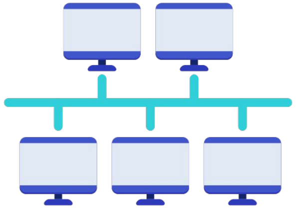
SDN - Network Slice Setup Optimization

Project Report - Networking II
Softwarized and Virtualized Mobile Networks
(prof. Fabrizio Granelli)

Samuele Pozzani

Project Goals



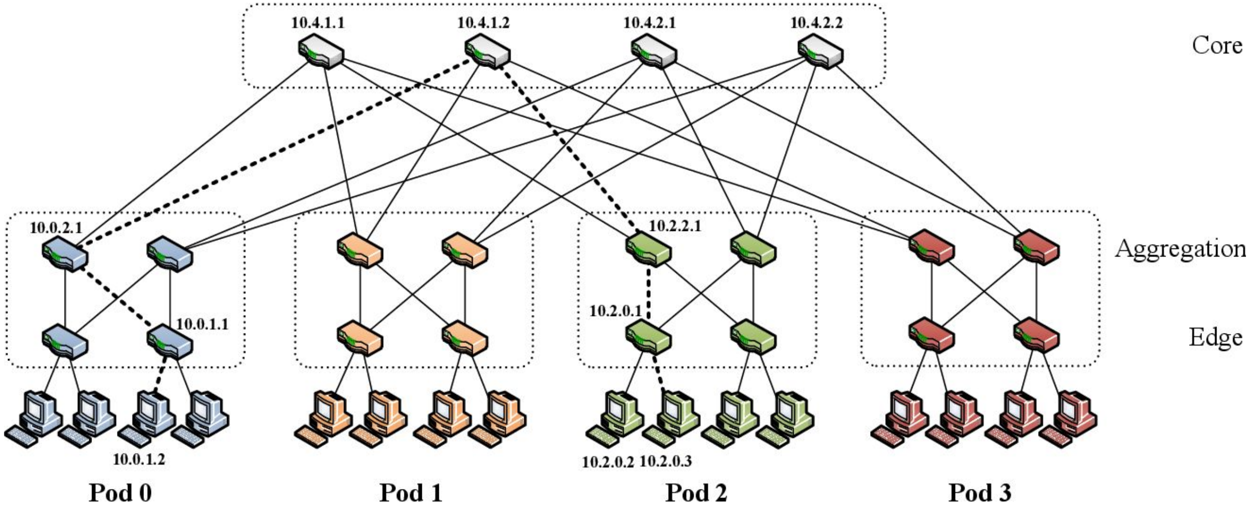
Simulate a network topology using Mininet

Develop a RYU-based SDN controller

Automatically optimize resources and provide QoS

Fat-Tree DC Network Topology

$K = 4$



Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat, *"A scalable, commodity data center network architecture"*, SIGCOMM 2008.

Samuele Pozzani

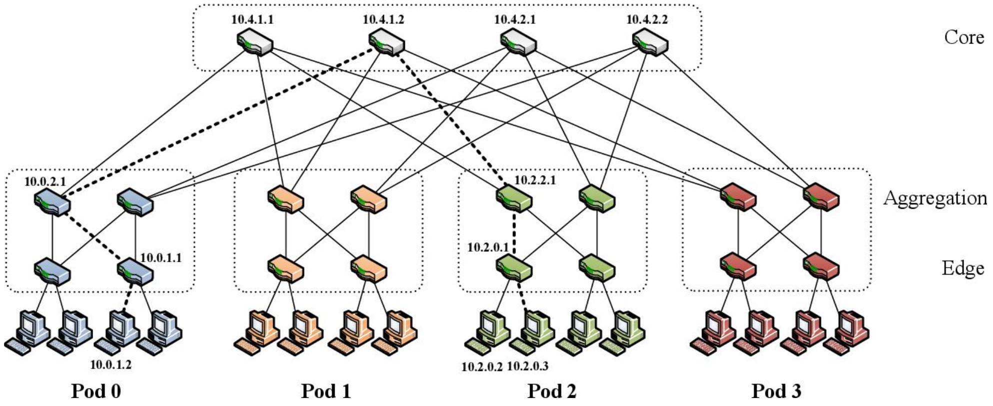
Two-Levels Routing - Proactive Flow Insertion

foreach *pod* x *in* $[0, k-1]$ **do**
  foreach *switch* z *in* $[(k/2), k-1]$ **do**
    foreach *subnet* i *in* $[0, (k/2)-1]$ **do**
      addPrefix(10.x.z.1, 10.x.i.0/24, i);
    end
    addPrefix(10.x.z.1, 0.0.0.0/0, 0);
    foreach *host ID* i *in* $[2, (k/2)+1]$ **do**
      addSuffix(10.x.z.1, 0.0.0.i/8,
      $(i-2+z)mod(k/2)+(k/2)$);
    end
  end
end

Algorithm 1: Generating aggregation switch routing tables. Assume Function signatures *addPrefix(switch, prefix, port)*, *addSuffix(switch, suffix, port)* and $addSuffix$ adds a second-level suffix to the last-added first-level prefix.

foreach j *in* $[1, (k/2)]$ **do**
  foreach i *in* $[1, (k/2)]$ **do**
    foreach *destination pod* x *in* $[0, (k/2)-1]$ **do**
      addPrefix(10.k.j.i,10.x.0.0/16, x);
    end
  end
end

Algorithm 2: Generating core switch routing tables.



Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat, *"A scalable, commodity data center network architecture"*, SIGCOMM 2008.

Samuele Pozzani

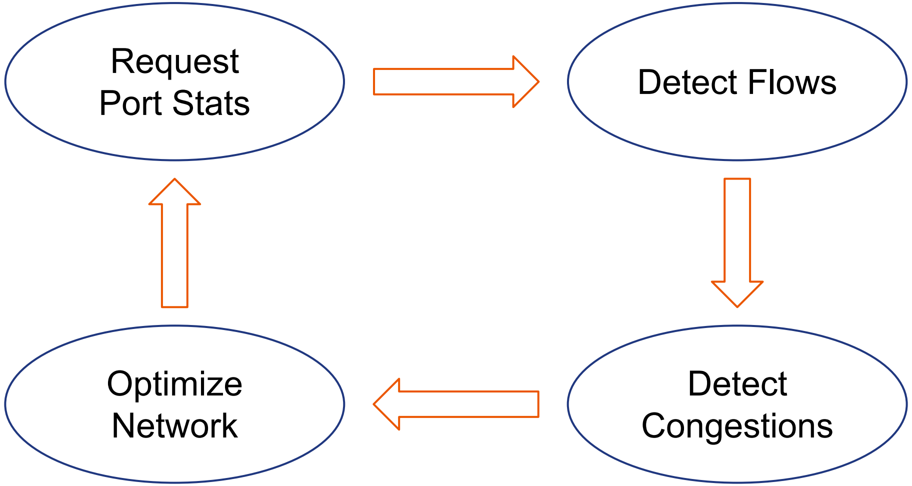
Network Slicing - Reactive Flow Insertion

foreach *pod* x *in* $[0, k-1]$ **do**
  foreach *switch* z *in* $[(k/2), k-1]$ **do**
    foreach *subnet* i *in* $[0, (k/2)-1]$ **do**
      addPrefix(10.x.z.1, 10.x.i.0/24, i);
    end
    addPrefix(10.x.z.1, 0.0.0.0/0, 0);
    foreach *host ID* i *in* $[2, (k/2)+1]$ **do**
      addSuffix(10.x.z.1, 0.0.0.i/8, $(i-2+z) mod (k/2) + (k/2)$);
    end
  end
end

Algorithm 1: Generating aggregation switch routing tables. Assume Function signatures *addPrefix(switch, prefix, port)*, *addSuffix(switch, suffix, port)* and *addSuffix* adds a second-level suffix to the last-added first-level prefix.

```
slices = {
    0: ['10.0.0.2', '10.3.0.2', '10.2.0.2',],
    1: ['10.0.1.2', '10.2.1.3',],
    2: ['10.0.1.3', '10.2.0.3', '10.2.1.2',],
}

# Check whether src host is in the same slice as dst host
if any( src in slice and dst in slice
    for slice in slices.values() ):

    # Compute target port number
    port = (dst.hostid - 2 + switch.number) % (K / 2) + (K / 2)

    # Add FlowTable entry to the switch identified by datapath
    add_two_level_flow (
        switch = switch.datapath ,
        ip = dst,
        mask = 0xFFFFFFFF,
        port = port + 1,
        timeout = 30
    )
```

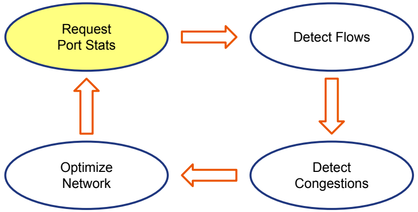
Flow Scheduler - Loop




```
from threading import Thread

class SDNController(app_manager.RyuApp):
    def __init__(self):
        self.scheduler = FlowScheduler()
        self.scheduler.start()

class FlowScheduler(Thread):
    def run(self):
        self.__main_loop()
```

Samuele Pozzani

Flow Scheduler - Port Stats Request




```c
/* Body of reply to OFPMP_PORT_STATS request. If a counter is unsupported,
 * set the field to all ones. */
struct ofp_port_stats {
    uint16_t length;          /* Length of this entry. */
    uint8_t pad[2];           /* Align to 64 bits. */
    uint32_t port_no;
    uint32_t duration_sec;    /* Time port has been alive in seconds. */
    uint32_t duration_nsec;   /* Time port has been alive in nanoseconds beyond
                                 duration_sec. */
    uint64_t rx_packets;      /* Number of received packets. */
    uint64_t tx_packets;      /* Number of transmitted packets. */
    uint64_t rx_bytes;        /* Number of received bytes. */
    uint64_t tx_bytes;        /* Number of transmitted bytes. */

    uint64_t rx_dropped;      /* Number of packets dropped by RX. */
    uint64_t tx_dropped;      /* Number of packets dropped by TX. */
    uint64_t rx_errors;       /* Number of receive errors.  This is a super-set
                                 of more specific receive errors and should be
                                 greater than or equal to the sum of all
                                 rx_*_err values in properties. */
    uint64_t tx_errors;       /* Number of transmit errors.  This is a super-set
                                 of more specific transmit errors and should be
                                 greater than or equal to the sum of all
                                 tx_*_err values (none currently defined.) */

    /* Port description property list - 0 or more properties */
    struct ofp_port_stats_prop_header properties[0];
};
OFP_ASSERT(sizeof(struct ofp_port_stats) == 80);
```

```python
class Switch():
    def __init__(self):
        self.port_stats = {
            i : PortStats()
            for i in range(1, FAT_TREE_K + 1)
        }

class PortStats():
    def update_stats(self, tx_bytes, rx_bytes):
        # tx/rx bytes since latest update
        self.dtx_bytes = tx_bytes - self.tx_bytes
        self.drx_bytes = rx_bytes - self.rx_bytes
        # Total amount of tx/rx bytes
        self.tx_bytes = tx_bytes
        self.rx_bytes = rx_bytes
```

Samuele Pozzani

Flow Scheduler - Detect Flows

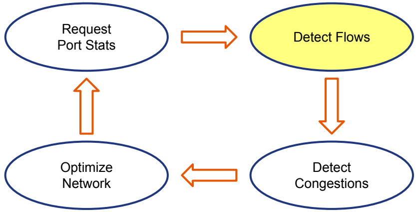
```
======= Core Switch Port Statistics
=======
c11:
      Port 0: [ TX: 8854     RX: 8586 ]
      Port 1: [ TX: 8586     RX: 4462 ]
      Port 2: [ TX: 70       RX: 4462 ]
      Port 3: [ TX: 70       RX: 70   ]
c22:
      Port 0: [ TX: 70       RX: 8586 ]
      Port 1: [ TX: 70       RX: 70   ]
      Port 2: [ TX: 8586     RX: 70   ]
      Port 3: [ TX: 70       RX: 70   ]
==========================================
=
class Flow():
    def __init__(self, switch_id, in_pod, out_pod):
        self.switch = Switch(switch_id)
        self.in_pod = in_pod
        self.out_pod = out_pod
```

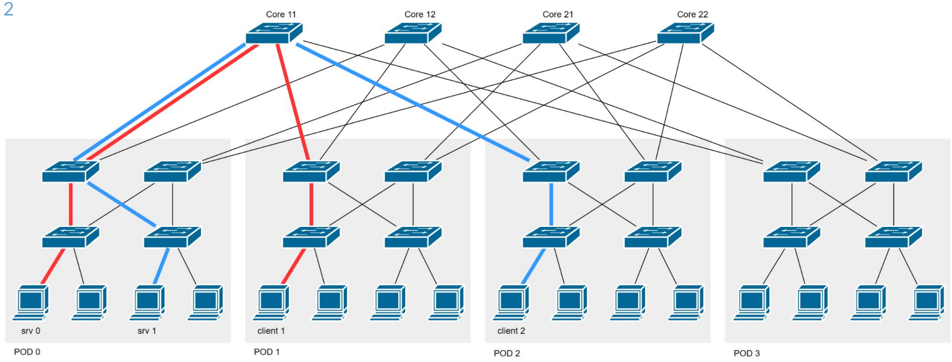
Flow from pod 0 to pod 1

Flow from pod 1 to pod 0

Flow from pod 2 to pod 0

Flow from pod 0 to pod 2



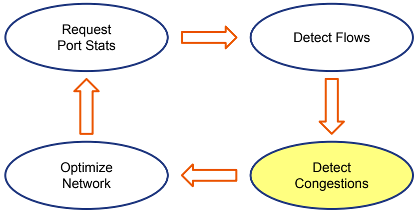


For more advanced techniques, refer to:

Mohammad Al-Fares, “*Hedera: Dynamic Flow Scheduling for Data Center Networks*”, NSDI 2010

Samuele Pozzani

Flow Scheduler - Detect Congestions




```
======= Core Switch Port Statistics
=======
c11:
      Port 0: [ TX: 8854      RX: 8586 ]
      Port 1: [ TX: 8586      RX: 4462 ]
      Port 2: [ TX: 70        RX: 4462 ]
      Port 3: [ TX: 70        RX: 70   ]
c22:
      Port 0: [ TX: 70        RX: 8586 ]
      Port 1: [ TX: 70        RX: 70   ]
      Port 2: [ TX: 8586      RX: 70   ]
      Port 3: [ TX: 70        RX: 70   ]
=========================================
```

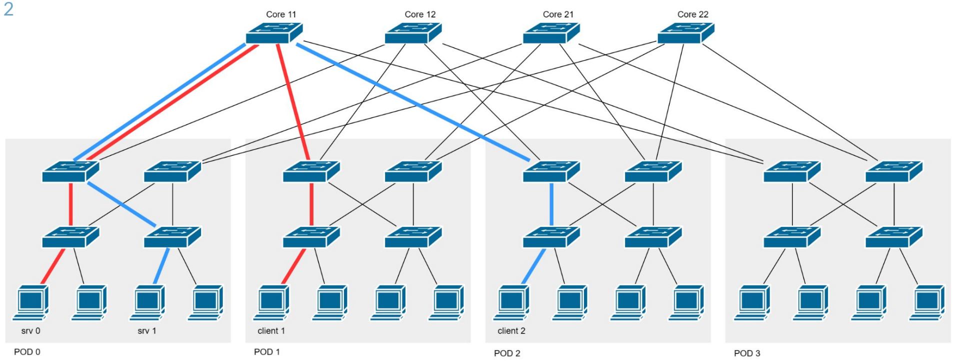
Flow from pod 0 to pod 1

Flow from pod 1 to pod 0

Flow from pod 2 to pod 0

Flow from pod 0 to pod 2

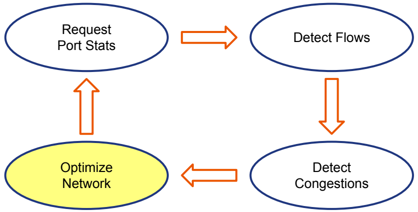
Discovered congested downlink from core switch c11 to pod 0




```python
class DownLink():

    def __init__(self, switch, dst_pod):
        self.switch: Switch = switch
        self.dst_pod: int = dst_pod
```

Samuele Pozzani

Flow Scheduler - Optimize Network



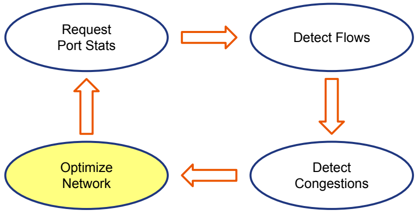
Foreach **congested downlink**:

1. **Find a service** inside the pod connected to the downlink
2. Search for a new **non-conflicting path**
3. If a path was found:
    a. **Re-route traffic** through the new path

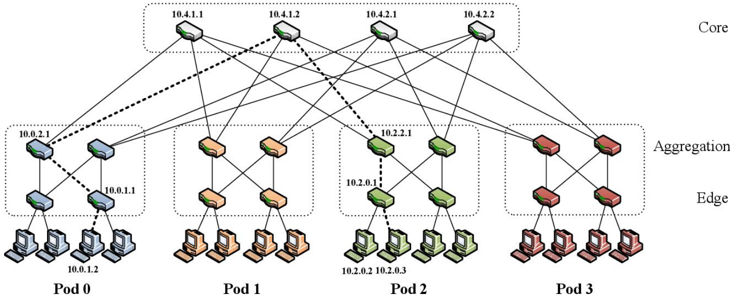
    Otherwise:

    b. **Migrate the service** to a new pod which is the destination of an available path
    c. **Re-route traffic** through the new path

Optimize Network - Create Path

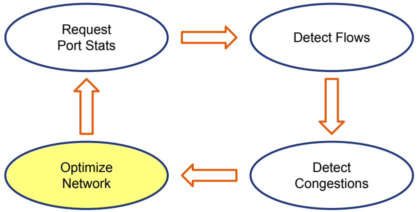



```python
def create_path(dst_service, via_switch):
    for switch in switches:
        if switch.is_core or switch.pod == dst_service.pod:
        # Do not update core switches and
        # switches in the same pod of the dst host
            continue

        if switch.is_edge:
            port = (K / 2) + via_switch.j      # Edge
        if not switch.is_edge:
            port = (K / 2) + via_switch.i      # Aggregate

        add_two_level_flow(
            datapath = switch.datapath,
            ip = dst_service.ip,
            mask = 0xFFFFFFFF,
            port = port,
            timeout = 30,
            priority = int(time()) & 0xFFFF   # High priority
        )
```

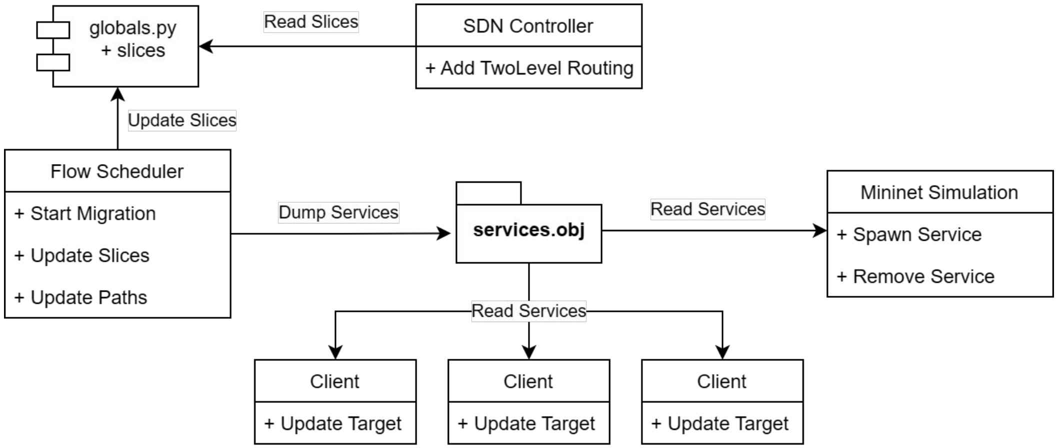


Samuele Pozzani

Optimize Network - Migrate Service



Enable **communication** between the simulation loop and the Flow Scheduler by **dumping** the services list on the **FileSystem**




```
slices = {
    0: ['10.0.0.2',],
    1: ['10.0.1.2', '10.2.0.2',],
    2: ['10.0.1.3', '10.2.0.3',],
}

services = {
    'apache_srv'    : '10.0.0.2',
    'mysql_srv'     : '10.0.1.2',
    'dotnet_be_srv' : '10.2.1.3',
}
```

Samuele Pozzani

Future Work

Improve flow scheduling

Hedera: Dynamic Flow Scheduling for Data Center Networks

Mohammad Al-Fares* Sivasankar Radhakrishnan*
Barath Raghavan† Nelson Huang* Amin Vahdat*
*{malfares, sivasankar, nhuang, vahdat}@cs.ucsd.edu †barath@cs.williams.edu

*Department of Computer Science and Engineering, University of California, San Diego
†Department of Computer Science, Williams College

Abstract

Today's data centers offer tremendous aggregate bandwidth to clusters of tens of thousands of machines. However, because of limited port densities in even the highest-end switches, data center topologies typically consist of multi-rooted trees with many equal-cost paths between any given pair of hosts. Existing IP multipathing protocols usually rely on per-flow static hashing and can cause substantial bandwidth losses due to long-term collisions.

In this paper, we present Hedera, a scalable, dynamic flow scheduling system that adaptively schedules a multi-stage switching fabric to efficiently utilize aggregate network resources. We describe our implementation using commodity switches and unmodified hosts, and show that for a simulated 8,192 host data center, Hedera delivers bisection bandwidth that is 96% of optimal and up to 113% better than static load-balancing methods.

their software on commodity operating systems; therefore, the network must deliver high bandwidth without requiring software or protocol changes. Third, virtualization technology—commonly used by cloud-based hosting providers to efficiently multiplex customers across physical machines—makes it difficult for customers to have guarantees that virtualized instances of applications run on the same physical rack. Without this physical locality, applications face inter-rack network bottlenecks in traditional data center topologies [2].

Applications alone are not to blame. The routing and forwarding protocols used in data centers were designed for very specific deployment settings. Traditionally, in ordinary enterprise/intranet environments, communication patterns are relatively predictable with a modest number of popular communication targets. There are typically only a handful of paths between hosts and secondary paths are used primarily for fault tolerance. In contrast, recent data center designs *rely* on the path mul-

Stateful services

Fault tolerance

Internet Engineering Task Force (IETF) D. Katz
Request for Comments: 5880 D. Ward
Category: Standards Track Juniper Networks
ISSN: 2070-1721 June 2010

Bidirectional Forwarding Detection (BFD)

Abstract

This document describes a protocol intended to detect faults in the bidirectional path between two forwarding engines, including interfaces, data link(s), and to the extent possible the forwarding engines themselves, with potentially very low latency. It operates independently of media, data protocols, and routing protocols.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc5880.

Network Slice Setup Optimization

Thank You

images by flaticon.com

Samuele Pozzani