S8: Evolving the Data Plane

Instructor:
Behnam Momeni

Fall 2018
Adding New Functionality to the Internet

• Adding new functionality to the data plane
• Future internet architectures
• Overlay networks
• Active networks
• OpenFlow’s next generation → P4
• Reading
  • P4: Programming Protocol-IndependentPacket Processors
  • Active network vision and reality: lessons from a capsule-based system
• Optional
  • XIA: Efficient Support for Evolvable Internetworking
  • A Case for End System Multicast
A History of Internet Evolution

• Many success stories

• 1983 → Flag day switch from NCP to IP
• 1988 → /etc/hosts to DNS
• 1996 → TCP SACK
• 1989-1994 → EGP to BGP [1..4]
A History of Internet Evolution

• But also many failures

• 1986 → IP Multicast
• 1997 → IntServ
• 1998 → DiffServ
• 1995 → IPv6
• Internet capabilities, traceback schemes, explicit congestion control, content-oriented processing, ...

A History of Internet Evolution

- Hard to change IP
  - ...especially after 1990

Applications

IP

Technology

Innovation both above and below IP
Clean-Slate vs. Evolutionary

- Successes of the 80s followed by failures of the 90’s
  - IP Multicast
  - QoS
  - RED (and other AQMs)
  - ECN
  - …
- Concern that Internet research was dead
  - Difficult to deploy new ideas
  - What did catch on was limited by the backward compatibility required
Outline

• Active Networks

• Architectures for Incremental Deployment

• P4

• Overlay Routing
Why Active Networks?

• Traditional networks route packets looking only at destination
  • Also, maybe source fields (e.g. multicast)
  • Problem: Rate of deployment of new protocols and applications is too slow
• Applications that do more than IP forwarding
  • Firewalls
  • Web proxies and caches
  • Transcoding services
  • Nomadic routers (mobile IP)
  • Transport gateways (snoop)
  • Reliable multicast (lightweight multicast, PGM)
Active Networks

- Nodes (routers) receive packets:
  - Perform computation based on their internal state and control information carried in packet
  - Forward zero or more packets to end points depending on result of the computation
- Users and apps can control behavior of the routers
- End result: network services richer than those by the simple IP service model
Variations on Active Networks

• Programmable routers (e.g. OpenFlow)
  • More flexible than current configuration mechanism
  • For use by administrators or privileged users

• Active control (e.g. SDN)
  • Forwarding code remains the same
  • Useful for management/signaling/measurement of traffic

• “Active networks”
  • Computation occurring at the network (IP) layer of the protocol stack → capsule based approach
  • Programming can be done by any user
  • Source of most active debate
Case Study: MIT ANTS System

- Conventional Networks:
  - All routers perform same computation
- Active Networks:
  - Routers have same runtime system
- Tradeoffs between functionality, performance and security
System Components

- Capsules

- Active Nodes:
  - Execute capsules of protocol and maintain protocol state
  - Provide capsule execution API and safety using OS/language techniques

- Code Distribution Mechanism
  - Ensure capsule processing routines automatically/dynamically transfer to node as needed
Capsules

- Each user/flow programs router to handle its own packets
  - Code sent along with packets
  - Code sent by reference
- Protocol:
  - Capsules that share the same processing code
- May share state in the network
- Capsule ID (i.e. name) is MD5 of code
Capsules

- Capsules are forwarded past normal IP routers
Capsules

- When node receives capsule uses “type” to determine code to run
- What if no such code at node?
  - Requests code from “previous address” node
  - Likely to have code since it was recently used
Capsules

- Code is transferred from previous node
  - Size limited to 16KB
  - Code is signed by trusted authority (e.g. IETF) to guarantee reasonable global resource use
Research Questions

• Execution environments
  • What can capsule code access/do?
• Safety, security & resource sharing
  • How isolate capsules from other flows, resources?
• Performance
  • Will active code slow the network?
• Applications
  • What type of applications/protocols does this enable?
Functions Provided to Capsule

• Environment Access
  • Querying node address, time, routing tables

• Capsule Manipulation
  • Access header and payload

• Control Operations
  • Create, forward and suppress capsules
  • How to control creation of new capsules?

• Storage
  • Soft-state cache of app-defined objects
Safety, Resource Mgt, Support

• Safety:
  • Provided by mobile code technology (e.g. Java)

• Resource Management:
  • Node OS monitors capsule resource consumption

• Support:
  • If node doesn’t have capsule code, retrieve from somewhere on path
Applications/Protocols

- Limitations
  - Expressible $\rightarrow$ limited by execution environment
  - Compact $\rightarrow$ less than 16KB
  - Fast $\rightarrow$ aborted if slower than forwarding rate
  - Incremental $\rightarrow$ not all nodes will be active

- Proof by example
  - Host mobility, multicast, path MTU, Web cache routing, etc.
Discussion

• Active nodes present lots of applications with a desirable architecture

• Key questions
  • Is all this necessary at the forwarding level of the network?
  • Is ease of deploying new apps/services and protocols a reality?
Outline

• Active Networks

• Architectures for Incremental Deployment

• P4

• Overlay Routing
Proposed Centric Networking

- Content: Named Data Networking
- Mobility: MobilityFirst
- Cloud: Nebula

Problem: Focusing on one communication type may hinder using other communication types, as occurred to IP

Can we support heterogeneous communication types on a single Internet architecture?
Future-Centric Networking

- Service, content, mobility, and cloud did not receive much attention before.
- Yet more networking styles may be useful in the future.
  - E.g., DTN, wide-area multicast, …?

Problem: Introducing additional communication types to the existing network can be very challenging.

Can we support **future** communication types **without redesigning** the Internet architecture?
Legacy Router May Prevent Innovation

“I got a computer with Awesome-Networking announced at Sigcomm 2022! Can I use it right now?”

“Ouch, we just replaced all of our routers built in 2012. Can you wait for another 10 years for new routers?”

Problem: Using a new communication type may require every legacy router in the network to be upgraded.

Can we allow use of a new communication type even when the network is yet to natively support it?
XIA’s Goals and Design Pillars

“Principal types”
- Support multiple communication types concurrently (heterogeneity)
- Support future communication types (flexibility)

“Fallbacks”
- Allow using new communication types at any point (incremental deployment)
Principal Types

Define your own communication model
# Principals

## Current Internet

- **IP address**: 128.2.10.162

## XIA

<table>
<thead>
<tr>
<th>Principal type</th>
<th>Type-specific identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>0xF63C7A4…</td>
</tr>
<tr>
<td>Service</td>
<td>0x8A37037…</td>
</tr>
<tr>
<td>Content</td>
<td>0x47BF217…</td>
</tr>
<tr>
<td>Future</td>
<td>...</td>
</tr>
</tbody>
</table>

- Hash of host’s public key
- Hash of service’s public key
- Hash of content
Principal Definition 1: Address Allocation and Intrinsic Security

- XIA uses self-certifying identifiers that guarantee security properties for communication operation
  - Host ID is a hash of its public key – accountability (AIP)
  - Content ID is a hash of the content – correctness
  - Does not rely on external configuration

- Intrinsic security is specific to the principal type

- Example: retrieve content using …
  - Content XID: content is correct
  - Service XID: the right service provided content
  - Host XID: content was delivered from right host
Principal Definition 2: Type-Specific Semantics

- **Contact a host**
  - Host: 0xF63C7A4...

- **Use a service**
  - Service: 0x8A37037...

- **Retrieve content**
  - Content: 0x47BF217...
Principal Definition 3: Type-Specific Processing

- **Type-specific processing examples**
  - Service: load balancing or service migration
  - Content: content caching
Routers with Different Capabilities

• Routers are **not** required to support every principal type
  • The only requirement: Host-based communication

- **Host-only router**
  - Common
  - Host

- **Service-enabled router**
  - Common
  - Host
  - Service

- **Content-enabled router**
  - Common
  - Host
  - Content
Using Principal Types that are Not Understood by Legacy Routers?

Want to communicate using content principals

Content-enabled router

Legacy router without content support

Content-enabled router
Fallbacks

Tomorrow’s communication types… today!
Fallbacks: Alternative Ways for Routers to Fulfill Intent of Packet

Intent: Retrieve Content

Fallback: Contact Host, who understands Content request

What the network does:

- With content-enabled routers, use Content for routing
- Otherwise, use Host for routing (always succeeds)
DAG-Based Address

Your address is more than a number
DAG (Direct Acyclic Graph)-Based Addressing Enables Fallbacks

Packet sender → Routing choice → Host

Another routing choice (with lower priority)

This host knows how to handle content request

Intent

Content

Fallback
Outline

- Active Networks
- Architectures for Incremental Deployment
- P4
- Overlay Routing
In the Beginning…

• OpenFlow was simple

• A single rule table
  • Priority, pattern, actions, counters, timeouts

• Matching on any of 12 fields, e.g.,
  • MAC addresses
  • IP addresses
  • Transport protocol
  • Transport port numbers
Over the Past Five Years…

Proliferation of header fields

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th># Headers</th>
</tr>
</thead>
<tbody>
<tr>
<td>OF 1.0</td>
<td>Dec 2009</td>
<td>12</td>
</tr>
<tr>
<td>OF 1.1</td>
<td>Feb 2011</td>
<td>15</td>
</tr>
<tr>
<td>OF 1.2</td>
<td>Dec 2011</td>
<td>36</td>
</tr>
<tr>
<td>OF 1.3</td>
<td>Jun 2012</td>
<td>40</td>
</tr>
<tr>
<td>OF 1.4</td>
<td>Oct 2013</td>
<td>41</td>
</tr>
</tbody>
</table>

Multiple stages of heterogeneous tables

Still not enough (e.g., VXLAN, NVGRE, STT, …)
“Classic” OpenFlow (1.x)

SDN Control Plane

Installing and querying rules

Target Switch
“OpenFlow 2.0”

SDN Control Plane

- **Configuring:** Parser, tables, and control flow
- **Populating:** Installing and querying rules

Compiler

- **Parser & Table Configuration**
- **Rule Translator**

Target Switch
Abstract Model

- Reconfigurable parser + Non-sequential match-action + Protocol independent processing

---

**Figure 2: The abstract forwarding model.**

- Switch Configuration:
  - Parse Graph
  - Control Program
  - Table Config
  - Action Set

- Forwarding rules:
  - Ingress Pipeline: Packet Mods + Egress Selection
  - Egress Pipeline: Packet Mods

---

The forwarding model is controlled by two types of operations: Configure and Populate. These operations add and remove entries to the match action tables that process these headers. The switch should not be tied to specific packet formats. Instead, the controller should define the packet parsing and processing in the field.

Protocol independence is supported by the switch. The controller should be able to reconfigure the parser, set the order of match stages, and add actions. Reconfiguration determines which protocols are supported and how new headers are defined. Our model deliberately allows processing during partial or full reconfiguration enabling upgrades with no downtime.

For the purposes of this paper, we assume that configuration and population are two distinct phases. In particular, the switch may process packets.

The controller programmer should not need to know the specifics of the underlying CPU, the function ASICs, NPUs, reconfigurable switches, soft-routers. The key generalization is that the switch need not process packets during configuration. The switch should not be tied to specific packet formats. Instead, the controller should define the packet parsing and processing in the field. Just as a C programmer does, the controller should be able to reconfigure the parser, set the order of match stages, and add actions.

Clearly, the configuration phase has little meaning in fixed-function ASIC switches. For this type of switch, the controller should not need to know the specifics of the underlying CPU, the function ASICs, NPUs, reconfigurable switches, soft-routers.
Simple Motivating Example

- Data-center routing
  - Top-of-rack switches
  - Two tiers of core switches
  - Source routing by ToR

- Hierarchical tag (mTag)
  - Pushed by the ToR
  - Four one-byte fields
  - Two hops up, two down
Header Formats

- Header
  - Ordered list of fields
  - A field has a name and width

```plaintext
header ethernet {
    fields {
        dst_addr : 48;
        src_addr : 48;
        ethertype : 16;
    }
}

header vlan {
    fields {
        pcp : 3;
        cfi : 1;
        vid : 12;
        ethertype : 16;
    }
}

header mTag {
    fields {
        up1 : 8;
        up2 : 8;
        down1 : 8;
        down2 : 8;
        ethertype : 16;
    }
}
```
• State machine traversing the packet
  • Extracting field values as it goes

```c
parser start {
    ethernet;
}

parser ethernet {
    switch(ethertype) {
        case 0x8100 : vlan;
        case 0x9100 : vlan;
        case 0x8000 : ipv4;
        . . .
    }
}

parser vlan {
    switch(ethertype) {
        case 0xaaaa : mTag;
        case 0x800 : ipv4;
        . . .
    }
}

parser mTag {
    switch(ethertype) {
        case 0x800 : ipv4;
        . . .
    }
}
```
Typed Tables

- Describe each packet-processing stage
  - What fields are matched, and in what way
  - What action functions are performed
  - (Optionally) a hint about max number of rules

```plaintext
table mTag_table {
  reads {
    ethernet.dst_addr : exact;
    vlan.vid : exact;
  }
  actions {
    add_mTag;
  }
  max_size : 20000;
}
```
Action Functions

- Custom actions built from primitives
  - Add, remove, copy, set, increment, checksum

```c
action add_mTag(up1, up2, down1, down2, outport) {
    add_header(mTag);
    copy_field(mTag.ethertype, vlan.ethertype);
    set_field(vlan.ethertype, 0xaaaa);
    set_field(mTag.up1, up1);
    set_field(mTag.up2, up2);
    set_field(mTag.down1, down1);
    set_field(mTag.down2, down2);
    set_field(metadata.outport, outport);
}
```
Control Flow

- Flow of control from one table to the next
- Collection of functions, conditionals, and tables
- For a ToR switch:

From **core** (with mTag)

ToR

From local **hosts** (with no mTag)

- Source Check Table
- Local Switching Table
- Egress Check

- mTag Table

Miss: Not Local
Control Flow

- Flow of control from one table to the next
  - Collection of functions, conditionals, and tables
- Simple imperative representation

```c
control main() {
    table(source_check);

    if (!defined(metadata.ingress_error)) {
        table(local_switching);

        if (!defined(metadata.outport)) {
            table(mTag_table);
        }
    }

    table(egress_check);
}
```
Outline

• Active Networks

• Architectures for Incremental Deployment

• P4

• Overlay Routing
Overlay Routing

- **Basic idea:**
  - Treat multiple hops through IP network as one hop in “virtual” overlay network
  - Run routing protocol on overlay nodes
  - Basic technique: tunnel packets

- **Tunnels**
  - IP-in-IP encapsulation
  - Poor interaction with firewalls, multi-path routers, etc.

- **Why?**
  - For performance – can run more clever protocol on overlay
  - For functionality – can provide new features such as multicast, active processing, IPv6
Overlay for Performance [S+99]

• Why would IP routing not give good performance?
  • Policy routing – limits selection/advertisement of routes
  • Early exit/hot-potato routing – local not global incentives
  • Lack of performance based metrics – AS hop count is the wide area metric

• How bad is it really?
  • Look at performance gain an overlay provides
Quantifying Performance Loss

• Measure round trip time (RTT) and loss rate between pairs of hosts
  • ICMP rate limiting

• Alternate path characteristics
  • 30-55% of hosts had lower latency
  • 10% of alternate routes have 50% lower latency
  • 75-85% have lower loss rates
Overlay for Features

• How do we add new features to the network?
  • Does every router need to support new feature?
  • Choices
    • Reprogram all routers → active networks
    • Support new feature within an overlay

• Examples:
  • IP V6 & IP Multicast
    • Tunnels between routers supporting feature
  • Mobile IP
    • Home agent tunnels packets to mobile host’s location
Overlay Example: Multicast

- Unicast: one source to one destination
- Multicast: one source to many destinations
- Two main functions:
  - Efficient data distribution
  - Logical naming of a group
Example Applications

- Broadcast audio/video
- Push-based systems
- Software distribution
- Web-cache updates
- Teleconferencing (audio, video, shared whiteboard, text editor)
- Multi-player games
- Server/service location
- Other distributed applications
Multicast – Efficient Data Distribution
Multicast Router Responsibilities

- Learn of the existence of multicast groups (through advertisement)
- Identify links with group members
- Establish state to route packets
  - Replicate packets on appropriate interfaces
  - Routing entry:

| Src, incoming interface | List of outgoing interfaces |
At which layer should multicast be implemented?
IP Multicast

- Highly efficient
- Good delay
End System Multicast

Overlay Tree

UCSD

Berkeley

MIT1

MIT2

CMU1

CMU2

UCSD

Berkeley

MIT1

MIT2

CMU1

CMU2
Potential Benefits Over IP Multicast

- Quick deployment
- All multicast state in end systems
- Computation at forwarding points simplifies support for higher level functionality
Concerns with End System Multicast

• Self-organize recipients into multicast delivery overlay tree
  • Must be closely matched to real network topology to be efficient

• Performance concerns compared to IP Multicast
  • Increase in delay
  • Bandwidth waste (packet duplication)
Next Lecture

• Middleboxes
• NFV
• Readings:
  • Network Functions Virtualisation (skim)
  • Design and Implementation of a Consolidated Middlebox Architecture (read whole paper)