VMware NSX Advanced Load Balancer (formerly Avi Networks) powered by Intel Xeon Scalable processors distributed over 1 million SSL transactions per second

Instead of purchasing, installing, and supporting appliance-based load balancers to direct your organization's web traffic, you can turn to a software-defined solution. Organizations enjoying the benefits of the software-defined data center (SDDC) transformation, including increased agility, scalability, and security, can seek those same benefits for load balancing with VMware NSX® Advanced Load Balancer™ (formerly from Avi Networks, now part of VMware).

In the Principled Technologies data center, we used a cluster of 16 Intel® Xeon® Scalable processor-powered servers with 64 virtual load balancers to explore how many SSL transactions per second the solution could handle. Processing and offloading encrypted transactions is a computationally intensive duty, but the VMware + Intel solution rose to the task, handling an average of 1.085 million SSL transactions per second using 64 VMware NSX (Avi Networks) distributed load balancers to represent a single virtual service.

This means that even at times of heavy traffic, the elastically scalable VMware NSX Advanced Load Balancer (Avi Networks) solution on Intel Xeon Scalable processor-powered servers can keep transactions moving and web traffic flowing smoothly.

Load balance over
1 million SSL transactions per second

Respond to network traffic fluctuations on demand with
elastic load balancing software

Dealing with peak demand is vital

Though peak times vary by industry, current events, and context, many organizations experience bursts of activity with many users accessing their applications. For example, ecommerce retail sites experience heavier-than-normal demand around Black Friday and Cyber Monday sales as record numbers of users make purchases. Colleges and universities see heavy web traffic at the beginning of the semester as students register for classes and make payments. Tax season, from the start of the year through mid-April, can strain resources for accountants, online tax preparers, and the IRS.

What is SSL/TLS?

SSL stands for secure sockets layer, and the more modern version TLS stands for transport layer security. SSL/TLS transactions establish authenticated and encrypted links between client and server. Because they involve encryption for security, SSL/TLS transactions require more computational power to complete than those transactions that don't require encryption.

Why is it important to consider peak web traffic when investing in data center resources? The inability to process peak-time transactions responsively can result in poor customer satisfaction and cause frustrated end users to abandon transactions, leading to lost business. While necessary, provisioning for peak times can result in an inefficient use of hardware throughout normal business cycles where hardware sits unused, taking up space and power, waiting for periods of higher usage.

Directing web traffic with load balancers

In a distributed and complex infrastructure, how do web transactions know to spread out and use resources from all available servers rather than just bottlenecking on one and creating a traffic jam? Load balancers handle this task, diverting web traffic to various servers to keep transactions flowing so customers don't experience delays. Traditionally, active-standby pairs of appliance-based load balancers have been used for load balancing. But this results in wasteful overprovisioning with unused capacity during non-peak times. In addition, enterprises using appliance-based solutions experience delays when procuring and deploying load balancing and must also overcome the challenges of operational complexity managing individual appliances and inconsistent solutions for data centers and public clouds.

About VMware NSX Advanced Load Balancer

VMware provides a software-defined architecture for load balancing with a single point of management, the VMware NSX Advanced Load Balancer Controller, and a distributed data plane of software load balancers called VMware NSX Advanced Load Balancer Service Engines (formerly known as Avi Service Engines). This active-active fabric allows for automatic placement of load balancers when needed, delivering on-demand scalability as network demands fluctuate.

According to VMware, NSX Load Balancer architecture offers per-application or per-tenant load balancing capabilities, and uses a flexible licensing model based on the number of CPU cores used for the load balancers in the data plane.[1] VMware collects and analyzes real-time application traffic data through telemetry sent by the VMware NSX Advanced Load Balancer Service Engines to the VMware NSX Advanced Load Balancer Controller. VMware analytics provide application health and performance insights which help in troubleshooting application issues and informs automation decisions such as autoscaling load balancers and fault recovery. These analytics also include End-to-End Timing, which measures the latency of a request from the client through the load balancer to the application server, giving administrators a detailed look at system response times at each hop.

VMware states that organizations can deploy these load balancers on bare metal servers, in virtual machines, or containers in the data center or in the cloud and can automate infrastructure in conjunction with most popular cloud controllers.[2]

To learn more, visit https://www.vmware.com/products/nsx-advanced-load-balancer.html.

Moving load balancing into the software-defined data center with VMware on 2nd Generation Intel Xeon Scalable processor-powered servers

To show that a software-defined solution with VMware NSX Advanced Load Balancers on servers powered by 2nd Generation Intel Xeon Scalable processors can handle many SSL transactions per second (TPS), we used a large configuration and recorded the SSL transactions per second that the solution distributed.

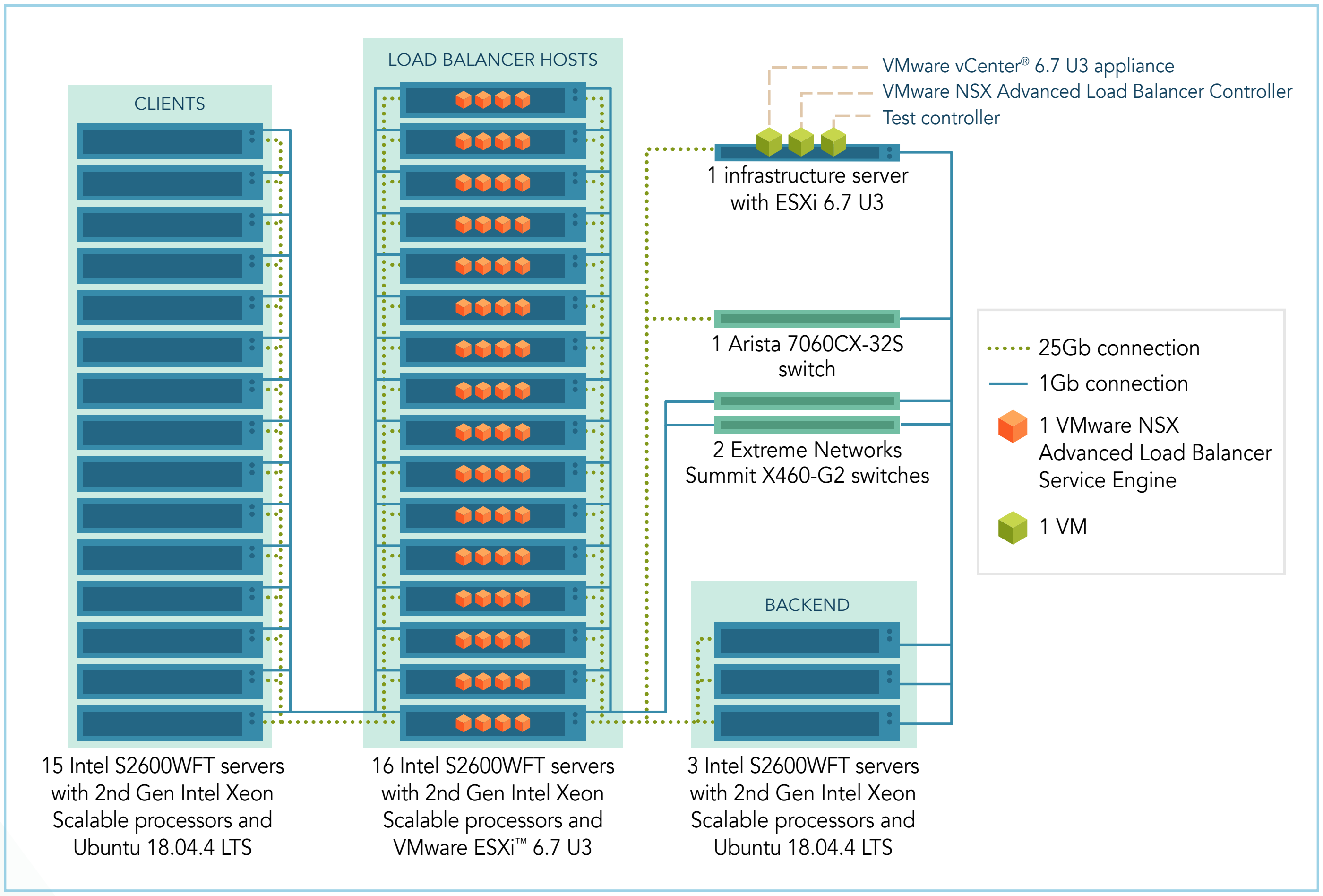

CLIENTS
LOAD BALANCER HOSTS
VMware vCenter® 6.7 U3 appliance
VMware NSX Advanced Load Balancer Controller
Test controller
1 infrastructure server with ESXi 6.7 U3
1 Arista 7060CX-32S switch
2 Extreme Networks Summit X460-G2 switches
BACKEND
25Gb connection
1Gb connection
1 VMware NSX Advanced Load Balancer Service Engine
1 VM
15 Intel S2600WFT servers with 2nd Gen Intel Xeon Scalable processors and Ubuntu 18.04.4 LTS
16 Intel S2600WFT servers with 2nd Gen Intel Xeon Scalable processors and VMware ESXi™ 6.7 U3
3 Intel S2600WFT servers with 2nd Gen Intel Xeon Scalable processors and Ubuntu 18.04.4 LTS

Figure 1: Diagram of the solution we tested. Source: Principled Technologies.

In our tests, we used 35 servers: 15 acting as clients, 16 VMware ESXi™ hosts for VMware NSX Advanced Load Balancers, three backend web servers, and one infrastructure server (see Figure 1). Client machines generated TCP connections to the backend server using OpenSSL on Ubuntu. Each backend server ran a web server on Ubuntu and hosted a simple website for the clients to target. The ESXi servers each hosted four VMware NSX Advanced Load Balancer Service Engines that handled the load balancing distribution. The infrastructure server ran ESXi and hosted the vCenter server, the test controller VM, and the VMware NSX Advanced Load Balancer Controller.

up to **1.17 MILLION SSL** transactions per second*

*steady state at 1.085 million

About 2nd Generation Intel Xeon Scalable processors

The 2nd Generation Intel Xeon Scalable processor platform offers multiple tiers of performance to support specific workloads, labeling them Bronze, Silver, Gold, and Platinum to help customers fit their needs. According to Intel, the 2nd Generation Intel Xeon Scalable platform can handle a variety of workloads, including enterprise, cloud, HPC, storage, and communications.[3] This new processor line also supports a new memory and storage technology to further accelerate workloads, Intel Optane™ DC persistent memory.

To learn more about the 2nd Generation Intel Xeon Scalable processor family, visit https://www.intel.com/content/www/us/en/products/processors/xeon/scalable.html.

The VMware and Intel Xeon Scalable processor-powered solution we tested handled a steady 1.085 million SSL TPS, proving that this software-defined load balancing solution can handle encrypted web traffic even during heavy usage. As the screenshot from our test shows (see Figure 1 in the science behind the report), RTT latency for clients was at 2 milliseconds, RTT latency for servers was 7 milliseconds, and app response was under 1 millisecond. These response times reflect that the configuration was able handle the goal of 1 million TPS that it was designed to achieve. According to VMware, leveraging Intel Xeon processors with AES-NI[4] to offload more cryptographic workloads into dedicated rather than general-purpose instructions could enhance VMware NSX Advanced Load Balancer performance. For maximum performance on a single SE, VMware recommends using the latest Intel NICs that support DPDK for fabric elasticity without loss of capacity and linear scale from Intel 2nd Generation Intel Xeon Scalable processors.[5]

Conclusion

VMware NSX Advanced Load Balancers, powered by 2nd Generation Intel Xeon Scalable processors, directed over 1 million SSL TPS. This strong performance indicates that an VMware-Intel solution could handle large numbers of encrypted transactions.

Moving from appliance-based load balancers to software-defined VMware NSX Advanced Load Balancers could enable your organization to modernize load balancing services with efficient use of standard computing infrastructure and reduce overprovisioning. Because VMware can elastically scale load balancing capacity up or down based on demand, your applications can better utilize available compute power from Intel Xeon Scalable processors.

For enterprises moving to software-defined data centers, the combination of VMware NSX Advanced Load Balancer deployed on servers with Intel Xeon Scalable processors represents a high-performance solution to load balance large volumes of encrypted traffic.

1 VMware, "VMware NSX Advanced Load Balancer (Avi Networks)," accessed March 10, 2020, https://www.vmware.com/products/nsx-advanced-load-balancer.html.

2 VMware, "VMware NSX Advanced Load Balancer (Avi Networks)," accessed March 10, 2020.

3 Intel, "Intel Xeon Scalable processors," accessed March 10, 2020, https://www.intel.com/content/www/us/en/products/processors/xeon/scalable.html.

4 Intel, "Intel Data Protection Technology with AES-NI and Secure Key," accessed March 23, 2020, https://www.intel.com/content/www/us/en/architecture-and-technology/advanced-encryption-standard-aes/data-protection-aes-general-technology.html.

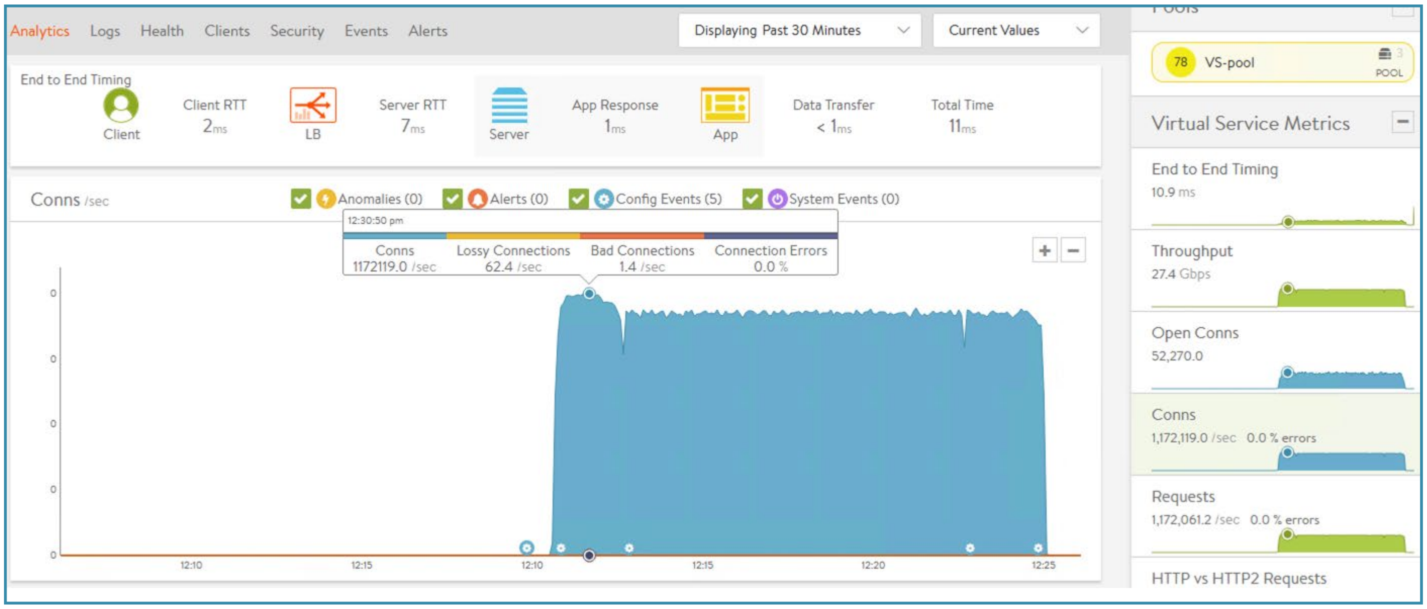
5 VMware, "SSL Performance," accessed March 23, 2020, https://avinetworks.com/docs/18.2/ssl-performance/.

We concluded our hands-on testing on March 18, 2020. During testing, we determined the appropriate hardware and software configurations and applied updates as they became available. The results in this report reflect configurations that we finalized on March 18, 2020 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

Table 1: Performance during a 10-minute run with 15 clients and 64 VMware NSX Advanced Load Balancer Service Engines.

Peak SSL TPS	1,172,119
Average SSL TPS at steady state	1,085,665



End to End Timing: Client RTT 2ms; Server RTT 7ms; App Response 1ms; Data Transfer < 1ms; Total Time 11ms

Anomalies (0) Alerts (0) Config Events (5) System Events (0)

12:30:50 pm

Conns	Lossy Connections	Bad Connections	Connection Errors
1172119.0 /sec	62.4 /sec	1.4 /sec	0.0 %

78 VS-pool

End to End Timing 10.9 ms

Throughput 27.4 Gbps

Open Conns 52,270.0

Conns 1,172,119.0 /sec 0.0 % errors

Requests 1,172,061.2 /sec 0.0 % errors

HTTP vs HTTP2 Requests

Figure 1: Screenshot of the Analytics screen for the virtual service in the VMware NSX Advanced Load Balancer Controller UI during the test run. As the screenshot from our test shows, RTT latency for clients was at 2 milliseconds, RTT latency for servers was 7 milliseconds, and app response was under 1 millisecond. These response times reflect that the configuration was able handle the goal of 1 million TPS that it was designed to achieve.

System configuration information

Table 2: The table below presents detailed information on the systems we tested.

Server configuration information	34 Intel Corporation S2600WFT
BIOS name and version	SE5C620.86B.02.01.0010.010620200716
Non-default BIOS settings	VMware ESXi™ hosts: Virtualization enabled All servers: Disable SpeedStep and Hardware P-States. Set Package C-State to C0/C1 state. Fan profile set to Performance. Fan PWM Offset set to 50%.
Operating system name and version/build number	Clients/Backend: Ubuntu 18.04.4 LTS (GNU/Linux 4.15.0-88-generic x86_64) ESXi hosts: VMware ESXi, 6.7.0, 15160138
Date of last OS updates/patches applied	02/26/2020
Power management policy	Performance
Processor	
Number of processors	2
Vendor and model	Intel® Xeon® Gold 6242
Core count (per processor)	16
Core frequency (GHz)	2.10
Stepping	7
Memory module(s)	
Total memory in system (GB)	192
Number of memory modules	12
Vendor and model	Micron MTA18ASF2G72PDZ-2G9E1
Size (GB)	16
Type	PC4-2933
Speed (MHz)	2,933
Speed running in the server (MHz)	2,933
Local storage (type A)	
Number of drives	1
Drive vendor and model	Intel SSD D3-S4610
Drive size (GB)	240
Drive information (speed, interface, type)	6Gb/s, SATA, SSD
Network adapter	
Vendor and model	Intel Ethernet Network Adapter X722
Number and type of ports	2x 10 GbE
Driver version	i40e 2.1.14-k

Server configuration information	34 Intel Corporation S2600WFT
Network adapter	
Vendor and model	Intel Ethernet Network Adapter XXV710
Number and type of ports	2 x 25 GbE
Driver version	i40e 2.1.14-k
Cooling fans	
Vendor and model	Nidec UltraFlo V60E12BS1B5-07A016
Number of cooling fans	6
Power supplies	
Vendor and model	Intel PSSF132202A
Number of power supplies	1
Wattage of each (W)	750

How we tested

Configuring the network switches for testing

We configured three switches for testing: two Extreme Networks Summit X460-G2 1Gb switches and one Arista 7060CX-32S 100Gb switch. We configured the two Extreme Networks switches with default settings for our management network, and cabled one port for management traffic and one port for the BMC on every server in the test bed. The Arista switch supported the 25Gb private test network for the VMware NSX Advanced Load Balancers. We connected 100Gb to 4 x 25Gb breakout cables to the switch and cabled a single 25Gb port on each server in the test bed. We configured each port to the default settings, but applied the following to configure BGP routing and VLAN 1 for testing.

```
interface Vlan1
   ip address 192.168.1.100/24
!
ip routing
!
router bgp 65000
   timers bgp 100 300
   maximum-paths 64 ecmp 64
   bgp listen range 192.0.0.0/8 peer-group Avi remote-as 65000
   neighbor Avi peer-group
   neighbor Avi maximum-routes 12000
```

Preparing the ESXi hosts

We used the following steps to configure the sixteen ESXi hosts we used for the VMware NSX Advanced Load Balancer Service Engines.

Installing VMware ESXi 6.7 U3

1. Attach the installation media.
2. Boot the server.
3. At the VMware Installer screen, press Enter.
4. At the EULA screen, to Accept and Continue, press F11.
5. Under Storage Devices, select the appropriate virtual disk, and press Enter.
6. At the keyboard layout, select US, and press Enter.
7. Enter the root password twice, and press Enter.
8. To start installation, press F11.
9. After the server reboots, press F2, and enter root credentials.
10. Select Configure Management Network, and press Enter.
11. Select the appropriate network adapter, and select OK.
12. Select IPv4 settings, and enter the desired IP address, subnet mask, and gateway for the server.
13. Select OK, and restart the management network.
14. Repeat steps 1 through 13 on the rest of the servers.

Deploying the VMware vCenter® Server 6.7 U3

1. On a Windows server or VM, locate the VMware-VCSA installer image.
2. Mount the image, navigate to the vcsa-ui-installer folder, and double-click win32.
3. Double-click installer.exe.
4. Click Install.
5. Click Next.
6. Accept the terms of the license agreement, and click Next.
7. Leave the default vCenter Server with an Embedded Platform Services Controller selected, and click Next.
8. Enter the FQDN or IP address of the host onto which the vCenter Server Appliance will be deployed.
9. Provide the server username and password, and click Next.
10. Accept the certificate of the host you chose to connect to by clicking Yes.
11. Provide a name and password for the vCenter Appliance, and click Next.
12. Set an appropriate Appliance Size, and click Next.
13. Select the appropriate datastore, and click Next.

14. At the Configure Network Settings page, configure the network settings as appropriate for your environment, and click Next.
15. Review your settings, and click Finish.
16. When the deployment completes, click Next.
17. At the Introduction page, click Next
18. At the Appliance configuration page, select the time synchronization mode and SSH access settings, and click Next.
19. Select Create a new SSO domain.
20. Provide a password, and confirm it.
21. Provide an SSO Domain name and SSO Site name, and click Next.
22. At the CEIP page, click Next.
23. At the Ready to complete page, click Finish.
24. When installation completes, click Close.
25. Using the vSphere web client, log into the vCenter server using the credentials previously provided.

Creating a cluster and adding the hosts to VMware vCenter

1. Once logged into the vCenter, navigate to Hosts and Clusters.
2. Select the primary site management vCenter.
3. Right-click the vCenter object, and select New Datacenter...
4. Enter a name for the new data center, and click OK.
5. Right-click the new data center, and click New Cluster...
6. Enter a name for the new cluster, and click OK.
7. Right-click the new cluster, and click Add hosts…
8. Enter the IP address and credentials for each ESXi server that will host VMware NSX Advanced Load Balancer service engines, and click Next.
9. Review the list of hosts, and click Next.
10. Click Finish.

Configuring each host for high performance

1. From the vCenter web UI, select a host on the left-hand side.
2. Navigate to Configure→Hardware→Power Management.
3. Click Edit.
4. Select High performance, and click OK.
5. Navigate to Configure→System→Advanced System Settings.
6. Click Edit.
7. Set Power.UseCStates and Power.UsePStates to 0.
8. Click OK.

Creating distributed switches for VMware NSX Advanced Load Balancer testing

We used the following steps to create two distributed switches for the VMware NSX Advanced Load Balancer environment: one for the 1Gb management network and one for the 25Gb data network. We named each distributed switch according to its purpose and selected the appropriate physical ports associated with each network.

1. From the vCenter web UI, navigate to Menu→Networking.
2. Right-click the datacenter, and click Distributed Switch→New Distributed Switch.
3. Enter a name for the Distributed Switch, and click Next.
4. Select 6.6.0 – ESXi 6.7 and later for the version, and click Next.
5. Enter a name for the port group, and click Next.
6. Review the setting for the new distributed switch, and click Finish.
7. On the left-hand side, right-click the new distributed switch.
8. Click Add and Management Hosts…
9. Select Add hosts, and click Next.
10. Click + New hosts…
11. Select all the hosts that will host VMware NSX Advanced Load Balancer service engines, and click OK.
12. Click Next.
13. Select the appropriate physical vmnic, and click Assign uplink.

14. Select the first uplink, check Apply this uplink assignment to the rest of the hosts, and click OK.
15. On the Manage VMkernel adapters, click Next.
16. On the Migrate VM networking, click Next.
17. Review the settings, and click Finish.
18. Repeat these steps for the second distributed switch.

Preparing the backend servers

We used the following steps to configure the three backend target servers using Nginx to host a simple website.

Installing Ubuntu 18.04.4 LTS

1. Boot the server to the Ubuntu 18.04 installation media.
2. Select English as the preferred language, and press Enter.
3. Select English (US) as the keyboard layout, and press Enter.
4. Select Install Ubuntu, and press Enter.
5. Ensure the 1Gb adapter has DHCP and the 40Gb adapter has no address, and press Enter.
6. To skip the proxy address page, press Enter.
7. To accept the default mirror address, press Enter.
8. Select Use an Entire Disk, and press Enter.
9. Select the M.2 SSD, and press Enter.
10. Select Done, and press Enter.
11. Select Continue, and press Enter.
12. Enter a name, hostname, username, and password. Select Done, and press Enter.
13. Select Done, and press Enter.
14. When the installation completes, remove the installation media, and reboot the server.

Updating and configuring the operating system

1. Log into the operating system as the user created in the previous section.
2. Disable the system firewall by running the following command:
   ```
   sudo ufw disable
   ```
3. Update the operating system and reboot by running the following commands:
   ```
   sudo apt-get -y update
   sudo apt-get -y upgrade
   sudo reboot
   ```
4. Run the scripts for configuring and tuning the backend servers found in the Configuration and test scripts section of this report.

Preparing the client servers

We used the following steps to configure the fifteen client servers used to generate the client connections to the target servers.

Installing Ubuntu 18.04.4 LTS

1. Boot the server to the Ubuntu 18.04 installation media.
2. Select English as the preferred language, and press Enter.
3. Select United States as your location, and press Enter.
4. When prompted to auto-detect keyboard layout, select No, and press Enter.
5. Select English (US) as the keyboard layout, and press Enter.
6. Cancel the network DHCP autoconfiguration, and select Configure network manually.
7. Configure the network settings as appropriate for your environment, and click Next.
8. Provide the desired hostname for the system, and press Enter.
9. When prompted to set up a user account, enter desired full name, username, password, and password confirmation, and press Enter.
10. When prompted to confirm timezone, ensure it is correct, and press Enter.
11. Select Guided - use entire disk and set up LVM, and press Enter.

12. Select the disk to partition, and press Enter.
13. On the partition disks confirmation screen, select Yes, and press Enter.
14. Select Continue, and press Enter to use the entire disk.
15. On the confirmation screen, select Yes, and press Enter.
16. Leave the HTTP proxy information field blank, and press Enter.
17. Select No automatic updates, and press Enter.
18. Leave all boxes unchecked, select Continue, and press Enter.
19. When the installation completes, remove the installation media, and reboot the server.

Updating and configuring the operating system

1. Log into the operating system as the user created in the previous section.
2. Disable the system firewall by running the following command:

   ```
   sudo ufw disable
   ```

3. Update the operating system and reboot by running the following commands:

   ```
   sudo apt -y upgradesudo apt -y updatesudo reboot
   ```

4. Run the scripts for configuring and tuning the client servers found in the Configuration and test scripts section of this report.

Preparing the VMware NSX Advanced Load Balancer Controller environment

Deploying the VMware NSX Advanced Load Balancer Controller VM

1. In vCenter, right-click the infrastructure host, and select Deploy OVF Template.
2. Select the location of the OVA file, and click Next.
3. Choose a name and target location for the controller VM, and click Next.
4. Select a compute resource for the controller VM, and click Next.
5. Review details, and click Next.
6. Select Thick Provision Eager Zeroed as the virtual disk format, select storage to deploy the VM to, and click Next.
7. Choose a management network for the controller VM, and click Next.
8. Choose appropriate network settings for the VM, and click Next.
9. Click Finish.

Configuring the VMware NSX Advanced Load Balancer Controller

1. Once the VMware NSX Advanced Load Balancer Controller completes its initial startup, open a web browser, and navigate to the VMware NSX Advanced Load Balancer Controller IP address to load the web UI.
2. Enter and confirm a password for the admin account, and click Create Account.
3. Enter the DNS and NTP information for the controller, and click Next.
4. Select VMware for the orchestrator integration, and click Next.
5. Enter the vCenter IP and credentials, set the permissions to Wr ite, and click Next.
6. Select the data center where VMware NSX Advanced Load Balancer will be deployed, and click Next.
7. Select the management network for VMware NSX Advanced Load Balancer, and click Next.
8. On the Tenant Settings screen, click No.
9. Once the VMware NSX Advanced Load Balancer Controller setup completes, it will redirect to the dashboard for the web UI.

Configuring the VMware NSX Advanced Load Balancer profiles for testing

1. From the VMware NSX Advanced Load Balancer Controller web UI, navigate to Templates.
2. Under Profiles→Application, click the pencil icon for System-Secure-HTTP to edit.
3. On the General tab, check the following settings: Connection Multiplex, X-Forwarded-For, and WebSockets Proxy.
4. On the Security Tab, check the following settings: SSL Everywhere, HTTP-to-HTTPS Redirect, Secure Cookies, HTTP-only cookies, Rewrite Server Redirects to HTTPS, X-Forwarded-Proto, and HTTP Strict Transport Security (HSTS) (set duration to 365 days and include Subdomains).
5. On the DDoS tab, set Client Header timeout to 10,000.
6. Set Client Body Timeout, HTTP Keep-Alive Timeout, and Post Accept Timeout to 30,000.

7. Set Client Max Body Size to 0 KB, Client Max Header Field Size to 12 KB, and Client Max Complete Header Size to 48 KB.
8. Click Save.
9. Under Profiles→TCP/UDP, click the pencil icon for System-TCP-Proxy to edit.
10. For the Timeout sections, set Idle Connections to TCP keepalive, set Idle Duration to 60 seconds, and check Ignore Time Wait.
11. For Retransmission Behavior, ensure that Max and Max SYN Retransmissions are set to 8.
12. For Buffer Management, set the Receive Windows to 64 KB.
13. For Max Segment Size, check Use Interface MTU.
14. For QOS & Traffic Engineering, ensure that all settings are unchecked.
15. Click Save.
16. Under Security→SSL/TLS Profile, click the pencil icon for System-Standard.
17. For version, ensure that only TLS 1.2 is selected.
18. For Cypher type, ensure that List is selected, and enable only the following cyphers:
    - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
    - TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
    - TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
    - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
    - TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
    - TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
    - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
    - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
    - TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
    - TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
    - TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
    - TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
19. Click Save.

Creating a new application pool

1. From the VMware NSX Advanced Load Balancer Controller web UI, navigate to Applications→Pools.
2. Click Create Pool.
3. Enter a name for the new pool. We used VS-Pool for the name.
4. Check Enable real time metrics, and click Next.
5. Enter the IP, and click Add Server for each of the backend Nginx servers.
6. Click Next.
7. Leave the default settings on the Advanced page, and click Next.
8. Review the settings, and click Save.

Configuring the default Service Engine group

1. From the VMware NSX Advanced Load Balancer Controller web UI, navigate to Infrastructure→Service Engine Group.
2. Click the pencil icon for Default-Group to edit the settings.
3. On the Basic Settings tab, check the box for Real-Time Metrics, and set it to 0 minutes.
4. Under High Availability & Placement Settings, select N + M (buffer).
5. Select Compact for VS Placement across SEs.
6. Enter 1 for the maximum Virtual Services per Service Engine, and uncheck Per-app SE mode.
7. Check SE Self-Election.
8. Under Service Engine Capacity and Limit Settings, enter 64 for Max Number of Services Engines.
9. Enter 20 GB of memory, 11 vCPU, 40GB disk size, and check the Memory Reserve and CPU Reserve boxes.
10. On the Advanced tab, select Host for Host Scope Service Engine within.
11. Select Exclude, and enter the IPs of any servers to exclude from hosting Service Engines.
12. Under Advanced HA & Placement, enter 1 for Buffer Service Engines.
13. Enter a minimum and maximum of 64 for Scale per Virtual Service, and check the CPU socket Affinity and Dedicated dispatcher CPU boxes.
14. Click Save.

Configuring the infrastructure networks

1. From the VMware NSX Advanced Load Balancer Controller web UI, navigate to Infrastructure→Networks.
2. Verify that the two distributed switches created on the VMware cluster for use by VMware NSX Advanced Load Balancer are listed.
3. Verify that the management pool defined in the Default-Cloud settings is assigned to the right distributed switch.
4. Click the icon for the distributed switch used for VMware NSX Advanced Load Balancer data traffic.
5. Ensure that a network pool is configured for auto-assigning IPs on the VMware NSX Advanced Load Balancer data network.

Configuring BGP Peering in VMware NSX Advanced Load Balancer Controller

1. From the VMware NSX Advanced Load Balancer Controller web UI, navigate to Infrastructure→Routing.
2. Select the BGP Peering tab, and click the pencil icon.
3. Enter 65,000 for the BGP AS, and select iBGP.
4. For Timers, enter 60 seconds for Keepalive Interval, and 180 seconds for Hold Time.
5. For Placement Network, select the distributed switch for VMware NSX Advanced Load Balancer data traffic.
6. Enter the IPv4 Prefix for the VMware NSX Advanced Load Balancer data network (192.168.1.0/24), and enter the IPv4 Peer address that is configured on the 25Gb switch (192.168.1.100).
7. Check the box for Advertise VIP.
8. Click Save.

Configuring access settings

1. From the VMware NSX Advanced Load Balancer Controller web UI, navigate to Administration→Settings.
2. Click Access Settings, and click the pencil icon to edit.
3. Ensure that HTTPS Access to System, HTTP Access to System, and Redirect HTTP to HTTPS are selected.
4. Select System-Default-Portal-Cert and System-Default-Portal-Cert-EC256 for the SSL/TLS Certificate.
5. Select System-Default-Secure-Channel-Cert for the Secure Channel SSL/TLS Certificate.
6. Click Save.

Creating a new Virtual Service

1. From the VMware NSX Advanced Load Balancer Controller web UI, navigate to Applications.
2. Click Create Virtual Service→Advanced Setup.
3. Enter a name for the virtual service.
4. Under the VIP address settings, enter 3.3.3.3 for the IPv4 VIP.
5. Under the Service Port settings, enter 443, and check SSL.
6. Under the Profiles settings, select System-Secure-HTTP for the Application Profile.
7. Under the Pool settings, select VS-pool.
8. Under the SSL Settings, select System-Default-Cert-EC for the SSL Certificate.
9. Click Next.
10. Leave the policies blank, and click Next.
11. Set the Metric Update Frequency to 0.
12. Under Client Log Settings, set the Non-significant log duration to 0.
13. Click Next.
14. Under Other Settings, check Advertise VIP via BGP.
15. Set the SE Group to Default-Group, and click Save.

Configuring the default cloud

1. From the VMware NSX Advanced Load Balancer Controller web UI, navigate to Infrastructure→Clouds.
2. Click the pencil icon for Default-Cloud to edit.
3. On the Infrastructure tab, enter the IP, Username, and Password for the test vCenter.
4. On the Data Center tab, select the test datacenter that contains the host that the VMware NSX Advanced Load Balancer Service Engines will run on.
5. On the Network tab, select the distributed virtual switch to be used for VMware NSX Advanced Load Balancer management traffic.
6. Select the Default-Group as the Template Service Engine Group.
7. Enter the proper IP subnet, and create an IP pool in that range to be used by VMware NSX Advanced Load Balancer to auto-IP VMware NSX Advanced Load Balancer Service Engines.
8. Click Save.

Running the test

We ran the test scripts in the Configuration and test scripts section. We created a run_test.sh script to control the input and output of each test run. Using these scripts, we started OpenSSL to create two thousand connections from each client for a duration of ten minutes. We also used the scripts to gather stats from the VMware NSX Advanced Load Balancer Controller for reporting the number of connections per second and connection distribution to the VMware NSX Advanced Load Balancer Service Engines. Additionally, the VMware NSX Advanced Load Balancer Controller web UI provides graphs showing real-time statistics in the environment, including connections per second, throughput, and end-to-end timing, which we took screenshots of for data reporting.

Configuration and test scripts

We ran the following scripts from a Linux controller VM connected to the test environment. Intel and VMware provided these test scripts and we verified the settings and commands used in them. The OpenSSL and Mellanox files are publicly available online. The exact files used for our test are available upon request. Email info@principledtechnologies.com for access.

Configuration scripts

We ran the following configuration scripts after installing Ubuntu on the clients and backend servers.

set_ips.sh

Sets the private IP and route info for the client and backend servers. Always run after reboot.

```
#!/bin/bash

#set client ips and route
for i in 'seq 21 35'; do
        echo $i
        ssh root@svr$i ifconfig enp24s0f0 192.168.1.$i/24
        ssh root@svr$i ip route add 3.3.3.3/32 via 192.168.1.100 dev enp24s0f0
done

#set backend ips
for i in 'seq 36 38'; do
        echo $i
        ssh root@svr$i ifconfig enp24s0f0 192.168.1.$i/24
done

copy_to_clients.sh - Copies the test files and OpenSSL to the client servers.
#!/bin/bash
for i in 'seq 21 38'; do
        echo svr$i
        scp ./client.tar.gz root@svr$i:/root
        ssh root@svr$i tar -xf /root/client.tar.gz
done
```

copy_mlnx_tuning_scripts.sh

Copies the Mellanox tuning files to the client and backend servers to allow IRQ pinning.

```
#!/bin/bash
for i in 'seq 21 38'; do
        echo svr$i
        scp ./mlnx_tuning_scripts_package.tar.gz root@svr$i:/root
        ssh root@svr$i tar -xf /root/mlnx_tuning_scripts_package.tar.gz
done
```

tune.sh

Tunes network settings. This script is called in the IRQ pinning scripts.

```
#!/bin/bash
ufw disable
iptables -F
#TCP Memory
echo 16777                > /proc/sys/net/core/rmem_max
echo 16777                > /proc/sys/net/core/wmem_max
echo 16777                > /proc/sys/net/core/rmem_default
echo 16777                > /proc/sys/net/core/wmem_default
echo 16777 16777 16777    > /proc/sys/net/ipv4/tcp_rmem
echo 53875 53875 53875    > /proc/sys/net/ipv4/tcp_wmem
echo 16777                > /proc/sys/net/core/optmem_max
echo 16777 16777  16777   > /proc/sys/net/ipv4/tcp_mem
echo 65536                > /proc/sys/vm/min_free_kbytes
#TCP Behavior
echo 0                    > /proc/sys/net/ipv4/tcp_timestamps
echo 0                    > /proc/sys/net/ipv4/tcp_sack
echo 0                    > /proc/sys/net/ipv4/tcp_fack
echo 0                    > /proc/sys/net/ipv4/tcp_dsack
echo 0                    > /proc/sys/net/ipv4/tcp_moderate_rcvbuf
echo 1                    > /proc/sys/net/ipv4/tcp_rfc1337
echo 600                  > /proc/sys/net/core/netdev_budget
echo 128                  > /proc/sys/net/core/dev_weight
echo 1                    > /proc/sys/net/ipv4/tcp_syncookies
echo 0                    > /proc/sys/net/ipv4/tcp_slow_start_after_idle
echo 1                    > /proc/sys/net/ipv4/tcp_no_metrics_save
echo 1                    > /proc/sys/net/ipv4/tcp_orphan_retries
echo 0                    > /proc/sys/net/ipv4/tcp_fin_timeout
echo 0                    > /proc/sys/net/ipv4/tcp_tw_reuse
echo 0                    > /proc/sys/net/ipv4/tcp_tw_recycle
echo 1                    > /proc/sys/net/ipv4/tcp_syncookies
echo 2                    > /proc/sys/net/ipv4/tcp_synack_retries
echo 2                    > /proc/sys/net/ipv4/tcp_syn_retries
echo cubic                > /proc/sys/net/ipv4/tcp_congestion_control
echo 1                    > /proc/sys/net/ipv4/tcp_low_latency
echo 1                    > /proc/sys/net/ipv4/tcp_window_scaling
echo 1                    > /proc/sys/net/ipv4/tcp_adv_win_scale
#TCP Queueing
echo 0                    > /proc/sys/net/ipv4/tcp_max_tw_buckets
echo 1025 65535           > /proc/sys/net/ipv4/ip_local_port_range
echo 1310720              > /proc/sys/net/core/somaxconn
echo 2621440              > /proc/sys/net/ipv4/tcp_max_orphans
echo 2621440              > /proc/sys/net/core/netdev_max_backlog
echo 2621440              > /proc/sys/net/ipv4/tcp_max_syn_backlog
echo 4000000              > /proc/sys/fs/nr_open

echo 4194304              > /proc/sys/net/ipv4/ipfrag_high_thresh
echo 3145728              > /proc/sys/net/ipv4/ipfrag_low_thresh
echo 30                   > /proc/sys/net/ipv4/ipfrag_time
echo 0                    > /proc/sys/net/ipv4/tcp_abort_on_overflow
echo 1                    > /proc/sys/net/ipv4/tcp_autocorking
echo 31                   > /proc/sys/net/ipv4/tcp_app_win
echo 0                    > /proc/sys/net/ipv4/tcp_mtu_probing
set selinux=disabled
ulimit -n 1000000
```

client_irq_setting.sh

Sets the IRQ pinnings for the client servers and network tunings. Always run after reboot.

```
#!/bin/bash
NIC=enp24s0f0
for i in 'seq 21 35'; do
        echo svr$i
        ssh root@svr$i "cd /root/mlnx_tuning_scripts_package; ./set_irq_affinity_cpulist.sh 0-95
${NIC}"
        ssh root@svr$i "ifaces=${NIC} source /root/client/scripts/tune.sh"
        ssh root@svr$i "ulimit -n 2000000"
done
```

install_nginx.sh

Installs Nginx on the backend servers.

```
#!/bin/bash
for i in 'seq 36 38';do
        ssh root@svr$i "apt-get update -y ; apt-get install -y nginx"
done
```

nginx_irq_setting.sh

Sets the IRQ pinnings for the backend servers and network tunings. Always run after reboot.

```
#!/bin/bash
NIC=enp24s0f0
for i in 'seq 36 38'; do
        echo svr$i
        ssh root@svr$i "cd /root/mlnx_tuning_scripts_package; ./set_irq_affinity_cpulist.sh 48-95
${NIC}"
        ssh root@svr$i "ifaces=${NIC} source /root/client/scripts/tune.sh"
        ssh root@svr$i "ulimit -n 2000000"
done
```

setup_nginx.sh

```
#!/bin/bash
for i in 'seq 36 38';
do
        echo svr$i
       # ssh root@svr$i apt install nginx
        ssh root@svr$i mkdir /var/log/nginx
        ssh root@svr$i cp /etc/nginx/nginx.conf /etc/nginx/nginx.conf.orig
        scp ./nginx/nginx.conf root@svr$i:/etc/nginx/nginx.conf
        scp -r ./nginx/pma root@svr$i:/root
        ssh root@svr$i pkill nginx
        ssh root@svr$i taskset -c 1-20 /usr/sbin/nginx -c /etc/nginx/nginx.conf
done
```

Nginx files for the backend server

index.html

```
<head>
test
</head>
```

nginx.conf

```
user root;
worker_priority 0;
worker_processes 20;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
        worker_connections 20800;
        multi_accept on;
        use epoll;
        accept_mutex on;
}

http {
        sendfile on;
        tcp_nopush on;
        tcp_nodelay on;
        keepalive_timeout 65;
        types_hash_max_size 2048;

        error_log off;
        access_log off;

        include /etc/nginx/mime.types;
        default_type application/octet-stream;

        server
        {
                listen 0.0.0.0:80 ;
                location /
                {
                        root  html ;
                        index /root/pma/index.html ;
                }
        }
}
```

Test scripts

kill_clients.sh

Stops the OpenSSL process on all clients. This script is called in start_clients.sh script.

```
#!/bin/bash
for i in 'seq 21 35'; do
        echo killing svr$i
        ssh root@svr$i pkill -9 openssl &
done
wait
```

connection.sh

Creates the test connections. This script is called in start_clients.sh script.

```
#!/bin/bash

######################################
############# USER INPUT #############
######################################
ip_address=3.3.3.3
_time=$8
```

```
clients=$6
portbase=443
cipher=$4
OPENSSL_DIR=/root/client/source/110h/
######################################
############# USER INPUT #############
######################################

#Check for OpenSSL Directory
if [ ! -d $OPENSSL_DIR ];
then
    printf "\n$OPENSSL_DIR does not exist.\n\n"
    printf "Please modify the OPENSSL_DIR variable in the User Input section!\n\n"
    exit 0
fi

helpAndError () {
    printf "\nThis script is to run the CPS testing HTTPS.\n"
    printf "\nTo use this script: ./download.sh \n"
    printf "\nTo do a dry-run, use the emulation flag:\n"
    printf "./download.sh --emulation\n\n"
    exit 0
}

#Check for h flag or no command line args
if [[ $1 == *"h"* ]]; then
    helpAndError
    exit 0
fi

#Check for emulation flag
if [[ $@ == **emulation** ]]
then
    emulation=1
fi

#cmd1 is the first part of the commandline and cmd2 is the second partrt
#The total commandline will be cmd1 + "192.168.1.1:4400" + cmd2
cmd1="$OPENSSL_DIR/apps/openssl s_time -connect"
cmd2="-new -www /index.html -cipher $cipher -time $_time"

#Print out variables to check
printf "\n Location of OpenSSL:           $OPENSSL_DIR\n"
printf " IP Addresses:                  $ip_address\n"
printf " Time:                          $_time\n"
printf " Clients:                       $clients\n"
printf " Port Base:                     $portbase\n"
printf " Cipher:                        $cipher\n\n"

printf "Press ENTER to continue"

#read

#Remove previous .test files
rm -rf ./.test_*

#Get starttime
starttime=$(date +%s)

#Kick off the tests after checking for emulation
if [[ $emulation -eq 1 ]]
then
    for (( i = 0; i < ${clients}; i++ )); do
        printf "$cmd1 $ip_address:$(($portbase)) $cmd2 > .test_$(($portbase))_$i &\n"
    done
```

```
        exit 0
    else
        for (( i = 0; i < ${clients}; i++ )); do
            $cmd1 $ip_address:$(($portbase)) $cmd2 > .test_$(($portbase))_$i &
            sleep 0.01
        done
    fi

    waitstarttime=$(date +%s)
    # wait until all processes complete
    while [ $(ps -ef | grep "openssl s_time" | wc -l) != 1 ];
    do
        sleep 1
    done

    total=$(cat ./.test_$(($portbase))* | awk '(/^[0-9]* connections in [0-9]* real/){ total += $1/$4 }
    END {print total}')
    echo $total >> .test_sum
    sumTotal=$(cat .test_sum | awk '{total += $1 } END { print total }')
    printf "Connections per second:      $sumTotal CPS\n"
    printf "Finished in %d seconds (%d seconds waiting for procs to start)\n" $(($(date +%s) -
    $starttime)) $(($waitstarttime - $starttime))
    rm -rf ./.test_*
```

start_clients.sh

Controls the start, number of connections, duration, and stop of a test run.

```
    #!/bin/bash
    if [ -z $1 ]
    then
            time=60
    else
            time=$1
    fi

    if [ -z $2 ]
    then
            clients=2000
    else
            clients=$2
    fi

    echo "time: ${time}"
    echo "concurrency for clients: $clients"
    for i in 'seq 21 35'; do
            echo "host: svr$i"
            ssh root@svr$i /root/client/scripts/connection.sh --servers 1 --cipher ECDHE-ECDSA-AES128-
    GCM-SHA256 --clients $clients --time $time &
    done

    sleep $(($time+60))
    ./kill_clients.sh
```

make_tools.sh

Run on VMware NSX Advanced Load Balancer Controller to create scripts for gathering stats.

```
    #!/bin/bash
    echo "show serviceenginegroup Default-Group summary | grep at_curr | awk '{print \$4}'" > /opt/names-
    cmd

    rm -f /opt/se
    /opt/avi/python/bin/avi_shell.sh --address localhost --user admin --password Password1 --file /opt/
```

```
names-cmd | grep 'Avi-se-' >> /opt/se

rm -f /opt/cpu-cmd
for se in 'cat /opt/se'; do
    echo "show serviceengine $se cpu | grep se_dp -A 1 | grep usage" >> /opt/cpu-cmd
done

rm -f /opt/flowtablestat-cmd
for se in 'cat /opt/se'; do
    echo "show serviceengine $se flowtablestat | grep -vw 0 | grep flow_probes" >> /opt/flowtablestat-
cmd
done

rm -f /opt/interface-cmd
for se in 'cat /opt/se'; do
    echo "show serviceengine $se interface | grep -vw 0 | grep error" >> /opt/interface-cmd
done

rm -f /opt/interfaceclear-cmd
for se in 'cat /opt/se'; do
    echo "clear serviceengine $se interface" >> /opt/interfaceclear-cmd
done

rm -f /opt/flowtableclear-cmd
for se in 'cat /opt/se'; do
    echo "clear serviceengine $se flowtablestat" >> /opt/flowtableclear-cmd
done
```

run_test.sh

We created this script as a test harness. It calls the test scripts above, controls test input, and gathers performance states.

```
#!/bin/bash
CLIENTS=1000
WARMUP=120
RUNTIME=600
COOLDOWN=120
TOTAL_TIME=$((WARMUP+RUNTIME+COOLDOWN))
if [ -z $1 ]
then
  echo Set test number!
  exit
else
  TEST_NUM=$1
fi

TEST_NAME=test${TEST_NUM}

. get-se-distribution
. clear-flowtablestat
. clear-interface

. get-cpu | tee ${TEST_NAME}_cpu-prerun.txt
. get-flowtablestat | tee ${TEST_NAME}_flowtablestat-prerun.txt
. get-interface | tee ${TEST_NAME}_interface-prerun.txt
. get-se-distribution | awk '/response_2xx/{print $4}' | sort -n | tee ${TEST_NAME}_se-prerun.txt
ssh root@10.209.100.2 "cd avi-scripts ; ./view_num_stime.sh" | tee ${TEST_NAME}_stime-prerun.txt
ssh root@10.209.100.2 "cd avi-scripts ; ./view_num_nginx.sh" | tee ${TEST_NAME}_nginx-prerun.txt
ssh root@10.209.100.2 "./run_all.sh 'ss -s'" | tee ${TEST_NAME}_ss-prerun.txt
sleep 5

ssh root@10.209.100.2 "cd avi-scripts ; ./start_clients.sh ${TOTAL_TIME} ${CLIENTS}" > ${TEST_NAME}_
output.txt &
sleep ${WARMUP}
```

```
. get-se-distribution | awk '/response_2xx/{print $4}' | sort -n | awk "{sum+=\$1}END{time=${WARMUP};
print \"Total: \",sum; print \"Time: \",time; printf(\"Avg: %.2f\n\",sum/time) }" | tee ${TEST_NAME}_
se-warmup.txt &
. get-cpu | tee ${TEST_NAME}_cpu-warmup.txt &
. get-flowtablestat | tee ${TEST_NAME}_flowtablestat-warmup.txt &
. get-interface | tee ${TEST_NAME}_interface-warmup.txt &
ssh root@10.209.100.2 "cd avi-scripts ; ./view_num_stime.sh" | tee ${TEST_NAME}_stime-warmup.txt &
ssh root@10.209.100.2 "cd avi-scripts ; ./view_num_nginx.sh" | tee ${TEST_NAME}_nginx-warmup.txt &
ssh root@10.209.100.2 "./run_all.sh 'ss -s'" | tee ${TEST_NAME}_ss-warmup.txt &
sleep ${RUNTIME}
. get-se-distribution | awk '/response_2xx/{print $4}' | sort -n | awk "{sum+=\$1}
END{time=${RUNTIME}; print \"Total: \",sum; print \"Time: \",time; printf(\"Avg: %.2f\n\",sum/time)
}" | tee ${TEST_NAME}_se-runtime.txt &
. get-cpu | tee ${TEST_NAME}_cpu-runtime.txt &
. get-flowtablestat | tee ${TEST_NAME}_flowtablestat-runtime.txt &
. get-interface | tee ${TEST_NAME}_interface-runtime.txt &
ssh root@10.209.100.2 "cd avi-scripts ; ./view_num_stime.sh" | tee ${TEST_NAME}_stime-runtime.txt &
ssh root@10.209.100.2 "cd avi-scripts ; ./view_num_nginx.sh" | tee ${TEST_NAME}_nginx-runtime.txt &
ssh root@10.209.100.2 "./run_all.sh 'ss -s'" | tee ${TEST_NAME}_ss-runtime.txt &
sleep ${COOLDOWN}
. get-se-distribution | awk '/response_2xx/{print $4}' | sort -n | awk "{sum+=\$1}
END{time=${COOLDOWN}; print \"Total: \",sum; print \"Time: \",time; printf(\"Avg: %.2f\n\",sum/time)
}" | tee ${TEST_NAME}_se-cooldown.txt &
#. get-cpu | tee ${TEST_NAME}_cpu-cooldown.txt &
#. get-flowtablestat | tee ${TEST_NAME}_flowtablestat-cooldown.txt &
#. get-interface | tee ${TEST_NAME}_interface-cooldown.txt &
ssh root@10.209.100.2 "cd avi-scripts ; ./view_num_stime.sh" | tee ${TEST_NAME}_stime-cooldown.txt &
ssh root@10.209.100.2 "cd avi-scripts ; ./view_num_nginx.sh" | tee ${TEST_NAME}_nginx-cooldown.txt &
ssh root@10.209.100.2 "./run_all.sh 'ss -s'" | tee ${TEST_NAME}_ss-cooldown.txt &
wait
```

This project was commissioned by VMware.

Principled Technologies®

Facts matter.®

Principled Technologies is a registered trademark of Principled Technologies, Inc.
All other product names are the trademarks of their respective owners.

DISCLAIMER OF WARRANTIES; LIMITATION OF LIABILITY:
Principled Technologies, Inc. has made reasonable efforts to ensure the accuracy and validity of its testing, however, Principled Technologies, Inc. specifically disclaims any warranty, expressed or implied, relating to the test results and analysis, their accuracy, completeness or quality, including any implied warranty of fitness for any particular purpose. All persons or entities relying on the results of any testing do so at their own risk, and agree that Principled Technologies, Inc., its employees and its subcontractors shall have no liability whatsoever from any claim of loss or damage on account of any alleged error or defect in any testing procedure or result.

In no event shall Principled Technologies, Inc. be liable for indirect, special, incidental, or consequential damages in connection with its testing, even if advised of the possibility of such damages. In no event shall Principled Technologies, Inc.'s liability, including for direct damages, exceed the amounts paid in connection with Principled Technologies, Inc.'s testing. Customer's sole and exclusive remedies are as set forth herein.