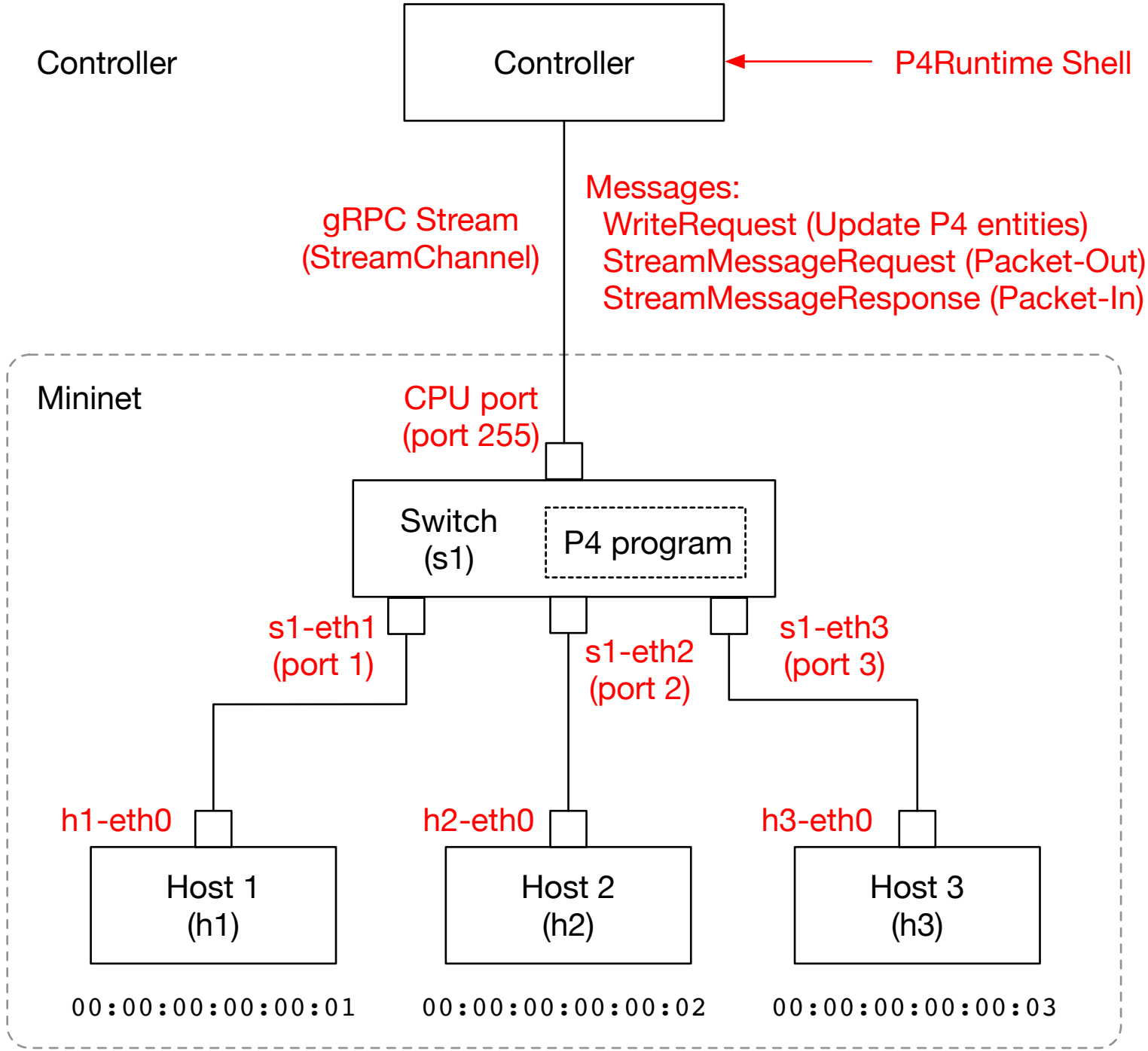
Controller
Controller
P4Runtime Shell
gRPC Stream
(StreamChannel)
Messages:
WriteRequest (Update P4 entities)
StreamMessageRequest (Packet-Out)
StreamMessageResponse (Packet-In)
Mininet
CPU port
(port 255)
Switch
(s1)
P4 program
s1-eth1
(port 1)
s1-eth2
(port 2)
s1-eth3
(port 3)
h1-eth0
h2-eth0
h3-eth0
Host 1
(h1)
Host 2
(h2)
Host 3
(h3)
00:00:00:00:00:01
00:00:00:00:00:02
00:00:00:00:00:03

実験の手順と概要

状態

コントローラ内のMACテーブル
（ shell.py - macTable[ ] ）

P4 table
“l2_match_table”

初期状態

host mac	port
(empty)	

match field		forward port
dest	src	
(empty)		

`ICMP echo request h1 -> h2`

・unmatch なので Packet-In
・まず `flooding` する
・`src mac (h1)` が未知なので ingress port とともに記録する

host mac	port
00:00:00:00:00:01	1

`ICMP echo request h2 -> h1`

・unmatch なので Packet-In
・まず `flooding` する
・`src mac (h2)` が未知なので ingress port とともに記録する

・src, dst が揃ったので往復のフローエントリを追加

host mac	port
00:00:00:00:00:01	1
00:00:00:00:00:02	2

match field		forward port
dest	src	
00:00:00:00:00:01	00:00:00:00:00:02	1
00:00:00:00:00:02	00:00:00:00:00:01	2

これ以降の `h1 <-> h2` 通信は match するので Packet-In / Flooding が生じない

Typical Experiment

STATE

MAC table in the controller
(shell.py - macTable[ ])

P4 table
“l2_match_table”

Initial State

host mac	port
(empty)	

match field		forward port
dest	src	
(empty)		

`ICMP echo request h1 -> h2`

- unmatch makes Packet-In
- 1st, do flooding
- src mac (h1) is new, record it with ingress port

host mac	port
00:00:00:00:00:01	1

`ICMP echo request h2 -> h1`

- unmatch makes Packet-In
- do flooding
- src mac (h2) is new, record it with ingress port

host mac	port
00:00:00:00:00:01	1
00:00:00:00:00:02	2

match field		forward port
dest	src	
00:00:00:00:00:01	00:00:00:00:00:02	1
00:00:00:00:00:02	00:00:00:00:00:01	2

. now src+dest pair all sets, register go and back entries

After that, all packets h1 <-> h2 will match and will not make Packet-In / Flooding anymore