Network-aware Applications

Ethernet Technology Summit, April 2013
Rob Sherwood
CTO - Controller Technologies; ONF Chair ArchWG
Outline

- **Problem:**
  - Application network requirements hard to express and enforce

- **Network-aware Applications vs. Application-aware networks**
  - Terminology and Context: My ONF hat as ArchWG chair
  - Proposed Architecture
  - Three Critical Properties

- **Example applications with requirements**
  - Multi-tenant VM placement
  - Elephant Flow Network Traffic Optimization
  - Network Security Monitoring and Enforcement

- **Implications for Application Developers**
  - Example Application to network APIs
  - Example Network to application APIs
Informal Terminology
Will define more formally in a bit.

- **Application**: code that *uses* the network
  - E.g., OpenStack/Nova, Oracle, Hadoop, backup process, security monitor, webserver, etc.
  - *Not* a learning switch or TE computation; they are network services
  - Different from many people’s definition of “application”

- **Network device**: any box that makes forwarding decisions
  - E.g., switch, router, software switch, IDS, firewall, load-balancer, NIC, ROADM, DSLAM, Wi-Fi access point, etc.
  - *Not* optical repeater, cable, splitter, transceiver; no decisions

- Intentionally avoided terms: they are too imprecise
  - Southbound/Northbound API
    - Inherently depends on perspective (“north from where?”)
Context: My ONF Architecture Hat

- Chair of Architecture and Framework working group

- Chartered to:
  - Define an Architecture for SDN
  - Describes APIs and their properties
  - Show that many use-cases fit into this architecture

- Lots of smart people, experience, points of view

- Growing Personal Conclusion:
  - Application-aware networking is the fundamental essence and value proposition of SDN
  - Many people starting to agree with me 😊
**Problem: App Requirements Hard to Express/Enforce**

Requirements described and enforced off-line, imprecisely, and statically.
Result: Network is Inflexible, Brittle, and Inefficient

Admins Must Manually Translate App Requirements to CLI

- **Off-line → Inflexible:**
  - Applications must describe initial requirements off-line, e.g., via email.
  - Might need to move existing Apps elsewhere, change IPs, re-map vlans, etc.
  - Reduces feature agility, “later” == “no” → *lost potential revenue*

- **Imprecise → Brittle:**
  - No formal language for describing requirements or low-level control of devices.
  - Did the admin understand the requirements correctly? Type the right config?
  - No formal/easy way to check if requirements are actually met; hard to audit!
  - Risks downtime → *higher opex cost*

- **Statically → Inefficient:**
  - Applications can’t change requirements over time: provision for the peak.
  - Provisioning guesswork: “how much bandwidth does my app *actually* need?”
  - No feedback from network: “Should I xfer the file now?”
  - Enforces low utilization and over provisioning → *higher capex costs*
Application-aware Networks vs. Network-aware Applications

More network awareness is more control but also more programming effort

- **Zero awareness** - “deploy and pray”
  - Fast, Fault-prone, Static – okay for some applications
  - Historically has worked well due to over provisioning
- Out-of-band coordination – Email, trouble tickets
  - Slow, less fault-prone, static – typical but inefficient
- Application-aware data plane network:
  - *Formal* DPI and traffic classification
  - Hard to scale, assumes choke point
- Application-aware control plane net:
  - *On-line* DB of *formal* application requirements and identifying characteristics
  - Manage DB through self-service portal
- **Network-aware applications** written to network APIs
  - *On-line, Formal, and Dynamically* communicates requirements
  - Reacts online to network changes, e.g., load, link down

Match effort to reward:
- Large peak/mean ratio
- Strict Application SLA
- Hyper-scale


**Solution: Express and Enforce Requirements via API**

Requirements described and enforced on-line, formally, dynamically

---

**Network Controller**

- Provides network stats up to Apps
- Translates requirements down to Devices

---

**Network Devices**

- Switch
- Firewall
- Host X
- Switch
- Backup Server Y

---

**Applications**

- Network
- Devices
- File Backup App

---

**Network Controller**

- Configures Network Policy
- Monitors Performance

---

**Opportunity to Standardize**

Standardized API/Protocol
- Enforced Behavior
- Low-level Control
- Capability Discovery
- Statistics and Faults

---

**1. Explicit Requirements**

- App’s Explicit Requirements
- Network Statistics, Hints, and Events

---

**2. Network Controller**

- Provides network stats up to Apps
- Translates requirements down to Devices

---

**3. Host X and Backup Server Y**
Three Critical Properties of this Architecture

1. Applications are network aware: **SDN-enabled Applications**
   1. Communicate their requirements/policies to the network
   2. Can monitor network state and adapt accordingly

2. Network is logically centralized: **SDN Network Controller**
   1. Controller translates from application requirement to low-level rules
   2. Controller summarizes the network state for applications

3. Well-understood driver-like model for devices: **SDN Datapath**
   1. Programmatic low-level control of all forwarding and configuration
   2. API for Capabilities advertisement and publishing statistics
   3. No resource contention with other entities
      - Controller “owns” this device, subject to capabilities advertisement
Outline

- Problem: Applications can’t express their network requirements
- Network-aware Applications vs. Application-aware networks
  - Terminology and Context: My ONF hat as ArchWG chair
  - Proposed Architecture
  - Three Critical Properties
- Example applications with requirements
  - Multi-tenant VM placement
  - Elephant Flow Network Traffic Optimization
  - Network Security Monitoring and Enforcement
- Implications for Application Developers
  - Example Application to network APIs
  - Example Network to application APIs
Example: Multi-tenant Virtual Machine Orchestration
Requirements described and enforced on-line, formally, dynamically

Network Controller
- Provides network stats up to Apps
- Translates requirements down to Devices

Quantum API
Link Utilization
For Net-aware VM Placement
(Blue Prints)

Network Controller

Applications
VM A,B: Allow
VM X,Y: Allow
Default: Deny

Network Devices
Server + VSwitch
Firewall
Server + VSwitch
GRE Tunnel

VM A
VM X
VM B
VM Y

OpenStack
+ Nova

Network Controller

OpenFlow + OVSDB
Firewall API

Server + VSwitch

Network Controller

Network Controller

Server + VSwitch

Firewall

Server + VSwitch

Firewall

Server + VSwitch

OpenFlow + OVSDB

Quantum API

VM A,B: Allow
VM X,Y: Allow
Default: Deny

Network Controller

Network Controller

Network Controller

Network Controller

Network Controller

Network Controller

Network Controller

Network Controller
Example: Elephant Flow Traffic Optimization

Requirements described and enforced on-line, formally, dynamically

1. Applications
   - Elephant Flow Traffic Optimization

2. Network Controller and Network Admin
   - Provides network stats up to Apps
   - Translates requirements down to Devices

3. Network Devices
   - Reactive Best-fit Flow Placement
   - Link Utilization To Dynamically Re-fit Traffic

Network Controller

- Monitors Performance
- Big Data Traffic Scheduler
- Big Data Set

- Router
- Host A
- OF
- OpenFlow

Reactive Best-fit Flow Placement

Link Utilization To Dynamically Re-fit Traffic

Monitors Performance
Example: Network Security Monitoring
Requirements described and enforced on-line, formally, dynamically

Network Controller and Network Admin

1. Re-route or Tap Flow To IDS Then BLOCK or ALLOW

Network Controller
- Provides network stats up to Apps
- Translates requirements down to Devices

Network Security Monitor
- Monitors Performance
- Statistics for Anomalies

Router
- Host A

Alert!

Host B

Applications

Network Devices
Outline

- Problem: Applications can’t express their network requirements
- Network-aware Applications vs. Application-aware networks
  - Terminology and Context: My ONF hat as ArchWG chair
  - Proposed Architecture
  - Three Critical Properties
- Example applications with requirements
  - Multi-tenant VM placement
  - Elephant Flow Network Traffic Optimization
  - Network Security Monitoring and Enforcement
- Implications for Application Developers
  - Example Application to network APIs
  - Example Network to application APIs
APIs from Application to Controller
Brainstorming: many possibilities

- Managing Connectivity:
  - “Allow host X talk to host Y”
  - Set default policy for the network: allow or deny?
  - Quantum-like virtual networking: groups of hosts

- Path selection: Optimize for latency, cost, bandwidth, reliability?

- Publish/subscribe traffic feeds
  - “Send the NASDAQ multicast stream to hosts X, Y, and Z”

- Hints for Efficiency:
  - “I am sending 10 GB from host X to host Y – please optimize”

- Active Monitoring: “Duplicate packets in flow F and send them to host Y”
APIs from Network to Controller
Brainstorming: many possibilities

- Current network load:
  - Bandwidth calendaring: “Is now a good time to send my file?”
  - Machine virtualization: “Where do I put my VMs for network load?”

- Network events:
  - Faults detection: “If a link went down, I want to annotate my data”
  - Security monitoring: “Tell me if traffic to my server exceeds 1Gbps”

- Passive monitoring:
  - Billing: “Tell me the top 10 applications by network utilization”
  - Auditing: “Tell me if ever host X tries to talk to host Y”
Conclusion

- **Network-aware Applications can improve:**
  - Deployment time by negotiating requirements on-line
  - Network resiliency by formalizing requirements
  - Network efficiency by dynamically re-negotiating requirements

- **Three critical properties of the architecture:**
  - Applications explicitly describe their requirements and read network state
  - Network controller translates requirements to network device API calls
  - Network devices implement a well-understood driver-model

- **My claims:**
  - For too long, we’ve dodged these problems by over-provisioning
  - Network-aware applications are the reality of the SDN vision
Backup Slides
Insight: Resource Allocation is a *Software* Problem
This is what server operating systems already do!

- Dynamically learn requirements
  - How much memory, disk space, CPU each app is using?
- Well-defined isolation mechanisms and monitoring
  - Virtual memory, disk quotas, process CPU priority
- Dynamically map requirements onto available resources
  - Schedule process 2 on CPU 4; swap out the memory on process 3

**But** will this work for networks?
- No way for apps to query network state or describe requirements
- Networks are distributed; no centralized scheduler like in OSes
- No “low-level” control of underlying network resources
Not Excluded from the Big Picture
The architecture document will cover these in detail.

- Multi-controller clusters: controllers are logically centralized, not physically

- Network device virtualization/slicing/partitioning (e.g., flowvisor, logical router instances): seems to violate the “controller owns device” constraint, but virtualization should only modify the capabilities advertisement, so 100% of control is still ceded

- Hybrid layering: SDN controller overlay on non-SDN controlled underlay or vice versa (instance of slicing - see above)

- Hybrid devices: if ships-in-the-night, see network device virtualization, else is not SDN because it violates the “controller owns device” constraint

- Hybrid networks: An SDN controller is not prevented from also speaking non-SDN protocols, e.g., BGP, I2RS, etc., but doing so adds additional complexity
Implications for Application/Controller APIs

This is what people have been calling the “North Bound APIs”

- Lots of “application to controller” protocols fit our criteria
  - IETF ALTO
  - OpenStack Quantum
  - Bandwidth Calendaring: “I will xfer from \( B \) bytes \( X \) to \( Y \) at time \( T \)”
  - Most OpenFlow controllers export a network graph (e.g., via REST)

- Each application has its own needs, requirements
  - Not yet clear what is overlap and what is unique

- Recommended next steps for ONF:
  1. Identify use-cases and *deploy*
  2. Consolidate APIs if/where it makes sense, and
  3. *only then* standardize APIs – probably a while from now
Implications for Controller/Device

- Lots of “controller to datapath” protocols fit our definition
  - OpenFlow, OFConfig – low-level control of forwarding tables and device config
    - Well, mostly, but we know the issues and are working on them – the intent is clear
    - OpenFlow is the only low-level forwarding table management protocol that we know of
  - NetFlow/Sflow: give raw packet statistics

- But many do not:
  - I2RS: Device is not fully owned by the controller; complicates controller scheduling
  - OSPF/BGP/ISIS: Device is not fully owned by the controller
  - CLI: Not “low-level” – can’t manage all aspects of the device

- And some are confusing, e.g., SNMP and NetConf depend on which MIB/Schema
  - Reading raw stats: yes – low-level visibility
  - Tweaking link weights, PING MIB: no – there are lower-level primitives for this

- Each network device has its own capabilities and forwarding/config knobs
  - E.g., vswitches vs. physical ethernet switches vs. firewalls vs. ROADM

- Need to (1) Identify and deploy use-cases, (2) consolidate APIs if/where it makes sense, and only then (3) standardize APIs – probably a while from now
Implications for Standardization

- A Complete SDN Solution will likely contain a mix of standards: north and south
- Some imported from other SDO’s and “blessed” by the ONF
  - Obvious: south - TCP, Ethernet, SSL/TLS; north - Quantum
  - Less obvious: south – SNMP, netconf; north– SNMP traps from controller, ALTA
- Some will be tailor made by ONF – once we figure out where the holes are
  - South—OpenFlow, OFConfig (really new yang model on top of existing netconf)
  - North—bandwidth calendaring, traffic steering APIs
- Two competing tensions in old vs. new:
  - Leverage already existing standards; don’t reinvent the wheel
  - Reduce complexity by abstracting away superfluous and historical differences
- Recommendations:
  - Proceed very tactically – better standards are just optimizations, not blockers
  - Decouple “protocol” architecture from “software” architecture