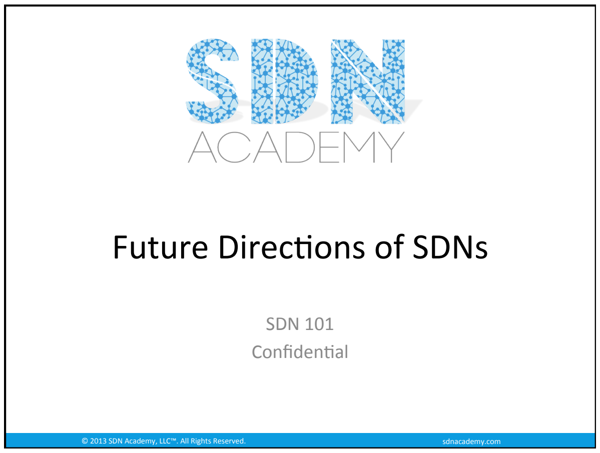
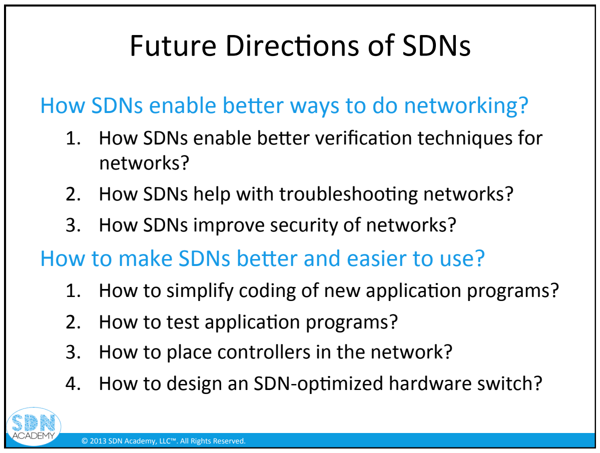


SDN ACADEMY

# Future Directions of SDNs

SDN 101
Confidential



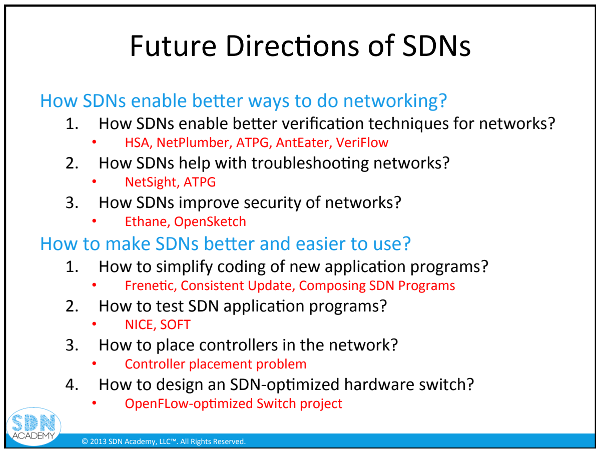
## Future Directions of SDNs

How SDNs enable better ways to do networking?

1. How SDNs enable better verification techniques for networks?
2. How SDNs help with troubleshooting networks?
3. How SDNs improve security of networks?

How to make SDNs better and easier to use?

1. How to simplify coding of new application programs?
2. How to test application programs?
3. How to place controllers in the network?
4. How to design an SDN-optimized hardware switch?



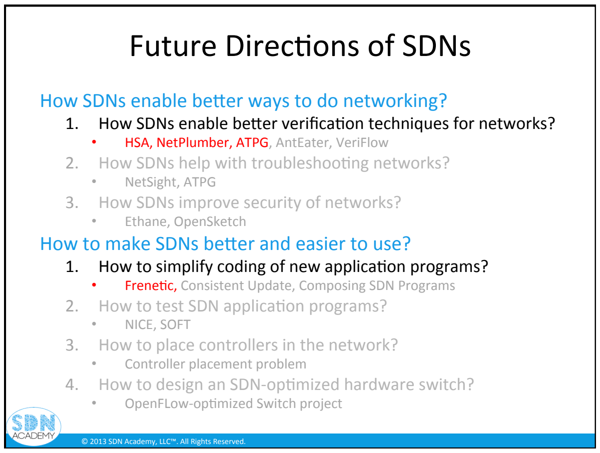
## Future Directions of SDNs

How SDNs enable better ways to do networking?

1. How SDNs enable better verification techniques for networks?
   - HSA, NetPlumber, ATPG, AntEater, VeriFlow
2. How SDNs help with troubleshooting networks?
   - NetSight, ATPG
3. How SDNs improve security of networks?
   - Ethane, OpenSketch

How to make SDNs better and easier to use?

1. How to simplify coding of new application programs?
   - Frenetic, Consistent Update, Composing SDN Programs
2. How to test SDN application programs?
   - NICE, SOFT
3. How to place controllers in the network?
   - Controller placement problem
4. How to design an SDN-optimized hardware switch?
   - OpenFLow-optimized Switch project



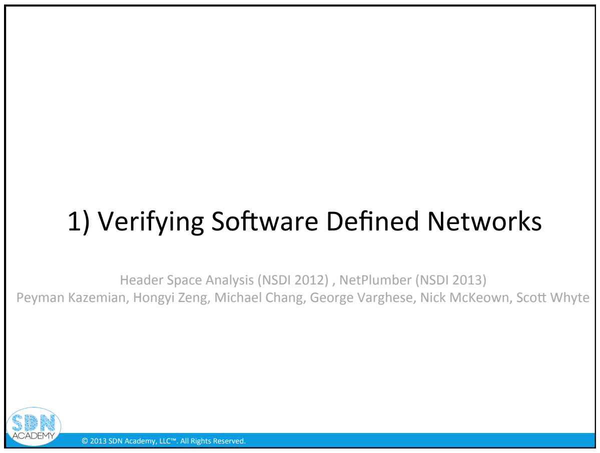
## Future Directions of SDNs

How SDNs enable better ways to do networking?

1. How SDNs enable better verification techniques for networks?
   - HSA, NetPlumber, ATPG, AntEater, VeriFlow
2. How SDNs help with troubleshooting networks?
   - NetSight, ATPG
3. How SDNs improve security of networks?
   - Ethane, OpenSketch

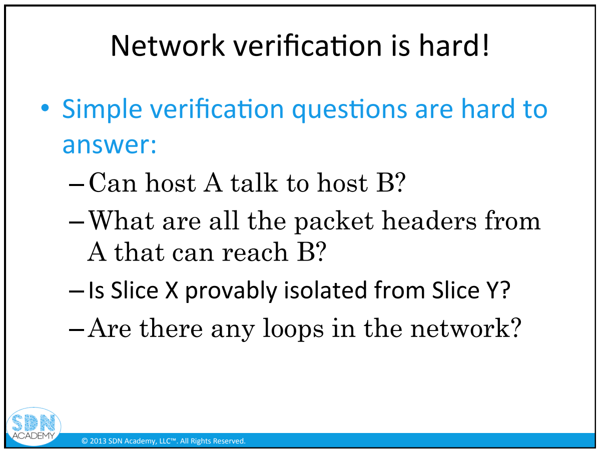
How to make SDNs better and easier to use?

1. How to simplify coding of new application programs?
   - Frenetic, Consistent Update, Composing SDN Programs
2. How to test SDN application programs?
   - NICE, SOFT
3. How to place controllers in the network?
   - Controller placement problem
4. How to design an SDN-optimized hardware switch?
   - OpenFLow-optimized Switch project



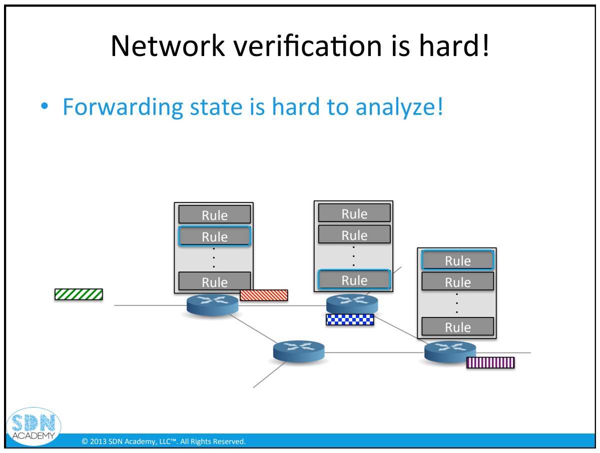
## 1) Verifying Software Defined Networks

Header Space Analysis (NSDI 2012) , NetPlumber (NSDI 2013)
Peyman Kazemian, Hongyi Zeng, Michael Chang, George Varghese, Nick McKeown, Scott Whyte



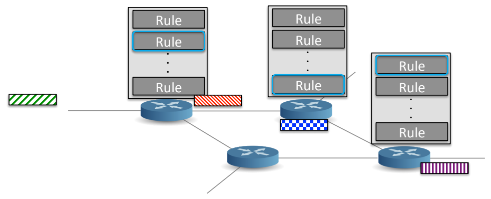
## Network verification is hard!

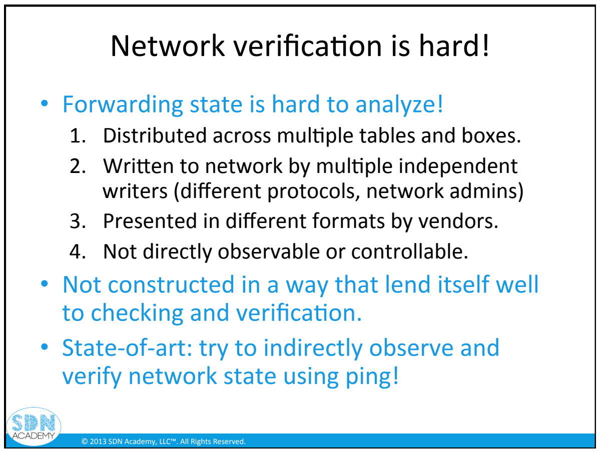
- Simple verification questions are hard to answer:
  - Can host A talk to host B?
  - What are all the packet headers from A that can reach B?
  - Is Slice X provably isolated from Slice Y?
  - Are there any loops in the network?



## Network verification is hard!

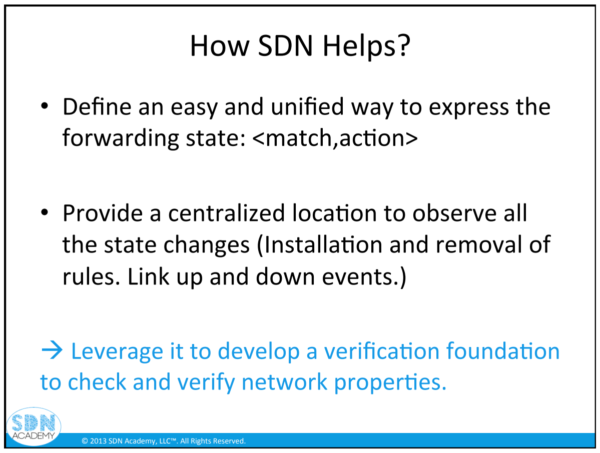
- Forwarding state is hard to analyze!





## Network verification is hard!

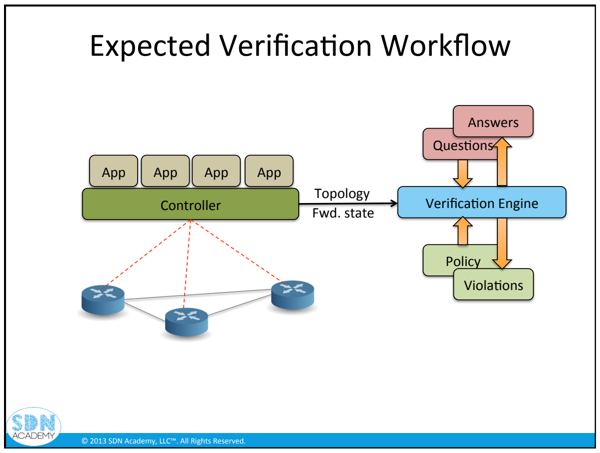
- Forwarding state is hard to analyze!
  1. Distributed across multiple tables and boxes.
  2. Written to network by multiple independent writers (different protocols, network admins)
  3. Presented in different formats by vendors.
  4. Not directly observable or controllable.
- Not constructed in a way that lend itself well to checking and verification.
- State-of-art: try to indirectly observe and verify network state using ping!



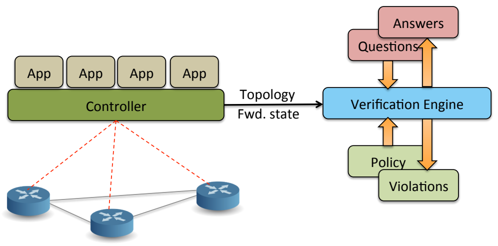
## How SDN Helps?

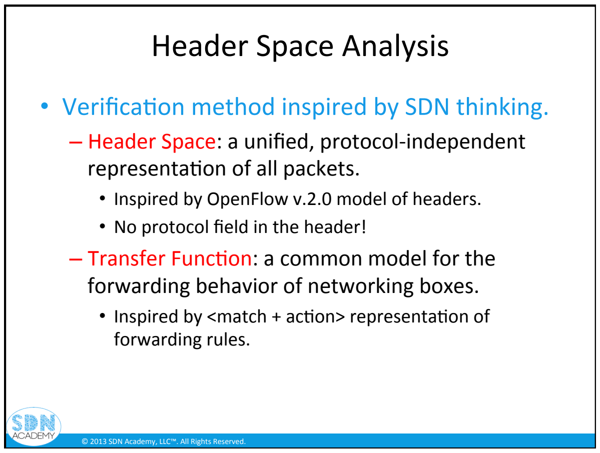
- Define an easy and unified way to express the forwarding state: <match,action>
- Provide a centralized location to observe all the state changes (Installation and removal of rules. Link up and down events.)

→ Leverage it to develop a verification foundation to check and verify network properties.



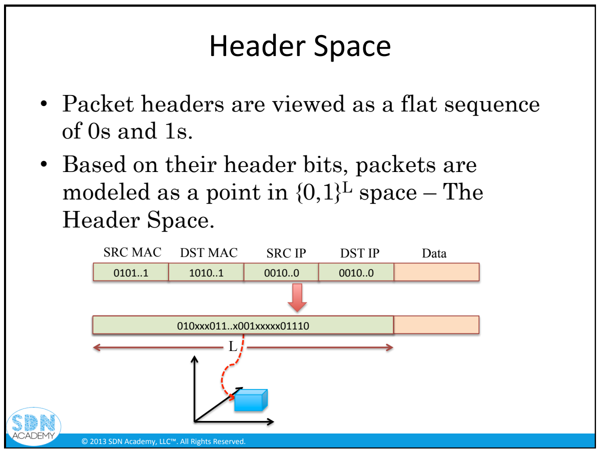
## Expected Verification Workflow





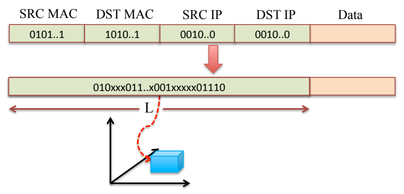
## Header Space Analysis

- Verification method inspired by SDN thinking.
  - Header Space: a unified, protocol-independent representation of all packets.
    - Inspired by OpenFlow v.2.0 model of headers.
    - No protocol field in the header!
  - Transfer Function: a common model for the forwarding behavior of networking boxes.
    - Inspired by <match + action> representation of forwarding rules.



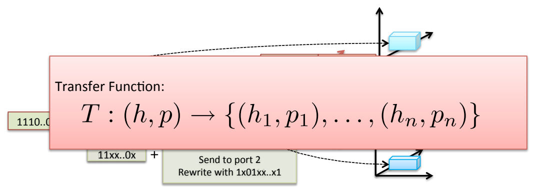
## Header Space

- Packet headers are viewed as a flat sequence of 0s and 1s.
- Based on their header bits, packets are modeled as a point in $\{0,1\}^L$ space – The Header Space.



## Transfer Function

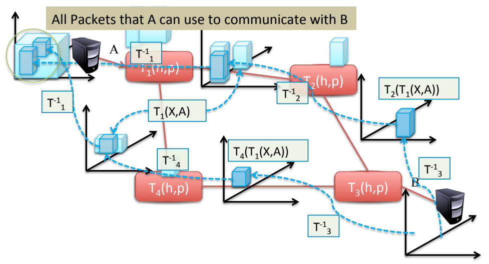
- Models the forwarding behavior of network boxes.



Transfer Function:

$$T : (h, p) \rightarrow \{(h_1, p_1), \ldots, (h_n, p_n)\}$$

## Reachability: Finding Connectivity

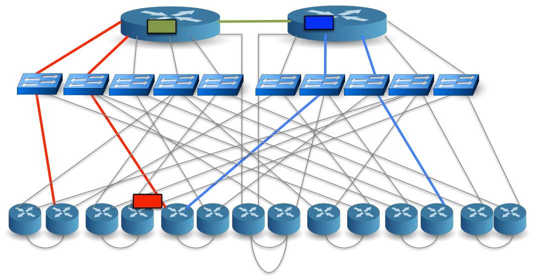


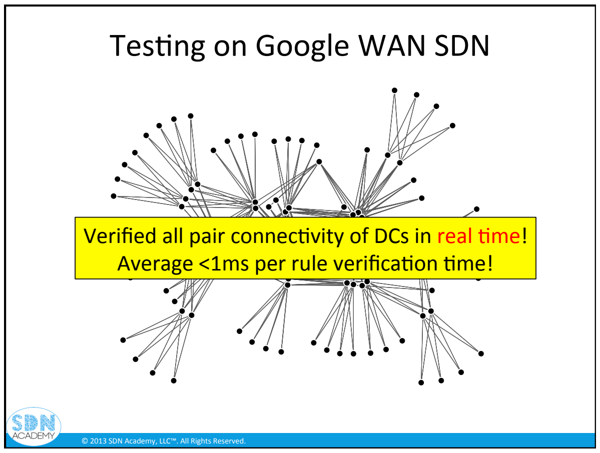
## Checking Predicates on Paths

- Once we do reachability, we can generalize to check *path predicates* such as:
  – Blackhole freedom (A → B and notice unexpected drop)
  – Loop freedom (No packet should be reachable from any port to itself)
  – Communication via middle box. (A→B packets must pass through C)
  – Maximum hop count (length of path from A → B never exceeds L)
  – Isolation of paths (http and https traffic from A→B don't share the same path)

## Testing on Stanford backbone network

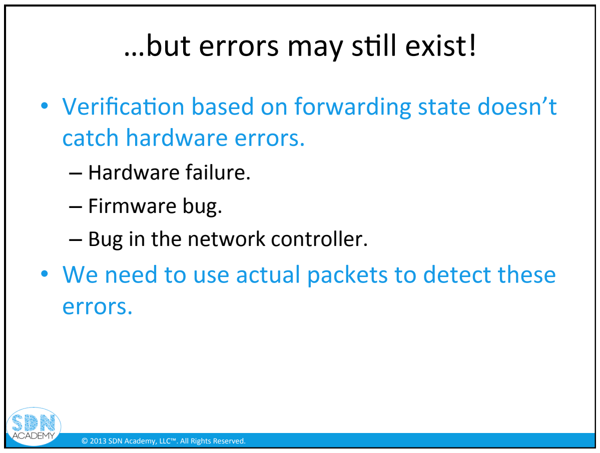
- Loop detection test





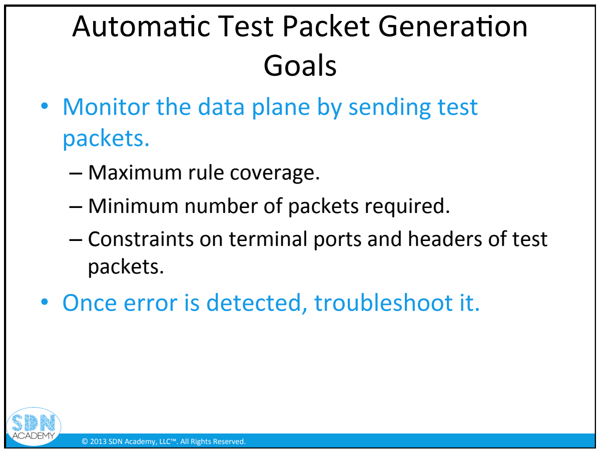
## Testing on Google WAN SDN

Verified all pair connectivity of DCs in real time!
Average <1ms per rule verification time!



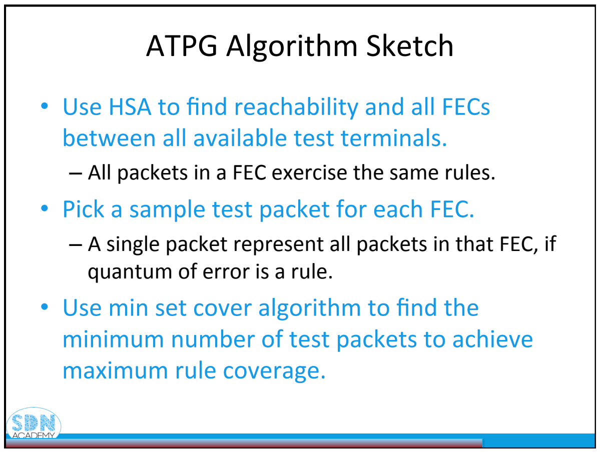
## …but errors may still exist!

- Verification based on forwarding state doesn't catch hardware errors.
  - Hardware failure.
  - Firmware bug.
  - Bug in the network controller.
- We need to use actual packets to detect these errors.



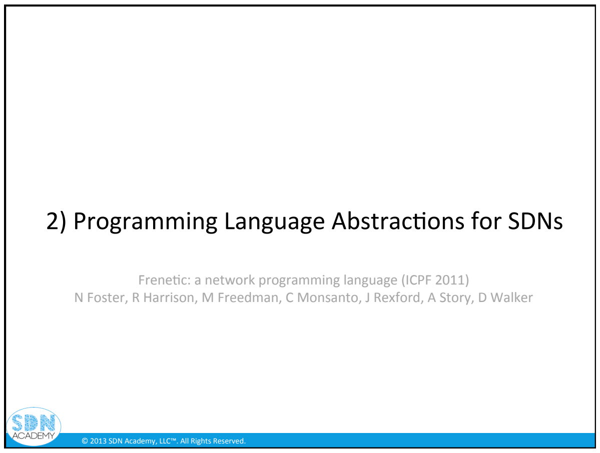
## Automatic Test Packet Generation Goals

- Monitor the data plane by sending test packets.
  - Maximum rule coverage.
  - Minimum number of packets required.
  - Constraints on terminal ports and headers of test packets.
- Once error is detected, troubleshoot it.



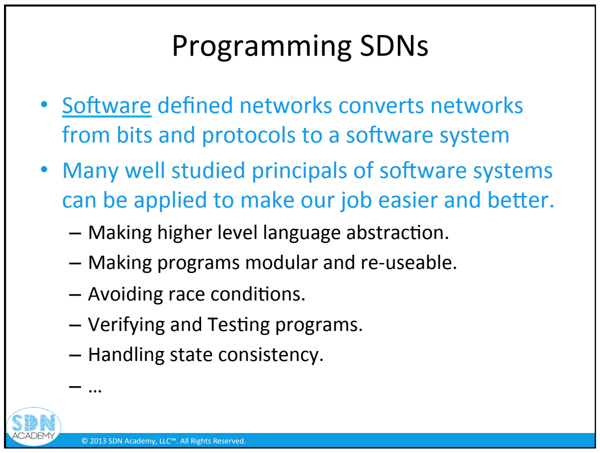
## ATPG Algorithm Sketch

- Use HSA to find reachability and all FECs between all available test terminals.
  - All packets in a FEC exercise the same rules.
- Pick a sample test packet for each FEC.
  - A single packet represent all packets in that FEC, if quantum of error is a rule.
- Use min set cover algorithm to find the minimum number of test packets to achieve maximum rule coverage.



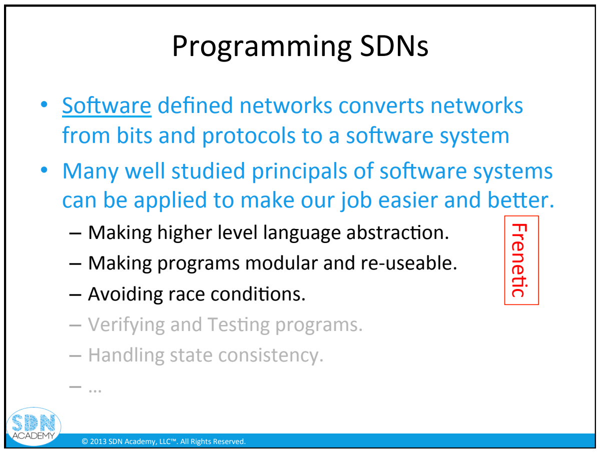
2) Programming Language Abstractions for SDNs

Frenetic: a network programming language (ICPF 2011)
N Foster, R Harrison, M Freedman, C Monsanto, J Rexford, A Story, D Walker



## Programming SDNs

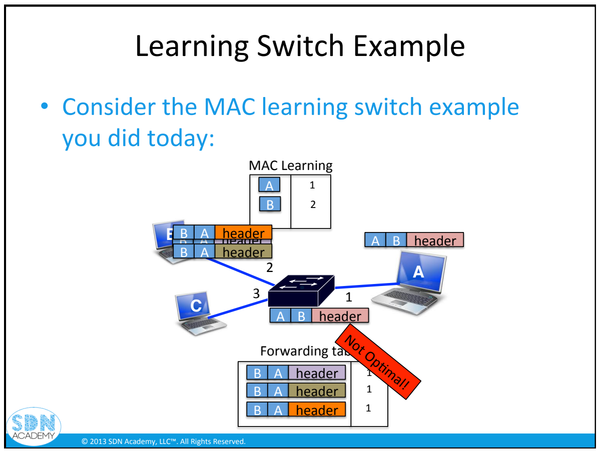
- Software defined networks converts networks from bits and protocols to a software system
- Many well studied principals of software systems can be applied to make our job easier and better.
  - Making higher level language abstraction.
  - Making programs modular and re-useable.
  - Avoiding race conditions.
  - Verifying and Testing programs.
  - Handling state consistency.
  - …



## Programming SDNs

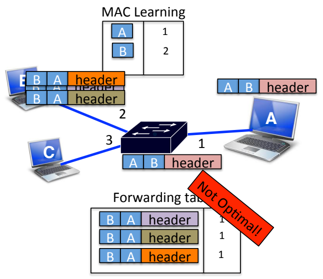
- Software defined networks converts networks from bits and protocols to a software system
- Many well studied principals of software systems can be applied to make our job easier and better.
  - Making higher level language abstraction.
  - Making programs modular and re-useable.
  - Avoiding race conditions.
  - Verifying and Testing programs.
  - Handling state consistency.
  - …

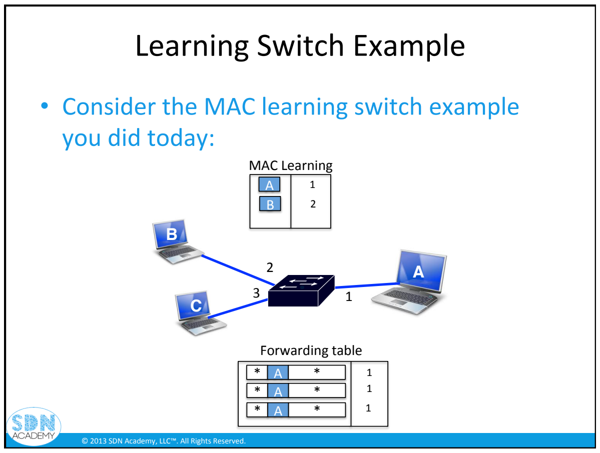
Frenetic



## Learning Switch Example

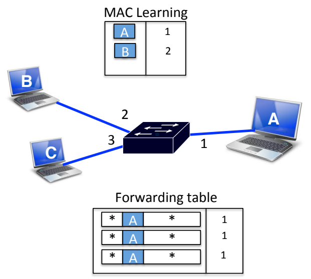
- Consider the MAC learning switch example you did today:

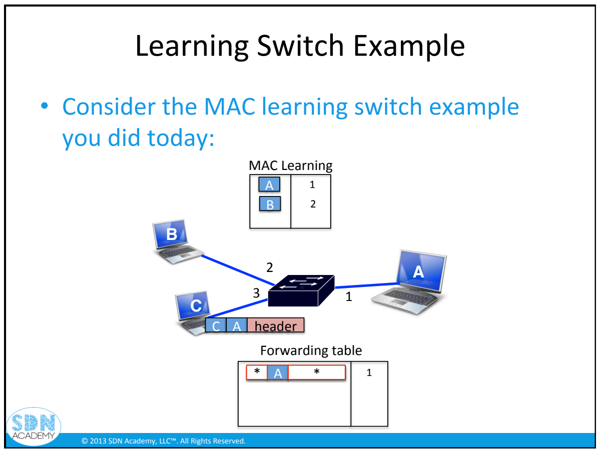




## Learning Switch Example

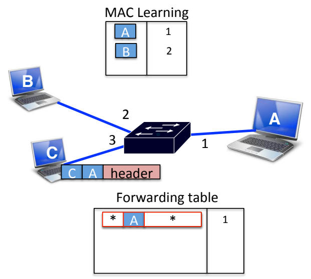
- Consider the MAC learning switch example you did today:

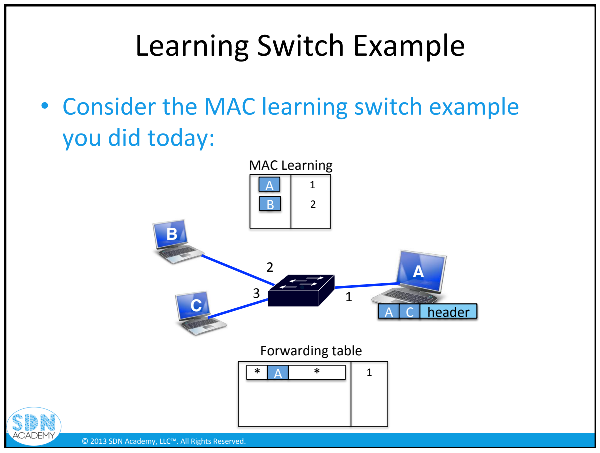




## Learning Switch Example

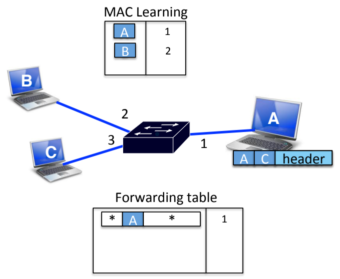
- Consider the MAC learning switch example you did today:

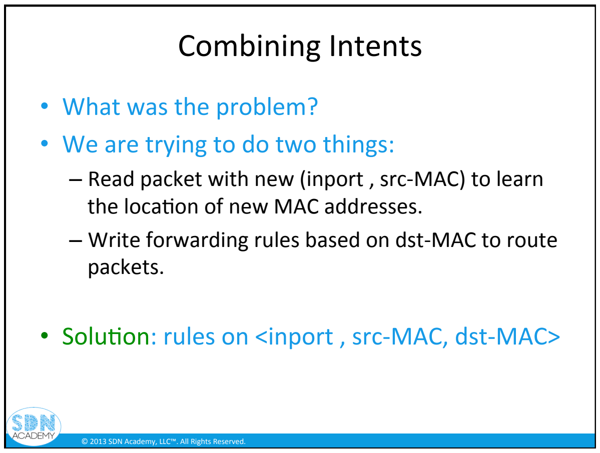




## Learning Switch Example

- Consider the MAC learning switch example you did today:

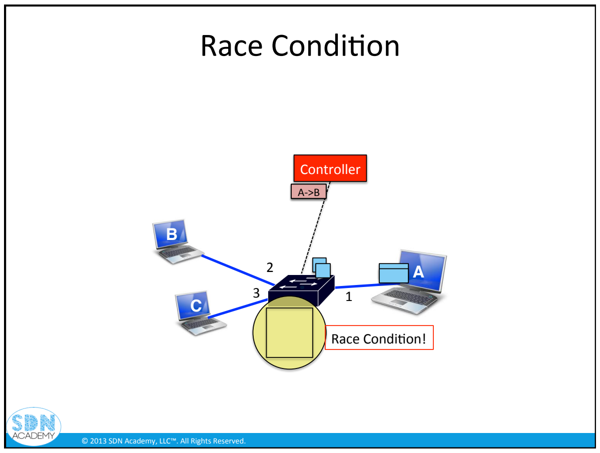




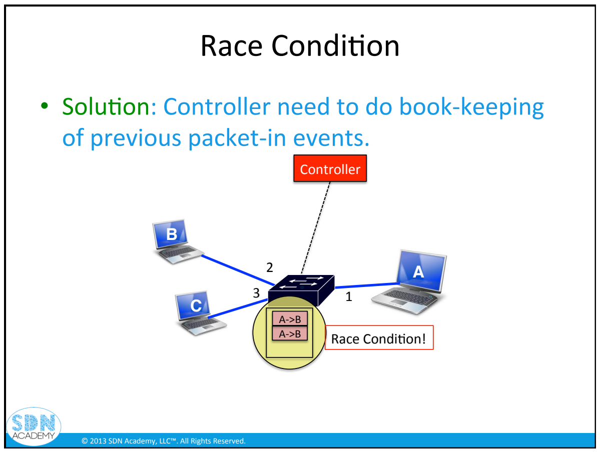
## Combining Intents

- What was the problem?
- We are trying to do two things:
  - Read packet with new (inport , src-MAC) to learn the location of new MAC addresses.
  - Write forwarding rules based on dst-MAC to route packets.

- Solution: rules on <inport , src-MAC, dst-MAC>

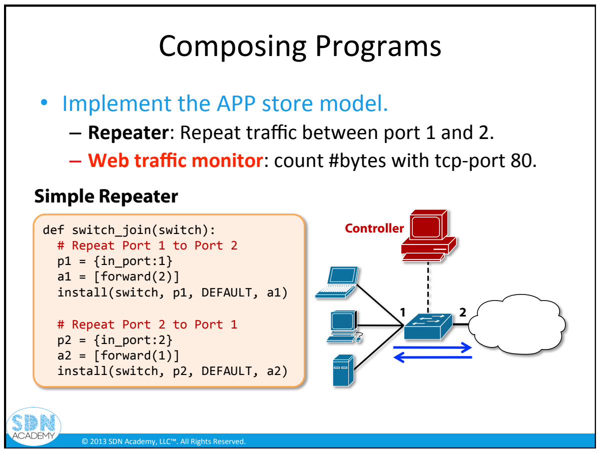


## Race Condition



## Race Condition

- Solution: Controller need to do book-keeping of previous packet-in events.



## Composing Programs

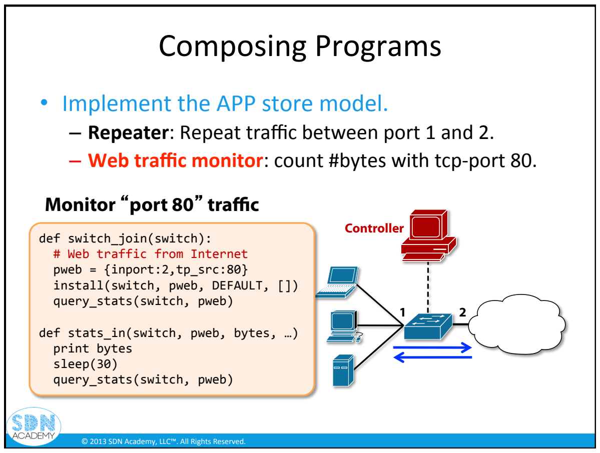
- Implement the APP store model.
  - **Repeater**: Repeat traffic between port 1 and 2.
  - **Web traffic monitor**: count #bytes with tcp-port 80.

**Simple Repeater**

```
def switch_join(switch):
  # Repeat Port 1 to Port 2
  p1 = {in_port:1}
  a1 = [forward(2)]
  install(switch, p1, DEFAULT, a1)

  # Repeat Port 2 to Port 1
  p2 = {in_port:2}
  a2 = [forward(1)]
  install(switch, p2, DEFAULT, a2)
```

Controller
1
2



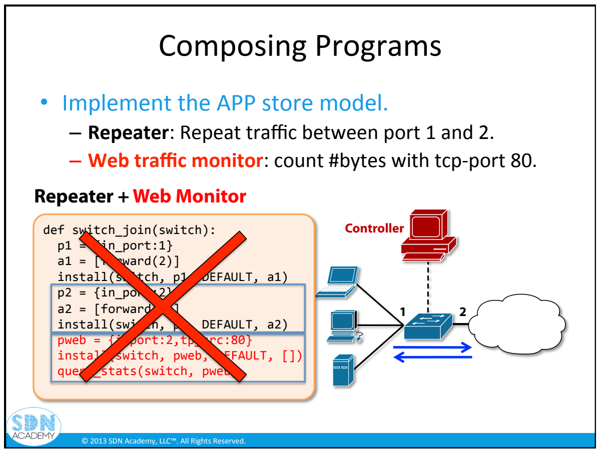
## Composing Programs

- Implement the APP store model.
  - **Repeater**: Repeat traffic between port 1 and 2.
  - **Web traffic monitor**: count #bytes with tcp-port 80.

**Monitor “port 80” traffic**

```
def switch_join(switch):
  # Web traffic from Internet
  pweb = {inport:2,tp_src:80}
  install(switch, pweb, DEFAULT, [])
  query_stats(switch, pweb)

def stats_in(switch, pweb, bytes, …)
  print bytes
  sleep(30)
  query_stats(switch, pweb)
```

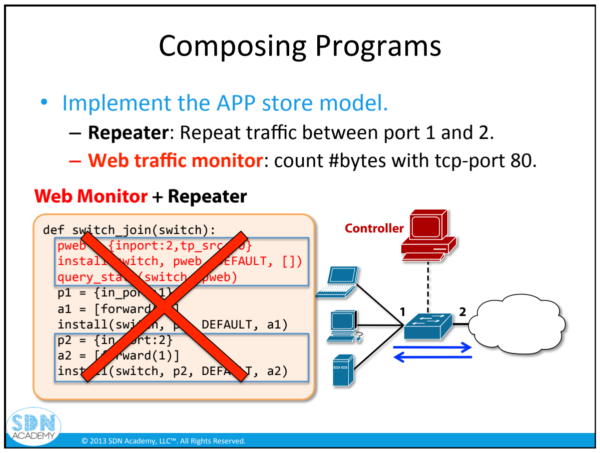


## Composing Programs

- Implement the APP store model.
  - **Repeater**: Repeat traffic between port 1 and 2.
  - **Web traffic monitor**: count #bytes with tcp-port 80.

**Repeater + Web Monitor**

```
def switch_join(switch):
  p1 = {in_port:1}
  a1 = [forward(2)]
  install(switch, p1, DEFAULT, a1)
  p2 = {in_port:2}
  a2 = [forward(1)]
  install(switch, p2, DEFAULT, a2)
  pweb = {inport:2,tp_src:80}
  install(switch, pweb, DEFAULT, [])
  query_stats(switch, pweb)
```

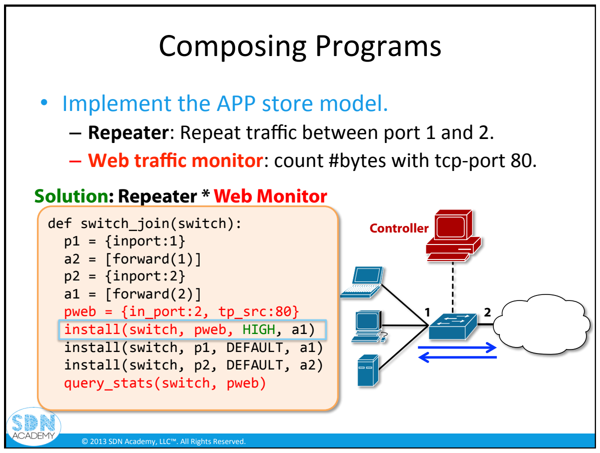


## Composing Programs

- Implement the APP store model.
  - **Repeater**: Repeat traffic between port 1 and 2.
  - **Web traffic monitor**: count #bytes with tcp-port 80.

**Web Monitor + Repeater**

```
def switch_join(switch):
  pweb = {inport:2,tp_src:80}
  install(switch, pweb, DEFAULT, [])
  query_stats(switch, pweb)
  p1 = {in_port:1}
  a1 = [forward(2)]
  install(switch, p1, DEFAULT, a1)
  p2 = {in_port:2}
  a2 = [forward(1)]
  install(switch, p2, DEFAULT, a2)
```

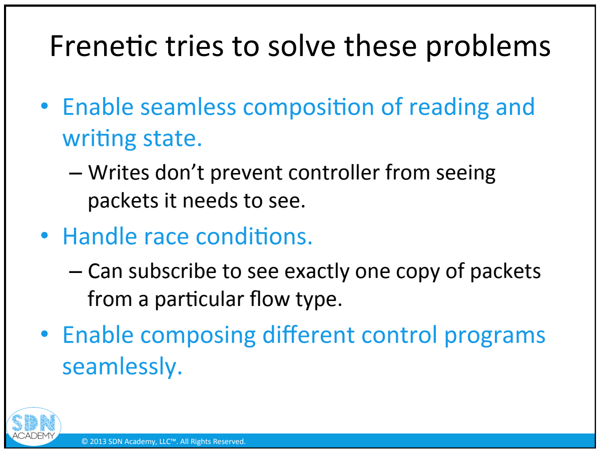


## Composing Programs

- Implement the APP store model.
  - **Repeater**: Repeat traffic between port 1 and 2.
  - **Web traffic monitor**: count #bytes with tcp-port 80.

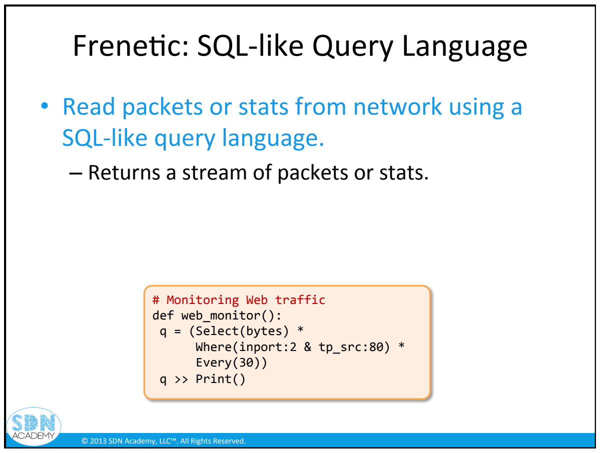
**Solution: Repeater * Web Monitor**

```
def switch_join(switch):
  p1 = {inport:1}
  a2 = [forward(1)]
  p2 = {inport:2}
  a1 = [forward(2)]
  pweb = {in_port:2, tp_src:80}
  install(switch, pweb, HIGH, a1)
  install(switch, p1, DEFAULT, a1)
  install(switch, p2, DEFAULT, a2)
  query_stats(switch, pweb)
```



## Frenetic tries to solve these problems

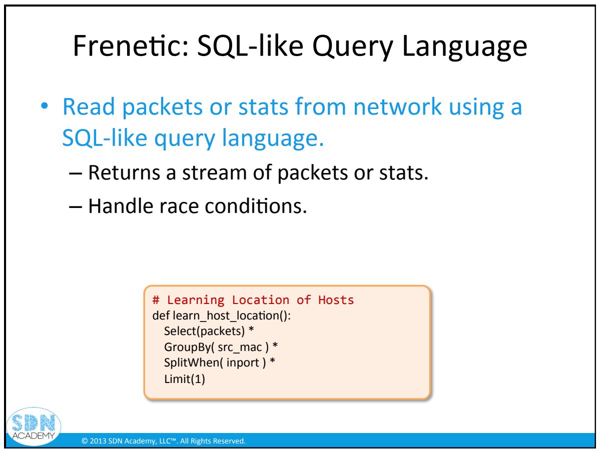
- Enable seamless composition of reading and writing state.
  - Writes don't prevent controller from seeing packets it needs to see.
- Handle race conditions.
  - Can subscribe to see exactly one copy of packets from a particular flow type.
- Enable composing different control programs seamlessly.



## Frenetic: SQL-like Query Language

- Read packets or stats from network using a SQL-like query language.
  - Returns a stream of packets or stats.

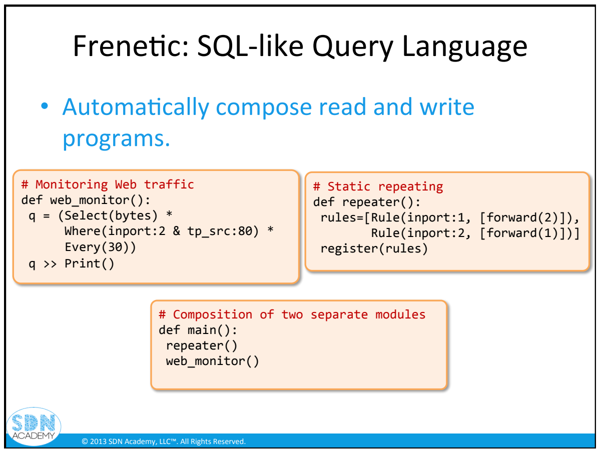
```
# Monitoring Web traffic
def web_monitor():
 q = (Select(bytes) *
      Where(inport:2 & tp_src:80) *
      Every(30))
 q >> Print()
```



## Frenetic: SQL-like Query Language

- Read packets or stats from network using a SQL-like query language.
  - Returns a stream of packets or stats.
  - Handle race conditions.

```
# Learning Location of Hosts
def learn_host_location():
  Select(packets) *
  GroupBy( src_mac ) *
  SplitWhen( inport ) *
  Limit(1)
```



## Frenetic: SQL-like Query Language

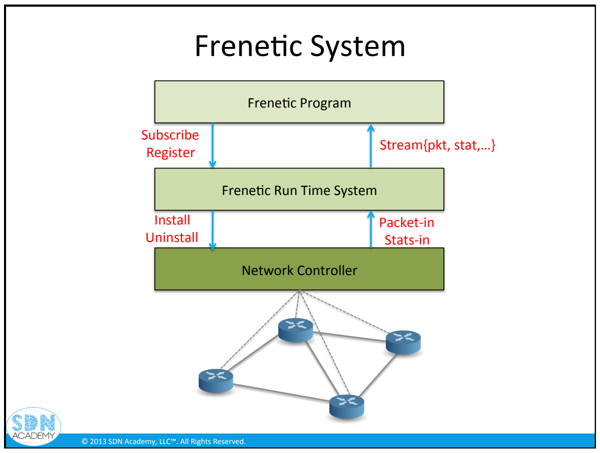
- Automatically compose read and write programs.

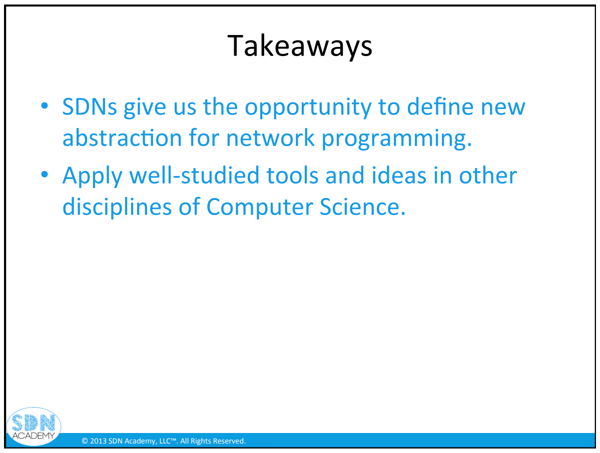
```
# Monitoring Web traffic
def web_monitor():
 q = (Select(bytes) *
      Where(inport:2 & tp_src:80) *
      Every(30))
 q >> Print()
```

```
# Static repeating
def repeater():
 rules=[Rule(inport:1, [forward(2)]),
        Rule(inport:2, [forward(1)])]
 register(rules)
```

```
# Composition of two separate modules
def main():
 repeater()
 web_monitor()
```

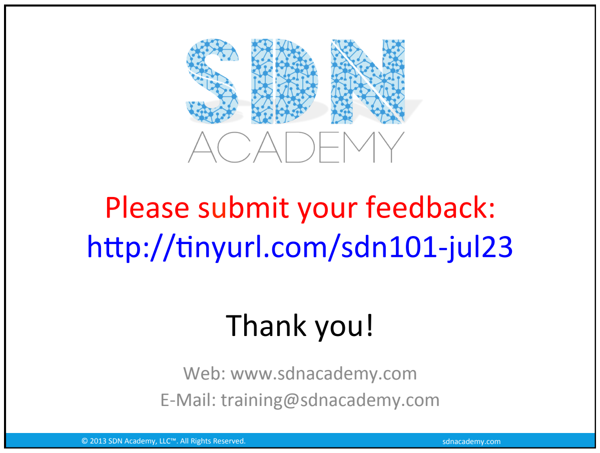
## Frenetic System





## Takeaways

- SDNs give us the opportunity to define new abstraction for network programming.
- Apply well-studied tools and ideas in other disciplines of Computer Science.



Please submit your feedback:
http://tinyurl.com/sdn101-jul23

Thank you!

Web: www.sdnacademy.com
E-Mail: training@sdnacademy.com