Implementation of Layer 2 Clos Fat-tree with Programmable Switches

IEEE 802.1-24-0013-00-ICne

Aditya Srivastava <Aditya.Srivastava@colorado.edu>
EthAirNet Associates
University of Colorado at Boulder

Roger Marks <roger@ethair.net>
EthAirNet Associates

14 March 2024

Related Contributions

- Data Center Collective Multicast using BARC-assigned Address Blocks
  https://www.ieee802.org/1/files/public/docs2024/cq-Marks-collective-multicast-0324-v00.pdf

- Collective Communication in a Layer 2 Clos Fat-tree
  IEEE 802.1-24-0012
  https://mentor.ieee.org/802.1/documents?is_group=ICne&is_year=2024&is_dcn=0012

- Observations of a Layer 2 Clos Fat-tree
  IEEE 802.1-24-0014
  https://mentor.ieee.org/802.1/documents?is_group=ICne&is_year=2024&is_dcn=0014

Introduction

- Methods of Collective Communication in a Clos Fat-tree computing networks were described in “Collective Communication in a Layer 2 Clos Fat Tree”

- This contribution describes the implementation of such methods in an emulation using programmable switches.

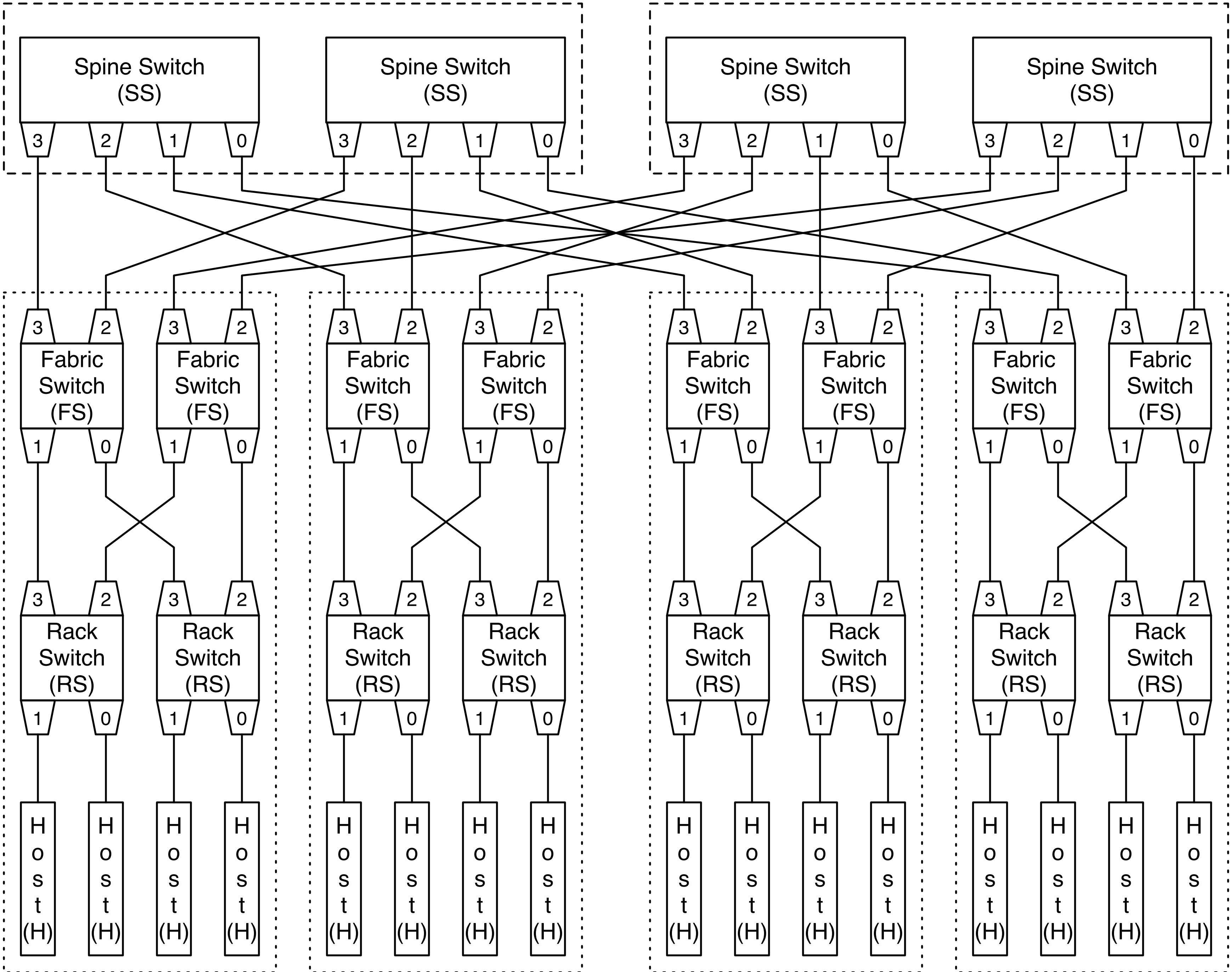
Network Topology Implementation

- Mininet based network virtualization emulates:
  - Hosts and Switches
  - Interfaces
  - Ports and Links

- k-ary Clos Fat-Tree Topology
  - k ports per switch, numbered 0 through k–1

- for each switch
  - $k/2$ down-facing ports per switch (numbered 0 through $k/2$–1)
  - $k/2$ up-facing ports per switch (numbered $k/2$ through k–1)

Network Topology Example (*k*=4)



Clos Fat-tree BARC Address Blocks (ABs)

		spine switch (SS)	fabric switch (FS)	rack switch (RS)	host (H)
MSB	AB[0]	0xBE (unicast) 0xBF (multicast)	0xFE (unicast) 0xFF (multicast)	0xEE (unicast) 0xEF (multicast)	0xAE (unicast) 0xAF (multicast)
	AB[1]	Spine Switch ID (sw)	Pod ID (p)	Pod ID (p)	Pod ID (p)
	AB[2]	Spine ID (s)	Spine ID (s)	Rack ID (r)	Rack ID (r)
	AB[3]	*	*	*	Host ID (h)
	AB[4]	*	*	*	*
LSB	AB[5]	*	*	*	*

After configuration:

▫Any unicast frame can be forwarded directly toward its destination address in any of these address blocks, by any switch, without a forwarding database.

▫The egress port is read directly from the destination address.

Switch Implementation

- Switch software written in P4 <https://p4.org>
    - specifies forwarding logic in network devices
    - supports control plane interactions

- bmv2 software switch emulation <http://bmv2.org>
    - executes compiled P4 instructions

- Install identical P4 code on each switch
    - location and address learning
    - Unicast Forwarding
    - Collective Registration (CoRe)
    - Collective Multicast Forwarding

- All switches initialize with identical configurations
    - switches learn identity/location & address blocks via BARC operation
      -based on Block Address Registration and Claiming (P802.1CQ)

```
// packet flow
V1Switch(
  Parser(),
  VerifyChecksum(),
  Ingress(),
  Egress(),
  ComputeChecksum(),
  Deparser()
) main;

// block address
register<bit<8>>(3) self;

// sample action
action barc_p_fs_low() {
  // calculate egress port
  egressPort = ingressPort + TREE_K/2;

  // modify frame
  hdr.proto.barc.S = BARC_P;
  hdr.proto.barc.BI.f0 = SPN_ID;
  hdr.proto.barc.BI.f1 = ingressPort;
  hdr.proto.barc.BI.f2 = self_2;
  hdr.proto.barc.BI.f3 = 0;
  hdr.proto.barc.BI.f4 = 0;
  hdr.proto.barc.BI.f5 = 0;

  // set egress port
  std_metadata.egress_spec = (bit <9>) egressPort;
}
```

```
// sample control block
control ingress() {
  ...
  else if (hdr.ethernet.dstAddr == NCB_DA) {
    if (hdr.ethernet.etherType == TYPE_BARC) {
      ...
      else if (hdr.proto.barc.S == BARC_P) {

        // load switch address
        self.read(self_0, (bit<32>) 0);
        self.read(self_1, (bit<32>) 1);
        self.read(self_2, (bit<32>) 2);

        // to fabric switch
        if (hdr.proto.barc.BI.f0 == FAB_ID) {
          if (ingressDir) {
            // low port of ingress
            barc_p_fs_low();
          }
          ...

// sample table
table mc_table {
      key = {
            hdr.ethernet.dstAddr : exact;
            ingressPort: exact;
      }
      actions = {
            multicast_to_group;
      }
}
```

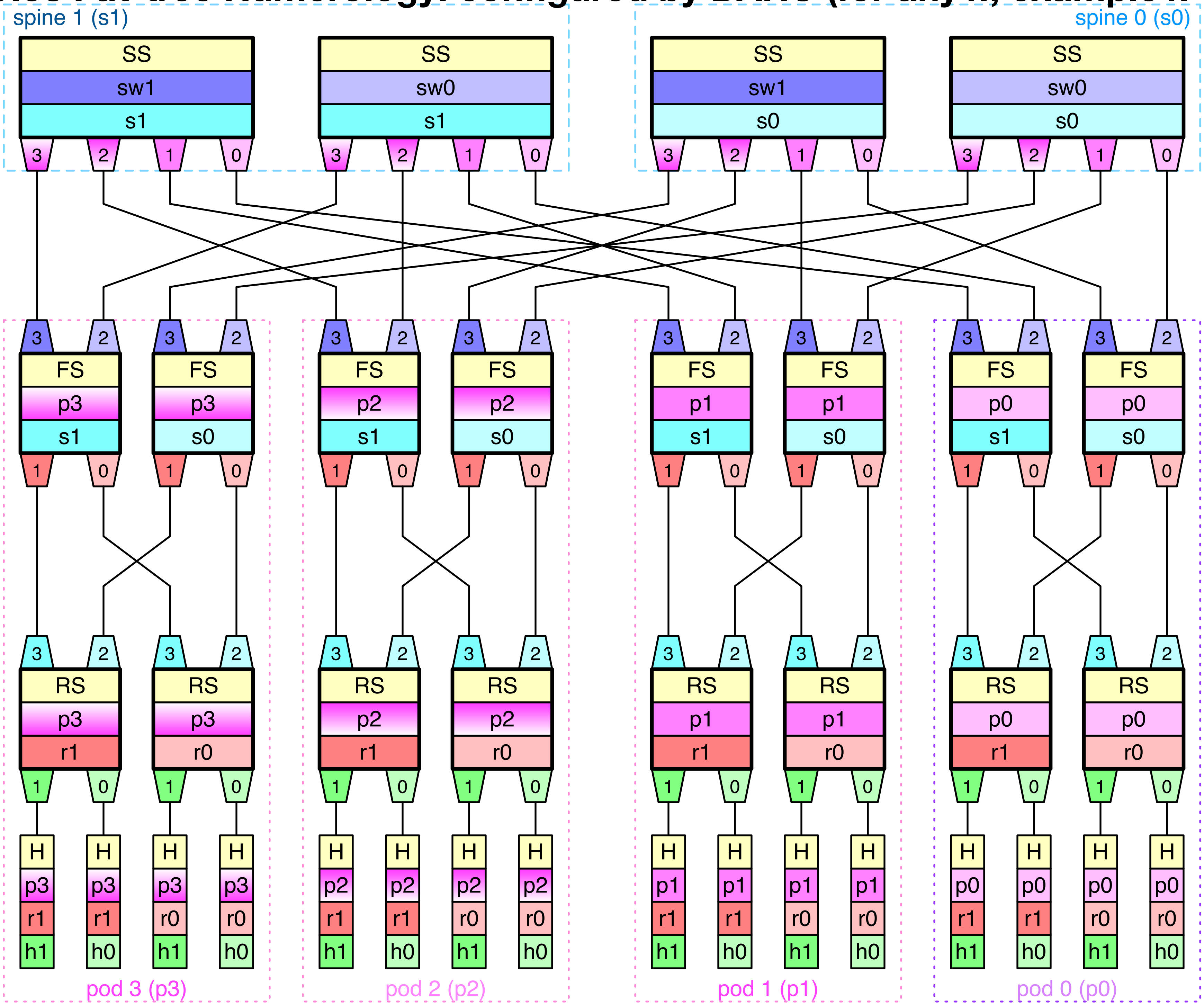
Host Implementation

- Hosts run Python3 on Linux

- Scapy packet manipulation library <https://scapy.net>
  - Define packet headers: Ethernet, BARC, Unicast, Multicast, CoRe
  - Craft packets by defining header fields
  - Send and listen for transmissions
  - Parse sent and received packets

- At launch, run BARC inquiry at each host
  - Inquiry initiates discovery of numerology by each switch
  - host awaits BARC proposal providing host address block assignment

Clos Fat-tree Numerology: configured by BARC (for any *k*; example *k*=4)



Switch roles:

Spine Switch (SS)

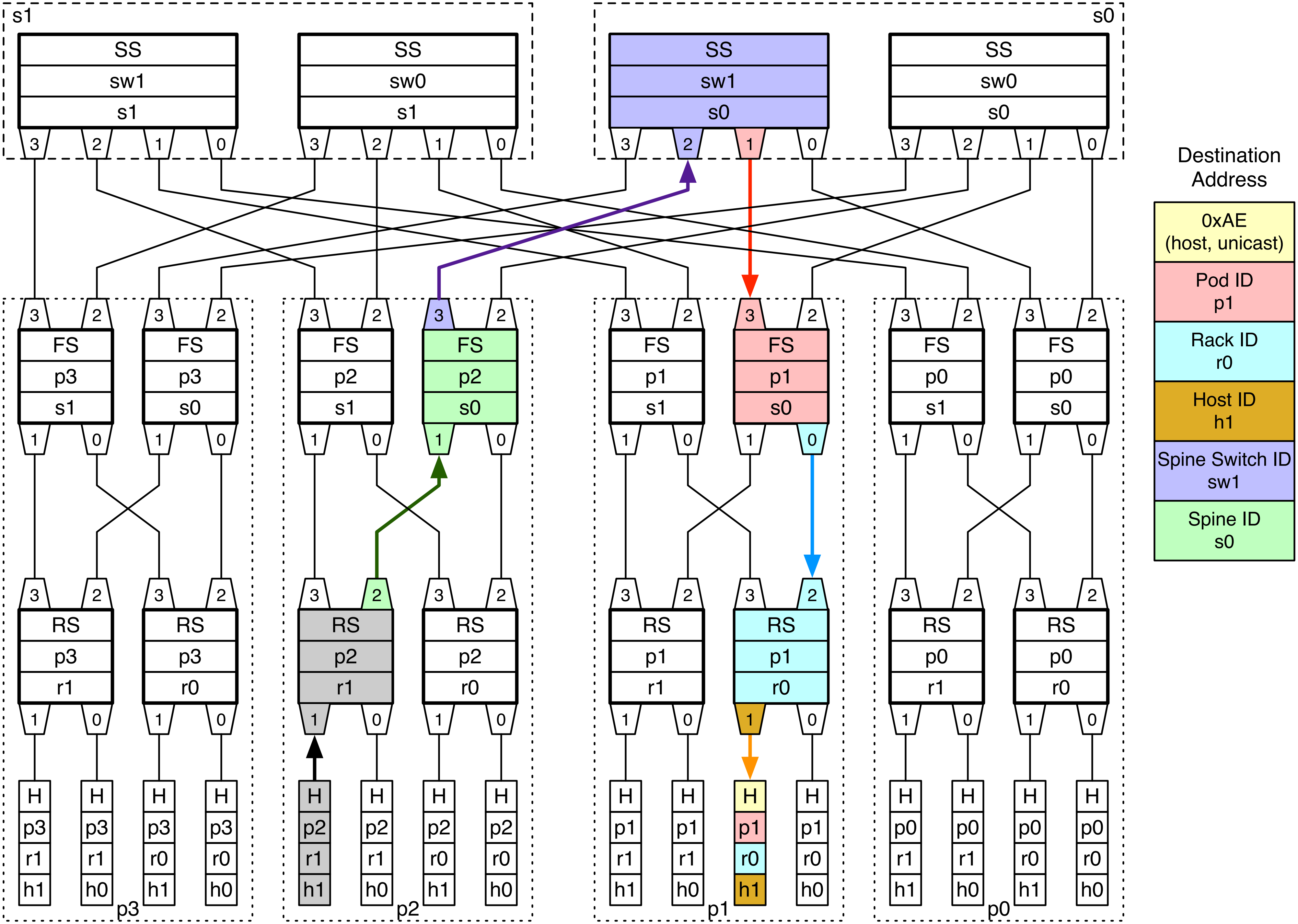
Fabric Switch (RS)

Rack Switch (RS)

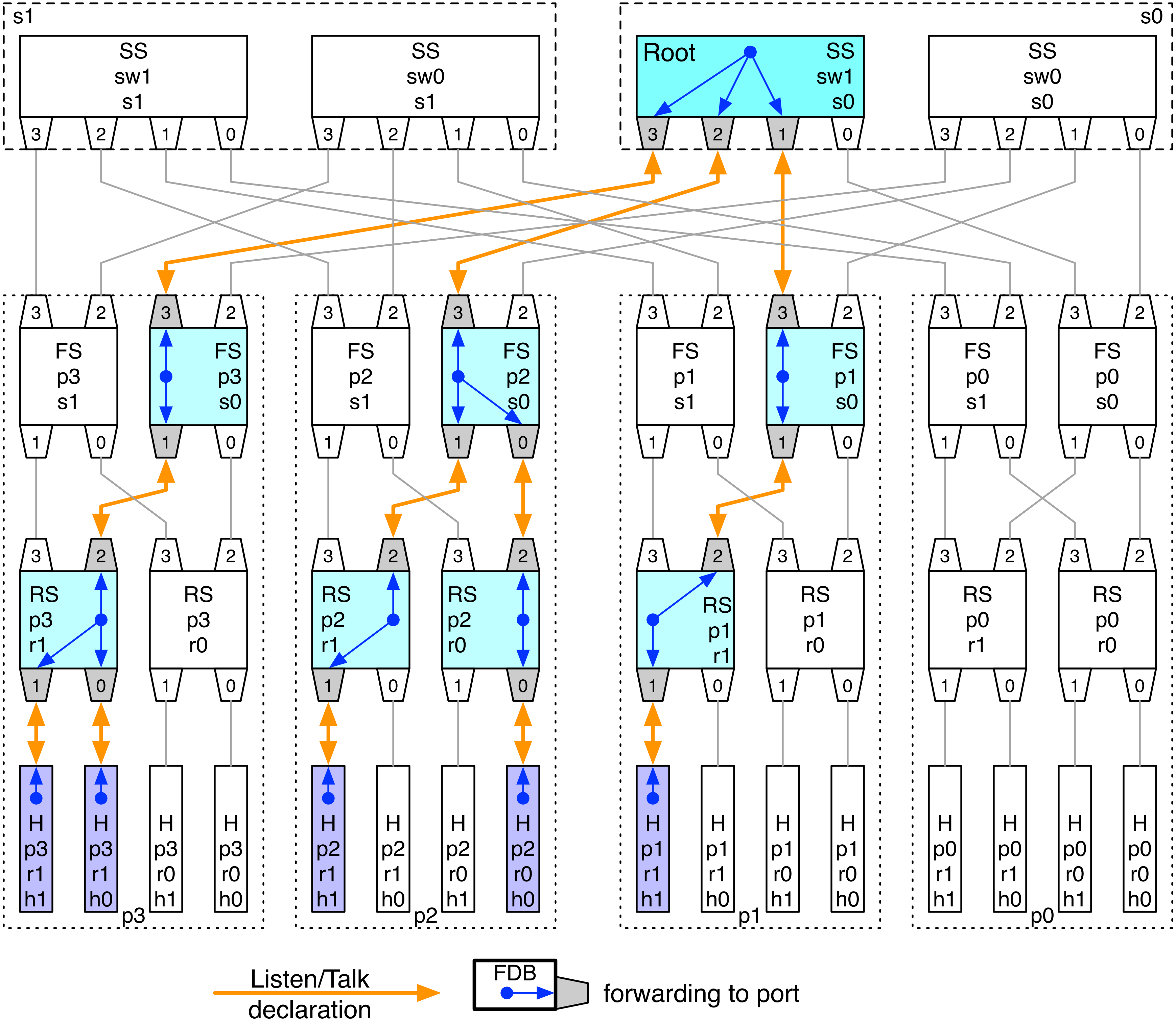
Each switch is uniquely identified by three fields, including one that identifies the role (spine, fabric, or rack).

Each host is uniquely identified by four fields (including a Host role).

Stateless Unicast forwarding (fully source-specified)



Collective Multicast in Clos Fat-tree



Control Plane Implementation

- Implemented in Python3
  - Apache Thrift RPC based communication <https://thrift.apache.org>
  - Configure tables, multicast groups, mirroring sessions, etc.

- Scapy for packet manipulation and packet parsing

- Controller in each switch
  - controls local control plane
    - * Control plane is fully distributed
    - * NO CENTRALIZED CONTROLLER
  - Constantly listens for forwarded packets
  - Collective Registration
  - Add entries to the multicast forwarding table

- Identical control plane software for all switches

```
from scapy import *

// headers
class BARC(Packet):
  name = "BARCPacket"
  fields_desc = [
    XByteField("subtype", 0x00),
    BitField("h", 0b0, 1),
    BitField("version", 0b000, 3),
    BitEnumField("S", BARC_I, 4),
    FieldListField("BI", [], XByteField("", 0x00)),
    BitField("BA", 0x000000000000, 48),
    BitField("Info", 0x000000000000, 48),
  ]

// sending packets
def send_bi(intf=get_intf()):
  # define fields
  ether = CEther(dst=xtos(NCB_DA),
                 src=get_src_addr(),
                 type=TYPE_BARC)

  barc = BARC(S=BARC_I, BI=[HST_ID, 0x00, 0x00,
                            0x00, 0x00, 0x00])
  # compile frame
  frame = ether / barc

  # send frame
  sendp(frame, iface=intf)
```

```
// receiving packets
AsyncSniffer(prn=process_pkt, store=False,
iface=intf).start()

// controller ops
from p4utils.utils.sswitch_thrift_API import
SimpleSwitchThriftAPI

# initialize the controller
controller = SimpleSwitchThriftAPI(thrift_port)

# listening for packets
sniff(iface=intf, prn=process)

# adding table entries
controller.table_add(
      "Ingress.mc_table",
      "Ingress.multicast_to_group",
            key,
            [i_mask],
)

# adding multicast groups
controller.mc_mgrp_create(grp)
node_handle = controller.mc_node_create(0, ports)
controller.mc_node_associate(grp, node_handle)

# adding mirror sessions
controller.mirroring_add(ctr_session, port)
```

Conclusions

- Stateless Layer 2 unicast can be implemented with BARC for a Clos Fat-tree using a programmable switch.

- Collective multicast can be constructed for a Clos Fat-tree.
    - within Layer 2
    - with a simple and efficient configuration protocol

- Communication patterns important for Computing Networks can be studied using programmable switches at Layer 2.
    - results can be useful in future Nendica studies on Computing Networks

- Observations are discussed in a followup Nendica contribution of 2024-03-14.