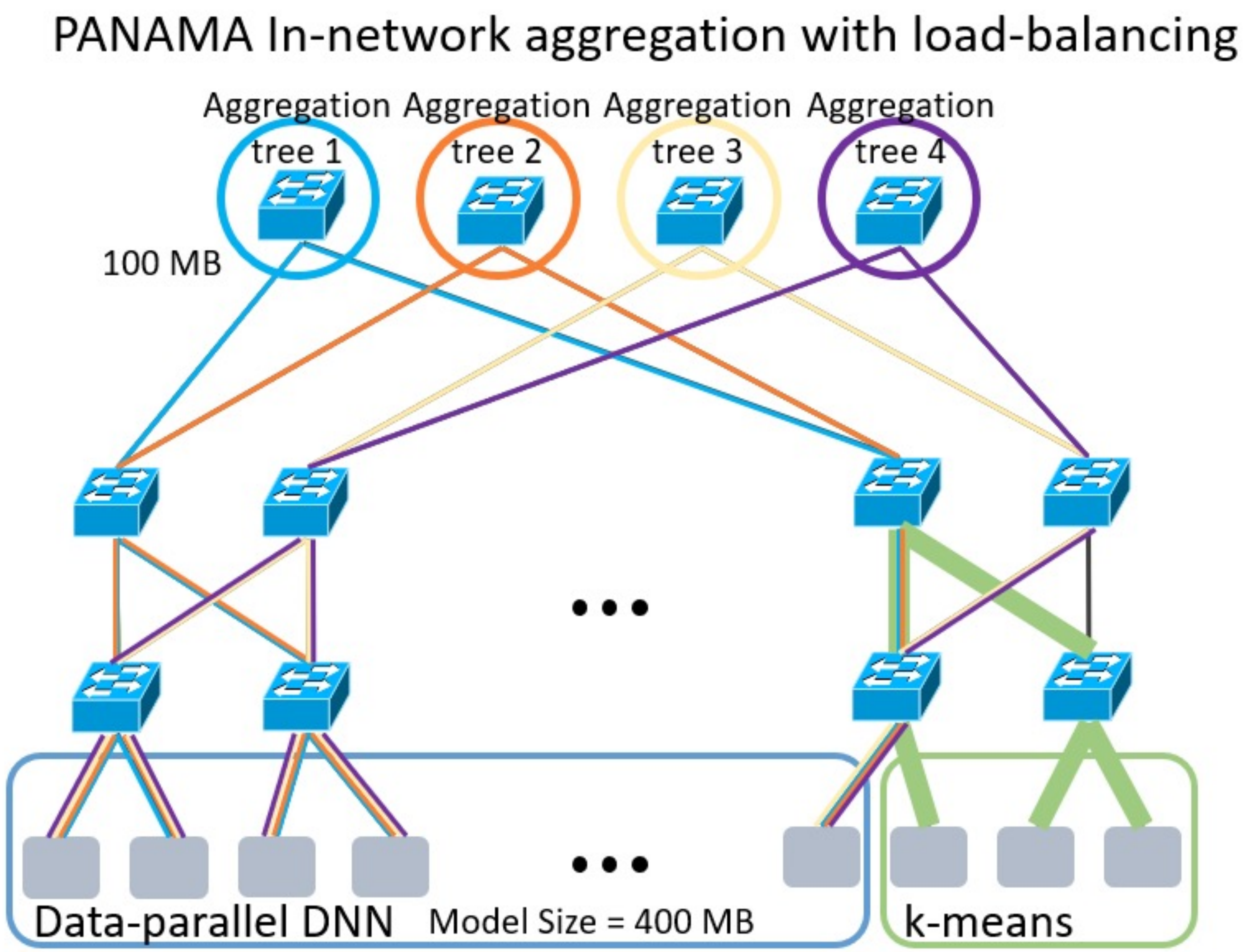
# LOAD-BALANCING MECHANISM

- ECMP could create imbalance
- PANAMA utilizes multiple aggregation trees per training job to spread the traffic across multiple paths



# LOAD-BALANCING MECHANISM

- Provide a set of IP multicast addresses representing the selected aggregation trees for a job.
- Workers distribute the gradient packets to different trees in a round robin fashion.

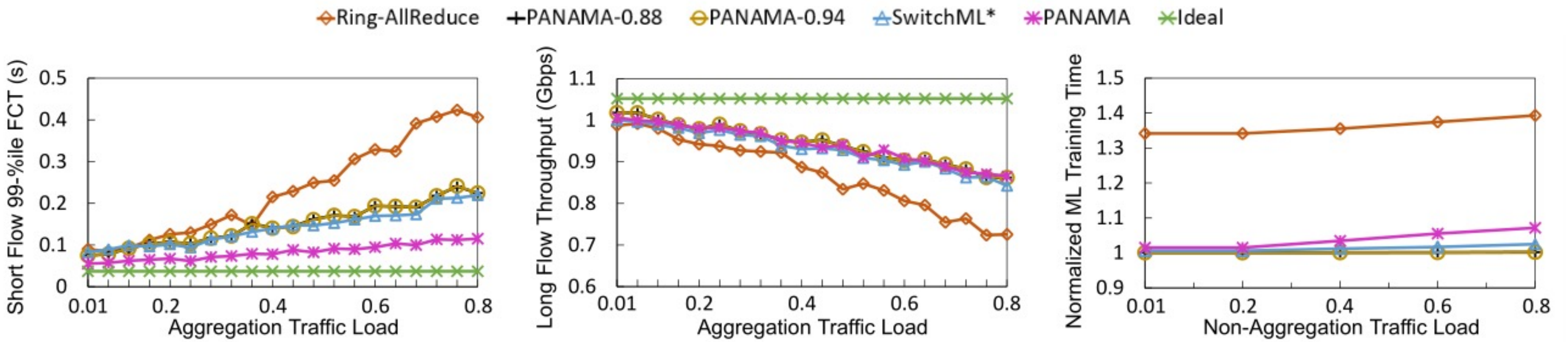


# SIMULATION

- PANAMA reduces the impact of aggregation traffic on short flows (<40 MB).

- In-network schemes improve throughput of long flows

- In-network schemes improve ML job completion time.




(a) 99%-tile FCT for non-aggregation short flows (<40 MB)
(b) Average throughput for non-aggregation long flows
(c) Average training time for ML jobs for increasing network load

Figure 8: Performance of PANAMA in a shared data center setting compared to other training schemes.

# SIMULATION

- PANAMA outperforms all other scenarios since it uses all four aggregation trees.

- The performance of in-network aggregation with a single aggregation tree can be worse than that of Ring-All Reduce.

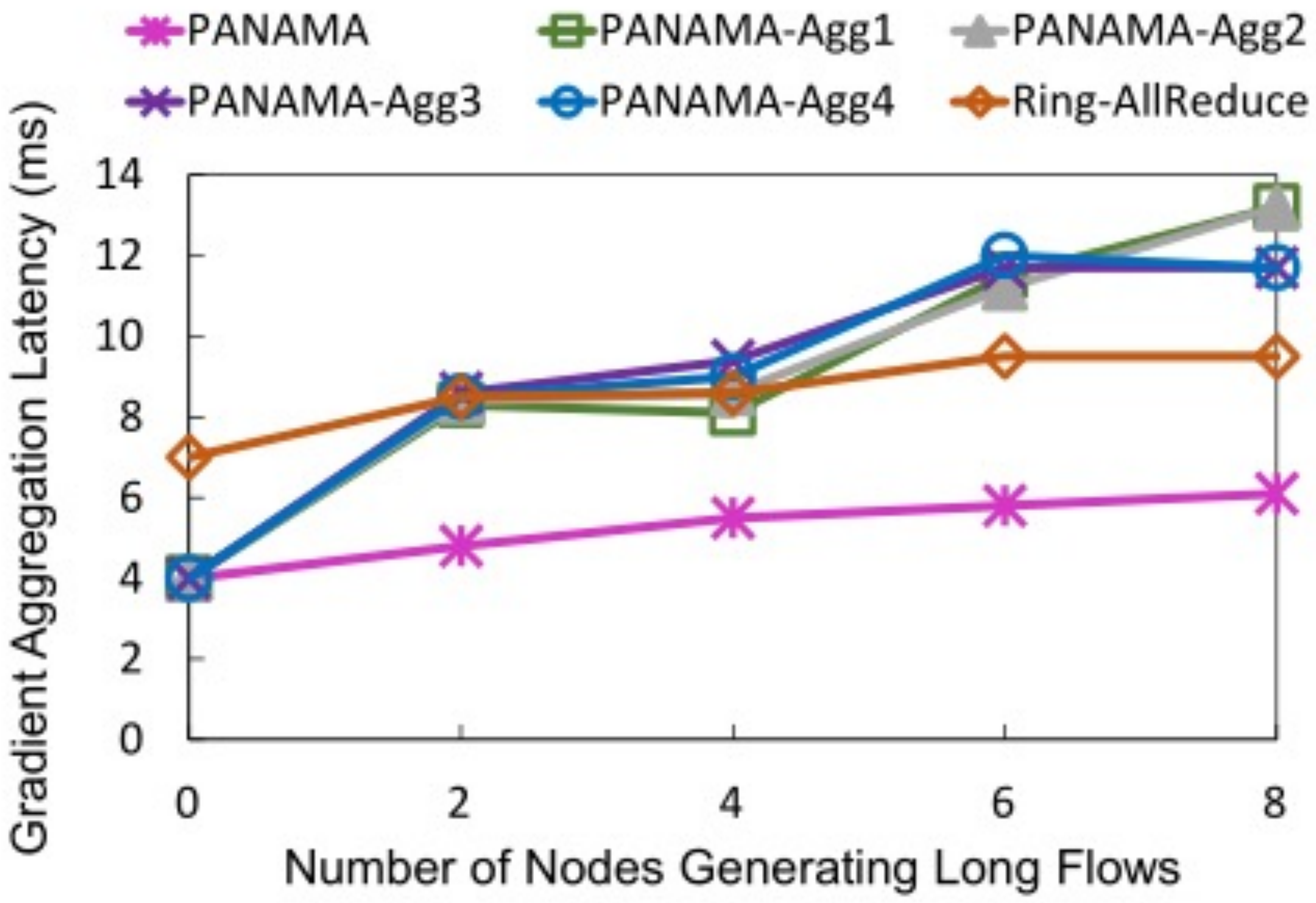
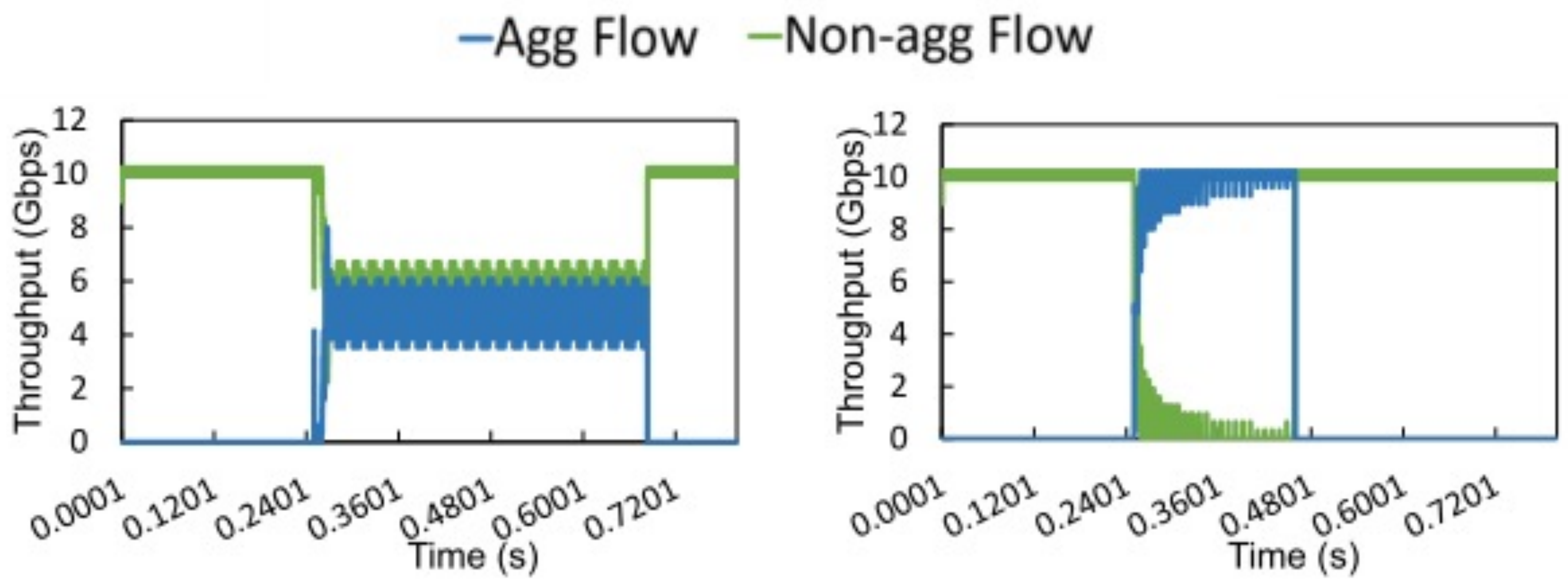



Figure 9: Impact of load balancing.

# SIMULATION

- PANAMA shares the link bandwidth equally between both flows.

- Without PANAMA's congestion control protocol, the latency-sensitive non-aggregation flow is starved.




(a) With PANAMA (b) Without PANAMA

Figure 10: PANAMA achieves a fair bandwidth allocation between aggregation and non-aggregation flows.

# THANKS!