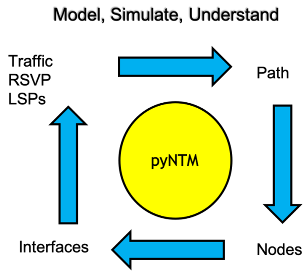
pyNTM Training Module 3 – modifying the Model and *what if* simulations



Network Traffic Modeler in Python3

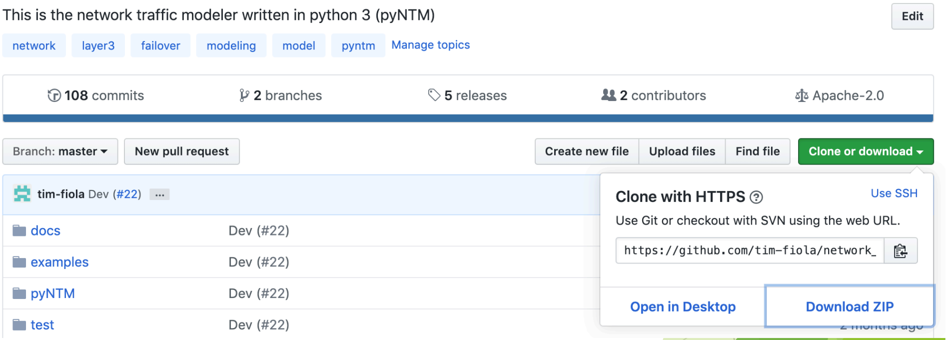
Course Topics

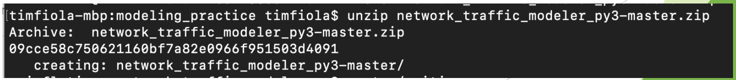
- Adding a new Node
- Adding a new link
- Adding traffic to the traffic matrix
- Changing Interface/Circuit capacity
- Changing an Interface metric
- Working with RSVP LSPs

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





Set up your virtual environment (optional)

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

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

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
```

```
[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
```

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
')}
>>>
```

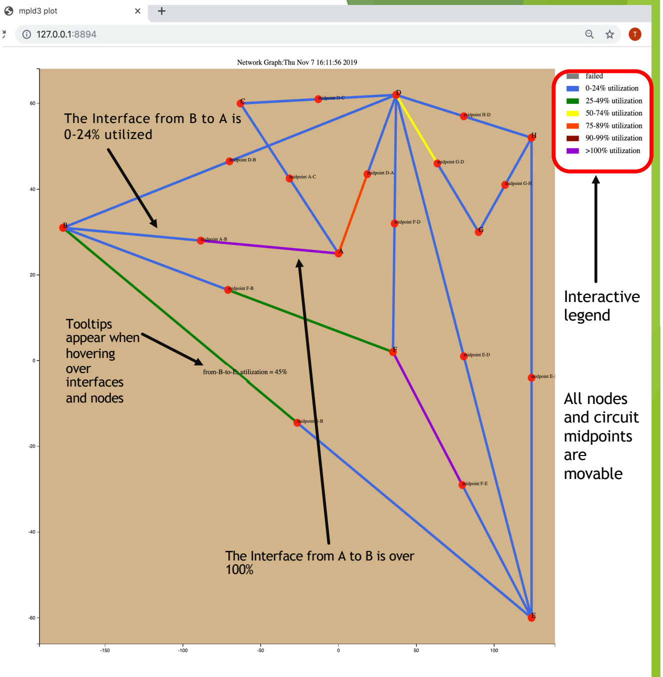
Visualization (beta) - optional

- 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 -

>>>
```



Add a new Node

- Import the Node object
- Define Node('Z')
- Assign latitude/longitude (y, x) coordinates
- Add Node('Z') to the Model
- Examine the Model
  - Now there are 9 Nodes
- An *orphan* Node has no interfaces

```
>>> from pyNTM import Node
>>>
>>> node_z = Node('Z')
>>>
>>> node_z.lat = 40
>>>
>>> node_z.lon = 50
>>>
>>> model1.add_node(node_z)
>>>
>>> model1
Model(Interfaces: 28, Nodes: 9, Demands: 11, RSVP_LSPs: 0)
>>>
>>> model1.get_orphan_node_objects()
[Node('Z')]
>>>
```

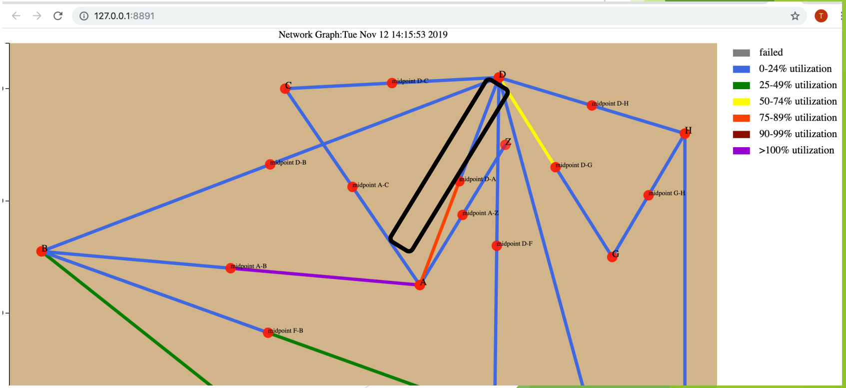
Add a new Circuit between Node('A') and Node('Z')

- Use the Model add_circuit call
  - New circuit will have a metric of 10 on both sides and a capacity of 200
- Update the simulation
- Visualize topology with new Node('Z') (optional)

```
>>> help(model1.add_circuit)

>>> model1.add_circuit(Node('A'), Node('Z'), 'a-to-z', 'z-to-a', 10, 10, 200)
>>>
>>> model1.update_simulation()
Routing the LSPs . . .
LSPs routed; routing demands now . . .
Demands routed; validating model . . .
>>>
```

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

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

add_circuit(node_a_object, node_b_object, node_a_interface_name, node_b_inte
rface_name, cost_intf_a=1, cost_intf_b=1, capacity=1000, failed=False, addre
ss=None) method of pyNTM.model.Model instance
    Creates component Interface objects for a new Circuit in the Model.
    The Circuit object will then be created during the validate_model() call
.

    :param node_a_object: Node object
    :param node_b_object: Node object
    :param node_a_interface_name: name of component Interface on node_a
    :param node_b_interface_name: name of component Interface on node_b
    :param cost_intf_a: metric/cost of node_a_interface component Interface
    :param cost_intf_b: metric/cost of node_b_interface component Interface
    :param capacity: Circuit's capacity
    :param failed: Should the Circuit be created in a Failed state?
    :param address: Optional.  Will be auto-assigned unless specified
    :return: Model with new Circuit comprised of 2 new Interfaces
~
```

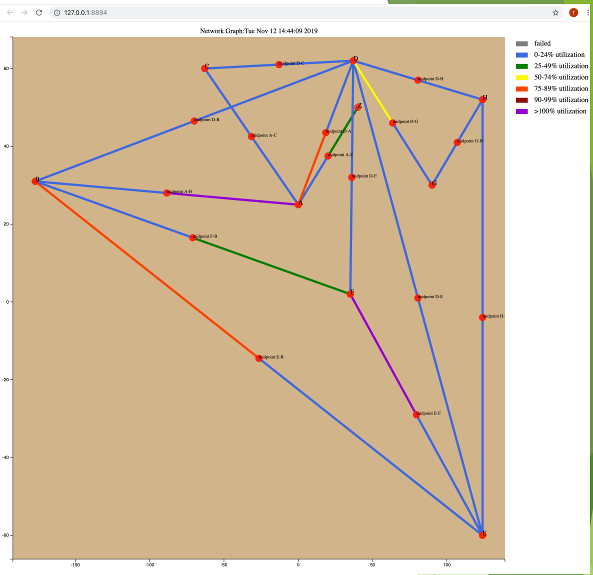


Adding a Demand to the traffic matrix

- Use the Model *add_demand* method
- Find new demand's path

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

add_demand(source_node_name, dest_node_name, traffic=0, name='none') method
 of pyNTM.model.Model instance
    Adds a traffic load (Demand) from point A to point B in the
    model and validates model.
    :param source_node_name: name of Demand's source Node
    :param dest_node_name: name of Demand's destination Node
    :param traffic: amount of traffic (magnitude) of the Demand
    :param name: Demand name
    :return: A validated Model object with the new demand
~
```

```
>>> help(model1.add_demand)

>>>
>>> model1.add_demand('Z', 'E', 75, 'z-to-e-initial')
>>>
>>> model1.update_simulation()
Routing the LSPs . . .
LSPs routed; routing demands now . . .
Demands routed; validating model . . .
>>>
```




```
>>> dmd_z_e = model1.get_demand_object('Z', 'E', 'z-to-e-initial')
>>> from pprint import pprint
>>> pprint(dmd_z_e.path)
[[Interface(name = 'z-to-a', cost = 10, capacity = 200, node_object = Node('Z'), remote_node_object = Node('A'),
 address = 3),
  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 = 10)]]
>>>
```

Changing Circuit Capacity

- Change the capacity of the Circuit between Node('A') and Node('B') from 100 to 200
- Change *capacity* attribute of each Interface in the Circuit
  - *circuits_with_mismatched_interface_capacity*
  - Capacities must match or Model will throw a *ModelException*
- Be sure to update the simulation after the change!
  - model1.update_simulation()

```
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)
```

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

>>> int_a_b = model1.get_interface_object('A-to-B', 'A')
>>>
>>> int_b_a = int_a_b.get_remote_interface(model1)
>>>
>>> int_a_b
Interface(name = 'A-to-B', cost = 4, capacity = 100, node_object = Node('A'), remote_node_object = Node('B'), ad
dress = 2)
>>>
>>> int_b_a
Interface(name = 'B-to-A', cost = 4, capacity = 100, node_object = Node('B'), remote_node_object = Node('A'), ad
dress = 2)
```

```
>>> int_a_b.capacity = 200
>>>
>>> model1.update_simulation()
Routing the LSPs . . .
LSPs routed; routing demands now . . .
Demands routed; validating model . . .
'network interface validation failed, see returned data'
[{'circuits_with_mismatched_interface_capacity': [Circuit(Interface(name = 'A-to-B', cost = 4, capacity = 200, n
ode_object = Node('A'), remote_node_object = Node('B'), address = 2), Interface(name = 'B-to-A', cost = 4, capac
ity = 100, node_object = Node('B'), remote_node_object = Node('A'), address = 2))]}]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/pyNTM/model.py", line 618,
 in update_simulation
    self.validate_model()
  File "/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/pyNTM/model.py", line 184,
 in validate_model
    raise ModelException((message, error_data))
pyNTM.exceptions.ModelException: ('network interface validation failed, see returned data', [{'circuits_with_mis
matched_interface_capacity': [Circuit(Interface(name = 'A-to-B', cost = 4, capacity = 200, node_object = Node('A
'), remote_node_object = Node('B'), address = 2), Interface(name = 'B-to-A', cost = 4, capacity = 100, node_obje
ct = Node('B'), remote_node_object = Node('A'), address = 2))]}])
>>>
>>> int_b_a.capacity = 200
>>>
>>> model1.update_simulation()
Routing the LSPs . . .
LSPs routed; routing demands now . . .
Demands routed; validating model . . .
```

Changing Interface cost (metric): Use Case

- From our prior visualization, we see the Interface on Node('F') facing Node('E') is over 100% utilized

```
>>> int_f_e = model1.get_interface_object('F-to-E', 'F')
>>> int_f_e.utilization
105.0
```

- There are two demands on that Interface; below we see the path(s) for each demand
  - Notice that the demand from Node('F') to Node('B') splits over 2 ECMP paths

```
>>> for dmd in int_f_e.demands(model1):
...     print(dmd)
...     pprint(dmd.path)
...     print()
...
Demand(source = F, dest = B, traffic = 50, name = "''")
[[Interface(name = 'F-to-B', cost = 6, capacity = 100, node_object = Node('F'), remote_node_object = Node('B'),
address = 13)],
 [Interface(name = 'F-to-E', cost = 3, capacity = 100, node_object = Node('F'), remote_node_object = Node('E'),
address = 14),
  Interface(name = 'E-to-B', cost = 3, capacity = 200, node_object = Node('E'), remote_node_object = Node('B'),
address = 11)]]

Demand(source = F, dest = E, traffic = 80, name = "''")
[[Interface(name = 'F-to-E', cost = 3, capacity = 100, node_object = Node('F'), remote_node_object = Node('E'),
address = 14)]]
```

Changing Interface cost (metric)

```
>>> int_f_e.cost
3
```

```
>>> int_f_e.cost = 5
>>> model1.update_simulation()
Routing the LSPs . . .
LSPs routed; routing demands now . . .
Demands routed; validating model . . .
>>>
>>> int_f_e.utilization
80.0
```

- Let’s see how changing the metric on the Interface on Node(‘F’) facing Node(‘E’)
- Simply modify the cost attribute on the Interface and update the simulation
- Examine the utilization

```
>>> for interface in model1.interface_objects:
...     if interface.utilization > 50:
...             print(interface.name,
...                     interface.node_object,
...                     interface.utilization)
...
D-to-G Node('D') 60.0
A-to-B Node('A') 68.0
A-to-D Node('A') 80.0
F-to-E Node('F') 80.0
>>>
```

FIN