Network Functions Virtualization (NFV)

Class 11

Danny Raz
Last week

- Resource allocation
- Placement

This week

- Continue Placement
- Performance and placement
Resource allocation in the Cloud

Key feature: cost effectiveness ➔ resource management

- **Where to acquire resources (CPU, Storage)?**
  - building the next data center (Google)
  - getting EC2 resources and how much (smaller users)

- **Where to place the service and the data?**
  - VM placement
  - service/data migration

- **Which location should serve a specific request?**
  - Load balancing
Placement of Network Functions - A Model

**Input**
- A set of flows, each with a path and a demand for each of the possible network functions.
- A set of datacenters locations, each with a size.
- A set of network functions realizations, each with capacity (amount of clients to be served), size, and establishment cost.

**Output**
- A placement of copies of the realization of the network functions and a rerouting of the flow into the DCs.

**Such that:** The demand for each flow and for each function is satisfied, the size constraints are met, and the overall cost is minimal.
**Input**

- A set of flows, each with a path and a demand for each of the possible network functions.
- A set of datacenters locations, each with a size.
- A set of network functions realizations, each with capacity (amount of clients to be served), size, and establishment cost.

**Output**

- A placement of copies of the realization of the network functions and a rerouting of the flow into the DCs.

**Such that:** The demand for each flow and for each function is satisfied, the size constraints are met, and the overall cost is minimal.
We have to think about the MODEL where the goal is to optimize REAL SYSTEMS
How good is this model?

- **Service chaining example**
  - CPE – FW – DPI

- **Can we use the previous model for function placement in this case?**

- **Can we find a better model?**
How good is this model?

• The order of the functions (per flow) is given

• No pre-defined paths
Service chain model – take 2

• Given
  - Set of services
  - Set of demands

• Find
  - Function placement
  - Flow routing
  - Cloud resource allocation
  - Network resource allocation

• Such that
  - Demands are satisfied
  - Overall operational cost is minimized
cloud network flow
cloud network flow
cloud network flow
cloud network flow
cloud network flow
cloud network flow
cloud network flow
cloud network flow
A network service $\phi \in \Phi$ is described by a chain of $M_\phi$ virtual network functions (VNFs).

- $(\phi, i)$ denotes the $i$-th function of service $\phi$.
- $(d, \phi, i)$ denotes the output of the $i$-th function of service $\phi$ for destination $d$.
- Function $(\phi, i)$ has resource requirement $r^{(\phi,i)}$ processing resource units per flow unit, scaling factor $\xi^{(\phi,i)}$ output flow units per input flow unit.
Service chain model – take 2

\[
\begin{align*}
\min & \quad \sum_{(u,v)} w_{uv} y_{uv} \\
\text{s.t.} & \quad \sum_{v \in \delta^-(u)} f_{v,u}^{(d,\phi,i)} = \sum_{v \in \delta^+(u)} f_{u,v}^{(d,\phi,i)} \quad \forall u, d, \phi, i \\
& \quad f_{p(u),u}^{(d,\phi,i)} = \xi(\phi,i) f_{u,p(u)}^{(d,\phi,i-1)} \quad \forall u, d, \phi, i \\
& \quad \sum_{(d,\phi,i)} f_{u,v}^{(d,\phi,i)} x_{v}^{(\phi,i+1)} \leq y_{uv} \leq c_{uv} \quad \forall (u, v) \\
& \quad f_{s(u),u}^{(d,\phi,0)} = \chi_u^{(d,\phi)} \quad \forall u, d, \phi \\
& \quad f_{u,p(u)}^{(d,\phi,M_{\phi})} = 0 \quad \forall d, \phi, u \neq d \\
& \quad f_{u,v}^{(d,\phi,i)} \geq 0, y_{uv} \in \mathbb{Z}^+ \quad \forall (u, v), d, \phi, i
\end{align*}
\]

Cost Function

Combined Flow Conservation

Service Chaining

Capacity

Sources and Demands

Fractional flows

Integer resources
Service chain model – take 2

\[
\begin{align*}
\text{min} & \quad \sum_{(u,v)} w_{uv} y_{uv} \\
\text{s.t.} & \quad \sum_{v \in \delta^-(u)} f_{uv}^{(d,\phi,i)} = \sum_{v \in \delta^+(u)} f_{uv}^{(d,\phi,i)} \quad \forall u, d, \phi, i \\
& \quad f_{p(u),u}^{(d,\phi,i)} = \xi^{(\phi,i)} f_{u,p(u)}^{(d,\phi,i-1)} \quad \forall u, d, \phi, i \\
& \quad \sum_{(d,\phi,i)} f_{uv}^{(d,\phi,i)} y_{uv}^{(\phi,i+1)} \leq c_{uv} \quad \forall (u, v) \\
& \quad f_{s(u),u}^{(d,\phi,0)} = \lambda_{u}^{(d,\phi)} \quad \forall u, d, \phi \\
& \quad f_{u,a(u)}^{(d,\phi,M_{\phi})} = 0 \quad \forall d, \phi, u \neq d \\
& \quad f_{uv}^{(d,\phi,i)} \geq 0, \ y_{uv} \in \mathbb{Z}^+ \quad \forall (u, v), d, \phi, i
\end{align*}
\]

Cost Function

Combined Flow Conservation

Service Chaining

Capacity

Sources and Demands

Fractional flows

Integer resources

Smaller Resource Granularity

Dedicated boxes

Commodity servers

Virtual machines

Containers

Grains
There is a fast approximation algorithm for the fractional NSDP that produces an $\varepsilon$ approximation solution in time $O(m^2nL/\varepsilon)$.

Use dynamic evolution of underlying queuing system to construct an iterative approximation to original static problem.

How good is this model?

- Previous models address placement in node (DC) granularity
- How about physical host granularity?
- Placement of VNF VMs in the physical hosts

Source: ETSI Ongoing PoC
Consider the following set of physical servers (a), and sequence of service chaining (b), each with a specified amount of traffic to be processed:
Motivation: placement (scheduling) example

Consider the following set of physical servers (a), and sequence of service chaining (b), each with a specified amount of traffic to be processed:

(a) Set of servers

(b) Sequence of service chaining
Motivation: placement (scheduling) example

Consider the following set of physical servers (a), and sequence of service chaining (b), each with a specified amount of traffic to be processed:

(b) Sequence of service chaining
Motivation: placement (scheduling) example

But, how should they be placed?
Motivation: placement (scheduling) example

But, how should they be placed?

Given sequence of chains:

Consider the following simplified placement extremes:

Distribute each chain between servers

Gather each chain on a specific server
Motivation: benefits & drawbacks of placement strategies

But, how should they be placed?

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Distributing VNFs</th>
<th>Gather VNFs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Networking traffic</td>
<td>Same subnet traffic (aggregate)</td>
<td>Reduced subnet traffic (split)</td>
</tr>
<tr>
<td>Availability level</td>
<td>0% available</td>
<td>66% available</td>
</tr>
<tr>
<td>(independent server failures)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware utilization</td>
<td>Better utilization of specialized hardware (if such exists)</td>
<td>Balanced usage of common HW</td>
</tr>
<tr>
<td>Migration/state management</td>
<td>External state transfer</td>
<td>Internal state transfer</td>
</tr>
<tr>
<td>(stateful VNFs)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Switching</td>
<td>Network switching capabilities</td>
<td>Virtual switching cost</td>
</tr>
</tbody>
</table>
Motivation: benefits & drawbacks of placement strategies

But, how should they be placed?
Optimizing the cost of virtual switching for service chaining

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Distributing VNFs</th>
<th>Gather VNFs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Networking traffic</td>
<td>Same subnet traffic (aggregate)</td>
<td>Reduced subnet traffic (split)</td>
</tr>
<tr>
<td>Availability level</td>
<td>0% available</td>
<td>66% available</td>
</tr>
<tr>
<td>(independent server failures)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware utilization</td>
<td>Better utilization of specialized hardware (if such exists)</td>
<td>Balanced usage of common HW</td>
</tr>
<tr>
<td>Migration/state management</td>
<td>External state transfer</td>
<td>Internal state transfer</td>
</tr>
<tr>
<td>(stateful VNFs)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Switching</td>
<td>Network switching capabilities</td>
<td>Virtual switching cost</td>
</tr>
</tbody>
</table>
Software switching
Software switches are designed to allow virtual machines to communicate. Few exemplary implementations of software switching includes:

1. **Linux Bridge**: switching in layer 2 bridge (integrated into Kernel 2.6)
2. **IP forwarding**
3. **Open vSwitch (OVS)**: open-source implementation of a virtual multilayer switch
   - Integrated into Linux Kernel 3.3
   - Multiple hypervisor support (e.g., Xen, KVM)
   - Multiple protocol support (e.g., OpenFlow, sFlow)
   - Integrated into OpenStack
   - Support DPDK acceleration
   - OpenFlow enabled
   - Flow-based forwarding
   - Control plane in user space

Recall our motivating example: given arbitrary set of service chains with different size and traffic requirements, develop scheduling (placement) algorithm that optimizes the cost of virtual switching.
Software switching: problem definition

Recall our motivating example: given arbitrary set of service chains with different size and traffic requirements, develop scheduling (placement) algorithm that optimizes the cost of virtual switching.

We want to define a model that for a given server captures the cost and limitations of virtual switching.

(2) Cost model for software switching:
- Physical interface networking stack
- Virtual interface networking stack
- OVS forwarding resolution
- Virtual machine workload
- Traffic requirements (amount, size)
- Hardware acceleration

(1) Outline the feasible boundaries:
- Network interface limits: line rate
- Servers limits: CPU and memory (physical)
- Configuration: isolation and CPU pinning
Cost model for software switching
- Physical interface networking stack
- Virtual interface networking stack
- OVS forwarding resolution
- Virtual machine workload
- Traffic requirements (amount, size)
- Hardware acceleration

Outline the feasible boundaries
- Network interface limits: line rate
- Servers limits: CPU and memory (physical)
- Configuration: isolation and CPU pinning

Measuring feasible boundaries

Recall our motivating example: given arbitrary set of service chains with different size and traffic requirements, develop scheduling (placement) algorithm that optimizes the cost of virtual switching.

We want to define a model that gives a server captures the cost and limitations of virtual switching.
We conduct an extensive and in-depth evaluation that examines the impact of service chain deployment on our placement strategies (gather and distribute).

**Environment – hardware**

- Server: ProLiant DL380p Gen8
  - 2 sockets: each Xeon(R) CPU E5-2697 (12 cores)
  - Intel 82599ES 10-Gigabit NIC
  - 2 NUMA of 12 banks (each is 16GB – total 384GB)
- Hyperthreading & turboboost: disabled
- Isolation: 4-12 (HV), 20-12 (VMs)

**Environment – software**

- Host (CentOS 3.10); Guest (Fedora 4.0.4)
- VM pinning
- Open vSwitch: 2.3.1
- TCP optimizations (offloading): disabled
- RSS and irqbalance:
  - queues set up according to kernel CPU

**Metrics and tools**

- Evaluated metrics: bandwidth, CPU utilization, packet processing capabilities, and response time
- Tools: sar, ping, sockperf (traffic generator), Linux counters
BW - single server

Note that up until a certain point, the VM’s BW is the bottleneck. From that point on, the OVS is the main factor.
BW – many servers

The difference between gather and distribute deployment might be as high as 50%!

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Gather</th>
<th>Distribute</th>
<th>CPU isolation</th>
<th>VM pinning</th>
<th>NIC offloading</th>
<th>Number of VMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment</td>
<td></td>
<td></td>
<td>10</td>
<td>On (10-23)</td>
<td>Off</td>
<td>{1 ... 50}</td>
</tr>
<tr>
<td>Flows</td>
<td></td>
<td>Distributed (50 flows)</td>
<td></td>
<td>On</td>
<td>Off</td>
<td></td>
</tr>
<tr>
<td>Packet size</td>
<td>100</td>
<td>1500</td>
<td>NIC offloading</td>
<td>On</td>
<td>Off</td>
<td></td>
</tr>
<tr>
<td>OVS mode</td>
<td>user</td>
<td>kernel</td>
<td>Tuning RSS</td>
<td>On</td>
<td>On</td>
<td></td>
</tr>
</tbody>
</table>

(a) 100 Byte Packet
Distribute is bounded by the wire saturation

(b) 1500 Byte Packet

50%
CPU usage – single server

Parameters

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Gather</th>
<th>Distribute</th>
<th>CPU isolation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flows</td>
<td></td>
<td>Distributed (50 flows)</td>
<td>8</td>
</tr>
<tr>
<td>VM pinning</td>
<td>Off (10-23)</td>
<td>On</td>
<td>Off</td>
</tr>
<tr>
<td>NIC offloading</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
</tr>
<tr>
<td>OVS mode</td>
<td>user</td>
<td>kernel</td>
<td>On On</td>
</tr>
<tr>
<td>Number of VMs</td>
<td>{1 ... 50}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The OVS has a bigger order of packets to process (square) compared to the VM, resulting in competition over resources.

The OVS packet processing workload is smaller, resulting in limitation caused by other resources (no CPU saturation).
CPU isolation: latency

The effect on performance when allocating dedicated cores to OVS (CPU isolation).

The less CPU is isolated, the lower latency (better) we observe!
More insights: correlating CPU and BW (gather)

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Gather</th>
<th>Distribute</th>
<th>CPU isolation</th>
<th>VM pinning</th>
<th>NIC offloading</th>
<th>Tuning RSS</th>
<th>Number of VMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment</td>
<td></td>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flows</td>
<td></td>
<td>Distributed (50 flows)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Packet size</td>
<td>100</td>
<td>1500</td>
<td></td>
<td>Off</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NIC offloading</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OVS mode</td>
<td>user</td>
<td>kernel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tuning RSS</td>
<td>On</td>
<td>On</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of VMs</td>
<td></td>
<td></td>
<td>{1 ... 50}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(a) OVS CPU utilization

(b) VMs CPU utilization

(c) Bandwidth

More OVS CPU Isolation

Implies

More Bandwidth
More insights: correlating CPU and BW (distribute)

Caggiani Luizelli, Raz, Saar and Yallouz, "The Actual Cost of Software Switching for NFV Chaining", IM '17.
Recall our motivating example: given arbitrary set of service chains with different size and traffic requirements, develop scheduling (placement) algorithm that optimizes the cost of virtual switching.

We want to define a model that given server captures the cost and limits of virtual switching.

(2) **Cost model for software switching**
- Physical interface networking stack
- Virtual interface networking stack
- OVS forwarding resolution
- Virtual machine workload
- Traffic requirements (amount, size)
- Hardware acceleration

(1) **Outline the feasible boundaries**
- Network interface limits: line rate
- Servers limits: CPU and memory (physical)
- Configuration: isolation and CPU pinning
Cost model for software switching

Our goal is to find a model that captures the cost of software switching:

- Must incorporate many parameters:
  - VM workload and sub-chain settings
  - Networking stack (physical and virtual)
  - OVS capabilities (and type)
  - Traffic requirements

We want to break our problem to small and measurable building blocks.
Cost model for software switching

Our goal is to find a model that captures the cost of software switching:

\[
C = \sum_{q=1}^{r} F_g(\varphi^q [1 \ldots i_q]) + \frac{r - 1}{r} \cdot \sum_{q=1}^{r} (\|F_g(\varphi^q [1 \ldots i_q])\|^r) - r \cdot F_g(\varphi^q [1 \ldots i_q])
\]

- sub-chain \( \varphi^q \)
- cpu-cost associated with deploying \( \varphi^q \) on a server (gather)
- cpu-cost associated with deploying \( \varphi^q \), \( r \) times on a server

Open vSwitch
Cost model for software switching

Our goal is to find a model that captures the **cost of software switching**:

\[
C = \sum_{q=1}^{r} F_g(\phi^q[1 \ldots i_q]) + \frac{r - 1}{r} \cdot \sum_{q=1}^{r} \left( \|F_g(\phi^q[1 \ldots i_q])\| - r \cdot F_g(\phi^q[1 \ldots i_q]) \right)
\]

Roughly,

- The left hand side captures the aggregated gather cost of each of the sub-chains deployed on the server

 sub-chain \( \phi^q \)

 cpu-cost associated with deploying \( \phi^q \) on a server (gather)

 cpu-cost associated with deploying \( \phi^q \), \( r \) times on a server

[Diagram of network with Open vSwitch]
Cost model for software switching

Our goal is to find a model that captures the cost of software switching:

\[ C = \sum_{q=1}^{r} F_g(\varphi^q[1 \ldots i_q]) + \frac{r - 1}{r} \cdot \sum_{q=1}^{r} \left( \|F_g(\varphi^q[1 \ldots i_q])\|^r \right) - r \cdot F_g(\varphi^q[1 \ldots i_q]) \]

Roughly,

- The left hand side captures the aggregated gather cost of each of the sub-chains deployed on the server.
- The right hand side captures the effort in distribution between the sub-chains which is measured by the distance between deploying \( r \) sub-chains and deploying a single sub-chain \( r \) times.
Cost model for software switching: validating results
NFV placement algorithm
NFV placement algorithm

Given service chain $\phi$ with $N$ VNFs, how should we deploy over $K$ servers?
- What policy should we follow?

$\phi = \phi_1 \phi_2 \ldots \phi_a \phi_{a+1} \ldots \phi_{N-1} \phi_N$

Server 1  Server 2  ...  Server $q$  ...  Server $K$
NFV placement algorithm: openstack/nova

Given service chain $\phi$ with $N$ VNFs, how should we deploy over $K$ servers?
- Nova-scheduler ignores chains and per VNFsquires all servers and selects one
- Global policy can be either load-balancing or save-energy (roughly distribute or gather)
- Note that in practice $N \ll K$, i.e. $N$ is constant, while $K$ can scale

Runtime analysis of $O(K^N)$ (when including anti/affinity considerations)

\[ \phi = \phi_1 \quad \phi_2 \quad \ldots \quad \ldots \quad \phi_\alpha \quad \phi_\alpha+1 \quad \ldots \quad \ldots \quad \phi_{N-1} \quad \phi_N \]

Server 1  Server 2  \ldots  Server $q$  \ldots  Server $K$
NFV placement algorithm: OCM optimal

Given service chain $\varphi$ with $N$ VNFs, how should we deploy over $K$ servers?

- Our approach focuses on iterating over all possible decompositions of the chain.
- All possible decompositions are captured by “bell number” -- $O(N^N)$ iterations.
- Each decomp is assigned to a server by reducing bi-partite matching to min-cost flow.

Runtime analysis of $O(N^N)$ iterations of $O(N^3K^2 + N^2K^3)$ min-cost flow.

$$\varphi = \varphi_1 \varphi_2 \ldots \varphi_i \ldots \varphi_a \varphi_{a+1} \ldots \varphi_{a+j} \ldots \varphi_{N-1} \varphi_{N-1} \varphi_N$$

Bi-partite matching problem between partitions and servers $O(N^3K^2 + N^2K^3)$.
Given service chain $\varphi$ with $N$ VNFs, how should we deploy over $K$ servers?

- A reasonable assumption is that a service chain enters and exits a server at most once.
- Now possible decompositions are captured by "integer composition" -- $O(2^N)$ iterations.
- Each decomp is assigned to a server by reducing bi-partite matching to min-cost flow.

Runtime analysis of $O(2^N)$ iterations of $O(N^3K^2 + N^2K^3)$ min-cost flow.

$$\varphi = \begin{array}{llllll}
\varphi_1 & \varphi_2 & \ldots & \varphi_i & \varphi_a & \varphi_{a+1} & \varphi_{a+j} & \varphi_{N-1} & \varphi_{N-1} & \varphi_N
\end{array}$$

bi partite matching problem between partitions and servers $O(N^3K^2 + N^2K^3)$
NFV placement algorithm: OCM (Operational Cost Minimization)

**Input:** \( S := \langle S_1 \rightarrow S_2 \ldots \rightarrow S_k \rangle \) : set of servers
\( \varphi := \langle \varphi_1 \rightarrow \varphi_2 \ldots \rightarrow \varphi_n \rangle \) : a service chain

1) \( \text{minCost} := \infty \) : minimum cost found so far
2) \( \text{deployMap} := \text{NIL} \) : maps all VNF \( \{\varphi_1, \varphi_2, \ldots, \varphi_n\} \) to a server \( S_j \in S \)
3) \( A := \{a_1, a_2, \ldots, a_m\} \) : all partition of \( \varphi \)
4) **for** every partition \( a \in A \)
   4a) \( C_j^i := \text{build function that measures deploying every part } \varphi_i \in a \text{ on every server } S_j \in S \)
   4b) \( G := \text{build reduction to minimum cost flow in a graph} \)
   4c) **if** \( \text{min}(G) \leq \text{minCost} \)
      update \( \text{minCost} \) and \( \text{deployMap} \)
5) **return** deployMap

3) **enumerating all partitions**
4a) follows the **cost model for software switching**
4b) reduction to **minimum cost flow**
NFV placement algorithm: evaluation

Analysis comparing existing openstack/nova solution to OCM

\[ \Omega: O(N^3 K^2 + N^2 K^3) \]
NFV placement algorithm: evaluation

Operational cost for requests that arrives (over time...)
- 200 servers
  - Each has 24 cores
- 700 chain requests
  - Requires 100K per request
  - Zero one property
  - Random size from 2 to 10
  - Each VNF requires 2 cores

---

**Kernel**

![Graph 1](image1.png)

**DPDK**

![Graph 2](image2.png)
The average operational cost (over time) for requests with different traffic requirements

- 200 servers
  - Each has 24 cores
- 700 chain requests
  - Zero one property
  - Random size from 2 to 10
  - Each VNF requires 2 cores
Summary and future work
Summary and future work

- Motivation: placement focusing on optimizing the operational cost (virtual switching)
- Measuring feasible boundaries
- NFV placement algorithm: OCM
  - enumerate all sub chain partitions
  - build cost function based on the cost model for software switching
  - reducing matching sub-chains to servers, to minimum cost flow
- Integrate OCM in OpenStack
  - Evaluate the performance in a real NFV-based deployment
- Analyze the effect of other parameters on the system: number of flows, DPDK/SR-IOV
- Non linear service function chaining
- Incorporate other considerations (e.g. latency, what does user want to optimize...
So what did we do today?

- Placement of NF chaining
  - second model
- Is it good enough
  - to be used in practice
- A different model and more on implementation details and performance

Next week:
- Future directions (yapfi)
- Student talks
- Recap
Thank you