API Design Challenges for Open Router Platforms on Proprietary Hardware

Jeffrey C. Mogul, Praveen Yalagandula, Jean Tourrilhes, Rick McGeer,
Sujata Banerjee, Tim Connors, Puneet Sharma
HP Labs, Palo Alto
{Jeff.Mogul,Praveen.Yalagandula,Jean.Tourrilhes,Rick.McGeer,Sujata.Banerjee,Tim.Connors,Puneet.Sharma}@hp.com

ABSTRACT

Most switch vendors have launched "open" platform designs for routers and switches, allowing code from customers or third-party vendors to run on their proprietary hardware. An open platform needs a programming interface, to provide switchlets sufficient access to platform features without exposing too much detail. We discuss the design of an abstraction layer and API designed to support portability between vendor platforms, isolation between switchlets and both the platform and other switchlets, high performance, and programming simplicity. The API would also support resource-management abstractions; for example, to allow policy-based allocation of TCAM entries among multiple switchlets.

1 INTRODUCTION

Traditionally, router and switch[1] platforms have either been commodity platforms running open but slow implementations, or proprietary hardware running closed but fast implementations. Most router vendors currently follow the closed-but-fast model, which gives them complete control over system quality, but has become a barrier to innovation.

Recently, major router vendors have initiated programs to provide *open router platforms* (ORPs), which allow third parties to develop software extensions for proprietary hardware. ORPs potentially support faster deployment of novel networking features; for example, one could deploy Stanford's OpenFlow [13] on an ORP.

While the typical vendor's approach to an ORP is to provide a Linux environment running on an x86 processor as part of the platform, the traditional Linux API is the wrong abstraction. These boxes are interesting precisely because they have specialized hardware features that standard Linux does not (should not) support.

We need an ORP API that offers controlled access to these hardware features. Ideally, this API would expose all of the functionality and performance of modern router hardware, while maintaining the useful properties of commodity operating systems: software portability between vendors, isolation between software components, easy management, etc. Such an API would also be the boundary between open-source upper layers, and lower layers that the router vendors insist on maintaining as proprietary trade secrets.

Previously, Handley *et al.* [7, 8] described XORP, an eXtensible Open Router Platform. XORP provides a nice abstraction for building relatively high-performance routers on top of commodity platforms. While XORP potentially could run on a proprietary-hardware open router platform (PHORP), we are not aware of such an implementation. We also believe that XORP's abstractions, such as its Forwarding Engine Abstraction (FEA), expose too little of the power of modern router hardware, and do not sufficiently address the scarcity of certain hardware resources.

In this paper we explore the design requirements for an "Open Router Proprietary-Hardware Abstraction Layer," or Orphal. Orphal's goals include support for portability of third-party components between different proprietary platforms; isolation between these components; exposing as much of the hardware's functionality as possible; and managing scarce hardware resources.

Casado *et al.* [4] argue that software-only routers are not fast enough, network processors are too complex to program, and hardware-based designs (including commodity forwarding chips) have been too inflexible. They propose a redesign of hardware-level forwarding mechanisms to provide a clean, simple, and flexible interface between hardware and software. We agree with them that the best path to flexible *and* efficient routers depends on co-evolving router hardware, extensible router software, and a clean interface between them.

Figure 1 shows how Orphal fits into a PHORP architecture. Orphal sits above the vendor-proprietary hardware and software, and also above the commodity hardware and operating system, although we see no reason to modify the standard OS APIs. (The figure shows Linux as the OS, but it could be any reasonable OS, and perhaps a virtual-machine layer as well.) In practice, Orphal would be implemented as a combination of device

[1] We use "router" and "switch" interchangeably in this paper.

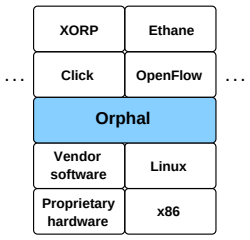

Figure 1: Layering in an open router platform

drivers and user-mode libraries.

One or more *switchlet* modules run above Orphal. In the figure, we show two: a Click [10]+XORP stack, and an OpenFlow [13]+Ethane [3] stack, but these are just examples. This is *not* an "active networks" approach; we expect switchlets to be installed by the router's owner.

This position paper describes some of the design challenges for Orphal. We first describe a high-level overview of a plausible design. Then, for concreteness, we focus on issues related to one particular kind of specialized hardware: Ternary Content Addressable Memories (TCAMs) used for line-rate lookups. This is motivated by our experience porting OpenFlow (see sec. 4).

2 ORPHAL API DESIGN OVERVIEW

Orphal's goals include *resource management; controlled sharing; isolation; hardware reprogrammability; performance; portability;* and *manageability.* Orphal differs from the API of a general-purpose OS mostly because Orphal must expose interesting, router-specific hardware without sacrificing run-time efficiency.

Resource management A high-performance router is inherently a real-time environment, with potentially scarce resources both in the commodity computation platform, and in the proprietary hardware. Routers are often required to enforce QoS requirements, which cannot be maintained if the router itself mis-manages its resources. Orphal needs to support resource management, including allocation of resources among switchlets, consistent with the overall QoS policy and performance constraints of the system.

Which resources need to be managed? We can assume that the commodity OS will manage commodity-hardware resources (CPU, RAM, stable storage), while Orphal will manage router-specific resources such as TCAM entries, hash-table entries, buffer space, programmable ASICs, etc. We also want to manage power-related resources (powering down idle line cards, per-port rate scaling, etc.) using Orphal. One challenge is to define Orphal's resource management so that it is portable across a range of router hardware with various interesting kinds of resources; we believe that this can be done using vendor-specific switchlets that Orphal invokes via upcalls (see section 2.1).

Controlled sharing Orphal must provide controlled sharing of abstract resources such as forwarding-table entries, as well as the real resources (such as hash-table and TCAM entries) used to implement these abstractions.

For example, if two switchlets want to control the actions for packets for a given destination – e.g., a firewall switchlet and a QoS switchlet – how should Orphal decide which switchlets get that control? If two switchlets want to process the same packet, which one gets it first? We believe that prior work on kernel packet filters [15, 20] provides some useful models; for example, Orphal could assign precedence (priority) levels to each switchlet, and let each switchlet declare whether lower-precedence switchlets should see the packets it handles.

Isolation One goal of an ORP is to allow composition of switchlets from different third-party component vendors. While we need not assume that switchlets might be malicious, the potential remains for unexpected "feature interactions." (This is a problem even when all components come from the same vendor.) Two switchlets running on top of Orphal should not accidentally interfere with each other, either directly or indirectly. Thus, the system must prevent switchlets from interfering with each other's code and private state. Isolation is usually accomplished either with a process-like abstraction, or using a virtual machine abstraction. This choice is likely to be made by the router vendor, and Orphal should support either model, as transparently as possible.

Hardware reprogrammability We expect some router platforms to provide *programmable* hardware (not just *configurable* hardware, such as TCAM tables). For example, an ASIC in the packet-processing fast path could support programmability for deep-packet inspection (DPI) operations [9]; NetFPGA [16] is another example. Given these programmable features, should Orphal provide an API allowing switchlets to, for example, push arbitrary microcode into an ASIC, or would it be safer to simply provide access to a platform-defined library of such functions?

Performance Orphal must deal with many performance-related issues, such as support for multi-core parallelism in switchlet execution; prioritizing CPU sharing among switchlets; rate-limiting features of the platform; etc. We have neither the experience nor the space to discuss these further.

Portability Orphal must expose the platform's hardware details enough to support high performance, but without exposing too much detail: that would compromise portability, and perhaps isolation. This is a difficult challenge, especially since we lack enough experience to know what really matters. We describe, in sec. 4, our

initial experiences trying to map Ethane's 10-tuple flow-description model onto a TCAM that supports 5-tuples.

Manageability Routers must already address many management issues, such as port and routing-protocol configuration. The introduction of open router architectures creates a new problem: given a multitude of separately developed switchlets, how does the router administrator create and preserve a stable configuration?

XORP, for example, provides a "router manager process" (rtmgr) [19] to handle some of these issues. Support for proprietary hardware probably complicates this task, because the introduction of a new switchlet can create new resource conflicts (e.g., not enough TCAM entries) and new feature interactions (competing uses for a given TCAM entry).

We believe the router manager will have to check that the system can support the switchlet's minimal requirements (e.g., that there are enough available TCAM entries for the switchlet to function) and to provide rollback to a previous configuration if a new one causes trouble.

The manager will also have to monitor each switchlet's dynamic resource consumption, including specialized hardware resources, so that the router administrator can make informed decisions.

We also expect administrators will want to upgrade a switchlet to a new version without rebooting the entire router. This may require Orphal support, especially to cleanly undo the hardware-related effects of an old (or failed) switchlet. For example, when a switchlet fails or is removed, its updates to the TCAM should be reversed.

2.1 What is a switchlet?

A *switchlet* is simply a module that runs on top of Orphal, with its own address space and thread(s) of control.[2] Orphal will support several switchlet categories, including:

- **per-packet switchlets**: These are invoked, similarly to Click *elements* [10], to handle specific packets. Since high-performance router designs try to avoid handling most packets in software, per-packet switchlets are mostly useful for exceptional packets.
- **per-flow switchlets**: Some router functions, especially for monitoring and sometimes for firewalling, are invoked once or a few times per flow. This is less likely to cause performance problems, although given mean flow lengths in the ballpark of 12 UDP packets to 50 TCP packets [2], such switchlets might still be reserved for exceptions.
- **control-plane functions**: These functions, such as routing protocols, management protocols, etc., typically are not directly related to the packet-forwarding fast path, and so are often handled in software. XORP provides a useful framework for these functions.
- **optimizer/helper modules**: We expect that the process of matching higher-level abstractions needed by switchlets to the lower-level hardware abstractions will require the use of optimization algorithms. Orphal invokes these via upcalls to optimizer switchlets. This form of policy-mechanism separation allows third parties to develop improved versions of these modules.
  Optimizer modules can also be used, for example, to provide a backing store for space-limited hardware resources. For example, Orphal could manage the hardware TCAM as a cache for a larger table managed by an optimizer module, in much the same way that an OS kernel manages a hardware Translation Buffer as a cache for its page tables.
  Additional "helper" switchlets can be used to provide policy-mechanism separation for functions such as detecting inter-switchlet conflicts in TCAM entries.

Orphal needs to balance switchlet portability against aggressive use of hardware functions that might not be present on all platforms. Thus, a switchlet can provide an optional software implementation for a function, to be used if Orphal cannot provide the necessary hardware support (either because it isn't there, or because it is oversubscribed).

For example, consider a Click module, such as the existing NetFlow package, that is most naturally implemented in hardware if the hardware is available. The module author could supply both a hardware-based (e.g., NetFPGA) version and a (less efficient) software-based version, and Orphal could transparently instantiate the most efficient version possible. (This leaves open the question of whether Orphal could feasibly change between versions dynamically; state synchronization and QoS maintenance might make this difficult.)

2.2 Example of Switchlets

We describe our initial experience implementing OpenFlow, and how it might be structured as switchlet, in sec. 4. Beyond that, we lack space to give detailed examples of possible switchlets, but here is a partial list:

- **Specialized firewall** switchlets could be triggered by DPI hardware to check unusual flows against security policies.
- **Specialized monitoring** switchlets could report on suspicious patterns of flow creations.
- **NAT** switchlets might require access to programmable packet-header rewriting hardware.
- **Dynamic VLAN** switchlets could implement setup protocols used to establish VLAN membership.

[2] Others have defined "switchlet" in different ways [1, 5, 17], but we can't think of a better term.

3 API DESIGN ISSUES

The goal of Orphal is to provide a clean interface between router-specific programmable hardware, and switchlets running on general-purpose CPUs within the router platform. Routers often have a number of interesting hardware features, such as programmable DPI engines, TCAMs for route lookups, and other route-lookup hardware such as hash tables and programmable header extractors. Future routers might have additional specialized hardware, such as programmable packet-header rewriters.

In this paper we limit our detailed discussion to TCAMs, since they are widely used for high-speed forwarding, present some interesting challenges, and are the focus of our current implementation work (see sec. 4).

3.1 TCAM API and Resource Management

Most high-performance router hardware includes Ternary Content Addressable Memories (TCAMs). One can think of a CAM as a table whose rows each include a tag field to match against; the CAM returns the matching row (if any). In a TCAM, tag-field entries are composed not just of binary 1s and 0s, but also "X" or "don't care" values. TCAMs thus allow more compact representations of lookup tables whose tag values can include wildcards. Routers use TCAMs for functions such as IP address lookups and firewall lookups, where these wildcards are common.

While TCAMs are often the preferred solution for lookup functions, various TCAM parameters are constrained by expense (TCAM structures take a lot of die area) and power consumption (a TCAM lookup requires all rows to be active, and TCAMs consume ca. 15W/chip [21].) Thus, TCAMs present some challenges for an open router platform, and we explore these as an example of a larger set of challenges that the API must meet:

- **Limited tag-field size**: TCAM tag widths are typically limited, often to ca. 144 bits (enough for an IP/TCP 5-tuple) [14]. A single TCAM entry might therefore be insufficient to support a firewall-entry match in a single lookup, since (especially with IPv6), too many packet-header bits must be checked. This can force the hardware to support multiple lookups per packet. The API must allow switchlets to express such multi-lookup rules.
- **Limited number of rows**: TCAMs are typically limited to a few thousand rows. Thus, the platform must treat TCAM rows as a scarce resource, to be allocated among potentially competing switchlets, and the API must allow switchlets to express resource requirements.
- **Multiple "owners" for one row**: Two different switchlets might want packets that match the same TCAM row (e.g., "all TCP packets to port 80"); the API needs to manage these conflicts. (See sec. 3.4.)
- **Multiple matching rows**: Because TCAMs support wildcards, two different rows might match the same packet. But TCAM-based designs always return the lowest-index entry that matches the packet. Two switchlets might create distinct TCAM entries that either overlap, or where one covers the other; what should the system do in this case? The API needs to manage these conflicts, too. (See sec. 3.4.)
- **TCAM optimization**: Given an abstract set of matching rules, one can generate an optimized set of TCAM entries that provide the most compact (and hence most space- and energy-efficient) representation [12, 14].
- **TCAM update (insertion) costs**: TCAM-based designs generally must trade off efficient lookups against insertion costs, which can be as high as $O(n)$ in the number of rows [6]. The API might need to manage this tradeoff; it might also need to synchronize between updates and lookups (or else lookups could yield bad results during updates) [18].

3.2 A typical TCAM-based hardware design

Figure 2 sketches part of an idealized TCAM-based hardware design, to make some of these design challenges concrete. Each line card would have an instance, possibly serving several ports.

An incoming packet is first processed by a **pseudo-header generator**, adding to the real packet header such fields as a VLAN tag, the ID of the port where the packet arrived, etc. Assuming that the TCAM is not wide enough to do a full lookup in one step, the **header extractor** manages a multi-stage lookup; it recognizes certain high-level patterns (e.g., "IPv4 packet" or "IPv6 packet"), extracts the header fields used in each stage (e.g., first the layer-2 headers, then the layer 3+4 headers), passes these to the TCAM, and decides whether to do the next lookup stage.

Many routers use one or more **hash tables** in addition to the TCAM. Hash tables provide a cheaper mechanism for doing exact-match lookups, such as "what's the next hop for this flow?", while TCAMs are appropriate for more complex lookups – especially those including wildcards – typical of QoS and firewall (access control list) functions. For firewall functions, the line card might also include a **port-range classifier**, since arbitrary ranges of port numbers (e.g., "1023–65535") could consume too many TCAM entries. Liu [11] described a range classifier that uses a modest-sized RAM-based lookup table.

Additional **sequencer/combiner** logic coordinates the multiple lookup stages and combines partial results to

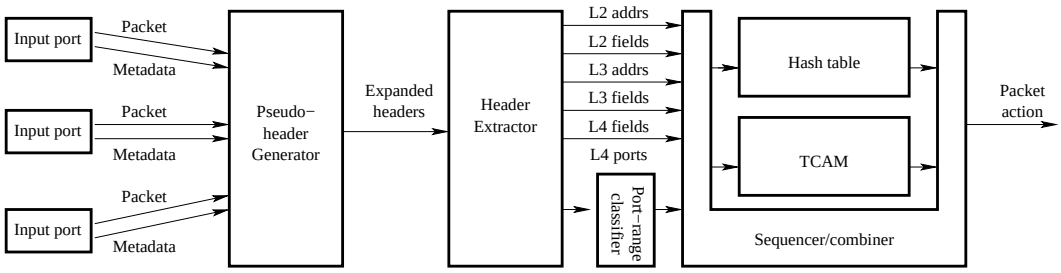

Figure 2: Idealized TCAM-based lookup path

generate a final result, indicating the action to take with the packet, such as the next-hop address and the output switch port.

The TCAM, of course, is a programmable resource, but potentially so are the other functional blocks (pseudo-header generator, header extractor, port-range classifier, hash table, sequencer/combiner).

Unlike a more abstract API such as XORP, Orphal exposes all of these distinct programmable resources, since they have differing characteristics that could be exploited by sophisticated switchlets.

3.3 What should the TCAM API expose?

There are many ways to organize TCAMs and the associated hardware, and if switchlets are to be portable between hardware platforms, the API must either hide this variation, or expose it in a useful way. Given the challenges listed in section 3.1 (and there are others), perhaps it is implausible to create an API that provides any generality across models and vendors. However, we suspect that by choosing the right level of abstraction for exposing the TCAM hardware, Orphal can meet its goals.

For example, XORP exposes a high-level "forwarding engine abstraction" (FEA), but Orphal must expose a lower-level abstraction if the switchlets are to exploit specialized hardware features. There are things that cannot be expressed explicitly at the FEA level – for example, that certain rules should be stored in the hash table instead of the TCAM.

There is a useful API abstraction intermediate between a raw-hardware "TCAM row" and a high-level "forwarding table entry." Although a TCAM optimizer module will need access to the raw row-level version ("put these bits here"), most switchlets will use a paravirtualized view of the TCAM (PV-TCAM), which will enable Orphal to provide the controlled sharing, isolation, and resource management properties described in section 2. PV-TCAM rows look almost like real TCAM rows, but with some additional meta-information, and without a fixed mapping to actual hardware rows.

The TCAM-user API will need to provide certain functions, including (among many others):

- **tcamAddRow(tag, action, ordering)**: Used to add a row with a given tag value and action, and an intra-switchlet value to control how rules are ordered. Returns either an opaque handle for the row, or a failure indication.
- **tcamDeleteRow(handle)**: does the obvious thing.
- **tcamGetRow(handle)**: returns the corresponding TCAM entry, including statistics.
- **tcamRegisterInterest(handle, callbackFunction)**: specifies a switchlet function to be called with each packet that matches the row; the default is no callback. This is the way that switchlets can receive packets and/or discover flows.
- **tcamConflictCallback(handle, callbackFunction)**: If another, higher-priority switchlet creates a TCAM row that conflicts with the one associated with the handle, this callback informs the current switchlet that the row has been reassigned to the other switchlet's purposes. Section 3.4 discusses conflicts in more detail.

The TCAM-optimizer API will need to provide certain functions, including (among many others):

- **Loading a set of TCAM rows**: The optimizer's output needs to be loaded into the TCAM; possibly this will require some synchronization so that packets are not processed when the TCAM is in an inconsistent state.
- **Obtaining the abstract state of the TCAM database**: The optimizer's input from Orphal will consist primarily of the union of the TCAM-user requests, plus some policy settings provided by a management layer.
- **TCAM usage statistics**: Typically, TCAMs support hit counters for each row.

3.4 TCAM row conflicts

Multiple switchlets might try to create conflicting TCAM rows. Orphal's approach is to detect these conflicts and resolve them using an inter-switchlet priority ranking. (This seems like the simplest approach, but we are exploring others.) When a low-ranking switchlet tries to create a new row that conflicts, Orphal simply rejects the attempt. However, a high-ranking switchlet can create a row that conflicts with an existing lower-ranking row, in

which case Orphal removes the low-ranking row, inserts the new one, and informs (via **tcamConflictCallback**) the low-ranking switchlet that it has lost the row. Orphal lets the switchlets figure out what to do in that case.

It is not easy to define what a "conflict" is, and conflict-checking is an expensive (NP-complete) process [12], so checking should not be embedded in Orphal *per se*. Instead, Orphal supports plug-in conflict-checking implementations using "helper" switchlets.

4 OUR EXPERIENCE WITH OPENFLOW

OpenFlow [13] is a centrally-managed flow-based network where switches are simple forwarding engines that classify packets into flows and act on them according to a policy supplied by a central controller. We are porting OpenFlow to a commercial switch, the ProCurve model 5406ZL, and here report some of the challenges.

OpenFlow could run entirely in the switch's software, but that would not support line-rate forwarding, so we need to use the TCAM hardware. The controller expects a flexible flow classifier, so the tricky part is to match OpenFlow's flow descriptions (a 10-tuple of physical ingress port and VLAN IDs; Ethernet source, destination and type; and the standard IP/TCP 5-tuple) with what the hardware supports. The challenges include:

- **Limited number of TCAM rows:** means not all flows can be classified in hardware. So, we insert a final wild card entry in the TCAM to divert packets from other flows to the software stack. We try to minimize such slow-path packets by keeping busy flows in the TCAM.
- **Limited tag-field size:** TCAM widths (e.g., 144 bits) are typically chosen to support lookup on the IP/TCP 5-tuple ($32+32+16+16+8=104$ bits). OpenFlow's 10-tuple, which includes 48-bit MAC addresses, is too big for such TCAMs. However, our switch supports multiple TCAM lookups/packet at line rates, so we support the OpenFlow tuple with a multi-stage lookup.

When a packet arrives for an unknown flow, the OpenFlow forwards it to the central controller, which updates that switch (and perhaps others) with new flow-specific forwarding rules. Using Orphal, we could implement OpenFlow as a switchlet that forwards no-match packets to the controller, and installs controller-supplied responses into the forwarding table. The controller deals in 10-tuples; we intend to use a helper switchlet to convert these into patterns that the switch's TCAM can handle. This helper could also be used by other switchlets, such as firewalls.

5 SUMMARY

Open router platforms offer tremendous flexibility, but exploiting the rich variety of router hardware creates complexity. Our goal for Orphal is to tame that complexity; we hope to demonstrate working systems in the near future.

REFERENCES

[1] D. S. Alexander and J. M. Smith. The Architecture of ALIEN. In *Proc. Intl. Working Conf. on Active Networks*, pages 1–12, 1999.

[2] M. Arlitt. Personal communication, 2008.

[3] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: taking control of the enterprise. In *Proc. SIGCOMM*, pages 1–12, Aug. 2007.

[4] M. Casado, T. Koponen, D. Moon, and S. Shenker. Rethinking Packet Forwarding Hardware. In *Proc. HotNets*, Oct. 2008.

[5] N. da Fonseca, J. Castro, A.P., and A. Rios. A procedure for resource allocation in switchlet networks. In *Proc. GLOBECOM*, volume 2, pages 1885–1888, Nov. 2002.

[6] B. Gamache, Z. Pfeffer, and S. P. Khatri. A fast ternary CAM design for IP networking applications. In *Proc. ICCCN*, pages 434–439, Oct. 2003.

[7] M. Handley, O. Hodson, and E. Kohler. XORP: an open platform for network research. *SIGCOMM CCR*, 33(1):53–57, 2003.

[8] M. Handley, E. Kohler, A. Ghosh, O. Hodson, and P. Radoslavov. Designing extensible IP router software. In *Proc. NSDI*, pages 189–202, Boston, MA, 2005.

[9] HP ProCurve. ProVision^{TM} ASIC: Built for the future. `http://www.hp.com/rnd/itmgrnews/built_for_future.htm`.

[10] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click Modular Router. *TOCS*, 18(3):263–297, 2000.

[11] H. Liu. Efficient Mapping of Range Classifier into Ternary-CAM. In *Proc. Hot Interconnects*, pages 95–100, Aug. 2002.

[12] R. McGeer and P. Yalagandula. Minimizing Rulesets for TCAM Implementation. Tech. Rep. HPL-2008-106, HP Labs, 2008.

[13] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM CCR*, 38(2):69–74, 2008.

[14] C. R. Meiners, A. X., and L. E. Torng. Algorithmic Approaches to Redesigning TCAM-Based Systems. In *Proc. SIGMETRICS*, June 2008.

[15] J. Mogul, R. Rashid, and M. Accetta. The packer filter: an efficient mechanism for user-level network code. In *Proc. SOSP*, pages 39–51, 1987.

[16] J. Naous, G. Gibb, S. Bolouki, and N. McKeown. A Programming Model for Reusable Hardware in NetFPGA. In *Proc. PRESTO*, Aug. 2008.

[17] J. E. van der Merwe and I. M. Leslie. Switchlets and Dynamic Virtual ATM Networks. In *Proc. 5th IFIP/IEEE Intl. Symp. on Integrated Network Management*, pages 355–368, 1997.

[18] Z. Wang, H. Che, and S. K. Das. CoPTUA: Consistent Policy Table Update Algorithm for TCAM without Locking. *IEEE Trans. Comput.*, 53(12):1602–1614, 2004.

[19] XORP Project. XORP Router Manager Process (rtrmgr) Version 1.4. `http://www.xorp.org/releases/1.4/docs/rtrmgr/rtrmgr.pdf`, 2007.

[20] M. Yuhara, B. N. Bershad, C. Maeda, and J. E. B. Moss. Efficient packet demultiplexing for multiple endpoints and large messages. In *Proc. USENIX Winter Tech. Conf.*, pages 153–165, 1994.

[21] F. Zane, G. Narlikar, and A. Basu. Coolcams: power-efficient TCAMs for forwarding engines. In *Proc. INFOCOM*, volume 1, pages 42–52, 2003.