XORP: An Open Platform for Network Research

Mark Handley Orion Hodson Eddie Kohler

ICSI Center for Internet Research, Berkeley, California

{mjh, hodson, kohler}@icir.org

Abstract

Network researchers face a significant problem when deploying software in routers, either for experimentation or for pilot deployment. Router platforms are generally not open systems, in either the open-source or the open-API sense. In this paper we discuss the problems this poses, and present an eXtensible Open Router Platform (XORP) that we are developing to address these issues. Key goals are extensibility, performance and robustness. We show that different parts of a router need to prioritize these differently, and examine techniques by which we can satisfy these often conflicting goals. We aim for XORP to be both a research tool and a stable deployment platform, thus easing the transition of new ideas from the lab to the real world.

1 Validating Internet Research

A yawning gap exists between research and practice concerning Internet routing and forwarding disciplines. The savvy researcher has the tools of theory and simulation at hand, but validating results in the real world is hard. Why should this be so?

For network applications research, we have access to languages, APIs, and systems that make development and deployment easy. For end-to-end protocol research, we have access to open source operating systems, such as Linux and FreeBSD. End-to-end protocols can be simulated and implemented in these systems. And since these operating systems are used in both research and production environments, migration from the research to the production environment is feasible. TCP SACK provides an excellent example [8].

Unfortunately the same cannot be said of router software. Router vendors do not provide API's that allow third party applications to run on their hardware. Thus, even conducting a pilot study in a production network requires the router vendor to implement the protocol. Unless the router vendor perceives a reward in return for the effort, they are unlikely to invest resources in the protocol implementation. Similarly, customers are unlikely to request a feature unless they have faith in existing research results or can experiment in their own environment. A catch-22 situation exists of not being able to prototype and deploy new experimental protocols in any kind of realistic environment. Even when vendors can be convinced to implement, it is not uncommon for initial implementations of a protocol to be found wanting, and the path to improving the protocols is often difficult and slow. Finally, network operators are almost always reluctant to deploy experimental services in production networks for fear of destabilizing their existing (hopefully money-making) services.

Thus, we believe the difficulty in validating Internet research is largely attributable to the absence of open Internet routers for researchers to experiment with and deploy new work on. Routing toolkits exist, but typically they implement a subset of IP routing functionality and are rarely used in production environments—routing and forwarding research requires access to real production traffic and routing information to be validated. Similarly, open-source-based testbed networks such as CAIRN [1] provide valuable tools for the researcher, but they rarely provide a realistic test environment and are usually limited to a small number of sites due to cost. A recent spate of research in open, extensible forwarding paths is moving in the right direction [6, 11], but a truly extensible, production-quality router would need routing daemons, forwarding information bases, management interfaces, and so on in addition to a forwarding path.

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.

We envision an integrated open-source software router platform, running on commodity hardware, that is viable as a research and as a production platform. The software architecture should be designed with extensibility as a primary goal and should permit experimental protocol deployment with minimal risk to existing services using that router. Internet researchers needing access to router software could then share a common platform for experimentation deployed in places where real traffic conditions exist. Researchers working on novel router hardware could also use the mature software from this platform to test their hardware in real networks. In these ways, the loop between research and realistic real-world experimentation can be closed, and innovation can take place much more freely.

1.1 Alternatives

Having motivated the need for an open router on which network research can be deployed, we discuss the alternatives in more detail—simulations and network testbeds.

First, we note that it has not always been so difficult to deploy experimental work on the Internet. Prior to the advent of the World Wide Web, the Net was predominantly non-commercial. Most of its users came from universities and other research labs. Whilst there was a tension between conducting research and providing a networking service, researchers could have access to the network to run experiments, develop and test new protocols, and so forth. With the advent of the Web, the network grew rapidly and commercial ISPs emerged. Even the academic parts of the network became reluctant to perform networking experiments for fear of disrupting regular traffic. In the commercial parts of the network, where interesting scaling phenomena started to emerge, it was nearly impossible to do any form of experimentation. Growth was so rapid that it was all ISPs could do to keep up with provisioning. These problems were recognised, and two main solutions emerged: network testbeds and network simulators.

First Workshop on Hot Topics in Networks, Princeton, New Jersey, October 28–29, 2002

Network testbeds rarely resulted in good network research. One notable exception was DARTnet [3], which used programmable routers that network researchers had access to. Among its achievements, DARTnet demonstrated IP multicast and audio and video conferencing over IP. It worked well because the network users were also the network researchers and so there was less tension involved in running experiments.

Over recent years, the majority of network research that involved testing protocols has taken place in network simulators such as *ns*. Among the desirable properties of simulators is the complete control they provide over all the parameters of a system, and so a large range of scenarios can be examined. Within the research community the *ns* simulator has been particularly successful[1]. Many researchers have contributed to improve *ns* itself, but an even greater number have used it in their research. Many published results are supported by publicly available simulation code and scripts. This has allowed for the direct comparison of contemporary networking algorithms and allowed for independent verification of results. It could therefore be argued that *ns* has increased the rigor of network research.

Conversely, it could equally well be argued that simulators make it *too easy* to run experiments and are responsible for numerous papers that bear little, or no, relationship to real networks. Accordingly there is understandable doubt about any claims until they've been demonstrated in the real world.

Even in skilled hands, simulators have limits. When work requires access to real traffic patterns, or needs to interact with real routing protocols, or relates to deployed implementations warts-and-all, there is no substitute for real-world experimentation.

2 Architecture and Requirements

We've hopefully convinced you that Internet research would be better off if changes and extensions could be provisionally deployed. No simulation or testing environment can provide similarly useful lessons—or convince conservative router vendors that extensions work well enough to deserve real-world deployment. We want a routing infrastructure that is at least partially open to research extensions. The infrastructure must, further, meet Internet robustness standards to merit deployment even on research networks like Abilene. But how can we make this happen? There are several possibilities:

- Individual researchers could convince big router vendors to implement their extensions. To put it succinctly, this seems unlikely.
- Vendors could open their internal APIs for experimental use. This also seems unlikely—vendors probably consider their APIs to be trade secrets, and don't want to facilitate an open market in routing software. Furthermore, legacy code inside most routers isn't particularly easy to extend.
- Researchers could deploy currently available routing daemons such as Zebra [14], MRTd [13] or GateD [10], on a conventional PC running an operating system such as Linux or FreeBSD. The principle problems here are extensibility, performance, and robustness. The forwarding paths on these operating systems are not optimally designed for performance under heavy load, or for extensibility. The routing daemons are not as robust as we would like, and not generally designed with extensibility as a high priority. Finally, they don't present an *integrated* router user interface to the network operator, which reduces the likelihood of acceptance.
- Researchers could develop a new open-source router, designed from the ground up for extensibility, robustness, and performance. It would take time to build such a router, and then to convince the network operator community that it met stringent robustness standards, but the resulting router would make a more useful research platform than any of the other choices.

[1] One could criticize the details of the *ns* architecture and some of the default settings, but that would miss the point.

We have chosen this last path, which we call the Extensible Open Router Platform, or XORP. Our design addresses the four major challenges in building an open-source router: traditional router features, extensibility, performance, and robustness.

Features. Real-world routers must support a long feature list, including routing protocols, management interfaces, queue management, and multicast.

Extensibility. Every aspect of the router should be extensible, from routing protocols down to details of packet forwarding. The router must support multiple simulatenous extensions as long as those extensions don't conflict. APIs between router components should be both open and easy to use.

Performance. XORP isn't designed for core routers, at least not initially. However, forwarding performance is still important: that is the purpose of a router. Scalability in the face of routing table size or number of peers is also critical.

Robustness. Real-world routers must not crash or misroute packets. A fragile router faces an extremely difficult deployment path. Extensibility makes robustness even more difficult to achieve: extensions are inherently experimental, yet their use should not compromise the router's robustness (except perhaps for a subset of packets intended for the extension).

The next sections present XORP in general and describe how we're achieving each of these goals.

3 XORP Overview

XORP divides into two subsystems. The higher-level (so-called "user-level") subsystem consists of the routing protocols themselves, along with routing information bases and support processes. The lower-level subsystem, which initially runs inside an OS kernel, manages the forwarding path—anything that needs to touch every packet—and provides APIs for the higher level to access. The goal is for almost all of the higher-level code to be agnostic as to the details of the forwarding path.

For user-level XORP, we've developed 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 for communication between these modules. This mechanism is called XORP Resource Locators (XRLs), and is conceptually similar to URLs. URL mechanisms such as redirection aid reliability and extensibility, and their human-readable nature makes them easy to understand and embed in scripting languages.

The lower level uses the Click modular router [6], a modular, extensible toolkit for packet processing on conventional PCs. Further work will help this architecture span a large range of hardware forwarding platforms, from commodity PC hardware, through mid-range PC-based platforms enhanced with intelligent network interfaces (such as Intel's IXP1200 [5, 12]), to high-end hardware-based forwarding engines. We may also support alternative forwarding paths, such as the FreeBSD forwarding path with AltQ queuing extensions [2] or an alternative extensible forwarding path such as Scout [11]. Some forwarding path choices may influence the functionality available for end users to extend or change. But for many aspects of research, such as routing protocol experiments that don't require access to the forwarding path, a conventional FreeBSD lower level would suffice.

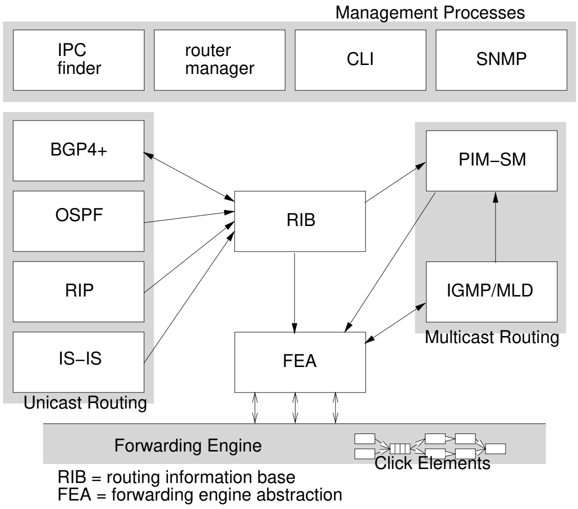


FIGURE 1—XORP High-level Processes

Figure 1 shows how a XORP router's user-level processes and Click forwarding path fit together. The shared user-level processes are the XORP architecture's most innovative feature. Four core processes are particularly worthy of comment: the *router manager*, the *finder*, the *routing information base*, and the *forwarding engine abstraction*.

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

The *finder* process stores mappings between abstracted application requests, such as "How many interfaces does this router have?", 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. The application typically caches this information so future calls circumvent the finder. Furthermore, the finder can instruct applications to update contact information. Thus it is easy to change how application requests are handled at run-time. We can trace XORP's communication pattern by asking the finder to map abstract requests to sequences of IPCs, some for tracing and some for doing the work. XORP processes can communicate without bootstrapping using the finder, but since XRLs are relatively low cost we have not found this necessary to date.

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 based, a conventional UNIX kernel, or an alternative method. 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. As with the finder, XORP processes can bypass the FEA when required.

4 SOLVING DESIGN CHALLENGES

This section shows how we address the four design challenges of traditional router features, extensibility, robustness, and performance. Space constraints prevent inclusion of much detail, but we do describe our IPC mechanism, XORP Resource Locators (XRLs), at length.

4.1 Features

To be successful, a router platform needs to be have good support for the routing and management protocols that are in widespread use today. The minimal list of routing and routing-support protocols is:

- **BGP4+** inter-domain routing.
- **OSPF** intra-domain routing.
- **RIPv2/RIPng** intra-domain routing.
- **Integrated IS-IS** intra-domain routing.
- **PIM-SM/SSM** multicast routing.
- **IGMPv3/MLD** local multicast membership.
- **PPP** for point-to-point links (as per RFC 1812).

With the exception of IS-IS, all of these are currently being worked upon within XORP. The aim is for both IPv4 and IPv6 support. MPLS is deliberately omitted at this stage. The multi-process architecture helps us to leverage existing code where appropriate; our OSPF and RIP implementations are derived from John Moy's *OSPFd* [9] and BSD's *routed* respectively.

For management interfaces, we are pursuing a command line interface resembling that of Juniper and intend to support SNMP.

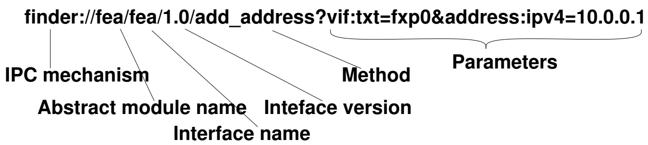
4.2 Extensibility

Open interfaces are the key to extensibility. Interfaces must exist at any point where an extension might plug in, and those interfaces must be relatively easy to use. 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.

4.2.1 XRLs

Most inter-process communication within XORP takes place via XORP Resource Locators, or 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.

As an example, the general form of one of the XRLs for the forwarding engine abstraction (FEA) process might be rendered in human readable form as:



The initial 'finder:' portion specifies the protocol; in this case the actual protocol has not yet been resolved. The first time this XRL is called, the client XRL library contacts the finder, which responds with a redirection to a new XRL containing the actual protocol to be used, together with all the parameters needed to contact the current FEA. Subsequent communication then goes directly between the client and the FEA process. If the FEA restarts, the client's XRL calls to the old FEA will fail, and it can consult the finder to update the redirection.

The XRL library, which all of our processes link against, takes an XRL and performs argument marshaling, then it invokes the specified transport mechanism, and handles the response or any failure that might occur. Unlike many RPC or remote method invocation mechanisms, XRLs don't try and hide from the programmer that off-process communication is taking place. While this makes the programmer's job harder at first, it is essential for robustness that the programmer is made aware that the failure modes of IPC are different from those of a local procedure call. To help the programmer and improve performance, an IDL and a stub generator exist, so most XRL clients never need to parse the human readable form.

The original 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:'. XRLs are general enough to encompass simple communication with the kernel via `ioctls`, and even signaling via `kill()`. At the present time, we have not had a need to add this functionality, but should the need arise, our architecture would support it. The current XRL library supports XORP-specific protocols for remote procedure call, one layered over TCP and the other over UDP, and a local procedure call mechanism for intra-process communication.

4.2.2 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. Consider the interaction between management mechanisms such as a command-line interface (CLI) and a new routing protocol. The author of each of the management processes has no knowledge of future routing protocols. At the same time, the author of each routing protocol has no knowledge of future management mechanisms. 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 default CLI. A fragment of a router configuration file might look like:

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

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 add a template file specifying the new functionality provided. Thus, the template entry for OSPF might contain the fragment:

```
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";
}
```

The configuration manager can read the template file, discover the new functionality, and know how to communicate with the process to use it. The new functionality is then immediately available through the CLI.

4.3 Performance

Simultaneouly satisfying the three goals of extensibility, performance, and robustness without compromise is probably not possible. Different parts of XORP have different priorities. User-level XORP does not need particularly performance-conscious design. Route updates can arrive at high rates, but nowhere near the rates of packet arrival. It is well within the capabilities of a desktop PC to handle the volume of data and computation required for unicast routing. (The main issues here are of the computational complexity of the protocol implementations.) At user-level, our priorities are extensibility and robustness.

The forwarding path, however, must touch every packet and performance is paramount. Initially we have focused on a PC-based hardware platform as this allows for easy testing and early deployment, but great care is being taken to design a modular forwarding path architecture which can support some or all of the forwarding functions happening in hardware. For our own work, the preferred forwarding path is based on the Click modular router [6]. Click provides a high-performance forwarding path in the kernel, running on commodity hardware, and allows for run-time extensions and reconfiguration. Click's modularity allows for many divisions of work between software and hardware. If researchers or developers decide they want to augment XORP with network processors[2] or special-purpose hardware [7], this will be an advantage.

Click forwarding paths are extensible in two key ways:

- Existing defined elements can by interposed into a forwarding path to add new functionality.
- New Click elements can be created, loaded as kernel modules, and then plumbed in to the forwarding path.

In the context of XORP, this extensibility can aid performance. For example, a network administrator can ensure that only a small subset of traffic goes through a possibly-slow extension, by defining a new special-purpose classifier that matches extension traffic and inserting it into the relevant place in the forwarding path.

4.4 Robustness

The routing and coordination processes in XORP run in user space on a traditional UNIX operating system. Routing processes

[2]Network processors [5] are interface cards containing a number of high-speed network interfaces, typically together with a processor and some specialized programmable hardware for performing forwarding. Typically the network processor can perform simple forwarding very well, but more complex functionality needs to be off-loaded to be handled by software running on the host processor [12].

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.

Multiple processes are used in Zebra [14] and Cisco's proposed ENA router operating system, but not by some of the larger commercial vendors today.

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 an exploitable vulnerability in a routing protocol is far more difficult to escalate into full control of the router.

Robustness in the forwarding path is also important, but solutions such as memory protection that work well in user-space are not acceptable. In the Click forwarding path robustness comes from the granularity and simplicity of Click's components. Each element is small enough to be well understood and tested in isolation. And since many of Click's components can be run and debugged in user-space, confidence about their robustness can be attained before being used in the kernel.

5 SUMMARY

We believe that much good Internet research is being frustrated by an inability to deploy experimental router software at points in the network where it makes most sense. These problems affect a wide range of research, including routing protocols themselves, active queue management schemes, and so-called "middlebox" functionality such as our own traffic normalizer [4]. Many of these problems would not exist if the router software market more closely resembled the end-system software market, which has well defined APIs for application software, and high performance reliable open-source operating systems that permit kernel protocol experimentation. Our vision for XORP is to provide just such an open platform; one that is stable and fully featured enough for serious production use (initially on edge-routers), but designed from the very outset to support extensibility and experimentation without compromising the stability of the core platform.

There are many unanswered questions that we still have to resolve. For example, network managers simply wish to say *what* a router should do; they don't typically care about how this is implemented. Click starts from the point where you tell it precisely how to plumb together the forwarding path. Thus we need a forwarding path configuration manager that, given the network manager's high-level configuration, determines the combination of click elements and their plumbing needed to implement the configuration. The simplest solution is a set of static profiles, but we believe that an optimizing configuration manager might be able to do significantly better, especially when it comes to reasoning about the placement and interaction of elements such as filters.

Thus, while XORP is primarily intended to *enable* research, we also believe that it will also further knowledge about how to construct robust extensible networking systems.

Finally, while we do not expect to change the whole way the router software market functions, it is not impossible that the widespread use of an open software platform for routers might have this effect. The ultimate measure of our success would be if commercial router vendors either adopted XORP directly, or opened up their software platforms in such a way that a market for router application software is enabled.

ACKNOWLEDGMENTS

The XORP project is funded by grants from Intel and the National Science Foundation (Award ANI-0129541). Atanu Ghosh, Adam Greenhalgh, Mark Handley, Orion Hodson, Eddie Kohler, Pavlin Radoslavov and Luigi Rizzo have contributed significantly to XORP.

REFERENCES

[1] CAIRN. Collaborative Advanced Interagency Research Network (Web site). `http://www.cairn.net/`.

[2] K. Cho. A framework for alternate queueing: towards traffic management by PC-UNIX based routers. In *Proc. USENIX 1998 Annual Technical Conference*, pages 247–258, June 1998.

[3] D. D. Clark, S. Shenker, and L. Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. *ACM Computer Communication Review*, 22(3):14–26, Oct. 1992.

[4] M. Handley, C. Kreibich, and V. Paxson. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *Proc. Usenix Security Symposium*, Aug. 2001.

[5] Intel Corporation. Intel IXP1200 network processor (Web site). `http://developer.intel.com/design/network/ixp1200.htm`.

[6] 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.

[7] J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor. Reprogrammable network packet processing on the field programmable port extender (fpx). In *Proc. ACM International Symposium on Field Programmable Gate Arrays (FPGA 2001)*, pages 87–93, Feb. 2001.

[8] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgment options. RFC 2018, Internet Engineering Task Force, Apr. 1996. `ftp://ftp.ietf.org/rfc/rfc2018.txt`.

[9] J. Moy. *OSPF Complete Implementation*. Addison-Wesley, Dec. 2000.

[10] NextHop Technologies. GateD releases (Web site). `http://www.gated.org/`.

[11] L. L. Peterson, S. C. Karlin, and K. Li. OS support for general-purpose routers. In *Proc. 7th Workshop on Hot Topics in Operating Systems (HotOS-VII)*, pages 38–43. IEEE Computer Society Technical Committee on Operating Systems, Mar. 1999.

[12] T. Spalink, S. Karlin, L. Peterson, and Y. Gottlieb. Building a robust software-based router using network processors. In *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 216–229, Oct. 2001.

[13] University of Michigan and Merit Network. MRT: Multi-threaded Routing Toolkit (Web site). `http://www.merit.edu/mrt/`.

[14] Zebra project. GNU Zebra routing software (Web site). `http://www.zebra.org/`.