ENVI:
Elastic resource flexing for Network functions Virtualization

* Lianjie Cao, Purdue University
Puneet Sharma, Hewlett Packard Labs
Sonia Fahmy, Purdue University
Vinay Saxena, Hewlett Packard Enterprise

* This work was funded by Hewlett Packard Labs and done during internship program.
Background
Network Functions Virtualization

Proprietary hardware
IPS/IDS  WAN Accelerator  Traffic Manager

Virtualization Platform
Virtualization
Cloudification

Service Function Chaining

Flexibility  Scalability  Elasticity  Agility

Auto Resource Flexing

CAPEX & OPEX Reduction
VNF Resource Flexing Example

HTTP caching proxy - Squid
VNF Resource Flexing Example

HTTP caching proxy - Squid
VNF Resource Flexing Example

HTTP caching proxy - Squid
VNF Resource Flexing Example

HTTP caching proxy - Squid
VNF Resource Flexing Example

Potential scaling point

Scaling up

Scaling out

HTTP caching proxy - Squid
Related Work

• Instance scaling detection
  • Low level infrastructure metrics: CPU, memory, network usages
  • Static rule-based policy: scale out if CPU > 80% ...

• Resource flexing
  • Simple scaling: E2@SOSP’15, Stratos
  • Traffic patterns assumption: CloudScale@SOCC’11, DejaVu@ASPLOS’12
  • Long term learning: DejaVu@ASPLOS’12

• Service function chaining
  • Interdependence across VNFs is largely ignored
VNF Scaling Detection

Performance tests on HTTP caching proxy Squid (using NFV-VITAL@NFV-SDN’15)
VNF Scaling Detection

Type I workload

Type II workload

Performance tests on HTTP caching proxy Squid (using NFV-VITAL@NFV-SDN’15)
VNF Scaling Detection

Performance tests on HTTP caching proxy Squid (using NFV-VITAL@NFV-SDN’15)
Challenges

- How to do VNF auto resource flexing efficiently and effectively?
- VNF scaling points depends on
  - Workload dynamics
  - Underlying infrastructure
  - Current resource allocations
  - VNF functionalities and implementations
- Costs associated with VNF scaling timing
  - Too soon → Increased costs
  - Too late → Increased SLA violation penalties
- Service function chaining
  - Interdependence across VNFs in forwarding graph
ENVI – Our Solution
ENVI Architecture

ETSI NFV Architecture

OSS/BSS

Service, VNF and Infrastructure Description

EMS
EMS
EMS

VNF

Orchestrator

VNF Manager(s)

Virtualized Infrastructure Manager(s)

NFVI

Virtual Computing
Virtual Storage
Virtual Network

Virtualization Layer

Computing Hardware
Storage Hardware
Network Hardware

Hardware resources
Collect VNF-level and infrastructure-level feature info (VNF dependent).

- Pull feature info from VNF monitor every interval $T$,
- Determine if scaling action is required every interval $W$,
- Push the scale vector with collected info to RFE.

- Receive scale vector from SDE,
- Evaluate overload situation of the entire SFC,
- Make resource flexing plan and push them to PE.

- Receive resource flexing plan from RFE,
- Convert plan to executable actions (platform dependent),
- Push actions to orchestrator for execution.
Key Contributions of SDE

• Infrastructure-level features
Key Contributions of SDE

- Infrastructure-level features
Key Contributions of SDE

• Infrastructure-level features + VNF-level features
  • Better understanding of VNF status
Key Contributions of SDE

• Infrastructure-level features + VNF-level features
  • Better understanding of VNF status
• Classification problem => “do not scale” or “scale”
  • Infeasible to formulate exact mathematical models
  • Leverage machine learning algorithms
Key Contributions of SDE

• Infrastructure-level features + VNF-level features
  • Better understanding of VNF status

• Classification problem => “do not scale” or “scale”
  • Infeasible to formulate exact mathematical models
  • Leverage machine learning algorithms

• Neural network model
  • Select input features and construct new features through hidden layers
  • Fit complex nonlinear functions
  • Model dependence of input features and data points
  • Four layers: Input layer, two hidden layers and output layer
Workflow of SDE

1. **Online**
   - Performance Tests

2. **Offline**
   - Training Data (Composite Features)
   - Train Neural Network Model
   - Scale Vector
   - Resource Flexing Engine
   - Decision Evaluation
Workflow of SDE

- Offline
  - Performance tests to cover different types of workload
  - Collect composite feature information as training data
  - Label data points with “do not scale” and “scale”
  - Train an initial model for each VNF
Workflow of SDE

- Offline
  - Performance tests to cover different types of workload
  - Collect composite feature information as training data
  - Label data points with “do not scale” and “scale”
  - Train an initial model for each VNF

- Online
  - Keep collecting information of all features
  - Generate scale vector based on current models
  - Evaluate and keep training models with latest data points (background)
  - Update current models periodically
Workflow of SDE

• Offline
  • Performance tests to cover different types of workload
  • Collect composite feature information as training data
  • Label data points with “do not scale” and “scale”
  • Train an initial model for each VNF

• Online
  • Keep collecting information of all features
  • Generate scale vector based on current models
  • Evaluate and keep training models with latest data points (background)
  • Update current models periodically

• Extending features
  • Domain knowledge, time series information, statistical information
ENVI Components

• VNF monitor
  • Develop VNF monitoring agent for each VNF
  • Convert raw info to key-value data

• Scaling Decision Engine

• Resource flexing engine
  • Break multi-VNF scaling down to single-VNF scaling
  • Redistribute flows
  • Scale resource allocation

• Placement engine
  • Use OpenStack nova-scheduler service by default
  • Compatible with other VNF placement algorithms, e.g., VNF-P@CNSM’14
Prototype Evaluation

(For Scaling Decision Engine)
Testbed

- 3 * HP DL360p blade servers: 2 * Intel Xeon E5-2680 v2, 212 GB RAM
- 2 * HP Z420 workstations: 1 * Intel Xeon E5-1620, 16 GB RAM
- 1 * HPE 5820X 10 GB Switch
- Running OpenStack Kilo
Testbed

- 3 * HP DL360p blade servers: 2 * Intel Xeon E5-2680 v2, 212 GB RAM
- 2 * HP Z420 workstations: 1 * Intel Xeon E5-1620, 16 GB RAM
- 1 * HPE 5820X 10 GB Switch
- Running OpenStack Kilo

<table>
<thead>
<tr>
<th>VNF</th>
<th>Suricata</th>
<th>Squid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Intrusion detection system</td>
<td>HTTP caching proxy</td>
</tr>
<tr>
<td>Version</td>
<td>3.2.1</td>
<td>3.5.20</td>
</tr>
<tr>
<td>Workload generator</td>
<td>hping3 &amp; iperf</td>
<td>Web Polygraph</td>
</tr>
<tr>
<td>Workload types</td>
<td>Malicious ratio 0% ~ 90%</td>
<td>Response size 10KB ~ 100KB</td>
</tr>
<tr>
<td>Experiment methodology</td>
<td>Packet rate changes randomly</td>
<td>HTTP request rate changes randomly</td>
</tr>
<tr>
<td></td>
<td>around capacity point per minute</td>
<td>around capacity point per minute</td>
</tr>
</tbody>
</table>
### Example Features

#### Suricata

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>app_layer.flow.dcerpc_tcp</td>
<td>decoder.ipraw.invalid_ip_version, dns.memcap_state</td>
</tr>
<tr>
<td>app_layer.flow.dcerpc_udp</td>
<td>decoder.ipv4, dns.memuse</td>
</tr>
<tr>
<td>app_layer.flow.dns_tcp</td>
<td>decoder.ipv4_in_ipv6, flow.emerg_mode_entered</td>
</tr>
<tr>
<td>app_layer.flow.dns_udp</td>
<td>decoder.ipv6, flow.emerg_mode_over</td>
</tr>
<tr>
<td>app_layer.flow.failed_tcp</td>
<td>decoder.ipv6_in_ipv6, flow.memcap</td>
</tr>
<tr>
<td>app_layer.flow.failed_udp</td>
<td>decoder.ltnull.pkt_too_small, flow.memuse</td>
</tr>
<tr>
<td>app_layer.flow.ftp</td>
<td>decoder.ltnull.unsupported_type, flow.spare</td>
</tr>
<tr>
<td>app_layer.flow.http</td>
<td>decoder.max_pkt_size, flow.tcp_reuse</td>
</tr>
<tr>
<td>app_layer.flow.imap</td>
<td>decoder.mpls, flow.mgr.bypassed_pruned</td>
</tr>
<tr>
<td>app_layer.flow.msn</td>
<td>decoder.null, flow.mgr.closed_pruned</td>
</tr>
<tr>
<td>app_layer.flow.smtp</td>
<td>decoder.ppts, flow.mgr.est_pruned</td>
</tr>
<tr>
<td>app_layer.flow.ssh</td>
<td>decoder.pppoe, flow.mgr.flows_timeoutout</td>
</tr>
<tr>
<td>app_layer.flow.tls</td>
<td>decoder.raw, flow.mgr.flows_removed</td>
</tr>
<tr>
<td>app_layer.flow.tx.dns_tcp</td>
<td>decoder.sctp, flow.mgr.flows_timeout</td>
</tr>
<tr>
<td>app_layer.flow.tx.http</td>
<td>decoder.tcp, flow.mgr.new_pruned</td>
</tr>
<tr>
<td>app_layer.flow.tx.smtp</td>
<td>decoder.teredo, flow.mgr.rows_busy</td>
</tr>
<tr>
<td>app_layer.flow.tx.tls</td>
<td>decoder.udp, flow.mgr.rows_checked</td>
</tr>
</tbody>
</table>

#### Squid

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU_Time</td>
<td>Number_of_clients_accessing_cache, icp.kbytes_sent</td>
</tr>
<tr>
<td>CPU_Usage</td>
<td>Number_of_file_desc, icp.pkts_recv</td>
</tr>
<tr>
<td>Cache_Hits</td>
<td>Number_of_queued_ICP_replies, icp.pkts_sent</td>
</tr>
<tr>
<td>Cache_Misses</td>
<td>Reserved_number_of_file_desc, icp.q_kbytes_recv</td>
</tr>
<tr>
<td>Cache_information_for_squid</td>
<td>Resource_usage_for_squid, icp.q_kbytes_sent</td>
</tr>
<tr>
<td>Connection</td>
<td>Select_loop_called, icp.queries_recv</td>
</tr>
<tr>
<td>Connection_information_for_squid</td>
<td>Storage_Mem_capacity, icp.queries_sent</td>
</tr>
<tr>
<td>Content-Type</td>
<td>Storage_Mem_size, icp.query_median_svc_time</td>
</tr>
<tr>
<td>Current_Time</td>
<td>Storage_Swap_capacity, icp.query_timeouts</td>
</tr>
<tr>
<td>DNS_Lookups</td>
<td>Storage_Swap_size, icp.r_kbytes_recv</td>
</tr>
<tr>
<td>File_descriptor_usage_for_squid</td>
<td>Store_Disk_files_open, icp.r_kbytes_sent</td>
</tr>
<tr>
<td>Filesqueued_for_open</td>
<td>Total_accounted, icp.replies_queued</td>
</tr>
<tr>
<td>Free_Ordinary_blocks</td>
<td>Total_free, icp.replies_revc</td>
</tr>
<tr>
<td>Free_Small_blocks</td>
<td>Total_in_use, icp.replies_sent</td>
</tr>
<tr>
<td>HTTP_Requests_(All)</td>
<td>Total_size, icp.reply_median_svc_time</td>
</tr>
<tr>
<td>Hits_as_%_of_all_requests</td>
<td>Total_space_in_arena, median_select_fds</td>
</tr>
<tr>
<td>Hits_as_%_of_bytes_sent</td>
<td>UP_Time, memPoolAlloc_calls</td>
</tr>
<tr>
<td>Holding_blocks</td>
<td>aborted_requests, memPoolFree_calls</td>
</tr>
<tr>
<td>ICP_Queries</td>
<td>average_select_fd_period, memPool_accounted</td>
</tr>
<tr>
<td>Internal_Data_Structures</td>
<td>client_http.all_median_svc_time, memPool_unaccounted</td>
</tr>
<tr>
<td>Largest_file_desc Currently_In_Use</td>
<td>client_http.errors, page_faults</td>
</tr>
<tr>
<td>Last_Modified</td>
<td>client_http.hit_median_svc_time, sample_end_time</td>
</tr>
<tr>
<td>Maximum_Resident_Size</td>
<td>client_http.hits, sample_start_time</td>
</tr>
<tr>
<td>Maximum_number_of_file_descriptors</td>
<td>client_http.kbytes_in, select_fds</td>
</tr>
<tr>
<td>Mean_Object_Size</td>
<td>client_http.kbytes_out, select_loops</td>
</tr>
<tr>
<td>Number_of_HTTPC_Messages_received</td>
<td>client_http.nh_median_svc_time, server.all.errors</td>
</tr>
<tr>
<td>Number_of_HTTPC_Messages_sent</td>
<td>client_http.nh_median_svc_time, server.all.kbytes_in</td>
</tr>
<tr>
<td>server.all.kbytes_out</td>
<td>server.ftp.kbytes_in, sysscalls.disk.closes</td>
</tr>
<tr>
<td>server.all.requests</td>
<td>server.ftp.kbytes_out, sysscalls.disk.opens</td>
</tr>
<tr>
<td>server.ftp.errors</td>
<td>server.ftp.requests, sysscalls.disk_reads</td>
</tr>
</tbody>
</table>
Evaluation Methodology

• Training and testing
  • Train neural network model with $n$ workload types, $n = 1, 2, \ldots, 9$
  • Run 5-fold cross-validation to verify trained model
  • Test the trained model on $10 - n$ workload types

• Metrics
  • Accuracy: $\frac{\text{correct predictions}}{\text{total predictions}}$ for overall correctness
  • Precision: $\frac{\text{true positives}}{\text{true positives} + \text{false positives}}$ for exactness
  • Recall: $\frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$ for completeness
  • ROC and AUROC: true positive rate vs false positive rate

• Compared with decision tree (DT), random forest (RF), logistic regression (LR) and rule-based approach as baseline (BL)
Suricata Results
Suricata Results

- NN ≈ LR > RF > DT
- 0.85 for all metrics

- NN > LR > RF > DT
- Slightly worse than infrastructure-level for DT, RF, LR
- Suricata is a relatively simple VNF
- Tight correlation with infrastructure resource usage
Suricata Results

Infrastructure-level features

- NN ≈ LR > RF > DT
- 0.85 for all metrics
- Slightly worse than infrastructure-level for DT, RF, LR

VNF-level features

- NN > LR > RF > DT
- Suricata is a relatively simple VNF
- Tight correlation with infrastructure resource usage

Composite features

- NN > LR > RF > DT
Suricata Results

- NN ≈ LR > RF > DT
- 0.85 for all metrics
- NN > LR > RF > DT
- Slightly worse than infrastructure-level for DT, RF, LR
- Suricata is a relatively simple VNF
- Tight correlation with infrastructure resource usage
- NN > LR > RF > DT
- Composite features > infrastructure-level features > VNF-level features
- Neural network outperforms other algorithms
Squid Results
Squid Results

- Low for all models and metrics
- Squid is a relatively complex VNF
- Infrastructure resource usage is not adequate

- NN > LR > RF > DT
- Much better than infrastructure-level for all models
- NN gets > 0.85 for all metrics

- NN > LR > RF > DT
Squid Results

- Low for all models and metrics
- Squid is a relatively complex VNF
- Infrastructure resource usage is not adequate

NN > LR > RF > DT
- Much better than infrastructure-level

Composite features
- NN > LR > RF > DT

- Composite features ~ VNF-level features > infrastructure-level features
- Neural network outperforms other algorithms
Conclusion

• Designed a modular framework for NFV resource flexing
• Combined infrastructure-level features and VNF-level features to understand VNF performance behavior
• Adopted neural network model to make VNF scaling decisions
• Evaluated scaling decision engine with two open source VNFs
Discussion

• Model Feature Set
  • Rely on vendors to expose relevant features

• Offline Model Training Overhead
  • Train a model for each VNF

• Online Model Evolution
  • Scoring function to evaluate false positive and false negative

• Finer-grained Resource Flexing
  • Customized dynamic resource sizing
Thank you!

Questions?