UNIVERSITY OF WATERLOO

# CS 856: Programmable Networks

## Lecture 6: Applications to Traditional Networks

Mina Tahmasbi Arashloo

Winter 2023

# Logistics

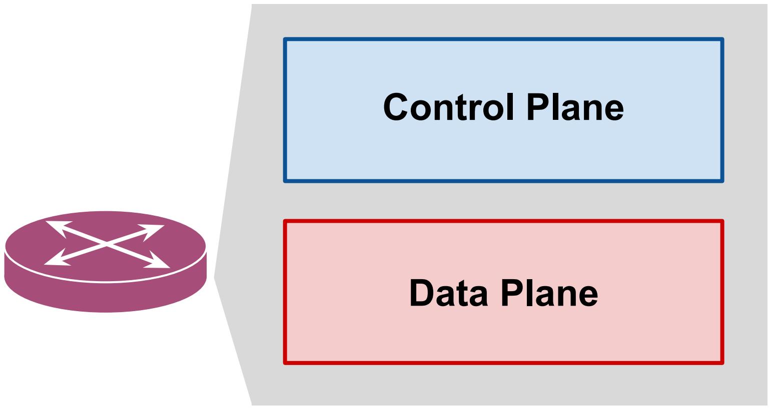
- Happy Reading Week!
    - Nothing due next week 🎉
- Next set of reviews are due **Monday, Feb 27, at 5pm.**
- Project progress report is due **March 2nd or 10th?**
    - Respond to the anonymous poll on slack by tomorrow night

# "Programming" traditional networks

- You don't have direct programmatic control over devices, just configuration interfaces, just for a handful of protocols

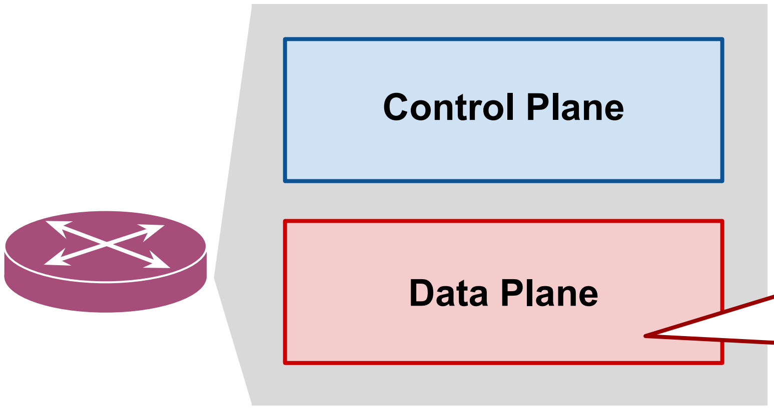
# "Programming" traditional networks

- You don't have direct programmatic control over devices, just configuration interfaces, just for a handful of protocols



# "Programming" traditional networks

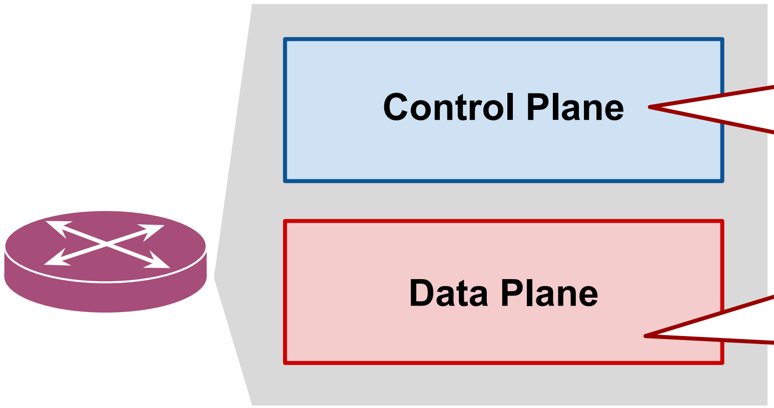
- You don't have direct programmatic control over devices, just configuration interfaces, just for a handful of protocols




- Exposes a fixed-function pipeline.
- Unless you were who built the chip, you couldn't change which headers are parsed, which tables match on which fields, and in what order they are executed.

# "Programming" traditional networks

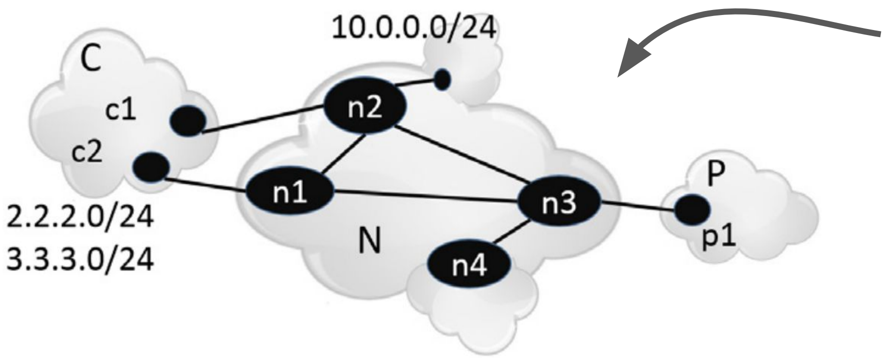
- You don't have direct programmatic control over devices, just configuration interfaces, just for a handful of protocols




- A (closed-source) OS and a set of apps that comes with the devices.
- Can only configure the already-implemented protocols through a limited and not-so-easy-to-use interface.

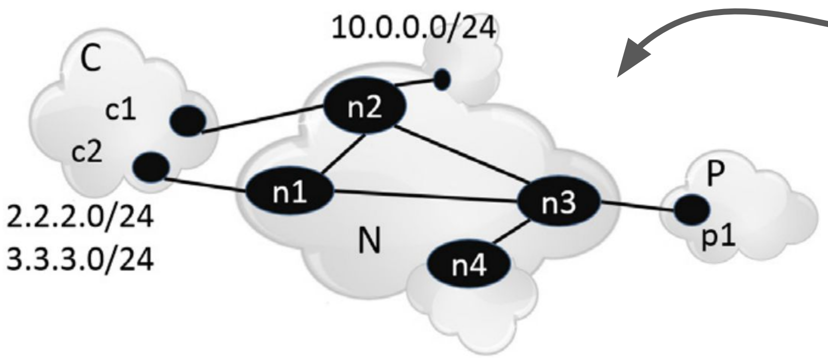
- Exposes a fixed-function pipeline.
- Unless you were who built the chip, you couldn't change which headers are parsed, which tables match on which fields, and in what order they are executed.

# A VERY simple example (from Batfish, NSDI'15)




//----------Configuration of n1----------
1 ospf interface int1_2 metric 1
2 ospf interface int1_3 metric 1
3 prefix-list PL_C 2.2.2.0/24 3.3.3.0/24
4 bgp neighbor c2 AS C apply PL_C
//----------Configuration of n2----------
1 ospf interface int2_1 metric 1
2 ospf interface int2_3 metric 1
3 ospf-passive interface int2_5 ip 10.0.0.0/24
4 ospf redistribute connected metric 10
5 prefix-list PL_C 2.2.2.0/24
6 bgp neighbor c1 AS C apply PL_C
//----------Configuration of n3----------
1 ospf interface int3_1 metric 1
2 ospf interface int3_2 metric 1
3 ospf interface int3_4 metric 1
4 ospf redistribute static metric 10
5 bgp neighbor p1 AS P Accept ALL
6 static route 10.0.0.0/24 drop, log

# A VERY simple example (from Batfish, NSDI'15)




//----------Configuration of n1----------
1 ospf interface int1_2 metric 1
2 ospf interface int1_3 metric 1
3 prefix-list PL_C 2.2.2.0/24 3.3.3.0/24
4 bgp neighbor c2 AS C apply PL_C
//----------Configuration of n2----------
1 ospf interface int2_1 metric 1
2 ospf interface int2_3 metric 1
3 ospf-passive interface int2_5 ip 10.0.0.0/24
4 ospf redistribute connected metric 10
5 prefix-list PL_C 2.2.2.0/24
6 bgp neighbor c1 AS C apply PL_C
//----------Configuration of n3----------
1 ospf interface int3_1 metric 1
2 ospf interface int3_2 metric 1
3 ospf interface int3_4 metric 1
4 ospf redistribute static metric 10
5 bgp neighbor p1 AS P Accept ALL
6 static route 10.0.0.0/24 drop, log

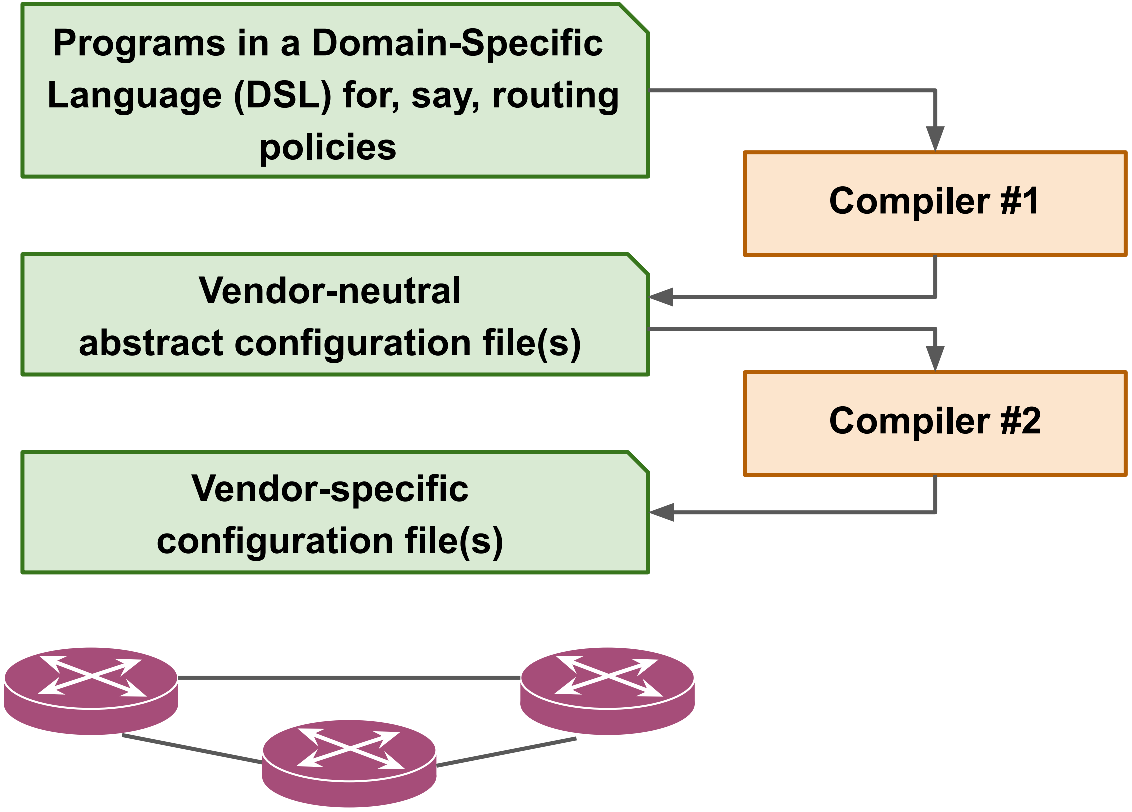
Now imagine managing a network this way

- for thousands of devices
- when every vendor has its own configuration interface
- when configuration files can grow to thousands of lines
- when operators can tweak the configuration by running ad-hoc commands using a CLI

# Abstraction and automation in traditional networks

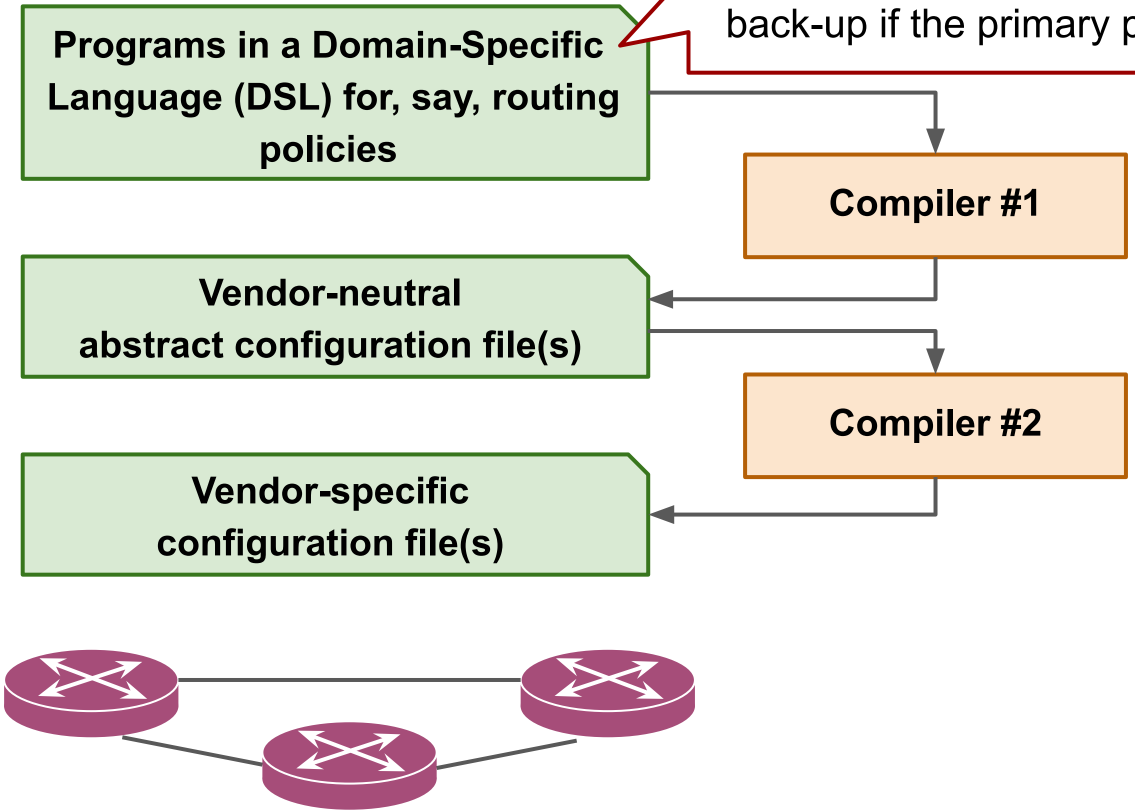
- Even without full programmability, we need abstraction and automation in traditional networks.
- Work on automated management tools predates SDN.
- But it has been affected by the focus on high-level well-defined abstractions in the research on SDN and programmable networks.
- We will discuss some examples today:
  - Automated configuration generation
  - Well-defined/formal specifications of protocols and device functionality
  - Automated validation

# Automated Configuration Generation

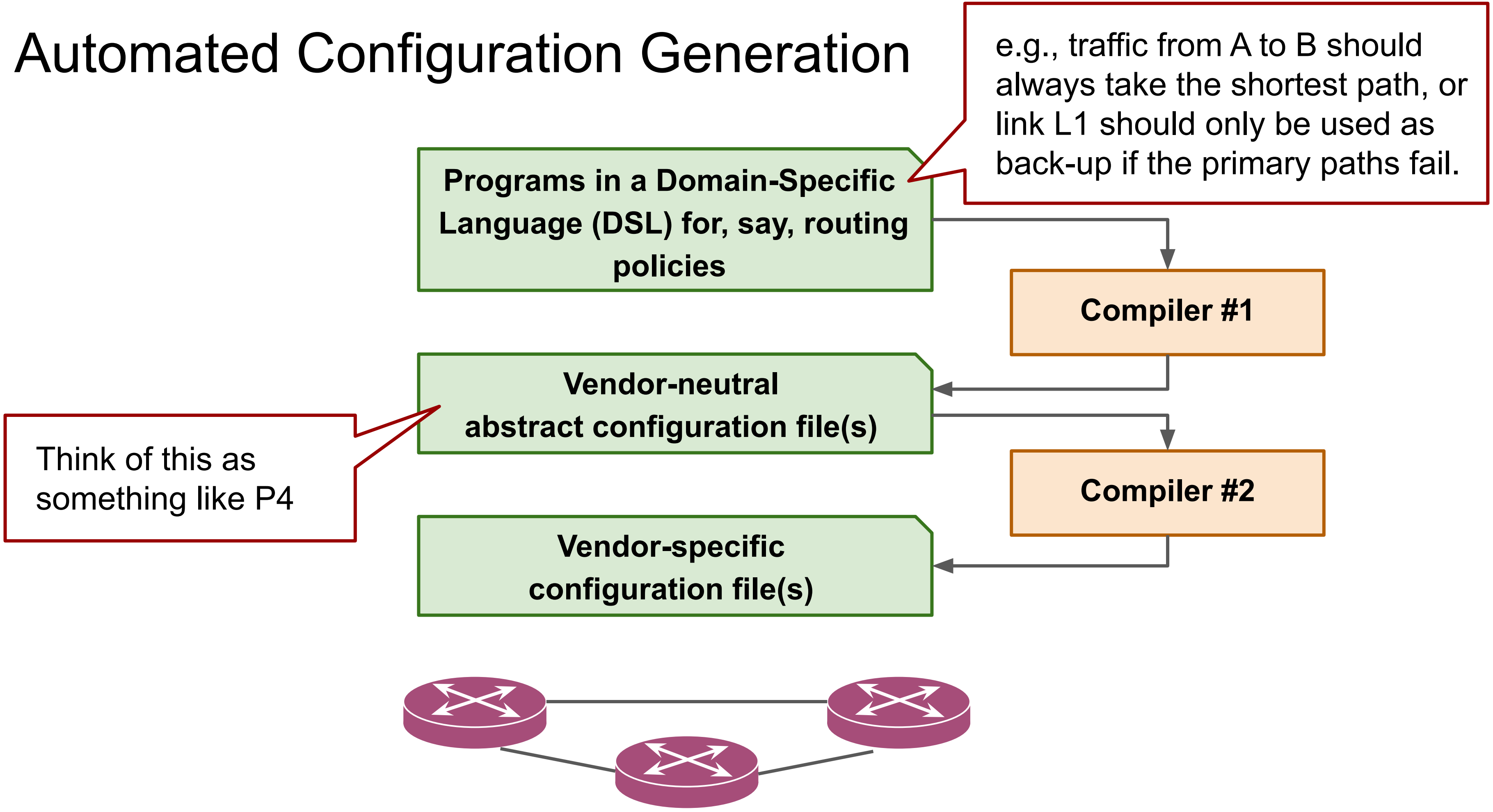


# Automated Configuration Generation

e.g., traffic from A to B should always take the shortest path, or link L1 should only be used as back-up if the primary paths fail.



# Automated Configuration Generation



# Automated Configuration Generation

e.g., traffic from A to B should always take the shortest path, or link L1 should only be used as back-up if the primary paths fail.

An example of this approach is Propane (SIGCOMM'16)

**Programs in a Domain-Specific Language (DSL) for, say, routing policies**

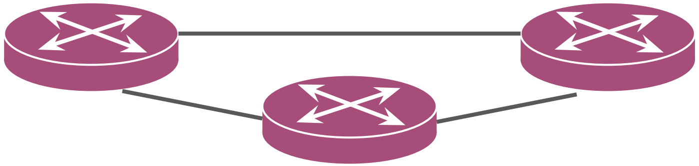
**Compiler #1**

**Vendor-neutral abstract configuration file(s)**

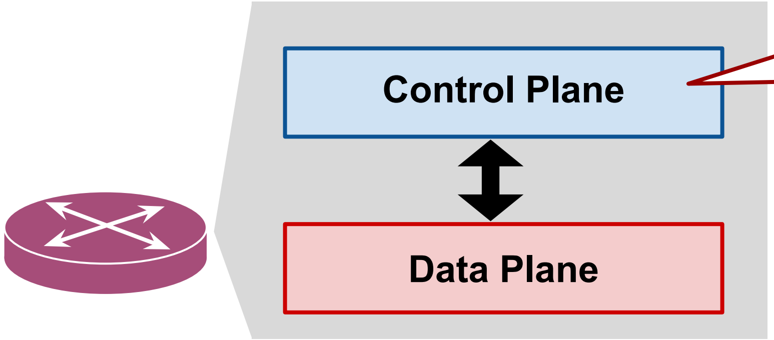
Think of this as something like P4

**Compiler #2**

**Vendor-specific configuration file(s)**

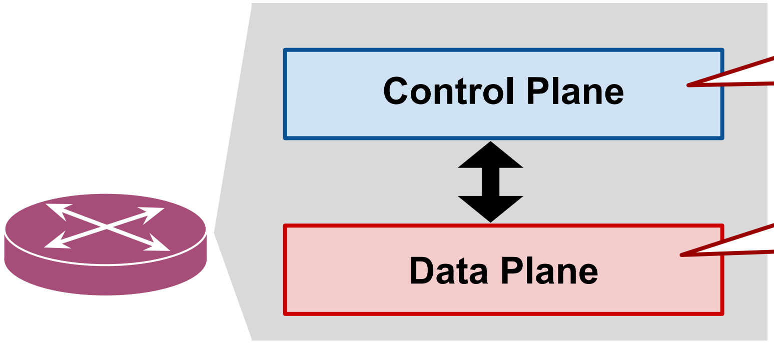


# Traditional networks: Not-so-well-defined specifications




- The software stack for the data plane (a.k.a the switch OS)
- Runs protocols like OSPF, BGP, etc
- Configures the data-plane accordingly

# Traditional networks: Not-so-well-defined specifications



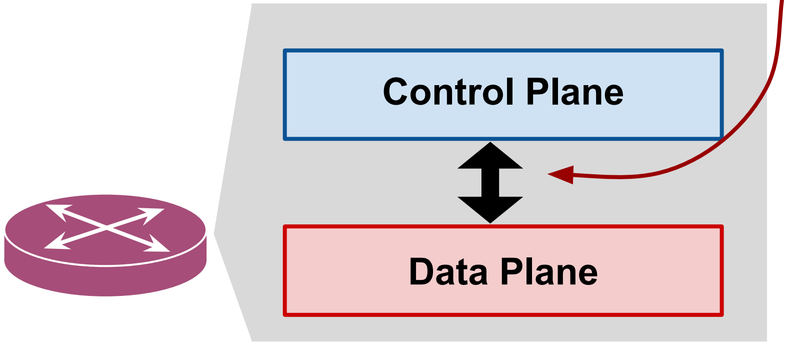

- The software stack for the data plane (a.k.a the switch OS)
- Runs protocols like OSPF, BGP, etc
- Configures the data-plane accordingly

- Come from different vendors
- Can have different capabilities and internals, and expose different configuration APIs.

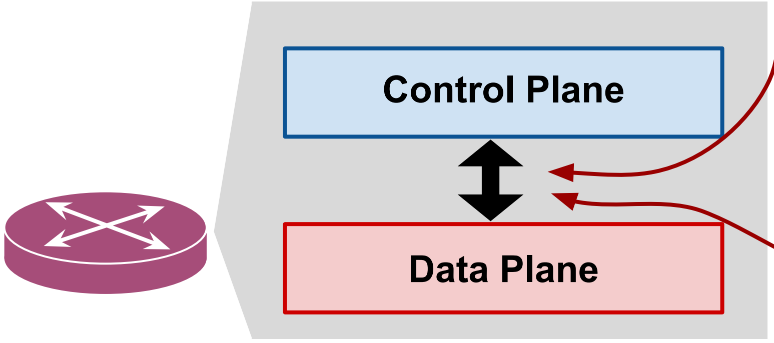
# Traditional networks: Not-so-well-defined specifications

*Informal specifications in English*

- Ambiguous
- difficult to maintain a complete updated version



# Traditional networks: Not-so-well-defined specifications




*Informal specifications in English*

- Ambiguous
- difficult to maintain a complete updated version

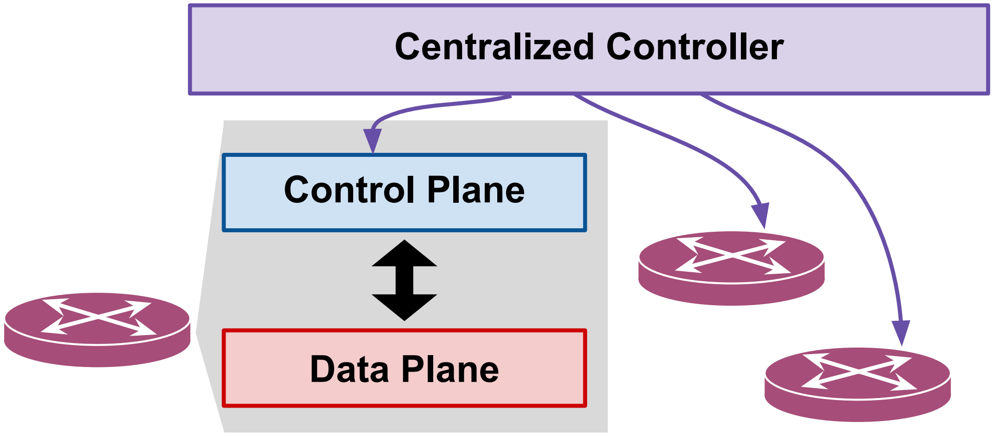
Without proper abstractions, you may not be able to reuse much of the switch software stack across different data planes

# Towards well-defined/formal specifications

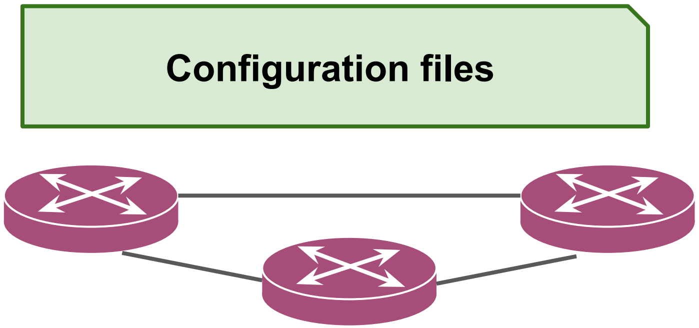
- Switch Abstraction Interface (SAI)
    - "a collection of C-style interfaces" for common functionality in traditional fixed-function switches/routers (e.g., destination-based forwarding, VLAN, ACL, etc.)
- Software for Open Networking in the Cloud (SONiC)
    - Open source network operating system based on Linux built on SAI
- Use P4 to specify (as opposed to program) the data-plane functionality
    - e.g., make SAI more well-defined by writing the interfaces and specifying the pipeline order in P4.

# Towards well-defined/formal specifications

- Even in traditional networks, you may still want to configure some functionality (e.g., ACLs) from a centralized "controller".
- Using unified abstractions in individual devices makes that a lot easier.
- E.g., using P4, specifically, we can use existing control interfaces and platforms like P4 Runtime



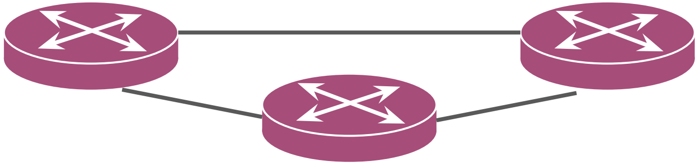
# Automated Validation



# Automated Validation

Properties we would like the network to satisfy

Configuration files

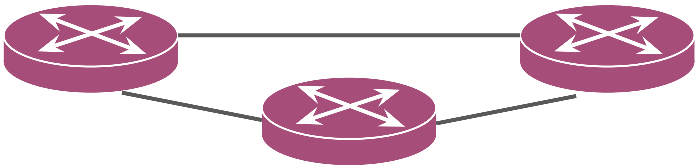


# Automated Validation

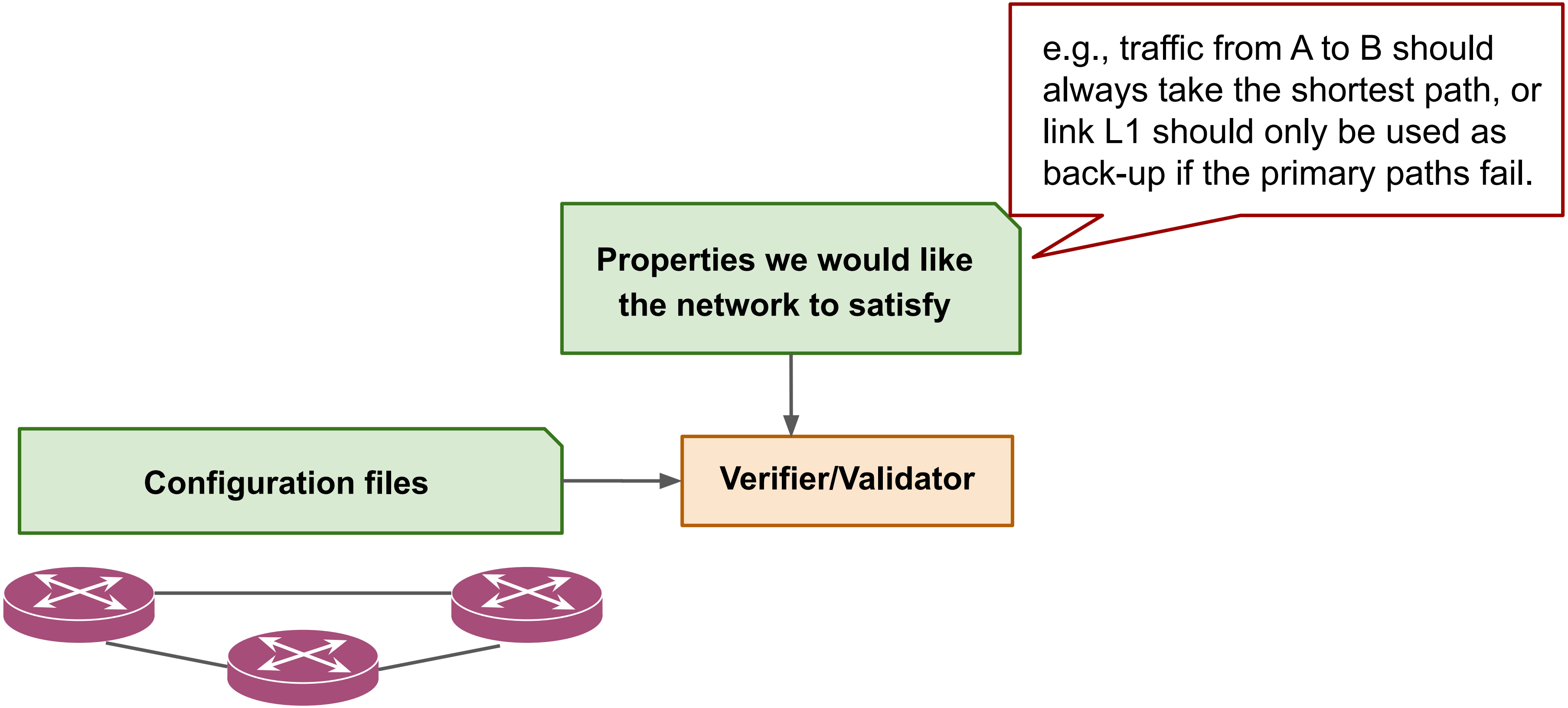
e.g., traffic from A to B should always take the shortest path, or link L1 should only be used as back-up if the primary paths fail.

**Properties we would like the network to satisfy**

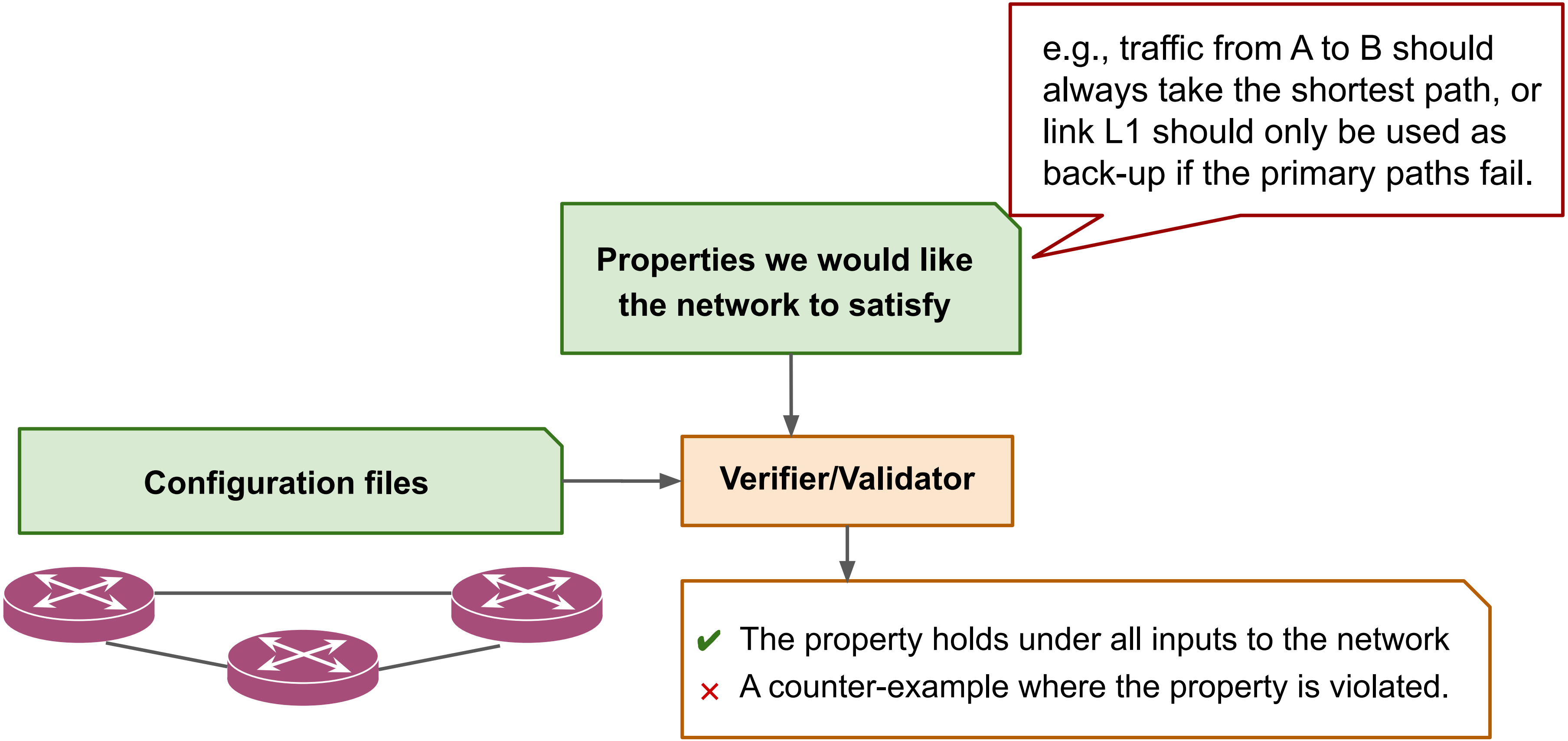
**Configuration files**



# Automated Validation



# Automated Validation

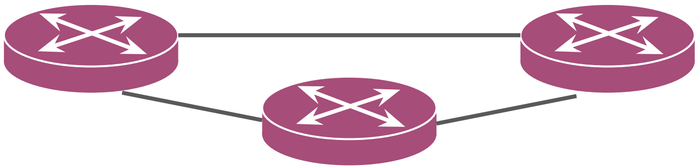


# Automated Validation

e.g., traffic from A to B should always take the shortest path, or link L1 should only be used as back-up if the primary paths fail.

**Properties we would like the network to satisfy**

**Configuration files**



**Test generation**

Test cases (e.g., packet traces) to exercise individual switches or end-to-end network paths for desired properties.

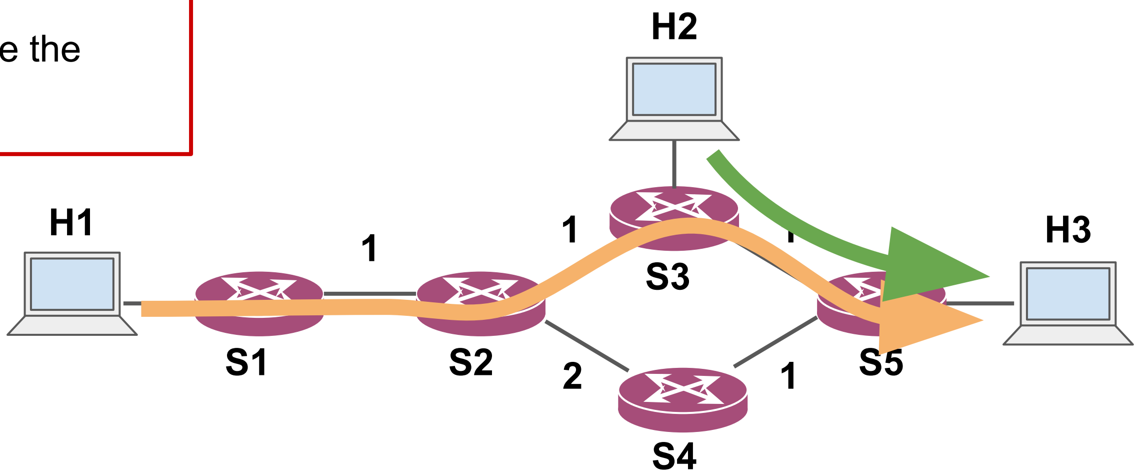
# How does abstraction help?

- Automated testing and verification did not start with and is not limited to programmable networks.
  - We will have a dedicated lecture on network verification next time.
- But, there is a rich literature on program verification and testing in the formal methods and PL community.
- With programming abstraction for a single device or collection of devices, we can reuse so much of that knowledge and expertise, as well as existing tools, and customize them to the networking domain.

# Paper 1: Central control over distributed routing

- Remember the "indirect control" example from lecture 1?

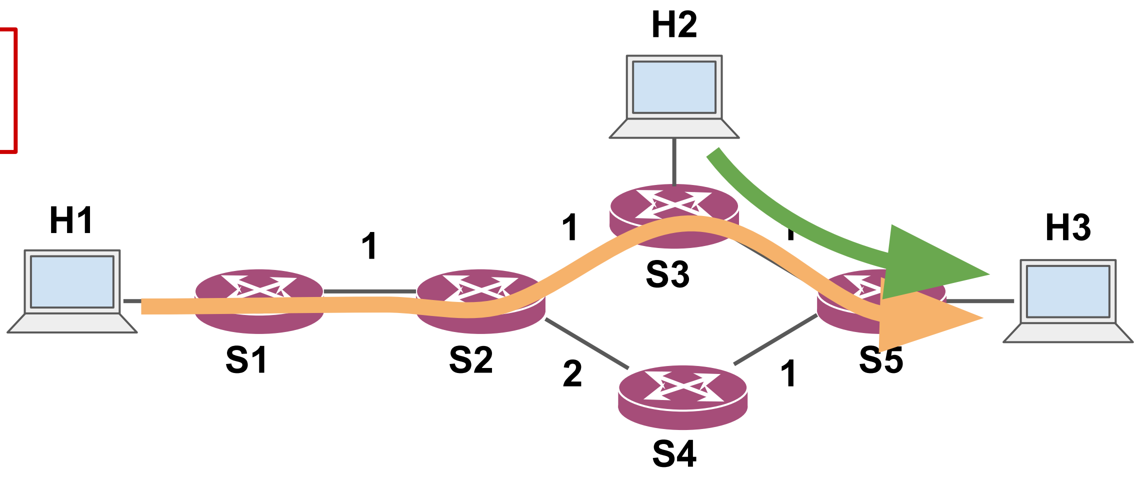
How can we get H1 to use the bottom path?



# Paper 1: Central control over distributed routing

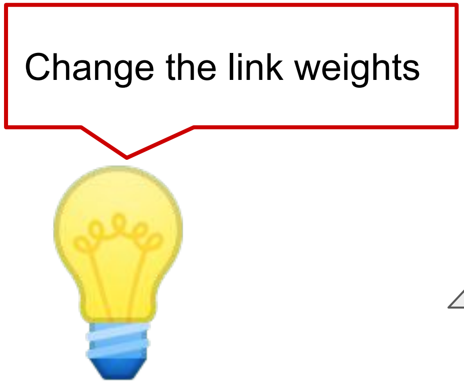
- Remember the "indirect control" example from lecture 1?

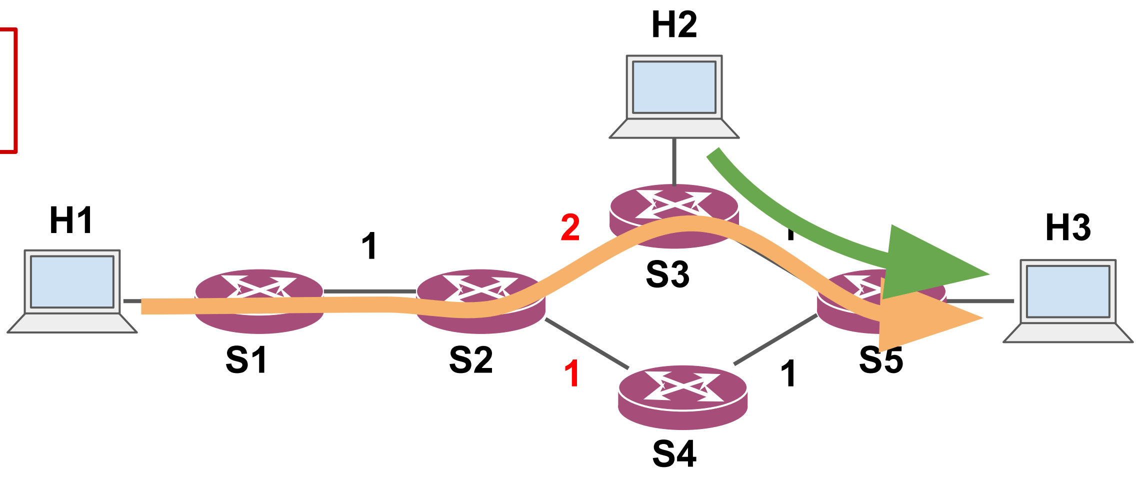
Change the link weights



# Paper 1: Central control over distributed routing

- Remember the "indirect control" example from lecture 1?

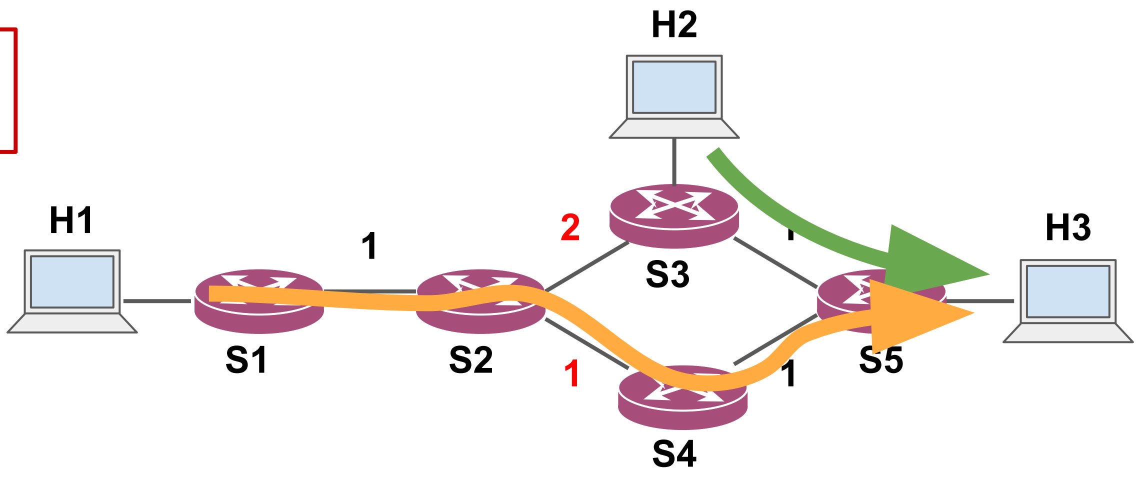


# Paper 1: Central control over distributed routing

- Remember the "indirect control" example from lecture 1?

Change the link weights



# Paper 1: Central control over distributed routing

- Remember the "indirect control" example from lecture 1?
- This paper proposes Fibbing, an approach to make that easier using abstraction and automation
- Fibbing allows operators to specify their desired network paths.
- It then introduces fake nodes and links into the distributed routing protocol, in a way that the computed paths over the augmented topology are the desired specified paths.
- Best paper award, SIGCOMM 2015

# Paper 2: Don’t Mind the Gap: Bridging Network-wide Objectives and Device-level Configurations

- Automated configuration generation for BGP
- Operators can specify network-wide routing objectives using Propane's domain-specific language
- The Propane compiler generates vendor-neutral abstract BGP configurations that satisfy those objectives.
- Best paper award, SIGCOMM 2016

# Additional Resources

- SwitchV (SIGCOMM 2022)
  - Google's use of P4 for specifying the behavior of fixed-function switches and the resulting verification/testing framework.
- P4 Integrated Network Stack (PINS) website