JOINT ROUTING AND PLACEMENT OF VIRTUAL NETWORK FUNCTIONS

Jorge Crichigno\textsuperscript{1,2}, D. Oliveira\textsuperscript{3}, M. Pourvali\textsuperscript{3}, N. Ghani\textsuperscript{3}, D. Torres\textsuperscript{2}

\textsuperscript{1}University of South Carolina, SC, USA
\textsuperscript{2}Northern New Mexico College, NM, USA
\textsuperscript{3}University of South Florida, FL, USA
Agenda

• Introduction
• Optimization model
• Numerical examples
• Concluding remarks
Introduction

- Network Function Virtualization (NFV) is a technology that permits the implementation of Network Functions (NFs) on datacenters’ commodity servers.
- Network functions include:
  - Firewall, access control lists
  - Routers, switches, NAT, DHCP

http://www.vmware.com/
Introduction

• Consider the weighted network below
• A set of datacenter that implement particular functions
• There is a set of function $F = \{0, 1\}$
• A client request is interested in both functions to apply them to a flow from ingress switch 0 to egress switch 3
• A datacenter $d$ implements $F_d \subseteq F$
• The cost and resources to implement a function are datacenter-dependent
• What should the path of the flow be, in order to minimize the routing and deployment costs?
Introduction

Example:

- Datacenter 1 implements functions 0 and 1 at costs 1 and 10
- Datacenter 2 implements functions 0 and 1 at costs 10 and 1
Example:

- Datacenter 1 implements functions 0 and 1 at costs 1 and 10
- Datacenter 2 implements functions 0 and 1 at costs 10 and 1
- The optimal solution places functions 0 and 1 at datacenters 1 and 2 respectively, and route the traffic through (0, 2), (2, 1), (1, 3)
Optimization Model

- The network is represented as a graph $G = (V, E)$
- Each link $(i, j) \in E$ has an associated cost $c_{ij}$
- The subset $D \subseteq V$ represents the set of datacenters
- A datacenter $d \in D$ implements a subset of functions $F_d \subseteq F$
- Each request $r \in R$ is characterized by a 3-tuple $(src_r, dst_r, F_r)$
- A datacenter has a set of resources $W = \{w_{d,1}, w_{d,2}, \ldots, w_{d,m}\}$
- To implement function $i \in F_d$, the datacenter uses $w_{d,1}^i, w_{d,2}^i, \ldots, w_{d,m}^i$
- The setup cost of an instance $i \in F_d$ is $c_{i_d}^i$
- Each instance $i \in F_d$ can serve up to $\lambda_d^i$ requests
- Variable $x_{r,d}^i$ indicates whether datacenter $d$ serves function $i \in F_r$ requested by $r \in R$
- Variable $y_d^i$ indicates the number of instances of function $i$ at $d$
- Variable $l_{r}^{ij}$ indicates whether link $(i, j) \in E$ is used by flow $r \in R$
Optimization Model

- The objective is the maximization of the number of satisfied network functions (NFs)

\[
\text{Max } F_1 = \sum_{r \in R} \sum_{i \in F_d} \sum_{d \in D} x_{r,d}^i = x_{0,1}^0 + x_{0,1}^1 + x_{0,2}^0 + x_{0,2}^1
\]
Optimization Model

- The objective is the maximization of the number of satisfied network functions (NFs)

\[
\text{Max } F_1 = \sum_{r \in R} \sum_{i \in F_d} \sum_{d \in D} x_{r,d}^i = x_{0,1}^0 + x_{0,1}^1 + x_{0,2}^0 + x_{0,2}^1
\]

- Minimization of the NF deployment cost

\[
\text{Max } -F_2 = \sum_{d \in D} \sum_{i \in F_d} c_{d} y_{d}^i = y_1^0 + 10y_1^1 + 10y_2^0 + y_2^1
\]
Optimization Model

- Minimization of the routing cost

\[
\text{Max - F3} = \sum_{r \in R} \sum_{(i,j) \in E} c^{i,j} l_r^{(i,j)} = \\
10l_0^{(0,1)} + l_0^{(0,2)} + 10l_0^{(1,0)} + l_0^{(1,2)} + l_0^{(1,3)} + l_0^{(2,0)} + l_0^{(2,1)} + l_0^{(2,3)} + l_0^{(3,1)} + l_0^{(3,2)}
\]
Optimization Model

- Requested functions 0 and 1 are only implemented in one datacenter

\[
\sum_{d \in D} x_{r,d}^i \leq 1 \quad \Rightarrow \quad x_{0,1}^0 + x_{0,2}^0 \leq 1 \quad \text{Function 0}
\]

\[
x_{0,1}^1 + x_{0,2}^1 \leq 1 \quad \text{Function 1}
\]
Optimization Model

• The total amount of resources (memory, CPU, storage) is limited at each datacenter

• E.g., 15 and 20 storage units used by an instance of function 0 and 1 respectively at datacenter 1. Datacenter has 100 storage units

\[
\sum_{i \in F_d} w_{d,j}^i y_d^i \leq w_{d,j} \quad \text{Datacenter 1, storage resource}
\]

\[
15y_1^0 + 20y_1^1 \leq 100
\]
Optimization Model

• There is a path from the ingress switch 0 to egress switch 3

Node 0: \[ (l_0^{(0,1)} + l_0^{(0,2)}) - (l_0^{(1,0)} + l_0^{(2,0)}) = 1 \]

Node 1: \[ (l_0^{(1,0)} + l_0^{(1,2)} + l_0^{(1,3)}) - (l_0^{(0,1)} + l_0^{(2,1)} + l_0^{(3,1)}) = 0 \]

Node 2: \[ (l_0^{(2,0)} + l_0^{(2,1)} + l_0^{(2,3)}) - (l_0^{(0,2)} + l_0^{(1,2)} + l_0^{(3,2)}) = 0 \]

Node 3: \[ (l_0^{(3,1)} + l_0^{(3,2)}) - (l_0^{(1,3)} + l_0^{(2,3)}) = -1 \]

\[
\sum_{j: (i, j) \in E} l_{ij}^r - \sum_{j: (j, i) \in E} l_{ji}^r = \begin{cases} 
-1; i = dst, src \neq dst \\
1; i = src, src \neq dst \\
0; otherwise.
\end{cases}
\]
Optimization Model

• If a function 0 is placed at datacenter 1, then the path from the ingress switch 0 to egress switch 3 must include datacenter 1

\[ l_0^{(1,0)} + l_0^{(1,2)} + l_0^{(1,3)} \geq x_{0,1}^0 \]
Optimization Model

- If a function 0 is placed at datacenter 1, then the path from the ingress switch 0 to egress switch 3 must include datacenter 1

\[ l_0^{(1,0)} + l_0^{(1,2)} + l_0^{(1,3)} \geq x_{0,1}^0 \]

- If a function 1 is placed at datacenter 1, then the path from the ingress switch 0 to egress switch 3 must include datacenter 1

\[ l_0^{(1,0)} + l_0^{(1,2)} + l_0^{(1,3)} \geq x_{0,1}^1 \]
Optimization Model

• If a function 0 is placed at datacenter 1, then the path from the ingress switch 0 to egress switch 3 must include datacenter 1

\[ l^{(1,0)}_0 + l^{(1,2)}_0 + l^{(1,3)}_0 \geq x_{0,1}^0 \]

• If a function 1 is placed at datacenter 1, then the path from the ingress switch 0 to egress switch 3 must include datacenter 1

\[ l^{(1,0)}_0 + l^{(1,2)}_0 + l^{(1,3)}_0 \geq x_{0,1}^1 \]

\[ \sum_{(d,j) \in E} l^{d,j}_r \geq x^i_{r,d} \]
Optimization Model

- Variables $x_{r,d}^i$, $y_d^i$, $l_r^{i,j}$ are binary, integer, and real – NP hard
- For large instances of the problem, finding the optimal solution is not practical

\[
\begin{align*}
\text{max } F &= w_1 \sum_{r \in R} \sum_{i \in F_r} \sum_{d \in D} x_{r,d}^i - w_2 \sum_{d \in D} \sum_{i \in F_d} c_d^i y_d^i \\
&\quad - w_3 \sum_{r \in R} \sum_{(i,j) \in E} e^{i,j} l_r^{i,j} \\
\sum_{d \in D} x_{r,d}^i &\leq 1 \quad r \in R, i \in F_r \\
x_{r,d}^i &\leq y_d^i \quad r \in R, i \in F_r, d \in D | i \in F_d \\
\sum_{i \in F_d} w_d^i y_d^i &\leq w_{d,j} \quad d \in D, r \in R, j \in \{1, 2, \ldots, |W_d|\} \\
\sum_{r \in R} x_{r,d}^i &\leq \lambda_d^i y_d^i \quad d \in D, i \in F_d \\
\sum_{j: (i,j) \in E} l_r^{i,j} - \sum_{j: (j,i) \in E} l_r^{j,i} &= \begin{cases} 
-1; & i = \text{dst}_r, \ src_r \neq \text{dst}_r \\
1; & i = \text{src}_r, \ src_r \neq \text{dst}_r \\
0; & \text{otherwise}. \ i \in V, r \in R \\
\end{cases} \\
\sum_{(d,j) \in E} l_r^{d,j} &\geq x_{r,d}^i \quad r \in R, i \in F_r, d \in D | i \in F_d \\
x_{r,d}^i &\in \{0, 1\} \quad r \in R, i \in F_r, d \in D | i \in F_d \\
y_d^i &\in \mathbb{Z}^+ \quad d \in D, i \in F_d \\
l_r^{i,j} &\in \{0, 1\} \quad r \in R, (i,j) \in E
\end{align*}
\]
Greedy Approach

- Greedy approach based on Dijkstra algorithm

Algorithm 1: Greedy Routing and Placement of NFs

1. INPUT: \( G(V, E), e_{ij} \forall (i, j) \in E, R, F, D \)
2. OUTPUT: \( x_{r,d}, y_{d}, t_{ij}^{r} \) values
3. set \( x_{r,d} = 0, y_{d} = 0, t_{ij}^{r} = 0 \) for all \( r \in R, i \in F_r, d \in D, (i, j) \in E \)
4. for all \( r \in R \) do
5. \( D(r) = \{ \} \)
6. \( k = 1 \)
7. for all \( i \in F_r \) do
8. \( d_k = \) datacenter that implements \( i \) at minimum cost and has enough resources to serve an additional request
9. update resources of \( d_k \)
10. update \( y_{d_k} \)
11. set \( x_{r,d_k} = 1 \)
12. \( D(r) = D(r) \cup d_k \)
13. \( k = k + 1 \)
14. end for
15. end for
16. for all \( r \in R \) do
17. \( src = src_r \)
18. \( C(r) = \{ src \} \)
19. for \( k = 1 \) to \( |D(r)| \) do
20. \( dst = d_k \)
21. if \( d_k \in C(r) \) then
22. \( SP = \text{Dijkstra}(src, dst) \)
23. set \( t_{ij}^{r} = 1 \) for all link \( (i, j) \in SP \)
24. \( C(r) = C(r) \cup d_k \)
25. \( C(r) = C(r) \cup j \), for all datacenter \( j \in SP, j \in D(r) \)
26. end if
27. \( src = dst \)
28. end for
29. \( dst = dst_r \)
30. \( SP = \text{Dijkstra}(src, dst) \)
31. set \( t_{ij}^{r} = 1 \) for all \( (i, j) \in SP \)
32. end for
33. return \( x_{r,d}, y_{d}, t_{ij}^{r} \)

Placement of network functions, one request at a time

Routing of flows through datacenters implementing the functions, one request at a time
Numerical Examples

- The number of types of resources at a datacenter was set to three (e.g., RAM, storage, CPU)
- The amount of resources of a type at a datacenter is uniform in [.33, 300]
- There are five network functions; each datacenter implements three functions
- The amount of resources of a type needed for an instance of a function is uniform in [0, 100]
- The cost of instantiate a function is uniform in [0, 100]
- Datacenters were randomly located in the topology below
When there is a small number of datacenters (3) and multiple requests (15), ILP has a comparable performance to that of LP; deployment cost increases with the number of function per request.

The gap of the heuristic increases with the number of function per requests; finding the optimal solution requires the evaluation of a large number of combinations.

\[
\text{Gap} = \frac{Ov_{LP} - Ov_{alg}}{Ov_{LP}}
\]

where \(Ov_{LP}\) is the optimal value obtained with the LP scheme, and \(Ov_{alg}\) is the optimal value obtained with the ILP or greedy heuristic.
Numerical Example 1

- Deployment cost increases with the number of functions per request
• For LP and ILP, the increase in routing cost is mostly flat; i.e., when the number of datacenters is small, routing is ‘less important’, because the implementation of functions are concentrated in few datacenters
• When there is a large number of datacenters (11) and multiple requests (15), ILP continues to have a comparable performance to LP.
• Deployment cost increases substantially when the number of functions per request increases from 1 to 3. However, the increase in cost is minimal when the number of functions per request increases from 3 to 5; i.e., a single instance serves multiple requests without an increase of deployment of functions.
• For LP and ILP, the routing cost increases with the number of function per requests; i.e., when the number of datacenters is large, routing is ‘more important’, because the implementation of functions are dispersed in many datacenters.
Concluding Remarks

- We are currently working on an optimization scheme for the joint routing and placement of virtual network functions (NFs) problem.
- The proposed ILP maximizes the number of satisfied NFs while at the same time minimizes the deployment and routing costs.
- A heuristics and ILP are currently being tested.
- The implementation of the proposed schemes in small testbeds using ONOS SDN is being implemented.

THANK YOU
Numerical Example 3

- ILP and LP performances similar; ~2% gap
- As the number of request increases, the heuristic gap substantially increases; finding the optimal solution requires the evaluation of a large number of combinations

\[
\text{Gap} = \frac{OV_{LP} - OV_{alg}}{OV_{LP}}
\]

where \( OV_{LP} \) is the optimal value obtained with the LP scheme, and \( OV_{alg} \) is the optimal value obtained with the ILP or greedy heuristic.
• Deployment cost increases with the number of requests; ILP performance is comparable to that of LP – 'small' performance gap
Numerical Example 3

- Routing cost increases with the number of requests; ILP performance is comparable to that of LP – 'small' performance gap
- While the gap of the routing cost of the greedy approach decreases with the number of requests, the number of satisfied requests is mostly flat
Introduction

About Linear Programming

• Many of the problems for which we want algorithms are *optimization* tasks

• Optimization tasks seek a solution that (1) satisfies certain constraints and (2) is the best, with respect to a criterion

• *Linear programming* describes a broad class of optimization tasks in which both the constraints and the optimization criterion are *linear functions*
About Reductions

- Sometimes a computational task is sufficiently general that any subroutine for it can also be used to solve a variety of other tasks, which at glance might seem unrelated.
- Once we have an algorithm for a problem, we can use it to solve other problems.