2014 ONUG Hackathon: Team Conglomerate

David Crosbie & Andreas Voellmy
Spiros Eliopoulos, Ed Henry, Eric Murray, Eric Yspeert
Use Case

Data Center

MPLS

Internet

Branch 1

expensive, high quality

Branch 2

cheap, low quality, redundant

Inefficient

Costly
What can we do about this?

• Use broadband (BB) link if application QoS requirement permits.
  – E.g. No SLA required for Apple update traffic.

• Broadband link quality varies: use BB link when it is good.
  – E.g. RTT over broadband may be good enough for Skype traffic most of the time.

• Impact: $2.5m over 400 branches.
  – Estimate: 70% of traffic is delay insensitive
  – Saves one T1 per branch
Challenges

- Application identification
  - Convenient to use domain names to identify application flows, but packets contain IP addresses, not domain names.
  - Need to associate domain name context with traffic flows.
  - Manual configuration not possible
    - Hundreds of sites
    - Domain name – IP address associations are dynamic
- Broadband connection quality varies dynamic
  - Need to monitor link qualities and react to changes.
- CE device is non-SDN, e.g. Cisco router.
Our WAN App

- Feature 1: Allow flexible app identification using domain names; controller tracks bindings and installs correct flows.

- Feature 2: Monitoring
  - Dynamically monitor WAN link quality
  - Monitor app SLA

- Feature 3: Constraint-based routing to satisfy SLAs.
  - Load balancing to utilize available capacity

- Feature 4: Incrementally deployable to each branch; replaces primary LAN switch with SDN-capable switch.
SDN Solution Overview

Data Center
+ Incremental
+ Low risk

GUI, hosted in data center

Simple, static policy-based routing config based on DSCP

Branch 1

Branch 2

Interop.com
2014 ONUG Hackathon

• Developed a basic version of our WAN app over two days at ONUG hackathon.
• Day 1:
  – Network simulation environment (DAY 1).
• Day 2:
  – Network controllers with basic domain name policies and dynamic link quality
  – Administrator GUI with link and application monitoring and policy editing.
• Day 3:
  – Demo!
DEMO

WAN Throughput at Branch 1

Application Policies

Policy 1

Policy 2

Link Quality

App Monitoring
WAN Throughput at Branch 1

Application Policies

Policy 1
* ==> HIGH

Policy 2
www.youtube.com ==> LOW,
* ==> HIGH

apply default policies
apply policies
Link Quality

- Broadband Link = HIGH quality
- Broadband Link = LOW quality

App Monitoring

Graph showing traffic patterns from H1 to H5, port 5001 and H2 to H6, port 5001.
Low quality app identification

WAN Throughput at Branch 1

Application Policies

Policy 1
* ==> HIGH

Policy 2
www.youtube.com ==> LOW,
* ==> HIGH

apply default policies
apply policies
Low quality app identification

ONUG Hackathon SDN WAN Optimizer

Policy 2

www.youtube.com => LOW,
* => HIGH

-link Quality

Broadband Link = HIGH quality

Broadband Link = LOW quality

App Monitoring

H1 -> H5, port 5001
H2 -> H6, port 5001
Broadband has high quality

WAN Throughput at Branch 1

Application Policies

Policy 1
* ==> HIGH

Policy 2
www.youtube.com ==> LOW,
* ==> HIGH
Broadband has high quality
How did we build it?

- Virtual Network Environment: Mininet
- SDN (OpenFlow) Controllers: Maple
- Web-based GUI: D3 + JS
Mininet (mininet.org)

• “Mininet creates a **realistic virtual network**, running **real kernel, switch and application code**, on a single machine (VM, cloud or native), in seconds, with a single command.”

• E.g. `sudo mn --switch ovsk --controller ref --topo tree,depth=2,fanout=3`

![Diagram of Mininet network showing hosts and OpenFlow switches](image-url)
Mininet Features

- xterm for each host.
- Convenience functions from mininet CLI.
- Can use remote controller.
- Can use custom topologies.
- Can create links with delay, bandwidth capacity, loss rate, etc.
- Use Python scripts to create custom networks.
- Uses Linux Containers for lightweight virtualization – can simulate network with hundreds of hosts & switches.
Current SDN Controllers

- Beacon
- Ryu
- NOX
- Daylight
- Floodlight
- OpenContrail
Problem with Current SDN Controllers

* Difficult to program, producing buggy programs
* Require assembly-like programming
Big Picture with Current SDN Controllers

User Level

- Make Decisions
- Generate Rules

Under the hood

- OF Controller Library
- OF Switches
SDN Controllers with Maple

User Level
- Make Decisions

Under the hood
- Generate Rules
- OF Controller Library
- OF Switches
Maple: Algorithmic SDN Programming

• Simplifies life: given a packet, what do you want to do with it?

• User defines a **packet-processing function** (not detailed rules) in an ordinary programming language, e.g. Java, Python, Haskell, etc.

• Abstraction: see every packet.

• Supports both algorithmic and declarative policies.

• Compiler & Runtime system automatically generate correct optimized flow rules.
What should our SDN switch do?

2. DNS bindings: sniff DNS queries/answers.
4. Choose up-link: mark high priority packets.
5. Monitor broadband quality: ping to track latency.
public class WANController extends MapleFunction {

    public WANController() { startBroadbandMonitor(); }

    protected Route onPacket(Packet p) {
        macLearn(p);
        sniffDNSQueries(p);
        updateAppCounters(p);
        List<Mod> mods = mods(p);
        return forward(p, mods);
    }
}
1. Monitoring Broadband Link Quality

```java
Variable<Integer> broadbandQuality;

void startBroadbandMonitor() {
    (new Timer()).schedule(pingTask, 10000);
}

TimerTask pingTask = new TimerTask() {
    public void run() {
        double avg = ping(PING_TARGET, NUM_PINGS);
        rtt = ewma(rtt, avg);
        if (rtt < RTT_THRESH) { broadbandQuality.write(1); }
        else { broadbandQuality.write(0); }
    }
}
```

User declares their data model & Maple maintains consistency automatically.
2. Basic MAC Learning

// Declare the mac table.
MapleMap<Long, SwitchPort> hostLocationMap = newMap("hostlocations", Long.class, SwitchPort.class);

// MAC learning
public void macLearn(Packet p) {
    hostLocationMap.put(p.ethSrc(), p.ingressPort());
}
public void sniffDNSQueries(Packet p) {
    if (isDNSResponse(p)) {
        Ethernet frame = p.getFrame();
        IPv4 packet = (IPv4) frame.getPayload();
        UDP segment = (UDP) packet.getPayload();
        Message dnsMessage = ((DNS) segment.getPayload()).message;
        handleDNSResponse(dnsMessage);
    }
}
public void handleDNSResponse(Message dnsMessage) {
    // If an A record:
    // update forward lookup table
    // update reverse lookup table
}

MapleMap<String, Integer> forwardMap =
    newMap("forwardMap", String.class, Integer.class);

MapleMap<Integer, String> reverseMap =
    newMap("reverseMap", Integer.class, String.class);
4. Update App Packet Counters

Counter app1Counter = newCounter("app1");
Counter app2Counter = newCounter("app2");

void updateAppCounters(Packet p) {
    if (isApp1(p)) { app1Counter.count();}
    if (isApp2(p)) { app2Counter.count();}
}
5. App Policy: Determine Packet Marking (1)

```java
List<Mod> mods(Packet p) {
    List<Mod> mods = new LinkedList<Mod>();
    if (!isPingProbe(p) && p.ingressPort().portID != UP_PORT) {
        if (desiredQuality(p) > broadbandQuality.read()) {
            mods.add(setIPTypeOfService(ToS_GOOD));
            highQualCounter.count();
        } else {
            lowQualCounter.count();
        }
    }
    return mods;
}
```

Can modify packets.
int desiredQuality(Packet p) {
    if (p.satisfies(isIPv4())) {
        int dstQual = desiredQuality(p.ipDst());
        int srcQual = desiredQuality(p.ipSrc());
        return Math.min(dstQual, srcQual);
    }
    return 1;
}

int desiredQuality(int ip) {
    String name = reverseMap.get(ip);
    if (null == name) return 1;
    Integer qual = policy.get(name);
    return (null == qual) ? 1 : qual;
}
Route forward(Packet p, List<Mod> mods) {
    List<Link> path;
    SwitchPort dstLoc = hostLocationMap.get(p.ethDst());

    if (null == dstLoc) {
        path = minSpanningTree();
        return route(path, edgePorts(), mods);
    } else {
        path = shortestPath(p.ingressPort(), dstLoc);
        return route(path, dstLoc, mods);
    }
}
Behind the Scenes

• Nontrivial app: ~200 LoC for controller, ~400 LoC for GUI.

• Maple lets you focus on the network logic, not the configuration.

• Maple takes care of:
  – Tracks key network state, such as topology.
  – Generates correct, optimized rules to forward along paths and modify packets.
  – Keep forwarding rules up to date when data model changes.
  – Collects flow counters.
  – Provides HTTP/REST API to data model.
Thank you!

- Come talk to us:
  - david@crosbie.com
  - andreas.voellmy@gmail.com
- Try it out and/or join the development: