Onix: A Distributed Control Platform for Large-scale Production Networks

Teemu Koponen  Nicira Networks
Martin Casado  Nicira Networks
Natasha Gude  Nicira Networks
Jeremy Stribling  Nicira Networks
Leon Poutievski  Google
Min Zhu  Google
Rajiv Ramanathan  Google
Yuichiro Iwata  NEC
Hiroaki Inoue  NEC
Takayuki Hama  NEC
Scott Shenker  International Computer Science Institute & UC Berkeley

OSDI’10 Proceedings of the 9th USENIX conference on Operating systems design and implementation
GINM PHD student Hongyunn Chen D05944018
1 Introduction

The most important challenges in building a production-quality control platform are:

• Generality
• Scalability
• Reliability
• Simplicity
• Control plane performance
2 Design

Understanding how Onix realizes a production-quality control platform requires discussing two aspects of its design: the context in which it fits into the network, and the API it provides to application designers.
2.1 Components

There are four components in a network controlled by Onix, and they have very distinct roles.

• Physical infrastructure
• Connectivity infrastructure
• Onix
• Control logic
2.1 Components

Server 1

Network Control Logic

NIB

Switch Import / Export

Distribution I / E

Onix

Server N

Network Control Logic

NIB

Distribution I / E

Switch Import / Export

Management Connectivity Network Infrastructure

Managed Physical Network Infrastructure
2.2 The Onix API

- “Data-centric” API

- Treat all networking actions as data actions
  - Read
  - Alter
  - Register for changes in network state
2.3 Network Information Base

• Network information base
  – Analogous to forwarding information base

• Graph of all network entities
  – Switches, ports, interfaces, links

• Applications read/register/manipulate NIB
2.3 Network Information Base

- NIB is a collection network entities
- Each entity is a key-value pair

Default network entity classes
### 2.3 Network Information Base

<table>
<thead>
<tr>
<th>Category</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>Find entities.</td>
</tr>
<tr>
<td>Create, destroy</td>
<td>Create and remove entities.</td>
</tr>
<tr>
<td>Access attributes</td>
<td>Inspect and modify entities.</td>
</tr>
<tr>
<td>Notifications</td>
<td>Receive updates about changes.</td>
</tr>
<tr>
<td>Synchronize</td>
<td>Wait for updates being exported to network elements and controllers.</td>
</tr>
<tr>
<td>Configuration</td>
<td>Configure how state is imported to and exported from the NIB.</td>
</tr>
<tr>
<td>Pull</td>
<td>Ask for entities to be imported on-demand.</td>
</tr>
</tbody>
</table>

**Functions provided by the ONIX NIB API**
3 Scaling and Reliability

• Partition
  – different subnet forwarding rules on a switch

• Aggregate
  – zoom-in/zoom-out at different aggregation levels

• Tradeoff with weaker consistency/durability
  – replicated transactional DB for network topology
  – one-hop DHT for link utilization info
3 Scaling and Reliability

• DHT with weak eventual consistency
  – Used for “high” churn events
  – Frequent updates

• Transactional store with strong guarantees
  – Used for “low” churn events
  – network policy
3 Scaling and Reliability

• Network element failure
  – discovered by traditional data plane mechanisms
  – application is in charge of deciding about the alternative policy after node/link failure

• ONIX instance failure
  – Option 1: other instances detect failure and take over
  – Option 2: have multiple instances manage a network element the network at all times

• Infrastructure failure
  – Use dedicated control backbone
4 Distributing the NIB

how Onix distributes its Network Information Base and the consistency semantics an application can expect from it.
4.1 Overview

Onix’s support for state distribution mechanisms was guided by two observations on network management applications.

First, applications have differing requirements on scalability, frequency of updates on shared space, and durability.

Second, distinct applications often have different requirements for the consistency of the network state they manage.
4.2 State Distribution Between Onix Instances

Applications can easily group NIB modifications together into a single transaction to be exported to the database.

For network state needing high update rates and availability, Onix provides a one-hop, eventually-consistent, memory-only DHT, relaxing the consistency and durability guarantees provided by the replicated database.

Onix DHT returns multiple values for a given key, and it is up to the applications to provide conflict resolution, or avoid these conditions by using distributed coordination mechanisms.
4.3 Network Element State Management

Onix turns OpenFlow events and operations into state that it stores in the NIB entities.

When an application adds a flow entry to a ForwardingTable entity in the NIB, the OpenFlow export component will translate that into an OpenFlow operation that adds the entry to the switch TCAM.

By creating and attaching Port entities with proper attributes to a ForwardingEngine entity (which corresponds to a single switch datapath), applications can configure new tunnel endpoints without knowing that this translates to an update transaction sent to the corresponding switch.
4.4 Consistency and Coordination

Onix expects the applications to register inconsistency resolution logic with the platform. Applications have two means to do so.

First, in Onix, entities are C++ classes that the application may extend, and thus, applications are expected simply to use inheritance to embed referential inconsistency detection logic into entities so that applications are not exposed to inconsistent state.

Second, the plugins the applications pass to the import/export components implement conflict resolution logic, allowing the import modules to know how to resolve situations where both the local NIB and the data source have changes for the same state.
5 Implementation

Onix consists of roughly 150,000 lines of C++ and integrates a number of third party libraries.

Onix is a harness which contains logic for communicating with the network elements, aggregating that information into the NIB

Onix currently supports C++, Python, and Java
# 6 Applications

<table>
<thead>
<tr>
<th>Control Logic</th>
<th>Flow Setup</th>
<th>Distribution</th>
<th>Availability</th>
<th>Integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethane</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Distributed virtual switch</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Multi-tenant virtualized datacenter</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Scale-out carrier-grade IP router</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Aspects of Onix especially stressed by deployed control logic applications.
7 Evaluation

- evaluate Onix in two ways: with micro-benchmarks, designed to test Onix’s performance as a general platform, and with end-to-end performance measurements of an in-development Onix application in a test environment.
7.1 Scalability Micro-Benchmarks

- Single-node performance

Figure 3: Attribute modification throughput as the number of listeners attached to the NIB increases.
7.1 Scalability Micro-Benchmarks

• Single-node performance

Figure 4: Memory usage as the number of NIB entities increases.
7.1 Scalability Micro-Benchmarks

- Single-node performance

Figure 5: Number of 64-byte packets forwarded per second by a single Onix node, as the # of switch connections increases.
### 7.1 Scalability Micro-Benchmarks

<table>
<thead>
<tr>
<th>Queries/trans</th>
<th>1</th>
<th>10</th>
<th>20</th>
<th>50</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queries/s</td>
<td>49.7</td>
<td>331.9</td>
<td>520.1</td>
<td>541.7</td>
<td>494.4</td>
</tr>
</tbody>
</table>

**Table 3**: The throughput of Onix’s replicated database.
Figure 7: A CDF showing the latency of updating a DHT value at one node, and for that update to be fetched by another node in a 5-node network.
Figure 8: A CDF of the perceived communication disruption time between two hosts when an intermediate switch fails. These measurements include the one-second (application-configurable) keepalive timeout used by Onix. The hosts measure the disruption time by sending a ping every 10 ms and counting the number of missed replies.
8 Related Work

An orthogonal line of research focuses on offering network developers an extensible forwarding plane (e.g., RouteBricks [11], Click [22] and XORP [17]).

Onix can be the platform for flexible data center network architectures such as SEATTLE [21], VL2 [14] and Portland [25] to manage large data centers.

Onix also follows the path of many earlier distributed systems that rely on applications’ help to relax consistency requirements in order to improve the efficiency of state replication. Bayou [31], PRACTI [2], WheelFS [29] and PNUTS [8] are examples of such systems.
9 Conclusion

Onix provides general tools for managing state, but it does not magically make problems of scale and consistency disappear.

We are still learning how to build control logic on the Onix API, but in the examples we have encountered so far management applications are far easier to build with Onix than without it.
Thank you