ONOS™ PROJECT WHITE PAPER

SDN CONTROL PLANE PERFORMANCE

Raising the bar on SDN performance, scalability, and high availability

This white paper, intended for a technical audience, describes the challenges of architecting a carrier-grade SDN control plane, provides an effective set of metrics to evaluate its “carrier-grade quotient”, and provides a performance evaluation of the ONOS™ Kingfisher release using these metrics.

July 2017

The promise of SDN

Software-Defined Networking (SDN) has become one of the hottest topics in the industry, and for a good reason. SDN is about separating the control plane from the data plane. As a result, SDN disaggregates proprietary closed boxes into the SDN control plane (SDN Network Operating System), the data plane and the applications and services. The SDN control plane behaves like the brain and uses appropriate abstractions, APIs and protocols to support a diversity of applications to control, configure, and manage the network.

SDN gives users centralized control, greater visibility, and greater choice because they can mix and match technologies at different layers. It enables greater innovation because each layer can evolve at its own pace. The end user result is new services and lower costs, which is the real promise of SDN.

Lack of performance - a significant barrier to SDN adoption

The technical and business benefits of SDN are well understood. Google, Facebook, Microsoft and others have successfully demonstrated the benefits of deploying SDN in both WAN and data center networks.

In order to deploy SDN in Service Provider networks, the SDN control plane[1] needs to have high performance, scale-out design and high availability. Building a “carrier-grade” SDN control plane that supports these requirements is a challenging design problem that requires thoughtful technical analysis of the tradeoffs between high availability, performance and scale as all three are closely related. In fact, the lack of a high performance SDN control plane platform has been a big barrier to SDN deployment and adoption.

Moreover, simplistic performance metrics such as “Cbench” do not provide a complete or accurate view of the performance and scale-out capabilities. A better set of measurements is required to determine what could be called the "carrier grade quotient" and it is the focus of this paper. Addressing the scale and performance in an accepted, analytical manner is critical to driving SDN adoption in real networks.

[1] In this paper, SDN Network OS and SDN Control Plane are used interchangeably.

SDN control plane metrics that matter

Performance

Consider for a moment that the SDN network operating system is a black box. Among its key performance metrics would certainly be the following:

- **Topology change latency:** This measures how quickly the operating system can respond to different types of topology events, such as link up/down, port up/down, or switch add/remove. The latency measures how quickly the state is updated in the global network view across the entire cluster, which includes the topology event which notifies applications about the change.
- **Flow setup throughput:** This measures the number of flows that can be set up by the operating system in response to application requests or to network events.
- **Northbound latency:** This measures how quickly the operating system is able to satisfy an application request and how quickly it can react to network events.
- **Northbound throughput:** This measures the ability of the operating system to handle an increasing number of application requests, and the maximum load supported.

The first step then is to define initial targets that will meet or exceed the bar for Service Provider networks. Working with service providers has led us to the following targets as a starting point.

- 1 Million flow setups/sec
- Less than 100ms latency for both topology change events and application requests (ideally, ~10 ms or lower)

Higher throughput and lower latency are always desirable — they enable new class of applications and capabilities. But we have to have a starting point that gives performance similar to existing distributed control planes.

Scalability

One of the benefits of SDN is the ability to aggregate all network state in a centralized location and enable service providers to instantiate a variety of new services which can use this state. Service providers will want to collect more and more real time network state and run more and more applications that consume this state. This in turn means that service providers should be able to scale the SDN control plane capacity to increase throughput while keeping the latency low and while maintaining an accurate global network view. Ideally, to scale an SDN OS, one should be able to simply add more compute resources (CPU and memory) to a cluster. This ability to incrementally add more control plane capacity is critical because it enables service providers to future-proof their networks and scale effectively to handle growth.

The scalability of an SDN OS is therefore defined as the ability to seamlessly add control plane capacity (servers in a cluster) to increase the SDN OS throughput and either keep the latency low or reduce it further while maintaining a logically centralized view of the network.

The question then is – how should the scalability of an SDN OS be measured? There are two key metrics:

- Increase in throughput metrics (defined above) with addition of servers.
- Unchanged or reduced latency (defined above) with addition of servers.

High Availability

High availability is a prerequisite for SDN adoption in Service Provider networks. While it is great that one can scale a SDN control plane by introducing additional servers, it still needs to be complemented by the control plane's ability to automatically handle failures of individual servers. One should also be able to perform software and hardware upgrades without impacting overall system operation. While the current release does not support in-service-upgrade, this functionality is in the plan for the upcoming release of ONOS.

One way to assess the resiliency of a distributed SDN control plane is to introduce failures and observe how the system responds. In a system designed for high availability operation, one should expect the following in the event of failures (outside the scope of this white paper, detailed results are available on the ONOS wiki):

- Redundancy measures automatically take effect and the system continues to operate with zero downtime.
- The performance of the system remain nominal and is proportional to the number of resources currently at its disposal.
- When the system heals and the instances rejoin, the system automatically rebalances workload to take advantage of the restored capacity.

Challenges in building a "carrier-grade" SDN control plane

A key challenge in building any distributed system is to ensure the individual instances function as a single logical entity. The complex details of how state is managed and coordination is achieved should be hidden from the applications by preserving simple, easy to understand abstractions. An important abstraction ONOS introduces is the Global Network View. The ability to operate on an accurate picture of the entire network greatly simplifies network control functions. For example, a use case that would have traditionally required the introduction of new distributed routing protocol such as OSPF can be solved by employing a simple shortest path algorithm on the network graph.

Why simple approaches don't work

This section explores the key challenges involved in maintaining a consistent and accurate picture of the network in a distributed SDN control plane. Arguably these problems do not exist in a single instance controller.

A single instance controller has direct visibility over the entire network and is capable of keeping this view up-to-date as the network evolves. Furthermore, it can present this view to any application with little overhead as the state can be maintained locally. But this simplicity comes at the expense of availability and scale. A controller outage will immediately render all network control impossible. The availability problem can be mitigated to an extent by running two controllers with one in hot-standby mode where this peer controller takes over control in the event the primary controller fails. However, each controller instance on its own must still be capable of managing the entire network and serving all the applications that are trying to monitor and program the network. As the size of SDN controlled networks grow and as more and more network control applications are introduced, these solutions fall short and new ones that truly scale are needed.

One way to architect a scalable SDN control plane is to run multiple controller instances with each instance responsible for controlling a subset of the network. By dividing the network amongst the set of available controllers, this approach enables managing a larger network by simply adding new controller instances. Furthermore, if a controller instance fails, responsibility for its portion of the network is automatically transferred to other running instances thereby ensuring uninterrupted network control.

However, this straightforward solution introduces new challenges, and most of these challenges revolve around how to effectively manage state. In such a distributed setting, each controller instance has direct visibility over a slice of the network. In order to preserve the Global Network View abstraction, the control plane has to somehow piece together an accurate picture of the network from these individual slices and present that to applications with minimal overhead. Furthermore, as the underlying network evolves, the control plane has to update the network view in order to maintain its consistency with the underlying network state.

In addition to Global Network View state, there is a variety of control plane state that needs to be tracked and made accessible to network control applications. Examples of this state include application intents, flows, flow stats, resource allocations, network policy information, switch-to-controller mappings and a host of other network control application-specific state. Each of these state come with their own unique consistency semantics, read-write access patterns, locality constraints and durability expectations. A well-architected SDN control plane implementation must be capable of meeting these unique state management needs without sacrificing performance and scalability.

A common approach often used in distributed SDN controllers is to offload all state management to a centralized data store. The data store itself is replicated for availability. While this approach seems reasonable, it suffers from a number of problems as is evident from the shortcomings unearthed during the research and prototyping efforts for ONOS.

Consistency or availability

First, distributed data stores are either architected for high availability or strong consistency. In the event of a network partition, they are forced to either choose strong consistency and risk being unavailable for updates, or choose availability and deal with the possibility of the system state diverging. This fundamental tradeoff (see CAP theorem) is unavoidable in any distributed state management system and the choice a data store makes has a significant impact on the overall system behavior and performance. As described earlier, there is a need to manage different types of state in the SDN control plane and not all of them require the same degree of consistency or availability. Trying to fit all this diverse state into a single data store that is either configured to provide high availability or strong consistency results in a system that has less than ideal performance profile or even worse, a system that behaves incorrectly.

Performance

Another challenge with centralized data store is that very little of the state is local to the controller instance. For state such as Global Network View, which is highly read intensive, having to fetch data from the centralized data store every single access can severely limit performance. Introduction of a cache does not really address the issue because caching introduces the problem of having to worry about maintaining coherence between the local cache and the data store.

Scaling challenges

Building a distributed data store that offers strong consistency while having the ability to scale-out is a non-trivial task. Several off-the-shelf open-source solutions in this space trade consistency for availability (e.g. Cassandra) and the ones that do offer strong consistency are not designed to scale out without negatively impacting throughput (e.g. Zookeeper).

Ultimately building a high performance SDN control plane that seamlessly scales and is highly available requires building and integrating state management solutions that are tailored to meet the unique requirements of each type of state in the control plane.

ONOS approach to performance, scalability and availability

ONOS is built from the ground up as a scalable distributed system that can seamlessly meet the demands for high performance and availability placed on a SDN control plane. ONOS achieves this by providing a host of simple and robust distributed state management primitives.

ONOS takes into consideration the unique properties of each type of control plane state and maps it to a solution that provides the best semantics and performance for that state.

ONOS employs a cluster of controller instances that work together to manage the network. As the demands on the SDN control plane grow, either due to an increase in the size of the network or due to an increase in the number of network control applications, it can scale by adding commodity servers to the controller cluster. ONOS automatically offloads a portion of the work to these new instances. From an architectural standpoint, there are no fundamental limits on how large an ONOS cluster can be, and it can seamlessly scale to support a rapidly growing network.

High availability is of prime importance to the ONOS architecture. Every critical control plane functionality is designed to automatically fail over with zero down-time in the event of an instance failure. The failover semantics are built into software and require no operator intervention. Furthermore, applications that are built to run on ONOS can leverage the same availability building blocks used in rest of the system to provide an always-on experience.

Instead of offloading all state management to a centralized data store, ONOS employs a collection of state management building blocks and matches these appropriately to control plane state. This approach towards distributed state management truly differentiates ONOS from other distributed SDN controllers.

Eventual consistency

Given the read intensive nature of the Global Network View (GNV), achieving high performance requires an approach that can provide access to this view with minimal overhead. However, simply caching this state locally introduces consistency concerns stemming from cached state being out of sync with network state.

ONOS addresses this particular problem using a novel approach. It treats the Global Network View as a state machine to which network events are applied in an order-aware manner. In order to provide low latency access, ONOS maintains a copy of the GNV state machine in memory on every controller instance. It solves the consistency problem using a logical clock to timestamp network events as soon as they are detected on the data plane and then fully replicates them across the control plane. Each instance evolves its local GNV state machine copy independently by using the logical timestamps to detect and discard out of order network events. A periodic, lightweight background task detects and updates GNV copies that are out of sync with each other and with the state of physical network state through a process known as "anti-entropy".

ONOS evaluations have shown that this simple approach described above works well for state that is eventually consistent. The GNV is an example of state that is eventually consistent and by employing this approach, ONOS can detect and respond to network events with very low latency. There are additional areas where this approach works well (including situations where the provenance of the data is outside of the controller, or where state is partitioned and a single instance acts on the data at a time). ONOS employs the same eventually consistent data store abstraction to manage all Intent-related information.

Strong consistency

While eventual consistency works very well for GNV and Intent-related state, it is not suitable when stronger consistency is mandated by a use case. Consider the example of switch-to-controller mapping. At any given point in time, ONOS guarantee that there exists a single controller that acts as the *master* for a given switch. This piece of information needs to be maintained in a strongly consistent manner with every instance agreeing on who the current master for a switch is in order to avoid the devices being programmed concurrently via multiple controllers. An eventually consistent solution is ill-suited for this particular problem for obvious reasons. Another area where strong consistency is important is for control plane state that tracks the network resource allocations, such as “Application X has 10G bandwidth allocated on link L1”. It is important that ONOS provide transactional semantics for such updates and an eventually consistent solution does not work in this situation.

Scaling is a problem often encountered when building a strongly consistent data store. The throughput of a strongly consistent data store is usually inversely correlated to cluster size due to increased coordination overhead. ONOS provides an elegant solution to this problem by sharding the larger data store into smaller units. Each unit, or shard, is given responsibility for a partition of the key space. The ownership for each shard resides with three controllers for high availability. Updates to a given shard are coordinated in a strongly consistent manner by employing a Replicated State Machine (RSM). The integrity of the shard RSMs is maintained via the RAFT consensus algorithm. With this approach, scaling merely involves adding new servers and then instructing ONOS to rebalance the shards to take advantage of the newly added instances.

State management building blocks for application developers

A unique feature in ONOS is that the core state management building blocks are exposed as simple abstractions for applications to build on top of. Application developers do not have to worry about all the complex details of state distribution and can instead focus on the core business logic they want to express.

Evaluation of ONOS performance, scale, and high availability

The following sections summarize the results of the four experiments[2] that measure and quantify the performance of ONOS subsystems.

These experiments are:

1. Latency of topology discovery
2. Flow subsystem throughput
3. Latency of intent operations
4. Throughput of intent operations

[2] In all experiments, the setup uses physical servers. Each server instance has a Dual Xeon E5-2670 v2 2.5GHz processor with 64GB DDR3 and 512GB SSD. Each server uses a 1Gb NIC for the network connection. The instances are connected to each other through a single switch.

By comparison with the experiment results published for ONOS Blackbird release, performance of ONOS Kingfisher release is further improved in terms of reduced topology discovery latency and increased intent operations throughput. Please refer to following subsections for more details.

Detailed information about the evaluation setups and test methodologies are available at https://wiki.onosproject.org/x/n4Q0 and evaluation results are available at https://wiki.onosproject.org/x/xIbV.

Latency of topology discovery

Scenario

Network events need to be handled with low latency to minimize the time that applications are operating on an outdated view of the topology. The main topology events that are important to measure are: adding/removing a switch, discovery of a port changing state along with the subsequent discovery or disappearance of infrastructure links, and discovery of hosts. It is important to quickly recognize that such events have occurred, and to swiftly update the network graph across the cluster so that all instances converge to the same view of the topology.

Goal

The goal of this experiment is to measure the latency of discovering and handling different network events and observe the effect of distributed maintenance of the topology state on latency. The system should be designed so that for any negative event, such as port/link/device-down, the latency is under 10 ms and the latency is unchanged regardless of the cluster size. Positive events, such as port/link/device-up, should be handled within 50 ms. It is more important to react more quickly to negative than to positive events, since the negative ones may affect existing flows.

Experiment setup

This experiment uses two OVS switches controlled by an ONOS instance. Port-up, port-down and switch-up events are triggered on one of the switches and the latency is measured as the time between the switch notification about the port event and the time the last ONOS instance has issued the topology update event. Using at least two switches is required in order to form an infrastructure link whose detection or disappearance is used to trigger the topology update. Since port up/down events trigger subsequent discovery or disappearance of links, we measure both port and link discovery latency in the same test we call link-up/down test, and we measure switch discovery latency in another test we call switch-up/down test. The detailed description of the tests to measure switch/port/link discovery latency can be found at https://wiki.onosproject.org/x/oYQ0.

To measure host discovery latency, a single OVS switch connecting two hosts is used and the latency is measured as the time between the first packet-in message triggered by one of the hosts and the time the last ONOS instance has issued the host added event. The detailed description of the host discovery tests can be found at https://wiki.onosproject.org/x/_Ymq.

Results

The link up/down test results are divided into link-up test result and link-down test result.

Link up test result:

- For a single instance, the latency is around 7 ms.
- For multi-node cluster, the latency is around 16ms.
- By comparison with Blackbird release, single instance latency stays the same and multi instance latency is reduced by ~25%.

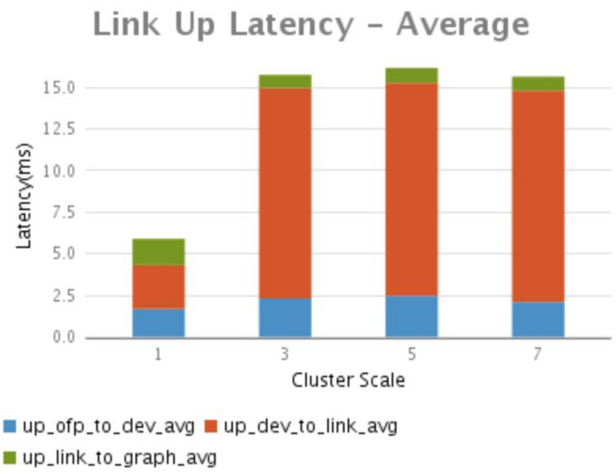

Figure 1

Each latency is further divided into three portions:

- up_ofp_to_dev_avg: time spent for ONOS to generate Device Event triggered by OF Port Status message
- up_dev_to_link_avg: time spent for ONOS to finish link discovery and generate Link Event
- up_link_to_graph_avg: time spent for ONOS to generate Graph Event

The initial increase in latency from single to multi-instance setups is expected. The extra time is spent in the link discovery stage and it is because an eventual consistency model is used by the link store. The increase doesn't happen for device or graph events because, as for the former, device events are fully replicated among the controller instances as soon as they are discovered in the data plane; and for the latter, topology store simply relies on the distributed versions of the device and link stores which means graph events are not delayed by the eventual consistency model but locally generated.

Link down test result:

- For all single and multi-instance setups, latency ranges between 3 - 5 ms.
- The latency numbers stay the same by comparison with Blackbird release.

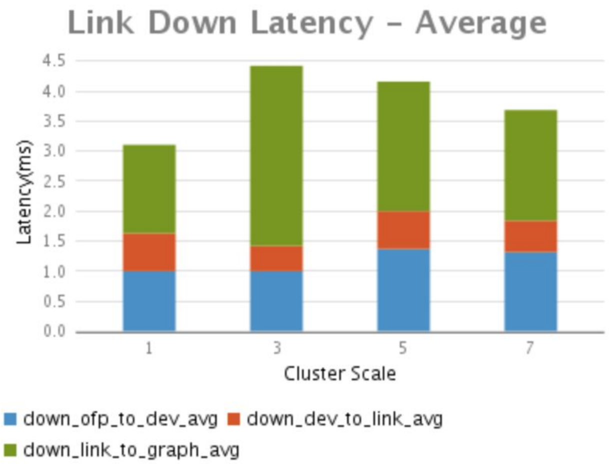

Figure 2

Each latency is further divided into three portions:

- down_ofp_to_dev_avg: time spent for ONOS to generate Device Event triggered by OF Port Status message
- down_dev_to_link_avg: time spent for ONOS to generate Link Event
- down_link_to_graph_avg: time spent for ONOS to generate Graph Event

The latency increase from single to multi-instance setup is as expected since adding instances introduces additional, though small, event propagation delays.

Due to the inherent nature of link detection (via LLDP and BDDP packet injection), the underlying port-up events result in more time-intensive operations required to detect link presence than do port-down events, which trigger an immediate link teardown and the resulting link-down event.

The switch-up/down test results are also divided into switch-up test result and switch-down test result.

Switch up test result:

- For all single- and multi-instance setups, latency ranges between 45 and 55 ms, which is reduced by ~30% by comparison with Blackbird release.

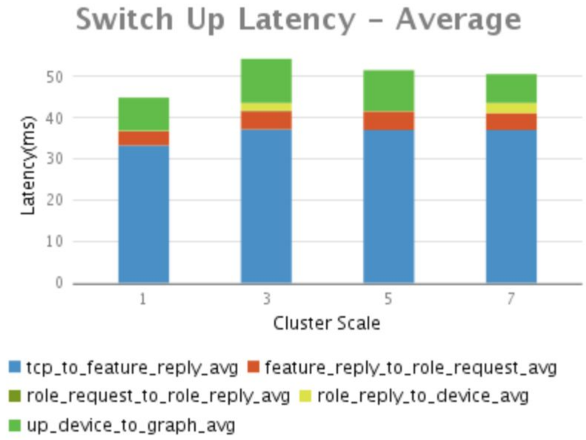

Figure 3

Each latency is further divided into five portions:

- tcp_to_feature_reply_avg: time spent from devices connection initiation to ONOS sending OF Feature Reply message
- feature_reply_to_role_request_avg: time spent for ONOS processing mastership election for the device and sending OF Role Request message
- role_request_to_role_reply_avg: time spent to get OF Role Reply message from device
- role_reply_to_device_avg: time spent for ONOS to generate Device Event
- up_device_to_graph_avg: time spent for ONOS to generate Graph Event

Inspection of the detailed results reveals that the latencies are mostly due to the times the switch takes to respond to OF feature request message with OF feature reply message. This is confirmed by the steady ~35 ms lag, shown as the blue portion of the bars in figure 3, which is independent of the number of cluster instances.

ONOS' combined contributions to the latency ranges between 10 and 20 ms. The time spent in electing an ONOS instance to function as the master for the switch (shown as the orange portion) slightly grows with the number of ONOS instances. The additional small coordination overhead is expected as this step requires full consensus between all cluster instances.

For the other part including time spent in updating device store (shown as the yellow portion) and topology store (shown as the green portion) across the entire cluster, there is no additional cost as the cluster size grows.

Switch down test result:

- For all single and multi-instance setups, latency ranges between 3 - 5 ms, which is reduced by ~60% by comparison with Blackbird release.

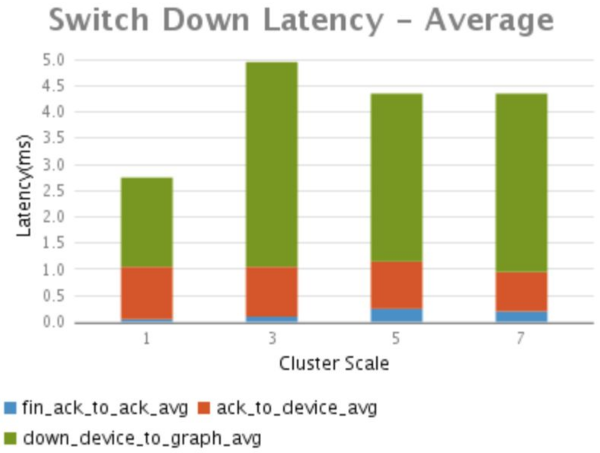

Figure 4

Each latency is further divided into three portions:

- fin_ack_to_ack_avg: time spent for TCP session teardown between ONOS and device
- ack_to_device_avg: time spent for ONOS to generate Device Event
- down_device_to_graph_avg: time spent for ONOS to generate Graph Event

Similar to the difference in link-down vs. link-up latencies, switch-down is significantly faster than switch-up, because there is no negotiation involved in switch-down processing. Once the control connection is torn down via TCP FIN, ONOS can categorically declare the switch as being unavailable and without any delay, remove it from its network graph.

Host discovery test result:

- For a single instance, the latency is around 4 ms.
- For multi-node cluster, the latency ranges between 75 and 125 ms.

- It is the first time we publish host discovery latency results in the white paper.

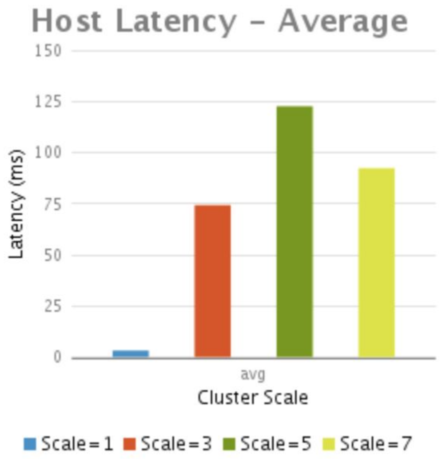

Figure 5

The increase in latency from single to multi-instance setups is again due to the selection of consistency models. Specifically, this is due to a configuration in how the raft protocol is used by default. By slightly relaxing some consistency and allowing any client to read from any up to date server for some operations, ONOS slightly increases latency for operations that must go through the raft leader. We are currently working on improving both how ONOS accesses the raft protocol and the way Copycat, the implementation of the raft protocol ONOS uses, handles these client/server interactions to better fit ONOS's use case.

The detailed results for link and switch event latency is available at https://wiki.onosproject.org/x/xobV and results for host event latency is available at https://wiki.onosproject.org/x/14bV.

Throughput of flow operations

Scenario

In a large network, it is likely that each instance in an ONOS cluster will serve a set of switches that share some geographic locality – call it a region. It is also expected that an ONOS cluster can serve multiple regions. Provisioning flows that span regions involves coordination and sharing of state between ONOS instances.

Goal

The goal of this experiment is to show how the flow subsystem of the ONOS Cluster performs and how it scales for managing flows that are within a region as well as flows that span multiple regions. More specifically, the experiment focuses on measuring the number of flow installations per second that ONOS provides as the number of instances is increased and as the number of regions spanned by the flows is increased.

An ONOS cluster should be able to process at least 1 Million flow setups per second, and a single instance should be able to process at least 250K flow setups per second. Furthermore, the rate should scale in a linear fashion with the number of cluster instances.

Experiment setup

In this setup the load is generated by a test fixture application (*DemoInstaller)* running on designated ONOS instances. To isolate the flow subsystem, the southbound drivers are “null providers,” which simply return without actually going to a switch to perform installation of the flow. (Note: These providers are not useful in a deployment, but they provide a frictionless platform for performance testing.)

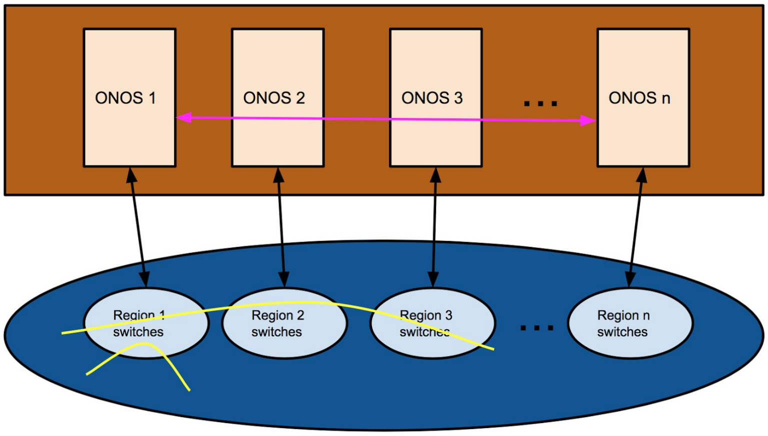

Figure 6

Constant Parameters:

- Total number of flows to install
- Number of switches attached to the cluster
- Mastership evenly distributed – each ONOS instance is master of the same number of switches

Varying Parameters:

- Number of servers installing flows (1, 3, 5 or 7)
- Number of regions a flow will pass through (1, 3, 5 or 7)

The set of constant and varying parameters above results in a total of sixteen experiments. At one end of the spectrum, all load is on one instance when both parameters are set to 1. In this case, only one instance is involved and there is no cluster coordination overhead. At the other end of the spectrum, the load is shared by all instances, all flows are handled by all instances and there is maximum amount of cluster coordination overhead.

The detailed test plan is available at https://wiki.onosproject.org/x/qoQ0.

Results

The above experiments yield the following observations:

- A single ONOS instance can install just over 700K local flow setups per second. An ONOS cluster of seven can handle 3 million local, and 2 million multi-region flow setups per second.
- With the same cluster size, the flow setup throughput is always lower in the multi-region case due to the cost of coordination between instances.
- In both single and multi-region cases, the flow setup throughput scales approximately linearly with respect to the size of the cluster.
- The average throughput numbers remain the same by comparison with Blackbird release.

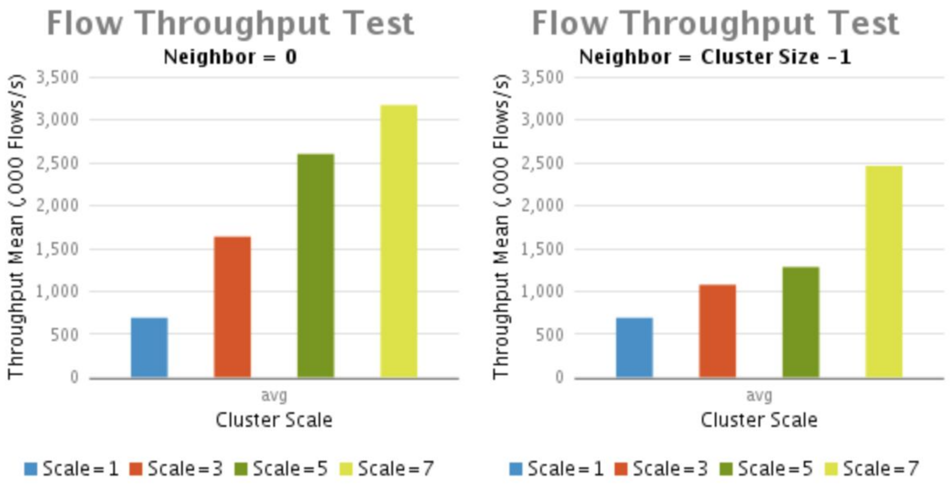

Figure 7

These results show that ONOS achieves not only the throughput performance objectives, but also the design objectives for scalability of the system.

The detailed results are available at https://wiki.onosproject.org/x/z4bV.

Latency of intent operations

Scenario

Applications interact with the ONOS intent subsystem by submitting or withdrawing intents. Both these operations are asynchronous, which means that the initial submit or withdraw operation returns almost immediately and after the intent has been processed and installed, an event will be generated to notify listeners about completion of the requested operation. The network environment interacts with the ONOS subsystem as well. While failover paths should be programmed into the data-plane to minimize the latency of responding to failures, there are times where data-plane failures require control-plane actions, and these actions should strive to have as low latencies as possible.

Goal

The goal is to measure how long it takes to completely process an application's submit or withdraw intent by ONOS and also to measure the length of time needed for a reroute in the case of a link failure. Finally, the goal is to understand how the intent subsystem behaves under load. This is accomplished by changing the batch size of requests submitted at the same time.

An ONOS cluster should be able to achieve latency under 50 ms for any single submit or withdraw intent request and under 20 ms for a reroute request. As the load increases with larger batch sizes, the overall latency is permitted to increase, but the average latency of an intent should decrease due to batching efficiency gains.

Experiment setup

In this experiment, a test application makes requests to ONOS intent subsystem to install or withdraw intent batches of varying sizes and measures how long it takes to fully complete each request. The setup also measures how long it takes to reroute an intent request after an induced network event.

As shown in the set-up below, five switches are controlled by three ONOS instances. The intents are installed by an application on the first instance. These intents are set up across the switches with the start and end time measured at the point of request. One of the links, between switches 2 and 4, can be cut to cause a reroute event.

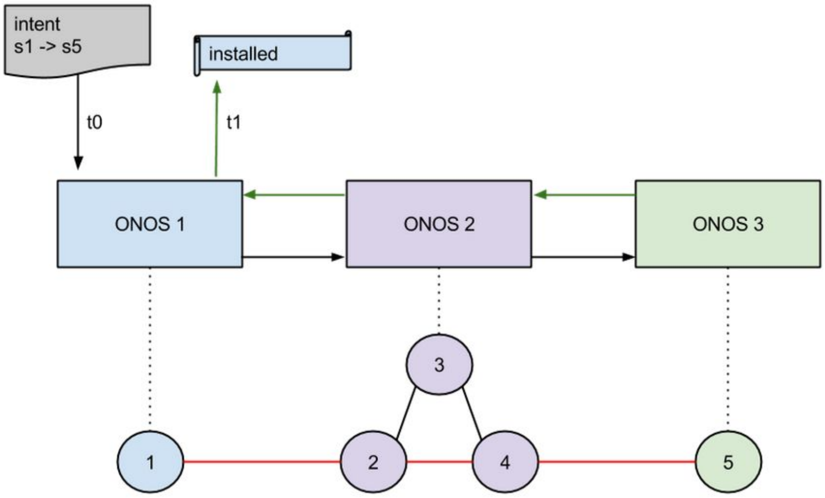

Figure 8

The detailed test plan is available at https://wiki.onosproject.org/x/pIQ0.

Results

Summary of the results:

- To submit, withdraw or reroute one intent, a single ONOS node reacts in 10~20 ms while multi-node ONOS reacts in 10~40 ms. The additional latency comes from coordination overhead in routing each intent request to its partition master and the subsequent routing of the resulting flow requests to each device master instance. Note however, that the cost is just a one-time step; there is no additional cost as the cluster grows.
- The latency grows with larger intent batch sizes. It takes ~20 ms more to handle intents with batch size 100 and 50~150 ms more to handle intents with batch size 1000. But average latency of handling one intent decreases as expected.
- The average latency numbers with batch size 1 and 100 remain the same by comparison with Blackbird release, and it is the first time we publish latency numbers with batch size 1000 in the white paper.

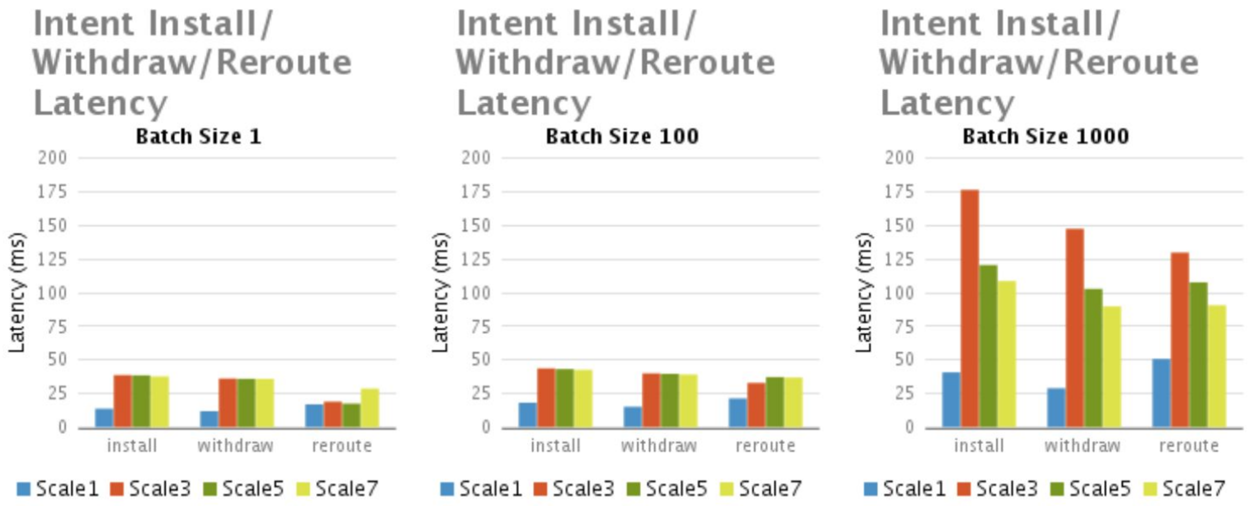

Figure 9

The detailed results are available at https://wiki.onosproject.org/x/yYbV.

Throughput of intent operations

Scenario

Dynamic networks undergo changes of connectivity and forwarding policies on an ongoing basis. Since the ONOS intent subsystem is the basis for provisioning such policies, it needs to be able to cope with a steady stream of requests. ONOS capacity to process intent requests needs to increase proportionally to the size of the cluster.

Goal

The objective of this scenario is to measure the maximum sustained throughput of the intent subsystem with respect to submit and withdraw operations. The goal is also to demonstrate that as the cluster size grows, so does its overall capacity to process intents.

A single-node cluster should be able to process at least 20K of intent operations per second and a 7-node cluster should have the capacity to process 100K of intent operations per second. The scalability characteristics should be approximately linear with respect to the size of the cluster.

Note: The order-of-magnitude difference between flow vs. intent performance goals stems from the fact that intent operations are inherently more complex than flow operations and in most cases involves processing multiple flows. Intent processing involves additional layer of work delegation (per-key mastership) and additional state distribution for HA to assure that in case of an instance failure, another instance in the ONOS cluster will continue the work on the intents that were submitted or withdrawn.

Experiment setup

In order to measure the maximum capacity of the system, a self-throttling test application (*IntentPefInstaller*) has been developed. This application subjects the intent subsystem to a steady stream of intent operations and it adjusts the workload based on the gap between pending operations and completed operations to keep the system running at maximum, but not saturated.

Two variants of this experiment are measured. One where all intent operations are handled locally on the same instance and the other where intent operations are a mixture of locally and remotely delegated work. The latter represents a scenario which is expected to be a typical case in production networks.

The detailed test plan is available at https://wiki.onosproject.org/x/poQ0.

Results

The following is the summary of the results:

- A single ONOS node can sustain more than 30K operations per second.
- 7-node ONOS cluster can sustain more than 200K operations per second.
- By comparison with Blackbird release, single node throughput stays the same and multi node throughput is increased by ~25% on average.

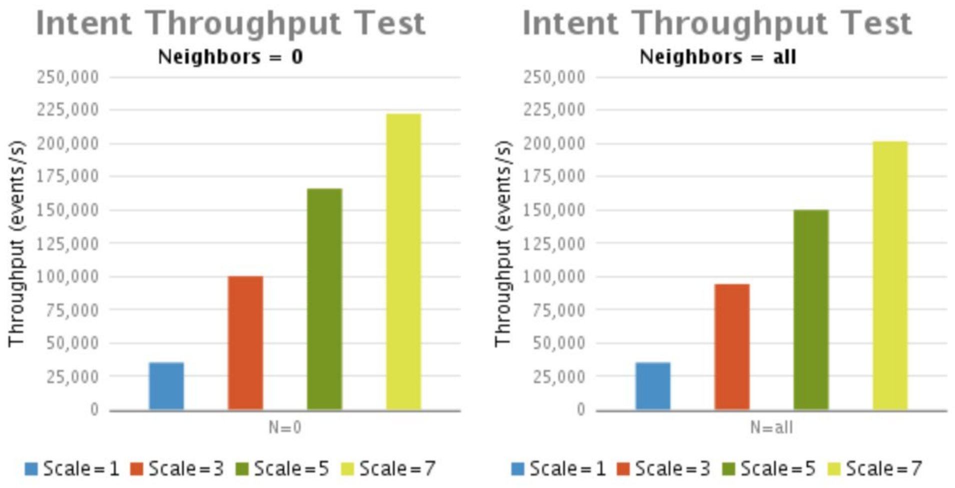

Figure 10

The linear scale-out characteristics of these results indicate that the design of the ONOS intent and flow subsystems successfully enables parallel processing of multiple independent operations in order to fold latencies and to achieve sustainable high throughput.

The detailed results are available at https://wiki.onosproject.org/x/y4bV.

Summary

Lack of a resilient, high performance SDN control plane has been a key barrier to SDN deployment. ONOS is the only open source distributed SDN Network Operating system that has been architected from the ground up to provide high availability, high performance and scale-out, defined a set of metrics to effectively evaluate and quantify these characteristics and published a comprehensive performance evaluation for its Kingfisher release.

ONOS aims to raise the bar for SDN control plane performance along with related measurement methodologies and launch an industry-wide movement towards openly providing similar performance measurements for all SDN control platforms for the benefit of the end-users.

Accelerating the Adoption of SDN & NFV

ONOS
Open Network Operating System

About ONOS

ONOS is the open source SDN networking operating system for Service Provider networks architected for high performance, scale and availability. The ONOS ecosystem comprises the ONF, organizations that are funding and contributing to the ONOS initiative, and individual contributors. These organizations include AT&T, China Unicom, Comcast, Google, NTT Communications Corp., SK Telecom Co. Ltd., Verizon, Ciena Corporation, Cisco Systems, Inc., Ericsson, Fujitsu Ltd., Huawei Technologies Co. Ltd., Intel Corporation, NEC Corporation, Nokia, Radisys and Samsung. See the full list of members, including ONOS' collaborators, and learn how you can get involved with ONOS at onosproject.org.

ONOS is an independently funded software project supported by The Linux Foundation, the nonprofit advancing professional open source management for mass collaboration to fuel innovation across industries and ecosystems.