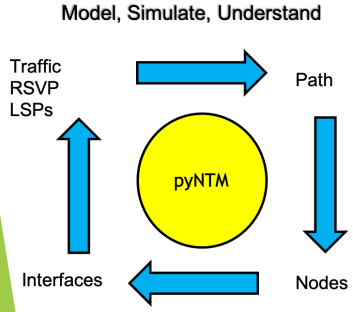
pyNTM Training Module 2 – getting started

Network Traffic Modeler in Python3

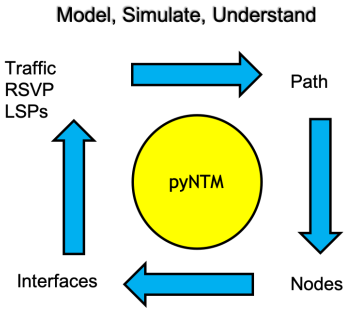


Versioning

- Training v1.0
- pyNTM v1.5

Course topics

- Use python3!
- What is pyNTM?
- Get pyNTM
- LIVE DEMO - setting up a virtual environment (if wanted/needed)
- About the Model file
- Setting up for the Exercises
- Converging the Model
- Visualization (beta)



Course topics (continued)

- Live Exercise 1: shortest path(s)
  - Examine IGP topology for the shortest path(s) between two Nodes
- Live Exercise 2: Failing an Interface
  - Failing an Interface and assessing impact on Interface utilization
- Live Exercise 3: Finding traffic Demands on an Interface
  - Determine which Demands are driving Interface utilization
- Live Exercise 4: Finding the path(s) of a specific Demand
  - Determine the ECMP path(s) a Demand takes through the IGP topology
- Live Exercise 5: Unfailing an Interface

What is pyNTM?

- pyNTM is the Network Traffic Modeler in python3
  - It is a modeling and simulation engine
  - It provides capability to define/modify a network topology
  - It provides capability to apply a traffic matrix to that topology to get simulation results
  - It does NOT provide a traffic matrix for your network
    - The user will have to create the traffic matrices for their network
    - Network modeling aside, in order to understand your network, it is imperative to understand the traffic matrix

Getting and using pyNTM

- Get pyNTM – 2 options
  - PyPI – PYthon Package Index
    - From OS CLI: *pip3 install pyNTM*
  - Download the repository from Github
    - https://github.com/tim-fiola/network_traffic_modeler_py3
    - pip3 installs current master branch
    - Dev branch has additional features under development
- Documentation
  - https://pyntm.readthedocs.io/en/latest/index.html
  - Docstrings are also available via help call for the def

```
>>> help(Model.update_simulation)

Help on function update_simulation in module pyNTM.model:

update_simulation(self)
    Updates the simulation state; this needs to be run any time there is
    a change to the state of the Model, such as failing an interface, adding
    a Demand, adding/removing and LSP, etc.

    This call does not carry forward any state from the previous simulation
    results.
```

About the model file

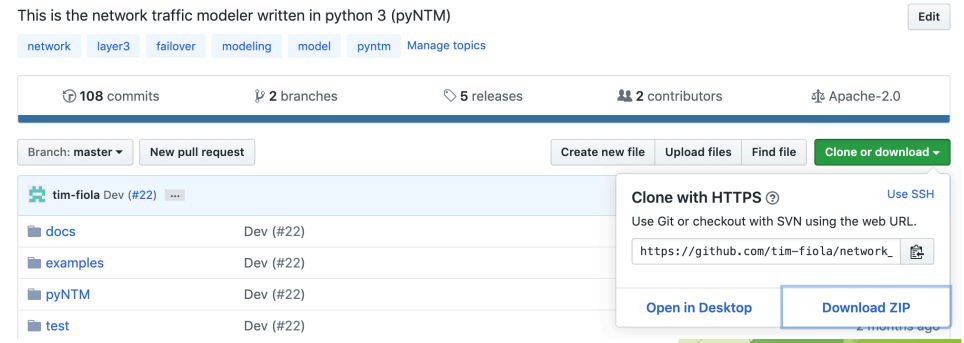
- Tab separated file that describes
  - Interfaces
  - Nodes
    - Nodes can be inferred from their interface objects
    - The NODES_TABLE is present in the event that
      - You want to add other attributes for a Node (latitude, longitude)
      - You want to load an *orphan node* to your model (a Node that has no interfaces)
  - Demands
  - RSVP LSPs (if applicable)
- Use the example model file as a template for making a model of your network

```
INTERFACES_TABLE
node_object_name        remote_node_object_name name    cost    capacity
A       B       A-to-B  4       100
A       C       A-to-C  1       200
A       D       A-to-D  8       150
B       A       B-to-A  4       100
B       D       B-to-D  7       200
B       E       B-to-E  3       200
D       B       D-to-B  7       200
D       C       D-to-C  9       150
D       A       D-to-A  8       150
D       E       D-to-E  4       100
D       F       D-to-F  3       100
C       A       C-to-A  1       200
C       D       C-to-D  9       150
E       B       E-to-B  3       200
E       D       E-to-D  4       100
F       D       F-to-D  3       100
F       B       F-to-B  6       100
B       F       B-to-F  6       100
G       H       G-to-H  4       150
H       G       H-to-G  4       150
G       D       G-to-D  2       50
D       G       D-to-G  2       50
H       E       H-to-E  4       100
E       H       E-to-H  4       100
E       F       E-to-F  3       100
F       E       F-to-E  3       100
H       D       H-to-D  4       100
D       H       D-to-H  4       100

NODES_TABLE
name    lon     lat
A       25      0
B       31      -177
C       60      -63
D       62      37
E       -60     124
F       2       35
G       30      90
H       52      124

DEMANDS_TABLE
source  dest    traffic name
A       B       50      ''
A       F       22      ''
A       E       24      ''
F       E       80      ''
F       B       50      ''
A       D       120     ''
D       A       10      ''
A       H       20      ''
C       E       20      ''
B       G       30      ''
E       C       20      ''
```

Exercise setup

Copy the repository zip file to a practice directory and unzip it

- Copying the repository will allow you to use some of the additional tools to improve your user experience
  - Visualization
  - Simple user interface




```
timfiola-mbp:modeling_practice timfiola$ unzip network_traffic_modeler_py3-master.zip
Archive:  network_traffic_modeler_py3-master.zip
09cce58c750621160bf7a82e0966f951503d4091
   creating: network_traffic_modeler_py3-master/
```

Set up your virtual environment (optional)

- Go into the archive directory
  - Look for *requirements.txt*
- Follow directions below to create the virtual environment
- Example is to the right →

A virtual environment provides an isolated environment and ensures no interference from existing installations and/or dependencies

```
timfiola-mbp:modeling_practice timfiola$ cd network_traffic_modeler_py3-master
timfiola-mbp:network_traffic_modeler_py3-master timfiola$ ls -lt
total 80
-rwxr-xr-x@  1 timfiola  935  11306 Nov 13 12:20 LICENSE
-rwxr-xr-x@  1 timfiola  935     25 Nov 13 12:20 Manifest.in
-rwxr-xr-x@  1 timfiola  935   1772 Nov 13 12:20 README.md
-rwxr-xr-x@  1 timfiola  935   3087 Nov 13 12:20 TODO.md
drwxr-xr-x@ 10 timfiola  935    320 Nov 13 12:20 docs
drwxr-xr-x@ 10 timfiola  935    320 Nov 13 12:20 examples
drwxr-xr-x@ 12 timfiola  935    384 Nov 13 12:20 pyNTM
-rwxr-xr-x@  1 timfiola  935     32 Nov 13 12:20 requirements.txt
-rwxr-xr-x@  1 timfiola  935     87 Nov 13 12:20 requirements_dev.txt
-rwxr-xr-x@  1 timfiola  935    344 Nov 13 12:20 setup.cfg
-rwxr-xr-x@  1 timfiola  935    927 Nov 13 12:20 setup.py
drwxr-xr-x@ 25 timfiola  935    800 Nov 13 12:20 test
timfiola-mbp:network_traffic_modeler_py3-master timfiola$
timfiola-mbp:network_traffic_modeler_py3-master timfiola$ virtualenv -p python3 venv
```

```
timfiola-mbp:network_traffic_modeler_py3-master timfiola$ source venv/bin/activate
(venv) timfiola-mbp:network_traffic_modeler_py3-master timfiola$
(venv) timfiola-mbp:network_traffic_modeler_py3-master timfiola$ pip install -r requirements.txt
Collecting networkx
```

Create your virtualenv

Create an isolated virtual environment under the directory "venv" with python3:

```
$ virtualenv -p python3 venv
```

Activate "venv" that sets up the required env variables:

```
$ source venv/bin/activate
```

Install required packages with "pip":

```
$ pip install -r requirements.txt
```

Let's get started!

- Switch to the *examples* directory in the repository
- Start python3
- Append parent directory to your sys path
  - Allows imports from folders in the parent
- Import the Model object
- Load Model from data file
  - *sample_network_model_file.csv* has Interfaces, Nodes, and Demands
  - IGP only
  - no RSVP LSPs in the file
- Observe node objects

```
>>> import sys
>>> sys.path.append('../')
```

```
>>> from pyNTM import Model
>>>
>>> model1 = Model.load_model_file('sample_network_model_file.csv')
```

```
>>> model1
Model(Interfaces: 28, Nodes: 8, Demands: 11, RSVP_LSPs: 0)
>>>
>>> model1.node_objects
{Node('E'), Node('H'), Node('B'), Node('C'), Node('A'), Node('D'), Node('F'), Node('G
')}
>>>
```

Converging the model

- In order to route the traffic across the network topology to get modeling data, the Model must be explicitly converged
- A Model should be converged after it is loaded from a file
- A Model should be re-converged if you make any change to the topology
  - Failing an Interface or Node
  - Un-failing an Interface or Node
  - New Node, Interface, traffic Demand, RSVP LSP, SRLG etc
  - Any change, otherwise, to the topology or traffic matrix
- Use the *update_simulation()* call on the Model object to converge the model
  - This will converge the model and run some internal validation checks

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

Converging the model (continued)

- Failing to run update_simulation() is likely to result in incorrect modeling and/or Model exceptions
- The time it takes to converge the model depends on
  - How many RSVP LSPs are present
  - Size of topology
  - Amount of entries in traffic matrix
  - The amount of processing/memory available on the machine
- It may take a couple minutes or more for the model to converge for very large model files

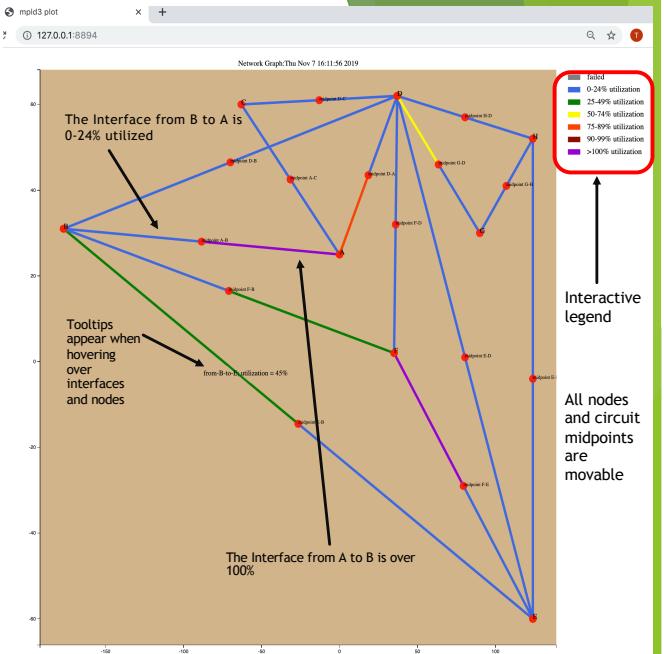
Visualization (beta) - optional

- Requires full repository download from GitHub or extract the module
  - Easy access via the virtual environment setup from earlier in this guide
  - Reference the *Exercise setup* earlier in this presentation for full instructions to set up the environment to use the repository
- Make sure the model is converged!
  - *model1.update_simulation()* ← *model1* is the Model object
- *graph_network_interactive.make_interactive_network_graph* call
  - Takes Model object as argument
  - uses mpld3 python package under the covers
- Produces interactive graph in browser with tool tips, an interactive legend, and draggable Nodes and Interface endpoints for easier viewing
- Uses a Node's lat/lon (y,x) attributes to position Node on layout

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

```
>>> from graph_network import graph_network_interactive
```

```
>>> graph_network_interactive.make_interactive_network_graph(model1)
>>> Serving to http://127.0.0.1:8891/    [Ctrl-C to exit]
127.0.0.1 - - [07/Nov/2019 15:28:48] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [07/Nov/2019 15:28:48] "GET /d3.js HTTP/1.1" 200 -
127.0.0.1 - - [07/Nov/2019 15:28:48] "GET /mpld3.js HTTP/1.1" 200 -
127.0.0.1 - - [07/Nov/2019 15:28:48] code 404, message Not Found
127.0.0.1 - - [07/Nov/2019 15:28:48] "GET /favicon.ico HTTP/1.1" 404 -

>>>
```



Live Exercise 1: Shortest path(s)

- Observe shortest path(s)
  - If there are multiple shortest paths, the ‘path’ value list will have multiple lists, each one a unique path
  - Each path list in the ‘path’ value list is an (ordered) list of Interfaces from source to destination
- In the example below, the shortest path from Node C to Node E
  - has a total IGP cost of 8
  - has a single shortest path
  - egresses 3 Interfaces, in order, from source to destination:
    - Node C to Node A
    - Node A to Node B
    - Node B to Node E

```
>>> from pprint import pprint
>>> 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'),
 remote_node_object = Node('A'), address = 14),
           Interface(name = 'A-to-B', cost = 4, capacity = 100, node_object = Node('A'),
 remote_node_object = Node('B'), address = 2),
           Interface(name = 'B-to-E', cost = 3, capacity = 200, node_object = Node('B'),
 remote_node_object = Node('E'), address = 3)]]}
>>>
```

A quick look at interface utilization

- Interface objects have several attributes and methods
- Quickly find the utilization of each interface with a simple *for* loop!
  - Interface.name
    - Name of the Interface
  - Interface.node_object.name
    - Name of the Node object that the Interface resides on
  - Interface.utilization
    - % traffic utilization
  - Easily add qualifiers to find Interfaces with specific qualifiers
    - For example, Interfaces with utilization 90% or above:

```
>>> 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
```

```
>>> from pyNTM import Interface
>>> dir(Interface)
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_key', 'add_to_srlg', 'capacity', 'cost', 'demands', 'fail_interface', 'failed', 'get_circuit_object', 'get_remote_interface', 'lsps', 'remove_from_srlg', 'reservable_bandwidth', 'srlgs', 'unfail_interface', 'utilization']
>>>
```

```
>>> for interface in model1.interface_objects:
...     print(interface.name, interface.node_object.name, interface.utilization)
...
E-to-D E 15.0
G-to-D G 0.0
A-to-D A 80.0
B-to-E B 45.0
H-to-E H 0.0
B-to-D B 7.5
E-to-H E 20.0
F-to-B F 25.0
H-to-D H 0.0
B-to-F B 11.0
D-to-C D 0.0
F-to-E F 105.0
E-to-B E 22.5
B-to-A B 20.0
D-to-G D 60.0
A-to-C A 10.0
A-to-B A 136.0
G-to-H G 0.0
D-to-B D 0.0
E-to-F E 11.0
D-to-A D 6.666666666666667
F-to-D F 0.0
D-to-H D 0.0
C-to-D C 0.0
H-to-G H 0.0
D-to-E D 0.0
C-to-A C 10.0
D-to-F D 0.0
>>>
```

Live Exercise 2: Failing an Interface/Circuit

- Circuit objects have two member Interface objects
  - One of the Interfaces fails, the Circuit and the remote Interface objects also fail
- Step 1: get Interface object
  - There are a few ways to do this
  - Technique 1: get interface via *get_interface_object* Model call
    - Need to know Node name and Interface name

```
>>> help(model1.get_interface_object)
```

```
Help on method get_interface_object in module pyNTM.model:

get_interface_object(interface_name, node_name) method of pyNTM.model.Model instance
    Returns an interface object for specified node name and interface name
(END)
```

```
>>> 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_nod
e_object = Node('B'), address = 12)
```

Step 1 (continued) – get Interface Object (technique 2)

- Get an interface object from a Node's interface list
  - Only need to know Node name
- Get the Node object
- List the Node's Interfaces
- Select the Interface from the list

```
>>> node_a = model1.get_node_object('A')
>>>
>>> node_a.interfaces(model1)
[Interface(name = 'A-to-D', cost = 8, capacity = 150, node_object = Node('A'), remote_no
de_object = Node('D'), address = 10), Interface(name = 'A-to-B', cost = 4, capacity = 10
0, node_object = Node('A'), remote_node_object = Node('B'), address = 2), Interface(name
 = 'A-to-C', cost = 1, capacity = 200, node_object = Node('A'), remote_node_object = Nod
e('C'), address = 11)]
>>>
>>> int_a_b_via_node_a = node_a.interfaces(model1)[0]
>>>
>>> int_a_b_via_node_a
Interface(name = 'A-to-D', cost = 8, capacity = 150, node_object = Node('A'), remote_nod
e_object = Node('D'), address = 10)
>>>
```

Live Exercise 2: Failing an Interface/Circuit (continued)

- Step 2: fail the interface object
  - Uses Interface *fail_interface* method

```
>>> int_a_b.fail_interface(model1)
>>>
```

- Step 3: update the simulation

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

- Step 4: check interface utilization

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

Live Exercise 3: Finding traffic demands on an interface

- Looking at the interface utilizations from Exercise 2, the interface from Node A to Node D shows 164% utilized
- Let's see which Demands (traffic) is driving that utilization
- Use the Interface *demands* method

```
>>> int_a_d = model1.get_interface_object('A-to-D', 'A')
>>>
>>> dir(int_a_d)
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__form
at__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_s
ubclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclas
shook__', '__weakref__', '_capacity', '_cost', '_failed', '_key', '_srlgs', 'add
_to_srlg', 'address', 'capacity', 'cost', 'demands', 'fail_interface', 'failed',
 'get_circuit_object', 'get_remote_interface', 'in_ckt', 'lsps', 'name', 'node_o
bject', 'remote_node_object', 'remove_from_srlg', 'reservable_bandwidth', 'reser
ved_bandwidth', 'srlgs', 'traffic', 'unfail_interface', 'utilization']
>>>
>>> help(int_a_d.demands)
```

```
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)
```

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

Live Exercise 3: Finding traffic demands on an interface (continued)

- There are 6 demands on the Interface

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

Live Exercise 4: Finding Demand path(s) in an IGP network

- From our Demand results in Exercise 3, find the path of *Demand(source = A, dest = B, traffic = 50, name = """)*
  - Keep in mind that the Circuit between Nodes A and B is failed (from Exercise 2)
- Get the demand object
  - Either by getting item 2 from *dmds_int_a_d* (list) (from Exercise 3)
    - The specific index for this demand will vary; check your own results

```
>>> dmd_a_b = dmds_int_a_d[2]
>>> dmd_a_b
Demand(source = A, dest = B, traffic = 50, name = "'")
>>>
```

  - Or using the Model *get_demand_object* method

```
Help on method get_demand_object in module pyNTM.model:

get_demand_object(source_node_name, dest_node_name, demand_name='none') method o
f pyNTM.model.Model instance
    Returns demand specified by the source_node_name, dest_node_name, name;
    throws exception if demand not found
(END)
```

```
>>> dmd_a_b = model1.get_demand_object('A', 'B', "'")
>>> dmd_a_b
Demand(source = A, dest = B, traffic = 50, name = "'")
>>>
```

Live Exercise 4: Finding Demand path(s) in an IGP network (continued)

- The demand has 2 ECMP paths
- The Demand path call returns a list of lists
  - Each component list shows the Interfaces the Demand egresses, in order, from the source Node to the Destination Node
- Keep in mind that we previously failed the Circuit between Nodes A and B

```
>>> int_a_b.failed
True
```

```
>>> dmd_a_b
Demand(source = A, dest = B, traffic = 50, name = "'")
>>>
>>> dir(dmd_a_b)
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__form
at__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_s
ubclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclas
shook__', '__weakref__', '_add_demand_path', '_key', 'dest_node_object', 'name',
 'path', 'source_node_object', 'traffic']
>>>
```

```
>>> len(dmd_a_b.path)
2
>>> for path in dmd_a_b.path:
...     pprint(path)
...     print()
...
[Interface(name = 'A-to-D', cost = 8, capacity = 150, node_object = Node('A'),
remote_node_object = Node('D'), address = 7),
 Interface(name = 'D-to-B', cost = 7, capacity = 200, node_object = Node('D'),
remote_node_object = Node('B'), address = 8)]

[Interface(name = 'A-to-D', cost = 8, capacity = 150, node_object = Node('A'),
remote_node_object = Node('D'), address = 7),
 Interface(name = 'D-to-E', cost = 4, capacity = 100, node_object = Node('D'),
remote_node_object = Node('E'), address = 9),
 Interface(name = 'E-to-B', cost = 3, capacity = 200, node_object = Node('E'),
remote_node_object = Node('B'), address = 11)]

>>>
```

Live Exercise 5: Unfailing an Interface

- Now that our failure analysis is complete, unfail the Interface on Node A facing Node B
  - The Interface on Node B facing Node A also unfails automatically

```
>>> int_a_b
Interface(name = 'A-to-B', cost = 4, capacity = 100, node_object = Node('A'),
remote_node_object = Node('B'), address = 12)
>>>
>>> int_a_b.failed
True
>>>
>>> int_b_a
Interface(name = 'B-to-A', cost = 4, capacity = 100, node_object = Node('B'),
remote_node_object = Node('A'), address = 12)
>>>
>>> int_b_a.failed
True
```

```
>>> int_a_b.unfail_interface(model1)
>>>
>>> int_a_b.failed
False
>>> int_b_a.failed
False
>>>
>>> model1.update_simulation()
Routing the LSPs . . .
LSPs routed; routing demands now . . .
Demands routed; validating model . . .
>>>
```

Simple User Interface tool (beta)

The Simple UI is designed to allow a user to gain insight about the network topology and how traffic transits the network

The Simple UI supports the following

Designed to allow non-coders to explore aspects of IGP topology; use cases include:

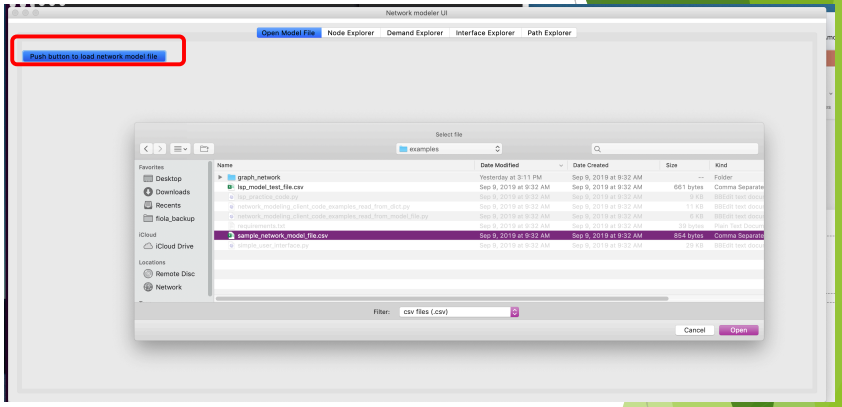
- Easily see which Demands source from a specific Node
- Select a Demand and see its path(s)
- Select an Interface from a path
- See Demands that egress an Interface
- Select Interfaces above a specific utilization

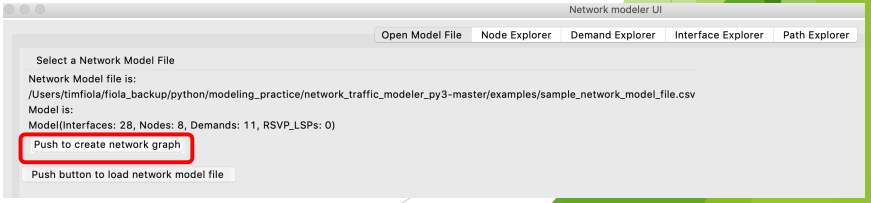
Limitations

- Does not support MPLS RSVP LSPs
- Does *not* support modifying the topology (failing/adding Interfaces, Nodes, etc)

Simple UI – getting started

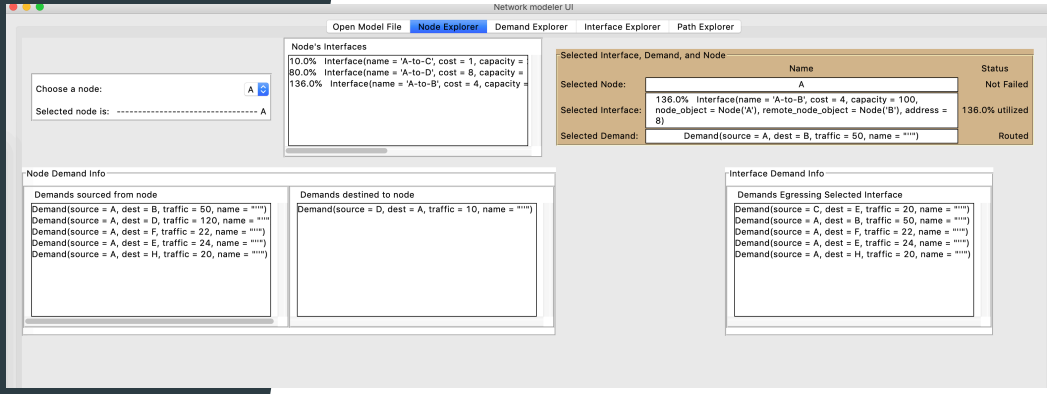
- In examples folder in repository:
  - *python3 simple_user_interface.py*
- Push button in upper right corner to open the model file
  - Select *sample_network_model_file.csv*
- Create an interactive network graph in a browser by pressing the button





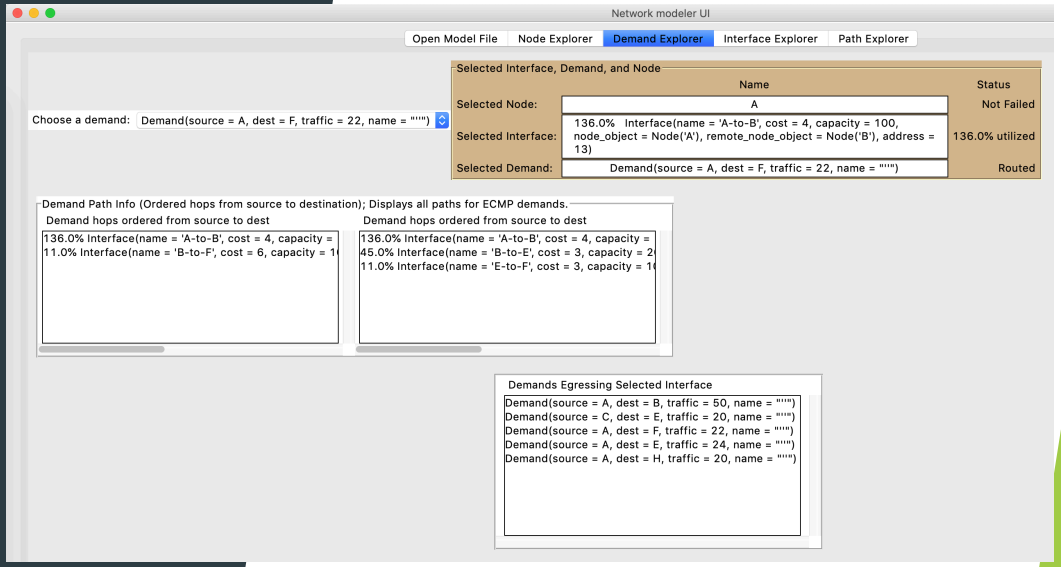
Node Explorer

- Select a Node
- View/select Node's Interfaces and Interface utilization
- View/select Demands (traffic) sourced from and destined to Node



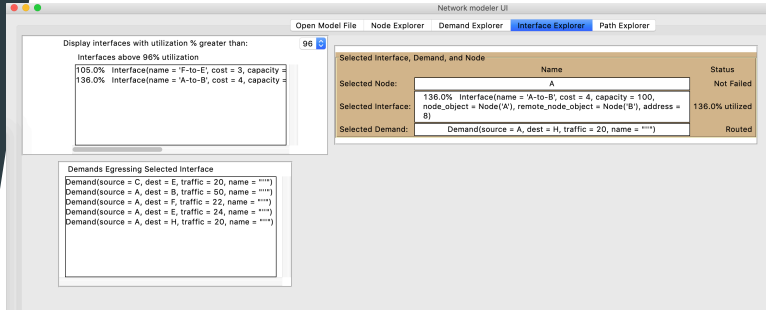
Demand Explorer

- Select a Demand object
- View/select Interfaces that Demand egresses from source to destination
- Displays multiple paths for a demand if multiple paths exist



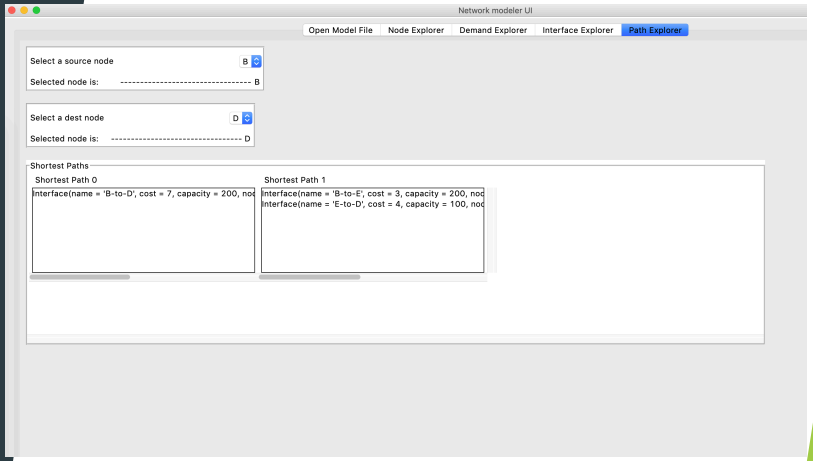
Interface Explorer

- Find/Select Interfaces with utilization greater than a user-defined percentage
- Find Demands (traffic) egressing the selected Interface



Path Explorer

- Select a source and destination Node
- Find all shortest paths between the 2 Nodes
- View/Select the Interfaces in any shortest path



FIN

BACKUP SLIDES

LIVE DEMO – virtual environment

- Benefits of virtual environment
  - Isolated environment to work in
  - Allows easy access to pyNTM beta features if you download the archive from github
    - Simple user interface
    - Network visualization
- https://pyntm.readthedocs.io/en/latest/install.html#virtualenv
- Download the archive from github
  - Download the zip file from https://github.com/tim-fiola/network_traffic_modeler_py3
  - Unzip the archive
  - In the archive, cd to directory that contains *requirements.txt* (*network_traffic_modeler_py3-master* folder)
  - Follow steps to the right to start venv→ → →
- You will now have a (venv) in front of your prompt
- To get easy access to the beta features
  - Switch to the *examples* directory in the archive
  - Start python3
  - Append the parent directory to sys.path

```
timfiola-mbp:network_traffic_modeler_py3-master timfiola$ ls -lt
total 80
-rw-r--r--@  1 timfiola  935  11306 Sep  9 09:32 LICENSE
-rw-r--r--@  1 timfiola  935     25 Sep  9 09:32 Manifest.in
-rw-r--r--@  1 timfiola  935   1772 Sep  9 09:32 README.md
-rw-r--r--@  1 timfiola  935   3087 Sep  9 09:32 TODO.md
drwxr-xr-x@ 10 timfiola  935    320 Sep  9 09:32 docs
drwxr-xr-x@ 10 timfiola  935    320 Sep  9 09:32 examples
drwxr-xr-x@ 12 timfiola  935    384 Sep  9 09:32 pyNTM
-rw-r--r--@  1 timfiola  935      8 Sep  9 09:32 requirements.txt
-rw-r--r--@  1 timfiola  935     87 Sep  9 09:32 requirements_dev.txt
-rw-r--r--@  1 timfiola  935    330 Sep  9 09:32 setup.cfg
-rw-r--r--@  1 timfiola  935    918 Sep  9 09:32 setup.py
drwxr-xr-x@ 25 timfiola  935    800 Sep  9 09:32 test
timfiola-mbp:network_traffic_modeler_py3-master timfiola$
```

Create your virtualenv

Create an isolated virtual environment under the directory "venv" with python3:

```
$ virtualenv -p python3 venv
```

Activate "venv" that sets up the required env variables:

```
$ source venv/bin/activate
```

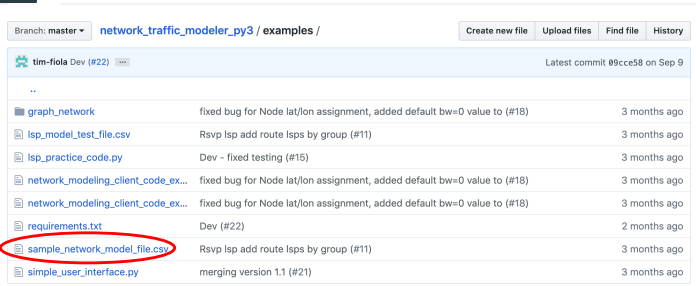
Install required packages with "pip":

```
$ pip install -r requirements.txt
```

```
/Users/timfiola/fiola_backup/python/modeling_practice/network_traffic_modeler_py3-master/examples
(venv) timfiola-mbp:examples timfiola$ python3
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.path.append('../')
>>>
```

Get sample model file

- Go to https://github.com/tim-fiola/network_traffic_modeler_py3/tree/master/examples
- Retrieve the ***sample_network_model_file.csv*** file and put it in the local directory you want to work in
  - wget https://raw.githubusercontent.com/tim-fiola/network_traffic_modeler_py3/master/examples/sample_network_model_file.csv
  - *OR* Download entire repository




```
timfiola-mbp:modeling_practice timfiola$ wget https://raw.githubusercontent.com/tim-fiola/network_traffic_modeler_py3/master/examples/sample_network_model_file.csv
--2019-11-07 12:05:37--  https://raw.githubusercontent.com/tim-fiola/network_traffic_modeler_py3/master/examples/sample_network_model_file.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.192.133, 151.101.0.133, 151.101.64.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.192.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 854 [text/plain]
Saving to: 'sample_network_model_file.csv'

sample_network_model_file.csv  100%[=========================================>]     854  --.-KB/s    in 0s

2019-11-07 12:05:37 (20.9 MB/s) - 'sample_network_model_file.csv' saved [854/854]

timfiola-mbp:modeling_practice timfiola$
timfiola-mbp:modeling_practice timfiola$
timfiola-mbp:modeling_practice timfiola$ vi sample_network_model_file.csv
timfiola-mbp:modeling_practice timfiola$
```