SCOR: Software-defined Constraint Optimal Routing platform for SDN

Siamak Layeghy

April 2018

THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

Outline

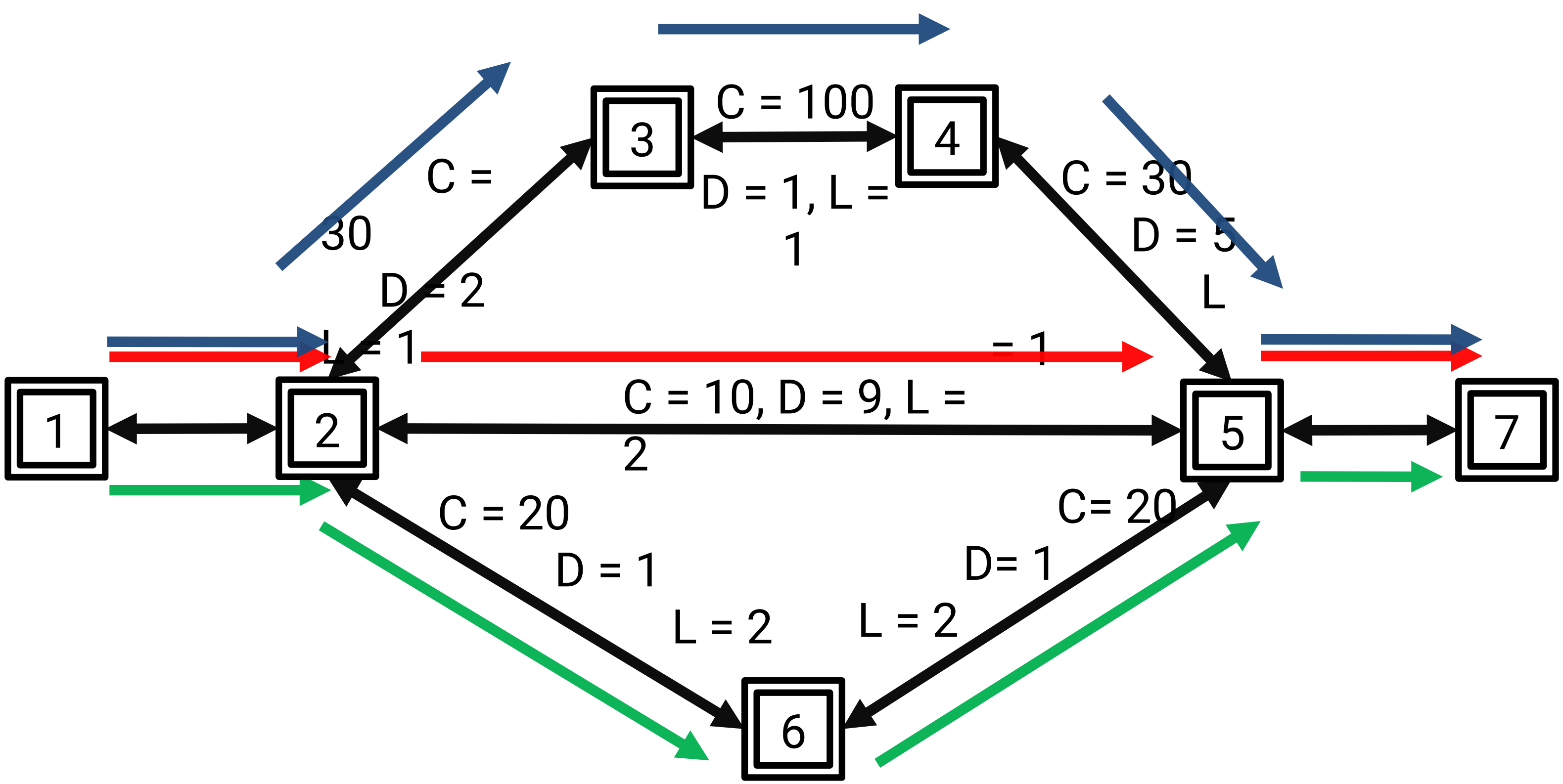
- **Background: QoS Routing**
- **SCOR’s Structure**
- **SCOR Models for QoS Routing**
- **Use Cases: ONOS apps**

QoS Routing

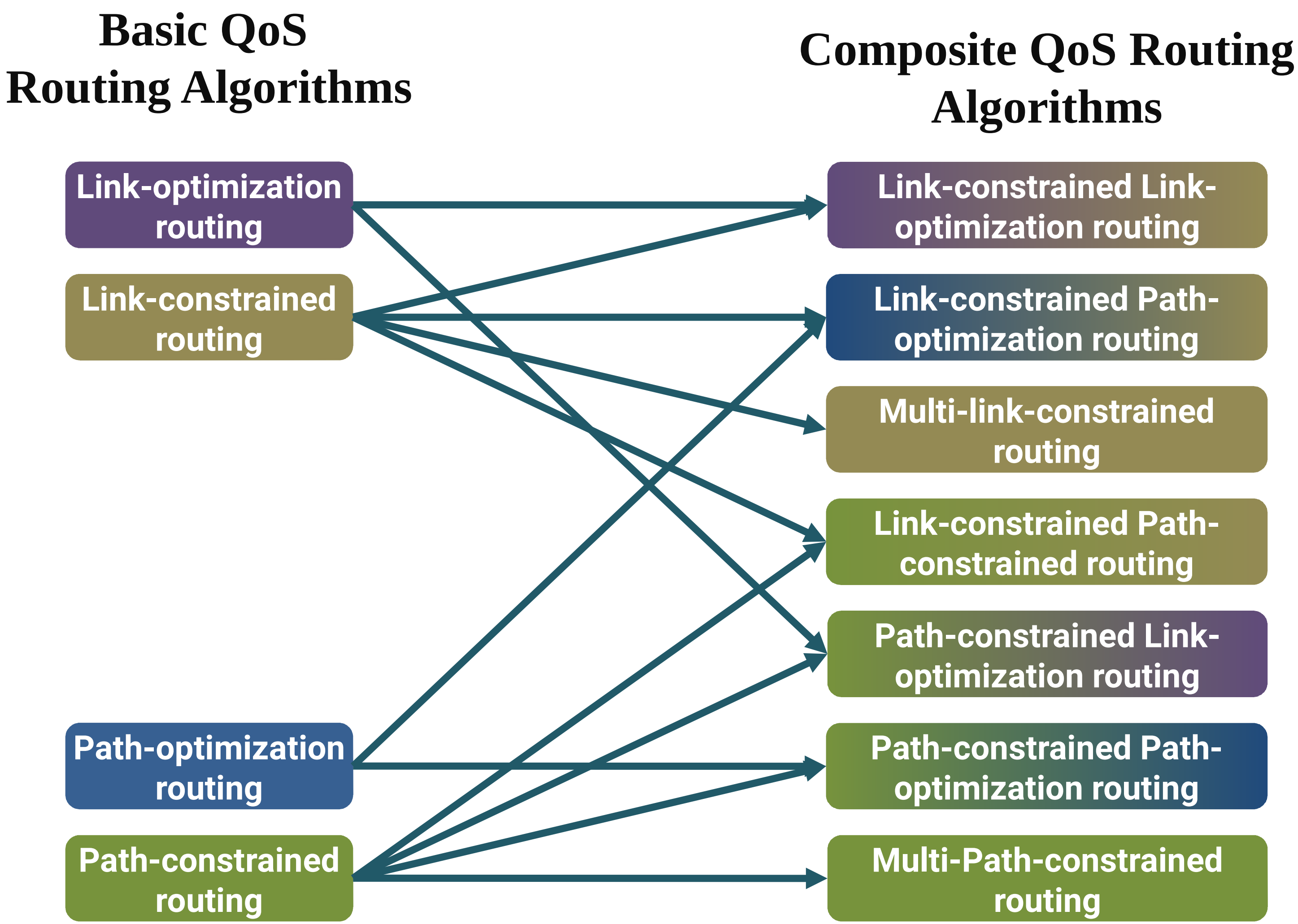
Shortest Path Routing

Widest Path Routing (Maximum Bandwidth Routing)

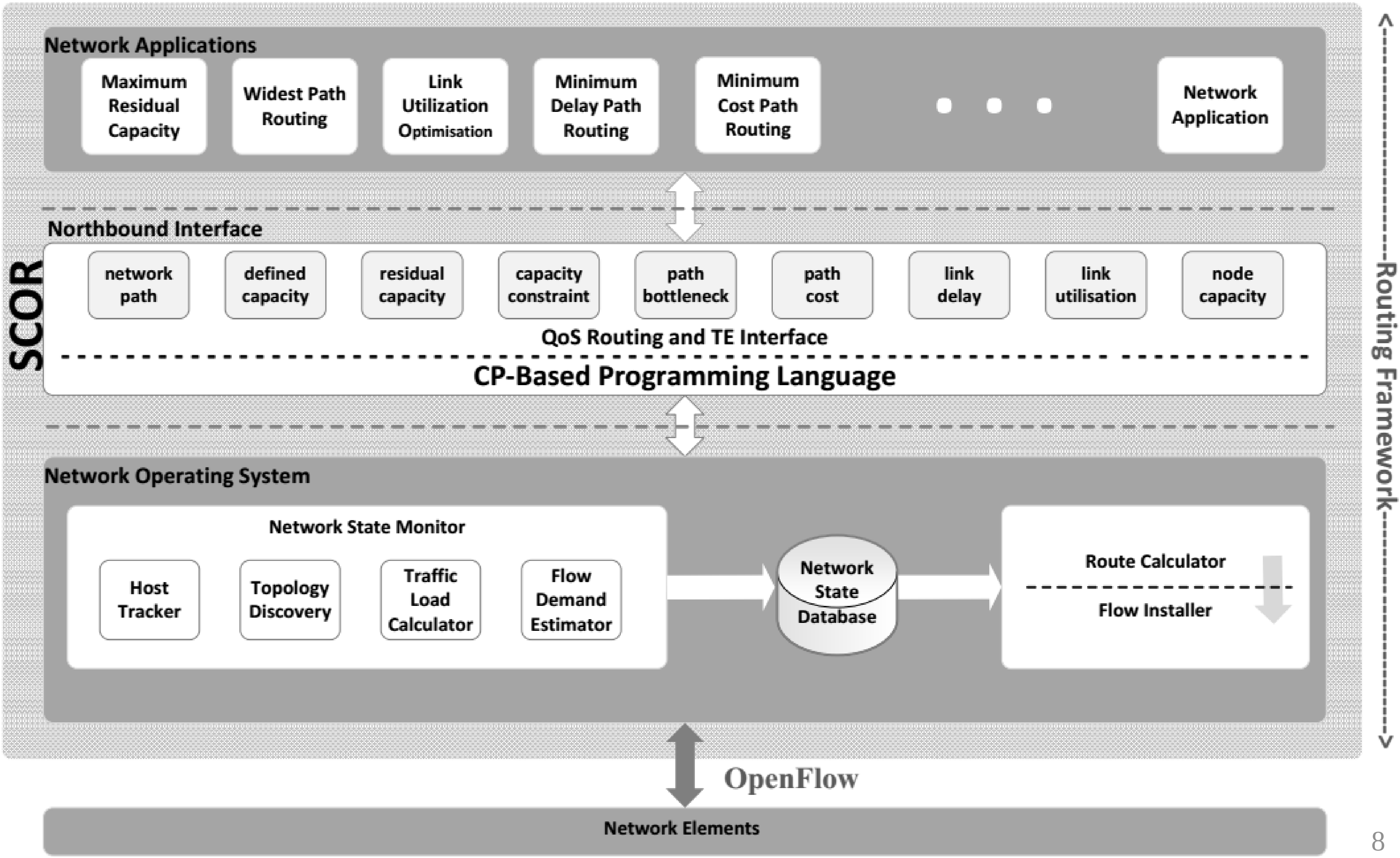
Minimum Delay Path Routing



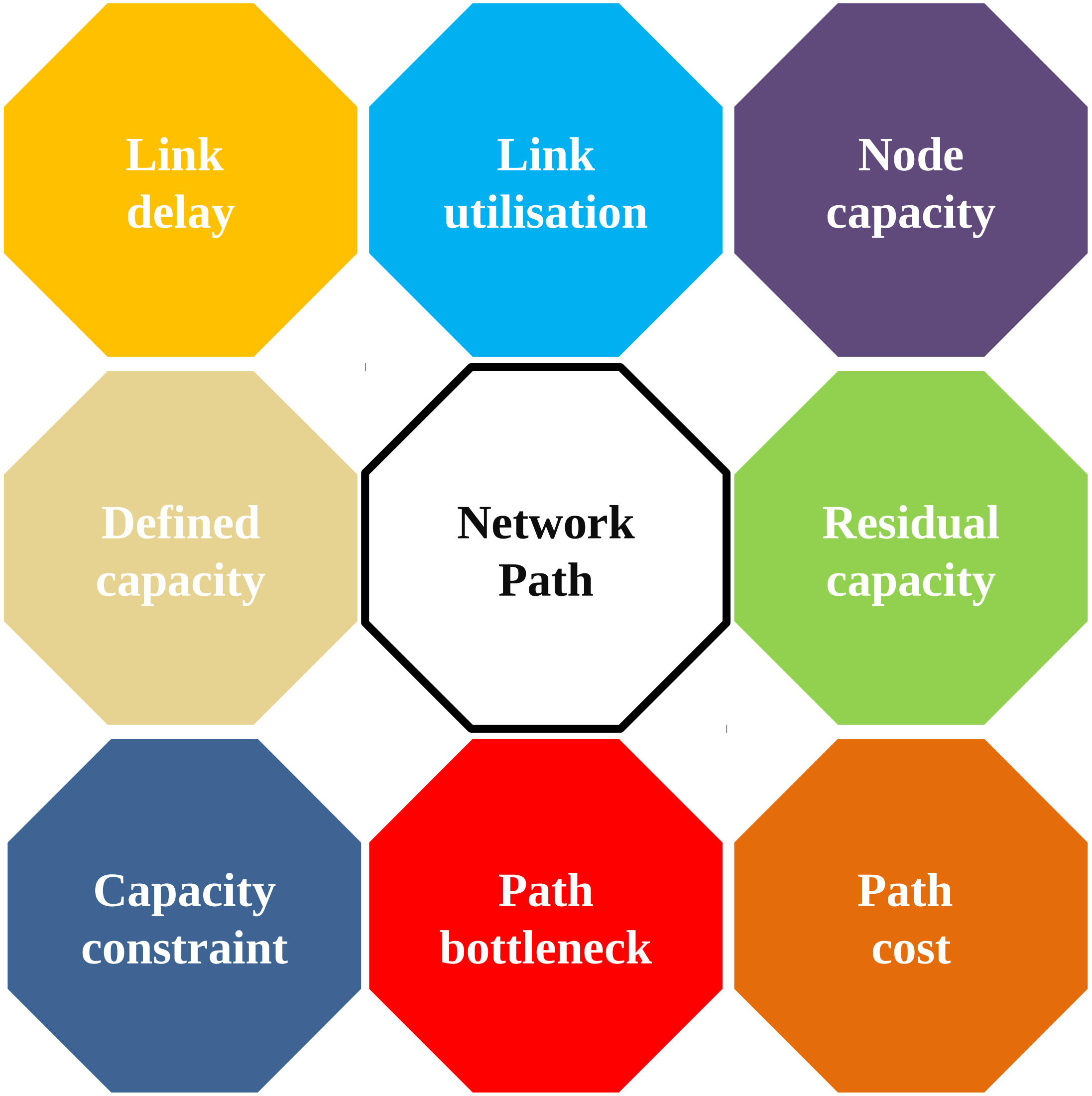
Routing & QoS Routing



SDN Routing Framework



SCOR: a New Northbound Interface for Routing



Introducing a New Northbound Interface for QoS Routing

Item	Predicate Name	Implemented Constraint/Defined Value
1	*network path*	$\sum_{\{v \mid (u,v)\in\mathcal{L}\}} f(u,v) - \sum_{\{v \mid (u,v)\in\mathcal{L}\}} f(v,u) = S_u \quad \forall u \in \mathcal{N}$
2	*defined capacity*	$c(u,v) \geq c_0 \quad \forall (u,v) \in P_f$
3	*residual capacity*	$r(u,v) = c(u,v) - f(u,v) \quad \forall (u,v) \in \mathcal{L}$
4	*capacity constraint*	$f(u,v) \leq c(u,v) \quad \forall (u,v) \in \mathcal{L}$
5	*path bottleneck*	$c_B[P_f] = \min_{(u,v)\in P_f} \{c(u,v)\}$
6	*path cost*	$a[P_f] = \sum_{(u,v)\in P_f} a(u,v) f(u,v)$
7	*link delay*	$D(u,v) = \frac{1}{c(u,v) - f(u,v)} \quad \forall (u,v) \in P_f$
8	*link utilisation*	$U(u,v) = \frac{f(u,v)}{c(u,v)} \quad \forall (u,v) \in P_f$
9	*node capacity*	$\sum_{\{v \mid (u,v)\in\mathcal{L}\}} f(u,v) + \sum_{\{v \mid (u,v)\in\mathcal{L}\}} f(v,u) \leq C_u \quad \forall u \in \mathcal{N}$

Introducing a New Northbound Interface for QoS Routing

Predicate 1: Network Path

```
forall(i in 1..N)(
        forall(j in 1..F)(
              flow_in_links[i] = sum(k in 1..L)(
                    if Links[k,2]=i then LPM[k, j] else 0 endif
                                  )
              ∧
              flow_out_links[i] = sum(k in 1..L)(
                    if Links[k,1]=i then LPM[k, j] else 0 endif
                                  )
              ∧
              flow_in_links[i] + (if i = s[j] then 1 else 0 endif) =
              flow_out_links[i] + (bfif i = t[j] then 1 else 0 endif)
              ∧
              flow_in_links[i ] <= 1
        )
  );
```

Predicate 2: Defined Capacity

```
forall(k in 1..L)(
      forall(j in 1..F)(
          if Links[k,4] < Limits[j] then LPM[k,j] = 0
          else true endif
      )
  );
```

Predicate 4: Capacity Constraint

```
include "Predicate_residual_capacity.mzn";
residual_capacity( LPM, Flows, Links, Residuals );
   ∧
forall(k in 1..L)(
         Residuals[k]>0
   );
```

Predicate 3: residual Capacity

```
forall(k in 1..L)(
         if sum(j in 1..F)(LPM[k,j])=0
         then Residuals[k] = Cmax
         else Residuals[k] = Links[k,4] - sum(j in 1..F)(Flows[j] × LPM[k,j])
         endif
   );
```

Introducing a New Northbound Interface for QoS Routing

Predicate 5: path cost

```
forall(j in 1..F)(
    Total_Cost[j] =
        sum(k in 1..L)(
            Links[k,3] × LPM[k,j] × Flows[j]
        )
);
```

Predicate 6: path bottleneck

```
forall(j in 1..F)(
    forall(k in 1..L)(
        Width[k] = Links[k,4] × LPM[k,j]
    )
    ∧
    forall(k in 1..L, where Width[k] !=0)(
        Bandwidth[k] = Width[k]
    )
    ∧
    Bottleneck[j] = min(Bandwidth)
);
```

Predicate 7: link delay

```
include “Predicate_capacity_constraint.mzn";
capacity_constraint( LPM, Flows, Links, Residuals );
  ∧
forall(k in L)(
    if sum(j in 1..F)(LPM[k,j])=0
    then Delay[k] = 0
    else Delay[k] = 1 / Residuals[k]
    endif
);
```

Introducing a New Northbound Interface for QoS Routing

Predicate 9: node capacity

```
forall(i in 1..N)(
    node_flow_in[i] = sum(k in 1..L, j in 1..F)(
                if Links[k,2]=i then LPM[k, j] × Flows[j]
                else 0
                endif
            )
    ∧
    node_flow_out[i] = sum(k in 1..L, j in 1..F)(
                if Links[k,1]=i then LPM[k, j] × Flows[j]
                else 0
                endif
            )
    ∧
    node_flow_in[i] + node_flow_in[i] <= Node_Capacity
);
```

Predicate 8: link utilisation

```
forall(k in 1..L)(
    Link_Utilisation[k] = (
                sum(j in 1..F)(
                        Flows[j] × LPM[k,j]
                )/ Link[k,4]
    ) × 100
);
```

Modelling various QoS Routing Algorithms in SDN

Algorithm	Metrics	Type
Shortest Path	Hop Count	Basic
Widest Path	Bandwidth	Basic
Minimum-Delay Path	Transmission Delay	Basic
Minimum-Loss Path	Packet Loss	Basic
Constrained-Delay Path	Transmission Delay	Basic
Constrained-Bandwidth Path	Bandwidth	Basic
Constrained-Bandwidth -Minimum-Delay Path	Bandwidth, Transmission Delay	Multi-constraint
Maximum-Residual-Capacity Path	Bandwidth	Complex
Constrained-Residual-Capacity Path	Bandwidth	Complex
Minimum-Link-Utilisation Path	Bandwidth/total delay	Complex
Constrained-Link-Utilisation Path	Bandwidth/total delay	Complex
Minimum-Congestion Path	Bandwidth/total delay	Complex
Constrained-Congestion Path	Bandwidth/total delay	Complex

Least Cost Path Routing

$$\text{minimise} \quad \sum_{(u,v)\in\mathcal{L}} a(u,v)f(u,v)$$

$$\sum_{\{v|(u,v)\in\mathcal{L}\}} f(u,v) - \sum_{\{v|(u,v)\in\mathcal{L}\}} f(v,u) = S_u, \qquad \forall u \in \mathcal{N}$$

$$f(u,v) = f_1(u,v) = d_1 x_1(u,v)$$

$$\text{minimise} \quad \sum_{(u,v)\in\mathcal{L}} a(u,v)x_1(u,v)$$

$$\sum_{\{v|(u,v)\in\mathcal{L}\}} x_1(u,v) - \sum_{\{v|(u,v)\in\mathcal{L}\}} x_1(u,v) = S_u/d_1, \quad \forall u \in \mathcal{N}$$

SCOR Model: Least Cost Path Routing

Model 1: Least Cost Path Routing in SCOR

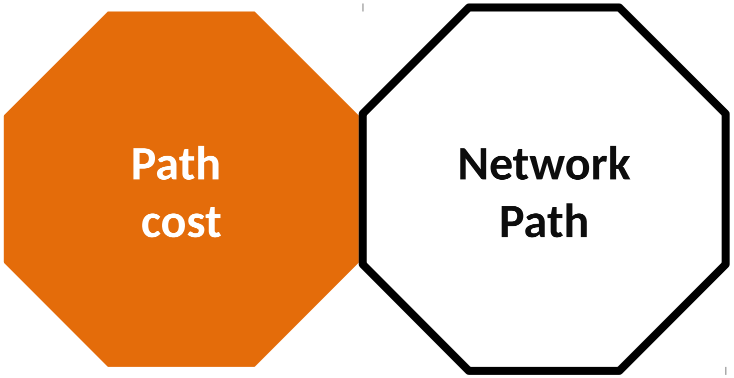
```
% Include item
include "Predicate_network_path.mzn";
include "Predicate_path_cost.mzn";

% Parameters
array[int, 4] of int : Links;
int : L = max(index_set_1of2(Links));
array[int] of int : Nodes;
int : Flows;
int : s;
int : t;

% Decision Variables
var int : Cost;
array[1..L, 1] of var 0..1 : LPM;

% Constraints item
constraint network_path(LPM, Links, Nodes, s, t);
constraint path_cost(LPM, Links, Cost, Flows);

% Solve item
solve minimize Cost;
```



Least Cost Path Constrained Capacity Routing

$$\text{minimise} \quad \sum_{(u,v)\in\mathcal{L}} a(u,v)f(u,v)$$

$$\sum_{\{v|(u,v)\in\mathcal{L}\}} f(u,v) - \sum_{\{v|(u,v)\in\mathcal{L}\}} f(v,u) = S_u, \qquad \forall u \in \mathcal{N}$$

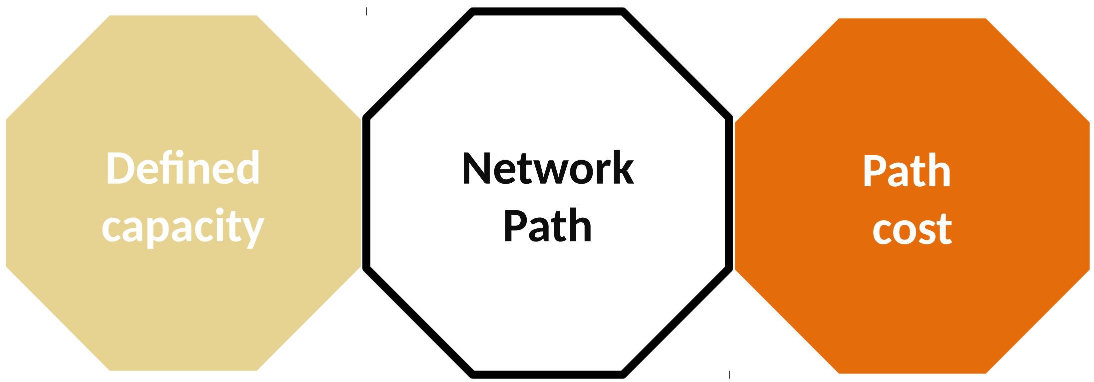
$$f(u,v) = f_1(u,v) = d_1 x_1(u,v)$$

$$\text{minimise} \quad \sum_{(u,v)\in\mathcal{L}} a(u,v)x_1(u,v)$$

$$\sum_{\{v|(u,v)\in\mathcal{L}\}} x_1(u,v) - \sum_{\{v|(u,v)\in\mathcal{L}\}} x_1(u,v) = S_u/d_1, \ \ \forall u \in \mathcal{N}$$

$$x_1(u,v) \leq c(u,v)/d_1 \qquad \forall (u,v) \in \mathcal{L}$$

SCOR Model: Least Cost Path Constrained Capacity Routing



Model 2: Least Cost Path with Defined Capacity Routing in SCOR

```
⋮
% Include item
include "Predicate defined capacity.mzn";
⋮
% Parameters
int : Limit;
⋮
% Constraints item
constraint defined_capacity(LPM, Links, Flows, Limit);
⋮
```

Maximum Residual Capacity Path Routing

$$\text{maximise}\{\mathcal{Z}\}$$

$$\mathcal{Z} \le r(u,v) \qquad \forall (u,v) \in \mathcal{L}$$

$$\sum_{\{v|(u,v)\in\mathcal{L}\}} x_j(u,v) - \sum_{\{v|(u,v)\in\mathcal{L}\}} x_j(v,u) = S_u^j/d_j \qquad \forall u \in \mathcal{N},\ j = 1..F$$

$$\sum_{j=1..F} d_j x_j(u,v) \le c(u,v) \qquad \forall (u,v) \in \mathcal{L}$$

$$r(u,v) = c(u,v) - f(u,v) \qquad \forall (u,v) \in \mathcal{L}$$

$$f(u,v) = \sum_{j=1..F} f_j(u,v) = \sum_{j=1..F} d_j x_j(u,v) \qquad \forall (u,v) \in \mathcal{L}$$

$$x_j(u,v) = \begin{cases} 0 & (u,v) \notin P_j \\ 1 & (u,v) \in P_j \end{cases}$$

$$S_u^j/d_j = \begin{cases} 1 & u = s_j, \\ -1 & u = t_j, \\ 0 & otherwise \end{cases} \qquad u \in \mathcal{N},\ j = 1..F$$

SCOR Model: Maximum Residual Capacity Path Routing

Model 3: Maximum Residual Capacity Routing in SCOR

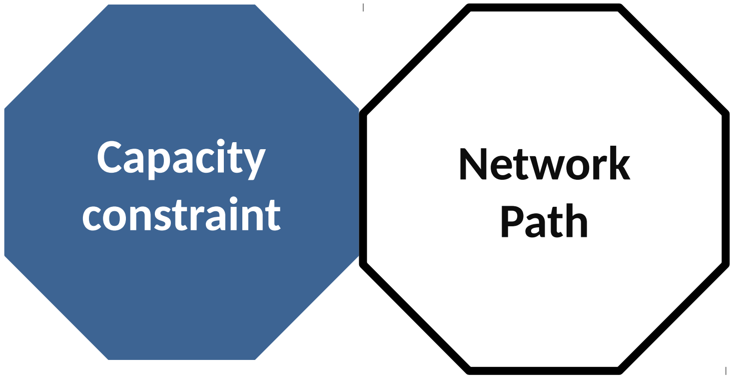
```
% Include item
include "Predicate network path.mzn";
include "Predicate capacity constraint.mzn";

% Parameters
array[int, int] of int : Links;
int : L = max(index_set_1of2(Links));
array[int] of int : Nodes;
array[int] of int : Flows;
int : F = max(index_set(Flows));
array[1..F] of int : s;
array[1..F] of int : t;

% Decision Variables
array[1..L, 1..F] of var 0..1 : LPM;
array[1..L] of var int : Residuals;

% Constraints item
constraint network_path(LPM, Links, Nodes, s, t);
constraint capacity_constraint(LPM, Links, Flows, Residuals);

% Solve item
solve maximize min(Residuals);
```



MRCPR in Procedural Programming

```
str(len(G.node.keys()))+","+str(len(edges))+") w/ error " + str(error)+ ": " +
str(shortestPathComputations)
    for node in G.node.iterkeys():
        print node, G.edge[node]
    return shortestPathComputations,count


def get_beta_hat(edges, commodities, error=GLOBAL_ERROR, karakosta=True):
    spc, beta_hat = maximum_concurrent_flow(edges, commodities, error=1., returnBeta=True,
karakosta=karakosta,
                                            scale_beta=False)
    return beta_hat, spc


def two_approx(edges, commodities, error=GLOBAL_ERROR, karakosta=True):
    beta_hat, spc = get_beta_hat(edges, commodities, error=1., returnBeta=True, karakosta=karakosta)
    scale_demands(commodities, beta_hat / 2.)
    return maximum_concurrent_flow(edges, commodities, error=error, karakosta=karakosta,
shortestPathComputations=spc)

def multi_route(edges, commodities, error=GLOBAL_ERROR, scale_beta=True, karakosta=False):
    beta_hat, spc = get_beta_hat(edges, commodities, error=1.)
    return maximum_concurrent_flow(edges, commodities, error=error, karakosta=karakosta,
shortestPathComputations=spc,

                             scale_beta=scale_beta, multi_route=True, beta_hat=beta_hat)
```

```
        commodityTable[commodity] = 0
    for head in G.edge.iterkeys():
        for tail, edge_dict in G.edge[head].iteritems():
            for commodity in commodities:
                if tail == commodity.sink:

 commodityTable[commodity]+=edge_dict[FLOW_ATTRIBUTE]/commodity.demand
    print "Lambda is " + str( min([x for x in commodityTable.itervalues()]))
    print "OBJECTIVE: ", calculate_dual_objective(G)
    print "SPC-"+str(karakosta)+"-"+str(twoApprox)+ " for G("
                edge[FLOW_ATTRIBUTE] = edge.get(FLOW_ATTRIBUTE, 0) + added_flow
```

Half-duplex Maximum Residual Capacity Path Routing

maximise$\{\mathcal{Z}\}$

$$\mathcal{Z} \leq r(u,v) \qquad \forall (u,v) \in \mathcal{L}$$

$$\sum_{\{v|(u,v)\in\mathcal{L}\}} x_j(u,v) - \sum_{\{v|(u,v)\in\mathcal{L}\}} x_j(v,u) = S_u^j/d_j \qquad \forall u \in \mathcal{N},\ j = 1..F$$

$$\sum_{j=1..F} d_j x_j(u,v) \leq c(u,v) \qquad \forall (u,v) \in \mathcal{L}$$

$$r(u,v) = c(u,v) - f(u,v) \qquad \forall (u,v) \in \mathcal{L}$$

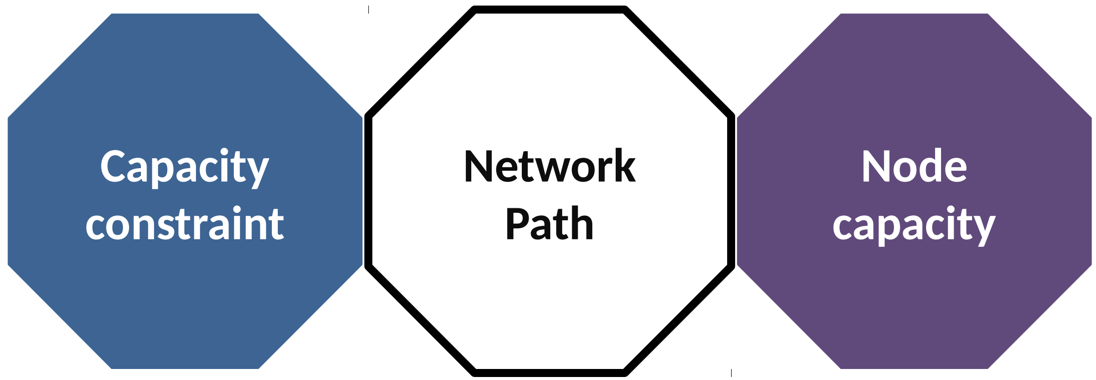
$$f(u,v) = \sum_{j=1..F} f_j(u,v) = \sum_{j=1..F} d_j x_j(u,v) \qquad \forall (u,v) \in \mathcal{L}$$

$$x_j(u,v) = \begin{cases} 0 & (u,v) \notin P_j \\ 1 & (u,v) \in P_j \end{cases}$$

$$\sum_{\{v|(u,v)\in\mathcal{L}\}} f(u,v) + \sum_{\{v|(u,v)\in\mathcal{L}\}} f(v,u) \leq C_u \qquad \forall u \in \mathcal{N}$$

$$S_u^j/d_j = \begin{cases} 1 & u = s_j, \\ -1 & u = t_j, \\ 0 & otherwise \end{cases} \qquad u \in \mathcal{N},\ j = 1..F$$

SCOR Model: Half-duplex Maximum Residual Capacity Path Routing



Model 4: Half-duplex Maximum Residual Capacity Routing in SCOR
(Only lines not included in maximum residual capacity routing)

⋮

% *Include item*

1 : **include** "*Predicate node capacity.mzn*";

⋮

% *Parameters*

2 : **array**[*int*] *of* **int** : $Nodes\_Capacities$;

⋮

% *Constraints item*

3 : **constraint** $node\_capacity(LPM, Links, Flows, Nodes\_Capacities)$;

⋮

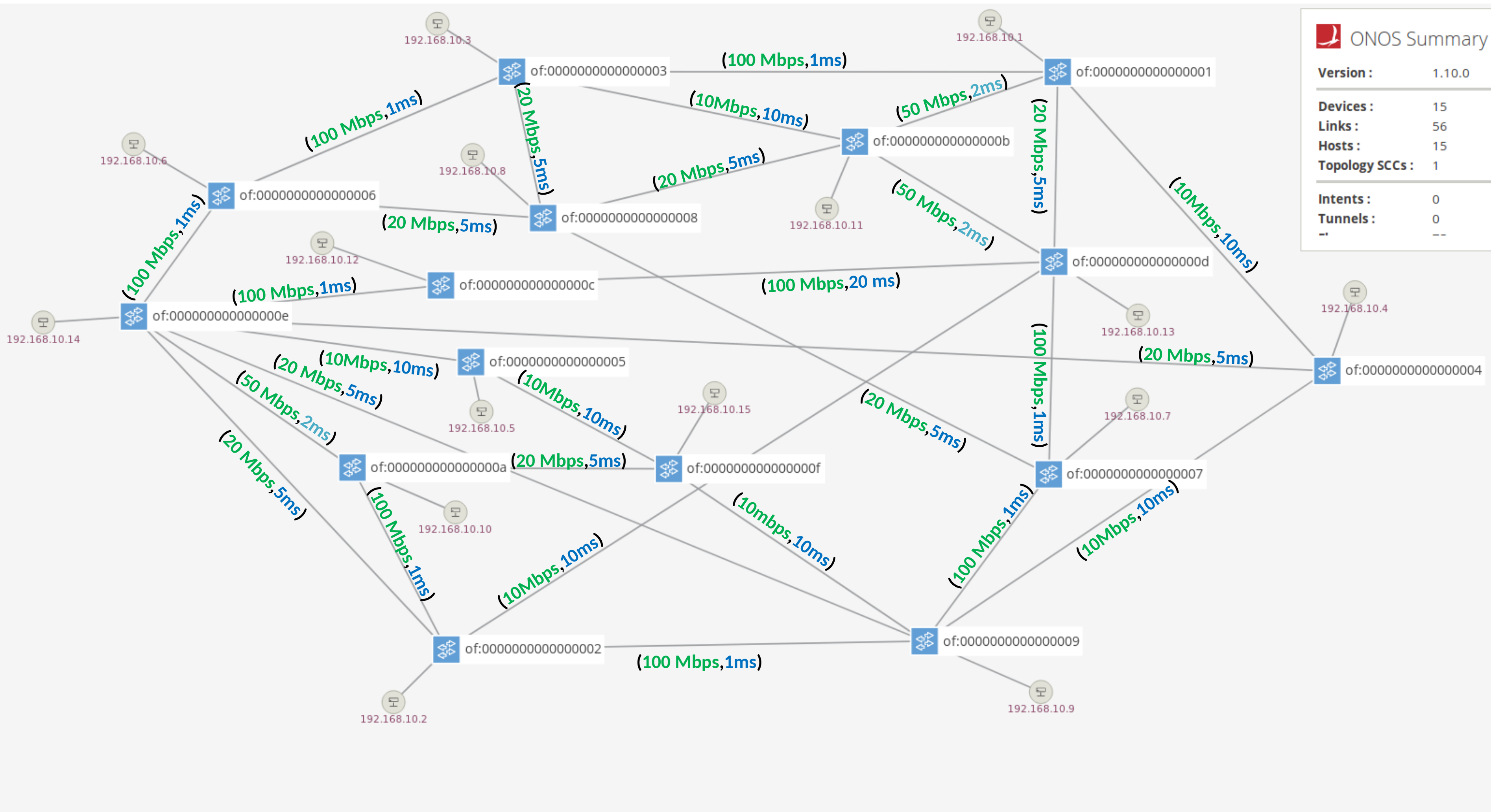
Conciseness & Completeness

QoS Routing Problem	SCOR Predicates	#L
Shortest Path	path cost	3
Widest Path [11]	path bottleneck	3
Bandwidth-Guaranteed [13]	capacity constraint	3
Bandwidth-Constrained [11]	defined capacity	3
Minimum-Loss [14]	path cost	3
Minimum-Delay [14]	path cost	3
Minimum-Delay [64]	delay	3
Delay-Constrained [11]	path cost	3
Least-Cost [11]	path cost	3
Maximum Residual Capacity [12]	capacity constraint	3
Minimum Link Utilisation [16]	link utilisation	4
Delay-Constrained Least-Cost [11]	path cost $\times 2$	4
Delay-Delay Jitter-Constrained [11]	path cost $\times 2$	4
Bandwidth-Delay-Constrained [11]	defined capacity path cost	4
Bandwidth-Constrained Least-Delay [11]	defined capacity path cost	4
Minimum-Cost Bandwidth-Constrained [15]	capacity constraint path cost	4
Delay-Constrained Bandwidth-Optimisied [11]	path cost path bottleneck	4
Widest Shortest Path [17]	path cost path bottleneck	6
Shortest Widest Path [17]	path cost path bottleneck	6

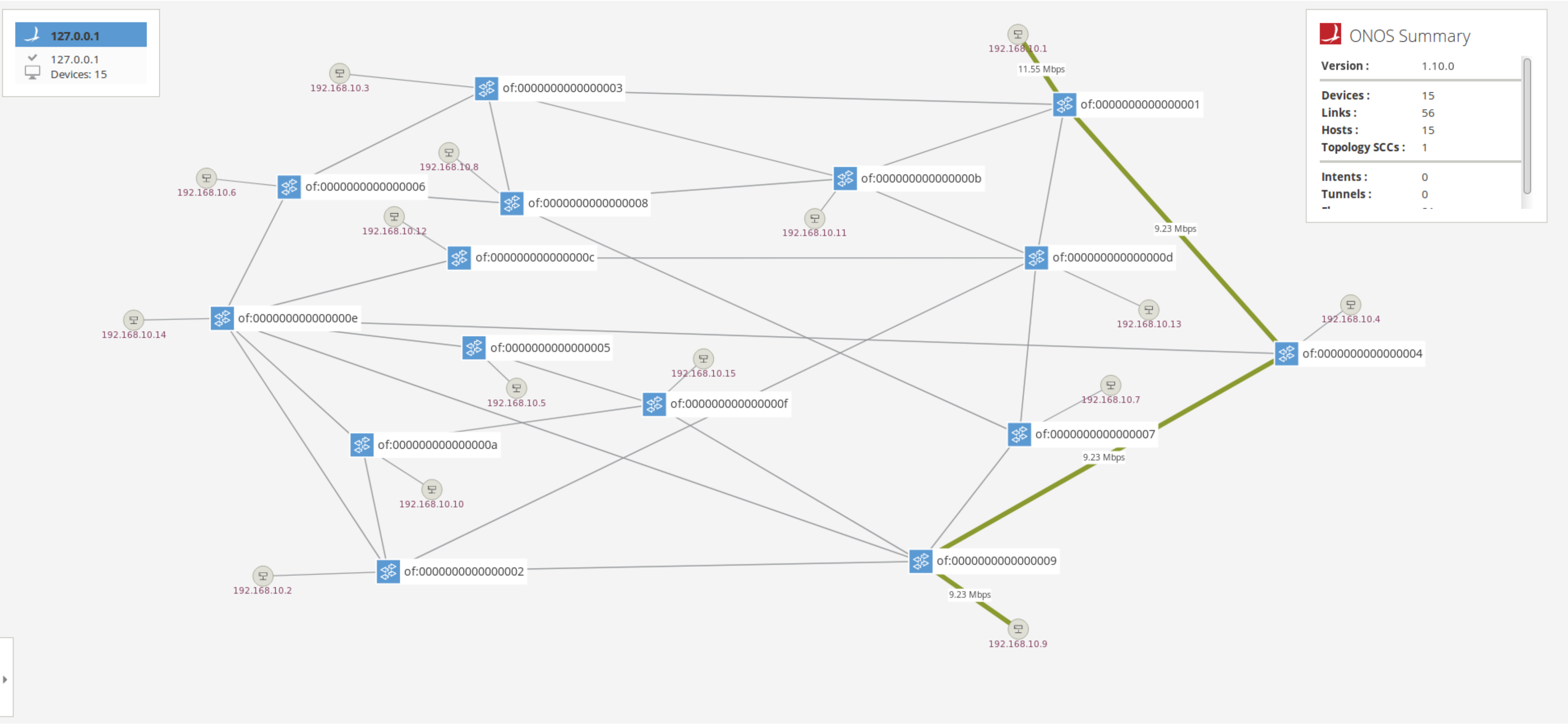
Real-World Use Cases: ONOS Apps

- **Max-BW Routing**
- **Min-Delay Routing**
- **Max-BW Constrained Delay Routing**
- **Min-Delay BW Constrained Routing**
- **Minimizing Max Link Utilization**
- **Maximizing Min Residual Capacity**

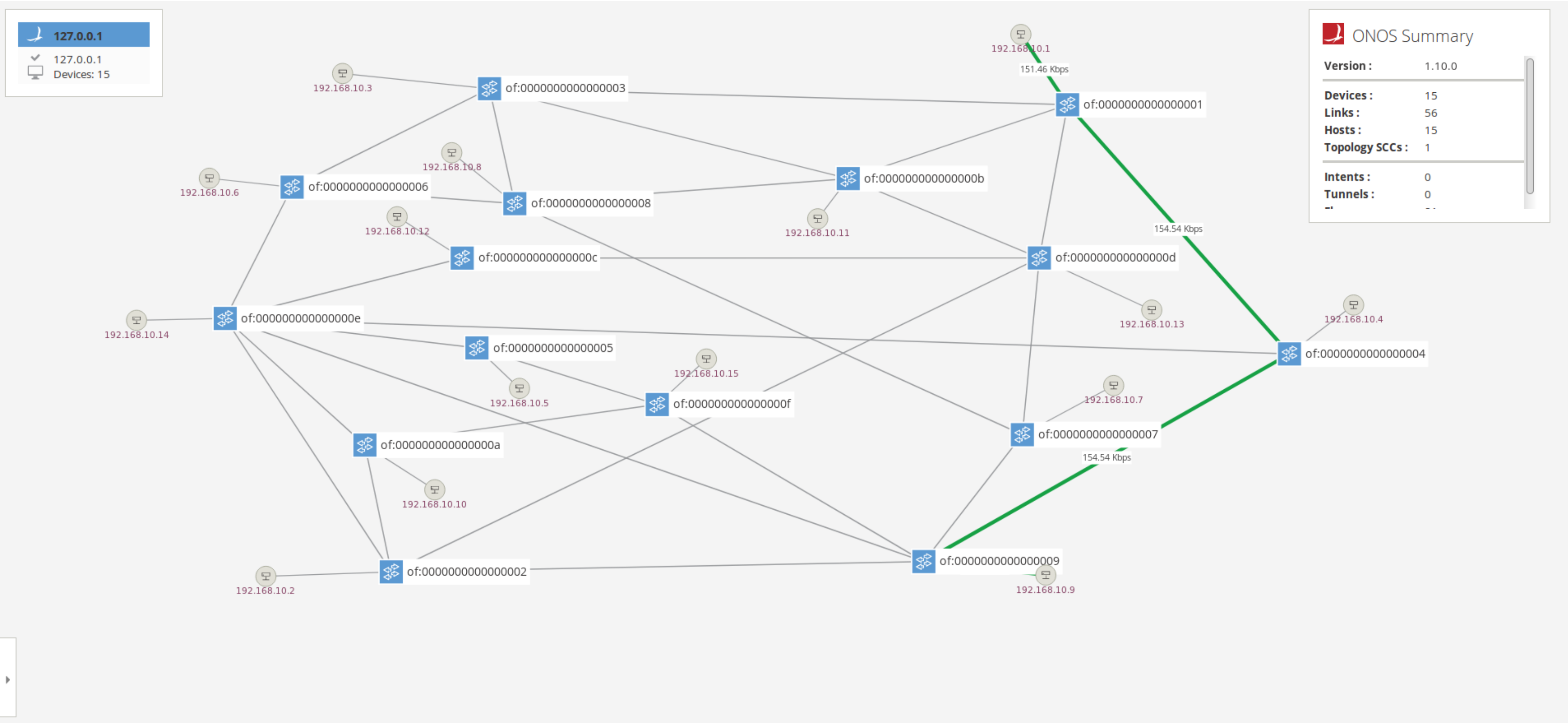
Real-World Use Cases: Scenario



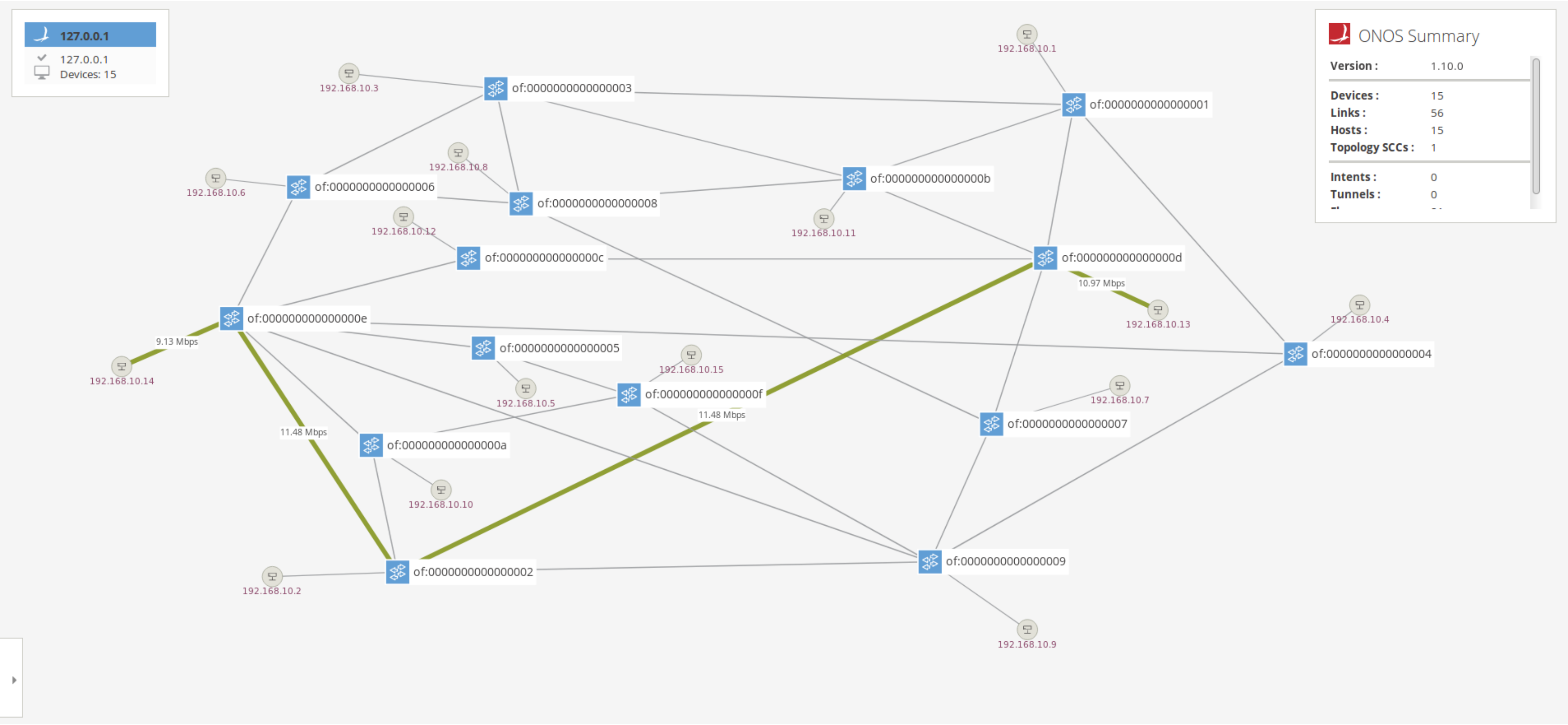
Maximum-Bandwidth Routing



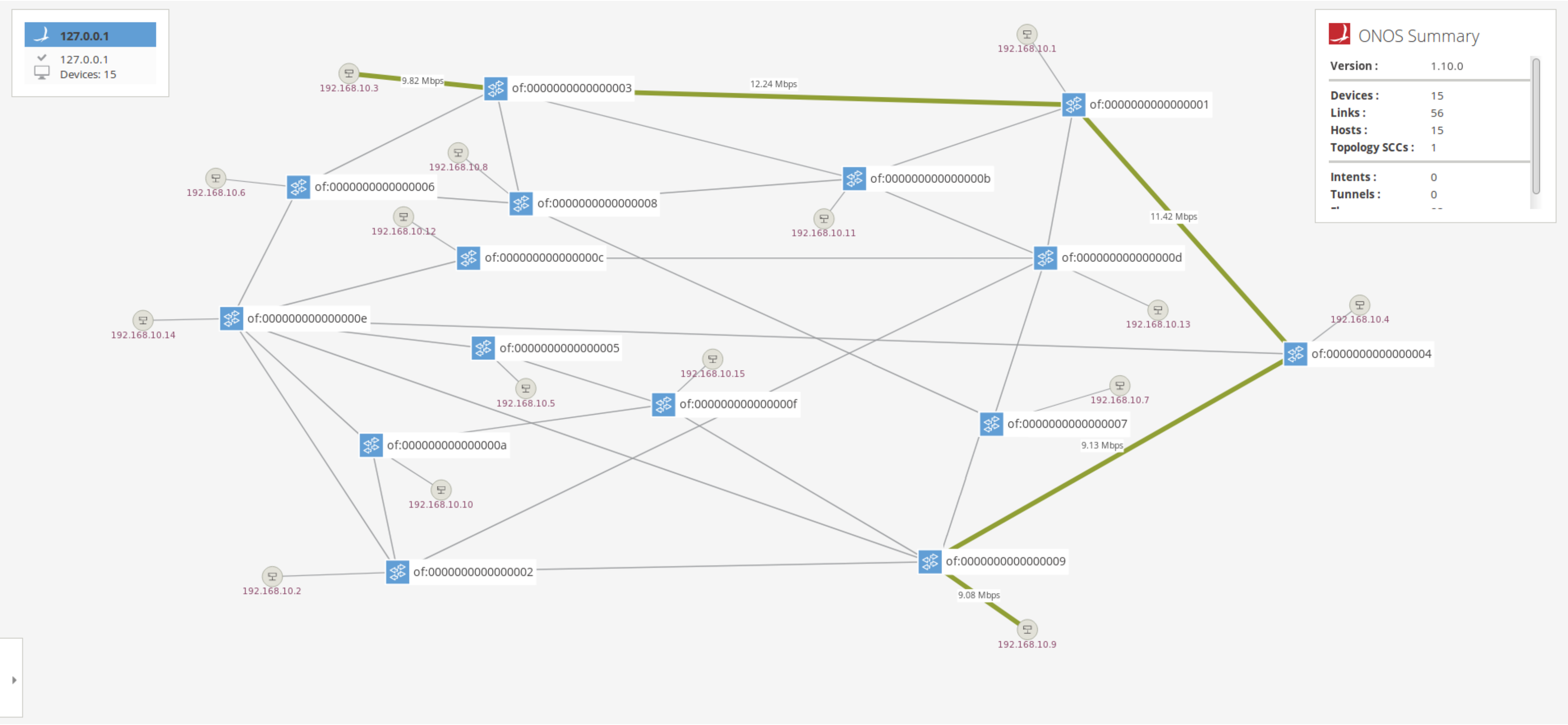
Minimum-Delay Routing



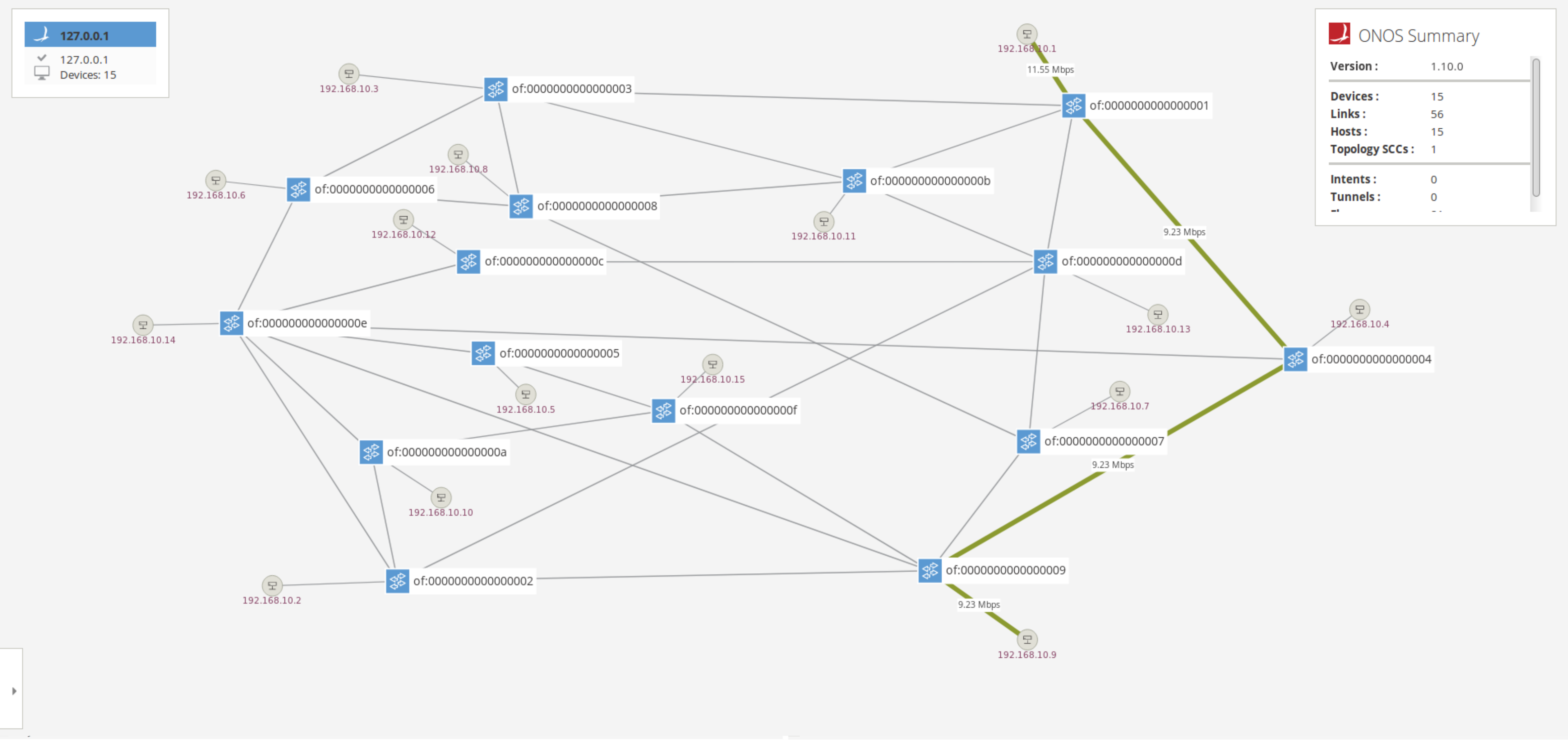
Max-BW Constrained Delay Routing



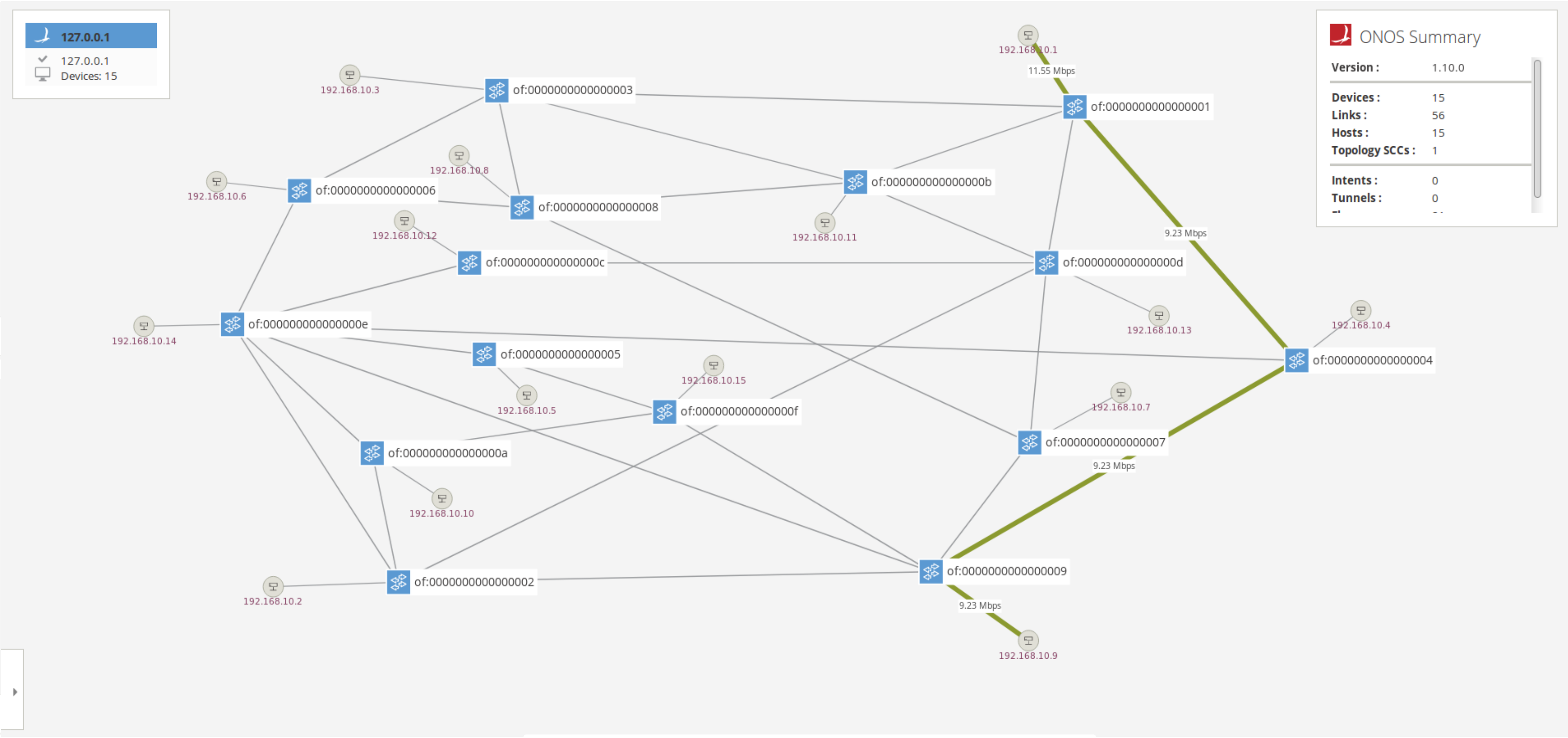
Min-Delay Constrained BW Routing



Minimizing Max Link Utilization



Maximizing Min Residual Capacity



Q & A
Thank You