# XORP: Breaking the Mould in Router Software

Mark Handley Adam Greenhalgh

Dept of Computer Science, University College London

{M.Handley, A.Greenhalgh}@cs.ucl.ac.uk

**Abstract:** Network researchers face a significant problem when deploying software in routers. Router platforms are generally not open systems. In this paper we discuss the problems this poses, and present an eXtensible Open Router Platform (XORP) which has the key goals are extensibility, performance and robustness. XORP is both a research tool and a stable deployment platform, thus easing the transition of ideas from the lab to the real world.

## 1 Research and Reality

The Internet has been fabulously successful; previously unenvisaged new applications appear frequently, and changing usage patterns have been accommodated with relative ease. But underneath the successful veneer, the low-level protocols that support the Internet have largely ossified, and stresses are beginning to show. Examples are security and convergence problems with BGP routing, the inability to deploy multicast, QoS, or IPv6, and the lack of effective defence mechanisms against denial-of-service attacks.

At the same time, participation in the IETF by the research community is at an all time low, and many researchers have moved into areas such as sensor-nets, Grid computing, and overlay networks where they perceive they can still make a difference. This trend is understandable, but seems a recipe for disaster in the long term, because the Internet industry is not doing research that would solve these problems. Thus it behoves us to ask two questions: *what is the root cause of this disconnect*, and *what can we do to solve it?*

The root cause appears to be that the router software market is closed, in the sense that if you buy a router from a vendor, then that router will only run that vendor's software. This makes it almost impossible for researchers to experiment in real networks, or to develop proof-of-concept code that might convince network operators that there are alternatives to current practice. Much innovation is also driven by startup companies, but the lack of open router APIs excludes this channel for change too.

The solution seems simple in principle: router software needs to have open APIs for extensibility. Unfortunately existing router software was not written with third-party extension in mind, so doesn't generally include the right hooks, extension mechanisms and security boundaries. How then can we enable a pathway that permits research and experimentation to be performed in production environments whilst minimally impacting existing network services? In part, this is the same problem that Active Networks attempted to solve, but we believe that a much more conservative approach is more likely to see real-world usage.

Our vision is of an integrated open-source software router platform, running on commodity hardware, that is viable as both a research and as a production platform. The software architecture should be designed with extensibility as a primary goal, permitting experimental protocol deployment with minimal risk to existing services. Internet researchers needing access to router software could then share a common platform for experimentation deployed in places where real traffic conditions exist. In these ways, the loop between research and realistic real-world experimentation can be closed, and innovation can take place much more freely.

We started work in early 2001, and made our 1.0 release this summer. We call it XORP.

## 2 The Vision

XORP stands for eXtensible Open Router Platform. It is called an *extensible* router platform for good reason. We believe that only by designing for extensibility from the outset can we simultaneously satisfy several different user groups:

- Network researchers needing a platform for experimentation.
- Network operators needing a low-cost stable routing platform on commodity hardware.
- Network equipment vendors with special purpose hardware.
- Network application writers looking for an open platform to support their applications.

Only time will tell if we're right, but in this paper we'll try and explain where we hope to go from here. Of course we'll only get there with good luck and a lot of help from all of these communities. First we'll discuss the XORP architecture, then return to how it helps these diverse groups.

## 2.1 XORP Overview

XORP divides into two subsystems. The user-level subsystem consists of the routing protocols themselves, along with routing information bases and support processes. The kernel-level subsystem manages the forwarding path and provides APIs for the user-level to access. The goal is for almost all of the user-level code to be agnostic as to the details of the forwarding path.

User-level XORP has a multi-process architecture, with one process per routing protocol, plus extra processes for management, configuration, and coordination. To enable extensibility we designed a novel inter-process communication mechanism, XORP Resource Locators (XRLs), for communication between these modules.

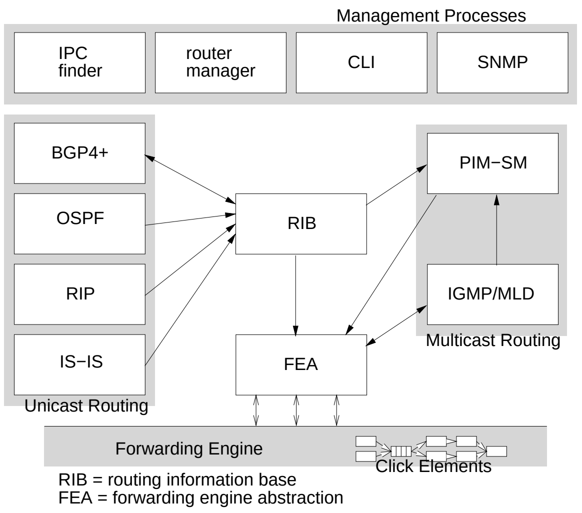

FIGURE 1—XORP High-level Processes

Figure 1 shows a XORP router's user-level processes and forwarding engine. The shared user-level processes are the XORP architecture's most innovative feature. Four core processes are particularly worthy of comment.

The *router manager* process manages the router as a whole. It maintains configuration information; starts processes such as routing protocols, as required by the configuration; and restarts failed processes as necessary.

The *ipc finder* process stores mappings between application requests, such as "What are this router's interfaces?", and the particular IPC calls necessary to answer those requests. Think of it as an IPC redirector: when an application wishes to make an IPC call, it consults the finder to discover how to do it. Thus it is easy to change how application requests are handled at run-time. We can also trace XORP's internal communications by asking the finder to map abstract requests to sequences of IPCs, some for tracing and some for doing the work.

The *routing information base* process (RIB) receives routes from the routing processes, and arbitrates which routes should be propagated into the forwarding path, or redistributed to other routing processes. The forwarding path is managed by the *forwarding engine abstraction* process (FEA). The FEA abstracts the details of how the forwarding path of the router is implemented and as a result, the routing processes are agnostic to whether the forwarding plane is Click [2] based, a conventional UNIX kernel, or novel forwarding hardware The FEA manages the networking interfaces and forwarding table in the router, and provides information to routing processes about the interface properties and events occurring on interfaces, such as an interface being taken down.

## 2.2 Extensibility

XORP's design encourages the construction of useful interfaces through multi-process design. A routing protocol process, for example, must communicate with other processes to install routes and discover information about the router itself. Open inter-process interfaces, built in to the system from the beginning, form the basic source of user-level XORP's extensibility.

Most inter-process communication within XORP takes place via XRLs. XRLs resemble the Web's URLs. They specify in human-readable form the type of IPC transport desired (the "protocol"), the abstract name for the entity being communicated with, the method being invoked, and a list of named arguments. Unlike URLs, they can also specify the nature of the response expected. Figure 2 shows one of the XRLs for the forwarding engine abstraction (FEA) process in human readable form. The initial 'finder:' tag specifies the protocol; in this case the actual protocol has not yet been resolved. The first time this XRL is called, the client contacts the finder, which redirects to a new XRL containing the actual protocol and parameters to be used to contact the current FEA. Subsequent communication bypasses the finder.

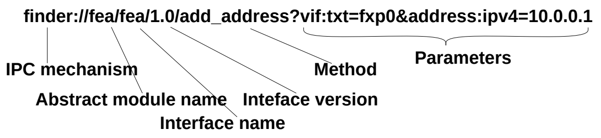

FIGURE 2—An example XRL

One motivation for XRLs was to encapsulate existing protocols within our consistent IPC framework. For example, we might wish to run third-party software that uses SNMPv3 for configuration. To integrate this software into our XRL-based management framework, we might write an SNMP 'protocol family' for the XRL client library. Then XORP processes could transparently interoperate with the third-party software via XRLs that start with 'snmp:'.

### 2.2.1 *XRL Example: Command-line Interface*

One of the biggest issues faced by an extensible router is the integration of separately maintained components into a coherent system. Our solution is for all management, including initial configuration, to take place using XRLs. To add support for a specific management mechanism, such as SNMP or a command-line interface, the protocol implementor writes simple text files that map management requests to XRL calls. These thin mapping files are easy enough to write that third parties might add them as new management interfaces become available.

To get more concrete, our configuration manager has a strict hierarchy of configuration parameters, which is directly reflected in our command line interface (CLI). The configuration manager takes a directory of template files, which define the possible configurable parameters for each XORP routing protocol, and generates mappings of configuration parameters to XRL dispatches. The designer of a new routing protocol can simply supply a template file specifying the new functionality provided. The configuration manager can read the template file, discover the new functionality, and know how to communicate with the process. The new functionality is then immediately available through the CLI. For example, a fragment of a router configuration file, and the corresponding part of a template file might look like:

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

```
hello-interval: uint {
   %set: xrl "ospf/ospf/1.0/set_hello_interval?
                  if:txt=$(IFNAME)&interval:i32=$(VALUE)";
   %get: xrl "ospf/ospf/1.0/hello_interval?if:txt
                  -> interval:i32";
}
```

## 2.3 Robustness

The routing and coordination processes in XORP run in user space on a regular UNIX operating system. Routing processes are protected from each other and can have their resources constrained according to administrative preference. Furthermore, routing processes can crash without affecting the kernel, forwarding plane, or each other. And if a routing protocol does crash, the RIB will remove its routes from the forwarding engine, and optionally inform the re-starting routing process of the routes it previously held.

A significant aspect of robustness is security. One benefit of being forwarding-path agnostic is that we can abstract privileged operations, such as sending on a raw socket, into the FEA via XRLs. This allows us to run many routing protocols in a sandbox. They have no interaction with the outside world except through XRLs and packets, and so a vulnerability in a routing protocol is far more difficult to escalate into full control of the router. Untrusted code might even be run in a virtual machine such as provided by Xen[1].

## 3 Satisfying Diverse Users

We now return to examine how XORP might satisfy the different user bases we mentioned earlier.

**Network Research:** A lot of work happens in simulation, but often it's hard to know whether the simulation bears much relationship to reality. There's really no substitute for trying something out in the real world. This is where XORP comes in, here are some examples of how XORP can be used:

- If XORP is used as a production router, it is easy to instrument it to perform measurements of traffic, routing messages, or practically anything else that goes on in a router.
- XORP can be used as a platform to develop new routing protocols. XORP has no built-in concept of which routing protocols exist, so you can easily add your own, together with an ASCII configuration template file, so the XORP command line interface knows what new functionality is available.
- XORP can be used as a network emulator using *imunes*[3]. XORP's Forwarding Engine Abstraction (FEA) provides an interface through which the whole control plane communicates with the forwarding plane and the outside world. It is possible to modify the FEA so that multiple emulated routers all exist on the same host. This allows experiments using XORP's real routing code to take place in a carefully controlled environment. It also facilitates new protocol development without needing a large lab of machines.
- XORP is scriptable. XORP's internal communication between processes uses XRLs. These have a canonical form that is ASCII, such as:

  ```
  finder://fea/fti/0.1/add_route ? net:ipv4net=10.0.0.1/8 & gateway:ipv4=192.150.187.1
  ```

  A command line program *call_xrl* allows any scripting language to make calls to any XORP process. We believe this scripting ability is unique as an enabler for novel uses of existing router code.

**XORP as a Low-Cost Router:** There are many organisations that need router functionality, but cannot justify buying an expensive commercial router. Although there are cheap home routers available these days, they are not really intended to run any non-trivial routing configuration. A modern $300 PC has enough forwarding capability to saturate a few 100Mb/s network links, so it's very viable to build your own PC-based router.

You should not need to know anything about how XORP works internally to use it. XORP has a single unified CLI which allows all the routing protocols, network interfaces, and so forth to be configured. In future releases, this CLI will also be extended to encompass additional router functionality such as queue management, QoS configuration, firewalls, NATs and DHCP configuration. XORP already supports IPv4 and IPv6, together with BGP4+ and RIP for unicast routing, PIM-SM and IGMPv2 for multicast, and limited SNMP support.

The XORP architecture permits different routing protocols to run in different security sandboxes. For example, BGP does not need access to the router's filesystem or need privileged access to communicate with its peers so, should something go wrong, it is much harder to compromise the rest of the router. The aim is for XORP to be more robust and secure than alternative router platforms.

It is very important to us that XORP is both very stable and has sufficient features for mission-critical production use. We will realise this goal only with extensive feedback from our users with regards to what works well in the real world, and what does not.

**XORP for Equipment Vendors:** Once XORP has proven itself as a stable software stack for PC routers, and its functionality and feature set filled out fully, we hope it will prove an attractive alternative to commercial stacks for network equipment vendors. XORP has a BSD-style license, which allows it to be used for any purpose.

We believe XORP's architecture is well suited as a software stack to control an advanced hardware forwarding plane. Our forwarding engine abstraction (FEA) process provides a key abstraction layer providing isolation between all the higher level routing functionality and the underlying operating system and forwarding engines. This should make XORP comparatively easy to port to new platforms and architectures.

**XORP for Network Application Writers**: In the long run, we hope that XORP will enable a class of software that currently does not exist: the *router application*. Currently, there is no market for third party software for mainstream commercial router platforms. This is clearly because of the lack of open APIs.

We believe that XORP's extensible architecture is a possible solution to this problem. XORP's novel inter-process communication mechanism, combined with it's run-time extensible router-manager process and CLI should permit a router operator to install a new binary application process on a XORP router, and for it to appear as an integrated part of the router from an operational point of view. We are very interested to see what novel network functionality this enables in the future.

## 4 Summary

We believe that innovation in the core protocols supporting the Internet is being seriously inhibited by the nature of the router software market. Furthermore, little long term research is being done, in part because researchers perceive insurmountable obstacles to experimentation and deployment of their ideas.

In an attempt to change the router software landscape, we have built an extensible open router software platform. The next few years will be critical. We have a stable core base running, available from http://www.xorp.org, consisting of around 500,000 lines of C++. In the next phase we need to involve the academic community, both as early adopters, and to flesh out the long list of desirable functionality that we do not yet support. If we are successful, XORP will become a true *production-quality* platform. We already have interest from a number of networking startup companies. The road ahead will not be easy, but we believe that some significant changes to permit innovation are necessary, and the consequences of a lack of innovation in the long run will be serious.

## Acknowledgments

XORP project was funded Intel and the NSF (Award ANI-0129541). Javier Cardona, Atanu Ghosh, Orion Hodson, Eddie Kohler, Pavlin Radoslavov and Luigi Rizzo have also contributed significantly to XORP.

## References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. m Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfi eld. Xen and the art of virtualization. In *Proc ACM SOSP*, pages 164–177. ACM Press, 2003.

[2] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Trans. on Computer Systems*, 18(3):263–297, Aug. 2000.

[3] M. Zec and M. Mikuc. Real-time ip network simulation at gigabit data rates. In *Proc. 7th Intl. Conference on Telecommunications (ConTEL)*, 2003.