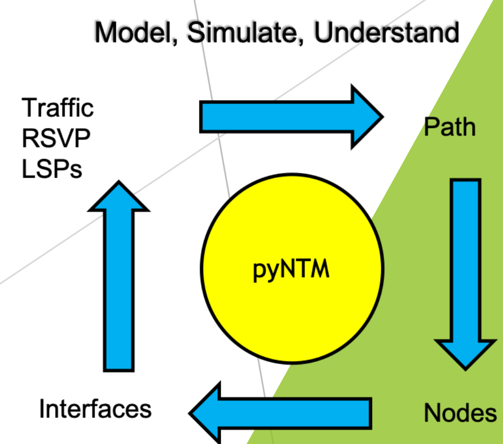
NANOG Session: open sourcing the network model and unlocking the value of understanding the wide area network

python3 Network Traffic Modeler (pyNTM)

Tim Fiola

Network Modeling and Automation Enthusiast

deck v11



Agenda

- Problem statement
- Network modeling is strategic
- What is a network model?
- We need open source tools in this space
- What is pyNTM and why is it helpful?
- pyNTM features and roadmap
- Can pyNTM help you now?
- Next steps
- Demo

Problem Statement – Understanding the wide area network during failure states and how to grow the network is difficult

- In a large, meshy network, it becomes difficult to understand how a given failure will truly impact interface utilization in other parts of the network
  - Leads to educated guessing and general *rules of thumb* on how/where to augment/grow the network

Understanding WAN behavior is difficult (continued)

- Auto-bandwidth RSVP network adds additional complexity
  - The demands (traffic) a link handles can change throughout the day/week/season
  - Bypass LSPs can have non-intuitive impacts
  - Auto-bandwidth LSP behavior can be non-deterministic
  - Adding capacity in one part of the network can impact LSP behavior in the opposite part of the network

Aggravating factors

- WAN capacity cannot be solved simply throwing money at the problem
  - WAN circuits are expensive
  - WAN circuits are not always available
  - It often takes a long time to turn up new capacity

Let's talk about modeling!

But why male models?

. . . well, the Wide-Area-Network kind

Network modeling provides insight into WAN behavior . . .

- Modeling allows unique, data-based understanding of how network will behave during
  - Failover
  - Changes in the traffic matrix
  - Changes in topology, such as adding RSVP mesh or changing a link metric
- Provides insight as to how auto-bandwidth LSP meshes will behave

. . . and this insight provides strategic value by allowing **efficient capital allocation**

- This increased understanding of the WAN helps prevent
  - Overbuilding WAN links, which strands **capital**
  - Underbuilding WAN links, which increases **risk**

A network model helps people in the following roles to perform better

- Capacity Planner
  - Plan network to optimize latency, cost, simplest topology, etc
- Network Engineer
  - Test different topologies
- Anyone working a maintenance
  - Simulate the effects of taking down a router for a maintenance
- Anyone with interest in network performance

The Network Model

An Overview

A network model has two input components

- Traffic Matrix
  - Each entry describes a *demand*
  - Each demand has
    - *magnitude*, which describes how much traffic is in that demand
    - A source and destination node
  - An example is on the next slide
- Topology
  - Layer 3 nodes
  - Circuits between layer 3 nodes
    - comprised of 2 unidirectional interfaces
  - Shared Risk Link Groups (SRLGs)
  - RSVP LSPs

Traffic Matrix

- The traffic matrix for a network will vary throughout the day, month, season, etc
- Getting good traffic matrices can be challenging . . .
  - . . . but understanding your network's traffic matrices allows for truly effective engineering and planning

Sample traffic matrix (Mbps)

Source \ Destination	A	B	C
A	-	45	120
B	60	-	
C	75	150	-

This example traffic matrix shows traffic sourced from Node A destined to Node C with a magnitude of 120Mbps

How will this traffic transit the network?

Network modeling provides *simulation* capability

- Applying the traffic matrix to the topology and converging the model produces a *simulation*
  - The simulation provides data on network behavior (state) for the given traffic matrix and topology
- For a given day, you can produce a simulation for different parts of the day by creating a traffic matrix and/or topology for each part of the day
  - What happens during a given failure if it were to occur at different parts of the day?
  - What is the best time to conduct a maintenance on a given router?
  - Where is the best place to augment the network to best handle our holiday traffic matrices?

Without modeling, *rules of thumb* are often used for WAN Engineering/Planning

- Rules of thumb
  - Are general
    - Ex: *Augment circuits when utilization reaches 50%*
  - May result in overbuilding (stranding capital) or underbuilding (increasing risk)
  - Do not necessarily protect against failures you are interested in
  - Do not provide any insight as to what failures may be interesting
  - May have unintended consequences

Modeling advantages and benefits

- Simulations provide
  - Insight as to how your network traffic will behave during a **failure** event
  - Insight as to how your network will behave with **additional traffic**
  - Better understanding of how **RSVP** LSP meshes will behave
- Modeling can show you where the WAN is **vulnerable**
  - What failures *SHOULD* you be interested in?
- A simulation engine provides a **platform** upon which to build sophisticated analysis tools
  - Ex: I want to design/plan a network to optimize costs

Modeling advantages and benefits (continued)

- Modeling wide area network behavior allows you to
  - Efficiently allocate capacity/**capital** where it's really needed
  - Plan for and understand events you **care** about
- Simulations produce actionable **DATA**!
  - A model is a great source from which to mine data
- At a minimum, modeling allows you to make a more **educated** decision
  - You don't need a sophisticated model to begin to reap the benefits of modeling

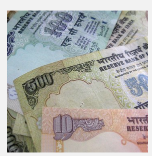
Some example use cases for network models

Understanding current network topology

How many ECMP paths does a given demand take across the IGP topology?

Understanding failover by modeling failures

- Links
- Nodes
- Shared risk link group(s) (SRLG)



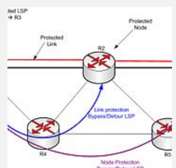
Understanding where it makes sense to augment a network

- Deploy capital where it's most needed
- Don't strand capital

More example use cases for network model

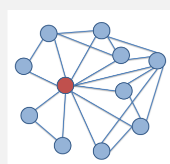
Understanding how changes in the network affect traffic flow

- More/less traffic
- Adding capacity to existing links
- New links
- Metric changes



RSVP Implementations

- Adding RSVP overlay to IGP network
- Adding/removing parallel LSPs
- Failover



What failures <u>*should*</u> I be interested in?

Making network modeling more accessible to everyone

- We need open-source tools that allow programmatic network modeling and simulation
- Specifically, there are two needed components
  - Open source modeling engines (pyNTM)
  - Open source tools to create reasonable traffic matrices
- Nice-to-have: open source GUI for visualization

So, what is pyNTM?

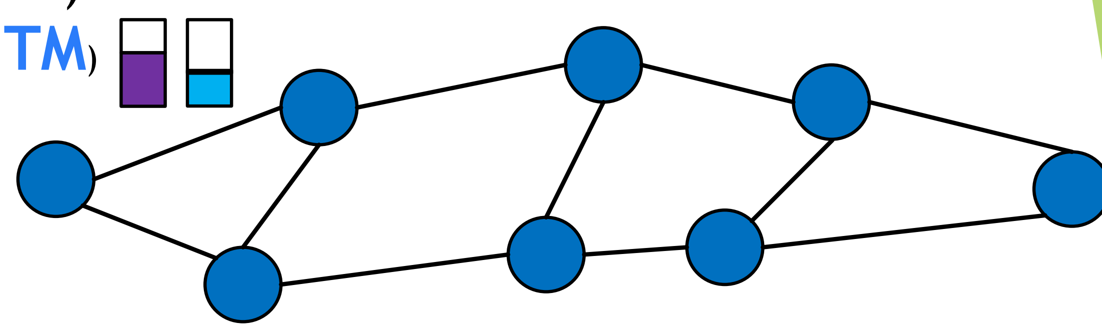
- pyNTM is the python3 Network Traffic Modeler
- pyNTM is an **open source** WAN *modeling engine*
- Applies a traffic matrix to a network topology to route traffic as the network would
  - Uses *networkx* module to get the topology path info
  - *networkx* is a GREAT tool to get path info in a topology . . .
  - . . . but there’s more to modeling than just path info
- pyNTM builds on *networkx* paths to create *network-specific state*

Networkx and pyNTM roles in pyNTM simulations

Demand (traffic) paths (**networkx**)

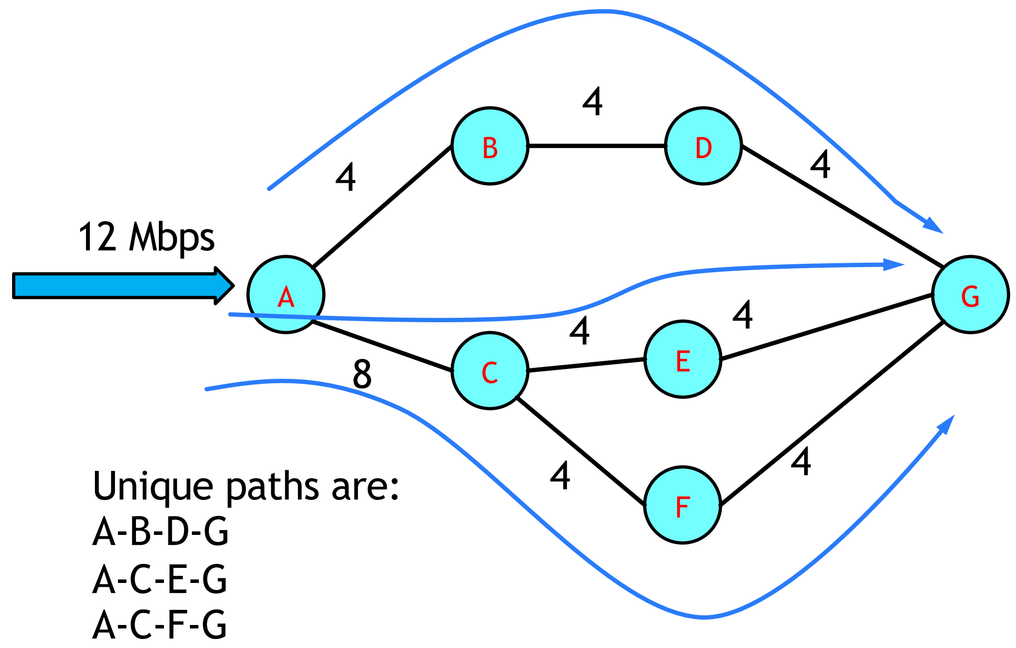
RSVP paths (**networkx**)

Interface utilization (**pyNTM**)
Interface reserved bandwidth (**pyNTM**)
Demand path’s interfaces (**pyNTM**)
RSVP LSP path’s interfaces (**pyNTM**)



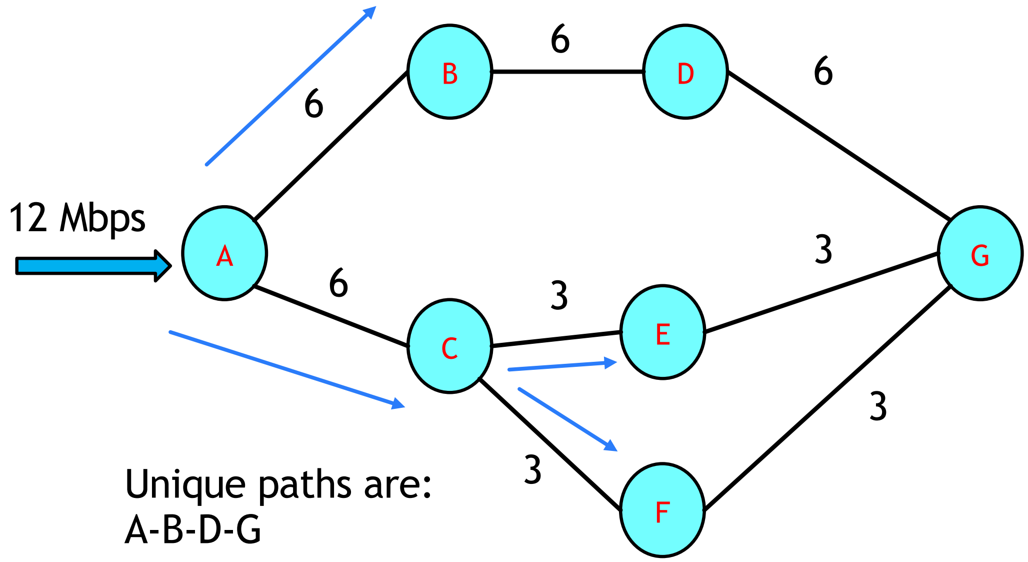
Networkx and pyNTM with ECMP traffic

- You can't model utilization from a demand with ECMP by splitting traffic evenly across all the unique equal-cost end-to-end paths
  - This would be end-to-end load balancing
- IGPs load balance hop-by-hop, not end to end
- Spreading the traffic evenly across the 3 unique end-to-end paths results in the traffic spread shown



Networkx and pyNTM with ECMP traffic (continued)

- pyNTM models ***hop-by-hop*** ECMP across the 3 unique paths
  - This is how OSPF and ISIS load balance
- Hop-by-hop load balancing results in the traffic spread shown
- This hop-by-hop spread is very different than the end-to-end load balancing traffic spread
- pyNTM models interface utilization from IGP (hop-by-hop) load balancing



Unique paths are:
A-B-D-G
A-C-E-G
A-C-F-G

Why is pyNTM helpful?

- PyNTM leverages path information from *networkx* in a *network state-specific context*, allowing for modeling of *network-specific* state:
  - Modeling utilization on interfaces
  - Modeling traffic consuming interface bandwidth
  - Modeling RSVP LSPs consuming reservable interface bandwidth
  - Determining the available path(s) that have a given amount of reservable bandwidth
  - IGP ECMP load-balancing splits
- pyNTM APIs allow for programmatic network modeling capability

Why is pyNTM helpful? (continued)

- pyNTM allows users to easily modify the network topology and determine alternate network state based on that change; for example:
  - Failing layer3 Nodes, Circuits, SRLGs, etc
  - Adding new nodes, interfaces, traffic demands, SRLGs to the topology
  - Adding new/additional auto-bandwidth LSPs to the topology
- pyNTM is specifically designed to easily relate objects in the model:
  - Traffic Demands
  - RSVP LSPs
  - End-to-end path info
  - Interfaces
  - Nodes

pyNTM Features and Roadmap

pyNTM features (as of v1.5)

- IGP (OSPF/ISIS) routing
- RSVP LSP Full Mesh
  - Traffic source and destination must match LSP source and destination
  - Auto-bandwidth LSPs
  - Fixed/manually-assigned setup bandwidth LSPs
- Shared Risk Link Groups (SRLGs)
- Currently supports modeling a single link between layer 3 nodes
  - For many use cases, it’s valid to combine multiple links with same cost between 2 nodes into a single link in a network model

Feature requests and pull requests are accepted on GitHub!

- Submit feature requests or pull requests at https://github.com/tim-fiola/network_traffic_modeler_py3/issues
- Current open feature requests include
  - IGP shortcuts
  - Multiple/parallel links between two layer 3 nodes
    - Modeling multiple/parallel links between nodes may incur a large performance cost
    - We have top people looking into that problem . . . *TOP* . . . *PEOPLE*
  - Assigning manual cost to RSVP LSP

Possible roadmap features

- It's helpful to have community input on these possible features and any others
  - Allowing only a % of interface capacity to be used for reservable bandwidth
  - Regional RSVP LSP meshes that stitch together at region boundaries
- Performance improvements and optimizations

Can pyNTM help you right now?

Options for modeling/simulation

Feature	Commercial Options	pyNTM
Cost	$$$	$0
APIs for programmatic modeling	**Y**	**Y**
Includes capability to create traffic matrix	Y	N
Sophisticated GUI for visualization	Y	N
Open Source	N	Y
Dependent on vendor	Y	N

Modeling with python should be a thing now!

- Python is a mature language
- Python is prevalent in the communications networking domain
- The need for network modeling is great since more and more networks are facing problems associated with scaling

Modeling with python should be a thing now! (continued)

- The capability for basic modeling in the commercial products is mature
  - They've been around for about 17-ish years
- Mature technologies can be modularized
  - One app to create a traffic matrix
  - One app to model using the traffic matrix

So who is pyNTM for?

- If your org/company can generate a reasonable traffic matrix
  - Access to data scientists
  - PMACCT and NetFlow
  - Forecasted traffic demands
- If your org has basic python coding skills
- If your org does not want to rely on and/or manage external modeling vendors
- If your WAN is IGP only
- If your WAN is IGP + full mesh RSVP

pyNTM provides the open source modeling and simulation engine and can help you today

- Otherwise, features are still being added!
  - I'm happy to talk and discuss features you need to model your network with pyNTM

Next steps

- Download pyNTM from PyPi via *pip3*
  - *pip3 install pyntm*
- Access the full repository on GitHub
  - https://github.com/tim-fiola/network_traffic_modeler_py3
  - Provides access to sample scripts
  - Provides access to beta features
    - Interactive visualization
    - Simple User Interface to help hu-mans explore model topology

Next Steps (continued)

- Access the **free** training modules
  - https://github.com/tim-fiola/network_traffic_modeler_py3/wiki
- Read the docs
  - Docstrings are real and are a thing
  - https://readthedocs.org/projects/pyntm/
- Contribute!
  - If you can enshrine network behavior in code or script useful workflows in pyNTM, please submit a pull request on GitHub!

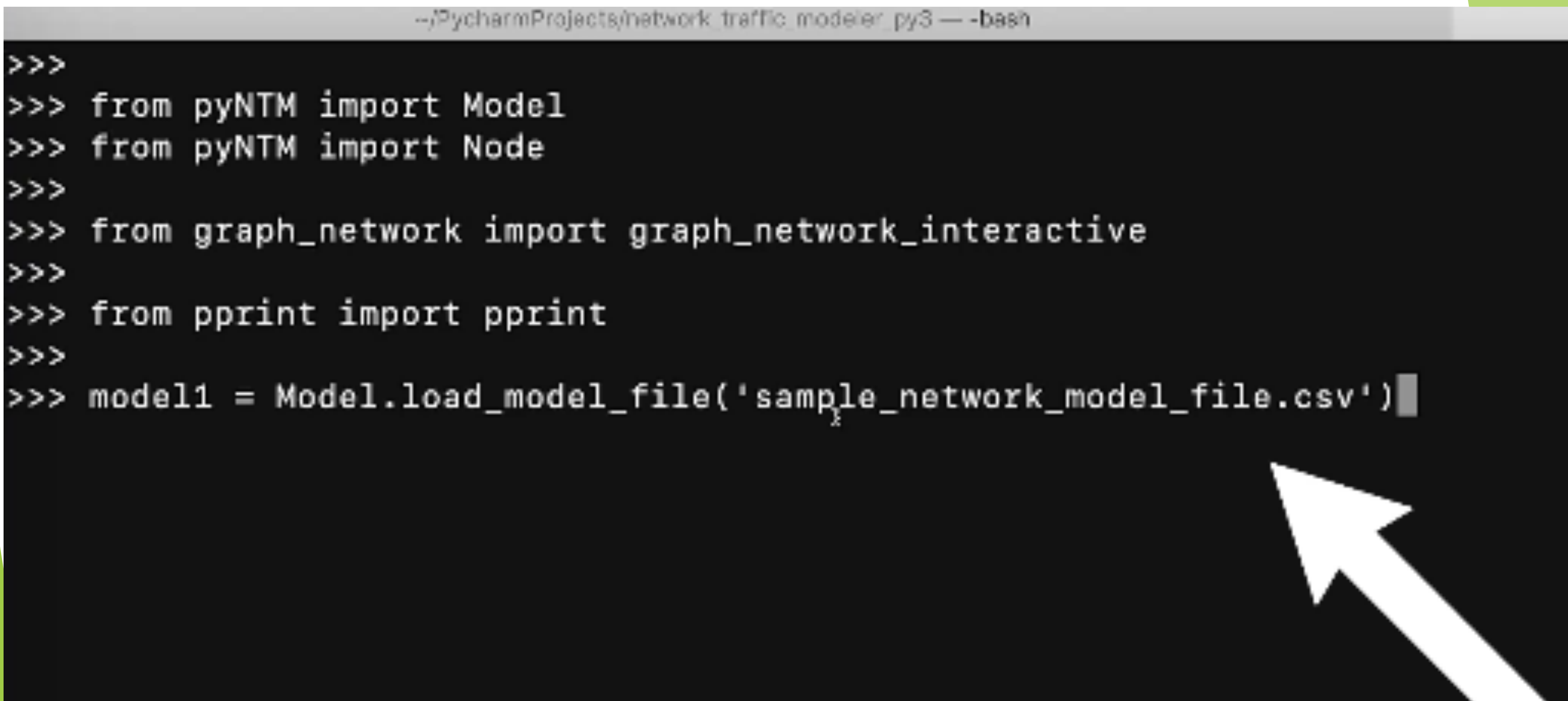
Notes about demos

- They always seem like a good idea, until they don’t
- In a room this size, a demo does NOT come through clearly
- SO . . .
  - We’re going to cowboy this a bit and show screenshots of the demo, not the demo recording
- The point of this is to show you that using the pyNTM APIs programmatically is a real thing

Demo Snapshots

- Load model file
- Look at interface utilization
- Visualize network (beta feature)
- Get shortest path between 2 layer 3 nodes
- Fail interface
- Visualize network
- Look at interface utilization during failure
- Get demands on an interface
- Get path info for an ECMP demand

Load Model File



Update simulation

```
>>>
>>> model1.update_simulation()
Routing the LSPs . . .
LSPs routed (if present); routing demands now . . .
Demands routed; validating model . . .
>>>
>>>
>>>
```

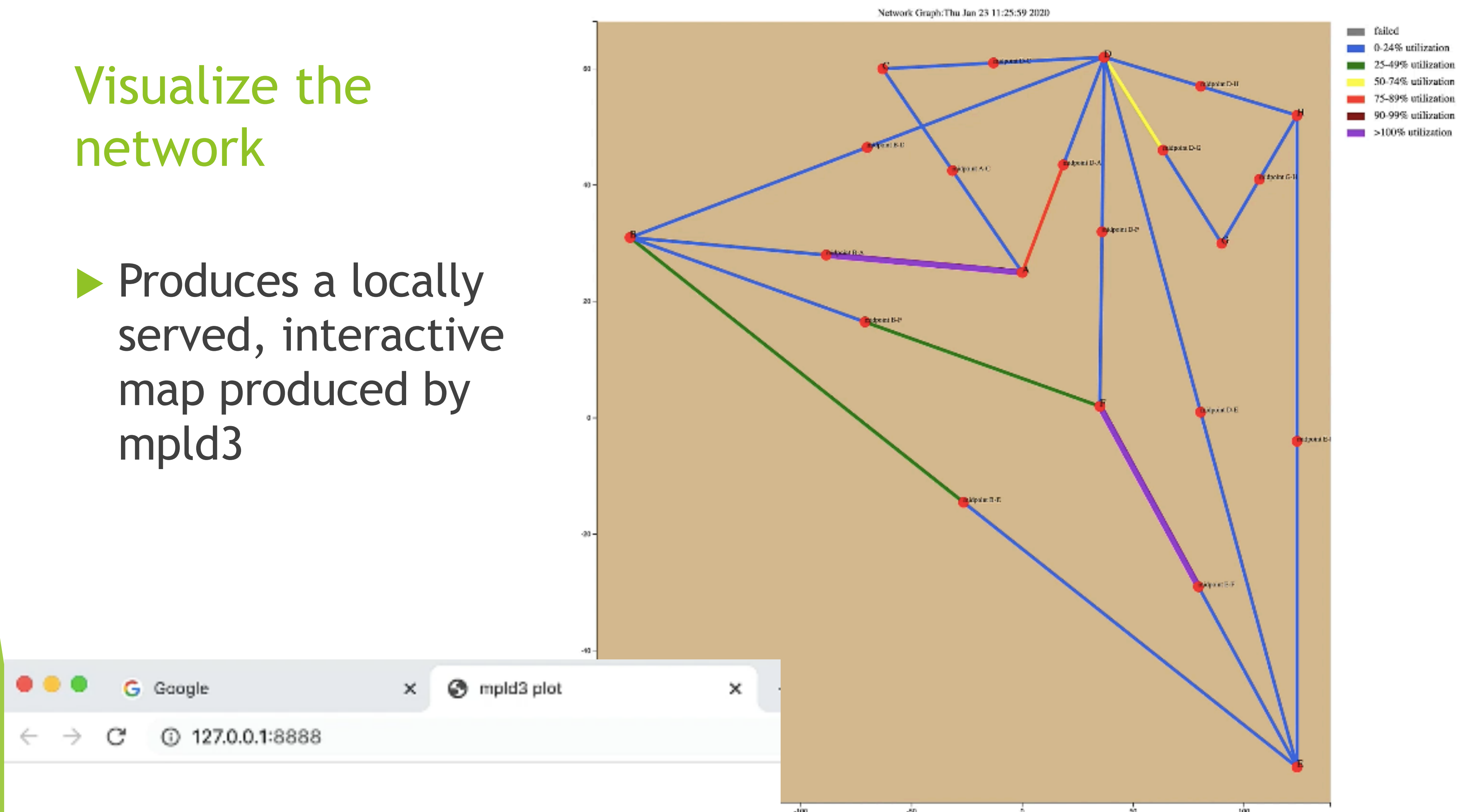
Visualize Network (beta feature)

```
>>> graph_network_interactive.make_interactive_network_graph(model1)
```

- The network visualization is a good tool to use in the training modules because it allows you to get a feel for the network and topology
- The network visualization is likely not practical for a larger network

Visualize the network

- Produces a locally served, interactive map produced by mpld3



Find shortest path(s) between 2 nodes

```
>>> sp_c_e = model1.get_shortest_path('C', 'E')
>>>
>>> pprint(sp_c_e)
{'cost': 8,
 'path': [[Interface(name = 'C-to-A', cost = 1, capacity = 200, node_object = Node('C'), rem
           Interface(name = 'A-to-B', cost = 4, capacity = 100, node_object = Node('A'), rem
           Interface(name = 'B-to-E', cost = 3, capacity = 200, node_object = Node('B'), rem
>>>
```

- Uses *get_shortest_path* method off of the Model object
- Returns a dictionary object with *cost* and *path* keys
- The *path* value is a list of lists of interfaces along the shortest path(s) from source to destination
  - Each shortest path would be a separate list in the path list

Look at interface utilization (only interfaces above 90%)

```
>>> for interface in model1.interface_objects:
...     if interface.utilization >= 90:
...         print(interface.name, interface.node_object.name, interface.utilization)
...
A-to-B A 136.0
F-to-E F 105.0
>>>
>>>
```

Fail an interface

```
>>> int_a_b = model1.get_interface_object('A-to-B', 'A')
>>>
>>>
>>> int_a_b
Interface(name = 'A-to-B', cost = 4, capacity = 100, node_object = Node('A'), remote_noc
>>>
>>> int_a_b.failed
False
>>>
>>> 
```

Fail an interface (continued)

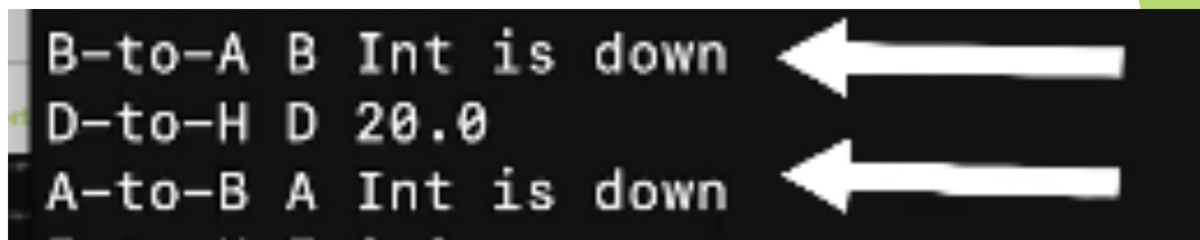
```
>>>
>>> int_a_b.fail_interface(model1)
>>>
>>>
>>> model1.update_simulation()
Routing the LSPs . . .
LSPs routed (if present); routing demands now . . .
Demands routed; validating model . . .
>>>
>>>
>>> 
```

Look at interface utilization during failure (all interfaces)

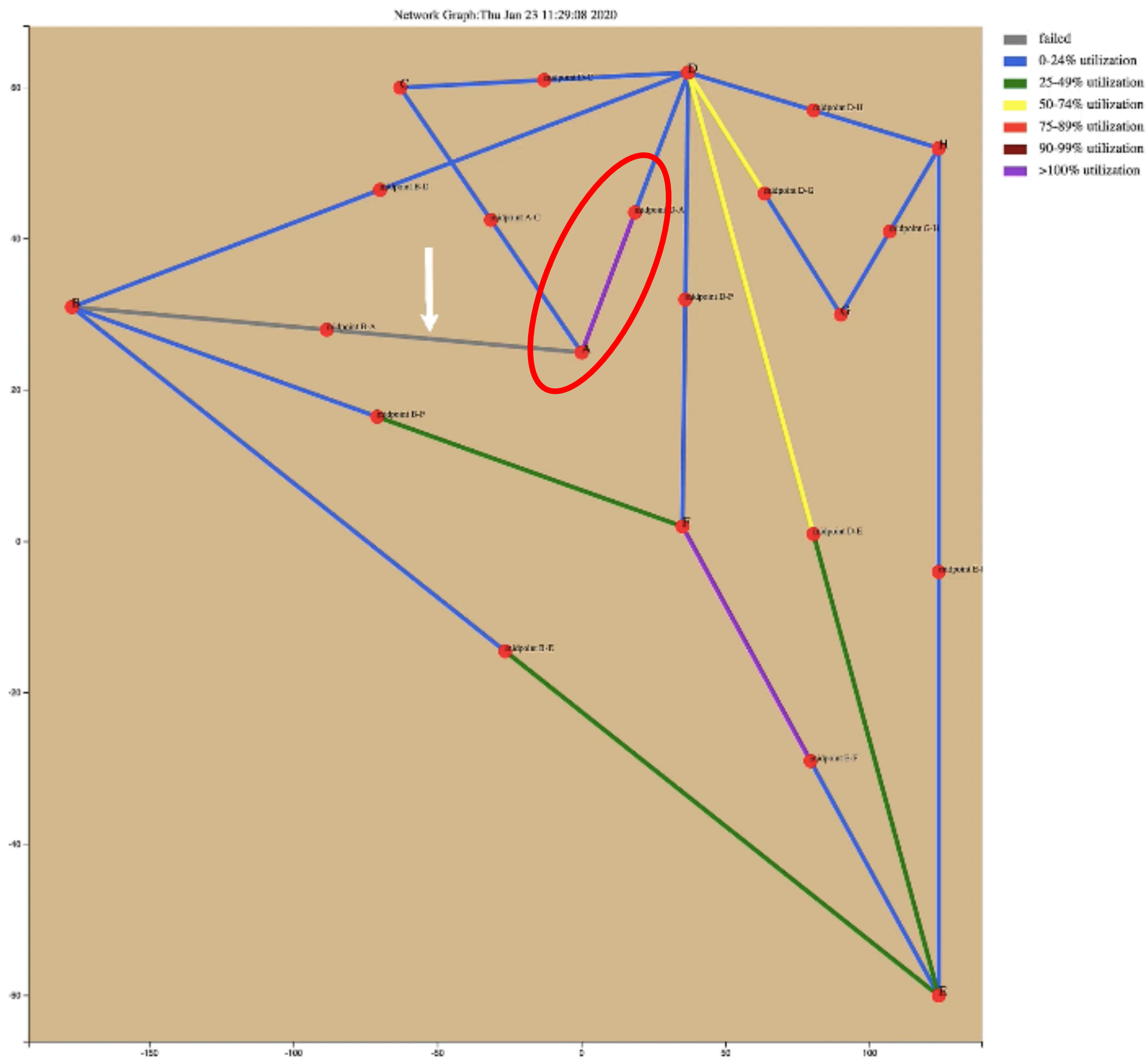
```
>>> for interface in model1.interface_objects:
...     print(interface.name, interface.node_object.name, interface.utilization)
...
B-to-D B 7.5
D-to-G D 60.0
E-to-D E 35.0
E-to-B E 25.0
B-to-A B Int is down
D-to-H D 20.0
A-to-B A Int is down
E-to-H E 0.0
F-to-D F 0.0
A-to-C A 5.0
D-to-F D 22.0
E-to-F E 0.0
H-to-G H 0.0
D-to-C D 6.666666666666667
H-to-D H 0.0
D-to-A D 13.333333333333334
D-to-E D 69.0
B-to-E B 7.5
C-to-A C 5.0
H-to-E H 0.0
F-to-B F 25.0
G-to-D G 0.0
B-to-F B 0.0
D-to-B D 12.5
C-to-D C 6.666666666666667
G-to-H G 0.0
A-to-D A 164.0
F-to-E F 105.0
>>>
```

Interface utilization highlights

- We failed interface from A to B
  - Interface B to A automatically entered failed state because failing one interface brings entire circuit down
- Interface from A to D is at 164% utilization



Visualize network with failure



Entire circuit between A and B is down

The interface from A to D shows purple (over 100% utilized

Get demands on interface . . . But first, docstrings!

~/PycharmProjects/network_traffic_modeler_py3 — -bash

```
Help on method demands in module pyNTM.interface:

demands(model) method of pyNTM.interface.Interface instance
    Returns list of demands that egress the interface
(END)
```

Get path info for a demand with equal cost multiple paths

```
>>> dmds_int_a_d = int_a_d.demands(model1)
>>>
>>> pprint(dmds_int_a_d)
[Demand(source = A, dest = E, traffic = 24, name = "''"),
 Demand(source = A, dest = H, traffic = 20, name = "''"),
 Demand(source = C, dest = E, traffic = 20, name = "''"),
 Demand(source = A, dest = B, traffic = 50, name = "''"),
 Demand(source = A, dest = F, traffic = 22, name = "''"),
 Demand(source = A, dest = D, traffic = 120, name = "''")]
>>>
```

These traffic demands are driving the utilization on that interface

Get paths for a demand with multiple, equal cost paths

```
>>> dmd_a_b = model1.get_demand_object('A', 'B', "''")
>>>
```

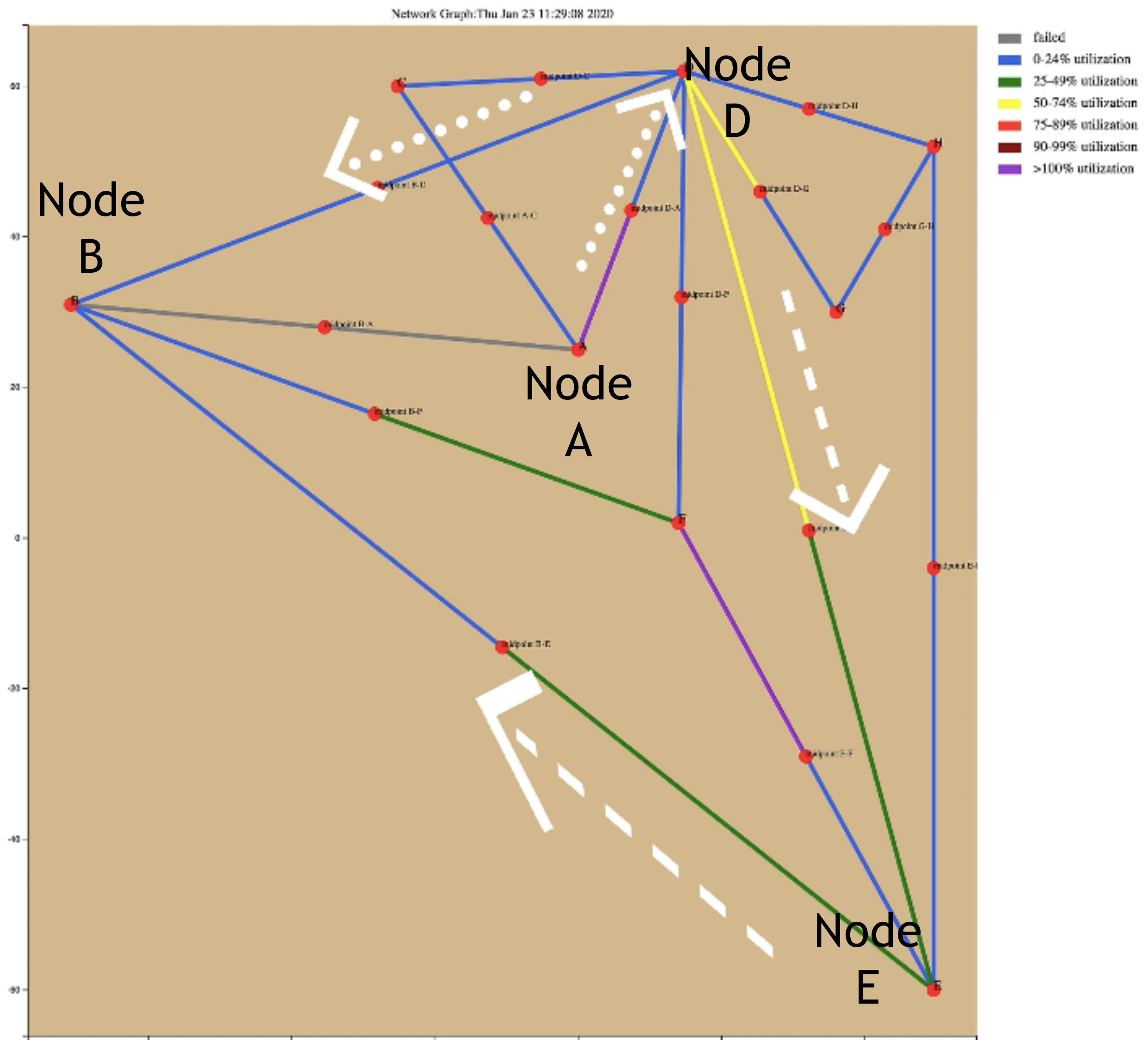
2 paths!

```
>>> for path in dmd_a_b.path:
...     pprint(path)
...     print()
...
[Interface(name = 'A-to-D', cost = 8, capacity = 150, node_object = Node('A'), rem
 Interface(name = 'D-to-B', cost = 7, capacity = 200, node_object = Node('D'), rem

[Interface(name = 'A-to-D', cost = 8, capacity = 150, node_object = Node('A'), rem
 Interface(name = 'D-to-E', cost = 4, capacity = 100, node_object = Node('D'), rem
 Interface(name = 'E-to-B', cost = 3, capacity = 200, node_object = Node('E'), rem

>>>
```

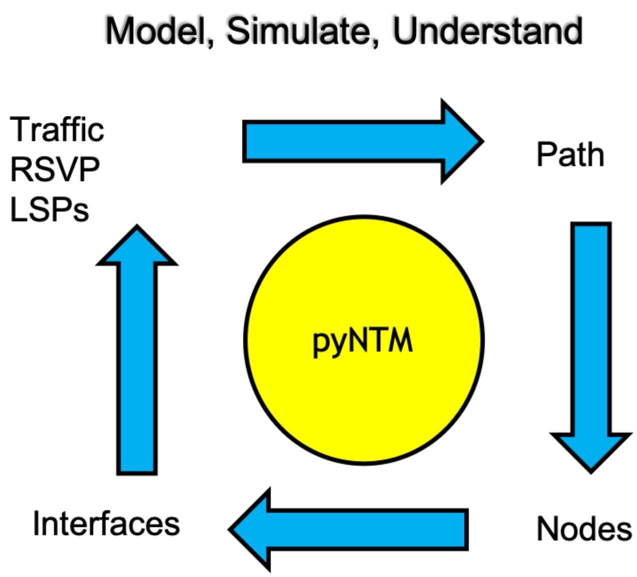
The *path* call returns a list of interfaces that the traffic demand egresses from source to destination

Visualization with path

- Paths for demand from A to B were
  - A→D, D→B
  - A→D, D→E, E→B



FIN


Model, Simulate, Understand
Traffic
RSVP
LSPs
Path
pyNTM
Interfaces
Nodes