Designing Extensible IP Router Software

Mark Handley
Department of Computer Science
UCL
Networking research: divorced from reality?

- Gap between research and practice in routing and forwarding.

- Most of the important Internet protocols originated in research, often at universities.
  - It used to be that researchers designed systems, built implementations, tried them out, and standardized the ones that survived and proved useful.
  - What happened?
Networking research: why the divorce?

*The commercial Internet:*
- Network stability is critical, so experimentation is difficult.
- Major infrastructure vendors not motivated to support experimentation.

*Network simulators:*
- High-quality simulators make a lingua franca for research.
Simulation is not a substitute for experimentation

- Many questions require real-world traffic and/or routing information.

- Most grad students:
  - Give up, implement their protocol in ns
  - Set ns parameters based on guesses, existing scripts
  - Write a paper that may or may not bear any relationship to reality
  - We need to be able to run experiments when required!
The state of the art

- Open APIs facilitate end-system protocol development:
  - WWW, RTP, SIP, RTSP, ...

- Open-source OSes do the same for kernel changes.
  - TCP SACK, IGMPv3, ...
  - Also a history of experimentation in commercial OSes (affiliated labs)
What about protocols that affect the routers?

Option 1:
1. Persuade Cisco to implement your protocol;
2. Persuade ISPs that your protocol won't destabilize their networks;
3. Conduct experiment.

Option 2:
1. Implement routing protocol part in MRTd, GateD, or Quagga;
2. Implement forwarding part in FreeBSD, Linux, Click, Scout, ...
3. Persuade network operators to replace their Ciscos with your PC;
4. Conduct experiment.
Likelihood of success?
Possible solutions

Solution 1:
A router vendor opens their development environment and APIs. Thus, new router applications can be written and deployed by third parties. Basic router functionality cannot be changed.

Solution 2:
Someone (hint, hint) builds a complete open-source router software stack explicit designed for extensibility and robustness. Adventurous network operators deploy this router on their networks; it develops a reputation for stability and configurability. Result: a fully extensible platform suitable for research and deployment.
Stalled Evolution in the Internet Core

*Stability matters above all else.*

- ISPs can’t afford routing problems.
- Won’t run anything experimental on production routers for fear of affecting production traffic.

Building stable routing protocol implementations is really hard.

- Big router vendors don’t have a lot to gain from innovation, because ISPs *can’t afford to run experimental code*.
- Startups can’t help because of the *closed nature of the software market for IP routers*.

*Important routing problems remain unaddressed.*
Extensible Router Control Plane Software

Extensibility could solve this problem:
- Allow experimental deployment of new routing protocols
- Enable router application market

Extensible forwarding planes exist:
- Network Processors, FPGAs, software (Click, Scout, ...)
- But not control planes: why?

The demands of router software make extensibility hard:
- Stability requirement
- Massive scale distributed computation
- Tight coupling between functions
- Routing protocols themselves not designed for extensibility
Four Challenges

**Features**
Real-world routers must support a long feature list.

**Extensibility**
Every aspect of the router should be extensible.
Extensions must be able to co-exist gracefully.

**Performance**
Not core routers (yet), but edge-routing is hard enough.
Scalability in routing table size, low routing latency.

**Robustness**
Must not crash or misroute packets.
Fast Convergence

Routing protocol implementations have often been *scanner-based*. □ Periodically a scanner runs to accumulate changes, update the forwarding table, notify neighbors, etc. □ Easy to implement. □ Low CPU utilization. □ Poor route convergence properties.

*Fast convergence is now a priority.* □ Event-driven router implementations are needed to respond to change as quickly as possible. □ Events processed to completion. □ Explicit dependency tracking. □ *Harder to implement, especially in an extensible way.*
Open source router software suite, *designed from the outset with extensibility in mind.*

- Main core unicast and multicast routing protocols.
- Event-driven multi-process architecture.
- BSD-style license
- 560,000 lines of C++
Contributions

- Staged design for BGP, RIB.
- Scriptable inter-process communication mechanism.
- Dynamically extensible command-line interface and router management software.
- Extensible policy framework.

*First fully extensible, event-driven, open-source routing protocol suite: [www.xorp.org](http://www.xorp.org).*
XORP Processes

Multi-process architecture, providing isolation boundaries between separate functional elements.

Flexible IPC interface between modules

RIB = routing information base
FEA = forwarding engine abstraction

Forwarding Engine
Click Elements

Multicast Routing

Unicast Routing
Outline of this talk

1. Routing process design
2. Extensible management framework
3. Extensible policy routing framework
4. Performance results

RIB = routing information base
FEA = forwarding engine abstraction

Multicast Routing

PIM–SM
IGMP/MLD

Management Processes

SNMP

Forwarding Engine

Click Elements
Routing Process Design

- How do you implement routing protocols in such a way that they can easily be extended in the future?
Conventional router implementation
Implementing for Extensibility

- Tightly coupled architectures perform well, but are extremely hard to change without understanding how all the features interact.

- Need an architecture that permits future extension, while minimizing the need to understand all the other possible extensions that might have been added.
  - We chose a data-flow architecture.
  - Routing tables are composed of dynamic processes through which routes flow.
  - Each stage implements a common simple interface.
BGP Staged Architecture

Routes from BGP Peers

Peer In → Filter Bank → Decision Process

Best Routes to Peers

Filter Bank → Peer Out

Peer In

Filter Bank

IGP routing information from RIB

Best routes to RIB

Filter Bank

Peer Out

Routes to BGP Peers
Messages

Unmodified routes stored at ingress
Changes in downstream modules (filters, nexthop state, etc) handled by PeerIn pushing the routes again.
BGP Staged Architecture
Decomposing BGP Decision

IGP routing information from RIB

Best routes to RIB

Best Routes to Peers

Peer In ▲

Filter Bank

Nexthop Resolver

Decision Process

Fanout Queue

Filter Bank

Peer Out

Filter Bank

Peer Out

Peer Out

Peer Out

Peer Out

Peer Out

Peer In ▲

Filter Bank

Nexthop Resolver

Nexthop Resolver

Nexthop Resolver

Peer In ▲

Filter Bank

Nexthop Resolver

Nexthop Lookup
Dynamic Stages

Problem 1: deleting 150,000 routes takes a long time.

Problem 2: peering may come up again while we’re still deleting the routes.

PeerIn is ready for peering to come back up.

PeerIn Went Down!

Deletion Stage does background deletion of routes.
Routing protocols can register interest in tracking changes to specific routes.
How do you implement a single unified router management framework and command line interface, when you don’t know what protocols are going to be managed?

Want to enable integration of future protocols from third party vendors, without having to change existing core XORP code.

Inter-process Communication

**XORP Resource Locators (XRLs):**

- URL-like unified structure for inter-process communication:
  
  **Example:**

  finder://bgp/bgp/1.0/set_bgp_as?as:u32=1777

  Finder resolves to a concrete method instance, instantiates transport, and performs access control.

  xtcp://192.1.2.3:8765/bgp/1.0/set_bgp_as?as:u32=1777
Inter-process Communication

- XRLs support extensibility by allowing “non-native” mechanisms to be accessed by unmodified XORP processes.
  - Add new XRL protocol families: eg kill, SNMP

- ASCII canonical representation means XRL can be scripted from python, perl, bash, etc.
  - XORP test harnesses built this way.
XORP Router Manager Process

- The XORP router manager is dynamically extensible using declarative ASCII template files linking configuration state to the XRLs needed to instantiate that configuration.
Router Manager template files
Map Juniper-style configuration state to XRLs

```
protocols ospf {
    router-id: 128.16.64.1
    area 128.16.0.1 {
        interface xl0 {
            hello-interval: 30
        } } }

protocols.ospf {
    area @: ipv4 {
        interface @: txt {
            hello-interval: u32 {
            } } }
```
Routing Policy

- How do you implement a routing policy framework in an extensible unified manner, when you don’t know what future routing protocols will look like?
AS-level Topology 2003
Source: CAIDA
Inter-domain Routing

Tier-1 ISPs

Tier-2 ISPs

AS 1

AS 2

AS 3

AS 4

AS 5

AS 6

AS 7

AS 8

AS 9

AS 10

Net: 128.16.0.0/16

ASPath: 5,2,1,3,6

Net 128.16.0.0/16 ASPath: 5,2,1,3,6

Tier-3 ISPs and Big Customers
Inter-domain Routing

Tier-1 ISPs

AS 1

Tier-2 ISPs

AS 3

AS 6

AS 7

Net: 128.16.0.0/16

AS 8

AS 9

AS 10

AS 2

AS 5

AS 4

Tier-3 ISPs and Big Customers

Route would Loop
Inter-domain Routing

Tier-1 ISPs

AS 1

AS 2

AS 3

AS 4

AS 5

AS 6

AS 7

AS 8

AS 9

AS 10

Tier-2 ISPs

Net: 128.16.0.0/16

Tier-3 ISPs and Big Customers

1,3,6

2,1,3,6

Prefer shortest AS path

Net: 128.16.0.0/16
Inter-domain Routing Policy

- Tier-1 ISPs
- Tier-2 ISPs
- Tier-3 ISPs and Big Customers

Net: 128.16.0.0/16

Only accept customer routes
Inter-domain Routing Policy

Net: 128.16.0.0/16

Don’t export provider routes to a provider

Tier-1 ISPs

Tier-2 ISPs

AS 1

AS 2

AS 3

AS 4

AS 5

AS 6

AS 7

AS 8

AS 9

AS 10

Tier-3 ISPs and Big Customers
Inter-domain Routing Policy

- Tier-1 ISPs
- Tier-2 ISPs
- Tier-3 ISPs and Big Customers

Net: 128.16.0.0/16

Prefer customer routes
Examples of Policy Filters

*Import filters:*
- Drop incoming BGP routes whose AS Path contains AS 1234.
- Set a LOCALPREF of 3 on incoming BGP routes that have a nexthop of 128.16.64.1

*Export filters:*
- Don’t export routes with BGP community \( xyz \) to BGP peer 128.16.64.1
- Redistribute OSPF routes from OSPF area 10.0.0.1 to BGP and set a BGP MED of 1 on these routes.
Where to apply filters?

Flow of incoming routes:

Originated ———> Accepted ———> Winner
Where to apply filters?

Flow of outgoing routes:

RIB

decision

winner

Post

redistributed routes

Routing Protocol

ready

Routing Protocol

ready

winner

ready
Filter Banks

Set a LOCALPREF of 3 on incoming BGP routes that have a nexthop of 128.16.64.1
Redistribute OSPF routes from OSPF area 10.0.0.1 to BGP and set a BGP MED of 1 on these routes.

```
1. routing protocol
   └── RIB
      └── same protocol

2. export: source match

3. export: redistribute selected

4. export: dest match, action
```
Redistribute OSPF routes from OSPF area 10.0.0.1 to BGP and set a BGP MED of 1 on these routes.
Policy Manager Engine

- Takes a complete routing policy for the router:
  1. Parses it into parts (1), (2), (3), and (4) for each protocol.
  2. Checks the route attribute types against a dynamically loaded set of route attributes for each protocol.

    - **bgp**
      - **aspath**: str, rw
      - **origin**: u32, r
      - **med**: u32, rw
    - **rip**
      - **network4**: ipv4net, r
      - **nexthop4**: ipv4, rw
      - **metric**: u32, rw

  3. Writes a simple stack machine program for each filter, and configures the filter banks in each protocol.
All filters use the same stack machine. Stack machine requests attributes by name from routing filter.

Protocol implementer only needs to write the protocol-specific reader and writer.
Summary: Design Contributions

- Staged design for BGP, RIB.
- Scriptable XRL inter-process communication mechanism.
- Dynamically extensible command-line interface and router management software.
- Extensible policy framework.
Evaluation

- Was performance compromised for extensibility?
Performance:
Time from received by BGP to installed in kernel

From BGP to Kernel

No routes
Full Feed (same peer)
Full Feed (different peer)
Performance:  
Where is the time spent?
Performance:
Time from received by BGP until new route chosen and sent to BGP peer
Current Status

Functionality:

Stable: BGP, RIPv2, RIPng, PIM-SM, IGMPv2, MLDv1, RIB, XRLs, router manager, xorp command shell.

In progress: OSPF, policy framework, IS-IS.

Supported Platforms:

Stable: FreeBSD, OpenBSD, NetBSD, MacOS, Linux.

Summary

- Designing for extensibility is difficult
  - Needs to be a priority from day one.
  - Anecdotal evidence shows XORP to be easy to extend (once you’ve learned the basics)

- Performance always at odds with modularity and extensibility.
  - For routing software, scalability matters most.
  - Modern CPUs change the landscape, and make better solutions possible.

- Only time will tell if XORP is adopted widely enough to change the way Internet protocols are *developed, tested,* and *deployed.*
Thanks

*XORP developers (past and present)*:

- Adam Barr, Fred Bauer, Andrea Bittau, Javier Cardona, **Atanu Ghosh**, Adam Greenhalgh, Tim Griffin, **Mark Handley**, Orion Hodson, Eddie Kohler, Luigi Rizzo, **Bruce M. Simpson**, **Pavlin Radoslavov**, Marko Zec

*Financial support for XORP has come from*:

- ICSI, Berkeley
- Intel Corporation
- National Science Foundation
- DARPA
- Microsoft Corporation
- The Royal Society
Registering Interest in Routes

Routes in RIB:

- 128.16.0.0/18
- 128.16.128.0/17
- 128.16.192.0/18

BGP

- Interested in 128.16.32.1
- Interested in 128.16.160.1