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

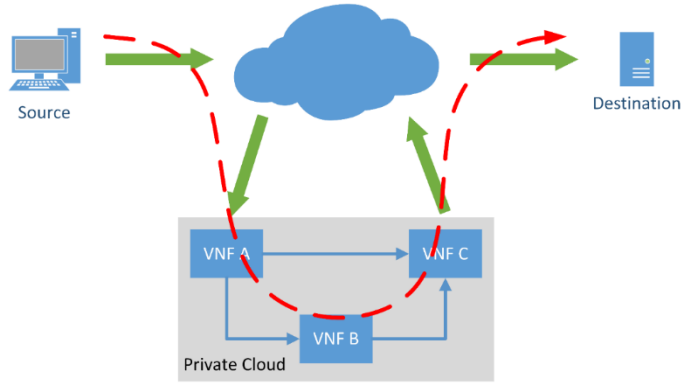
Traffic Manager

Virtualization

Virtualization Platform

Cloudification

Service Function Chaining

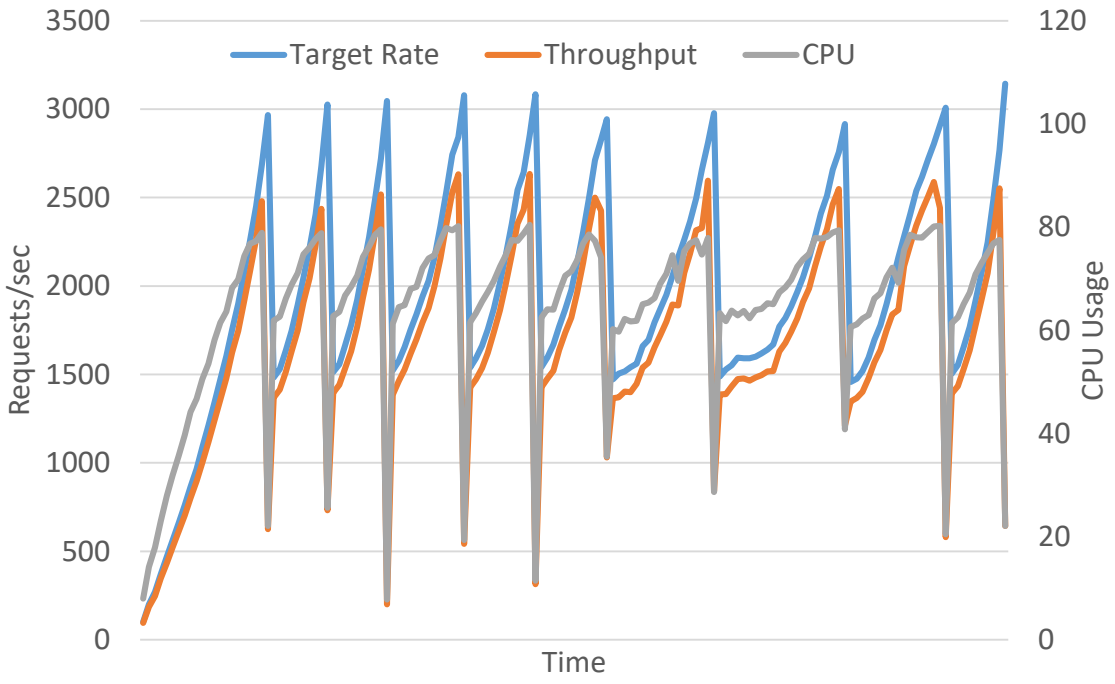


Flexibility | Scalability | Elasticity | Agility

Auto Resource Flexing

CAPEX & OPEX Reduction

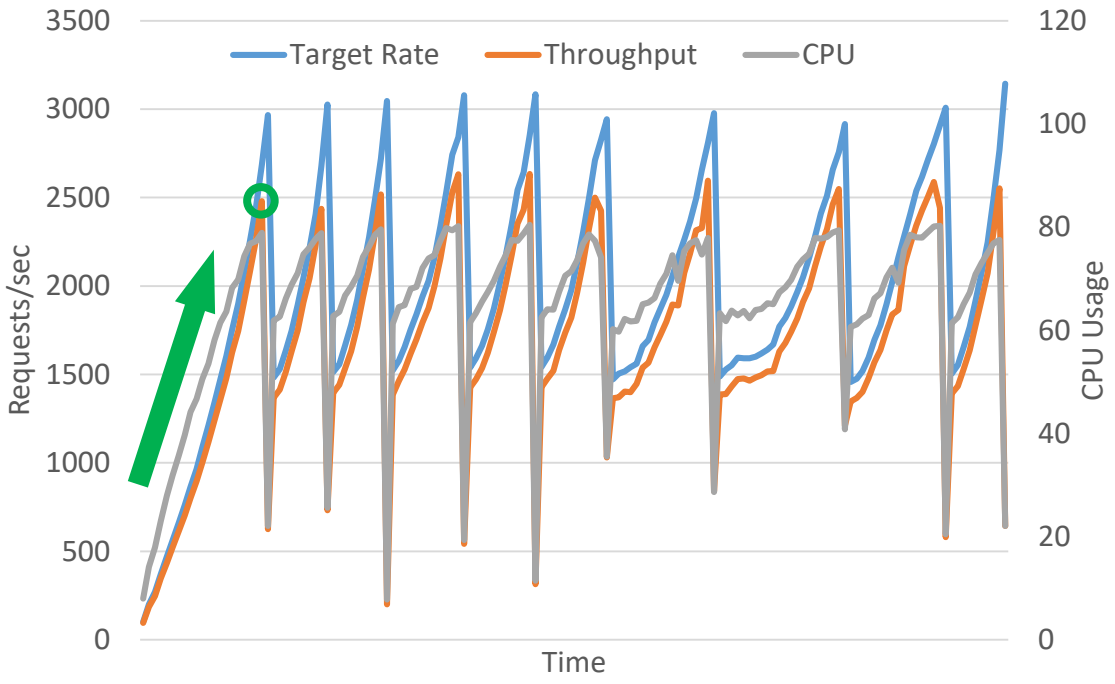
VNF Resource Flexing Example



Instance 1	
Flavor	small
vCPU	1
RAM	2 GB
Disk	10 GB

HTTP caching proxy - Squid

VNF Resource Flexing Example



Instance 1	
Flavor	small
vCPU	1
RAM	2 GB
Disk	10 GB

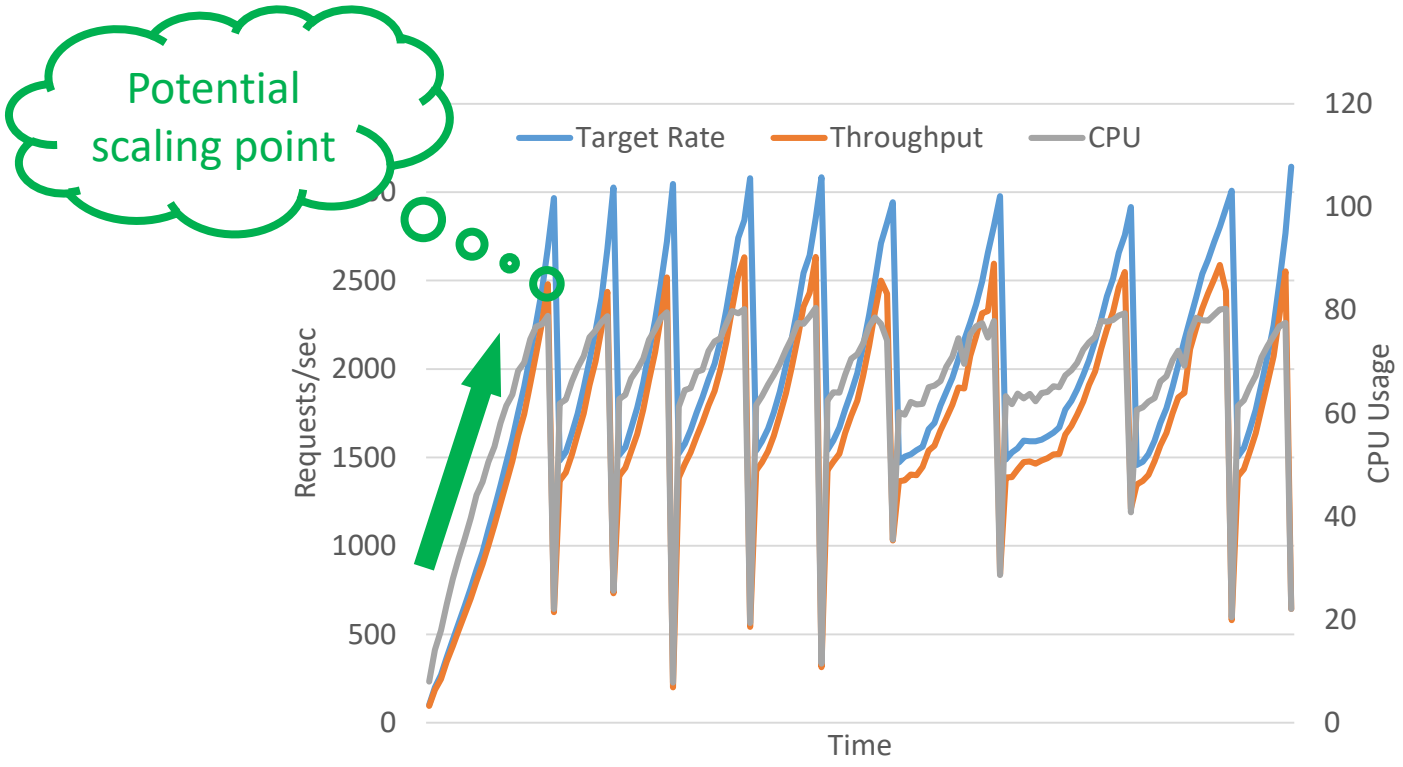
HTTP caching proxy - Squid

VNF Resource Flexing Example

Instance 1	
Flavor	small
vCPU	1
RAM	2 GB
Disk	10 GB

HTTP caching proxy - Squid

VNF Resource Flexing Example



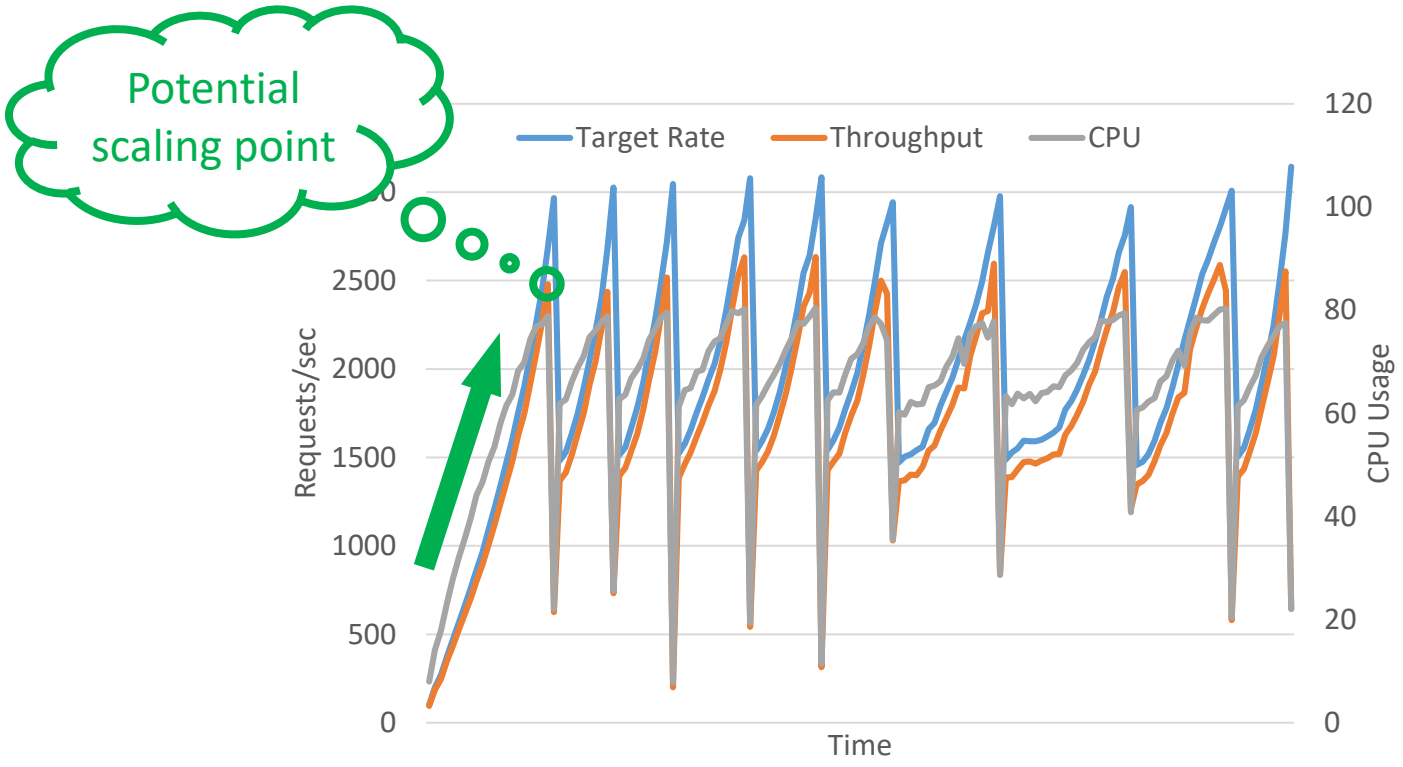
Instance 1	
Flavor	small
vCPU	1
RAM	2 GB
Disk	10 GB

Scaling up

Instance 1	
Flavor	medium
vCPU	2
RAM	4 GB
Disk	20 GB

HTTP caching proxy - Squid

VNF Resource Flexing Example



Instance 1	
Flavor	small
vCPU	1
RAM	2 GB
Disk	10 GB

Scaling up

Instance 1	
Flavor	medium
vCPU	2
RAM	4 GB
Disk	20 GB

Scaling out

Instance 1	
Flavor	small
vCPU	1
RAM	2 GB
Disk	10 GB

Instance 2	
Flavor	small
vCPU	1
RAM	2 GB
Disk	10 GB

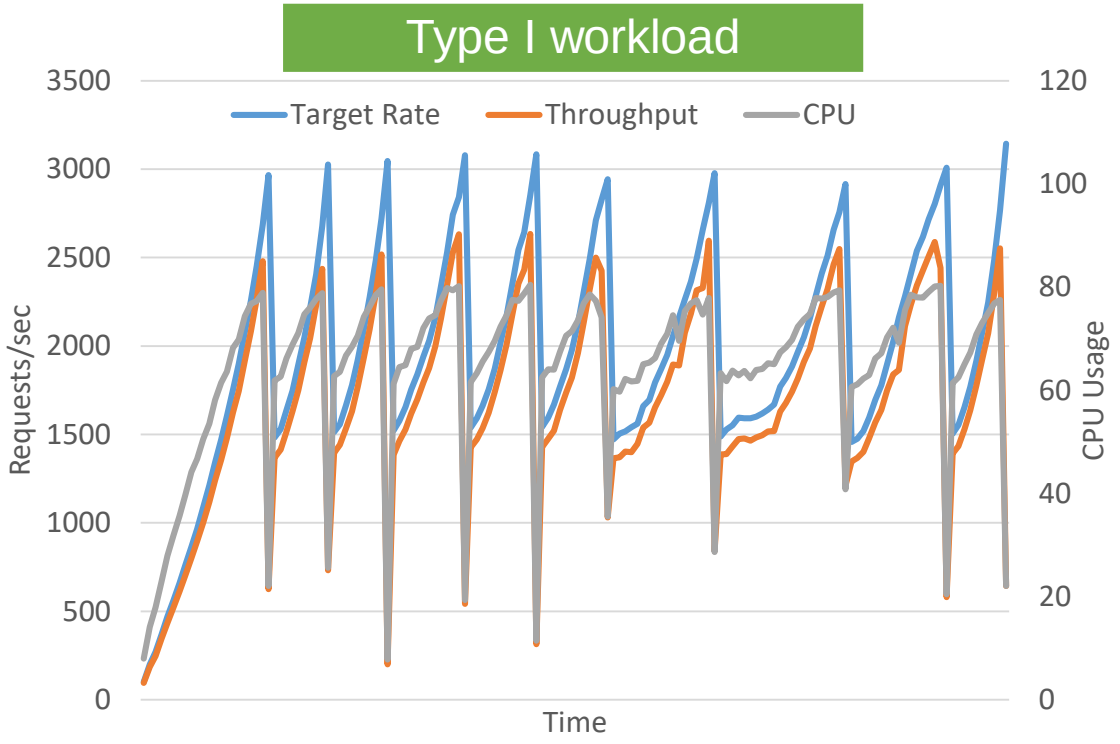
HTTP caching proxy - Squid

Related Work

Google Cloud Platform

- 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

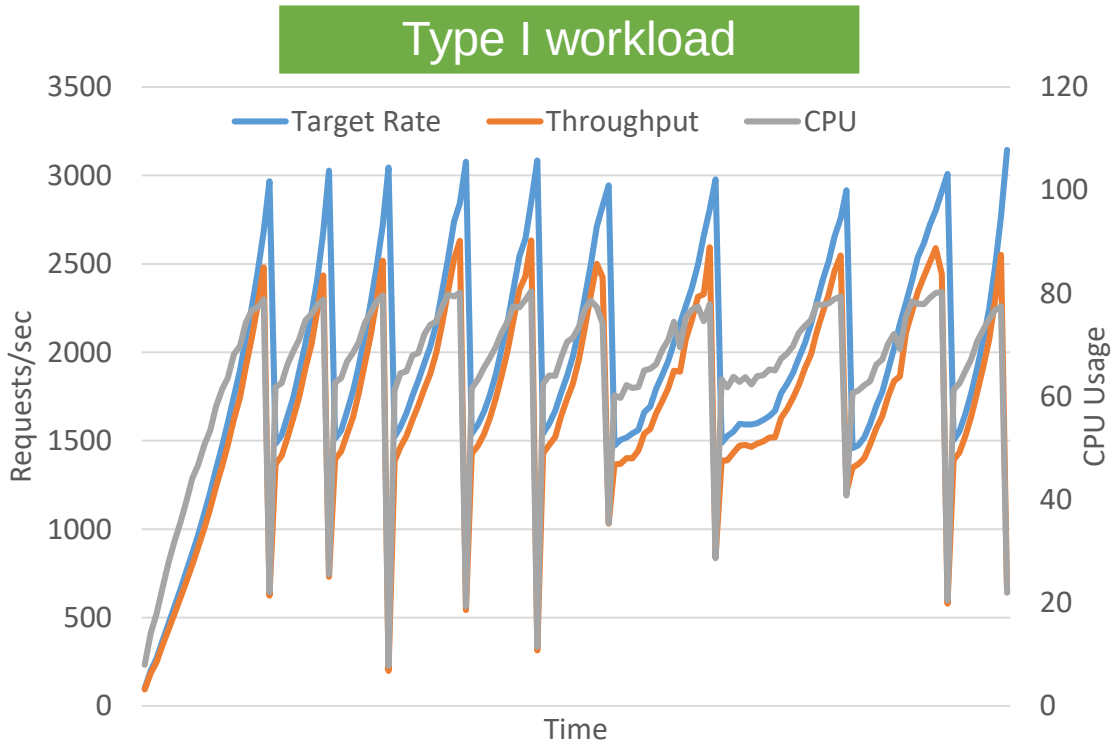
VNF Scaling Detection

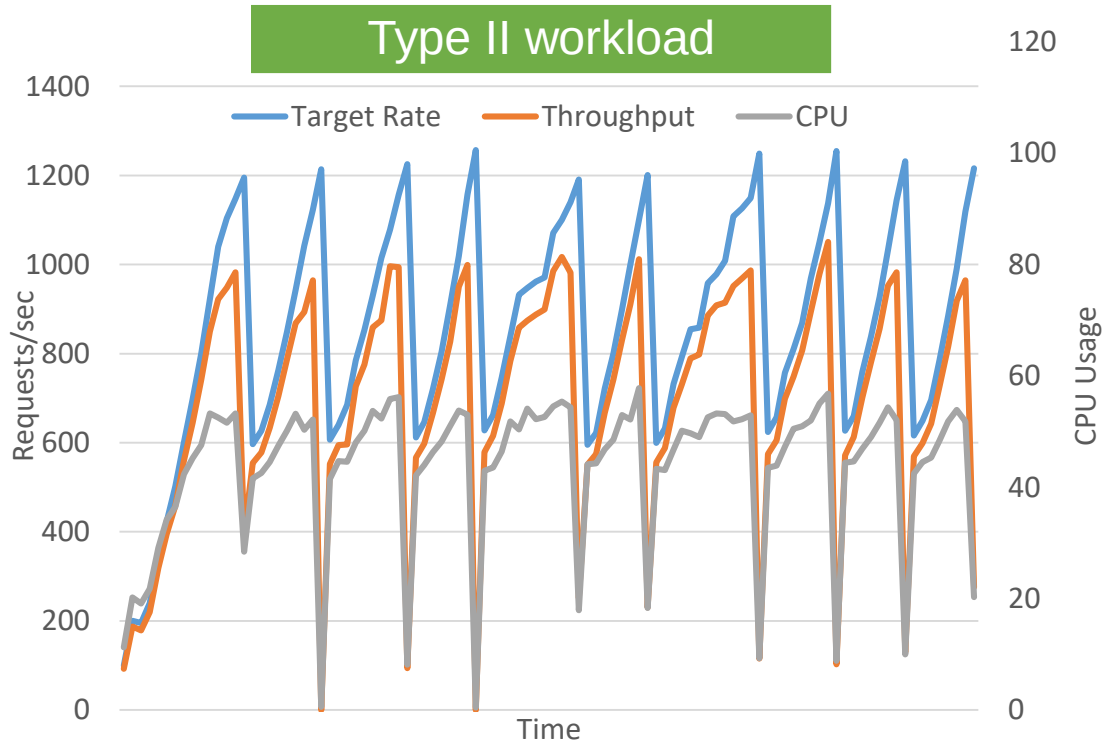


Instance 1	
Flavor	small
vCPU	1
RAM	2 GB
Disk	10 GB

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

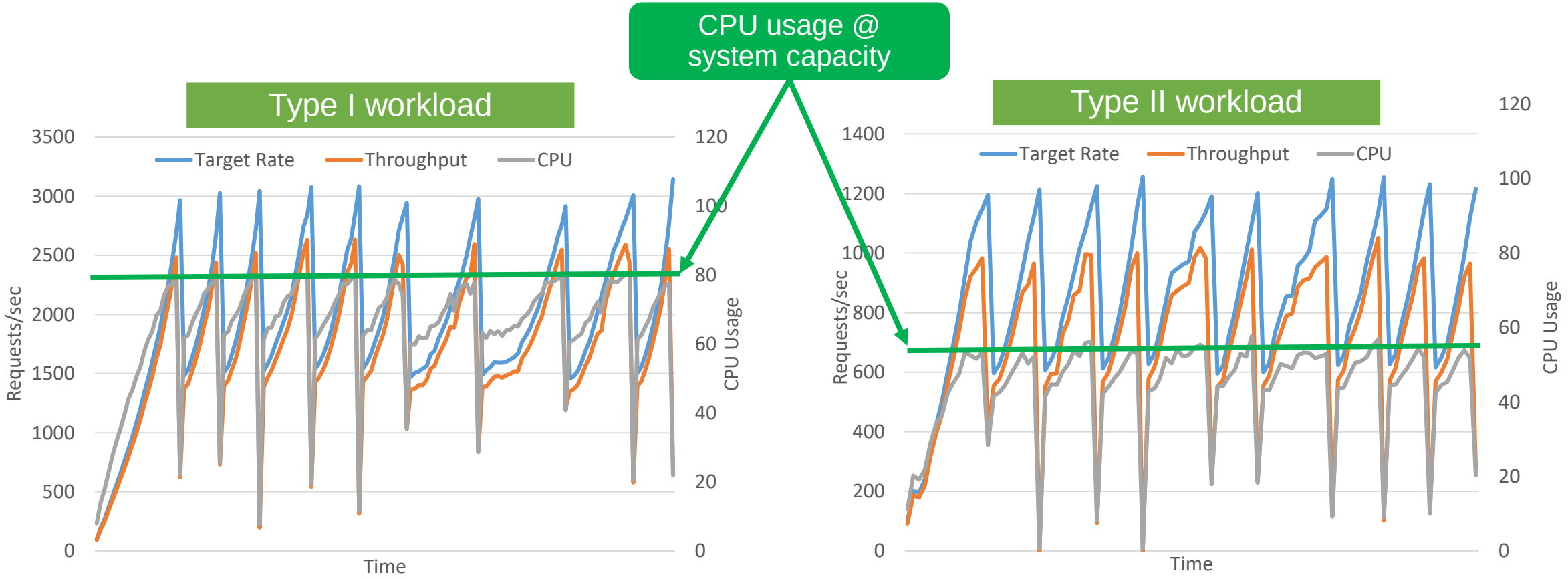
VNF Scaling Detection





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

VNF Scaling Detection



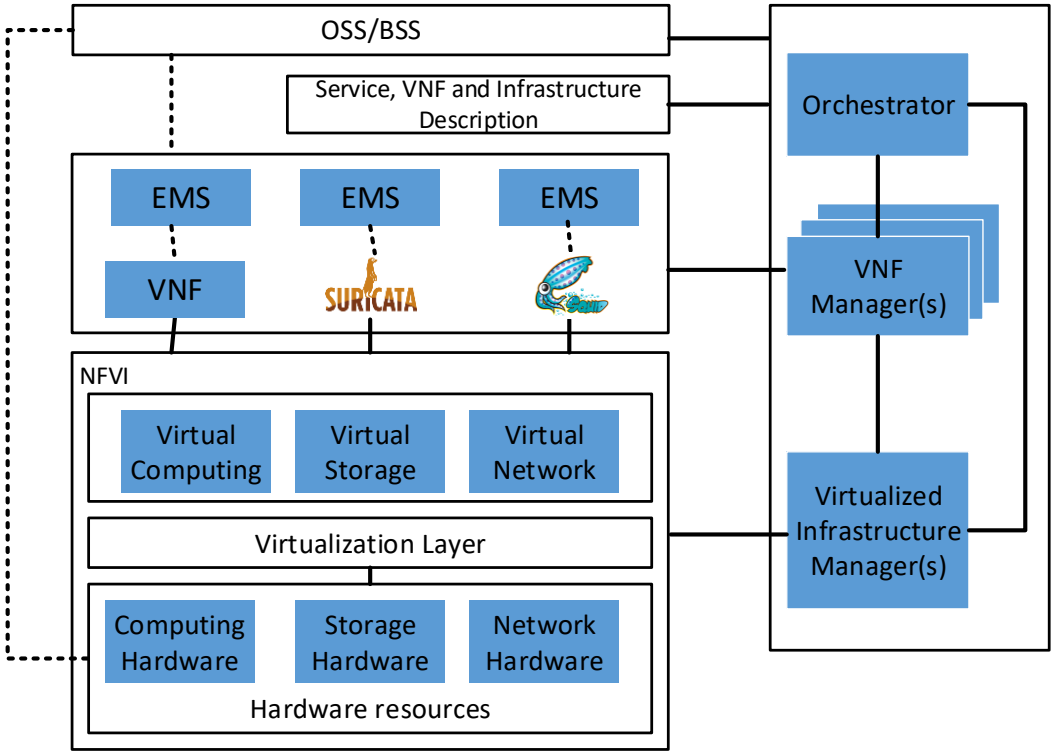
Performance tests on HTTP caching proxy Squid (using NFV-VITAL@NFV-SDN'15)

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

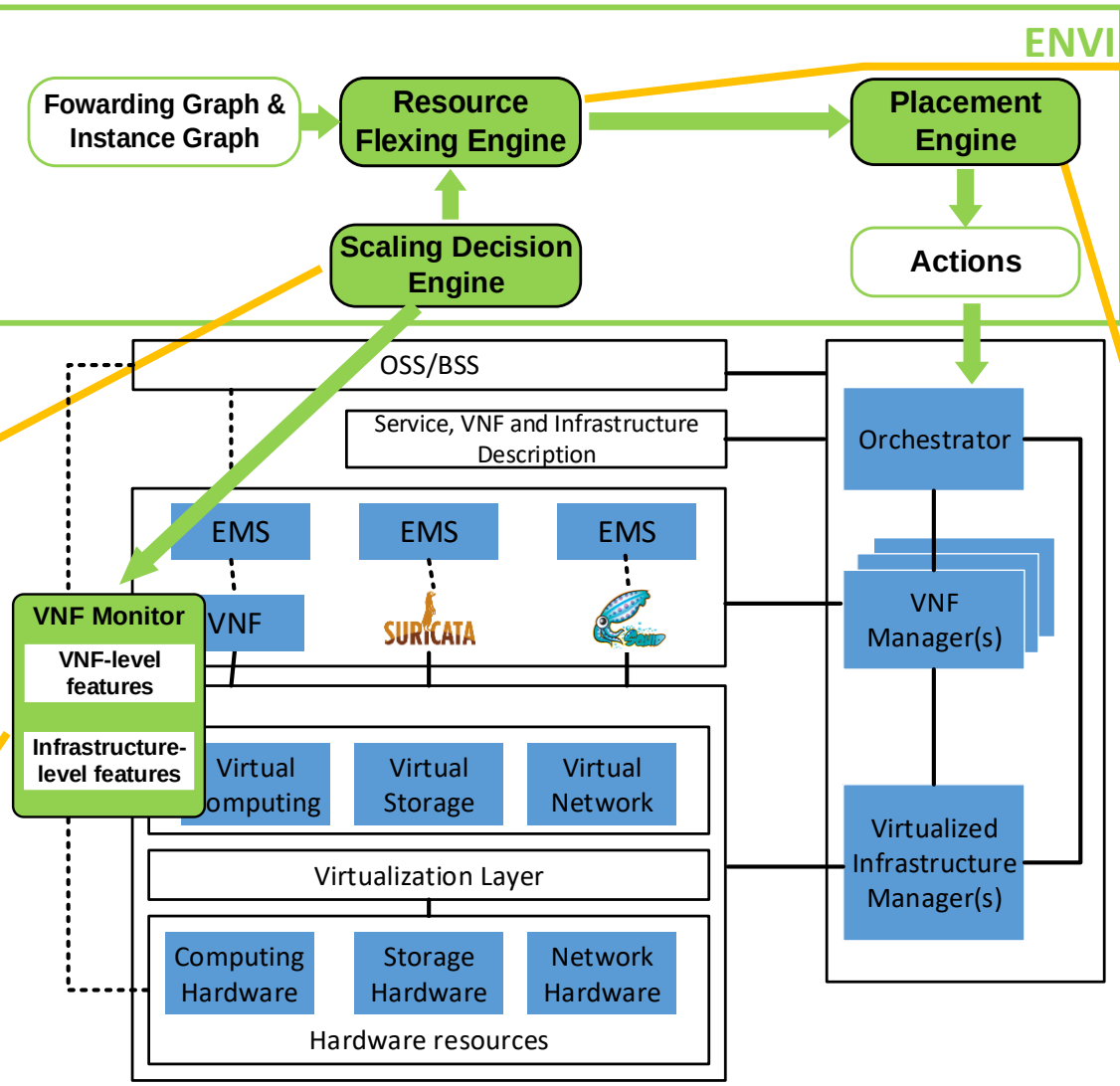


ETSI NFV Architecture

ENVI Architecture

- 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.

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



ETSI NFV Architecture

- 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

- Infrastru

```
ubuntu@squid-1:~$ squidclient mgr:info
HTTP/1.1 200 OK
Server: squid/3.3.8
Mime-Version: 1.0
Date: Tue, 30 Aug 2016 17:41:03 GMT
Content-Type: text/plain
Expires: Tue, 30 Aug 2016 17:41:03 GMT
Last-Modified: Tue, 30 Aug 2016 17:41:03 GMT
X-Cache: MISS from squid-1
X-Cache-Lookup: MISS from squid-1:3128
Via: 1.1 squid-1 (squid/3.3.8)
Connection: close

Squid Object Cache: Version 3.3.8
Start Time:     Tue, 30 Aug 2016 12:16:15 GMT
Current Time:   Tue, 30 Aug 2016 17:41:03 GMT
Connection information for squid:
        Number of clients accessing cache:      2
        Number of HTTP requests received:       4831603
        Number of ICP messages received:        0
        Number of ICP messages sent:    0
        Number of queued ICP replies:   0
        Number of HTCP messages received:       0
        Number of HTCP messages sent:   0
        Request failure ratio:   0.00
        Average HTTP requests per minute since start:   14875.2
        Average ICP messages per minute since start:    0.0
        Select loop called: 103885425 times, 0.188 ms avg
Cache information for squid:
        Hits as % of all requests:      5min: 0.0%, 60min: 0.0%
        Hits as % of bytes sent:        5min: -0.0%, 60min: -0.0%
        Memory hits as % of hit requests:       5min: 0.0%, 60min: 0.0%
        Disk hits as % of hit requests: 5min: 0.0%, 60min: 0.0%
        Storage Swap size:      0 KB
        Storage Swap capacity:   0.0% used, 100.0% free
        Storage Mem size:       216 KB
        Storage Mem capacity:    0.1% used, 99.9% free
        Mean Object Size:       0.00 KB
        Requests given to unlinkd:      0
Median Service Times (seconds)  5 min    60 min:
        HTTP Requests (All):   0.00000  0.00000
        Cache Misses:          0.00000  0.00000
        Cache Hits:            0.00000  0.00000
        Near Hits:             0.00000  0.00000
        Not-Modified Replies:  0.00000  0.00000
        DNS Lookups:           0.00000  0.00000
        ICP Queries:           0.00000  0.00000
```

```
ubuntu@squid-1:~$ squidclient mgr:5min
HTTP/1.1 200 OK
Server: squid/3.3.8
Mime-Version: 1.0
Date: Tue, 30 Aug 2016 17:43:05 GMT
Content-Type: text/plain
Expires: Tue, 30 Aug 2016 17:43:05 GMT
Last-Modified: Tue, 30 Aug 2016 17:43:05 GMT
X-Cache: MISS from squid-1
X-Cache-Lookup: MISS from squid-1:3128
Via: 1.1 squid-1 (squid/3.3.8)
Connection: close

sample_start_time = 1472578635.380593 (Tue, 30 Aug 2016 17:37:15 GMT)
sample_end_time = 1472578935.384405 (Tue, 30 Aug 2016 17:42:15 GMT)
client_http.requests = 0.003333/sec
client_http.hits = 0.000000/sec
client_http.errors = 0.000000/sec
client_http.kbytes_in = 0.000000/sec
client_http.kbytes_out = 0.010000/sec
client_http.all_median_svc_time = 0.000000 seconds
client_http.miss_median_svc_time = 0.000000 seconds
client_http.nm_median_svc_time = 0.000000 seconds
client_http.nh_median_svc_time = 0.000000 seconds
client_http.hit_median_svc_time = 0.000000 seconds
server.all.requests = 0.000000/sec
server.all.errors = 0.000000/sec
server.all.kbytes_in = 0.000000/sec
server.all.kbytes_out = 0.000000/sec
server.http.requests = 0.000000/sec
server.http.errors = 0.000000/sec
server.http.kbytes_in = 0.000000/sec
server.http.kbytes_out = 0.000000/sec
server.ftp.requests = 0.000000/sec
server.ftp.errors = 0.000000/sec
server.ftp.kbytes_in = 0.000000/sec
server.ftp.kbytes_out = 0.000000/sec
server.other.requests = 0.000000/sec
server.other.errors = 0.000000/sec
server.other.kbytes_in = 0.000000/sec
server.other.kbytes_out = 0.000000/sec
icp.pkts_sent = 0.000000/sec
icp.pkts_recv = 0.000000/sec
icp.queries_sent = 0.000000/sec
icp.replies_sent = 0.000000/sec
icp.queries_recv = 0.000000/sec
icp.replies_recv = 0.000000/sec
```

Key Contributions of SDE

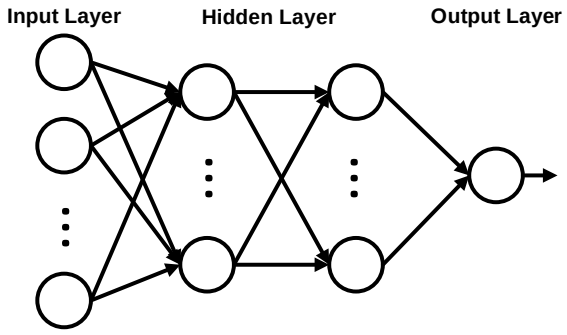
- Infrastructure-level features + VNF-level features
  - Better understanding of VNF status

Key Contributions of SDE

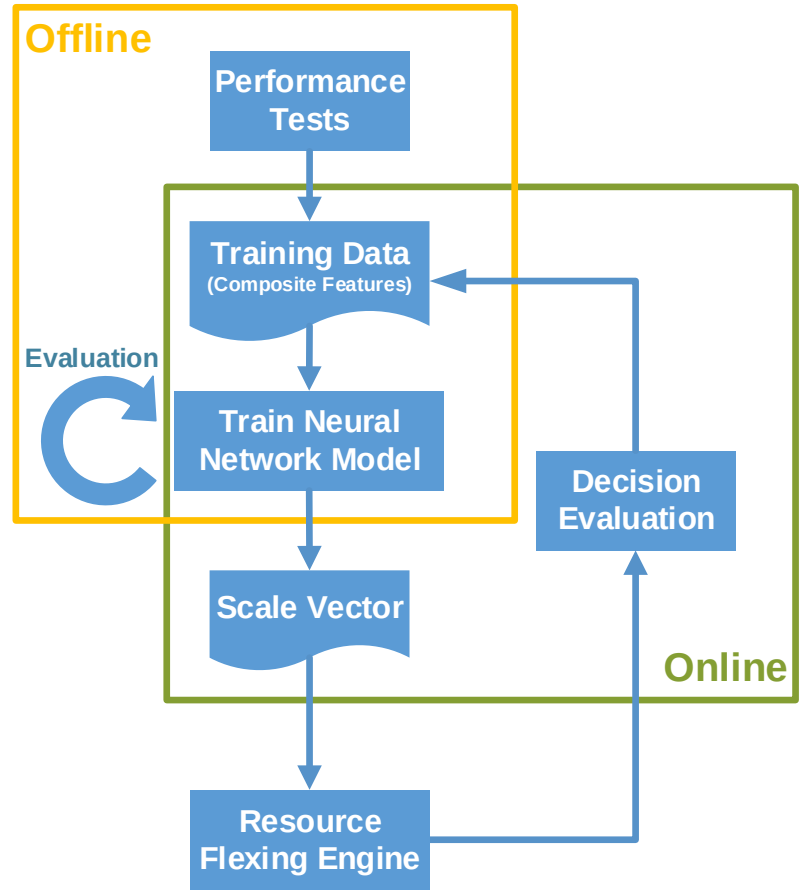
- Infrastructure-level features + VNF-level features
  - Better understanding of VNF status
- Classification problem => “do not scale” or “scale”
  - Infeasible to formulate exact mathematical models
  - Leverage machine learning algorithms

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

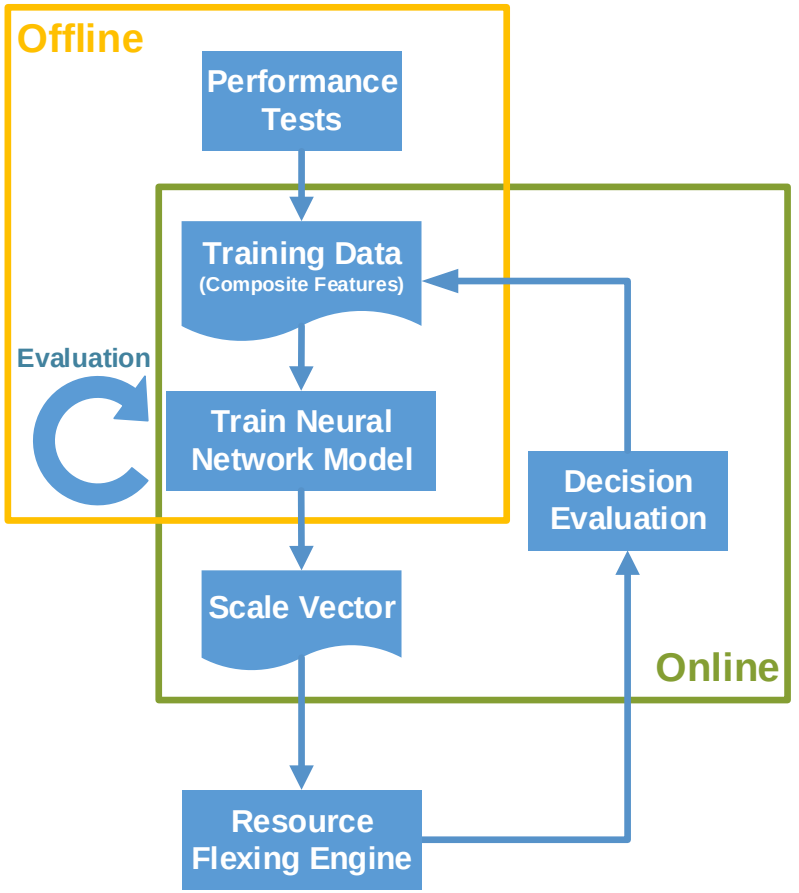


Workflow of SDE



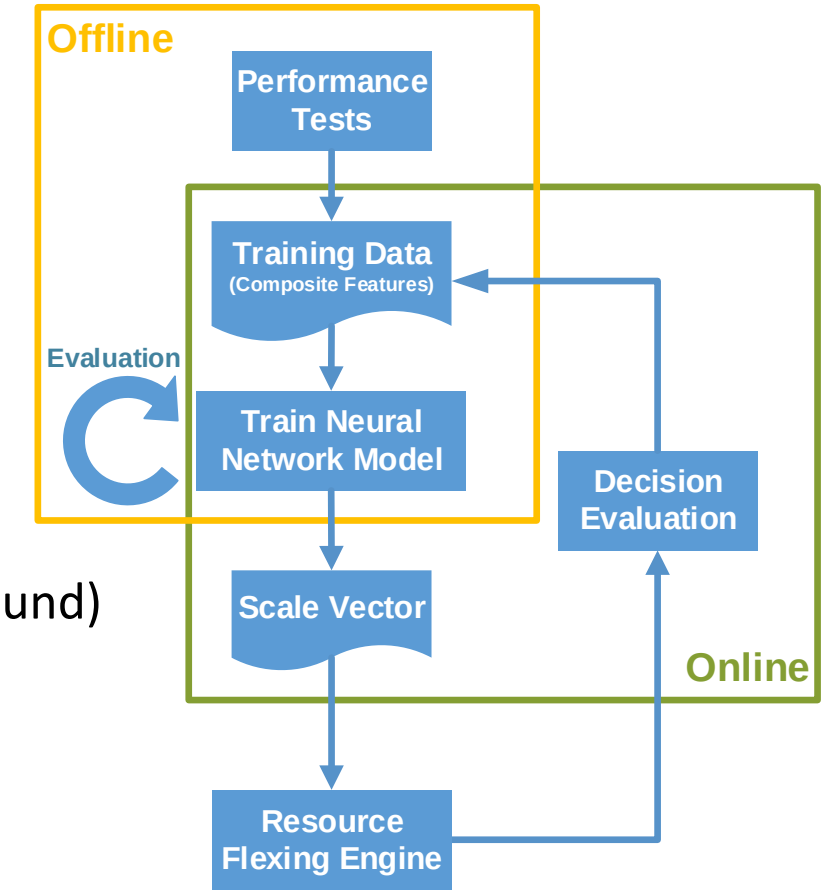
Workflow of SDE

- 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



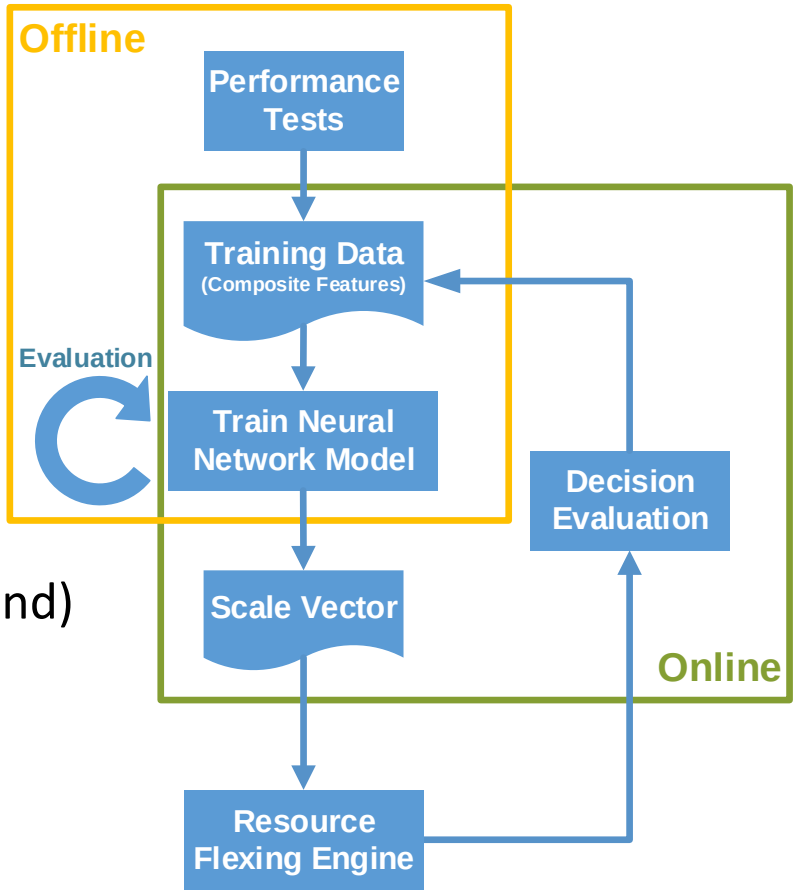
Workflow of SDE

- 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

VNF	Suricata	Squid
Functionality	Intrusion detection system	HTTP caching proxy
Version	3.2.1	3.5.20
Workload generator	hping3 & iperf	Web Polygraph
Workload types	Malicious ratio 0% ~ 90%	Response size 10KB ~ 100KB
Experiment methodology	Packet rate changes randomly around capacity point per minute	HTTP request rate changes randomly around capacity point per minute

Example Features

Suricata

app_layer.flow.dcerpc_tcp	decoder.ipraw.invalid_ip_version	dns.memcap_state
app_layer.flow.dcerpc_udp	decoder.ipv4	dns.memuse
app_layer.flow.dns_tcp	decoder.ipv4_in_ipv6	flow.emerg_mode_entered
app_layer.flow.dns_udp	decoder.ipv6	flow.emerg_mode_over
app_layer.flow.failed_tcp	decoder.ipv6_in_ipv6	flow.memcap
app_layer.flow.failed_udp	decoder.ltnull.pkt_too_small	flow.memuse
app_layer.flow.ftp	decoder.ltnull.unsupported_type	flow.spare
app_layer.flow.http	decoder.max_pkt_size	flow.tcp_reuse
app_layer.flow.imap	decoder.mpls	flow_mgr.bypassed_pruned
app_layer.flow.msn	decoder.null	flow_mgr.closed_pruned
app_layer.flow.smb	decoder.pkts	flow_mgr.est_pruned
app_layer.flow.smtp	decoder.ppp	flow_mgr.flows_checked
app_layer.flow.ssh	decoder.pppoe	flow_mgr.flows_notimeout
app_layer.flow.tls	decoder.raw	flow_mgr.flows_removed
app_layer.tx.dns_tcp	decoder.sctp	flow_mgr.flows_timeout
app_layer.tx.http	decoder.tcp	flow_mgr.new_pruned
app_layer.tx.smtp	decoder.teredo	flow_mgr.rows_busy
app_layer.tx.tls	decoder.udp	flow_mgr.rows_checked
capture.kernel_drops	decoder.vlan	flow_mgr.rows_empty
capture.kernel_packets	decoder.vlan_qinq	flow_mgr.rows_maxlen
decoder.avg_pkt_size	defrag.ipv4.fragments	flow_mgr.rows_skipped
decoder.bytes	defrag.ipv4.reassembled	http.memcap
decoder.dce.pkt_too_small	defrag.ipv4.timeouts	http.memuse
decoder.erspan	defrag.ipv6.fragments	tcp.invalid_checksum
decoder.ethernet	defrag.ipv6.reassembled	tcp.memuse
decoder.gre	defrag.ipv6.timeouts	tcp.no_flow
decoder.icmpv4	defrag.max_frag_hits	tcp.pseudo
decoder.icmpv6	detect.alert	tcp.pseudo_failed
decoder.invalid	dns.memcap_global	tcp.reassembly_gap

Squid

CPU_Time	Number_of_clients_accessing_cache	icp.kbytes_sent
CPU_Usage	Number_of_file_desc	icp.pkts_recv
Cache_Hits	Number_of_queued_ICP_replies	icp.pkts_sent
Cache_Misses	Reserved_number_of_file_desc	icp.q_kbytes_recv
Cache_information_for_squid	Resource_usage_for_squid	icp.q_kbytes_sent
Connection	Select_loop_called	icp.queries_recv
Connection_information_for_squid	Storage_Mem_capacity	icp.queries_sent
Content-Type	Storage_Mem_size	icp.query_median_svc_time
Current_Time	Storage_Swap_capacity	icp.query_timeouts
DNS_Lookups	Storage_Swap_size	icp.r_kbytes_recv
File_descriptor_usage_for_squid	Store_Disk_files_open	icp.r_kbytes_sent
Files_queued_for_open	Total_accounted	icp.replies_queued
Free_Ordinary_blocks	Total_free	icp.replies_recv
Free_Small_blocks	Total_in_use	icp.replies_sent
HTTP_Requests_(All)	Total_size	icp.reply_median_svc_time
Hits_as_%_of_all_requests	Total_space_in_arena	median_select_fds
Hits_as_%_of_bytes_sent	UP_Time	memPoolAlloc_calls
Holding_blocks	aborted_requests	memPoolFree_calls
ICP_Queries	average_select_fd_period	memPool_accounted
Internal_Data_Structures	client_http.all_median_svc_time	memPool_unaccounted
Largest_file_desc_currently_in_use	client_http.errors	page_faults
Last-Modified	client_http.hit_median_svc_time	sample_end_time
Maximum_Resident_Size	client_http.hits	sample_start_time
Maximum_number_of_file_descriptors	client_http.kbytes_in	select_fds
Mean_Object_Size	client_http.kbytes_out	select_loops
Number_of_HTCP_messages_received	client_http.miss_median_svc_time	server.all.errors
Number_of_HTCP_messages_sent	client_http.nh_median_svc_time	server.all.kbytes_in
server.all.kbytes_out	server.ftp.kbytes_in	syscalls.disk.closes
server.all.requests	server.ftp.kbytes_out	syscalls.disk.opens
server.ftp.errors	server.ftp.requests	syscalls.disk.reads

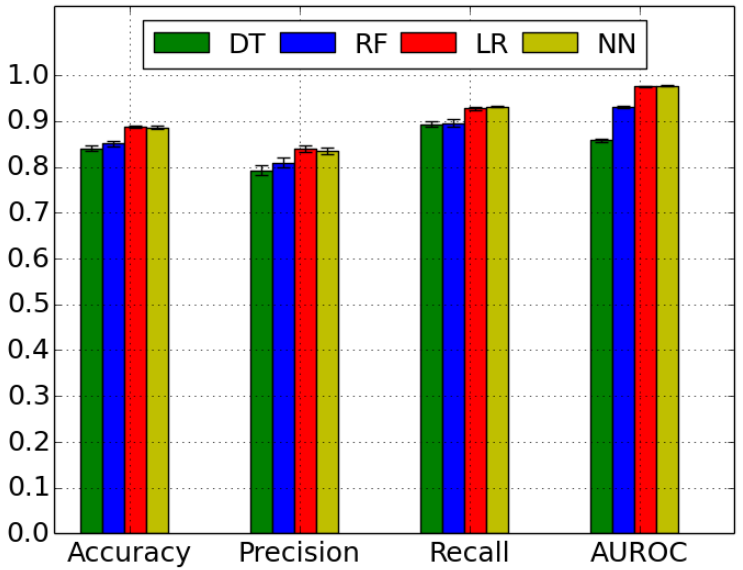
Evaluation Methodology

- Training and testing
  - Train neural network model with n workload types, $n = 1,2, \dots, 9$
  - Run 5-fold cross-validation to verify trained model
  - Test the trained model on $10 - n$ workload types
- Metrics
  - Accuracy: $\frac{correct\ predictions}{total\ predictions}$ for overall correctness
  - Precision: $\frac{true\ \ positives}{true\ \ positives + false\ \ positives}$ for exactness
  - Recall: $\frac{true\ \ positives}{true\ \ positives + false\ \ negatives}$ 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

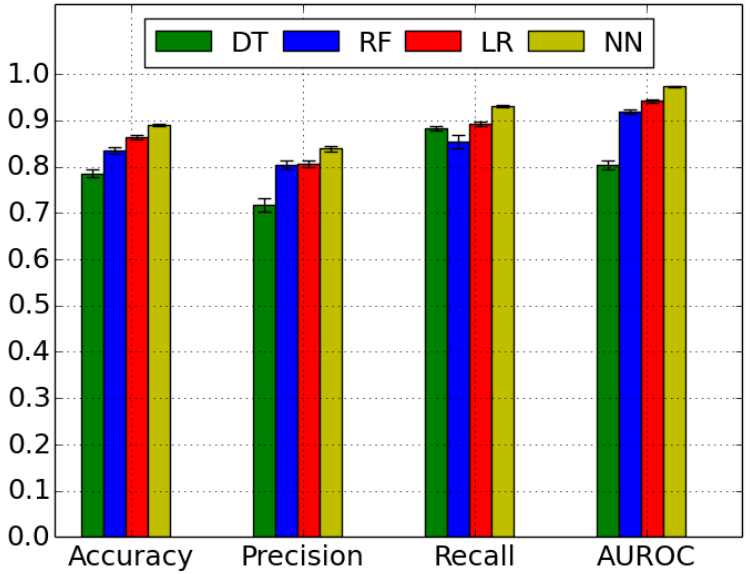
Suricata Results

Infrastructure-level features



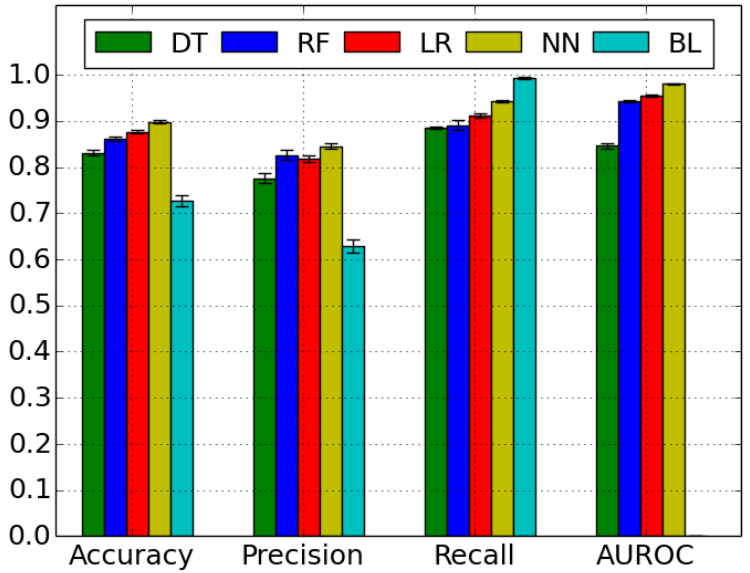
- NN ≈ LR > RF > DT
- 0.85 for all metrics

VNF-level features



- 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

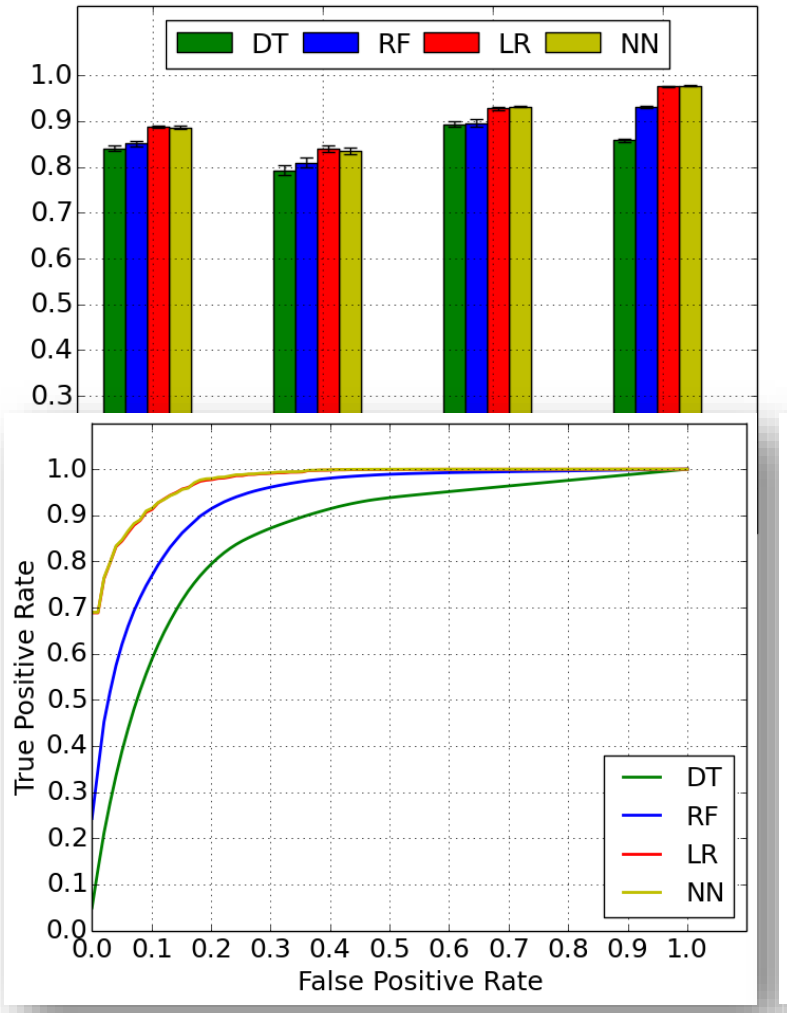
Composite features



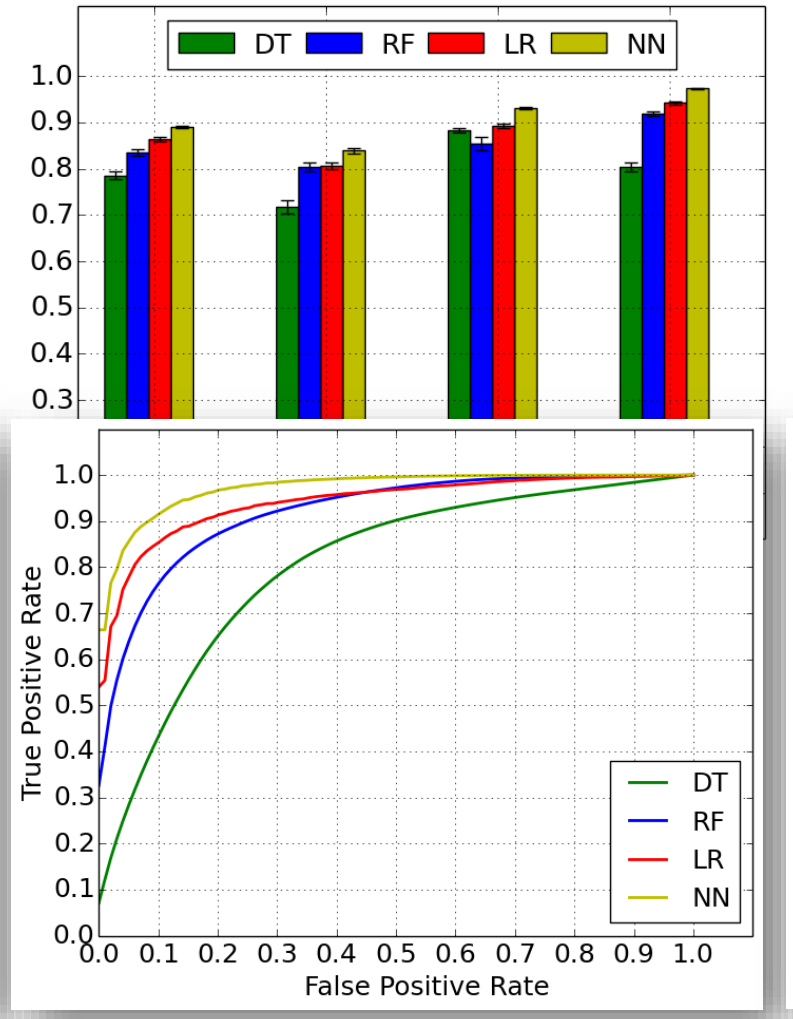
- NN > LR > RF > DT

Suricata Results

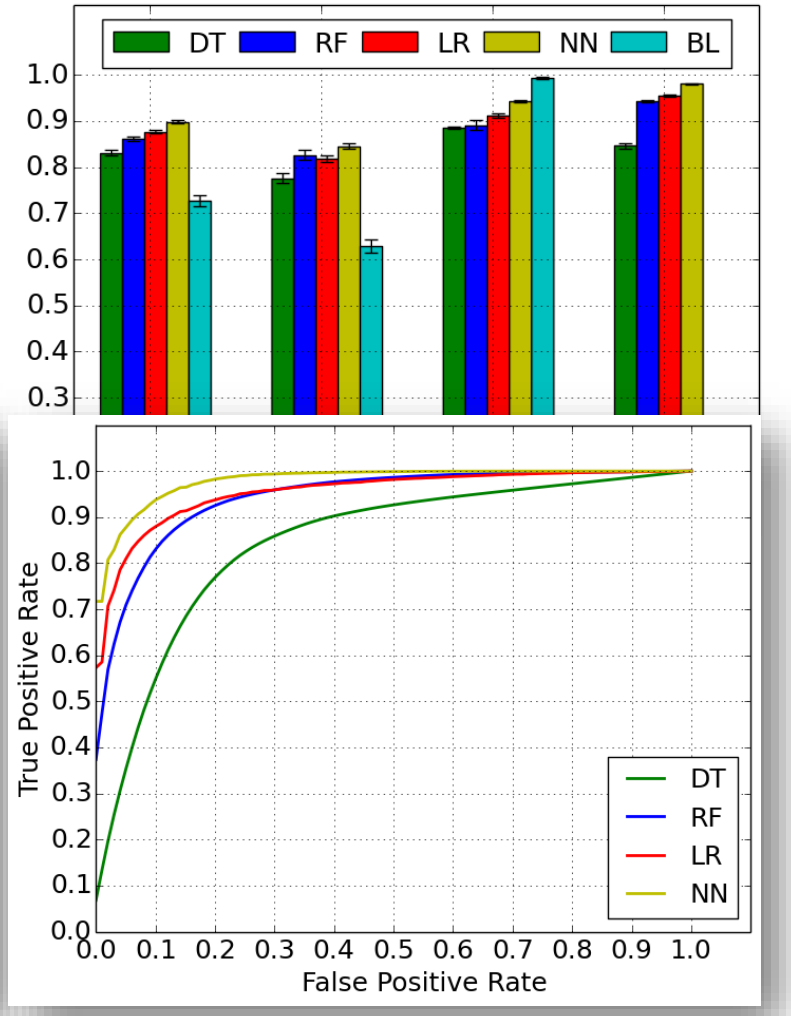
Infrastructure-level features



VNF-level features

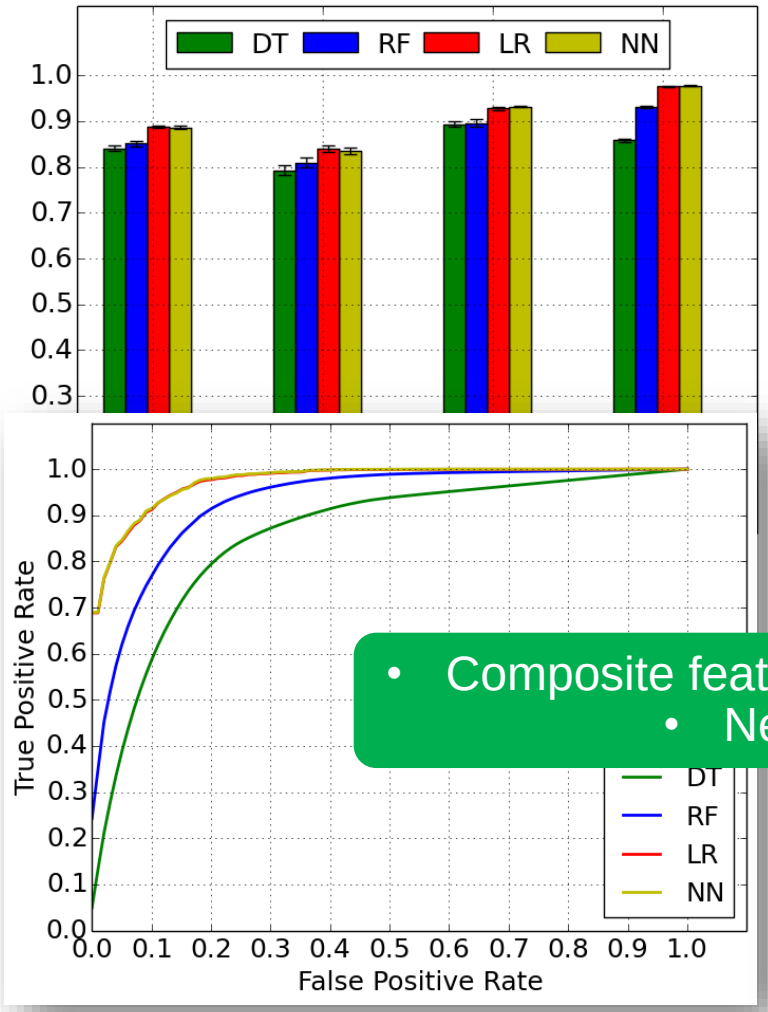


Composite features

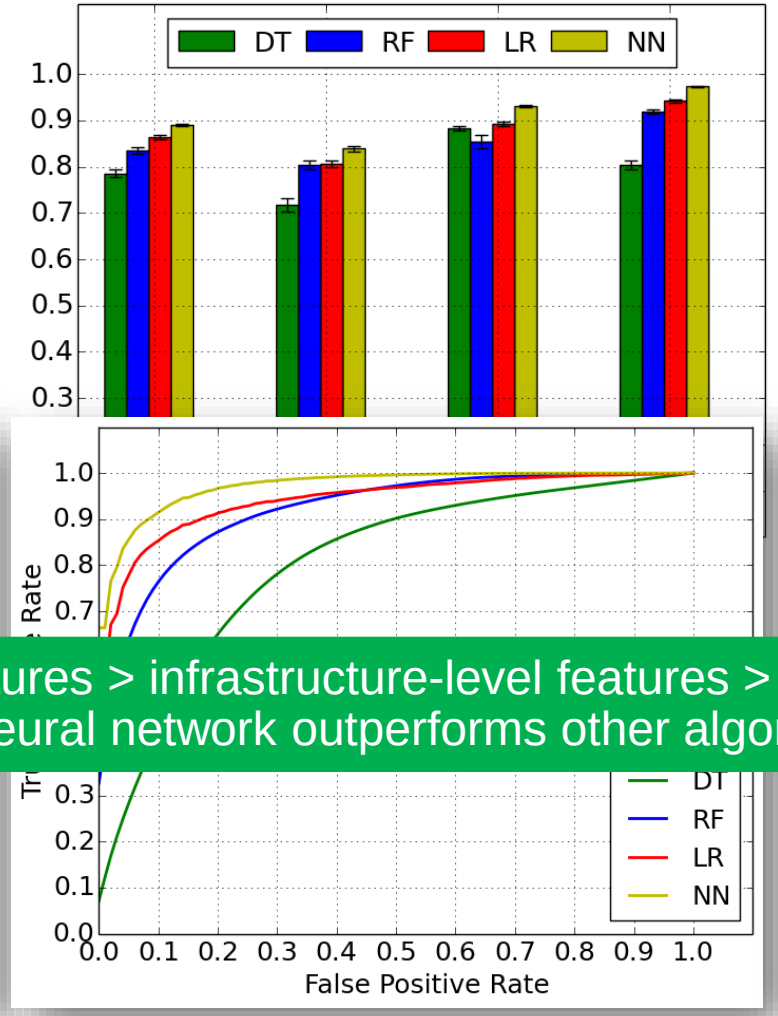


Suricata Results

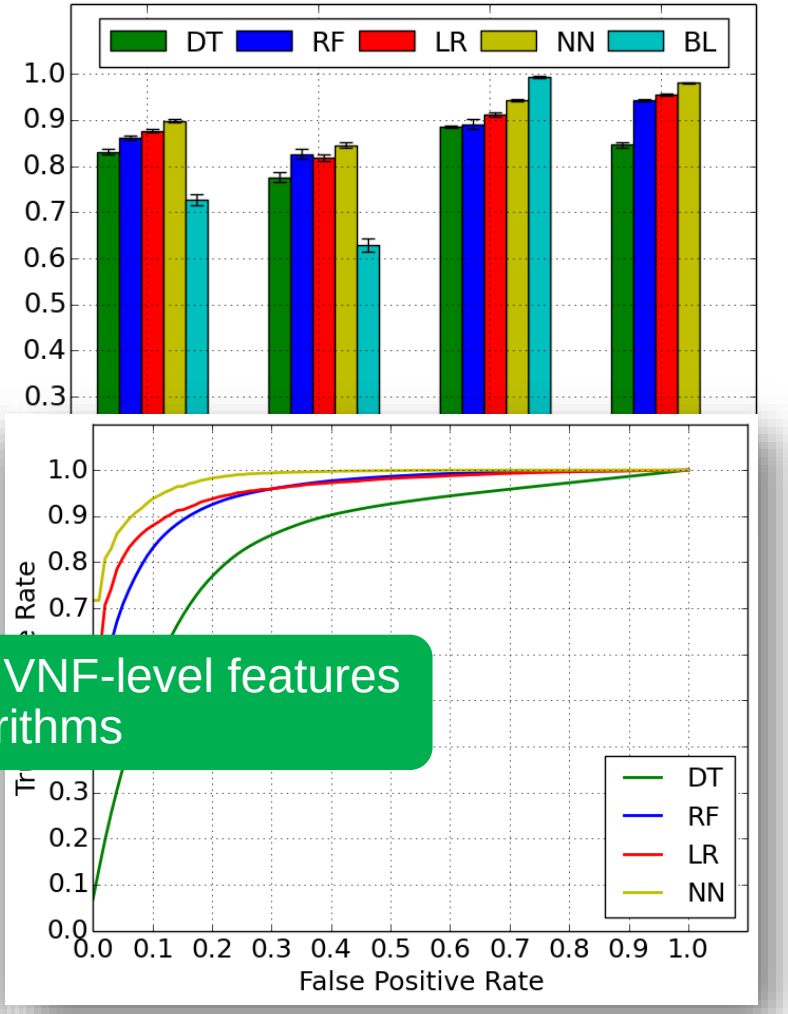
Infrastructure-level features



VNF-level features



Composite features

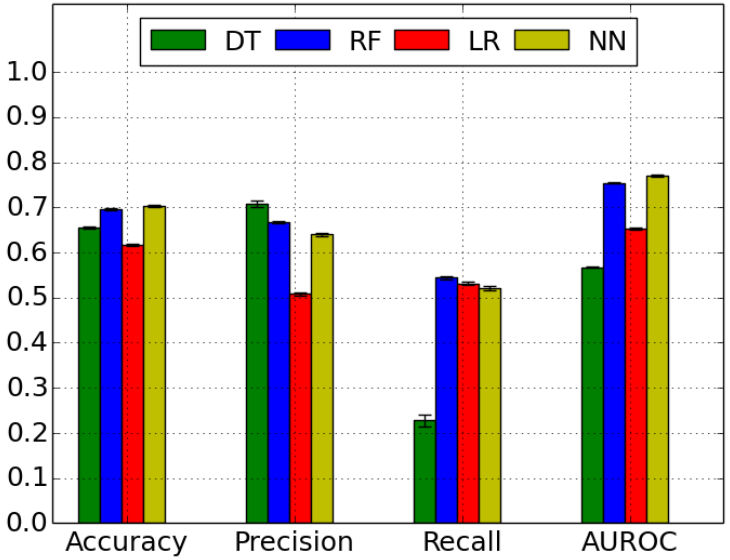


- Composite features > infrastructure-level features > VNF-level features
- Neural network outperforms other algorithms

Squid Results

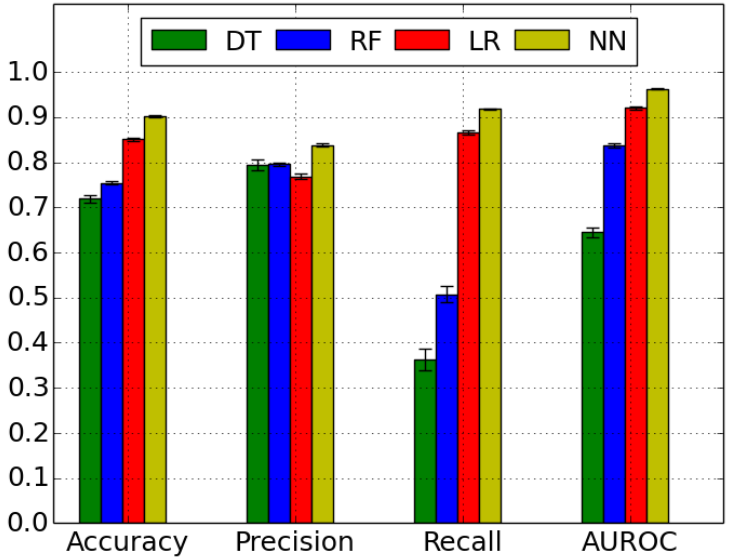
Squid Results

Infrastructure-level features



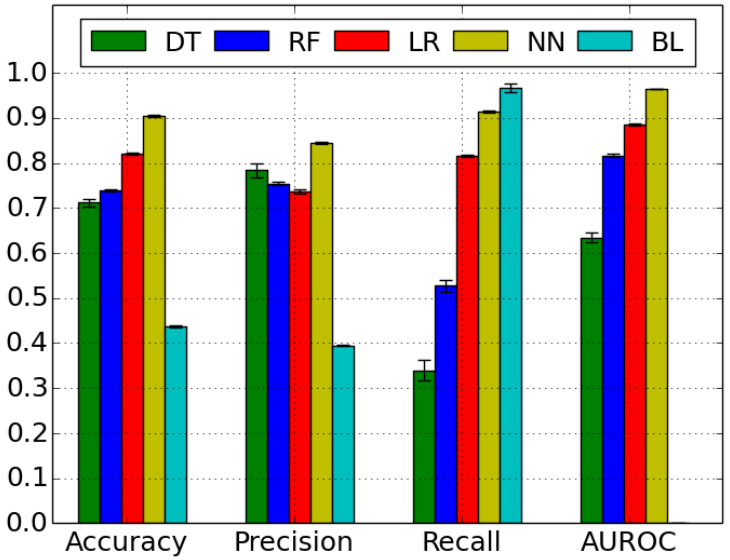
- Low for all models and metrics
- Squid is a relatively complex VNF
- Infrastructure resource usage is not adequate

VNF-level features



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

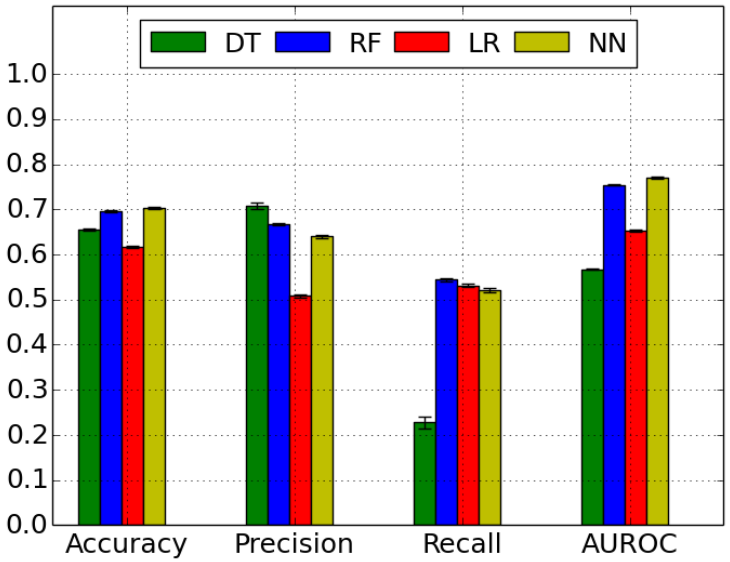
Composite features



- NN > LR > RF > DT

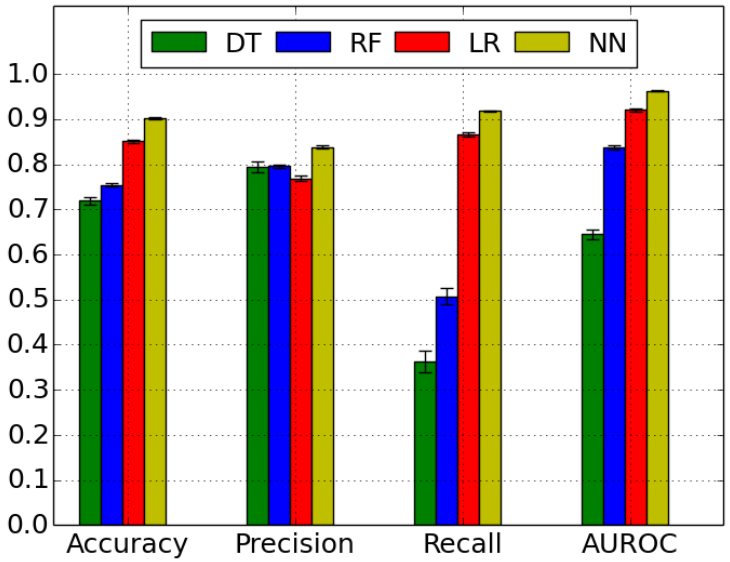
Squid Results

Infrastructure-level features



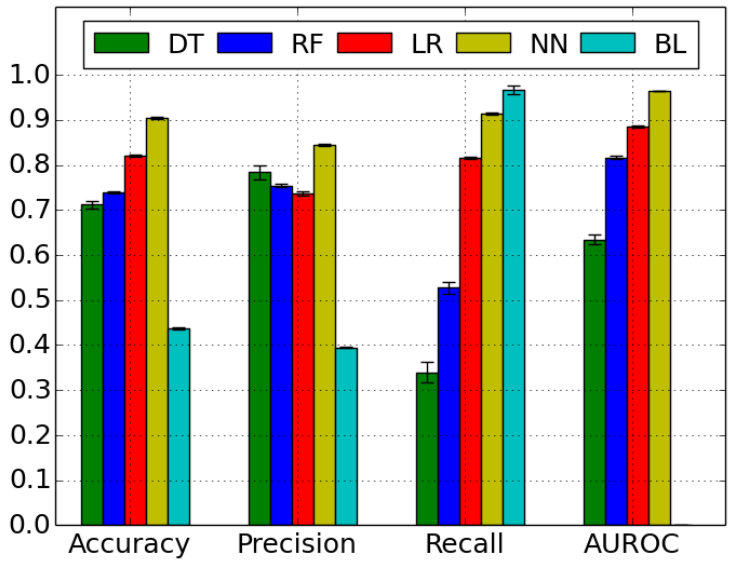
- Low for all models and metrics
- Squid is a relatively complex VNF
- Infrastructure adequate

VNF-level features



- 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?