NetSlicer:
Automated and Traffic-Pattern Based Application Clustering in Datacenters

Liron Schiff ¹, Ofri Ziv ¹, Manfred Jaeger ² and Stefan Schmid ²,³

¹GuardiCore, Israel  ²AAU, Denmark  ³Univie, Austria
Agenda

- The problem
- Our ML approach
- Challenges we encountered
- The NetSlicer algorithm
- Evaluation
- Use case
The network is the sum of its applications

- a (virtual/physical) machine
- an internet host
- an instance at remote cloud
The network is the sum of its applications

Grouping nodes is necessary for visibility and policy creation
Leveraging existing data to group nodes

Many definitions, many sources

- Orchestration (e.g., tags, names, desc.)
- Inventory (e.g., CMDB)
- DevOps (e.g., puppet recipes, ansible scripts)
Leveraging existing data to group nodes

Many definitions, many sources

- Orchestration (e.g., tags, names, desc.)
- Inventory (e.g., CMDB)
- DevOps (e.g., puppet recipes, ansible scripts)

Limited by

- Availability (not always defined)
- Relevance (may not be up to date)
- Consistency (may not be standardized)
Machine Learning Based Labeling
Cluster nodes into **tiers**

**Tier** := a set of nodes with similar connections
Machine Learning Based Labeling

- Cluster nodes into **tiers**
- Cluster tiers into **apps**

**Tier** := a set of nodes with similar connections

**App** := a set of tightly coupled tiers
Machine Learning Based Labeling

- Cluster nodes into **tiers**
- Cluster tiers into **apps**
- Identify roles (Infras, services)

**Tier** := a set of nodes with similar connections

**App** := a set of tightly coupled tiers

**Role** := a functionality type (may repeat across the network)
Machine Learning Based Labeling

- Cluster nodes into **tiers**
- Cluster tiers into **apps**
- Identify roles (Infras, services)
- Generate (simple) labels
  - Based on services
  - Based on machine names
Example - ungrouped
Example - grouped by application

* Exclude: Infra
Why is it challenging?
Why is it challenging?

- Load balancing
  - Same tier nodes are not “identical”
  - Different servers may serve different clients
Why is it challenging?

- Load balancing
- Infras and monitors
  - Add “noisy” connections
  - Over-connected network
Why is it challenging?

- Load balancing
- Infras and monitors
- Non-standard ports
  - Customized and random port numbers
  - Harder to identify apps and filter noise

---

### Services

<table>
<thead>
<tr>
<th>Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificate Services (CertSvc)</td>
</tr>
<tr>
<td>Distributed File System (Dfs)</td>
</tr>
<tr>
<td>Event Log</td>
</tr>
<tr>
<td>Exchange Server</td>
</tr>
<tr>
<td>Fax Service</td>
</tr>
<tr>
<td>File Replication</td>
</tr>
<tr>
<td>Local Security Authority (LSASS)</td>
</tr>
<tr>
<td>Netlogon</td>
</tr>
<tr>
<td>Remote Storage</td>
</tr>
<tr>
<td>Terminal Services</td>
</tr>
</tbody>
</table>

### Protocols

<table>
<thead>
<tr>
<th>Protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCOM</td>
</tr>
<tr>
<td>FTP</td>
</tr>
<tr>
<td>HADOOP</td>
</tr>
<tr>
<td>RPC</td>
</tr>
</tbody>
</table>

Why is it challenging?

- Load balancing
- Infras and monitors
- Non-standard ports
- Uncovered nodes
  - Missing connection / L7 data
Why is it challenging?

- Load balancing
- Infras and monitors
- Non-standard ports
- Uncovered nodes
  - Missing connection / L7 data
Why is it challenging?

- Load balancing
- Infras and monitors
- Non-standard ports
- Uncovered nodes
- Big data
  - Tens of thousands of nodes (and millions of links)
  - (almost) unlimited external internet hosts
Introducing the NetSlicer Algorithm
Introducing the NetSlicer Algorithm

- Start with an annotated graph
  - each link represents connections at a single port
Introducing the NetSlicer Algorithm

Merge nodes at 3 phases:
Introducing the NetSlicer Algorithm

Merge nodes at 3 phases:

- Detect “clusters”
  - Strongly connected components
Introducing the NetSlicer Algorithm

Merge nodes at 3 phases:

- Detect “clusters”
  - Strongly connected components
- Detect tiers
  - based on neighborhood similarity
- Detect applications
  - based on weighted connectivity
Detecting Tiers

- Not directly connected
A customized similarity measure

- Endpoint definition \((u,p,t) \in \Phi:\)
  - \(u\): a node id
  - \(p\): a service port number
  - \(t\): type (\(\mathbb{C}\):client / \(\mathbb{S}\):server)

<table>
<thead>
<tr>
<th>node id</th>
<th>connected endpoints</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>((a,80,\mathbb{C}), (b,80,\mathbb{C}), (d,21,\mathbb{S}), (e,21,\mathbb{S}))</td>
</tr>
<tr>
<td>y</td>
<td>((c,23,\mathbb{C}), (b,80,\mathbb{C}), (e,21,\mathbb{S}), (e,21,\mathbb{S}))</td>
</tr>
<tr>
<td>b</td>
<td>((x,80,\mathbb{S}), (y,80,\mathbb{S}))</td>
</tr>
</tbody>
</table>
A customized similarity measure

- **Endpoint definition** \((u,p,t) \in \Phi:\)**
  - \(u\): a node id
  - \(p\): a service port number
  - \(t\): type (\(\odot\):client / \(\oplus\):server)

- **Weight function** \(W_v: \Phi \rightarrow \mathbb{R}^+\)
  - set weights to endpoint with respect to \(v\)
  - E.g., \(W_v(\phi) = 1\) if \(\phi\) connected with \(v\), o.w. 0

<table>
<thead>
<tr>
<th>node id</th>
<th>connected endpoints</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>((a, 80, \odot), (b, 80, \odot), (d, 21, \oplus), (e, 21, \oplus))</td>
</tr>
<tr>
<td>y</td>
<td>((c, 23, \odot), (b, 80, \odot), (f, 21, \oplus), (e, 21, \oplus))</td>
</tr>
<tr>
<td>b</td>
<td>((x, 80, \oplus), (y, 80, \oplus))</td>
</tr>
</tbody>
</table>
A customized similarity measure

- Similarity by normalized dot product:
  \[ Sim(u, v) = \sum_{\phi \in \Phi} W_u(\phi)W_v(\phi)/(|W_u| \cdot |W_v|), \]
  \[ |W_x| = \sqrt{\sum_{\phi \in \Phi} W_x(\phi)^2}. \]

<table>
<thead>
<tr>
<th>Node ID</th>
<th>Connected Endpoints</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>(a, 80, ⊙), (b, 80, ⊙), (d, 21, ⊙), (e, 21, ⊙)</td>
</tr>
<tr>
<td>y</td>
<td>(c, 23, ⊙), (b, 80, ⊙), (f, 21, ⊙), (e, 21, ⊙)</td>
</tr>
<tr>
<td>b</td>
<td>(x, 80, ⊙), (y, 80, ⊙)</td>
</tr>
</tbody>
</table>
A customized similarity measure

- Similarity by normalized dot product:
  \[ \text{Sim}(u, v) = \sum_{\phi \in \Phi} W_u(\phi)W_v(\phi) / (|W_u| \cdot |W_v|), \]
  where \( |W_X| = \sqrt{\sum_{\phi \in \Phi} W_X(\phi)^2} \).

- Consider noisiness of endpoints
  - \( \text{noise}(\phi) = \max_{u,v} (1 - \text{Sim}(u,v)) \)
  - \( W_v(\phi) = (1 - \text{noise}(\phi))^\alpha \)
A customized similarity measure

- Similarity by normalized dot product:
  - \( Sim(u, v) = \sum_{\phi \in \Phi} \frac{W_u(\phi)W_v(\phi)}{|W_u| \cdot |W_v|}, \)
  - where \( |W_x| = \sqrt{\sum_{\phi \in \Phi} W_x(\phi)^2}. \)

- Consider noisiness of endpoints
  - \( \text{noise}(\phi) = \max_{u,v} (1 - Sim(u,v)) \) connected to \( \phi \)
  - \( W_v(\phi) = (1-\text{noise}(\phi))^\alpha \)

- Similarity and noise are computed in several rounds
An iterative process

- computes similarity and noise
- merges similar nodes
- stops when no similar pair found

Detecting Tiers

DC-01:389 is noisy

DataProc-db:27017 is not noisy
Detecting Applications

- From tiers to applications
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links

```
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links

```

```
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links

```

```
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links

```

```
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links

```

```
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links

```

```
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links

```

```
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links

```

```
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links

```

```
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links

```

```
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links

```

```
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links

```

```
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links

```

```
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links

```

```
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links

```

```
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links

```
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links
  - Ignore weak/infra links
Detecting Applications

- From tiers to applications
  - Start from graph of tier nodes
  - Assign weights to (merged) links
  - Ignore weak/infra links
  - Group connected components
Implementation

- Programed in Python
- Parallel design
- Optional initial noise configuration
  - for better and faster results
  - e.g., super noisy ports and nodes
- Modular
  - any clustering alg. for tiers and apps (given similarity values)
Evaluation

Datasets:
- 3 datacenter networks
- monitored at core VMs
- ground truth for monitored
- available online

<table>
<thead>
<tr>
<th></th>
<th>net1</th>
<th>net2</th>
<th>net3</th>
</tr>
</thead>
<tbody>
<tr>
<td>dominat OS</td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#nodes</td>
<td>3003</td>
<td>2507</td>
<td>4588</td>
</tr>
<tr>
<td>#monitored</td>
<td>153</td>
<td>143</td>
<td>174</td>
</tr>
<tr>
<td>#unmonitored</td>
<td>2850</td>
<td>2364</td>
<td>4414</td>
</tr>
<tr>
<td>ground-truth apps</td>
<td>40</td>
<td>58</td>
<td>85</td>
</tr>
</tbody>
</table>

Distribution of nodes to app sizes
Evaluation

- Datasets:
  - 3 datacenter networks
  - monitored at core VMs
  - ground truth for monitored
  - available online

- Compared with:
  - louvain modularity
  - node2vec (+ HAC/HDBSCAN)
  - and mixtures

- Scoring:
  - Adjusted Random Index (ARI) from ground truth

---

<table>
<thead>
<tr>
<th></th>
<th>net1</th>
<th>net2</th>
<th>net3</th>
</tr>
</thead>
<tbody>
<tr>
<td>dominat OS</td>
<td>🌐</td>
<td></td>
<td>🌐</td>
</tr>
<tr>
<td>#nodes</td>
<td>3003</td>
<td>2507</td>
<td>4588</td>
</tr>
<tr>
<td>#monitored</td>
<td>153</td>
<td>143</td>
<td>174</td>
</tr>
<tr>
<td>#unmonitored</td>
<td>2850</td>
<td>2364</td>
<td>4414</td>
</tr>
<tr>
<td>ground-truth apps</td>
<td>40</td>
<td>58</td>
<td>85</td>
</tr>
</tbody>
</table>

Distribution of nodes to app sizes
Results

*NetSlicer

Analyst Compatible
Use case: Microsegmentation Workflow
Use case: Microsegmentation Workflow

NetSlicer

aggregated connections
Use case: Microsegmentation Workflow

NetSlicer

aggregated connections

Label suggestions

User reviews labels

DB: MongoDB

App: Ecomm

Infra: AD

App: DataProc

DB: MongoDB
Use case: Microsegmentation Workflow

- **NetSlicer**
  - Aggregated connections
  - Label suggestions

- **User reviews labels**
  - User defines label based security policy

- **DB:** MongoDB
- **App:** Ecomm, DataProc
- **Web:** apache
- **Infra:** AD
Use case: Microsegmentation Workflow

- **NetSlicer**: Aggregated connections
- **Label suggestions**: User reviews labels
- **User defines label based security policy**: Labels
- **Policy rules**

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ecomm.apache</td>
<td>Ecomm.Mongo:27018</td>
<td>allow</td>
</tr>
<tr>
<td>Ecomm</td>
<td>AD:389</td>
<td>allow</td>
</tr>
<tr>
<td>Users</td>
<td>DataProc:80</td>
<td>allow</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The NetSlicer Algorithm

- Customized similarity
- Considers real life scenarios
- Promising initial results
- Parallel and modular design
Future work

- More datasets
- Clustering processes
  - (multiple apps per server)
- Mixing with more clustering algs.
- Convergence time bounds
Questions?

datasets: https://www.guardicore.com/labs/datacenter-traces/
more works: https://www.guardicore.com/labs/research-academic/
me: liron.schiff@guardicore.com
Architecture

Virtual Servers
- Hypervisor Collector
- Agent

Bare Metal
- SPAN / TAP / NPB
- Agent
- SPAN Collector

Public Cloud
- Agent
- AWS

Containers
- Agent
- OpenShift
- Subetones

Guest

Infrastructure

security policy

Aggregation

Management and Deception
SaaS / On-Prem