Architectural Tenets

- **High-availability, scalability and performance**
  - required to sustain demands of service provider & enterprise networks

- **Strong abstractions and simplicity**
  - required for development of apps and solutions

- **Protocol and device behaviour independence**
  - avoid contouring and deformation due to protocol specifics

- **Separation of concerns and modularity**
  - allow tailoring and customization without speciating the code-base
ONOS Distributed Architecture

- **Apps**
- **NB Core API**
- **Distributed Core**
  (state management, notifications, high-availability & scale-out)
- **SB Core API**

- Providers
  - Protocols
- Providers
  - Protocols
- Providers
  - Protocols
- Providers
  - Protocols
ONOS Subsystems - Today & 2015

- External Apps
- REST API
- CLI
- Mobility
- Proxy ARP
- L2 Forwarding
- SDN IP / BGP
- Packet / Optical
- Application
- UI Extension
- Security
- Config
- Storage
- Region
- Core
- Cluster
- Leadership
- Event
- Messaging
- Graph
- OSGi / Apache Karaf
- Device Cfg.
- Discovery
- Network Virt.
- Tenant
- Driver
- Path
- Tunnel
- Intent
- Security
- Mastership
- Topology
- Network Cfg.
- Flow Objective
- Group
- Device
- Link
- Host
- Flow Rule
- Packet
- OpenFlow
- NetConf
- OVSDB

Available today
Roadmap items for 2015
New Platform Subsystems

- **Applications Service**
  - facilitates easy software deployment across the entire cluster
  - used for apps, drivers and protocol providers

- **Component Configuration Service**
  - facilitates software component configuration across the entire cluster

- **Device Drivers Service**
  - brokerage for device-specific code

- **Flow Objective Service**
  - allows programming flow rule tables in pipeline-agnostic manner
  - utilizes device drivers
State Management Architecture

“The unavoidable price of reliability is simplicity.”
— C.A.R. Hoare

“Simplicity is prerequisite for reliability.”
— E. W. Dijkstra
Control Plane State

- Topology
- Flows
- Intents
- Switch to controller mapping
- Resource allocations
- Network Configuration
- And a plethora of application generated state
Topology

- Observed network state
Topology

- Observed network state
- Each controller directly observes a subset of network
Topology

- Observed network state
- Each controller directly observes a subset of network
- Applications access Global Network View in its entirety
Topology

- Observed network state
- Each controller directly observes a subset of network
- Applications access Global Network View in its entirety
- Data plane is source of truth
Flows

- Data plane forwarding rules
- Naturally partitioned by forwarding element (switch)
- Control plane is the source of truth
**Intents**
- Application specific network policy definitions
- Immutable and durable
- Intent State is eventually consistent

**Device Mastership**
- Switch to Controller Assignment
- Strongly consistent

**Resource State**
- Bandwidth allocations, calendering, etc.
- Strongly consistent and durable
- Needs to scale
Drawback of a single (Logically) Central Datastore

- Tuned for either high availability or strong consistency
- Fails to exploit the access patterns and locality constraints of some state
ONOS: Polyglot State Management

- There is no one solution to rule them all
- A core set of state management primitives
Topology

- Fully replicated in-memory state
- Applications access topology state at memory speed
Topology

- Fully replicated in-memory state.
- Applications access topology state at memory speed
• By exploiting the nature of updates guarantees

Monotonic Read Consistency
State Management Primitive

EventuallyConsistentMap<K, V, T extends Timestamp>
Flows

- Primary-backup replication
- Switch master is *primary* and next master is *backup*
- Fast read/write access
- Minimal overhead on failover
Primitive for strong consistency...

ConsistentMap\langle K, V \rangle

Think Java’s ConcurrentMap\langle K, V \rangle in a distributed setting.
State requiring strong consistency

Replicated State Machine with RAFT Consensus
State requiring strong consistency

Replicated State Machine with RAFT Consensus

Partitioned for scalability
Other State Management Primitives

- Transactional Map
- Distributed Set
- Atomic Counter
- Leadership Service
Flow Objective Abstraction

- **Problem:** Applications currently must be pipeline aware, effectively making applicable on specific HW.
Flow Objective Abstraction

- **Problem:** Applications currently must be pipeline aware, effectively making applicable on specific HW. **Flow objectives enable developers to write applications once for all pipelines**
Applications use Objective to take advantage of multi-table architectures.

Other services also make use of the Objective service (e.g., Intent Service).

Device driver translates objectives to the specific flow rules for a given device.
Flow Objectives

- Flow Objectives describe a SDN application’s objective behind a flow it is sending to a device.

- We currently only have three types of objectives:
  1. Filtering Objective
  2. Forwarding Objective
  3. Next Objective
Filtering Objective

- Filter -> only Permit or Deny options
- On criteria (match fields)

Example:
Peering Router
Switch Port: X
Permit: MAC 1, VLAN 1, IP 1, 2, 3
Permit: MAC 1, VLAN 2, IP 4, 5
Filtering Objective

- Filter -> only Permit or Deny options
- On criteria (match fields)

Example:
Peering Router

Switch Port : X

Permit: MAC 1, VLAN 1, IP 1, 2, 3
Permit: MAC 1, VLAN 2, IP 4, 5
Next Objective

- Next -> next hop for forwarding
- Similar to OF group
- Keyed by a NextId used in Forwarding Objectives
Forwarding Objective

- Forwarding: \{ Selector -> Next Id \}
- Forwarding Types: Specific or Versatile
  - Specific -> MAC, IP, MPLS forwarding tables
  - Versatile -> ACL table
- NextId is resolved to whatever the driver previously built for the corresponding Next Objective
Objectives - Simpler applications

```java
for (InterfaceIpAddress ipAddr : intfIps) {
    log.debug("adding rule for IP: ", ipAddr.ipAddress());
    selector = DefaultTrafficSelector.builder();
    treatment = DefaultTrafficTreatment.builder();
    selector.matchEthType(Ethernet.TYPE_IPV4);
    selector.matchIPDst(ipAddr.valueOf(ipAddr).ipAddress(), 32));
    treatment.transition(Criteria.ETH_TYPE));
    rule = new DefaultFlowRule(deviceId, selector.build(),
                                treatment.build(), HIGHEST_PRIORITY, appID,
                                0, true, FlowRule.Type.IP);
    ops = install ? ops.add(rule) : ops.remove(rule);
}

private void processIntFilters(boolean install, Set<Interface> intfs) {
    log.info("Processing {} router interfaces", intfs.size());
    for (Interface intf : intfs) {
        FilteringObjective.Builder fob = DefaultFilteringObjective.builder();
        fob.withKey(Criteria.matchInPort(intf.connectPoint().port()));
        .addCondition(Criteria.matchEthDst(intf.mac()));
        .addCondition(Criteria.matchVlanId(intf.vlan()));
        intf.ipAddresses().stream()
            .forEach(ipaddr -> fob.addCondition(Criteria.matchIPDst(ipaddr.subnetAddress())));
        fob.permit().fromApp(appId);
        flowObjectiveService.filter(deviceId, Collections.singletonList(fob.add()));
    }
}
```
Flow Objective Summary

- *Flow Objective Service*: Abstraction for applications to be pipeline unaware while benefiting from scalable, multi-table architectures
- Aims to make it simple to write apps
- First attempt at achieving interoperability between OF 1.3 implementations
Building Network Applications

Objective: Connect Host 1 and Host 2
Building Network Applications

Objective: Connect Host 1 and Host 2

1. Read/discover the topology
Building Network Applications

Objective: Connect Host 1 and Host 2
1. Read/discover the topology
2. Compute a path
Building Network Applications

Objective: Connect Host 1 and Host 2
1. Read/discover the topology
2. Compute a path
3. Build flow objectives for each device
Building Network Applications

Objective: Connect Host 1 and Host 2
1. Read/discover the topology
2. Compute a path
3. Build flow objectives for each device
4. Install rules in consistent way
Building Network Applications

Objective: Connect Host 1 and Host 2
1. Read/discover the topology
2. Compute a path
3. Build flow objectives for each device
4. Install rules in consistent way
Building Network Applications

What can go wrong?
Building Network Applications

What can go wrong?

- Missing / rejected / dropped rules
  - Monitor devices connections
  - Send barriers between rule updates
  - Poll flow state
Building Network Applications

What can go wrong?

- Missing / rejected / dropped rules
  - Monitor devices connections
  - Send barriers between rule updates
  - Poll flow state

- Topology changes
  - Listen to switch, port, link and host events
  - Compute new path that leverage or remove old flows
Building Network Applications

- Each application requires complex path computation and rule installation engines and state machines
- Inconsistent behavior in the face of failures
  - Failures may be handled in different ways (or not at all)
- Bugs need to fixed in multiple places (applications)
- Expensive to upgrade/refactor behavior across all applications; e.g.
  - Improve performance
  - Support new types of devices
  - Implement better algorithms
- Difficult or impossible to resolve conflicts with other applications
Intent Framework

• Provides high-level, network-centric interface that focuses on what should be done rather than how it is specifically programmed
• Abstracts unnecessary network complexity from applications
• Maintains requested semantics as network changes
• High availability, scalability and high performance
Intent Example

Host to Host Intent
Intent Example

Host to Host Intent

submit()

Intent Service API
Intent Example

Host to Host Intent

Path Intent

Path Intent
Intent Example

- **Host to Host Intent**

- **Path Intent**
- **Flow Rule Batch**

**COMPILATION**

**INSTALLATION**
Intent Compilers

• Produce more specific Intents given the environment
• Works on high-level network objects, rather than device specifics
• “Stateless” – free from HA / distribution concerns
Intent Installers

• Transform Intents into device commands
• Allocate / deallocate network resources
• Define scheduling dependencies for workers
• “Stateless”
Under Development

- More generic / improved abstractions
  - Connectivity with constraint
  - Default policy
  - Filtering / Monitoring

- Composition (Frenetic/CoVisor-like)
  - Between Intent primitives in a given application
  - Across multiple applications

- Device update optimizations
  - Batching
  - Minimize flow table updates
Example Applications

- **SDN-IP Peering**
  - Connect internal BGP software daemon to external BGP routers
  - Install learned routes to forward IP traffic to appropriate egress point

- **Multi-level (IP / Optical) Provisioning**
  - Provision optical paths/tunnels with constraints

- **Content Acquisition / Video Streaming (DirecTV)**
  - Establish multicast forwarding from a sender to set of receivers

- **Virtual Network Gateway (vBNG)**
  - Provide connectivity between a private host and the Internet

- **Bandwidth Calendaring**
  - Establish tunnels with bandwidth guarantees between two points at a given time
Intent Framework Summary

- Intents are a *network-centric programming abstraction* that reduce application complexity.

- Intents provide *device-agnostic behavior* with *persistency* and *high performance* across network failures.

- Intent framework has moved from prototype to *production* deployments.
Control Plane Performance

- Throughput of proactive provisioning actions
  - path flow provisioning
  - global optimization or rebalancing of existing path flows

- Latency of responses to topology changes
  - path repair in wake of link or device failures

- Throughput of distributing and aggregating state
  - batching, caching, parallelism
  - dependency reduction

- Controller vs. device responsibilities
  - defer to devices to do what they do best, e.g. low-latency reactivity, backup paths
Performance Metrics

● Device & link sensing latency
  o measure how fast can controller react to environment changes, such as switch or port down to rebuild the network graph and notify apps

● Flow rule operations throughput
  o measure how many flow rule operations can be issued against the controller and characterize relationship of throughput with cluster size

● Intent operations throughput
  o measure how many intent operations can be issued against controller cluster and characterize relationship of throughput with cluster size

● Intent operations latency
  o measure how fast can the controller react to environment changes and reprovision intents on the data-plane and characterize scalability
Device & Link Sensing Latency

1 - Device Connect/Disconnect to ONOS1;
2a - OF Protocol Exchanges in connect event only;
3 - ONOS Core Event Processing
Link Up/Down Latency

- Since we use LLDP & BDDP to discover links, it takes longer to discover a link coming up than going down.
- Port down event trigger immediate teardown of the link.
Flow Rule Operations Throughput
Flow Throughput results

- Single instance can install over 500K flows per second
- ONOS can handle 3M local and 2M non local flow installations
- With 1-3 ONOS instances, the flow setup rate remains constant no matter how many neighbours are involved
- With more than 3 instances injecting load the flow performance drops off due to extra coordination requires.
Intent Throughput Experiment
Intent Throughput Results

Intent Operation Throughput

Sustained overall rate

- Processing clearly scales as cluster size increases
Intent Latency Experiment
Intent Latency Results

- Less than 100ms to install or withdraw a batch of intents
- Less than 50ms to process and react to network events
  - Slightly faster because intent objects are already replicated
Key Takeaways

● Lack of high performance, scalable, highly available SDN control plane and solutions are key barriers to SDN adoption in service provider networks

● ONOS addresses this challenge with logically centralized but distributed architecture that provides performance, scale, HA together

● Comprehensive set of metrics and measurements for ONOS Blackbird release are published on the wiki @ http://bit.ly/blackbird-eval
Summary

ONOS continues to evolve to:

● operate ever more efficiently and reliably
● support larger scale networks
● further simplify the programming model for applications
● broaden the set of supported devices and protocols
● simplify SDN deployments
Software Defined Transformation of Service Provider Networks

Join the journey @ onosproject.org