What is SDN?

Not quite “software-defined”
  • Network control planes were always software defined

Not quite “centralized control”

*Separation* of control and data planes
  • Enables centralization but not centralization is not prerequisite
Why separate control and data plane?

Arbitrary control over how packets are forwarded

- Complex requirements can be hard to specify as distributed, local rules
  - Suppose you want all paths in the network to be of length 10

Efficiency
Traffic engineering case study
Traffic engineering journey

1. SPF with static cost
2. SPF with load-based cost – BAD!
3. CSPF (used in MPLS)
4. SDN
Limitations of static-cost SPF
Limitations of SPF with load-based costs

Increase cost of C-G

Decrease cost of C-G
CSPF

Each ingress router measures the traffic that it is sending to other routers.

Ingress router finds paths that can accommodate its traffic:
- Shortest path that meets the capacity constraint (CSPF)

Ingress router asks other routers if they can use the path:
- Necessary because all ingress routers are operating independently
Possible solution with CSPF
But CSPF has issues too

Local, greedy allocation
(Distributed CSPF)

Globally optimal allocation
(Centralized)
SWAN: SDN based TE
Inter-DC WAN: A critical, expensive resource

But it was being used highly inefficiently
Inefficiency of the inter-DC WAN

Normalized throughput on a busy link between data centers:

Average utilization = 46%

Normalized traffic on a busy link between data centers
Root cause: Service-level allocations

Operators configure individual services with maximum sending rate

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>.....</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEA → NYC (80)</td>
<td>10</td>
<td>15</td>
<td>5</td>
<td>.....</td>
</tr>
<tr>
<td>SEA → CHI (100)</td>
<td>20</td>
<td>20</td>
<td>10</td>
<td>.....</td>
</tr>
<tr>
<td>.....</td>
<td></td>
<td></td>
<td></td>
<td>.....</td>
</tr>
</tbody>
</table>

**Inefficient:** The combined maximum is uncommon

**Unreliable:** Load can exceed capacity when failures occur

**Slow to change:** Must change all allocations to add services or network links
Centralized control can increase efficiency

Service 1
- Priority: Bg
- Weight: 1

Service 2
- Priority: Bg
- Weight: 2

Service 3
- Priority: Non-bg
- Weight: 1

Average utilization = 46%

Peak before adapting

Peak after adapting

>50% peak reduction
Service 1
- Priority: Bg
- Weight: 1

SWAN

1. Traffic demand → Controller
2. Service allocations
3. Network configuration → Services

Controller

Services

WAN
Challenge: Congestion during network updates

Link capacity: 10
Flow size: 6.6
Solution: Congestion-free update plans

Link capacity: 10
Flow size: 6.6
Computing congestion-free update plans

Leave scratch capacity $s$ on each link
  - Guarantees a plan with at most $\left\lceil \frac{1}{s} \right\rceil - 1$ steps

Find a plan with minimum number of steps using an LP
  - Search for a feasible plan with 1, 2, ..., max steps

Use scratch capacity for background traffic
  - Bound its experienced congestion
Efficiency improvement with SWAN

Throughput
(relative to optimal)
Why not centralized traffic engineering?

Robustness (recall the first design goal of the internet)
- Controller failure (topic of next class)
- Communication failure

Scalability: Eventually need to distribute in some manner

Reaction time to some types of events
- On the other hand, convergence issues with distributed routing means that it too can be slow for some types of events
Can you implement any packet forwarding behavior with SDN?

No

• Example: Add 10 bytes to the packet at every hop

We haven’t fundamentally changed the data plane behavior

• We just changed where the control planes runs and how data plane is configured
Programmable data planes
Take 1: Active networking

Packet forwarding uses a program
  • The program could be carried by the packet itself

Good idea?

+ Most flexible
- Performance concerns
Programmable data planes
Take 2: PISA

PISA: Protocol independent switch architecture (originally called RMT reconfigurable match tables)

P4: A language to program such data planes
PISA cannot do everything

Not Turing complete

Examples of things not possible
  • Schedule packets
  • Manipulate payloads
  • Manipulate state programmatically

Attempts a balance between high performance and flexibility
Network virtualization

Hardware virtualization abstracts away the hardware from the operating system

Network virtualization abstracts away the physical network from the network stack
  • Can run multiple virtual networks on the same physical network
  • Can pretend to be a different network (e.g., pretend that all hosts are directly connected)

Works by intercepting and manipulating network packets
Network virtualization example

10.10.10.1

Host1

10.10.10.1 \(\rightarrow\) 10.10.10.2

1.1.1.1 \(\rightarrow\) 2.2.2.2

10.10.10.1 \(\rightarrow\) 10.10.10.2

Host2

10.10.10.2

10.10.10.1 \(\rightarrow\) 10.10.10.2

10.10.10.1 \(\rightarrow\) 10.10.10.2

1.1.1.1 \(\rightarrow\) 2.2.2.2
SDN and network virtualization

Not the same thing though often confused

SDN enables network virtualization
  • Would be pretty hard to implement virtual networks if the control plane was co-located with data plane
Key takeaways

SDN: Separation of control and data plane

SDN use cases
  • Easier implementation of policies (4D)
  • Efficiency (e.g., SWAN)
  • Network virtualization (e.g., cloud networks)

Data plane programmability via PISA is an area of active investigation