White Paper

Communications Service Providers
Network Automation

FRINX Tests Performance of Model-Driven Network Automation

Tests using Intel® Xeon® Scalable processor-based servers show FRINX UniConfig can scale to 50,000 CPE devices or 2,000 service provider routers with all updates to the config data store performed in 29 msec or less in 95% of the tests[1]

Table of Contents

Network automation for communications service providers (CoSP) has evolved to keep pace with the complexity of network designs. Many networks support service delivery using thousands of routers, switches, customer premises equipment (CPE) devices, network interface devices (NIDs), firewalls, and other systems. Configuring and reconfiguring all of these devices in support of a new service, protocol change, or other update is daunting.

Today's network configuration automation technology counts on device models and service models using languages such as YANG and JSON to automatically translate the network administrator's network design into configurations for the affected network devices. Even with these models, there is complexity as some network devices support only one of the proliferation of network modeling languages, and some older products don't support any models—relying instead on a command line interface (CLI) for configuration.

FRINX offers its FRINX Machine, a new generation of network configuration tool that adds an abstraction layer that allows for communications with the network devices using any supported modeling language or CLI that is native to the network device. FRINX Machine also supports service model development that simplifies a new service deployment by translating these requirements into device models based on YANG and CLI commands that are pushed to the devices.

FRINX Machine is based on open source components and consists of the following products: UniConfig for network control, UniFlow for creating and operating workflows, and UniResource for managing an inventory of physical and logical assets and resources.

The performance of the solution lies with UniConfig, the network control element, which communicates with the network devices. To demonstrate its control performance, FRINX, an Intel® Network Builders ecosystem partner, tested its UniConfig software in the Intel Network Builders lab using Intel® Xeon® Scalable processors to determine how responsive is the controller.

FRINX UniConfig Network Controller

The UniConfig software consists of three layers that network managers can access individually or via the UniConfig node manager API.

The southbound layer provides connectivity to a wide range of network devices using NETCONF or a command line interface (CLI) via Telnet and secure shell (SSH). This layer provides transparent access to CLI devices, and it includes an open source device library (translation units) that maps data in OpenConfig format to vendor-specific CLI implementations and vice versa. OpenConfig is an open source API for network telemetry and automation.[2]

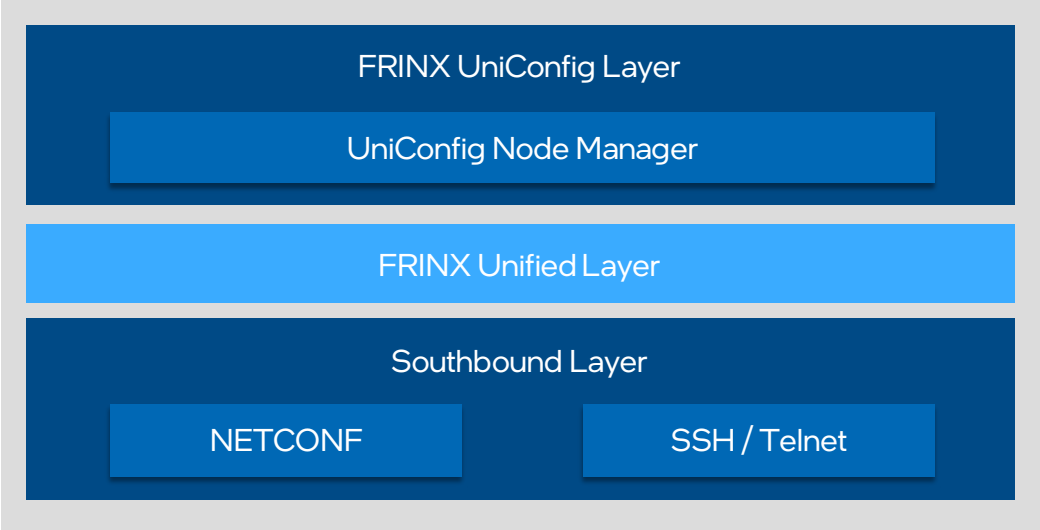

Figure 1. UniConfig's three layers

The unified layer combines all the devices regardless of how they were mounted (e.g., NETCONF and CLI). These devices are then accessible under a unified mount point to the UniConfig layer. This provides a layer of abstraction between the southbound protocols and the user intent (top layer) that is to be applied to the network. The unified layer also provides native YANG models as well as OpenConfig YANG models to vendor-specific YANG translation capabilities.

The UniConfig layer enables reading and writing of YANG-based configurations to and from devices. It also adds the capability to create configuration snapshots that can be committed and can be rolled back by the system in the event of a failure. The UniConfig layer can compare network intent from profiles located in the configuration data store. The software analyzes the differences between intended state and actual state from data located in the operational data store. It then applies the new state to the devices connected through lower layers via atomic operations (commit). This functionality saves resources and enables very high transaction throughput by sending only the changed configuration elements from the most recent operation and not the complete configuration.

The UniConfig layer can also build snapshots of all or a subset of devices and move them from the current configuration to any snapshot in a single transaction. Finally, the UniConfig layer includes the "dry-run manager" that allows testing of NETCONF and CLI configuration changes before they are applied to the network.

The functionality of the UniConfig layer is accessible via a REST interface and client libraries. Those client libraries make the UniConfig API available through popular programming languages and allow users to build applications using the UniConfig functionality without having to interact with the REST API directly.

UniConfig Test Setup

To determine performance of the controller, one server was used as the UniConfig device under test (DUT) and two other servers were used to generate packets and provide test analytics and reports (see Table 1 and Figure 2). Three scenarios were tested: one with 20,000 mounted CPE devices, one with 50,000 mounted CPE devices, and one with 2,000 complex service provider routers. The smaller CPE devices required less configuration by the controller (about 1,000 lines of JSON configuration) whereas the service provider routers had much more complex configurations (about 600,000 lines of JSON configuration).

Performance is also impacted by the number of applications that are seeking to concurrently configure the devices on the network. The tests were configured for both five threads and 10 threads, two levels that are common in real world networks.

NODE NAME	RAM	CORES	CPU	SOFTWARE
zs25	376 GB	80 cores	Intel® Xeon® Gold 6230N CPU @ 2.30 GHz	UniConfig
zs24	187 GB	72 cores	Intel® Xeon® Gold 6140 CPU @ 2.30 GHz	Netconf Test Tool 1 (TT1); jmeter scripts
zb19	251 GB	88 cores	Intel® Xeon® CPU E5-2699 v4 @ 2.20 GHz	Netconf Test Tool 2 (TT2)

Table 1. Hardware setup: DUT (Node zs25) and additional servers used to generate packets and provide test analytics and reports

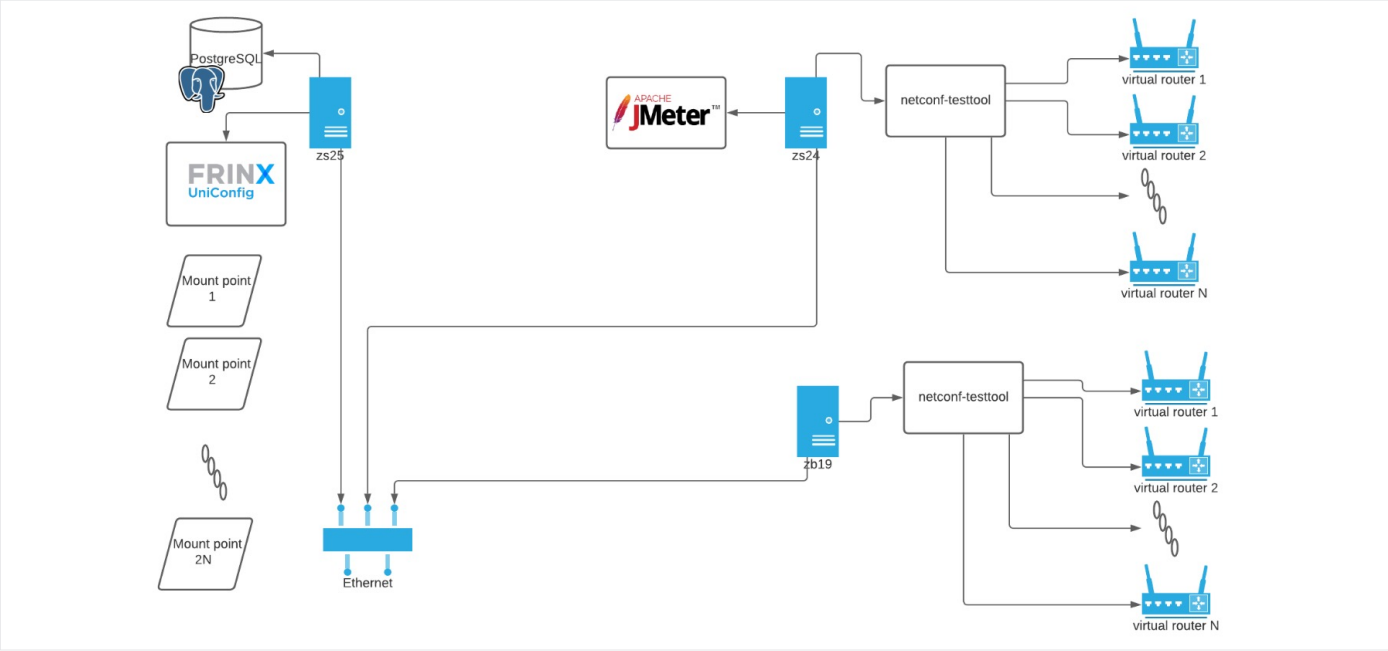

Figure 2. Lab test topology

Three test cases were designed to explore the scale and performance behavior of a single instance of the UniConfig network controller. Key test case parameters included the following:

- **Number of mounted devices:** 20,000 CPE connections were emulated in two test cases; 50,000 CPE connections were emulated in two of the test cases and 2,000 service provider router connections were emulated in two of the test cases.
- **Number of JSON lines:** This is an indication of the complexity of the configuration; 1,000 configuration lines were used in the four CPE use cases and 600,000 lines of configuration were used in the two service provider use cases.
- **Concurrent application threads:** These are top-down application requests/threads to the controller and were either 5 or 10 concurrent requests.

Table 2 summarizes the different test cases.

Test cases A1 and A2 were designed to test behavior with 20,000 devices under control with a small sized configuration each. This was designed to test the behavior of the controller for customer premises equipment (CPE) configurations where typically thousands or tens of thousands of devices with discrete configurations have to be managed and reconfigured efficiently. Two test case variations (A1 and A2) are differentiated by changing the number of concurrent application threads that are requesting configuration changes on the controller from 5 to 10.

Test cases B1 and B2 were designed to demonstrate the impact of scaling up the number of devices under control to 50,000 devices while the number of configured devices stays the same as the 20,000 devices configured in tests A1 and A2. The difference is that 50,000 devices are mounted on the controller and 20,000 of those are being configured by the test script. The B1 and B2 test variations reflect a change in the number of concurrent application threads that are requesting configuration changes on the controller from 5 to 10.

TEST CASE NUMBER	TEST CASE PARAMETERS
Test A1	20,000 devices mounted; 20,000 devices configured with 1,000 lines of config each – 5 concurrent application threads
Test A2	20,000 devices mounted; 20,000 devices configured with 1,000 lines of config each – 10 concurrent application threads
Test B1	50,000 devices mounted; 20,000 devices configured with 1,000 lines of config each – 5 concurrent application threads
Test B2	50,000 devices mounted; 20,000 devices configured with 1,000 lines of config each – 10 concurrent application threads
Test C1	2,000 devices mounted, 2,000 configured with 600,000 lines of config each – 5 concurrent application threads
Test C2	2,000 devices mounted, 2,000 configured with 600,000 lines of config each – 10 concurrent application threads

Table 2. Test case number and description

Test cases C1 and C2 were designed to test the controller scale and performance when connecting to 2,000 service provider routers with very large configurations. The configurations were built based on actual routers in use in a production network. The configuration file size was 200,000 lines of CLI commands, which is represented by 600,000 JSON-based configuration lines in UniConfig. Similar to the previous test cases, the number of concurrent application threads that are requesting information is either 5 or 10.

Test Results: Application Response Time

Table 3 summarizes the in-depth test results for all six test cases.

TEST CASE	A1	A2	B1	B2	C1	C2
Devices mounted on controller	20,000	20,000	50,000	50,000	2,000	2,000
Lines of JSON config per device	1,000	1,000	1,000	1,000	600,000	600,000
Devices configured	20,000	20,000	19,995	19,990	1,995	1,990
Threads	5	10	5	10	5	10
Total requests	160,000	160,000	159,960	159,920	15,960	15,920
Duration	00:31:08	00:31:02	00:30:29	00:30:17	02:56:51	02:20:45
Errors	0	0	0	0	0	0
Requests per second	85.6	85.9	87.5	88.0	1.5	1.9
PUT if [ms] (95th percentile/max)	19/220	19/207	19/231	20/212	22/345	23/382
PUT vrf [ms] (95th percentile/max)	19/200	19/246	19/199	19/197	22/345	22/80
PUT bgp [ms] (95th percentile/max)	20/200	20/268	20/198	21/198	22/440	23/402
POST: RPC calculate-diff (PUT) [ms] (95th percentile/max)	301/475	867/1,248	283/465	827/1,104	15,963/21,939	24,693/25,592
POST: RPC commit (PUT) [ms] (95th percentile/max)	217/842	509/1,000	214/453	500/789	16,276/22,703	25,153/26,102
GET if present [ms] (95th percentile/max)	24/171	24/245	24/211	25/288	28/98	29/88
GET vrf present [ms] (95th percentile/max)	24/161	23/268	23/195	24/293	26/1,075	27/366
GET bgp present [ms] (95th percentile/max)	24/148	24/167	24/209	24/202	27/85	28/343
UC Max used cores [% CPU]	1,048	878	800	800	1,654	2,226
UC used heap before [GB]	54	54	127	127	162	162
UC used heap after [GB]	56	56	138	138	269	269

Table 3. Test results by test case[1]

Test Results with 20,000 CPE Devices (A1 & A2)

The key takeaway from the A1 test is that the 95th percentile of all updates to the config data store (the combination of all GET and PUT operations—see results in Table 4) are performed in less than 24 msec, with the average being 19.33 msec and the maximum response time 220 msec. The most critical value for UniConfig RPCs is the performance of the commit operation (commit RPC). The 95th percentile of all commit operations in test A1 were observed to be finished in under 217 msec and the maximum observed time was 842 msec (see Table 4) out of 160,000 requests.

TEST CASE	A1	A2
Devices mounted on controller	20,000	20,000
Lines of JSON config per device	1,000	1,000
Devices configured	20,000	20,000
Threads	5	10
Average PUT operations 95th percentile (msec)	19.33	19.33
Average GET operations 95th percentile (msec)	24	23.67
Average Commit RPC 95th percentile (msec)	217	509

Table 4. Test results for 20,000-device configurations (tests A1 & A2)[1]

Test A2 added five additional threads with minor changes to the performance (see Table 4). The 95th percentile of all updates to the config data store (all GET and PUT operations) are performed in less than 24 msec with an average of 19.33 msec and the maximum response time being 268 msec. The most critical value for UniConfig RPCs is the performance of the commit operation. The 95th percentile of all commit operations in test A2 were observed to be finished in under 509 msec and the maximum observed time was 1,000 msec out of 160,000 requests.

Test Results with 50,000 CPE Devices (B1 & B2)

The key takeaway from the test B1 (see Table 5) is that the 95th percentile of all updates to the config data store (all GET and PUT operations) are performed in less than 24 msec with an average of 19.33 msec and a maximum response time of 231 msec. The most critical value for UniConfig RPCs is the performance of the commit operation. The 95th percentile of all commit operations in test B1 were observed to be finished in under 214 msec and the maximum observed time was 453 msec out of 160,000 requests.

The key takeaway from the test B2 (see Table 5) is that the 95th percentile of all updates to the config data store (all GET and PUT operations) are performed in less than 25 msec with an average performance of 24.33 msec and the maximum response time being 293 msec. The most critical value for UniConfig RPCs is the performance of the commit RPC operation. The 95th percentile of all commit operations in test B2 were observed to be finished in under 500 msec and the maximum observed time was 789 msec out of 160,000 requests.

TEST CASE	B1	B2
Devices mounted on controller	50,000	50,000
Lines of JSON config per device	1,000	1,000
Devices configured	20,000	20,000
Threads	5	10
Average PUT operations 95th percentile (msec)	19.33	20
Average GET operations 95th percentile (msec)	23.67	24.33
Average Commit RPC 95th percentile (msec)	217	500

Table 5. Test results for 50,000-device configurations (tests B1 & B2)[1]

Test Results with 2,000 Service Provider Routers (C1 & C2)

The key takeaway from the test C1 (see Table 6) is that the 95th percentile of all updates to the config data store (all GET and PUT operations) are performed in less than 28 msec with an average performance of less than 23 msec and the maximum response time being 1,075 msec. The most critical value for UniConfig RPCs is the performance of the commit operation. The 95th percentile of all commit operations in test C1 were observed to be finished in under 16,276 msec and the maximum observed time was 22,703 msec (see Table 6) out of 16,000 requests.

TEST CASE	C1	C2
Devices mounted on controller	2,000	2,000
Lines of JSON config per device	600,000	600,000
Devices configured	2,000	2,000
Threads	5	10
Average PUT operations 95th percentile (msec)	22.00	22.67
Average GET operations 95th percentile (msec)	27.00	28.00
Average Commit RPC 95th percentile (msec)	16,276	25,153

Table 6. Test results for 2,000-device configurations (tests C1& C2)[1]

The key takeaway from the test C2 is that the 95th percentile of all updates to the config data store (all GET and PUT operations) are performed in less than 29 msec with the maximum response time being 402 msec. The most critical value for UniConfig RPCs is the performance of the commit operation. The 95th percentile of all commit operations in test C1 were observed to be finished in under 25,153 msec and the maximum observed time was 26,102 msec out of 16,000 requests.

Persistence and High Availability

All tests were performed with the UniConfig in-memory data store. UniConfig also provides an option to store device and configuration data in an external database (PostgreSQL). The stored data is used to provide high availability in cases where an instance of the controller goes down and another instance takes over without the need to perform a full reconciliation from the network.

The tests showed an impact of external persistence from the PostgreSQL. This impact was less than 5 percent of additional median response time for small configurations (tests A and B) and less than 40 percent of additional median response time for large configurations (tests C) with up to 50 concurrent threads.

Conclusion

Remote and edge networks are a big part of the future of CoSP service delivery plans, which makes automating the updating of network configurations an essential tool for cost-effective server operation. The time and complexity of maintaining and updating these networks is growing as more uCPEs are deployed and more backbone routers are needed to aggregate that traffic.

As shown in these test results, FRINX UniConfig is fast at upgrading tens of thousands of low-complexity devices and thousands of highly complex systems. CPE systems take only milliseconds to update and a whole network of 20,000 devices can be updated in just over a half hour. For routers with 600 times the code complexity, updating a complex service on a router takes place in about 4.2 seconds or less. Whether its CPE or backbone routers or any other network equipment, UniConfig, running on Intel Xeon Scalable processor-based servers, can provide CoSPs with a significant speed advantage in updating and maintaining their networks.[1]

Learn More

FRINX

Intel® Xeon® Scalable processors

Intel® Network Builders

Notices & Disclaimers

[1] Testing done by FRINX between November 2020 and February 2021. The zs25 server used four Intel® Xeon® Gold 6230N processors (microcode: 0x5002f01) with 20 cores each, operating at 2.3 GHz. The server featured 376 GB of RAM. Intel® Hyper-Threading Technology was enabled, as was Intel® Turbo Boost Technology 2.0. BIOS version was SE5C620.8 6B.02.01.0012.070720200218. Intel® Ethernet Network Adapter X722 (10GBASE-T) provided network access. The operating system was Ubuntu Linux release 20.04.1 LTS with kernel 5.4.0-42-generic. Postgre SQL database v12.5 was used. Compiler GCC was version 9.3.0. The workload was UniConfig v.4.2.5

[2] https://www.openconfig.net

The zs24 server utilized four Intel Xeon Gold 6140 processors (microcode: 0x2006906) each with 18 cores operating at 2.3 GHz. The server featured 187 GB of RAM. Intel Hyper-Threading Technology was enabled, as was Intel Turbo Boost Technology 2.0. BIOS version was SE5C620.86B.02.01.0012.070720200218. Intel Ethernet Network Adapter X722 (10GBASE-T) provided network access. The operating system was Ubuntu Linux version 20.04.1 LTS with kernel 5.4.0-42-generic. Compiler GCC was version 9.3.0. Apache jMeter 5.3 was used for load testing. The workload was Netconf-testtool v.1.4.2.

The zb19 server utilized four Intel Xeon processor E-5-2699 v4 (microcode: 0xb000038) each with 22 cores operating at 2.2 GHz. The server featured 251 GB of RAM. Intel Hyper-Threading Technology was enabled, as was Intel Turbo Boost Technology 2.0. BIOS version was SE5C610.86B.01.01.0029.052820200607. Intel Ethernet Network Adapter X722 (10GBASE-T) provided network access. The operating system was Ubuntu Linux release 20.04.1 LTS with kernel 5.4.0-42-generic. Compiler GCC was version 9.3.0. The workload was Netconf-testtool v.1.4.2.

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

0521/DO/H09/PDF ♻ Please Recycle 347007-001US