XORP: An eXtensible Open Router Platform

Atanu Ghosh Mark Handley Orion Hodson
Eddie Kohler **Pavlin Radoslavov**
International Computer Science Institute

Adam Greenhalgh
University College London

Luigi Rizzo
University of Pisa

Outline

1. Motivations

2. XORP introduction

3. XORP IPC mechanism

4. What does it take to implement a routing protocol?

5. Dependency tracking mechanism

6. Conclusions

Networking research: divorced from reality?

- Gap between research and practice
- Most of the important Internet protocols originated in research
- It used to be that researchers designed systems, *build 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
  - Nice tool, but usually too abstract from reality

Simulation is not a substitute for experimentation

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

- Many people:
  - 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!

Options

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

Options (cont.)

- Option 2:
  - Implement routing protocol part in MRTd, GateD, or Zebra;
  - Implement forwarding part in FreeBSD, Linux, Click, etc;
  - Persuade network operators to replace their Ciscos with your PC;
  - Conduct experiment.

Likelihood of success?

Possible solutions

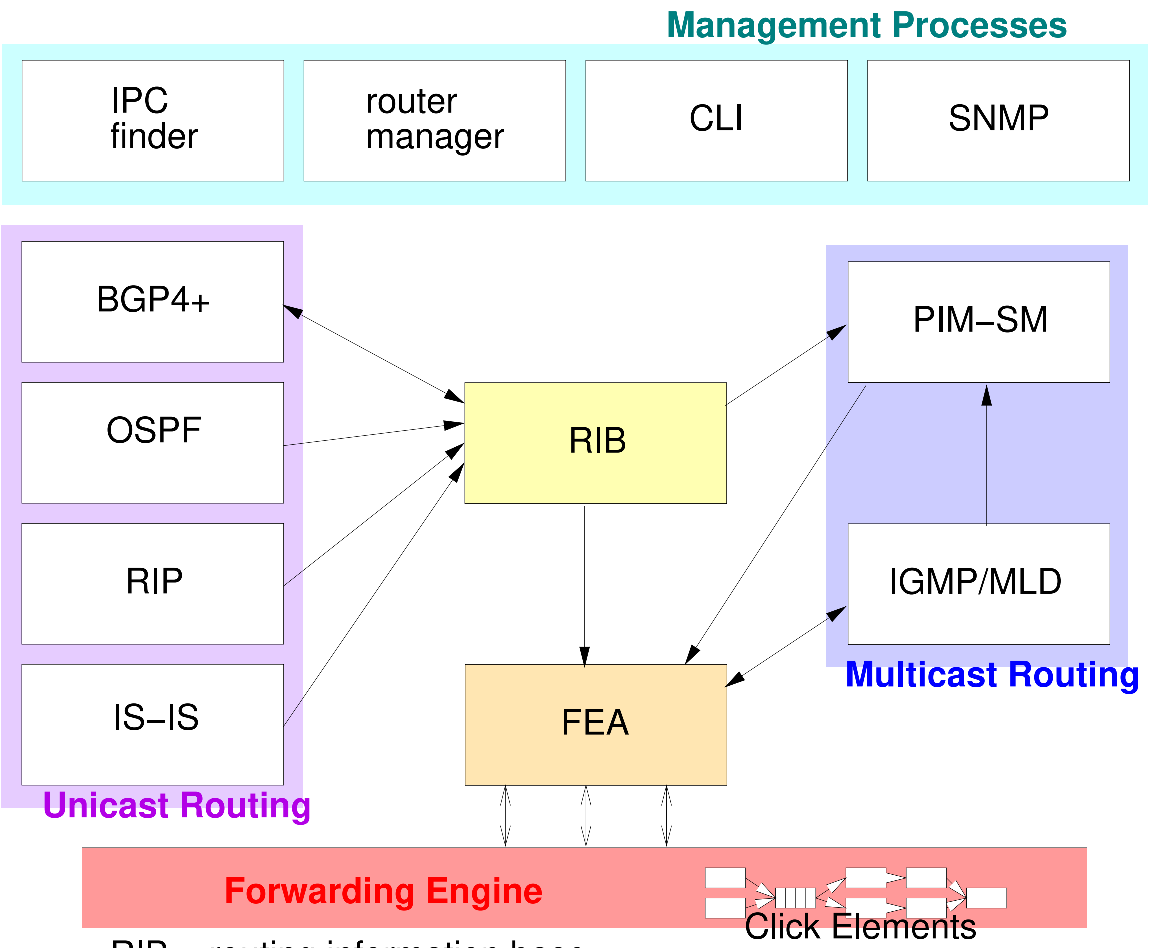
- Solution 1: A router vendor opens their development environment and APIs:
  - Third-party router applications
  - Basic router functionality cannot be changed
- Solution 2: Someone (*hint, hint*) builds a complete open-source router software stack explicitly designed for **extensibility** and **robustness**:
  - Adventurous network operators deploy this router on their networks
  - Result: a fully extensible platform suitable for **research** and **deployment**

XORP: eXtensible Open Router Platform

Complete software stack for an IP router:

- Routing protocols: unicast and multicast
  - Protocols can be run in simulation-like environment
- Management Interfaces
- Forwarding path

XORP Architecture



RIB = routing information base
FEA = forwarding engine abstraction

Challenges

- **Features**: real-world routers support a long feature list
- **Extensibility**:
  - Every aspect of the router should be extensible
  - Multiple extensions should be able to coexist
- **Performance**: raw forwarding performance; routing table size (not core routers; even edge routing is hard enough)
- **Robustness**: must not crash or misroute packets

XORP Features

- IPv4 and IPv6

- Unicast routing protocols: BGP4+, OSPF, RIPv2/RIPng, IS-IS

- Multicast: PIM-SM/SSM, IGMPv1,2,3/MLDv1,2

- DHCP, PPP

- Management: CLI, SNMP, WWW

- Forwarding path: UNIX (native), Click

Extensibility: Intra-router APIs

Separate abstract request (API) from concrete request (which process? which arguments? which version?)

In particular, the caller:

- Should not care about IPC mechanism
- Should not know in advance which process is relevant
  . . . unless required

Extensibility: XRLs (XORP Resource Locators)

XORP IPC mechanism (like URLs for IPC):

`finder://fea/fea/1.0/add_address4?vif:txt=fxp0&addr:ipv4=10.0.0.1`

- Library marshals arguments, implements transport, handles responses
- Redirection into a single XRL or an XRL sequence
- Programmer explicitly handles failure

Extensibility: XRLs (XORP Resource Locators)

XORP IPC mechanism (like URLs for IPC):

`finder://fea/fea/1.0/add_address4?vif:txt=fxp0&addr:ipv4=10.0.0.1`

IPC mechanism: `finder`, `xudp`, `snmp`, . . .

- Library marshals arguments, implements transport, handles responses
- Redirection into a single XRL or an XRL sequence
- Programmer explicitly handles failure

Extensibility: XRLs (XORP Resource Locators)

XORP IPC mechanism (like URLs for IPC):

`finder://fea/fea/1.0/add_address4?vif:txt=fxp0&addr:ipv4=10.0.0.1`

Module/process name: `fea`, `rib`, `bgp`, ...

- Library marshals arguments, implements transport, handles responses

- Redirection into a single XRL or an XRL sequence

- Programmer explicitly handles failure

Extensibility: XRLs (XORP Resource Locators)

XORP IPC mechanism (like URLs for IPC):

`finder://fea/fea/1.0/add_address4?vif:txt=fxp0&addr:ipv4=10.0.0.1`

Interface name: `fea`, `routing-process`, ...

- Library marshals arguments, implements transport, handles responses
- Redirection into a single XRL or an XRL sequence
- Programmer explicitly handles failure

Extensibility: XRLs (XORP Resource Locators)

XORP IPC mechanism (like URLs for IPC):

`finder://fea/fea/1.0/add_address4?vif:txt=fxp0&addr:ipv4=10.0.0.1`

Version number

- Library marshals arguments, implements transport, handles responses

- Redirection into a single XRL or an XRL sequence

- Programmer explicitly handles failure

Extensibility: XRLs (XORP Resource Locators)

XORP IPC mechanism (like URLs for IPC):

```
finder://fea/fea/1.0/add_address4?vif:txt=fxp0&addr:ipv4=10.0.0.1
```

Method name: `delete_address4`, `get_mtu`, ...

- Library marshals arguments, implements transport, handles responses
- Redirection into a single XRL or an XRL sequence
- Programmer explicitly handles failure

Extensibility: XRLs (XORP Resource Locators)

XORP IPC mechanism (like URLs for IPC):

`finder://fea/fea/1.0/add_address4?vif:txt=fxp0&addr:ipv4=10.0.0.1`

Arguments

- Library marshals arguments, implements transport, handles responses
- Redirection into a single XRL or an XRL sequence
- Programmer explicitly handles failure

Defining XRL interface

XRL interface is defined in XRL-specific files:

```
interface pim/0.1 {
/**
 * Enable a PIM virtual interface.
 *
 * @param vif_name the name of the vif to enable.
 * @param fail true if failure has occurred.
 * @param reason contains failure reason if it occurred.
 */
enable_vif ? vif_name:txt -> fail:bool & reason:txt

...
}
```

Using XRLs: C++

All header files are auto-generated; developer implements only XRL handlers:

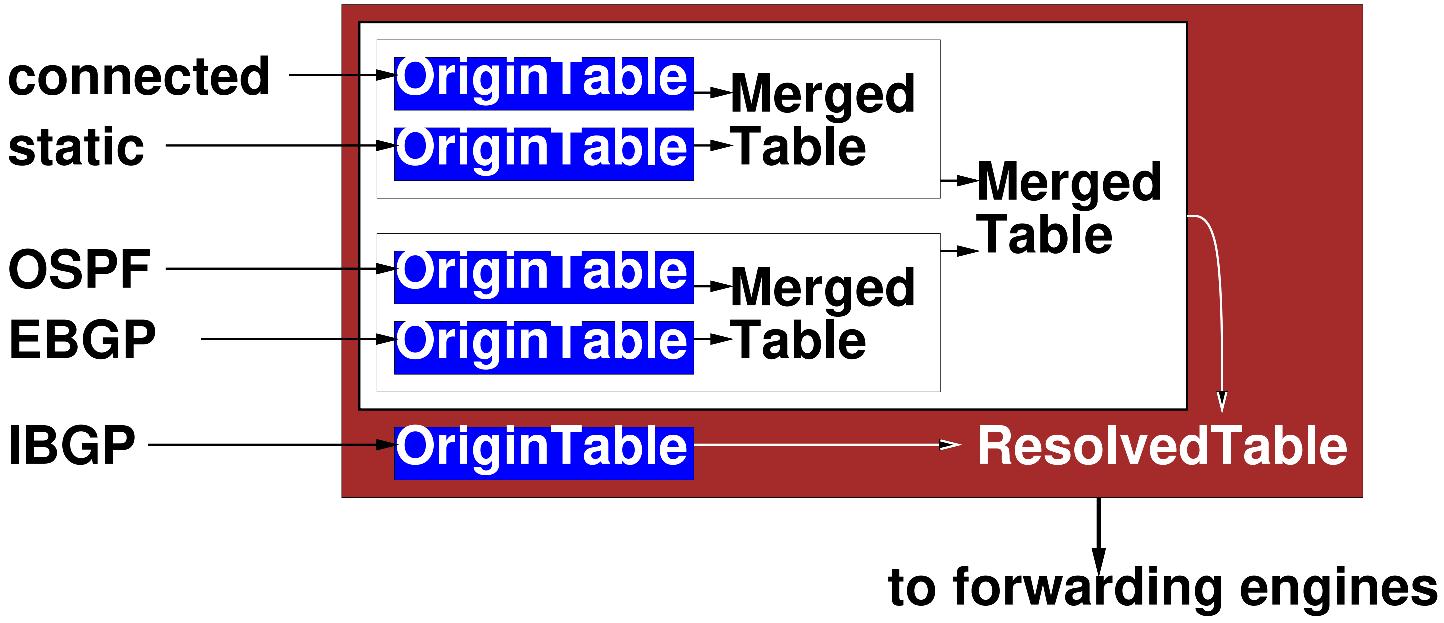
```
XrlCmdError XrlPimNode::pim_0_1_enable_vif(
    // Input values,
    const string&        vif_name,
    // Output values,
    bool&                fail,
    string&              reason)
{
    fail = enable_vif(vif_name, reason);
    return XrlCmdError::OKAY();
}
```

Using XRLs: Shell Script

Everything is ASCII text:

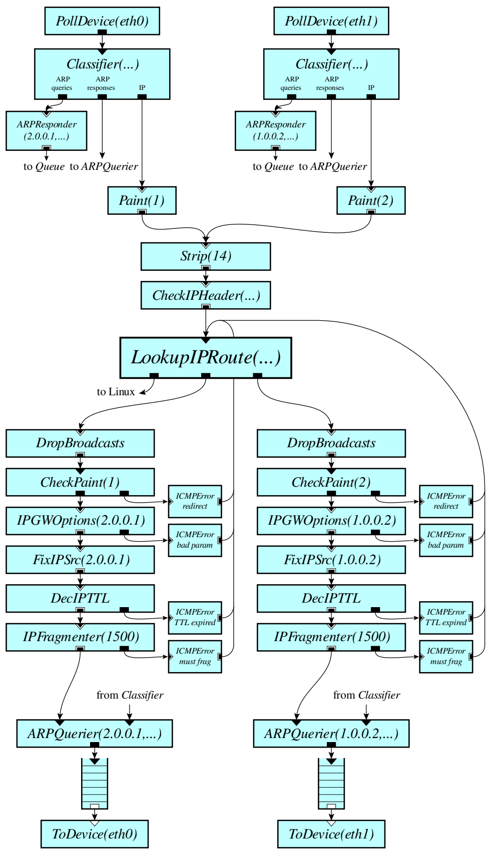
```
pim_enable_vif()
{
    vif_name=$1
    XRL="finder://$PIM_TARGET/pim/0.1/enable_vif"
    XRL_ARGS="?vif_name:txt=$vif_name"
    call_xrl $XRL$XRL_ARGS
}
```

Extensibility: RIB



- Object-oriented routing table design
- Add new merged tables implementing new merging policies, . . .

Extensibility/performance: Click forwarding path



Fast kernel forwarding; easy to write extensions

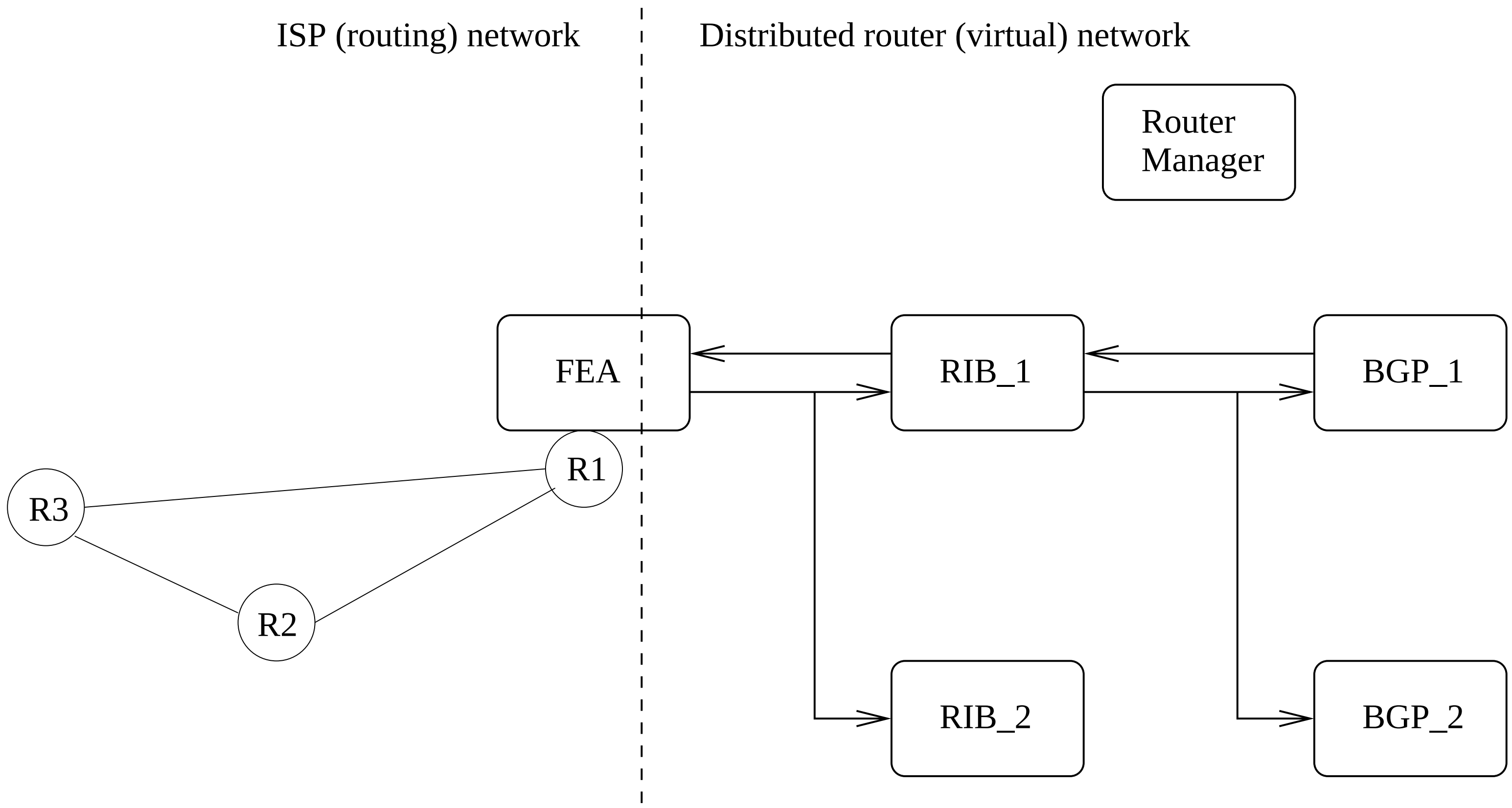
Robustness

- Policy decision: Strong robustness for user-level processes
  - Difficult to get performance, robustness, and extensibility simultaneously
- Facilitated by multi-process design
  - Automatically restart processes that crash
- XRL sandboxes
  - All interaction with router through XRLs
  - Redirect XRLs to run new protocols in a sandbox

Improving robustness and performance: distributed router

- XRLs can be sent across network
- Each routing process can run on a separate machine
- Only the FEA must run on the machine with the forwarding engine:
  - The memory and the CPU are not the bottleneck
  - Improved robustness through hot-swapping of routing modules

Example of a distributed router



Distributed router (cont.)

- The Router Manager coordinates the modules and the interaction among them.

- A routing protocol instance doesn't care whether it is part of a distributed router, or whether it is running as a backup

- Potential issues:
  - Communication latency
  - Bandwidth overhead
  - Synchronization

What does it take to implement a routing protocol?

PIM-SM (Protocol Independent Multicast-Sparse Mode): case-study

- Fairly complicated protocol (protocol specification is 100 + 25 pages), full of tiny details:
  - Early specifications (two RFCs) easy to read, difficult to decode and implement
  - Lastest spec is much more "implementor-friendly"
- Lots of routing state and state dependency

0. Get yourself into the right mindset

Think SIMPLICITY and CONSISTENCY:

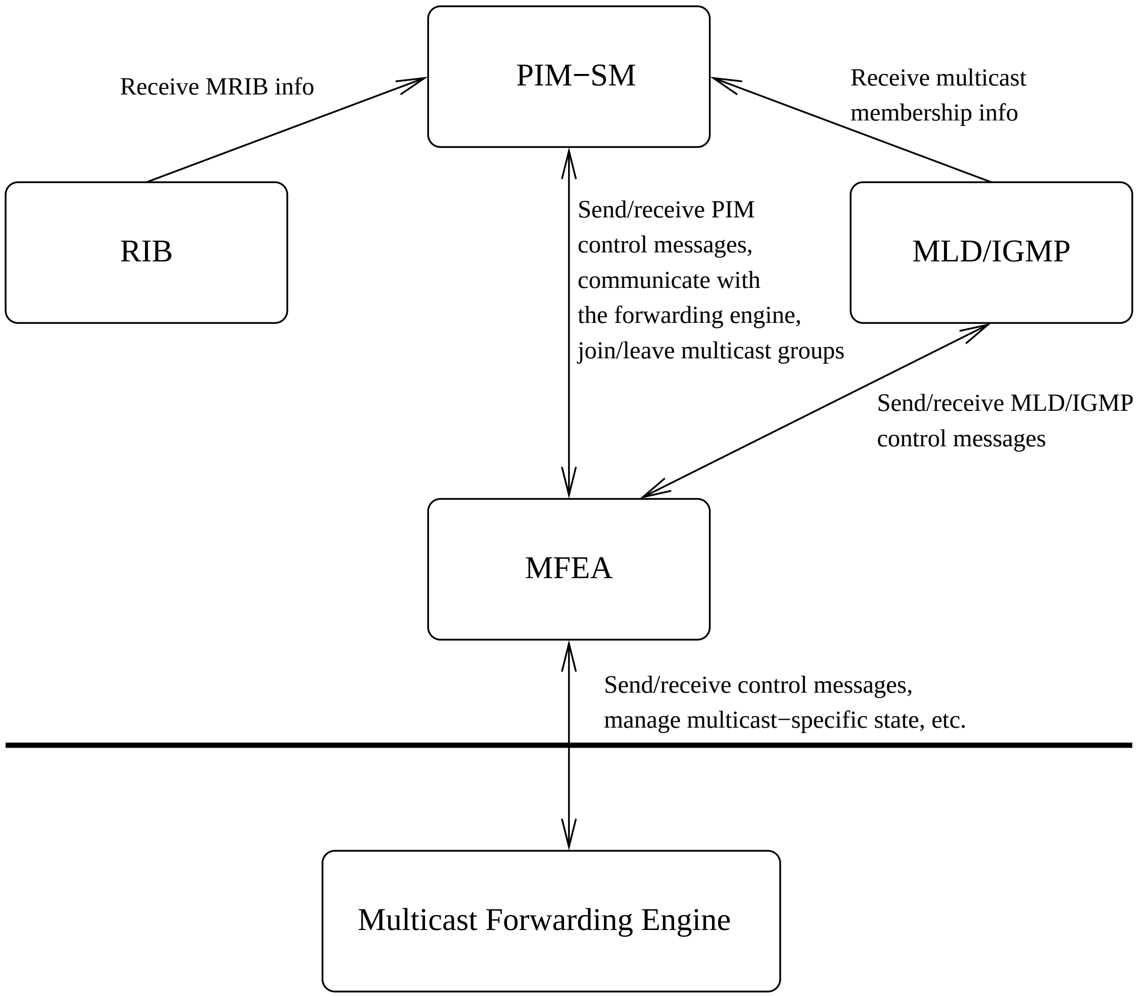
- Simplicity gives you lots of space for maneuvers
- Consistency (*e.g.*, in variables naming): things don't get into your way when you shuffle them around
- Which one comes first would be a trade-off
- Don't go into extremes

Forget (for now) the word "optimization"!!

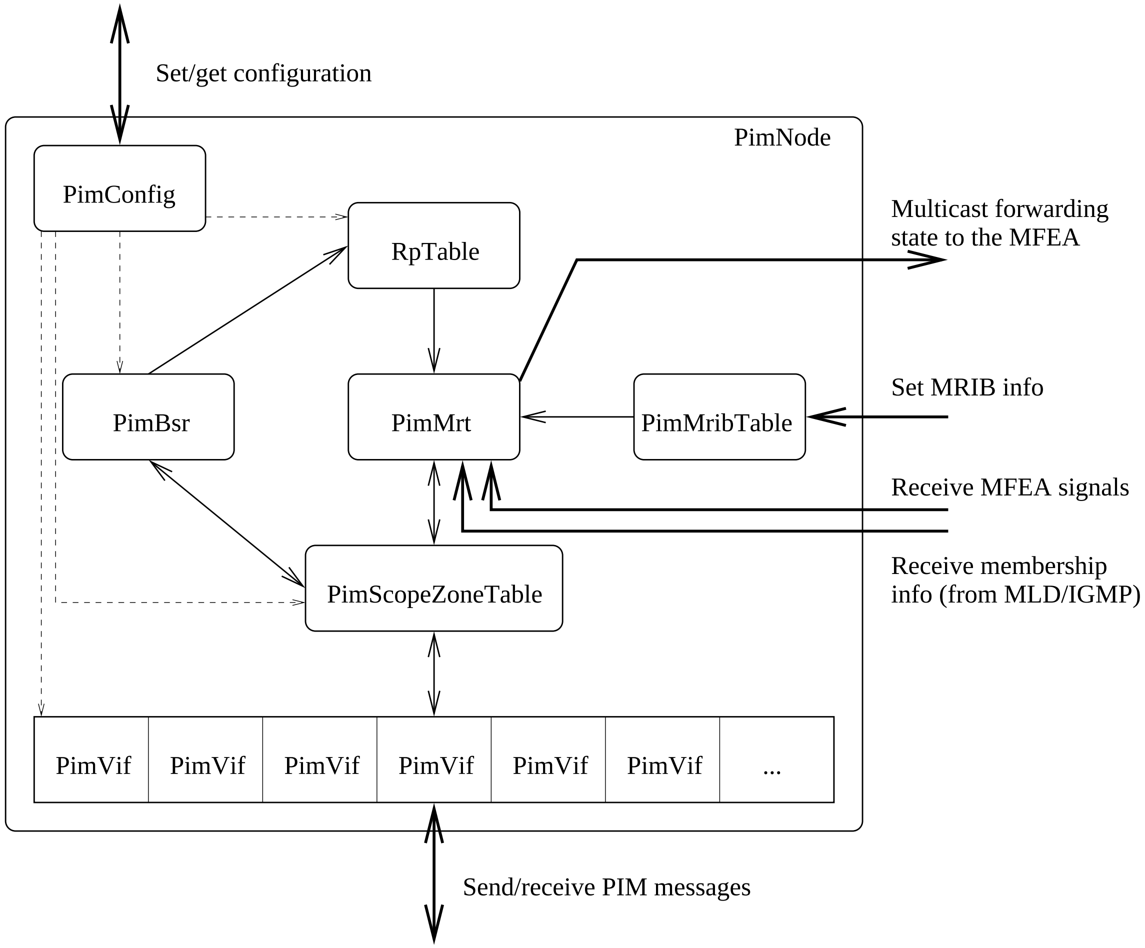
PIM-SM may have lots of routing state:

- So what, by the time the implementation is ready for prime-time, the price of memory will fall in half!
- Premature optimization results in complicated design, which is a sure sign for disaster!
- Solve performance issues when you do testing and profiling (*i.e.*, after the implementation is completed)

1. Design and understand the interaction with other modules



2. Break-down the protocol into semi-independent units



Protocol units break-down

- Probably the most difficult part
- There is no way you will get it right the first time!
- Simplicity comes first!

3. Protocol units implementation

- If you got your design right, in this stage you need to concentrate only on the protocol detail
- Be consistent!
- Each unit must respond to common methods/commands. E.g.: start/stop/enable/disable.
- Try to avoid implementation-specific assumptions

4. Testing, testing, testing

- If you don't test it, it doesn't work!
- Detailed testing takes time
- If you can, build a testing framework that allows you to perform automated testing any time you change something
- Now you can profile and optimize

Dependency tracking mechanism

- For each input event, what are the operations to perform and their ordering

- If the protocol is simple, you can take care of this by hand

- Unfortunately, this is not the case with PIM-SM: total of 50 input events, and 70 output operations.

PIM-SM dependency tracking mechanism

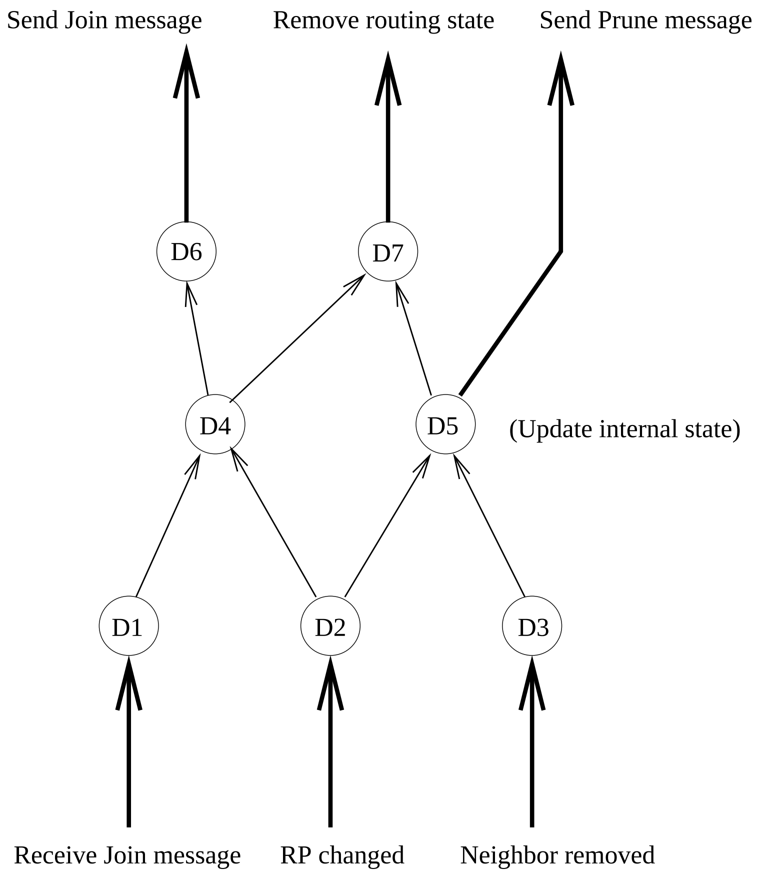
PIM-SM spec has tens of macros like:

```
pim_include(S,G) =
    { all interfaces I such that:
      ( (I_am_DR( I ) AND lost_assert(S,G,I) == FALSE )
        OR AssertWinner(S,G,I) == me )
       AND  local_receiver_include(S,G,I) }
```

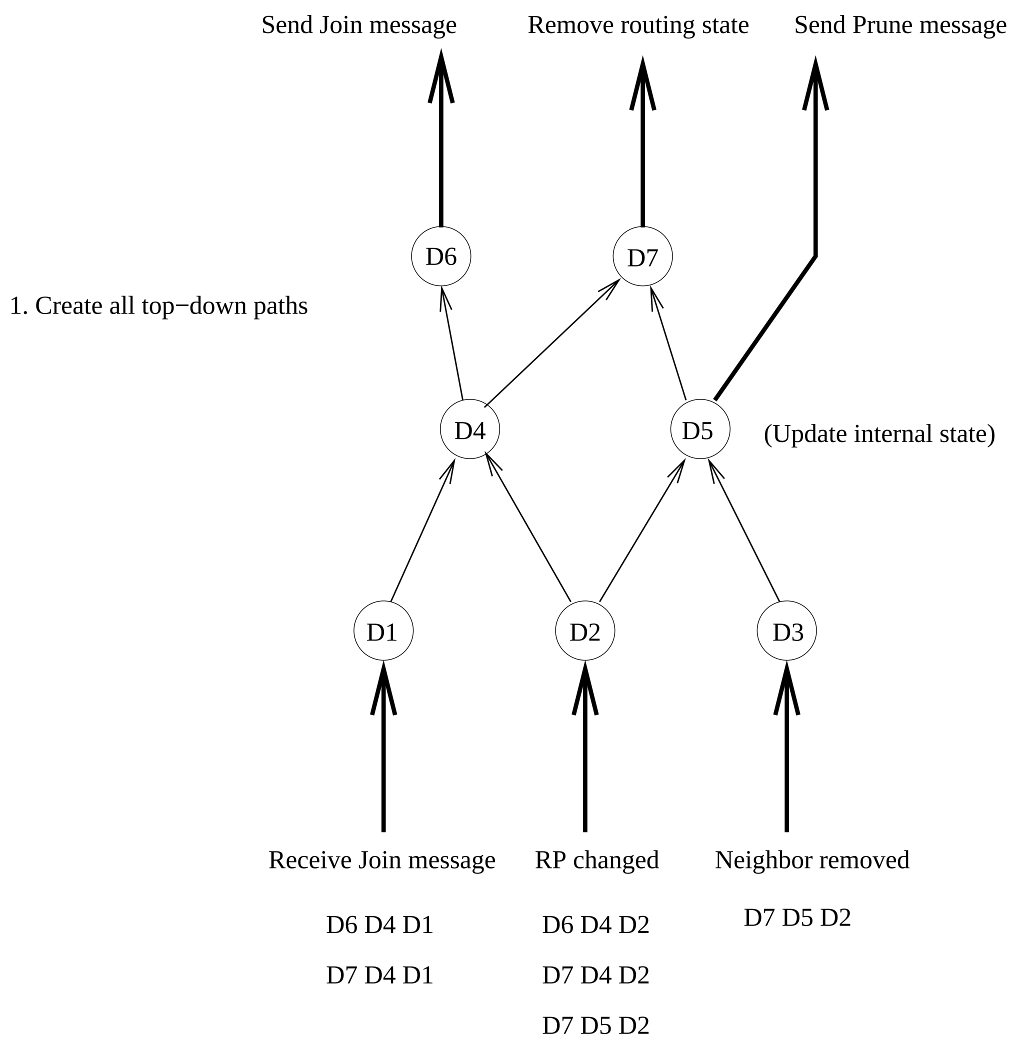
The corresponding state dependency rule is:

```
void
PimMreTrackState::track_state_pim_include_sg(list<PimMreAction> action_list)
{
    track_state_i_am_dr(action_list);
    track_state_lost_assert_sg(action_list);
    track_state_assert_winner_sg(action_list);
    track_state_local_receiver_include_sg(action_list);
}
```

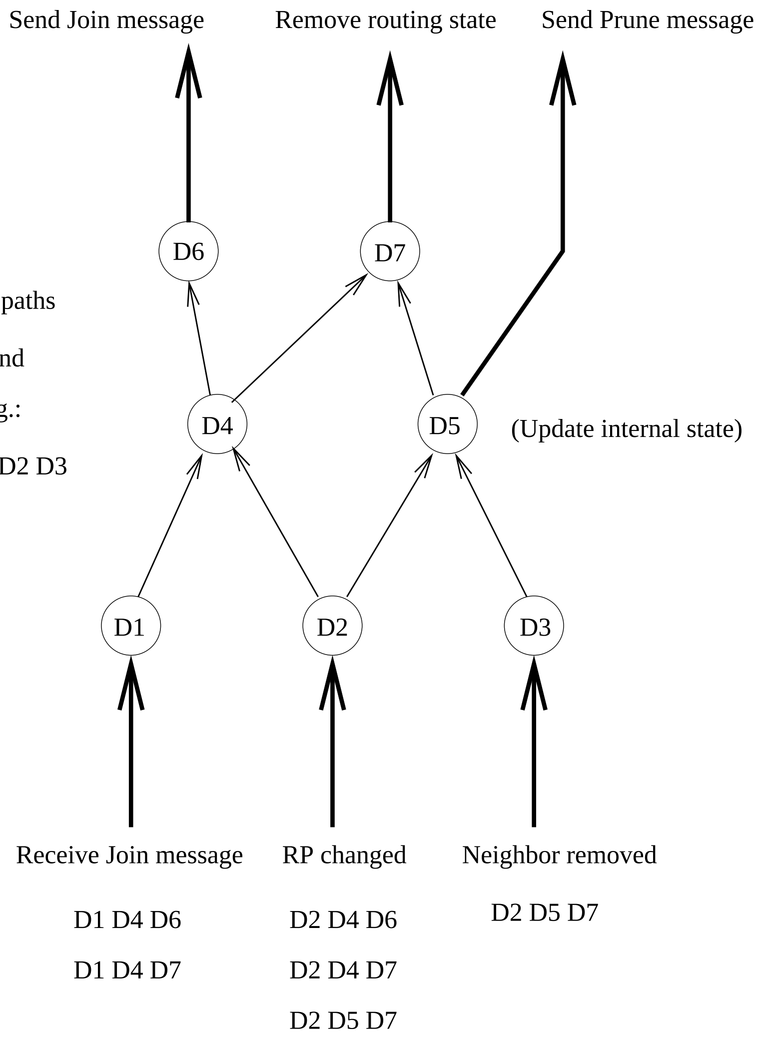
Dependency tracking



Dependency tracking (2)



Dependency tracking (3)



1. Create all top−down paths

2. Remove duplicates and reverse the paths. E.g.:

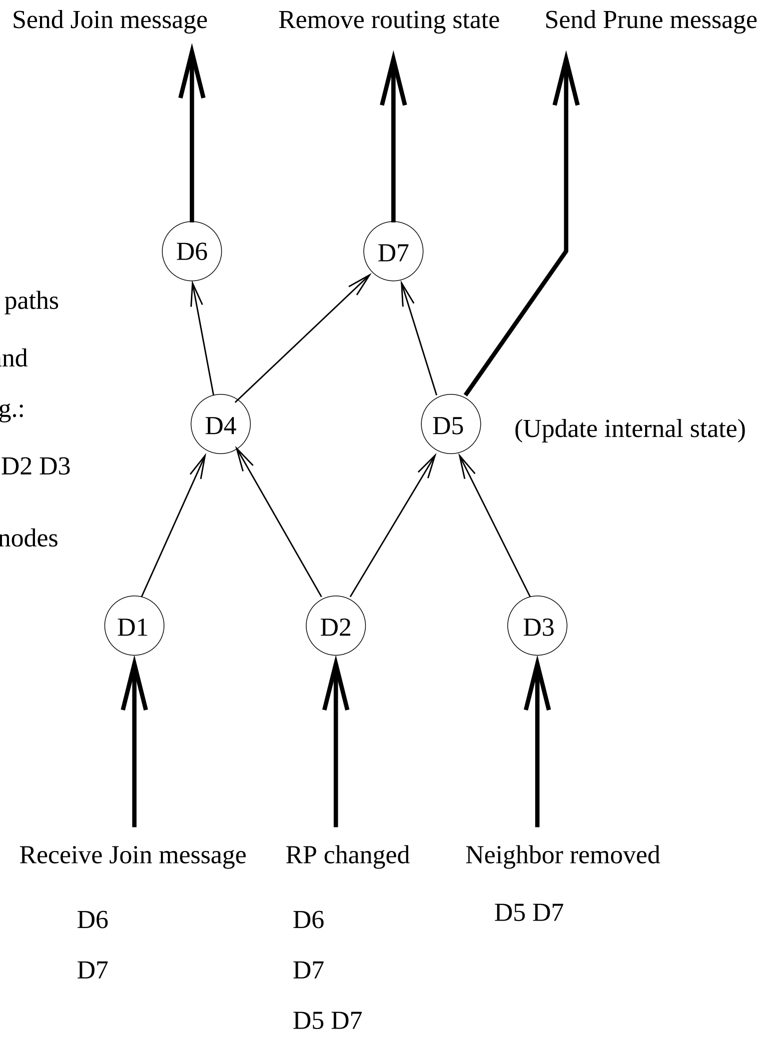
D3 D2 D2 D1 −> D1 D2 D3

Dependency tracking (4)

1. Create all top−down paths
2. Remove duplicates and
   reverse the paths. E.g.:

   D3 D2 D2 D1 −> D1 D2 D3

3. Remove non−output nodes
   from each list



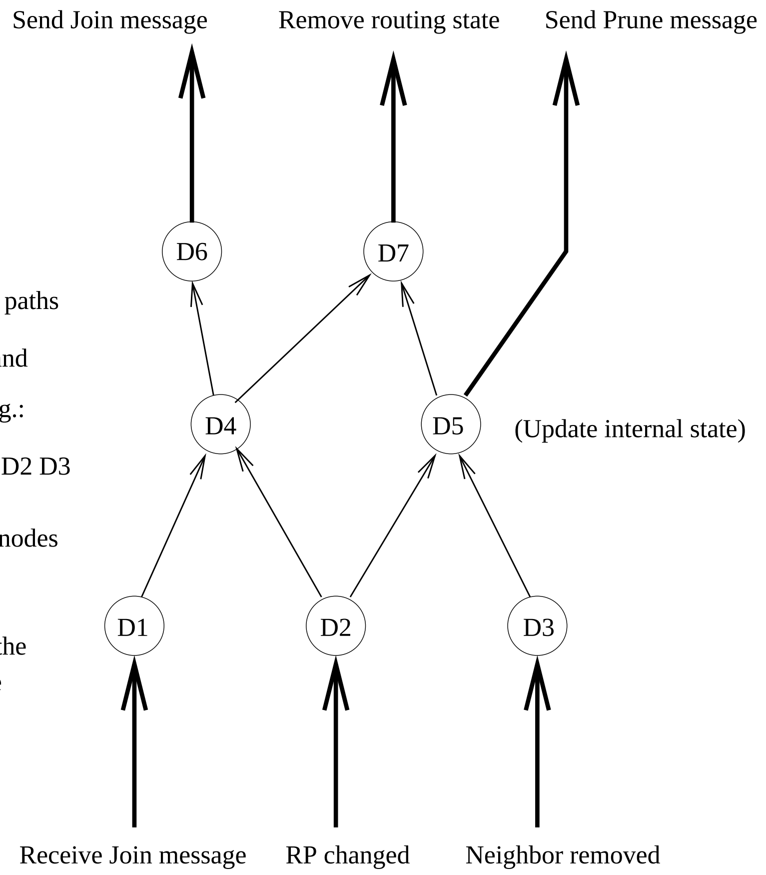
Dependency tracking (5)

1. Create all top−down paths
2. Remove duplicates and
   reverse the paths. E.g.:

   D3 D2 D2 D1 −> D1 D2 D3

3. Remove non−output nodes
   from each list

4. Recursively pop−up the
   actions that are at the
   head of a list only



Receive Join message	RP changed	Neighbor removed
D6	D6	D5 D7
D7	D7	
	D5 D7	
D6 D7	D6 D5 D7	D5 D7

Dependency tracking usage

- The unidirectional "graph" is semi-defined by the state computation macros

- For each macro, write the corresponding state dependency rule

- All state dependency is pre-computed once on start-up

- If the spec changes, the rules are easy to update

- If the spec does not use macros for state computation, write your own macros

Status

- Completed: core design, IPC, RIB, BGP, PIM-SM, IGMP, FEA

- In progress: OSPF, RIP adaptation, IPv6, Click integration,

- Future work: create XORP simulation environment

- First preliminary release early December: **http://www.xorp.org/**

Summary

- XORP tries to close the gap between **research** and **practice**

- Routing architecture designed for **extensibility** and **robustness**.

- Can be used to build distributed routers

- XORP simulation environment can facilitate protocol development: the simulation and the real-world prototype use exactly same code