Rule-based Forwarding (RBF): improving the Internet's flexibility and security

Lucian Popa* Ion Stoica* Sylvia Ratnasamy†

1 Introduction

From active networks [33] to the more recent efforts on GENI [5], a long-held goal of Internet research has been to arrive at a network architecture that is *flexible*. The allure of greater flexibility is that it would allow us to more easily incorporate new ideas into the Internet's infrastructure, whether these ideas aim to improve the existing network (*e.g.*, improving performance [36, 23, 30], reliability [12, 13], security [38, 14, 25], manageability [11], *etc*.) or to extend it with altogether new services and business models (*e.g.*, multicast, differentiated services, IPv6, virtualized networks).

An unfortunate stumbling block in these efforts has been that flexibility is fundamentally at odds with another long-held goal: that of devising a *secure* network architecture. As one example of this tussle: several schemes seek the flexibility of forwarding packets along multiple paths (*e.g.*, [36, 12, 13]), to improve performance and reliability while schemes such as network capabilities [38, 37, 29] seek to constrain packets to a single "approved" network path. In a different example: active filtering [14, 22] seeks to block unwanted traffic based on source addresses; but yet other proposals [32, 35] enable attackers to legitimately conceal their addresses when routing through middleboxes.

In this paper, we propose a new architectural concept – that of packet *rules* – and develop a *rule-based* forwarding architecture (RBF) that we argue is both flexible and secure. A rule is a simple `if-then-else` construct that describes the manner in which the network should – or should not – forward packets. In RBF, instead of sending packets to a destination (IP) address, end-hosts send packets using the destination's rule. At a high-level, the reason RBF achieves both security and flexibility is that, with rules, a user must specify both *what* packets it is willing to receive as well as *how* it wants these packets forwarded and processed by the network. More specifically, RBF is designed to ensure the following properties hold at all times:

1. **rules are mandatory:** every packet must contain a rule; any packet without a rule is dropped.
2. **rules are provably valid:** a rule is deemed *valid* if all recipients (end-hosts, middleboxes and/or routers) named in the rule have explicitly agreed to receive the associated packet(s). Given a rule, any router, middlebox or end-user can verify the rule's validity.
3. **rules are provably safe:** rules cannot exhaust network resources; *e.g.*, rules cannot compromise or corrupt routers nor cause packet forwarding loops.
4. **rules allow flexible forwarding:** rules can select arbitrary forwarding paths and/or invoke functionality made available by on-path routers; both path and function selection can be conditioned on (dynamic) network and packet state.

The first two properties enable security by ensuring the network will not forward a packet unless it has been explicitly cleared by its recipients while the third property ensures that rules cannot be (mis)used to attack the network itself. As we shall show, our final property enables flexibility by allowing a user to give the network fine-grained instructions on how to forward his packets.

In the remainder of this paper, we present RBF, a forwarding architecture that meets the above properties.

2 Design Rationale and Overview

In this section, we present the answers to three key questions for the RBF design: (1) What can rules do?, (2) How are rules distributed to routers and end-hosts?, and (3) How to ensure rules are valid?

The RBF architecture aims to provide end users extra control over the forwarding of their packets as well as over the packets that can reach them; however, RBF is not concerned with route discovery and route computation, which we argue should remain under the control of network owners. For this reason, we implement the RBF functionality above the network layer (IP).

2.1 What can rules do?

Broadly speaking, previous architectures that aim to provide flexible forwarding *e.g.*, [33, 17, 36, 32, 35, 30, 28] can be divided into four classes based on whether end-users are allowed to (i) modify router forwarding state, or/and (ii) modify forwarding information in packet headers (*e.g.*, destination address). These classes are: (1) Modify both router forwarding state and forwarding information in packet headers (*e.g.*, most Active Network proposals [33], ESP[17]). (2) Modify router state but not packet headers (*e.g.*, Active Networks focusing on "active storage"[34]). (3) Modify packet forwarding information but not routing state (*e.g.*, i3[32], DOA[35]). (4) Modify neither router forwarding state nor packet state (*e.g.*, IP). [1]

We argue that the third class provides the best tradeoff between flexibility and security. Allowing data packets to modify router forwarding state poses significant security risks. Indeed, at the limit, an application could

*University of California, Berkeley
†Intel Labs, Berkeley

[1] Of course, routers perform logging and monitoring tasks which change their state, and routers modify the TTL value which represents state in packet, but in this section we refer to *user-controlled forwarding* abilities. We also ignore IP loose-source routing which is rarely used due to security concerns.

implement complicated distributed protocols (*e.g.*, routing protocols) whose safety is notoriously hard to verify. This eliminates the first two classes. In contrast, the last class offers limited flexibility, as end-users can exercise no control on packet forwarding. This leaves us with the third class, to which RBF belongs.

With RBF, end-users control packet forwarding using rules. Each packet contains a rule and a set of *attribute-value* pairs in its header. Upon receiving a packet, the router executes the packet's rule. The rule takes as input the attribute values in the packet header, as well as forwarding state exposed by the routers under the form of *router attributes*. Rules may *update* the value of the packet attributes, *forward* the packet via IP or to layers above (transport or application), or *drop* the packet. In contrast, rules *cannot* modify the router attributes.

A rule can be represented by a simple transition table: based on the current value of packet attributes and router attributes, the rule may generate a new set of packet attributes and forward/drop the packet. In practice, rules are encoded using an *if-then-else* tree-like structure, which has a more compact representation (§3).

This generic structure enables RBF to provide a rich set of forwarding functionalities, including explicit middlebox traversal, multi-path routing, anycast, multicast, and loose source routing. We illustrate the generality of rules through a set of examples in Section 4.

2.2 How are rules distributed?

Distributing rules to routers: Routers need to know the rules associated with the packets they receive. There are two basic approaches by which routers can obtain rules.

First, rules can be carried in packets; this frees routers from maintaining per-rule state, and implementing costly rule distribution protocols. This approach however incurs a high overhead on the data path as rules increase packet headers, and routers need to verify rules to ensure their validity.

The second approach is to install rules at routers; this incurs a lower overhead on the data plane, as packets need only to carry rule identifiers instead of the rules themselves. However, the process of obtaining rules can be complex; the router can either get rules in advance, in which case it may need to store a huge number of rules or the router can download rules on demand, in which case it may need to buffer the packets it receives until it obtains the rules for those packets. Moreover, the packets installing rules have to themselves travel on rules, which makes this process very difficult.

In this paper, we chose to have packets carry rules. In Section 7 we estimate the overhead of rules.

Distributing rules to end-hosts: RBF leverages the DNS infrastructure to distribute rules. Upon a DNS lookup, instead of returning the destination IP address, the DNS server returns the rule of the destination.

In Section 4 we show how servers can protect against DoS attacks, by redirecting their DNS entry to a large entity and by creating per-client rules (*i.e.*, rules that drop packets from any other source than one client).

2.3 How is rule validity ensured?

To guarantee rule validity, each rule is certified by a third party certifying authority, called Rule Certification Entity, or RCE for short. The RCE guarantees that all nodes whose addresses appear explicitly in a rule (*i.e.*, destinations, middleboxes and indirection routers) agree with the rule. In addition, the RCE may verify the rule for forwarding-loops before certifying it (§6).

Upon receiving a packet, an RBF router verifies the rule's signature. If the verification fails, the router drops the packet; otherwise, the router applies the rule. To verify the rule certificate, a router needs to know the public keys of all RCEs. We believe this is a a reasonable assumption as we expect the number of RCEs to be relatively small.

Destinations can validate rules that allow traffic only from certain sources. Malicious senders can attempt to spoof their addresses to bypass rule directives and mislead routers into not dropping their packets. To prevent this attack, RBF assumes the existence of an *anti-spoofing* mechanism. In this paper we propose the use of passports [24], but other alternatives are possible (§6).

To make sure rules can only be used for a limited amount of time, rules have associated *leases*(§6).

2.4 Summary

RBF can be succinctly described as: (1) Every packet contains a rule; there are no exceptions and no special traffic. (2) A rule is a set of forwarding directives associated to the packet by end-users; the expressivity of rules enables forwarding flexibility. (3) The rule bears a trusted entity's signature, which guarantees the rule is valid and safe. (4) Routers verify the rule signature and forward conforming to the rule's directives.

3 Rule Specification

RBF represents rules as a sequence of actions that can be conditioned by *if-then-else* structures of the form:

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

Conditions are *comparison operators* applied on the packet and router attributes.

The actions can be: (1) *update* the value of the packet attributes; (2) *drop* the packet and (3) *forward* it to the network interfaces via IP or to higher layers implementing the functionalities requested by the rule.

The packet attribute set consists of the five-tuple in the IP header (*i.e.*, IP addresses, ports, protocol type), and a number of custom attributes with *user defined semantics.* For simplicity, RBF does not allow rules to add new packet attributes. On the other hand, router attributes may include the router's IP address, AS number, congestion level, flags indicating whether the router implements a specific functionality such as intrusion detection and so forth.

Every rule has a unique identifier (ID). The rule ID is defined as the concatenation of a hash of the rule owner's public key and of an index unique to the owner.

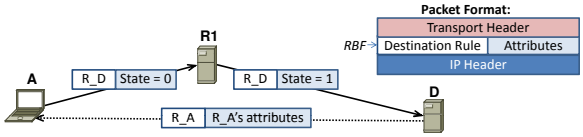

Figure 1: Simple indirection example.

For example, the following rule forwards a packet to a destination D via a waypoint router R1; a packet attribute named state indicates whether or not the packet has already visited R1:

```
R_D:
  if(packet.state == 0)      //from source to R1
    if(router.address != R1)
      sendto R1
    else packet.state = 1    //to D
  if(packet.state == 1)
    sendto D
```

where the sendto action is a short form for the following: change the destination address to D and forward the packet using IP if D is not the local address, or forward it to the transport layer otherwise. Once the packet is forwarded or dropped, the rule execution stops; the packet is by default dropped if it is not explicitly forwarded by the rule. Fig.1 illustrates this example, when D is communicating with another host A, who's rule is R_A.

While RBF does not allow rules to modify the packet payload or replicate packets, RBF enables rules to *invoke* such functionality at middleboxes or routers, if available. This allows RBF to leverage recent advancements in router design that enable network operators to provide new functionality through router extensions [8, 9, 26]. While the functionality invoked by a rule at a router may modify the packet payload, may replicate packets and may change the router state, such actions are done by code that is controlled by the network operator.

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, each 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.

Expressiveness: At a high-level, our rule specification can be viewed as defining a finite state machine (FSM): the state is encoded in the packet attributes, while the input is represented by the attributes of the routers along the data path. The rule specifies the transition function of the FSM. It is not difficult to show that RBF can theoretically implement *any deterministic* forwarding function that can change the packet attributes only. However, there are forwarding functions that cannot be efficiently expressed in RBF. This limitation is similar to the impracticality of implementing complex functions with the simple FSM mechanism due to the exponential growth in the number of states. In particular, forwarding decisions based on any functions other than comparisons of packet and router attributes (*e.g.*, sum, hash, logarithm) are not practically expressible in RBF.

4 Examples of RBF usage

RBF gives end-users four basic types of control: (1) block unwanted packets in the network; (2) redirect packets through a sequence of waypoints; (3) use enhanced functionality at routers (if available) and middleboxes; (4) use router state in the forwarding decision and record such state. Next, we present several examples to illustrate RBF's flexibility.

Port-based filtering: Web server D can use the following simple rule (registered under its DNS name) to make sure that it receives only packets destined to 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 can use the following rule to route all the other incoming traffic through a packet scrubber[3], deployed either by D's provider, or a third party:

```
R_mbox_port:
  if(packet.dst_port == 80) sendto D
  else
    if(packet.state == 0)          //before scrubber
      if(router.address != Scrb) sendto Scrb
      else
        packet.state = 1           //mark scrubbed
        invoke Scrb_service        //scrub
    else if(packet.state == 1) sendto D //scrubbed
```

Thus, similar to other previous proposals [32, 35], RBF provides explicit support for middleboxes, such as WAN optimizers, proxies, caches, compression or encryption engines, transcoders, intrusion detection (IDS) boxes.

Secure Middlebox Traversal: In the previous example, an attacker can directly send a packet with the state attribute value set to 1 such as to appear that the packet has already visited the middlebox. The destination can protect against this behavior in two ways.

In the first approach, D can simply ensure that the packet does indeed arrive from the middlebox when the state attribute is set to the value of 1, *i.e.*, `if (packet.source != Scrb) drop`. For this purpose, the IP source address attribute has to be set when the packet gets to Scrb (before invoking the service), *i.e.*, `packet.source = Scrb`. This rule relies on the anti-spoofing mechanism used in RBF to block packets with spoofed source address attributes.

Note that to avoid legitimate packets to be dropped by the anti-spoofing mechanism, the source address attribute has to be set at all off-path waypoints and routers that change the destination address; we omit this in the presented rules for readability purposes.

In a second approach, the destination can use stronger cryptographic guarantees. For this purpose, the middlebox has to offer a functionality that creates a cryptographic proof, which guarantees the packet has visited it. The destination itself implements another functionality that verifies these proofs before delivering the packet to the application. Both these functionalities are invoked by the rule (at Scrb and respectively at D). This process can be generalized to arbitrary rules (the verification functionality has to use static analysis).

DoS Protection: To protect against DDoS attacks, a server, D can create a custom rule for each client; this rule drops packets from any source other than the client.

By controlling the number the number of rules granted at a given time, D controls the maximum number of active clients. An example of a rule similar to a capability is:

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

Similarly to capability based architectures [38, 37], 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. Unfortunately, this opens the capability distribution protocol itself to DoS attacks (dubbed Denial of Capabilities, *e.g.*,[29]).

In RBF, a destination D can contract with a large entity E and redirect it's DNS entry to E. Each requester will then contact E instead of contacting D. E forwards the rule requests to D, but acts as a rate throttler, limiting the rate of rule requests; thus, D cannot be under DoS attacks. D creates rules and replies back to the requesters it wants to approve (*e.g.,* this decision is based on its current load, as in [38]). E forwards requests to D conforming to a policy that may contain: a maximum rule request rate, a white-list of always allowed senders/prefixes, a black-list of denied senders and other parameters such as a desired request service discipline (*e.g.,* fair queuing across senders), *etc.* D can update this policy at any time.

The assumption here is that a E has typically far more resources than D. D can employ multiple such entities and leverage DNS round-robin to further diffuse potential attacks. Also note that, DoS attacks on the rule granting process are less effective than on the data plane [25].

Alternatively, E could directly create and return a rule for each requester, on D's behalf. For this purpose, D provides E a rule template parameterized by the requester's address and a policy to grant rules. In this case, E has to incorporate the functionality of an RCE and certify rules. Compared to throttling rule requests to D, this approach avoids involving the destination in the rule granting process, but requires some of the RCEs to be able to withstand DoS attacks, and may also require more frequent policy updates between D and E.

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 some of their routers; this functionality maintains (soft) state at routers to create a multicast (reverse path) tree. This approach implements essentially an overlay multicast solution, which leverages the IP multicast functionality at on-path routers. For simplicity, here we consider only single-source multicast trees.

Source S wants to send packets to a multicast group uniquely identified by M. S advertises (*e.g.,* on the web) that receivers should use the following registration rule:

```
R_multicast_registration:
  if(router.multicast_available and
            packet.crt_router != router.address)
    packet.crt_router = router.address
    invoke multicast_registration
  sendto S
```

where the `crt_router` attribute makes sure multicast registration is called just once at each multicast router.

When joining the multicast tree, a receiver, D, sends a registration packet using the above rule. Prior to sending this packet, D creates a rule to receive the multicast packets sent by S and inserts it in the packet's payload:

```
R_mcast_forwarding_to_D:
  packet.dst_port = PORT_D_LISTENS_MCAST_M
  sendto D
```

The packet payload also contains the identifier M. The first multicast enabled router R processing the registration packet, stores the mapping M→R_mcast_forwarding_to_D. R creates its own rule to receive these multicast packets, replaces D's rule in the packet, and sends the packet further.

The registration continues recursively. To send a multicast packet, S sends a copy of the packet to every router from which it has received a registration.

On top of the vanilla multicast functionality, this approach can easily implement other functionalities, such as access control and traffic accounting, which have been previously proposed to "fix" the IP multicast [21].

Other Examples: RBF enables a plethora of useful examples not presented here, such as: *secure loose path forwarding*[30, 28], *multipath forwarding* , *anycast*, *on-path redirection* (*e.g.,* use router attributes to track the availability of a disconnection-prone destination and implement DTN[19]), *path logging* (*e.g.,* record on-path router information, such as the max/min link bandwidth, forwarding table size, packet counters, number of neighbors, queue size, up time, *etc.*). More importantly, all these individual examples can be combined as needed.

Source Control: In some cases, the source may also desire control over how its outgoing packets are forwarded; for example, to send packets through an anonymizer. We do not elaborate in this paper, but in such cases, a packet can also contain a *source rule*. A packet is always forwarded first on the source rule (if present) and once the source rule has been completed, the packet is forwarded as per the destination rule. We use a well-known packet attribute to denote which rule is currently active and only allow this attribute to be set, thus ensuring that control does not return to the source rule once the destination rule has been activated. (This is verified by static analysis before rule certification.)

5 Rule Certification and Creation

To certify rule R, an end-host D sends a certification request to an RCE; the rule to access the RCE is provided to D by its ISP. Upon receiving this request, the RCE uses a challenge response mechanism to verify that D is indeed the owner of the key used to name R and that D owns the only (assume for now) IP address explicitly specified

Attacks / Mechanisms	Certification	Lease	Anti-Spoofing	Rule Structure
Rule Spoofing	×	×		
Infinite Loops	×			×
DoS Attack	×	×	×	×
State Corruption				×
Man-in-the-middle	×			
Rule Violation			×	
Replay Attack		×		

Table 1: Attacks and Defense Mechanisms

in R; for this, D's return rule has to contain only that same address used in R.[2] Next, the RCE verifies whether R is well formed; if so, the RCE signs R and sends it back to D. If the rule contains multiple end hosts, the RCE asks every end-host that appears in the rule, besides the owner, to sign the rule (the certification request contains a contact rule for each host). Only after all end-hosts sign the rule, the RCE verifies the rule and signs it as well.

To protect against DoS attacks, RCEs control the number of clients that can reach them (*e.g.*, through the number of ISPs they have contracts with), and limit the rate of certification requests to a contracted rate.

To create rules in the first place, we envision that users rely on applications similar to firewall configuration software. Rule creation could also be delegated; for instance enterprises could create rules for their employees. DHCP servers can be extended to work with RBF (along with the address, they create and return a rule).

6 Security Analysis

We consider attacks from malicious hosts and routers and we discuss two broad types of attacks:

A. Creation of Malicious Rules: (1) *Rule Spoofing:* an attacker creates a rule that sends traffic to an address it does not own; (2) *DoS the network:* create infinite loops, amplify traffic, slow down routers; (3) *DoS hosts*; (4) *Corrupt router internal state*.

B. Misuse of Existing Rules: (1) *Replay attack:* use a rule for a longer time than its creator intended. (2) *Rule violation:* violate rule directives, *e.g.*, sources avoid their packets being dropped, bypass middleboxes; (3) *Man-in-the-middle attack:* modify rules inserted by other hosts.

RBF uses three mechanisms to protect against these attacks: (1) rule certification; (2) rule leases and (3) anti-spoofing. Table 1 shows the mechanisms used against each attack.

When it certifies a rule, a RCE associates to the rule an expiration time stamp as required by the requester; there is a maximum allowed lease value, to prevent attacks when IP addresses change. A router drops a packet if its current time exceeds the rule expiration time. We assume that all routers and RCEs are synchronized, via NTP[7] as recommended [10]. We have a working solution that does not require any router clock synchronization, but which we do not detail here.

[2]To certify its *first* rule, D sends the first certification request to its RCE C with no return rule, but with a request to certify a simple rule R_S allowing traffic only from C to D. C checks R_S to be indeed as such and simply replies on this rule.

Alternatively to using passports[24] for anti-spoofing, to protect against DoS, RBF can simply leverage the already existing ingress filtering, deployed by over 75% of today's ASes [16]. Thus, only hosts in less than 25% of the ASes can bypass rule dropping directives based on source address and participate in a DoS attack; the default-off nature of RBF additionally scales down the bots available to attackers since it reduces the spreading potential of viruses. Moreover, an attack will stop once the lease of the rule expires; the victim can detect the attacker, and stop providing new rules to the attacker for a period of time. RBF can further incentivize the deployment of ingress filtering; destinations under attack can simply deny rules to requesters from ASes known not to ingress filter.

To prevent forwarding-loops, RCEs use *static analysis* to detect whether rules can create cycles. Since rules have a simple format, we use methods similar to symbolic execution (but much simplified) to identify the potential for loops in the FSM governing the rule behavior; in such cases, rules are not signed.

In RBF, an attacker can also send packets with random certifications, causing the router to try to verify their signature. RBF routers can simply blacklist such attackers, if the anti-spoofing mechanism prevents the attackers from concealing their identity. Moreover, this attack can only occur at the first RBF router. For this reason, the incentives for it are low, since the attacker targets its own access route and at most other collocated users, and the routers can detect attackers more easily.

Malicious routers on the path from the RCE to a host could certify rules in the name of the host (rule spoofing). To prevent this threat, ISPs could upload to the RCEs the mapping between the IP and the public key of their customers (or sign such a mapping). However, since the malicious routers can already significantly affect the traffic to hosts (drop/alter/multiply) this solution may be overkill for most cases.

7 Related Work and Next Steps

Preliminary evaluation: We have implemented a compiler that translates rules from the high-level language used in the examples of this paper into a compact format that is carried in packets and interpreted by routers. In our current implementation, many common forwarding scenarios (unicast, routing-via-middleboxes, filtering, multicast) can be expressed with 60-70Byte rules (this value including all the RBF fields and a 40B signature), while some of our more complex rules could take as much as 100B (*e.g.*, the secure version of the scrubber example in §4) or even 130B (*e.g.*, loose source routing with four hops). However, a complete evaluation is a topic for future work.

We also built a prototype rule forwarding engine using Click[27]. We have applied RBF on top of RouteBricks[18]. Preliminary results show that interpreting rules does not degrade throughput for packets larger than 300B compared to running [18] alone, while

	Security				Flexibility								
Functionality / Architecture	Receiver Reachability Control	Host DoS Protection	Network Safety (*e.g.* loops)	Router Safety	Control over path (*e.g.* loose path, middlebox)	Multiple Paths	Invoke Router Extensions (*e.g.* IDS, multicast)	Use Router State in Forwarding (*e.g.* DTN)	Record Router State (*e.g.* network probing, ECN)	Mobility	Select Dest. (*e.g.* anycast)	Both Source & Dest. Control of Flexibility	Keep State at Routers (*e.g.* for reliable multicast)
RBF	Yes	Yes	Yes	Yes	Yes, secure	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Active Networks	No	~ Hard	No	~ Hard	Yes, not secure	Yes	Yes	Yes	Yes	~ Yes	Yes	No	Yes
ESP	No	No	Yes	Yes	IP, not secure	No	Only ESP	Yes	Yes	No	No	No	Yes
i3, DOA	No	No	Yes	Yes	Yes, not secure	No	No	No	No	Indirection	Limited	Yes	No
Platypus, SNAPP	No	No	Yes	Yes	Yes, secure	Yes	No	No	No	No	No	No	No
TVA, SIFF	No	Yes	Yes	Yes	IP, not secure	No	No	No	No	No	No	No	No
NUTSS	Yes	Yes	Yes	Yes	Yes, secure	No	No	No	No	No	No	No	No
PushBack, AITF, StopIt	No	Yes	Yes	Yes	IP, not secure	No	No	No	No	No	No	No	No
Predicate routing, Off-by-default	Yes	~Yes	Yes	Yes	IP, not secure	No	No	No	No	No	No	No	No

Figure 2: Related work comparison.

for small packets, the throughput degradation is always less than 30%. Routing on a 8-core (Intel Xeon X5560) machine gave us forwarding speeds of up to 23Gbps. Furthermore, RBF could leverage specialized hardware, such as network processors.

Recall also that routers must verify rule signatures. While signature verification is expensive, there are two reasons we argue this overhead is manageable. First, only routers at trust boundaries need to authenticate rules; border routers see lower traffic loads than core ones. Second, routers can cache the results of authentication checks, maintaining a hash of the rule and its signature; on subsequent packets, routers need only to verify this hash. Thus verifications are required only for the new arriving flows at border routers (assuming a large enough cache); *e.g.*, traffic measurements [1] show that new flow arrivals represent less than 1% of link capacity, rates that can be handled with commercial modules (*e.g.*, for ECC[2, 4]).

Related Work: RBF is inspired by, and augments, several directions in past research. In general, RBF's contribution is in offering both flexibility and security, where prior approaches tended to focus on one or the other.

Fig. 2 presents a synthesis of the security and flexibility abilities of RBF compared to several previous works: Active Networks [33, 34], ESP[17], i3[32], DOA[35], Platypus[30], SNAPP[28], TVA[38], SIFF[37], NUTSS[20], PushBack[22], AITF[14], StopIt[25], Predicate Routing[31], Off-by-default[15].

Several recent proposals call for open router APIs [8, 9, 26] and APIs to modify flow entries in switches[6]. RBF is complementary to these efforts; we offer an end-to-end data plane design that allows endpoints to use the new functionality these router architectures enable.

Incremental Deployment: All RBF benefits shown in Fig. 2, except the first two (*i.e.*, reachability and DoS) 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 can use multihoming to handle legacy traffic, although they will be vulnerable to DoS attacks on the legacy interfaces.

References

[1] CAIDA: www.caida.org/data/realtime/.
[2] Certicom Suite B IP Core, http://www.certicom.com.
[3] Cisco Traffic Anomaly Detector: www.cisco.com/en/us/products/ps5892/.
[4] CLP-17: High Performance Elliptic Curve Cryptography (ECC) Point Multiplier Core, http://www.ellipticsemi.com/products-clp-17.php.
[5] Global Environment for Network Innovations, http://www.geni.net/.
[6] The OpenFlow Switch Consortium: www.openflowswitch.org.
[7] RFC 1305 - Network Time Protocol. 1992.
[8] Juniper Networks Delivers Platform for Customer and Partner Application Development. In *Press Release*, Dec. 2007.
[9] Cisco Opens Routers to Customers and Third-Party Applications. In *Press Release*, April 2008.
[10] T. Akin. Hardening Cisco Routers. *O'Reilly*, 2002.
[11] R. Alimi, Y. Wang, and Y. R. Yang. Shadow configuration as a network management primitive. *SIGCOMM*, 2008.
[12] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *SOSP*, 2001.
[13] K. Argyraki and D. R. Cheriton. Loose Source Routing as a Mechanism for Traffic Policies. In *ACM SIGCOMM Workshops*, 2004.
[14] K. Argyraki and D. R. Cheriton. Active Internet Traffic Filtering: Real-time Response to Denialof-Service Attacks. In *USENIX Tech. Conf.*, 2005.
[15] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker. Off by Default! In *ACM HotNets*, 2005.
[16] R. Beverly and S. Bauer. The Spoofer project: Inferring the extent of source address filtering on the Internet. In *SRUTI Workshop*, 2005.
[17] K. L. Calvert, J. Griffioen, and S. Wen. Lightweight Network Support for Scalable End-to-End Services. In *ACM SIGCOMM*, August 2002.
[18] 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.
[19] K. Fall. A Delay-Tolerant Network Architecture for Challenged Internets. In *ACM SIGCOMM*, 2003.
[20] S. Guha and P. Francis. An End-Middle-End Approach to Connection Establishment. In *ACM SIGCOMM*, 2007.
[21] H. W. Holbrook and D. R. Cheriton. IP multicast channels: EXPRESS support for large-scale single-source applications. *ACM SIGCOMM*, 1999.
[22] J. Ioannidis and S. M. Bellovin. Implementing Pushback: Router-Based Defense Against DDoS Attacks. In *NDDS*, 2002.
[23] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. *ACM SIGCOMM*, 2002.
[24] X. Liu, A. Li, X. Yang, and D. Wetherall. Passport: Secure and Adoptable Source Authentication. In *USENIX NSDI*, 2008.
[25] X. Liu, X. Yang, and Y. Lu. To Filter or to Authorize: Network-Layer DoS Defense Against Multimillion-node Botnets. In *ACM SIGCOMM*, 2008.
[26] 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.
[27] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek. The click modular router. *SIGOPS Oper. Syst. Rev.*, 33(5):217–231, 1999.
[28] B. Parno, A. Perrig, and D. G. Andersen. SNAPP: Stateless Network-Authenticated Path Pinning. In *ACM ASIACCS*, 2008.
[29] 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.
[30] B. Raghavan and A. C. Snoeren. A System for Authenticated Policy-Compliant Routing. In *ACM SIGCOMM*, 2004.
[31] T. Roscoe, S. Hand, R. Isaacs, R. Mortier, and P. Jardetzky. Predicate Routing: Enabling Controlled Networking. In *ACM Hotnets*, 2002.
[32] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *ACM SIGCOMM*, 2002.
[33] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A Survey of Active Network Research. *IEEE Commun.*, 1997.
[34] D. L. Tennenhouse and D. J. Wetherall. Towards an Active Network Architecture. *SIGCOMM Comput. Commun. Rev.*, 37(5), 2007.
[35] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes no longer considered harmful. In *OSDI*, 2004.
[36] W. Xu and J. Rexford. MIRO: Multi-path Interdomain ROuting. In *ACM SIGCOMM*, 2006.
[37] 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.
[38] X. Yang, D. J. Wetherall, and T. Anderson. A DoS-limiting Network Architecture. In *ACM SIGCOMM*, 2005.