Building Extensible Networks with Rule-Based Forwarding

Lucian Popa[*†] Norbert Egi[‡] Sylvia Ratnasamy[§] Ion Stoica[*]

Abstract

We present a network design that provides flexible and policy-compliant forwarding. Our proposal centers around a new architectural concept: that of packet *rules*. A rule is a simple if-then-else construct that describes the manner in which the network should – or should not – forward packets. A packet identifies the rule by which it is to be forwarded and routers forward each packet in accordance with its associated rule. Each packet rule is certified, guaranteeing that all parties involved in forwarding a packet agree with the packet's rule. Packets containing uncertified rules are simply dropped in the network. We present the design, implementation and evaluation of a Rule-Based Forwarding (RBF) architecture. We demonstrate flexibility by illustrating how RBF supports a variety of use cases including content caching, middlebox selection and DDoS protection. Using our prototype router implementation we show that the overhead RBF imposes is within the capabilities of modern network equipment.

1 Introduction

A central component of a network design is its forwarding architecture that determines the manner in which packets are transported between two endpoints. Today's Internet offers users a simple forwarding model: a user hands the network a packet with a destination address and the network makes a best-effort attempt to deliver the packet to the destination. Although simple, this architecture is also fairly limited and there have been repeated calls to extend the Internet's forwarding architecture for greater *flexibility*—allowing, for example, the user to select the path his packets should traverse [20, 44, 47, 49] or to specify whether packets can/should be processed by middleboxes and active routers [47, 49, 29, 48, 25].

Achieving a flexible forwarding architecture has thus been a long-held, if elusive, goal of Internet research [47, 49, 29, 20, 48, 25, 40]. Our work in this paper shares this goal. Our point of divergence from prior efforts starts with the observation that forwarding flexibility is inherently coupled with issues of *policy*.

Our thesis is that achieving flexibility is not just a matter of augmenting packets with more expressive forwarding directives that routers execute. Rather, in addition, *for each forwarding directive that enhances flexibility, the parties involved in forwarding should be able to set policies that constrain that directive*. By the policy of entity A (host, middlebox operator or ISP) we refer to the decision whether to approve or reject a forwarding directive based on A's business or technical goals. By forwarding directive we refer to instructions provided by endpoints to routers and middleboxes on how to forward their packets. For example, a forwarding directive could specify that sender S can forward its packets through middlebox M before reaching destination D. An example of policy would be M refusing to accept packets from S.

To better illustrate our thesis, consider its application to the Internet. Since the main forwarding directive in IP is for sender S to send packets to destination D, D should be able to specify that the traffic from S should not reach it, *i.e.,* either by explicitly allowing or denying packets from D. Unfortunately, IP does not provide such functionality, effectively leaving the end-hosts vulnerable to DoS attacks. Unsurprisingly, this lack of functionality has been identified as one of the main security vulnerabilities of the Internet, and several solutions have been proposed to address this limitation [51, 52, 21, 32, 37, 22].

Of course, forwarding directives and policies are only as good as the ability of the network to enforce them and to guarantee their authenticity. What complicates policy enforcement is the involvement of multiple parties in achieving the packet's flexible behavior—the network service providers along the path, potential middlebox operators and, of course, the source and destination. As such, the network must ensure that a packet's forwarding directive complies with the policies of *all* parties involved. In our previous middlebox example, the network must ensure that M is willing to relay packets from S to D. If M does not approve, the network should simply drop the packets before reaching M.

In this paper, we propose a new *rule-based* forwarding architecture, RBF, that treats flexibility and policy enforcement as equal design goals. RBF is based on a new architectural concept – that of packet *rules*. In RBF, instead of sending packets to a destination (IP) address, end-hosts send packets to a rule. Rules are created by destinations. A sender fetches the destination's rule from a DNS-like infrastructure and inserts it in the packets sent to that destination.

[*]University of California, Berkeley
[†]ICSI, Berkeley
[‡]Lancaster University
[§]Intel Labs, Berkeley

A rule is a simple `if-then-else` construct that describes the manner in which the network should – or should not – forward packets. For example, a destination A can receive packets only from source S using the rule:

R_A: `if (pkt.source` $\neq$ `S) drop pkt`

Or a mobile client B might route certain video content through a 3rd-party transcoding proxy with:

R_B: `if (pkt.URL` $=$ `hulu.com) sendPktTo trnscdPrxy`

The above examples are anecdotal (we present precise syntax and additional examples in §3) but serve to illustrate how destinations can control and customize how the network forwards their packets in a manner not easily accommodated by current IP. In effect, with rules, a receiving host must specify both *which* packets it is willing to receive as well as *how* it wants these packets forwarded and processed by the network.

The rule-based architecture we develop offers the following properties:

Rules are mandatory: routers drop packets without rules

Rules are provably authorized: all recipients (end-hosts, middleboxes and/or routers) named in the rule must explicitly agree to receive the associated packet(s). Routers, middleboxes and end-users can verify a rule's authorization.

Rules are provably safe: rules cannot exhaust network resources; *e.g.,* rules cannot compromise or corrupt routers nor cause forwarding loops.

Rules allow flexible forwarding: rules are a (constrained) program that allows a user to "customize" how the network forwards its packets.

The first two properties assist in policy enforcement by ensuring a packet is only forwarded if explicitly cleared by all recipients (*i.e.,* if it conforms with the policies of all recipients) specified in the rule. Since RBF defines policies on rules, any recipient will have the ultimate say on whether to accept any rule that contains forwarding directives sending packets to it. Since all forwarding directives are encoded into rules, we achieve our goal of enabling any entity affected by a forwarding directive to constrain that directive.

The third property ensures rules cannot be (mis)used to attack the network itself. As we shall show, the last property provides flexibility since users can give the network fine-grained instructions on how to handle their packets, enabling: explicit use of in-network functionality at middleboxes and routers, loose path forwarding, multi-path forwarding, anycast, multicast, mobility, filtering of undesired senders/ports/protocols, recording of on-path information, *etc.* In the remainder of this paper, we present the design, implementation and evaluation of a forwarding architecture that meets the above properties.

RBF relates to an extensive body of work on both forwarding flexibility and policy enforcement. We discuss related work in detail later in this paper and here only note that, at a high level, we believe what distinguishes RBF is its focus on *simultaneously* supporting flexibility and the multi-party policy requirements that such flexibility implies. As we shall see, this goal leads us to a design that differs significantly from prior proposals.

Finally, we note up-front that RBF is more complex than the existing IP forwarding architecture, which is frequently cited for its simplicity. In addition, RBF relies on strong assumptions such as anti-spoofing, the existence of rule-certifying authorities and a DNS-like infrastructure to distribute rules. The gain, relative to today's IP forwarding, is significantly improved flexibility and security; we posit that the greater complexity of our solution is a perhaps inevitable consequence of this richer service model.

2 Design Rationale and Overview

We start with the goal of network flexibility and allowing users control over how the network processes their packets. The abstraction that perhaps best supports flexibility is simply that of a *program*, leading to an architecture where users write packet-processing programs that routers execute. This vision of code-carrying packets is, of course, the cornerstone of active networking [48, 50] and we borrow this as our starting point in designing RBF. However, as we shall see, RBF severely dials back on the full-fledged generality of the original active networks' vision to arrive at a significantly simpler and safer architecture.

Rules are thus a form of program. The challenge then is to appropriately constrain these programs/rules to ensure that they cannot harm the network or other hosts. The key insight behind RBF is that these constraints must extend along *two* dimensions. First, rules must be *safe*, *i.e.,* guaranteed not to corrupt or exhaust network resources. In addition, however, we must constrain rules to respect the *policies* of all stakeholders involved—source, destination, middleboxes and ISPs. This latter requirement is unique and yet critical to networking contexts but was under appreciated in early active networking proposals.

To address policy safety, RBF incorporates two key design decisions:

(D1) Layering: we believe network operators will be unwilling to relinquish control of route discovery and computation and hence we layer RBF above current IP forwarding and do not allow rules to modify the IP-layer forwarding information base (FIB).

(D2) Verifiable stakeholder agreement: we require that a rule be authorized by all entities it explicitly names (*e.g.,* destination, middleboxes or routers). This ensures agreement of the stakeholders' policies with

the rule's intent; in particular it also ensures that rules cannot violate ISPs' routing policies, since providers must explicitly agree to have their routers named in rules. To achieve this property, in RBF rules are certified by trusted third parties, which in turn gather proofs of policy compliance from each of the rule participants.

To address rule safety, we impose strict constraints on rule syntax, such that safety can be verified through simple static analysis:

(C1) Rules cannot directly modify router state. This avoids corruption of router state. However, this can be a limiting restriction, particularly to network operators who wish to expose in-network services such as caching or monitoring to end users. To accommodate this, RBF allows operators to deploy specialized packet-processing functions at their routers and allows rules to *invoke* these functions. Such "router-defined functions" do allow rules to update router state, but only indirectly via code installed, and hence presumably trusted, by operators. This model for router-defined functions thus represents a middle ground in the tradeoff between flexibility and safety.

(C2) The rule "instruction set" is limited to only four possible *action* statements: (a) `forward` the packet to the underlying IP layer, (b) `invoke` a router-defined function, (c) `modify` the packet header and (d) `drop` the packet, plus conditionals that determine whether an action should be taken based on reading packet headers and router state. Note that there is no action that allows backward jumps across rule statements. This prevents looping or resource exhaustion at routers and ensures execution time is linear in program size.

The above constraints represent a stark departure from the rich generality of the active networks vision. Indeed, rules are more a sequence of packet steering directives, rather than a full-fledged program. The benefit is *verifiable rule and policy safety*. Moreover we find that, despite these constraints, rules suffice to express a wide variety of forwarding behaviors as we will later illustrate.

2.1 Architecture Overview and Assumptions

We now provide a brief overview of the main components and assumptions of an RBF architecture. Figure 1 illustrates the forwarding architecture of an RBF-enabled router. On receiving a packet, the router hands it to the *rule forwarding engine*, which processes the packet's rule. Such processing may involve reading router state that the network operator has opted to expose; we term such state *router attributes*. Based on information in the packet header (*packet attributes*) and router attributes, the rule forwarding engine may update the packet's attributes (including its destination), invoke router functions, drop the packet and/or hand the packet to the underlying IP forwarding engine. Recall that for safety reasons the rule is not allowed to update router state.

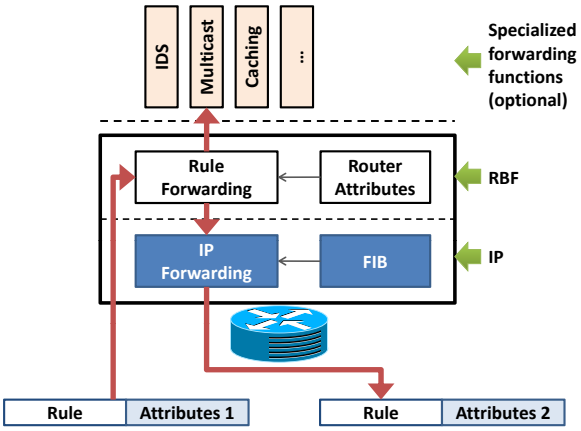


Figure 1: RBF router and rule forwarding

The design of a rule-based architecture involves the design of rules themselves as well as the surrounding infrastructure required to support the distribution, processing and securing of rules. Consequently, the RBF architecture consists of four main components:

- The specification of packet *rules* – their syntax, packet encoding, constraints on what rules can and cannot do.
- Certificate authorities called *Rule Certification Entities* (RCEs) that certify rules after checking that they are well formed, and that every destination specified in the rule agrees with (*i.e.*, has signed) the rule.
- *Modified IP routers* that verify rule certificates and process packets as described above.
- A *modified DNS infrastructure* that either directly resolves a host D's domain name to D's rule, or resolves D's domain name to another rule resolution server which in turn provides D's rule.

Assumptions: RBF builds on three major assumptions.

First, RBF assumes the existence of an anti-spoofing mechanism. This is required because rules may use source and destination IP addresses in their decision process and hence addresses must be legitimate, otherwise policy compliance cannot be enforced.[1] In this paper we assume the use of ingress filtering, although RBF can accommodate alternate solutions, *e.g.*, Passports [36]. The rationale behind our choice of ingress filtering is described in §4.

Second, we assume routers know the public keys of RCEs and can thus verify rule certificates. We assume the number of RCE organizations is relatively small and these keys can be statically configured at routers, akin to

[1]Note that any solution for blocking undesired traffic inside the network requires a way to identify sources. Anti-spoofing identifies users based on their addresses. An alternative, is to identify users by their access path [51, 52], but this approach ties communications to a specific path restricting flexibility (*e.g.*, for mobility, traffic engineering, multi-path forwarding).

how browsers today are configured with the list of major certificate authorities.[2] Note that although we assume a small number of RCE organizations, we envisage each organization will run geographically replicated instances of their service for improved scalability and robustness.

Finally, we assume that the rule resolution infrastructure (whether DNS or the resolution servers the DNS points to) is well provisioned, akin to how major Internet services (Google, DNS, Amazon) operate today, relying on engineering approaches such as maintaining a presence at major ISPs, IP anycasting, bandwidth provisioning, and so forth. As described in §4, we make this assumption to protect against "denial of rule" attacks.

Clearly, these assumptions are significant and may impede an immediate deployment of RBF in practice. And even with these assumptions, the resulting RBF design is far from trivial (for this reason, we in fact offload some of the details to an extended technical report [42]). However, we hope through the design presented in this paper to start a focused discussion about how best to practically introduce flexibility and security into the Internet and about what set of primitives routers must support to achieve this goal. In this paper we present one solution to this problem; in §10 we succinctly discuss the arguments that have led us to these specific assumptions and design.

3 The RBF Data Plane

In this section we describe the key components of the RBF data plane: rule syntax and how routers verify and execute rules. We then present examples of how rules are used.

3.1 Rule Specification

RBF represents a rule as a sequence of actions that can be conditioned by *if-then-else* instructions:

```
if (<CONDITION>) ACTION1
else ACTION2
```

Conditions are *comparison operators* applied to packet and router attributes. An action can be one of:

1. `forward` the packet to the underlying IP engine;
2. `invoke` a local function available at the router;
3. `update` the value of the packet attributes;
4. `drop` the packet.

Packet attributes include the standard IP header five-tuple (IP addresses, ports, protocol type) and, optionally, a number of custom attributes with user-defined semantics. For simplicity, RBF does not allow rules to dynamically add new attributes. Router attributes may include, for example, the router's IP address, AS number, link congestion levels, and flags indicating whether the router implements a specific function (*e.g.*, a rule can check `router.local_cache` to discover whether the router maintains a local content cache). Rules are allowed to update packet attributes, but not router attributes.

Each rule has an associated *lease* that ensures the rule can only be used for a limited period of time (§4.3). Also, every rule has an identifier (ID) defined as the concatenation of a hash of the rule owner's public key and an index unique to the owner, *hash(PK_owner):index*. In Section 7 we present an optimization to reduce packet overhead and identify most rules by using a hash over their content. This optimization can be used in the common case when there is no need for multiple rules with the same identifier; for example, mobile hosts may require different rules with the same identifier (see §3.4).

The following is an example of a rule that forwards a packet to destination `D` via a waypoint router `R1`; a packet attribute `visitedR1` indicates whether the packet has already visited `R1`:

```
R_D:
  if(packet.visitedR1 == FALSE) //from src. to R1
    if(router.address != R1)
      sendto R1
    else packet.visitedR1 = TRUE //to D
  if(packet.visitedR1)
    sendto D
```

where `sendto` involves setting the IP destination address to `D` and then handing the packet to the underlying IP forwarding engine (assuming, of course, that `D` is not the local address). Rule execution terminates at a `sendto` or `drop` action; the packet is dropped if the rule does not arrive at an explicit `sendto`. Finally, rules can `invoke` local functions at the router; after the invocation the packet is returned to the forwarding layer.

3.2 Distributing Rules to Routers

To forward a packet, a router must first obtain its rule. There are two potential approaches: (1) rules are carried in packets, (2) routers use an out-of-band mechanism to obtain rules. In RBF, we choose to carry rules in packets since the second approach would require complex rule distribution and storage protocols, and would incur extra delays in communication setup (in fact this approach would likely require special "rule-less" traffic to install rules). The tradeoff is higher overhead on the data path as rules increase packet size and routers must verify each packet's certificate; our evaluation in §7 suggests this overhead is acceptable given the capabilities of modern network equipment.

A packet with source `S` and destination `D` must include a *destination rule*, `R_D`, which is the rule specified and owned by `D`. In addition, a packet may include a *return rule*; this is the rule specified and owned by `S` and is used for return traffic from `D` to `S`.

[2] Some may regard this model of security unsatisfactory, we discuss alternatives to this deployment in §4.

3.3 Rule Verification

As mentioned earlier, rules are certified by a Rule Certification Entity (RCE) and all packets carry a signature that routers must verify. The verification load at routers is eased by two factors. First, only routers at trust boundaries need to verify rules. Second, routers can cache verification results by maintaining a hash of the rule and its signature. With caching, the full signature verification is only required for the first packet forwarded on a new rule (as long as the verification result is cached). Thus, verifications can be limited only to border routers and, assuming a large enough cache, the verification rate is given by the arrival rate of packets with new rules. By contrast, the signature length adds to the overhead of *every* packet.

Different cryptographic solutions offer different tradeoffs between signature length, signing time (incurred only at RCEs), verification time (incurred at routers) and security. Our current RBF design assumes Elliptical Curve Cryptography (ECC) because ECC signatures are shorter than RSA ones, while exhibiting similar security properties. At the same time, verification time in ECC is typically longer than RSA's. However, in practice verification can be accelerated using ASIC-based implementations or dedicated specialized co-processors. Such implementations are already commercially available [5, 7, 8] and incorporated into network appliances and routers. Furthermore, traffic measurements [4] show that new flow arrivals represent less than 1% of the link capacity on average and less than 5% of the total number of packets, a volume that can be accommodated using commercial ECC modules [5, 7] or recent research proposals [53, 34]. We evaluate different signature mechanisms briefly in § 7 and in greater detail in [42].

3.4 Examples of RBF usage

To illustrate the application of rules, we present a series of example usage scenarios; the rule syntax in these examples is largely identical to the high-level rule language supported by our RBF prototype router (§6), with simplifications for readability as appropriate.

Port-based filtering: A web server, `D`, uses the following simple rule to ensure it only receives packets on port 80:

```
R_filter_port:
  if(packet.dst_port != 80) drop;
  sendto D
```

Middlebox Support: In addition to accepting traffic directly on port 80, `D` might use the following rule to route all other incoming traffic through a packet scrubber [2, 6]. This functionality can be deployed either by `D`'s provider (as a router function), or by a third party (at a middlebox `Scrb`) as presented below:[3]

```
R_mbox_port:
  if(packet.dst_port == 80)
    sendto D                        //directly to D
  else
    if(packet.scrubbed == FALSE)    //before scrubber
      if(router.address != Scrb)
        sendto Scrb
      else                          //at scrubber
        packet.scrubbed = TRUE      //mark scrubbed
        invoke Scrb_service         //scrub
    else
      sendto D                      //after scrubber
```

Thus, similar to previous proposals [47, 49], RBF provides explicit support for middleboxes such as WAN optimizers, proxies, caches, encryption engines, transcoders, SSL offloaders, intrusion detection, *etc*.

Secure Middlebox Traversal: In the previous example, an attacker can directly send a packet with the attribute values set so as to appear that the packet has already visited the middlebox. More generally, one should be able to enforce that rule directives are respected when the rule participants (sources, middleboxes) are not trusted.

One approach to protect against this behavior is to leverage RBF's assumption that sources cannot spoof their addresses. More specifically, after each middlebox the rule can verify that the packet has indeed been sent by the required middlebox, since middleboxes/waypoints need to set the (non-spoofable) source address attribute in packets (for brevity we omit this in the presented examples); see [42] for more details on this approach.

In an alternate approach, special cryptographic functions deployed at middleboxes and destinations can be used to create/verify proofs guaranteeing the packet has visited the middlebox, as follows:

```
R_mbox_port_crypto:
  if(packet.dst_port == 80)
    sendto D                        //directly to D
  else
    if(packet.proven == FALSE)
      if(router.address != Scrb)    //before scrubber
        sendto Scrb
      else                          //at scrubber
        if(packet.scrubbed == FALSE)
          packet.scrubbed = TRUE
          invoke Scrb_service       //(1) scrub
        else                        //scrubbed
          packet.proven = TRUE
          invoke Prove              //(2) create proof
    else                            //proven
      if(router.address != D)
        sendto D
      else
        invoke VerifyAndDeliver     //check proof at D
```

In this example, the `Prove` function at the middlebox signs the immutable part of the packet header and/or payload, and adds this signature as an attribute to the packet header. In turn, the `VerifyAndDeliver` function at `D` checks the middlebox signature and, if the check succeeds, delivers the packet to the end application. Note that checking the signature requires that `D` knows the public or shared key(s) of middlebox(es); for efficiency, the middlebox could sign the hash chain of a batch of packets.

[3] Note that `Scrb` can represent the address of a load balancer used with several physical middleboxes.

DoS Protection: To protect against DDoS attacks, a server D can create a custom rule for each client that drops packets from any source other than the client. By controlling the number of rules active at a given time, D controls the maximum number of active clients (each rule has an associated lease period). An example of a rule similar to a network capability [52, 51] is:

```
R_filter_src:
  if(packet.source != requester_IP)
    drop;
  ...//rest of the rule
```

Similarly to capability based architectures [52, 51], our solution is based on the premise that destinations are able to grant rules on demand, and that any requester can ask for a destination's rule. In RBF, this task falls to the rule resolution infrastructure and raises the possibility of a "denial of rule" attack on this infrastructure (akin to denial-of-capability attacks in capability-based systems[41]). We present the details of rule resolution and discuss denial-of-rule attacks in §4.

Mobility: Host D changes its network IP address due to physical movement. In RBF, D can continue an existing communication without having to re-establish it. To achieve this, D creates a rule for the new address with the same ID as the rule used in the existing communication, and places it in the packet as the return rule.

Multicast: For security reasons, RBF does not support packet replication, and thus multicast cannot be implemented entirely at the RBF layer. Instead, multicast can be implemented by invoking multicast functionality deployed by ISPs at a subset of their routers; this functionality maintains (soft) state at routers to create a (reverse path) multicast tree. This approach implements essentially an overlay multicast solution, which leverages the IP multicast functionality at on-path routers (see [42] for details).

On-path Caching: Consider an ISP I that deploys caching functionality at some of its (border) routers. A web-service D can contract with I and use this functionality. For this purpose, D creates and publishes the following rule:

```
R_caching:
  if(router.caching_available and
           packet.crt_router != router.address)
    packet.crt_router = router.address
    invoke Caching
  sendto D
```

where the `crt_router` attribute makes sure the caching functionality is called just once at each caching-enhanced router.

In this example, the caching functionality can decide to respond to the requester directly and not forward the packets further to D, which reduces latency for the requester and traffic load at D. A similar rule can support recent proposals for content-centric routing [35, 33].

Other Examples: Our technical report [42] provides examples of applying RBF to a range of additional applications, including: secure loose path forwarding [44, 40], multipath forwarding, network diagnostics, anycast, reverse traceroute (path recording), delay-tolerant networking and even source control over middlebox or path selection. Importantly, these individual examples can be combined as needed. For example, a content distribution network can distribute load among multiple sites using anycast and, at the same time, protect its servers with on-path IDS functionality provided by ISPs.

4 The RBF Control Plane

In RBF, ISPs provide their clients with rules to access the local DNS server and a Rule Certification Entity (RCE), which can certify clients' rules. This information can be provided through a modified DHCP service, similar to the way ISPs or organizations provide the IP address of DNS servers today.

In this section, we describe the RBF mechanisms for rule creation and certification (§4.1), rule distribution (§4.2), lease enforcement (§4.3) and anti-spoofing (§4.4).

4.1 Rule Creation and Certification

To receive traffic, a client must create a rule that allows one or more sources to send traffic to it. Before distributing this rule, the client must ask an RCE to certify it. RCE certification guarantees that rules obey the policies of all stakeholders. In particular, certification guarantees the following properties:

1. Every destination in the rule (*i.e.*, any address that appears as an argument of a `sendto` instruction) has agreed to receive packets using that rule;
2. The operators providing router functions invoked by the rule approve the rule behavior;
3. The rule cannot cause infinite loops;
4. The rule cannot bypass ISP routing policies.

A client can either create rules itself and directly ask an RCE to certify these rules, or use a trusted DHCP-like service to create and certify rules on its behalf. In the remainder of this section we present the former case.

As described above, the ISP provides each client with a rule to access an RCE that has a contract with the ISP. The following example shows a possible rule that allows a client D to access an RCE named C:

$R_{D \rightarrow C}$: `if (source == D) sendto C`

Before certifying a rule, an RCE verifies that the rule has been authorized by each destination that appears in the rule. A client who has created a rule authorizes it by simply signing the rule with its private key. A client that appears in the rule as a destination, other than the rule's creator, will first verify that the content of the rule obeys

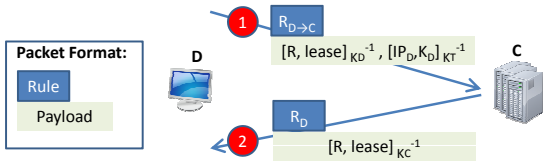


Figure 2: Rule Certification

its policies before signing the rule. For example, an intrusion detection box may verify that the destination indeed belongs to a client allowed to use the service (*e.g.*, based on a contract between the client and the provider of the intrusion detection service), a waypoint router may verify that the final destination is allowed to use source-routing, *etc.*

Let (K_D, K_D^{-1}) denote the (public, private) key pair of client D, and let $\texttt{IP}_D$ be the IP address of D. To prove to an RCE that the client signing the rule with private key K_D^{-1} indeed owns IP address $\texttt{IP}_D$, client D sends a certificate along with the signed rule that binds its public key K_D and IP address $\texttt{IP}_D$. This certificate is signed by an entity T, *i.e.*, $[IP_D, K_D]_{K_T^{-1}}$, where K_T^{-1} represents the private key of T. Clearly, the RCE must trust entity T. In fact, in our solution we will assume that T is itself an RCE.

Next, we present the rule certification process in detail, initially for the case in which the rule has a single destination, and then for the case in which the rule has multiple destinations or waypoints/middleboxes.

Certify single-destination rules: Assume destination D wishes to certify a rule R that forwards packets only to its address $\texttt{IP}_D$, *e.g.*, `R: sendto` $\texttt{IP}_D$. Also, assume D already has a rule $\texttt{R}_D$ on which it can be reached by the RCE C. D obtains this rule as part of the bootstrapping process, which we discuss later.

Fig. 2 shows the certification of D's rule, R, by C:

1. Host D signs rule R with its private key, and sends it to C using rule $\texttt{R}_{D\to C}$. In addition, D sends the certificate binding its public key and address, *i.e.*, $[IP_D, K_D]_{K_T^{-1}}$. Upon receiving this request, C verifies the certificate as well as the signature of the requested rule. These ensure that the request has been made by the owner of $\texttt{K}_D$ and that the requester is also the owner of $\texttt{IP}_D$. In addition, C verifies that R is well formed (see §5).
2. If rule verification succeeds, C signs the rule with its private key and sends it back to D using the return rule in its certification request, $\texttt{R}_D$. At this point, host D can distribute rule R to other hosts directly (as a return rule) or through DNS.

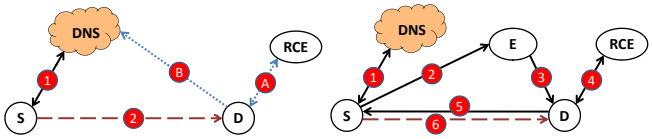
The certification procedure (Fig. 2) needs only to guarantee the authenticity of the request. Since rules are public, confidentiality is not a concern. Since the lease is an absolute value (§4.3), the only effect of replaying rule requests is increased traffic at the RCE. The maximum lease value that C can sign for a rule is negotiated between D's ISP and C. Furthermore, RCEs can limit the number of clients contacting them and can limit each user's certification rate, as we discuss in this section.

Certify multiple destination rules: In this case, every destination (*i.e.*, any host, middlebox, or waypoint router that appears as an argument of a `sendto` instruction) in a rule must agree to receive packets on that rule, *i.e.*, the rule must respect its policies. In particular, every such destination must sign the rule. One of the destinations, D, collects the signatures of all the other destinations along with their certificates binding their public keys to their addresses. D then sends this information to its RCE. In turn, the RCE verifies that all destinations in the rule have signed the rule and sends the signed rule back to D. The lease signed by the RCE has the minimum duration between the requested lease and the leases of all the certificates binding the addresses and the keys of the participants.

Certify rules invoking functions: Operators providing router functions can restrict which rules can invoke these functions. The certification process is similar to certifying multiple destination rules. The identifiers of functions whose invocation requires authorization are represented as hashes of public keys. RCEs certify a rule containing such an invocation only if the rule is signed with the private key corresponding to the function identifier.

Bootstrapping: To certify rules, client D needs to (1) know the rule to contact an RCE, C; (2) provide C with a return rule to receive the certified rule; and (3) obtain the certificate from a trusted authority that signs the binding between D's key K_D and its address IP_D. We assume the ISP provides D with a rule to access an RCE C (similarly to how ISPs today bootstrap clients' access to the DNS). Given this initial rule, we use a simple request-response exchange between the client and the RCE to obtain both the certificate binding the client's IP address to its key as well as its *first* rule. Due to space constraints, we refer the reader to our extended technical report [42] for more details on the bootstrapping process.

RCE load and availability: To control its certification load, an RCE can rate-limit the number of certification requests that it processes from each individual client. Clients are identified by IP address; the anti-spoofing mechanism prevents clients from impersonating each other. Alternatively, clients can be identified by "personalized" rules provided by the ISP to the customer to access the RCE; such rules may have a finer granularity than the anti-spoofing mechanism. RCEs can indirectly protect themselves against link-level DoS attacks by controlling the number of clients under contract.



(a) Regular Distribution (b) DDoS protection

Figure 3: Rule Distribution (solid lines = rule lookup process; dashed lines = data communication; dotted lines = setup)

RCEs must be highly available to enable rule certification at any time. RCEs can meet this requirement by using multiple servers and multiple sites. ISPs and destinations can protect themselves against RCE unavailability by contracting with multiple RCEs.

RCE Key Distribution and Revocation: In this paper we do not explore solutions for the distribution and revocation of RCE keys to routers. Here, we simply mention two possible approaches towards this goal. In one approach, RCE keys could be distributed and revoked using DNSSEC. For example, in the `txt` or other RR type, one DNS entry contains the number of RCEs and, for each RCE, there is one DNS entry (based on its index such as "ID24.rce") that contains the RCE's key. Routers periodically update the RCE keys. In another approach, RCEs could be deployed along AS boundaries, such that each AS would have its own RCE. This approach has the advantage that additional security can be enforced, *e.g.*, the trust in some RCEs can be restricted to their own address ranges. Secure BGP could be used to distribute RCE keys in this case, but at the expense of extra complexity.[4]

4.2 Rule Distribution

RBF uses an extended DNS infrastructure to distribute rules, as illustrated in Fig. 3(a). The destination D creates and certifies a rule for itself (step A) and inserts it into the DNS (step B). A sender S that wants to contact D looks up D's name in the DNS; the DNS is extended to return D's rule rather than its address (step 1). After obtaining a rule to D, S directly sends packets to D (step 2). Note that for practical purposes the rules of the DNS root servers need to have long leases (to avoid tedious reconfiguration or refresh protocols), as with today's long-lived addresses.

In Section 3.4 we pointed out that rules can be used to block DDoS attacks. This relies on (1) the ability to distribute customized rules to different senders (*i.e.*, give a sender S a rule that drops all packets not generated by S) and on (2) the ability to protect the rule distribution itself from DoS attacks.

To protect against DDoS attacks, client D can contract with a large entity E, and redirect its DNS entry to E, by registering E's rule under its DNS name. Fig. 3(b) illustrates this approach. DNS will reply to a lookup for D's name with E's rule (step 1). The DNS entry that contains E's rule must belong to a new type of DNS RR. This new class of entries is returned directly to clients by DNS resolvers. Upon a receipt of such an answer to its DNS query, the requester will continue the DNS lookup by contacting E (step 2). E rate-limits rule requests and forwards them to D (step 3), thus protecting D from DoS attacks. For the authorized requesters, D creates rules (step 4) and replies back to the requesters (step 5). E forwards requests to D conforming to a policy (see §3.4), which can be updated by D at any time.

Note that some malicious users may still get their requests forwarded by E and authorized by D. To alleviate this attack, E can employ fair queuing across senders, and D can blacklist known attackers at E. Such an approach offers a protection similar to network capabilities that apply per-source fair queuing at routers [37].

4.3 Rule Leases

The lease is an expiration time stamp certified along with the rule description. A router drops a packet if its current time exceeds the rule expiration time. For simplicity, in this paper we assume that all routers and RCEs are synchronized via NTP [14] as recommended by router manufacturers [19]. We present a solution that does not rely on global clock synchronization in [42].

4.4 Anti-Spoofing Mechanism

If a source can spoof addresses on packets it sends, it can send packets to a destination D even if the rule does not allow it to, and in this way evade D's policy. Moreover, one can mount a DDoS attack by using a single rule distributed by a malicious source to a set of colluders. To address this problem, RBF can use a previously proposed anti-spoofing mechanism. In this paper, we propose the use of ingress filtering, which is already deployed by over 75% of today's ASes [23]. When deploying RBF, RBF routers could also be used to apply ingress filtering. Note that if malicious ASes do not apply ingress filtering, DoS protection is not fully compromised as only hosts in these ASes can launch attacks.

Instead of ingress filtering, RBF could leverage other anti-spoofing mechanisms such as Passport [36]. However, Passport [36] requires a secure routing layer and incurs extra overhead in packets.

The anti-spoofing mechanism requires middleboxes and routers that change a packet's destination address also to change the packet's source address attribute.

5 Security Analysis

The RBF design aims to achieve the following three goals: (i) *policy enforcement* – ensure that the authorized rules respect the policies of all participants (routers, middleboxes, destinations), and packets with unauthorized

[4]Note that DNSSEC could also potentially be used to distribute keys when RCEs are deployed along AS boundaries.

\Mechanisms Properties\	Certification	Lease	Anti-Spoofing	Static Analysis
No Rule Forging	×	×		
No Rule Tampering	×			
No Rule Evasion		×	×	
Network Safety	×			×

Table 1: Properties and Defense Mechanisms

rules are dropped inside the network; (ii) *rule enforcement* - rules cannot be used by malicous senders and, if senders or rule participants are untrusted, respect of rule directives can be enforced; and (iii) *rule safety* rules cannot be used to attack the network. Next, we summarize RBF's security properties, the threat model and assumptions under which they hold, and the mechanisms that allow RBF to meet these goals. We present a detailed analysis and proofs of RBF's security properties in [42].

Assumptions: We assume that DNS resolution is secure, that distribution of RCE keys to routers is secure, and that RCEs are not malicious.

Attackers: An attacker in RBF can be any host, middlebox, or router: sources can attempt to attack destinations by forging, evading or tampering with their rules; destinations can try to attack the network by creating rules that waste resources and slow down routers; middleboxes and routers can attempt both of these attacks.

Security Properties: We decompose the aforementioned security goals into four specific desired properties:

1. **No Rule Forging:** A host S cannot manufacture a rule that sends packets to another host D, unless D explicitly agrees with this rule, *i.e.,* destinations and middleboxes control the creation of rules that send traffic to them.
2. **No Rule Tampering:** Sources, routers and middleboxes cannot tamper with the destination's rules.
3. **No Rule Evasion:** Host S cannot send packets to destination D, if D's rules do not accept packets from S.
4. **Network Safety:** A destination D cannot create unsafe rules. In particular, D cannot create rules that (a) cause infinite loops, (b) corrupt router state, (c) DoS routers or RCEs, or (d) violate ISP policies.

Mechanisms and Defenses: RBF uses four mechanisms to achieve the above properties: (1) rule certification, (2) rule leases, (3) anti-spoofing, and (4) static analysis. Table 1 summarizes which mechanisms serve to meet the four security properties.

6 Implementation

This section describes our prototype RBF router and rule compiler.

6.1 An RBF Rule Compiler

Our prototype offers users a high-level language largely identical to the syntax used in this paper in which to write rules. We wrote an RBF compiler in C++ that translates this high-level language into a compact rule format carried in packets. This compact format uses: 8B(ytes) for public-key hashes, 3B for the user-local index, 3B to identify the RCE, 3B to identify router-defined functions that do not require approval to be invoked and 8B for those that do, and 2B as the default RBF packet attribute values.[5] For the lease we use an absolute expiration time consisting of first 4B of the NTP format, with second-level granularity and a wrap-around period of 136 years. For efficiency, we use variable-length encoding in representing the internal rule structure. The maximum rule description size is 256B in our implementation.

6.2 A Prototype RBF Router

Rationale: We implemented RBF forwarding using Click [39] and RouteBricks [26]. Most commercial routers implement packet processing using ASICs or specialized network processors (NPs) rather than general-purpose CPUs and, as such, our software-based prototype is not entirely representative of currently deployed routers. To a large extent, our choice of prototyping platform is borne of necessity since commercial routers are closed. Beyond necessity, however, we believe a software-based prototype is valuable for multiple reasons. First, recent research [26, 31, 27] has demonstrated that, with modern multi-core servers, it is now possible to build high-speed software routers up to edge and even core speeds. Secondly, while not directly reusable, several aspects of our implementation architecture such as our approach to partitioning tasks across multiple cores should apply to network processor-based routers. Finally, several research [12, 28] and commercial switches [3] augment ASIC-based switches with some number of co-located general-purpose cores or servers for greater flexibility in packet processing – our prototype architecture is directly applicable to such platforms.

Design requirements: We build our prototype in the context of modern multi-core servers that incorporate multiple processors or "sockets", each with multiple cores [17, 1]. As shown in Fig. 1, the software stack of an RBF router includes the following key components: (1) an IP forwarding module, (2) the rule execution engine, and (3) some (possibly zero) number of specialized forwarding function modules. All packets traverse the rule execution and IP forwarding components, while different subsets of packets may traverse one or more specialized functions. In addition, the resources required to process a packet may vary widely across functions; *e.g.,* an en-

[5] Our current prototype only supports this default size.

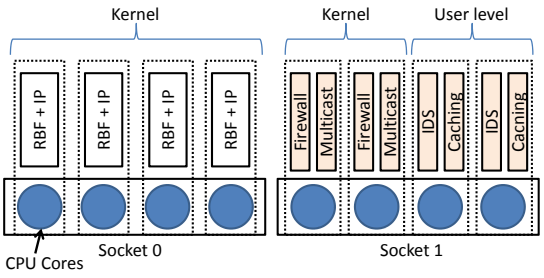


Figure 4: Core Allocation Example in RBF Router

cryption function would use lots of CPU but little cache, while a caching module may use more cache and less CPU. At a high level, our design goal is to balance high performance (*i.e.*, making efficient use of resources) with performance isolation, both across different functions, and between functions and the rule execution engine (*i.e.*, sharing resources in a fair manner).

Approach: In its full generality, the above goal requires contention-aware scheduling that simultaneously takes into account the multiple resources (cores, various caches, memory bandwidth, I/O bandwidth) for which tasks might contend. For modern multi-core systems, this is in itself an area of active research [24, 54] and beyond the scope of this paper. Instead, in our prototype, we address the issue as follows. The IP forwarding module and the rule execution engine are the central, most critical, components of the router and hence we assign these to a socket of their own and do not run specialized functions at cores in this socket. This avoids having the IP and rule execution engines contend with specialized functions for cache, CPU and other resources at the cost of some potential inefficiency since these "reserved" cores (if unused) cannot be used by specialized functions (if needed). We then assign specialized functions to the remaining "unreserved" cores. We rely on the existing (Click and Linux in our implementation) system schedulers to ensure fair sharing of CPU resources between functions on the same core.

To achieve high performance, we run a single thread performing both IP forwarding and rule execution at each of the reserved cores; this ensures that packets that do not invoke any specialized functions are processed entirely by a single core avoiding potentially expensive cache misses and inter-core synchronization [26]. Packets that invoke specialized functions must be relayed across cores and hence incur corresponding performance overheads due to cache misses and so forth. To improve the efficiency of such transfers when these functions are implemented in user space, we use shared memory pages and event queues. In our current prototype, when a rule invokes a user-level function, we make a single copy of the packet to the shared memory. An example of the resulting system architecture is depicted in Fig. 4.

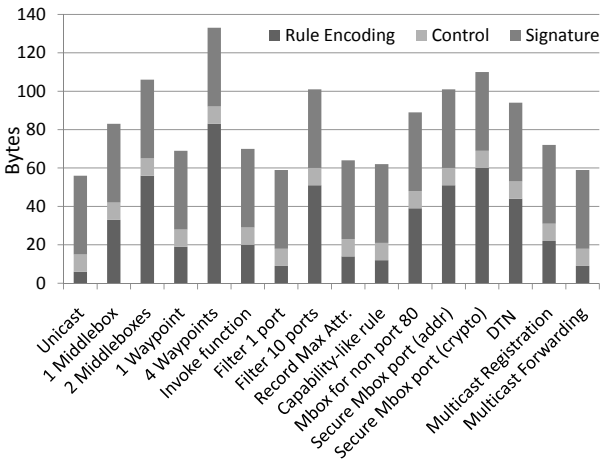


Figure 5: Rule Sizes

7 Evaluation

We use our prototype to evaluate the overhead RBF imposes on packets (§7.1), routers (§7.2) and RCEs (§7.4).

7.1 Packet Size Overhead

Fig. 5 presents rule sizes (in bytes) for a range of examples, including those from §3.4. The figure captures all the RBF-related fields and presents the size broken down into (a) the rule and the associated attributes' binary encoding; (b) the control fields used for the lease, RCE identification, to specify whether the return rule is in the packet and so forth; and (c) the rule signature. We assume a 41B signature obtained using ECDSA with ECC public keys for RCEs derived from the NIST B-163 or K-163 curves [18], offering 80 bits of security. Note that RBF is independent of the exact signature scheme used and that smaller (and faster) signatures can be used. However, shorter RCE keys may require more frequent updates to compensate for the lower security guarantees. The rules in Fig. 5 do not contain an identifier, and are identified by endpoints and routers using a hash over their content. Rule identifiers are required for rules whose content may change during a communication (such as the rules of mobile hosts) and incurs an additional 11B overhead in our implementation (8B for the hash of the public key and 3B for the user-selected index). Note that the rule identifier need be unique only with respect to a single communication endpoint (*i.e.*, all parties that a host X communicates with should have unique rule identifiers).

From Fig. 5 we can see that many common forwarding scenarios (unicast, routing via middleboxes, rules for DoS protection) can be expressed with around 60-80B rules while more complex rules (*e.g.*, loose source routing, secure middleboxes, anycast) can take as much as 140B. The average rule size across all examples we have implemented is 85B, representing 13% overhead for an average packet of 630B[4] and 6% overhead for a 1500B

packet. By comparison, using RSA-1024 signatures (instead of ECDSA) would incur 27% overhead on a 630B packet and 11% overhead on a 1500B packet.

Potential Optimization - Rule Caching: Per-packet overhead can be significantly reduced by *caching* rules at endpoints and routers; packets whose rules have been cached need only carry rule identifiers. There are two opportunities for caching. First, destinations can cache return rules; this allows the return rule to be eliminated from all but the first packet in a source-to-destination exchange. Second, rules can also be cached at routers. Here, however, we must ensure no packet carrying only a rule identifier arrives at a router that does not store the corresponding rule description. This might occur, for example, due to a route change or when a router deletes the rule from its cache. In such cases, the router can simply drop the packet in question, if the endpoints include the rule on all retransmissions and during periods of high packet loss. Of course, caching imposes additional storage overhead at routers as we evaluate shortly.

In summary, based on our evaluation, we see that the per-packet overhead due to RBF can range from as low as 24B when using caching and up to $\sim$250B in the bad case where there is no caching and the packet carries complex destination and return rules.

7.2 Router Overhead

In this section, we evaluate the overhead RBF imposes on routers for rules that do not invoke specialized processing functions; we consider router functions in the following section. The primary overhead RBF imposes on routers is the additional processing required to execute and authenticate rules and the additional storage capacity required if rules are cached. In this paper we do not evaluate rule authentication, which we assume is done by specialized hardware at trust-boundary routers; in [42] we present an evaluation for software rule authentication using RSA signatures, and show that our software router is not significantly slowed down when forwarding realistic traffic traces and performing verifications (the slowdown is less than 10%).

Rule Forwarding: We first measure the overhead of rule processing by comparing the performance of RBF-on-RouteBricks to that of unmodified RouteBricks running on a single high-end server machine. We use a dual-socket server with four 2.8GHz Intel Xeon (X5560) cores per socket to (from) which we generate (sink) traffic over two dual-port 10G NICs. In this experiment, we use all 8 cores to forward packets.

Fig. 6 plots forwarding rates for some of the examples from Fig. 5. The first column represents a packet stream with sizes generated based on a packet trace collected on the Abilene backbone [11]; since the packets from the trace do not have rules, we add to each packet the slowest

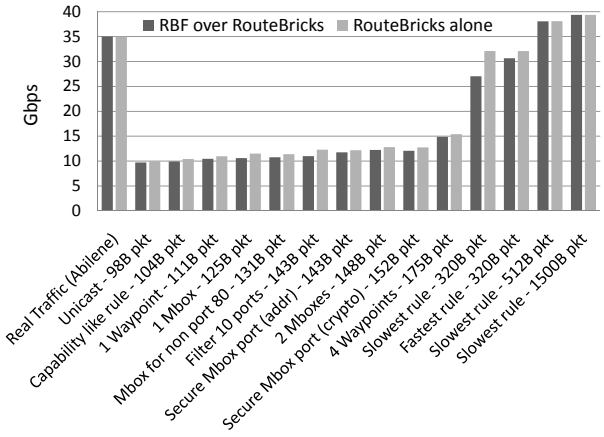
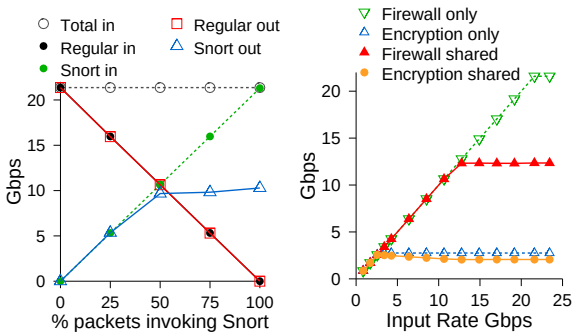


Figure 6: Forwarding speed for RBF over RouteBricks

rule that fits in the packet. By "slowest" we mean the rule that takes the longest time to forward, as determined by the number of conditions and actions encountered during forwarding. To capture the performance impact for small packets, we profile each rule without any payload and with no return rules. In the figure, packet sizes are shown next to the example name and entries are sorted in order of increasing packet size; the packet size also includes the Ethernet and IP headers. The last columns depict forwarding of larger packets, *i.e.*, that also contain data payload. To see the impact of the type of rule for these packets, we profiled them with the fastest and the slowest rules. Note that all rules are profiled in the worst case, meaning that the longest path through the rule is considered. For the slowest rule we use a 145B anycast rule which selects one out of 10 destinations based on the value of a packet attribute.

Overall, we see in Fig. 6 that the performance degradation due to RBF's more complex per-packet processing is always modest ($<$15%) and virtually non-existent at larger packet sizes. For small packets the CPU is the forwarding bottleneck, and RBF's added processing slows the router. For larger packets the I/O system is the bottleneck, and there are enough free CPU cycles to execute rules. A fine-grained profile of the rule execution module showed that it uses between 120 CPU cycles per packet for the fastest rule and 600 CPU cycles for the slowest rule; in comparison, the IP router used in our experiments requires around 3000 cycles per packet without rule execution. Also note that compared to the network-level forwarding results from Fig. 6, application-level goodput is further reduced by the RBF header.

Router cache sizes: We earlier proposed that routers cache rule authentications and/or rule descriptions. In each case, the number of cache entries required depends on the number of distinct rules the router sees. If we assume that all packets in a flow share the same rule, then the number of distinct rules passing through a given



(a) Isolation of Forwarding (b) Fairness Across Functions

Figure 7: Isolation and Fairness of Router Functions

router varies between the worst case of O(#flows) to the best case of O(#destinations) seen by the router. The former corresponds to a destination that uses a different rule for every source it communicates with, the latter to a destination that uses a single rule for all potential sources.

In our implementation, each cached authentication is 19 bytes – 11B for the rule identifier and an 8-byte hash value used to verify whether the rule has changed since it was authenticated. Each router uses its own secret hash function to prevent attackers from using hash collisions. Thus, one million rules would require only 19MB of memory. For caching entire rules, Fig. 5 reveals average and worst-case rule sizes of 85 and 133 bytes, respectively. If we conservatively assume traffic is uniformly distributed across these forwarding categories, we arrive at an estimated cache size of 85MB (average) to 133MB (worst-case) for 1M rules, which is within the scope of memory available in current routers.

7.3 Router Functions

Our router prototype supports specialized functions implemented at either kernel- or user-level. We currently support three router functions: (i) the Snort IDS [13] adapted to run as a user-level function, (ii) a kernel-mode firewall implemented in Click and (iii) a kernel-mode encryption engine also implemented in Click. Each function runs as a separate process/kernel thread isolated from the packet forwarding path through queues. We measure performance and fairness using the above functions on the same hardware as before. We dedicate four cores to the standard forwarding path and the remaining four cores to custom functions.

Fig. 7(a) illustrates the resource isolation between forwarding and router functions; the function used in this experiment is Snort (running on four cores). To generate traffic we use real traces of (moderately) malicious traffic created particularly for IDS testing [10, 30]. The average packet size of the trace used was 1065 bytes. To avoid biasing our results, we modify Snort not to drop any malicious packets so packets are only dropped due to resource exhaustion. Our test maintains constant total input traffic while increasing the percentage of input traffic that invokes Snort (X-axis). We see from Fig. 7(a) that Snort traffic does not affect the "regular" traffic that does not invoke Snort, in the sense that no regular traffic is dropped, even as a growing percentage of input Snort traffic is dropped. We observed the same isolation when using traces with small packets (see [42]).[6]

Fig. 7(b) illustrates isolation between router functions. We run three experiments: (1) all traffic invokes the firewall function and no traffic invokes encryption; (2) all traffic invokes encryption; and (3) equal halves of traffic invoking the firewall and encryption. Fig. 7(b) plots the resulting forwarding rates under increasing input traffic. In the third (shared) test the CPU is shared fairly between functions (we use Click-level scheduling); thus, the ratio between the maximum throughputs achieved by each router function is expected to roughly match the ratio between the throughputs of the functions when running in isolation. In Fig. 7(b) the encryption throughput is higher for a mix of firewalled and encrypted traffic than 50% of that when encryption is executed alone because the trace contains large packets. In this case, the CPU is not the bottleneck for the firewall functionality but is the bottleneck for encryption (since it is more CPU-intensive), and thus encryption ends up using the leftover firewall CPU cycles. If small packets are used, both functionalities achieve around 50% of their throughput in isolation [42]. Note that the high rates achieved by running each function in isolation illustrates the benefit of running instances of a single function at multiple cores (as opposed to one function per core) since this allows the unused resources from one function to be seamlessly utilized by other functions.

7.4 RCE Load

We use a simple back-of-the-envelope calculation to estimate the total number of RCE servers required for the Internet. The bulk of requests to RCEs are determined by IP address changes and per-client certifications requested by sites that protect against DoS (by redirecting DNS requests to powerful entities, see §3.4,§4.2). Note that in the latter case, requests to RCEs are made only for approved customers. There are currently around 700 million hosts in the Internet [9]; given the current trend of smart mobile devices we consider 1 billion hosts. We assume a worst-case scenario in which all hosts request certifications in the same second; these requests are made either by hosts individually to certify a rule or by

[6] We also measured the performance of the system with all the eight cores running both forwarding and Snort, and all the packets directed to Snort. While this configuration does not provide isolation for the regular traffic, it can forward a higher total throughput of 22Gbps.

	Flexibility Available to End-hosts						Security / Policy Compliance				
Property / Architecture	Explicit Middlebox Support	Multiple Paths	Invoke Router Extensions (*e.g.* IDS, multicast)	Use Router State in Forwarding (*e.g.* anycast, DTN)	Record Router State (*e.g.* network probing, ECN)	Mobility	Policy Compliant Loose Paths	Policy Compliant In-network Functionality Use	Receiver Reachability Control	Host DDoS Protection	Safety of Network & Routers (*e.g.* loops, break ISP policies)
RBF	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Active Networks	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No
ESP	No	No	Yes	No	Yes	No	No	No	No	No	Yes
i3, DOA	Yes	No	No	No	No	Yes	No	No	No	No	Yes
Platypus, SNAPP	Yes	Yes	No	No	No	No	Yes	No	No	No	Yes
TVA, SIFF	No	No	No	No	No	No	No	No	No	Yes	Yes
NUTSS	Yes	No	No	No	No	No	Yes	No	Yes	No	Yes
PushBack, AITF, StopIt	No	No	No	No	No	No	No	No	No	Yes	Yes
Predicate routing, Off-by-default	No	No	No	No	No	No	No	No	Yes	No	Yes
ICING	Yes	No	No	No	No	No	Yes	No	Yes	No	Yes

Figure 8: A comparison of RBF to:Active Networks [48, 50], ESP[25], i3[47], DOA[49], Platypus[44], SNAPP[40], TVA[52], SIFF[51], NUTSS[29], PushBack[32], AITF[21], StopIt[37], Predicate Routing[45], Off-by-default[22], ICING[46]

websites hosts are trying to access. We implemented RCE rule certification in software using RSA signatures, and measured it on the same 8-core server used throughout our evaluation. We find a single server can achieve a certification rate of over 16,000 rules per second. Based on benchmarks of our implementation and assuming an oversubscription rate of 10× (ISPs today commonly oversubscribe by 100×), the total load due to certifying rules above could be accommodated by around 6,000 servers; *e.g.*, handled by 20 RCEs with 300 servers each. Hardware implementations might reduce this number by more than an order of magnitude. For example, using recent ECC prototypes [53, 34] a single ASIC could potentially perform 40,000 RCE certifications per second, requiring a total of only 2500 such devices.

8 Related Work

RBF is inspired by and extends several directions in past research. RBF's contribution is in offering extensive flexibility while respecting policies, where prior approaches tended to focus on one or the other. Fig. 8 compares the flexibility and security features of RBF with those of previous proposals. RBF is complementary to recent efforts proposing open router APIs [15, 16, 38, 12] – we offer an overall network design by which endpoints use the new functionality these router architectures promise to enable. This paper extends an earlier position paper [43] that argued the case for a rule-based architecture.

A key feature that distinguishes RBF from previous proposals and allows it to achieve both flexibility and policy compliance is its division of functionality between the data and control planes. Active Networks typically make little use of the control plane, as they deploy the forwarding functionality and enforce security on the data plane. This makes policy compliance hard to achieve. In contrast, more recent proposals such as OpenFlow [12] rely heavily on the control plane and install flow state in the network to make sure the data plane respects the appropriate policies. This approach, while simplifies the data plane, results in a more rigid architecture. For example, supporting host mobility and traffic engineering require tearing down the old paths and instantiating new ones. These are expensive operations which have a negative impact on the scalability of these proposals. In contrast, with RBF, each packet contains (in its rule) enough information to prove to routers that it respects the policies of all participants involved in forwarding the packet. RBF achieves this property despite the fact that neither the routers nor the packet contain the policies. Thus, RBF retains the datagram model of the IP, unlike other recent proposals (*e.g.*, network capabilities [52, 51], ICING [46] and OpenFlow [12]), which are more akin to a connection-oriented model. Finally, while overlay-based architectures can implement more sophisticated data plane or control plane mechanisms, they cannot leverage support at routers and are thus less powerful.[7]

9 Incremental Deployment

All the benefits of RBF shown in Fig. 8 except receiver reachability control and DDoS protection can be achieved with a partial deployment of RBF routers and middleboxes. In an initial phase, RBF routers could support both RBF and legacy (non-RBF) traffic. To also offer DoS protection and reachability control, individual ASes can upgrade to RBF by dropping legacy traffic. Hosts in such ASes can use multihoming to handle legacy traffic, although they will be vulnerable to DoS attacks on legacy interfaces.

10 Discussion

We have presented RBF, an architecture we have argued strikes a desirable balance between flexibility and the ability to guarantee policy compliance of all network entities. We started this work with two high level goals in mind. First, we wanted a *complete* architecture that supports not only previously proposed communication primitives, but also future ones. Second, we wanted an

[7]For example, overlay architectures can only drop unwanted packets at overlay nodes and hence cannot create a network that is fundamentally default-off; once the network-layer address of a node is known, it can always be attacked at the underlying network layer.

efficient architecture in which a packet unwanted by a receiver along its path is dropped as early as possible.

While completeness in this context is difficult to formalize, intuitively we have reduced it to (1) supporting arbitrary communication paths, and (2) allowing all network entities (*i.e.,* sender, receivers, middleboxes, and routers) to be involved in the decision process. In other words, we wanted to be able to define virtually any forwarding path and give all involved parties a say in defining it. We noted that such a path can be encoded by associating with each node an "if-then-else" code snippet, which specifies the next node down the path. We further noted that allowing different network entities to define the communication pattern is equivalent to allowing them to define these code snippets. This is roughly what the RBF proposal is.

These goals are ambitious – they subsume, unite and extend many years of proposals for greater flexibility and security in networks – and much of RBF's complexity follows from these goals.

Finally, one might question whether a relaxation of RBF's goals might lead to a significantly simpler design. This is a valid question that we leave for future work. We believe that understanding the fundamental tradeoffs associated with these goals is critical and, at the very least, that RBF is a step toward arriving at such an understanding.

Acknowledgments: We would like to thank our shepherd, Brad Karp, the anonymous reviewers as well as Gautam Altekar, Katerina Argyraki, Byung-Gon Chun, Mihai Dobrescu, Ali Ghodsi, Brighten Godfrey, Gianluca Iannaccone, Jayanth Kannan, Eddie Kohler, Petros Maniatis, Maziar Manesh, Sergiu Nedevschi, Costin Raiciu, Arsalan Tavakoli and Matei Zaharia for their feedback on various stages of the paper.

References

[1] AMD Opteron Server Processor. http://www.amd.com/.
[2] Arbor Networks Peakflow: www.arbornetworks.com/en/peakflow-ip-flow-based-technology.html.
[3] Arista 7100 Series Switches. http://www.aristanetworks.com/en/7100Series.
[4] CAIDA: www.caida.org/data/realtime/.
[5] Certicom Suite B IP Core, http://www.certicom.com.
[6] Cisco Traffic Anomaly Detector: www.cisco.com/en/us/products/ps5892/.
[7] CLP-17: High Performance Elliptic Curve Cryptography (ECC) Point Multiplier Core, http://www.ellipticsemi.com/products-clp-17.php.
[8] Elliptic Curve Point Multiply and Verify Core, http://www.ipcores.com/elliptic_curve_crypto_ip_core.htm.
[9] Internet Systems Consortium, Internet Host Count History, https://www.isc.org/solutions/survey/history, 2009.
[10] Lincoln laboratory: Darpa intrusion detection data set (week 6). http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/.
[11] NLANR: Internet Measurement and Analysis. http://moat.nlanr.net.
[12] Open Flow Switch Consortium. http://www.openflowswitch.org.
[13] Snort IDS/IPS, http://www.snort.org.
[14] RFC 1305 - Network Time Protocol. 1992.
[15] Juniper Networks Delivers Platform for Customer and Partner Application Development. *Juniper Press Release*, Dec. 2007.
[16] Cisco Opens Routers to Customers and Third-Party Applications. *Cisco Press Release*, April 2008.
[17] Next Generation Intel Microarchitecture (Nehalem). http://intel.com, 2008.
[18] Digital Signature Standard (DSS). *Federal Information Processing Standards Publication*, June 2009.
[19] T. Akin. Hardening Cisco Routers. *O'Reilly*, 2002.
[20] K. Argyraki and D. R. Cheriton. Loose Source Routing as a Mechanism for Traffic Policies. In *ACM SIGCOMM Workshops*, 2004.
[21] K. Argyraki and D. R. Cheriton. Active Internet Traffic Filtering: Real-time Response to Denialof-Service Attacks. In *USENIX Annual Conf.*, 2005.
[22] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker. Off by Default! In *ACM HotNets*, 2005.
[23] R. Beverly and S. Bauer. The Spoofer project: Inferring the extent of source address filtering on the Internet. In *SRUTI Workshop*, 2005.
[24] S. Boyd-Wickizer, R. Morris, and M. F. Kaashoek. Reinventing Scheduling for Multicore Systems. In *HotOS*, 2009.
[25] K. L. Calvert, J. Griffioen, and S. Wen. Lightweight Network Support for Scalable End-to-End Services. In *ACM SIGCOMM*, August 2002.
[26] M. Dobrescu, N. Egi, K. Argyraki, B.-g. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: Exploiting Parallelism to Scale Software Routers. In *ACM SOSP*, 2009.
[27] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy. Towards High Performance Virtual Routers on Commodity Hardware. In *Proceedings of 4th Conference on Future Networking Technologies (ACM CoNEXT 2008)*, Madrid, Spain, December 2008.
[28] A. Greenhalgh, F. Huici, M. Hoerdt, P. Papadimitriou, M. Handley, and L. Mathy. Flow processing and the rise of commodity network hardware. *SIGCOMM Comput. Commun. Rev.*, 39(2), April 2009.
[29] S. Guha and P. Francis. An End-Middle-End Approach to Connection Establishment. In *ACM SIGCOMM*, 2007.
[30] J. W. Haines, R. P. Lippmann, D. J. Fried, E. Tran, S. Boswell, and M. A. Zissman. 1999 DARPA Intrusion Detection System Evaluation: Design and Procedures. In *MIT Lincoln Laboratory Technical Report*.
[31] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: A GPU-Accelerated Software Router. In *ACM SIGCOMM*, 2010.
[32] J. Ioannidis and S. M. Bellovin. Implementing Pushback: Router-Based Defense Against DDoS Attacks. In *NDDS*, 2002.
[33] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking Named Content. In *ACM CoNEXT)*, 2009.
[34] K. Järvinen et al. Fast point multiplication on Koblitz curves: Parallelization method and implementations. *Microprocess. Microsyst.*, 33(2), 2009.
[35] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. In *ACM SIGCOMM*, 2007.
[36] X. Liu, A. Li, X. Yang, and D. Wetherall. Passport: Secure and Adoptable Source Authentication. In *USENIX NSDI*, 2008.
[37] X. Liu, X. Yang, and Y. Lu. To Filter or to Authorize: Network-Layer DoS Defense Against Multimillion-node Botnets. In *ACM SIGCOMM*, 2008.
[38] J. C. Mogul, P. Yalagandula, J. Tourrilhes, R. McGeer, S. Banerjee, T. Connors, and P. Sharma. API Design Challenges for Open Router Platforms on Proprietary Hardware. In *ACM Hotnets*, 2008.
[39] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek. The click modular router. *SIGOPS Oper. Syst. Rev.*, 33(5):217–231, 1999.
[40] B. Parno, A. Perrig, and D. G. Andersen. SNAPP: Stateless Network-Authenticated Path Pinning. In *ACM ASIACCS*, 2008.
[41] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y.-C. Hu. Portcullis: Protecting Connection Setup from Denial-of-Capability Attacks. In *ACM SIGCOMM*, 2007.
[42] L. Popa, N. Egi, S. Ratnasamy, and I. Stoica. Rule-based Forwarding (RBF): Improving the Internet's Flexibility and Security. *UCB Technical Report*, 2010. http://www.eecs.berkeley.edu/%7Epopa/rbfTechReport.pdf.
[43] L. Popa, I. Stoica, and S. Ratnasamy. Rule-based Forwarding (RBF): improving the Internet's flexibility and security. In *ACM Hotnets*, 2009.
[44] B. Raghavan and A. C. Snoeren. A System for Authenticated Policy-Compliant Routing. In *ACM SIGCOMM*, 2004.
[45] T. Roscoe, S. Hand, R. Isaacs, R. Mortier, and P. Jardetzky. Predicate Routing: Enabling Controlled Networking. In *ACM Hotnets*, 2002.
[46] A. Seehra, J. Nous, M. Walfish, D. Mazieres, A. Nicolosi, and S. Shenker. A Policy Framework for the Future Internet. In *ACM Hotnets*, 2009.
[47] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *ACM SIGCOMM*, 2002.
[48] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A Survey of Active Network Research. *IEEE Comm.*, 1997.
[49] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes no longer considered harmful. In *OSDI*, 2004.
[50] D. Wetherall. Active network vision and reality: Lessons from a capsulebased system. In *ACM SOSP*, 1999.
[51] A. Yaar, A. Perrig, and D. Song. SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks. In *IEEE Symp. on Sec. and Priv.*, 2004.
[52] X. Yang, D. J. Wetherall, and T. Anderson. A DoS-limiting Network Architecture. In *ACM SIGCOMM*, 2005.
[53] Y. Zhang, D. Chen, Y. Choi, L. Chen, and S.-B. Ko. A high performance ECC hardware implementation with instruction-level parallelism over GF(2163). *Microprocess. Microsyst.*, 34(6), 2010.
[54] S. Zhuravleva, S. Blagodurov, et al. Addressing Shared Resource Contention in Multicore Processors via Scheduling. In *ASPLOS*, 2010.