2019

SDN Throwdown Program

-Hong Yan-Shanglin Guo-Guanxuan Li-

-CONET-

CONTENTS


1
Overview

Overview

Aim: Maintain the 4 best LSPs in all kinds of network conditions.

1 → 2 → 3 → 4

Gather Network Telemetry Statistics

Topology
Links status
Nodes Geolocation
Links loads
Link qualities

Cost Function

Computation of the LSPs cost based on the collected statistics.

Routing Algorithm

Choose 4 best paths according to the cost of all potential LSPs.

Update LSPs

REST API


2
Cost Function

Cost Function

1. Nodes Geolocation
2. Links Status
3. Links Load
4. Links Quality (RTT & Lost Percent)

Cost Function

1 Nodes Geolocation

Calculate the distances in the link_cost function:

```
for node in data['topology']['nodes']:
    if node['hostName'] == source:
        result = view_node(node)
        source_lat = result.json(
        )['topology']['coordinates']['coordinates'][1]
        source_lng = result.json(
        )['topology']['coordinates']['coordinates'][0]
        break
distance = distance_on_unit_sphere(
    source_lat, source_lng, dest_lat, dest_lng)
```

Cost Function

2 Links Status

Extract Links status via SocketIO_Client notification:

```
if(name == "linkEvent" and data['object']['operationalStatus'] == 'Down'):
    ips = data['object']['id'].split('_')
    failed_source = ip2host[ips[0][1:]]
    failed_destination = ip2host[ips[1]]
    failed_set.add((failed_source, failed_destination))
    failed_set.add((failed_destination, failed_source))
    update_lsp()
```

Cost Function

2 Links Status

Extract Links status via SocketIO_Client notification.

```
def link_cost(source, destination):
    if ((source, destination) in failed_set):
        return float('inf')
```

Cost Function

Links Load 3

Gather Links Load via API (gather_statistics()).

```python
def cal_bandwidth(linkname):
    # get statistics
    bandwidth_stats = gather_statistics("interfaces", requested_fields_bandwidth)

    # calculate egress traffic
    bandwidth_egress = bandwidth_stats[linkname]['interface_stats.egress_stats.if_bps']
    bandwidth_egress = [x for x in bandwidth_egress if x is not None]
    bandwidth_egress = sum(bandwidth_egress) / len(bandwidth_egress)

    # calculate ingress traffic
    bandwidth_igress = bandwidth_stats[linkname]['interface_stats.ingress_stats.if_bps']
    bandwidth_igress = [x for x in bandwidth_igress if x is not None]
    bandwidth_igress = sum(bandwidth_igress) / len(bandwidth_igress)

    # add both traffic
    bandwidth_avg = round(bandwidth_egress + bandwidth_igress, 1)

    return bandwidth_avg
```

Cost Function

Links Quality (RTT & Lost Percent) 4

Gather Links Quality via API (gather_statistics()).

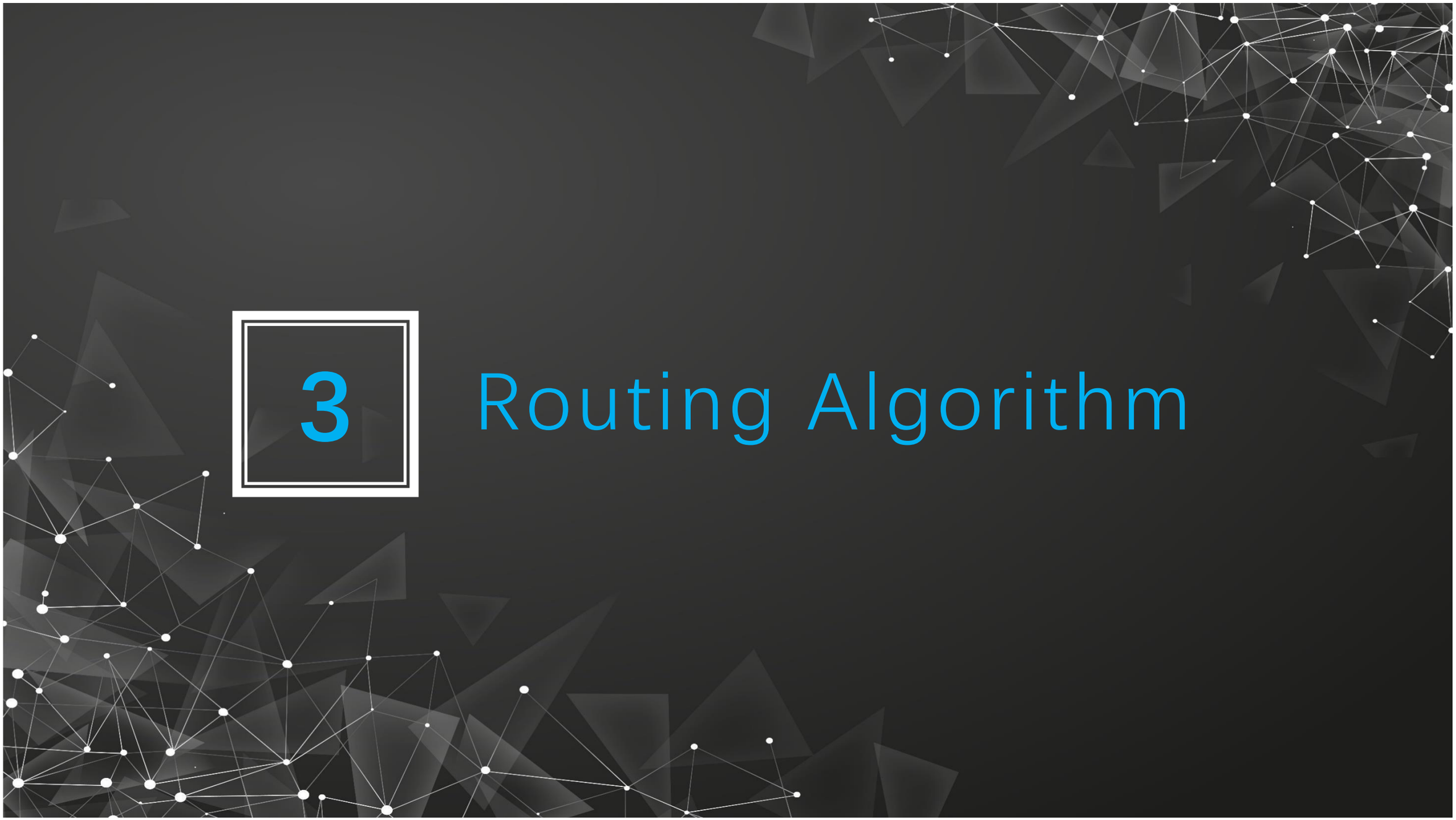
```python
def cal_rtt(linkname):
    # get statistics
    rtt_stats = gather_statistics("interfaces", requested_fields_delay_rtt)
    # calculate avg rtt
    rtt = rtt_stats[linkname]['average_rtt']
    rtt = [x for x in rtt if x is not None]
    rtt = sum(rtt) / len(rtt)
    # avg rtt
    rtt = round(rtt, 1)
    return rtt

def cal_loss_percent(linkname):
    # get statistics
    l_p_stats = gather_statistics("interfaces", requested_fields_delay_loss_percent)
    # calculate avg l_p
    l_p = l_p_stats[linkname]['loss_percent']
    l_p = [x for x in l_p if x is not None]
    l_p = sum(l_p) / len(l_p)
    # avg l_p
    l_p = round(l_p, 1)
    return l_p
```

Cost Function

Combine all the elements above to calculate the links cost.

```
distance = distance_on_unit_sphere(
    source_lat, source_lng, dest_lat, dest_lng)
bandwidth = cal_bandwidth(linkname)
rtt = cal_rtt(linkname)
loss = cal_loss_percent(linkname)
cost = distance + bandwidth * bandwidth_weight \
    + rtt * rtt_weight + loss * loss_weight
```


3
Routing Algorithm

Routing Algorithm

1. Use DFS to traverse all the possible LSPs.
2. Only select LSPs containing less than 4 hops.
3. Calculate the LSPs cost for the chosen LSPs.
4. Choose the 4 best LSPs as the best solution.
5. Update the best LSPs solution properly.

Routing Algorithm - DFS

Use **DFS** to traverse all the possible LSPs
and select those containing less than 4 hops.

1 + 2

```
def find_all_paths(graph, start, end, path=[]):
    path = path + [start]
    if start == end:
        return [path]
    paths = []
    for node in graph[start]:
        if node not in path:
            newpaths = find_all_paths(graph, node, end, path)
            for newpath in newpaths:
                if len(newpath) <= 4:
                    paths.append(newpath)
    return paths
```

Routing Algorithm

3 + 4

Calculate the LSPs cost and choose the 4 best LSPs.

```python
def find_best_paths(paths):
    best_paths = []
    cost = []
    for path in paths:
        c = 0
        for hop in range(len(path)-1):
            c += link_cost(path[hop], path[hop+1])
        cost.append(c)
    sorted_cost = sorted(cost)
    top4 = sorted_cost[:4]
    for i in top4:
        best_paths.append(paths[cost.index(i)])
    return best_paths
```

Routing Algorithm

Update the best LSPs properly.

Multithreading

Refresh LSPs every 5 mins

```
def get_traffic(thread_name, delay):
    while True:
        time.sleep(delay)
        print("update lsp by traffic")
        update_lsp()
```

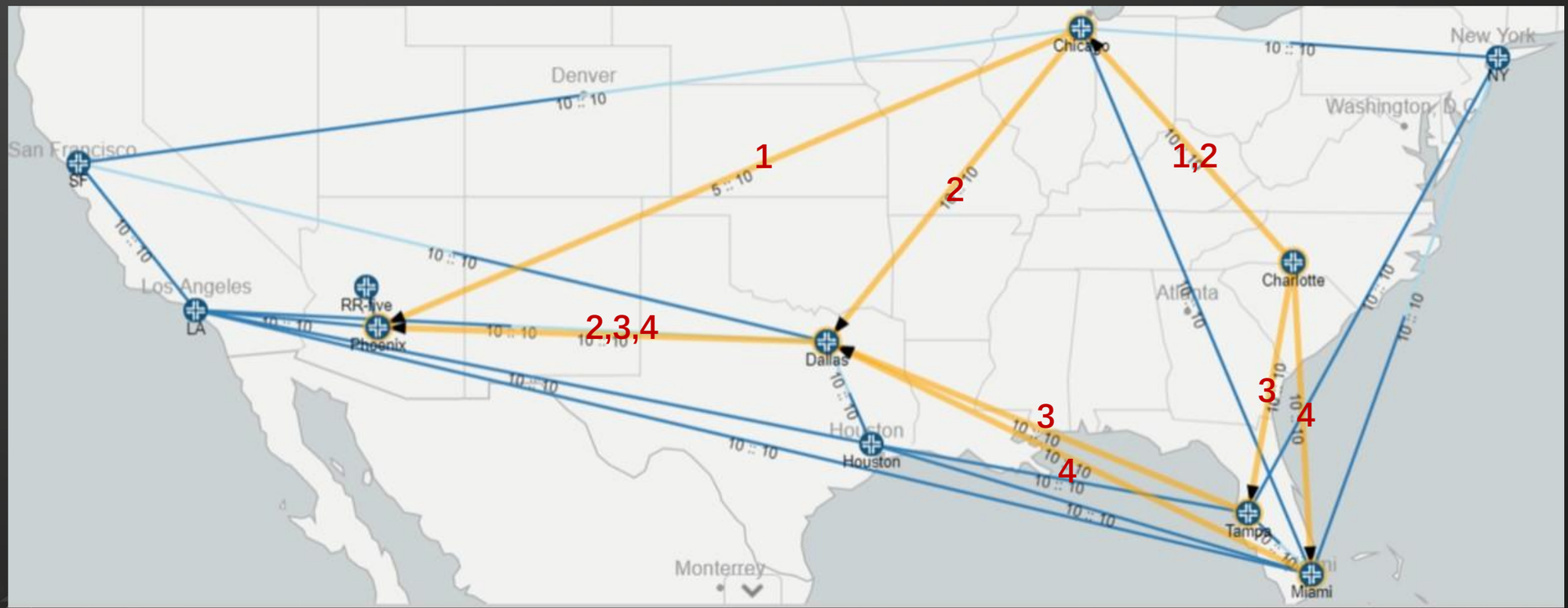
Refresh LSPs when link failure occurs.

```
connect_to_northstar()
data = gather_topology(auth_header)
mapper, ip2host, mapper2port = parse_topology(data)
# update_lsp()
thread.start_new_thread(get_traffic, ("UpdateThroughTraffic", 60*5))
socketIO = SocketIO("https://" + server_ip, 8443,
                    verify=False, headers=auth_header)
ns = socketIO.define(NSNotificationNamespace, '/restNotifications-v2')
print("start socket")
socketIO.wait()
```

4 Demo

Demo

When all the links work, the 4 best LSPs are shown below.



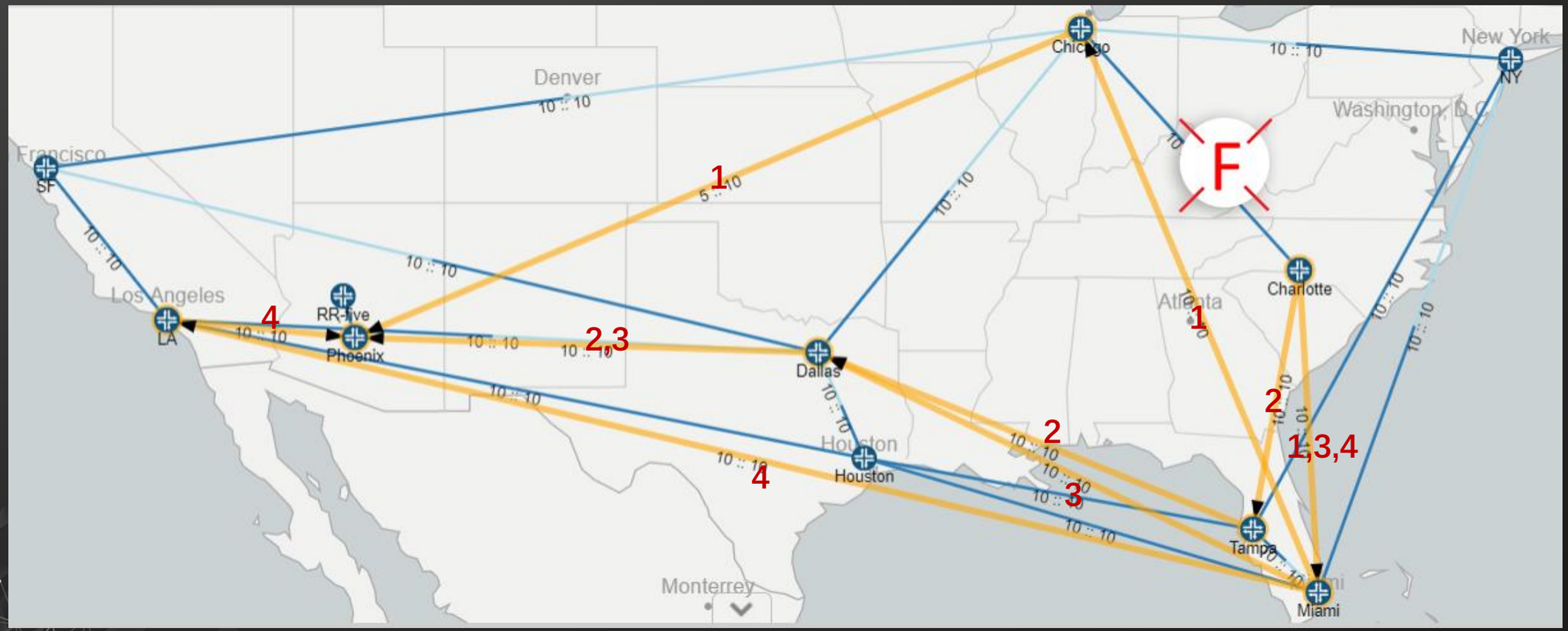
Demo

We emulate several link down situations to test our work.

```
#test-------------------------------
failed_source = "Phoenix" #None
failed_destination = "Chicago" #None
failed_set.add((failed_source, failed_destination))
failed_set.add((failed_destination, failed_source))
#test end---------------------------------
```

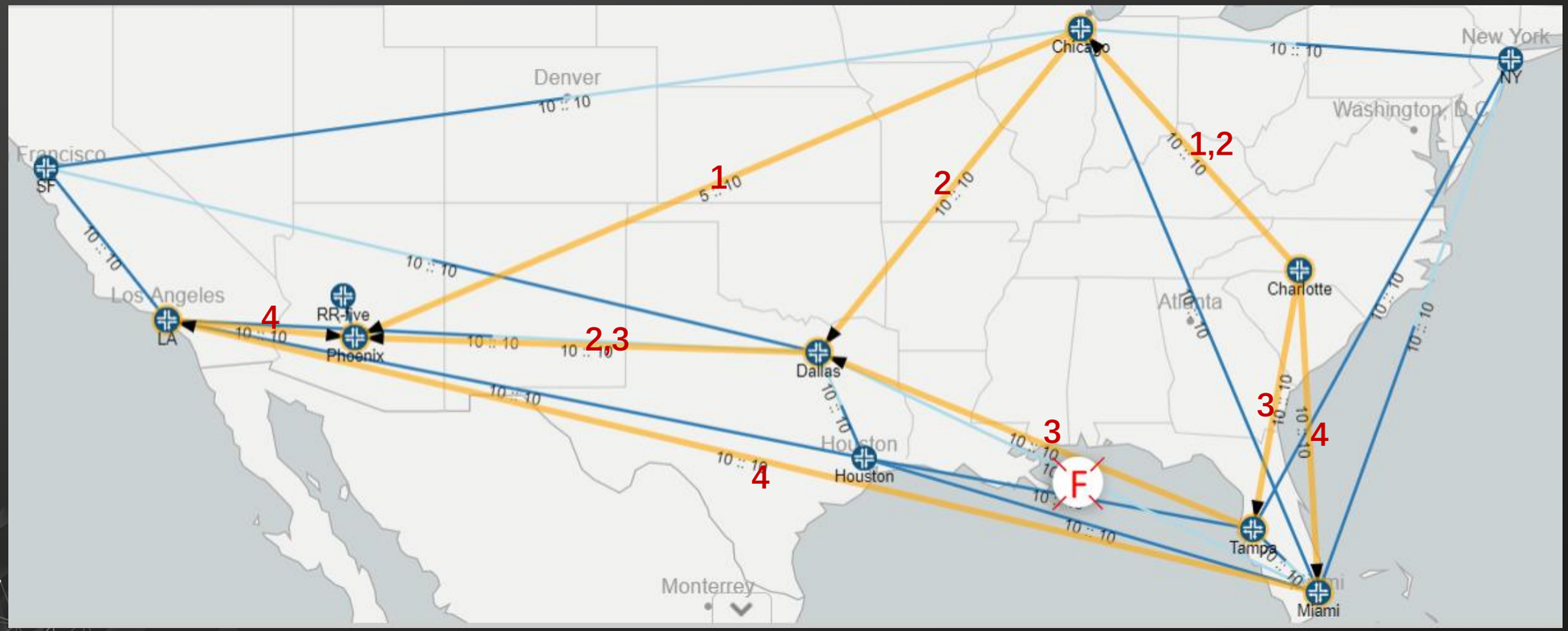
Demo

1 Chicago – Charlotte : Down



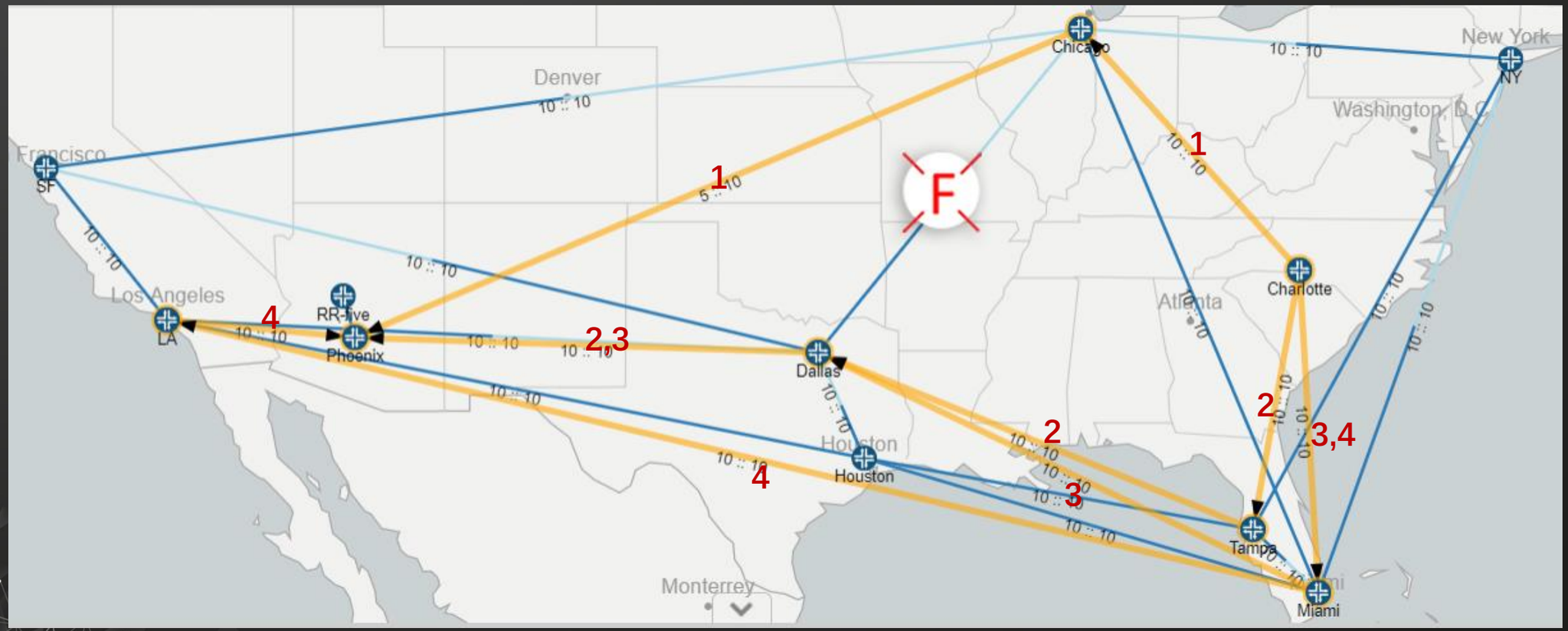
Demo

2 Miami – Dallas : Down



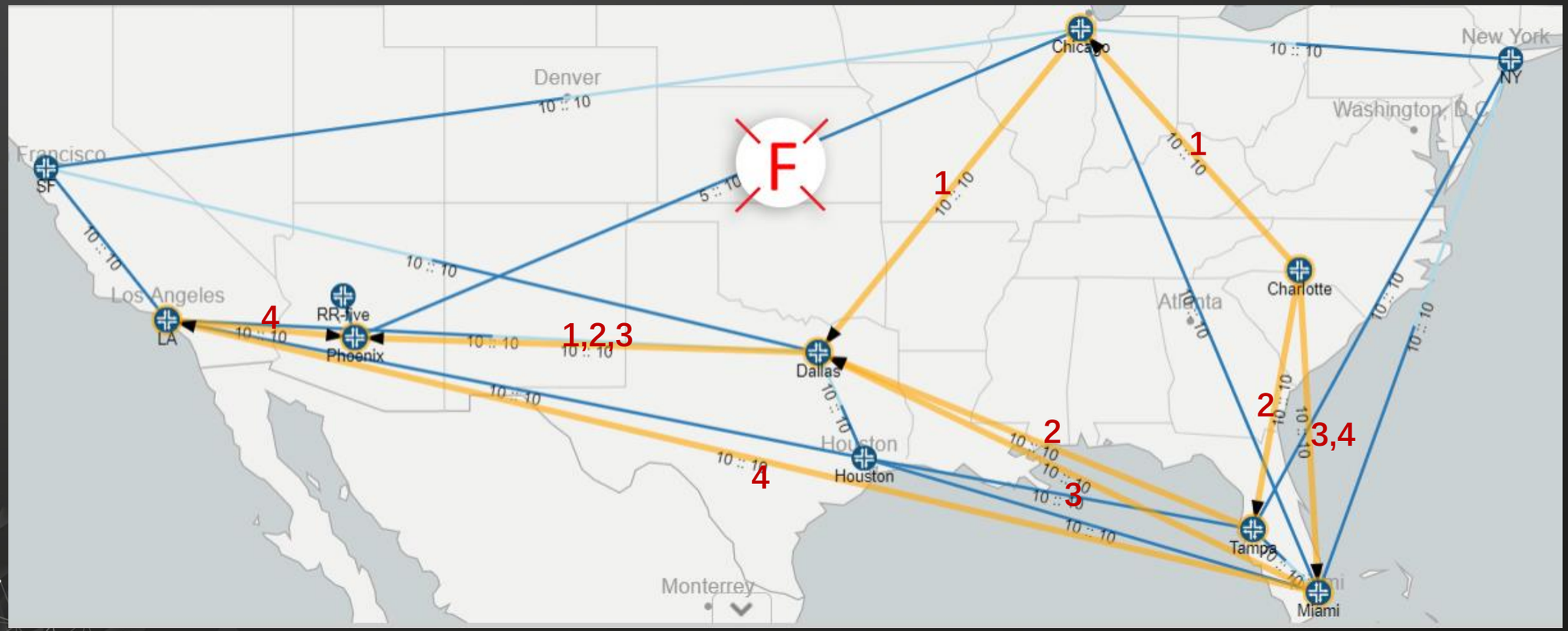
Demo

3 Chicago – Dallas : Down



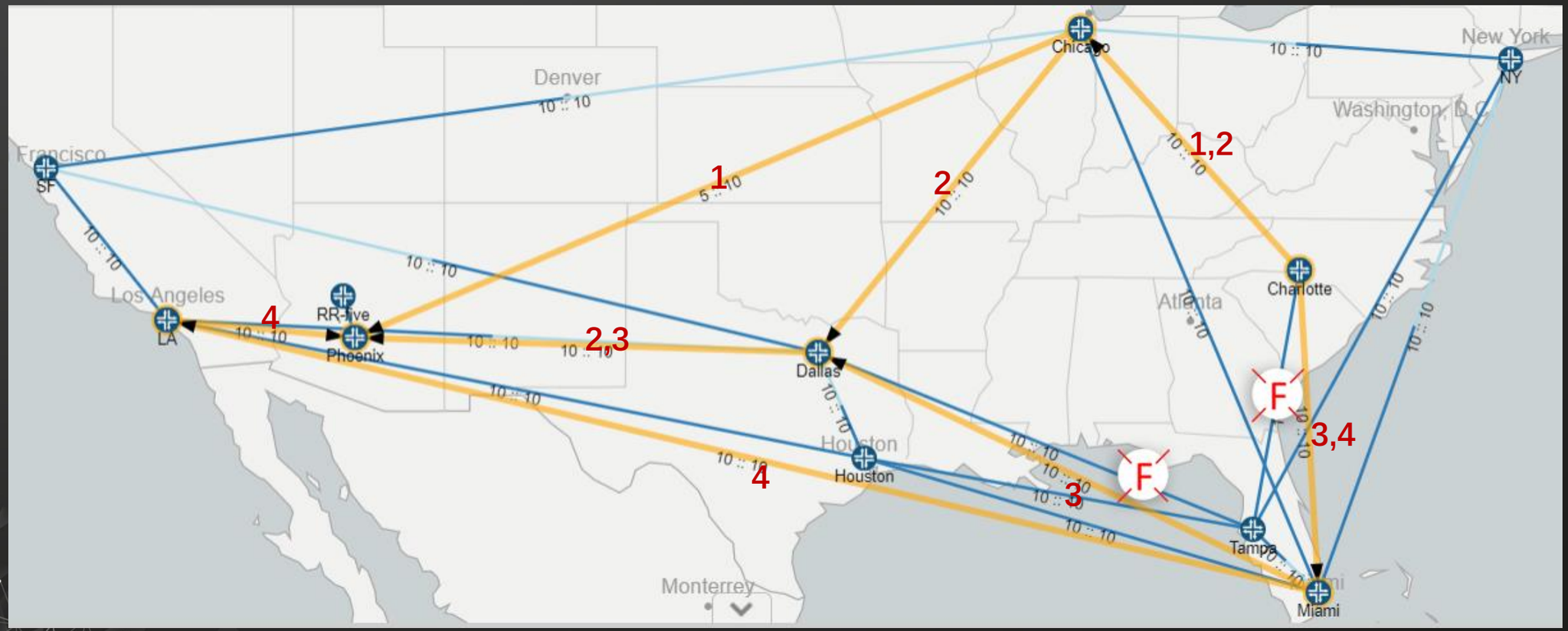
Demo

4 Chicago – Phoenix : Down



Demo

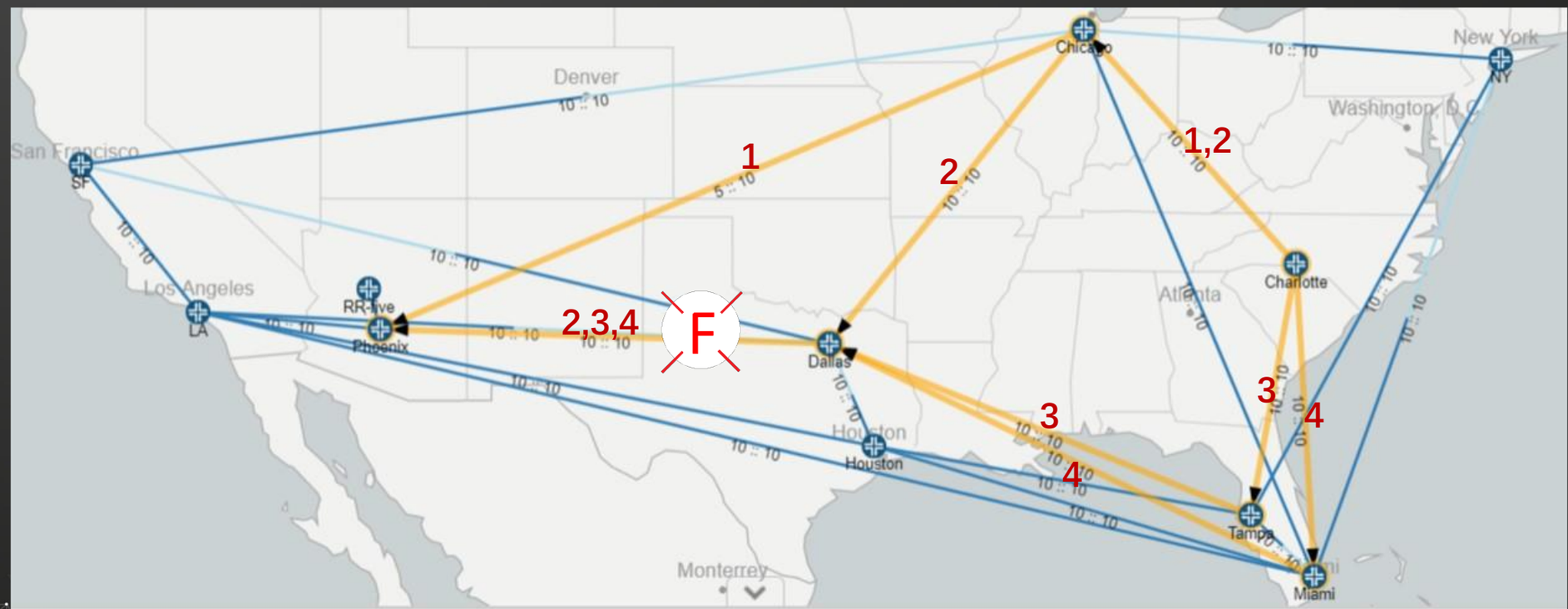
5 Tampa – Dallas / Charlotte : Down



Demo

6 Phoenix – Dallas : Down

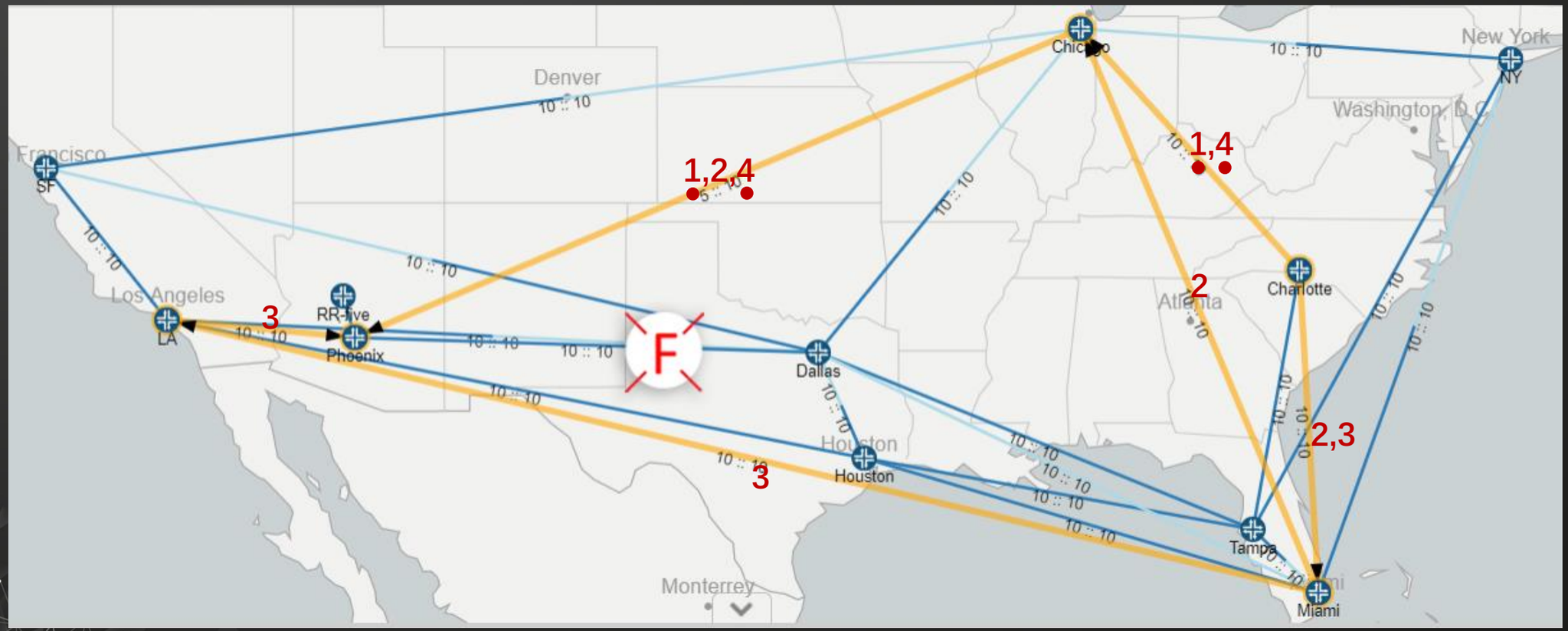
[2005.2668728197184, 2507.1913489611743, 2793.1591741066222, inf, inf, inf]



Demo

6 Phoenix – Dallas : Down

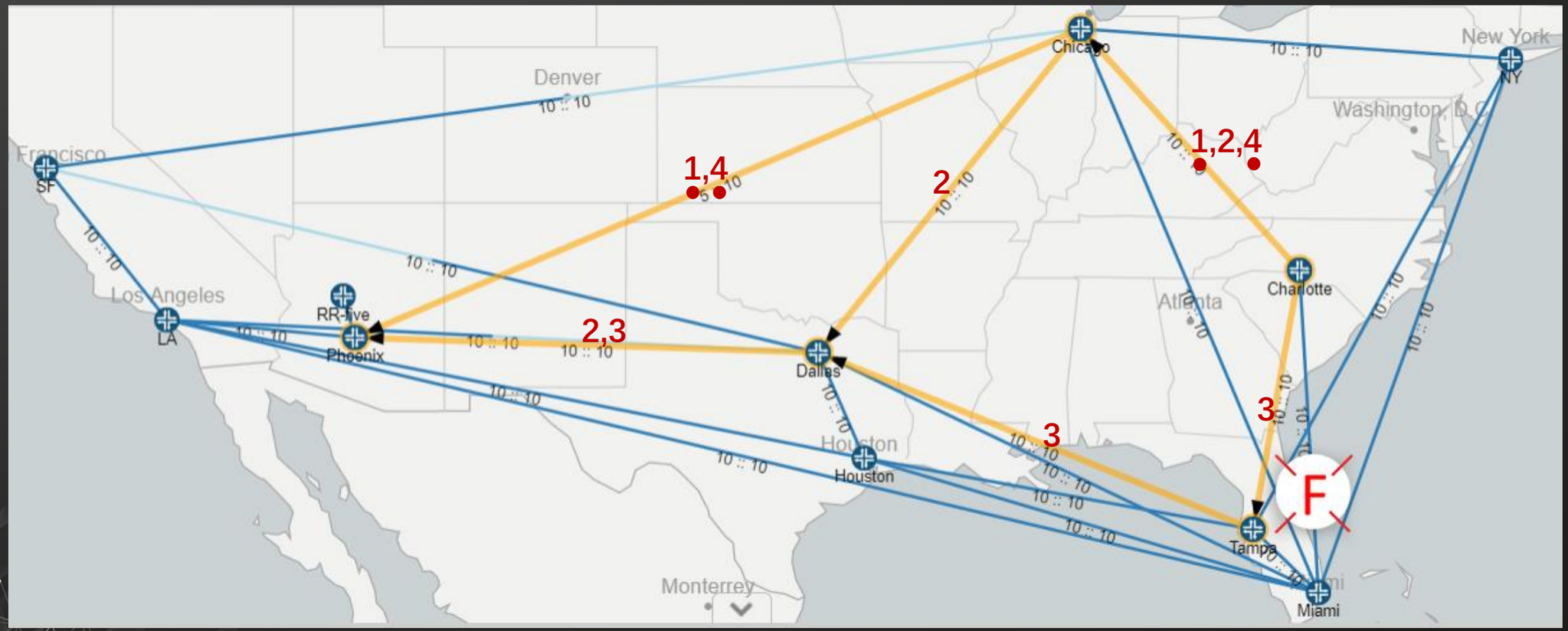
LSP 1 = LSP 4



Demo

7 Miami – Charlotte : Down

LSP 1 = LSP 4



Thank You

Q&A

- CONET -