Appearing in *IEEE International Conference on Field-Programmable Technology* (FPT 2014), December 10–12, 2014

RotoRouter: Router Support for Endpoint-Authorized Decentralized Traffic Filtering to Prevent DoS Attacks

Albert Kwon, Kaiyu Zhang, Perk Lun Lim, Yuchen Pan,
Jonathan M. Smith, André DeHon
Dept. of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA, USA
Email: kwonal@mit.edu

***Abstract*—RotoRouter addresses Denial-of-Service (DoS) attacks on networks with a novel protocol and router implementation. Sets of RotoRouters cooperate in detecting and filtering out invalid network traffic before it reaches network endpoints; a new router-enforceable connection protocol queries destination endpoints to authorize traffic flows and uses per-packet digital signatures to distinguish allowed from disallowed connections. A RotoRouter prototype was implemented on a four-port 1000BASE-T NetFPGA-10G platform and supports 1024 simultaneous active connections using 74 BRAMs (less than one quarter of the available NetFPGA-10G BRAMs). It is able to sustain 800 Mbps per port throughputs for 1500B packets with less than 0.3μs latency, even during a DoS attack. With additional logic and memory resources, the required validation and switching operations scale to port speeds in excess of 10 Gbps and links with more than 10,000 active flows.**

***Index Terms*—Routing, Denial-of-Service, Software Defined Networking, NetFPGA**

I. Introduction

Network security is an urgent need as constant connectivity is expected in both civilian and military contexts. Solutions exist for securing networks against some security threats, *e.g.*, IPsec for guaranteeing confidentiality and integrity [20], firewalls [6] for limiting exposure to malicious traffic and filtering out known malicious attacks, and intrusion detection systems for threat awareness [27]. Threats to network availability however, such as denial-of-service (DoS) and distributed denial-of-service (DDoS) attacks, remain very difficult to address. The Internet's blend [7] of anonymity, ability to connect to any node, and best effort packet-forwarding work in favor of the attacker and make DoS attacks very hard to prevent. As an example, in September 2012, many large banks, including Bank of America and Wells Fargo, were attacked by botnets, making the websites unavailable for hours [32].

One potential solution is to allow communications only between known parties. This can be enforced with IPsec [20] by taking advantage of its built-in handshake. Unwanted packets (*e.g.*, packets from botnets, which are commonly used for distributed DoS) can then be identified and filtered accordingly. However, this does not fundamentally solve the DoS problem: any machine connected to the Internet can still send packets to any destination and therefore nothing prevents an attacker from saturating a host's or Internet Service Provider's network ports.

To mitigate attacks on network availability, the routers in the network must assist with the filtering. In particular, if the router can validate the packet source and distinguish between wanted and unwanted traffic, then it can filter out any bad packets before they reach the endpoint. Our Route-orderly-traffic-only Router (RotoRouter) aims to do exactly this. We add a public-key verifiable connection header so that packets can be securely associated with a particular connection and introduce a protocol to allow the router to verify that the destination has agreed to participate in the connection. This allows the router to filter out any packets that are not associated with a connection that the destination endpoint approves as desirable.

Our work establishes that the proposed router support is feasible and inexpensive, and demonstrates that the new protocol is an effective and promising solution to DoS attacks. Our contributions include:

1) A new router-enforceable protocol for DoS prevention that extends IPsec.
2) A router design that supports connection enforcement for IBR [14], including DoS prevention.
3) A hardware router architecture supporting the protocol.
4) An implementation on the NetFPGA-10G platform [4] and evaluations of throughput, latency, and area.
5) Demonstration of dynamic behavior under DoS attack.
6) Assessment of the computational requirements of the design and its scalability to higher link speeds.

II. Background

This section reviews component technologies for RotoRouter, and discusses related work.

IPsec: We exploit the IPsec [20] protocol, which provides confidentiality and integrity for IP traffic on a per-packet basis. Specifically, we use the Encapsulating Security Payload (ESP) packet format where the payload is AES-encrypted for privacy and validity.

IBR: We also use Introduction Based Routing (IBR) [14], a distributed and scalable IPsec key management tool leveraging on-node reputations for authenticating communicating nodes. IBR allows hosts to establish secure IPsec connections and provides a basis for hosts to decide when to accept connections and to share information about good and bad network actors. IBR maintains a network of trust relations between nodes and creates incentives for them to not misbehave. The benefit of IBR is the reduced ability of repeatedly misbehaving nodes to communicate with other nodes through feedback. IBR alone does not address DoS; without network cooperation, the host network link can still be flooded with unwanted traffic.

Programmable Routers, Software Defined Networking, and OpenFlow: High-performance router architectures [29] typically consist of a *switch fabric* interconnecting line cards that carry out packet processing tasks such as address lookups (in the forwarding information base, or FIB), checksum computations and framing. Control plane processing (such as route updates, reflected in the route information base, or RIB)) is separated [38] from forwarding operations that are required on a per-packet basis. OpenFlow [23] routers have used this separation and a standardized interface to flow tables (which contain per-connection information such as source and destination addresses and port numbers) to enable programmable control planes.

While not OpenFlow compatible, ROTOROUTER's architecture is similar to that of an OpenFlow switch, *i.e.* a switching plane and flow table(s) to route defined flows in hardware. New flows are processed by a software control plane which installs appropriate flow rules. OpenFlow switches typically use *centralized* controllers and a single global policy, while ROTOROUTER uses *decentralized* control, where the endpoint hosts authorize flows. While we will describe how our solution integrates with IBR (Sec. VI), the solution is orthogonal to the specific policy the endpoints use to authoritze connections; a ROTOROUTER simply implements the decisions that the endpoints make.

NetFPGA: NetFPGA [15] is an FPGA board that supports prototyping and experimenting with network protocols and network hardware. We use the NetFPGA-10G board that supports four 10 Gbps network links [4]. The gross structure of the NetFPGA OpenFlow router [25] is similar to ours, but lacks the hardware packet verification.

Prior Work on DoS Defense: DoS and DDoS have been ongoing concerns for more than a decade. This has motivated several approaches [30] to limiting its effects. Ingress validation attempts to prevent source address spoofing by validating the packets as they enter the network [21]; this, however, depends on trust of the entire router infrastructure. Many approaches attempt to statistically identify DoS traffic and filter it out; these filtering schemes have non-zero false-positive and false-negative rates. Traceback and marking approaches attempt to identify the point of origin for packets once a flow has been identified as bad. For example, StackPi [37] identifies paths from the source so that packets can be attributed to their point of origin independent of the packet's labeled IP address to provide a basis for filtering and pushback; this depends on wide acceptance within the router infrastructure and is also susceptible to imperfect classification and hence false positives. Active defenses based on pushback [18] towards the sources of aggregated traffic are intended to be implemented in IP routers and depend on reliable identification of packet flows associated with the DoS attack.

Traffic Validation Architecture (TVA) [39] is closely related to ROTOROUTER; TVA also validates connections with the destination. TVA relies on packets taking fixed route paths through the network and an attacker ignorant of the credentials; given this, TVA is able to avoid public key cryptography and the need to provide credentials along with every packet, but requires per-router secrets. TVA simulations have demonstrated impressive scalability and used a kernel extension for a software emulation of the required routing functionality. ROTOROUTER is a concrete hardware implementation of the required router with extensive support for cryptographic operations; while we report on support for IBR [14], ROTOROUTER could accelerate TVA or similar schemes.

III. ROTOROUTER IDEA

The ROTOROUTER is a router, *i.e.*, a network layer store-and-forward packet switch. Many advantages of packet switching stem from incorporating information in each packet adequate to reach its destination, requiring routers to maintain very little if any state to process the packet. In practice, the addition of "soft" (not required but useful) state can increase performance [26] in common cases such as that of TCP/IP connections, where state created early in the connection's life can accelerate processing for later packets associated with that same logical connection.

DoS/DDoS provides some especially interesting opportunities for use of soft-states, since the important decision is whether to drop or not drop packets belonging to a connection. This decision can be made after a few packets have already been forwarded, and will still be an effective protection against a DoS attack.

To allow for efficient filtering and verification before the packets reach an endpoint, we need to enable the router to validate traffic going through it. We call the set of packets associated with an end-to-end IP connection (same source and destination IP and port) a *flow*. We give each flow a unique *connection ID* (Sec. V-A) that allows the router to differentiate flows without exposing additional information about the connection.

To prevent spoofing of flows, we add a field called the *connection signature* (Sec. V-B). The connection signature is a public-key digital signature generated by the source using its private key that allows anyone with the public key to verify that (1) the packet originated from the source, and (2) the packet has not been modified since its initial dispatch. The router can use this signature to determine if the packet is legitimate.

Finally, the router must understand allowed and disallowed flows to filter malicious packets. The router by itself cannot determine this since the endpoints make their own decisions about which flows, and hence which packets, they wish to allow. As such, we enable the endpoints to provide this information to the router. When a router sees a flow for the first time, it asks the destination endpoint if this flow is wanted, and acts according to the answer. Therefore, any router can learn about the connection, and routers do not need any global coordination to setup new flows. If a path is rerouted during the connection, the new routers seeing the flow can also validate the flow with the destination.

IV. THREAT MODEL

Our threat model is an active, malicious attacker who can: (1) launch a DoS attack on any node connected to the Internet, (2) can sniff any packets in the network, (3) can insert any packets into the network (including controlling the headers), and (4)

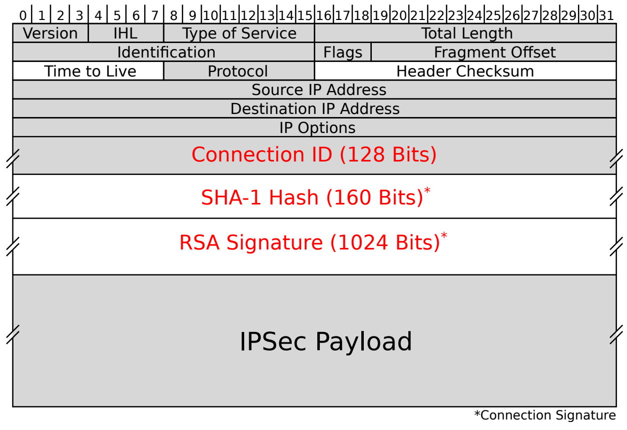


Fig. 1: A typical packet for a router-enforceable connection protocol. Fields included in the connection signature SHA checksum are shaded grey. Added Connection ID and Signature fields are shown in red.

can modify or replace routers under his/her control (including injecting, duplicating, modifying, removing, or misrouting packets). The attacker cannot see the secrets that exist within a good host (a compromised host is reclassified as an attacker). We do not limit the attackers capabilities outside of the good hosts and routers, but we do assume the majority of the routers are not actively participating in the attack.

To prevent DoS in this setting, a ROTOROUTER must identify and isolate the packets that come from the attacker and filter them at line-rate. Also, we shouldn't interfere with legitimate activities by dropping benign packets. The ROTOROUTER protocol must be resistant to attacks during both setup and normal operation; the attacker may attempt to create packets to either break or bypass the protocol, or slow it down (thus DoSing the router). Finally, the router should be able to identify misbehaving routers and limit damage they might cause.

V. ROUTER-ENFORCEABLE CONNECTION PROTOCOL

We introduce a new router-enforceable connection protocol that implements the idea described in Sec. III. We add two fields to the conventional IPv4 header, the connection ID and connection signature; the entire payload is IPsec encrypted. The augmented packet format is shown in Fig. 1. We next describe the protocol and the components in more detail.

A. Connection ID

The conventional Internet uses datagram (best effort) forwarding, and Internet routers can determine the next hop using only the packet's destination address. While this scheme eases the task of routers, it also makes it easy for attackers to send packets used for DoS. Attackers only need to know the destination system's IP address to attack it.

To address this issue, we use a connection ID that identifies both the source and the destination. It is assigned when the two endpoints agree to establish a connection. For IPv4 in particular, IDs are a 128 bit number that includes the source and destination IPs (64 bits) and a 64 bit random number. The random number allows unique port-to-port flows without exposing the port numbers to traffic snooping. By using random bits with the source-destination pair, we make it more difficult for attackers to obtain a valid ID. This also makes flows directional; a packet coming from an attacker to a well behaved node is different from the reverse path.

B. Signature

The connection ID by itself does not prevent spoofing attacks, nor does it prevent a third party from modifying the packet. To mitigate these issues, we add the connection signature. The connection signature is a digital signature based on SHA. First, using SHA-1, the endpoint computes the hash value of the connection ID, the payload, and the IP Header except for the IP checksum and time-to-live. The source then digitally signs the result of SHA with RSA using its private key [34], and the signature is appended to the IP header as the connection signature. The connection signature is 1184 bits: 160 bits for the SHA-1 hash, and 1024 bits for the RSA signature on the hash.

This signature is checked by the router at every step: each router can acquire the public key, compute the SHA based on the packet, and verify the signature using the public key and its SHA result. Any packet with a signature mismatch is discarded. This guarantees two things: (1) integrity and (2) authenticity of data. The SHA behaves as an integrity check on the packet (no tampering). The signing of SHA using the RSA private key guarantees that the packet indeed came from the claimed source. Moreover, IPsec already includes a sequence number that can be used in replay attack detection.

The public-key signature does place a large signing burden on the sender that will demand hardware support in the host network interface card. A Xilinx Virtex6-LX240T (40nm) can support RSA decryption at 23 Mbps [19]. Since we only need to encrypt the signature portion of the packet, on a 1500 byte packet (12,000 bits), 23 Mbps supports $23\ \mathrm{M} \times \left(\frac{12000}{1024}\right) \approx$ 270 Mbps packet throughput.

C. Key Management

In our protocol, a private key for the source is required to sign the packets, and the public key associated with the private key is required for validation of the signature. We call this key pair the *connection key pair*. A public/private key pair is generated or supplied when a connection is established. To avoid requiring any router state, we leave the key management entirely to the endpoints (Sec. VI). That is, once the key pair is generated, the source holds the private key, and the destination holds the public key. When a router sees the flow for the first time, the router requests the public key from the destination. This allows us to have *no* private state in the router.

Note that the public key used for connection validation is separate from the symmetric key used by IPsec for confidentiality. *The message contents remain opaque to the router.*

VI. NEW CONNECTIONS

Here, we discuss how new connections are set up, and the ROTOROUTER's role in such an environment.

A. Introductions

For establishing new connections, we use the IBR protocol as the baseline (Section II). IBR begins with *a priori* connections between certain nodes that trust each other (*e.g.*, through exchange of keys via offline methods). Any connection thereafter will be bootstrapped from these *a priori* connections. To illustrate how connections are established, consider a simple network with three nodes A, B and C. A and B have a pre-established connection as do B and C. If A wants to establish a connection with C, A must request an *introduction* from B. B communicates with C about the request, and if C accepts the introduction, then a connection between A and C is established. A connection between A and C persists until either chooses to close it. At the end of the connection, both will exchange feedback on whether they have behaved positively or negatively during the connection, and also forward the feedback to *introducer* B. In the event that A has been misbehaving towards B, the reputation of A with B is going to be negative. When A requests an *introduction* from B to C, B can refuse the introduction (*introduction denied*). Also if C has a negative reputation for A, C can refuse the introduction (*introduction declined*). With IBR, each node maintains discretion over authenticating nodes attempting to connect. Over time, trusted relations between nodes are established and the network will isolate misbehaving nodes.

One of the main features of IBR is its key management. The protocol provides a secure way to exchange connection keys used for IPsec through introductions. Taking advantage of the end-to-end connections and key-management of IBR, we extend IBR using the ideas of our router-enforceable connection protocol. We associate the end-to-end connection with two connection IDs (one for each direction), and distribute the connection public key used for verifying packets through the same channel used for the IPsec keys.

B. Connection Validation

Once the connection keys have been distributed, the router can ask the destination for the public key associated with the flow for signature validation. The validation request flows along an established connection like all other traffic in the system. An introduction may be needed if the router has not yet been in contact with a specific destination; as a result the router will have a connection ID and IPsec session ID key to communicate securely with the destination endpoint. The router sends a verification request packet that contains the connection ID of the new flow as the payload, and the endpoint replies with a packet that contains the allowed field and the public key of the connection. Messages used for validation are also encrypted IPsec packets using an ESP payload.

Since these connection verification requests and responses occur over established connections, an attacker cannot exploit them to mount an DoS attack. Section IX describes how we deal with attackers sending excessive connections into the router. If the router itself tries to send excessive traffic over one of its connections, that connection will be shut down and not allow reconnection. We also discuss more broadly how we deal with rogue routers in Section IX.

VII. RotoRouter Architecture

The basic architecture of RotoRouter derives from that of traditional IP routers (*e.g.* [29], [35]), However, to allow for efficient enforcement of the protocol described in Sec. V, the RotoRouter incorporates several additional specialized hardware components. Here, we highlight the key modules that differentiate RotoRouter, and describe our prototyping efforts. Fig. 2 shows the basic architecture.

A. On-chip Processor

When a flow is first seen, the router must ask the endpoints to provide the necessary information about the flow. To set up the new flow, we provide a processor on-chip, to which the IPv4 header and the connection ID are passed. The on-chip processor runs software to generate the request message to the destination. The whole packet for the new flow is buffered while servicing a miss, and all ports (including the input port that received the packet that missed) still operate normally without blocking for flows that are already in the flow table. If the result from the destination disallows the packet from moving forward, the router discards the packet. Otherwise, it is routed normally. The processor also maintains a route table which it consults to route the validation request to the destination endpoint and to configure the flow table.

As a latency optimization for setting up new flows, we allow a small number of packets to pass through the router until the destination responds to the public key request. This avoids a large setup latency for new connections due to waiting for the key. If too many packets are received before the public key is returned, or the response from the sink disallows the flow, then the router can cache the flow as malicious, and filter accordingly. This is a tradeoff between the volume of DoS traffic we allow through and the latency impact on valid traffic; a modest number of packets from a previously behaving link should not present an appreciable DoS load on the endpoint.

B. Flow Table

To avoid contacting the endpoint for each packet, we use a flow table to cache the current working set of active flows through the router. This *flow table* maps a connection ID to a set of information about the flow. Every connection ID (and thus flow) is associated with a tuple of: (i) a valid flow boolean, (ii) an RSA public key, (iii) a suspicious flow boolean, (iv) an output port, and (v) a non-validated-packet count. The valid flow boolean value indicates if the connection should be allowed, and the public key, associated with the private key of the source, is used to validate the connection signature.

Upon receiving a new packet, the connection ID is used as the key for the flow table. If the flow is found in the table, then the router checks the validity of the packet and makes the decision to route based on the allowed field. If the flow is not found in the table, then software is invoked to resolve the miss on the on-board processor described in Sec. VII-A. When a flow is inserted into the flow table, the forwarding port is also computed from the associated IP source and destination and inserted with the flow.

Because the flow table is accessed for every packet, fast table access is crucial to the overall performance of the router. We therefore cannot keep all possible flows in our table. To

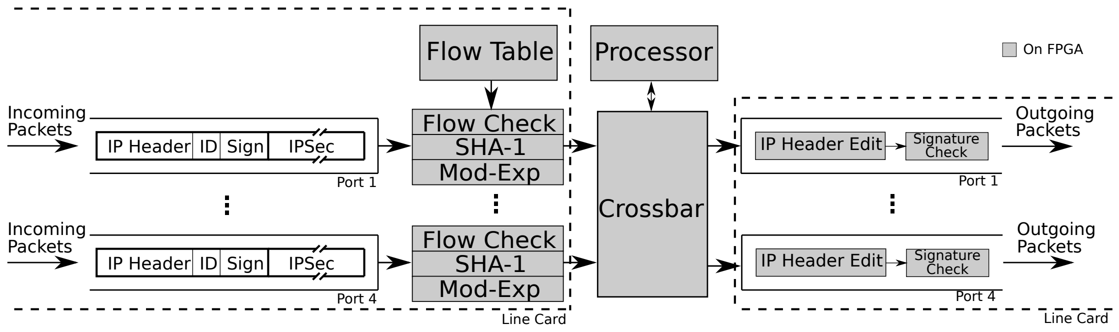


Fig. 2: Overview of ROTOROUTER Architecture

resolve this, the flow table in the router acts as a cache of active flows in our architecture. An effective table size is readily achievable; *e.g.*, [3] suggests that the number of active flows in a datacenter rack is under 10,000. Furthermore, to avoid unnecessary conflict misses, a content-addressable-memory (CAM) should be used. We use the dynamic-multi-hash-cache (dMHC) architecture, which gives us near associative cache-miss rates while using fewer resources on the platform and increasing throughput compared to a conventional CAM [10].

We note here that the flow table caches *negative* results as well. That is, there are flows in the table that specify that a connection is disallowed. This is especially useful for DoS attack. Once the router sees packets the destination considers as part of a DoS attack, it caches the connection as disallowed. Thereafter, every packet associated with the connection will be discarded quickly without placing a burden on the processor or slowing the resolution of legimate packets.

In our current architecture, a single flow table with a first-in-first-out (FIFO) queue has enough bandwidth to service all ports. For higher throughput switches, it may be useful to have a flow table per port or line card (Sec. X).

C. On-chip Cryptography

For verification of the connection signature, we dedicate two hardware cores per input port in ROTOROUTER: (1) SHA-1 [11] and (2) modular-exponentiation (mod-exp). The SHA-1 core is used to compute the hash over the fields highlighted in Fig. 1 and to verify the integrity of the packet. However, computing the hash for every packet before routing could result in an increase in the latency, because the router would be forced to use store-and-forward, meaning the router latency would be $12\mu s$ for a 1500 byte packet. We can avoid this problem by initially allowing cut-through routing and verifying the hash of the packet after routing. If the signature does not match, then we mark it in our flow table as suspicious, and we force future packets on the flow to be held for signature verification. This way, no additional latency is added for packet validation during the normal, non-attack, case. Allowing just one packet through initially is insufficient to cause DoS, so doing late verification does not undermine the strength of our protection.

The connection signature also requires a modular exponentiation to verify the authenticity. The verification of the digital signature on the SHA-1 hash prevents spoofing and tampering of packets. We perform the validation on every packet, so the efficiency of the mod-exp module is crucial for the performance of our router.

Unfortunately, a pure software implementation of mod-exp cannot support the 1 Gbps line rate. In the ROTOROUTER design, we dedicate significant hardware exclusively to doing mod-exp operations. A naive hardware implementation of mod-exp is also too slow. For faster mod-exp, we employ Montgomery Multiplication [24]. Montgomery Multiplication breaks the single, expensive step of a modular division into multiple fast steps of bitshift and addition. Despite the extra steps, the resulting latency for a single Montgomery Multiplication operation is lower than that of a naive implmentation. Furthermore, we pipelined our design between the multipliers to increase our throughput. Each multiplier is currently a sequential set of add and Montgomery reduce steps.

Even with Montgomery Multiplication, a large public exponent would still result in a large, slow mod-exp computation. We mitigate this issue by requiring a small public key exponent (such as 17). Since we are only using the keys for signing (not data encryption), the usage of such small public keys is acceptable [12].

VIII. RESULTS

In this section, we highlight our implementation results, summarizing the resources required and the performance achieved both in a normal setting and when under a DoS attack.

A. NetFPGA Prototype

To evaluate the hardware impact of the security features of ROTOROUTER, we implemented the design on the NetFPGA-10G platform [4]. The platform contains a Xilinx Virtex 5 (65 nm) FPGA (xc5vtx240tffg1759-2) [36] for hardware prototyping and four SFP+ modules that support 1 and 10 Gbps connections. The core of ROTOROUTER was developed using Bluespec System Verilog [5] including the processor, crossbar, mod-exp, and flow-table. Many of the infrastructure elements necessary for exterior communication (such as gigabit Ethernet and PCIe) and interior communication (between different modules) were provided by the NetFPGA open source library [2]. For our prototype, we are using 1 Gbps connections to work with our commodity workstation ethernet 1000Base-T NICs.

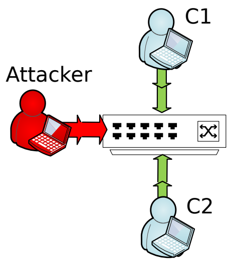
The resource utilization is shown in Tab. I. ROTOROUTER uses approximately 5 times the resources of an open source

TABLE I: FPGA Resource Utilization

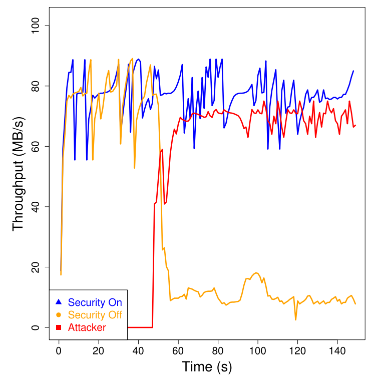
Module	Area LUTs	Area BRAMs	Clock (MHz)
Crossbar w/ Buffers	8249	16	300
Flow Table	38	74	350
Processor	26985	52	200
SHA-1 Module [17]	4×1005	0	125
Mod-Exp	73591	0	200
RotoRouter	112883	142	125
IPv4 Router [1]	22523	35	150
Total available	149760	324	-

TABLE II: Module Throughput

	Crossbar	Flow Table	SHA-1	Mod-Exp
Clock Speed (MHz)	300	350	125	200
Individual Throughput (Gbps)	19.2	515	4×0.8	4×1.2
Effective Throughput @ 125 MHz (Gbps)	8	184	3.2	4.8



(a) Experimental Setup



(b) Goodput across the RotoRouter and attacker traffic

Fig. 3: Experimental Result

IPv4 router implementation [1]. The security modules (processor, SHA-1, and mod-exp) that are uncommon to conventional router add most of the hardware overhead, with the mod-exp modules consuming considerable hardware resources due to the signature size.

For the basic design, we have two BRAMs FIFOs on each input and output port. Each 36Kb BRAM on the Virtex 5 FPGA can hold three 8×1500 byte=12,000b packets. These can easily be enlarged by using additional BRAMs.

B. Throughput of Modules

Based on the clock rates shown in Tab. I, we can compute the theoretical limits of the throughput of the individual modules. The current implementation of crossbar runs at 300 MHz and routes 64 bits per cycle. The throughput is therefore $350 \cdot 64 = 19.2$ Gbps. The flow table runs at 350 MHz, and with minimal packets size of 184 bytes, the total throughput is greater than $350 \cdot 184 \cdot 8 = 515$ Gbps. The SHA-1 core runs at 125 MHz and has 81 rounds/block with 512 bits/block, so it has a throughput of $(125 \cdot 512)/81 = 790$ Mbps. Finally, rated at a clock speed of 200 MHz and processing 1024 bits/2048 cycles, the Montgomery Multiplication module has a throughput of 100 Mbps. This throughput is still sufficient to handle 1 Gbps connections since it is applied to only 1024b of every packet. As a result, the 100 Mbps is effectively $8 \times 1500/1024 \cdot 100$ Mbps = 1.2 Gbps for 1500 byte packets.

C. Router Throughput

The throughput of RotoRouter is limited by the slowest module needed for packet verification. With our current implementation, the SHA-1 core limits the port throughput to 800 Mbps. The integrated design runs at 125 MHz and moves packets in 64-bit blocks, giving us $125 \cdot 64 = 8$ Gbps of throughput. We run the mod-exp unit on a separate clock at 200 MHz to prevent it from becoming a bottleneck. The throughput of different modules is summarized in Tab. II.

D. Impact of Attack

To demonstrate the effectiveness of our protection, we set up an isolated network testing environment using commodity workstations and a sample attack workload against a video chat application (See Fig. 3a). We have two Debian Sid Linux machines with Asus P8C motherboard with Gigabit Ethernet ports (labeled C1 and C2) communicating through our RotoRouter prototype and an attacker who performs a DoS attack by flooding the network with junk messages. We measure the throughput first with the router security features disabled (thus a typical IPv4 router) and then with the features enabled. In the goodput (throughput of legitimate data) graph shown in Fig. 3b, the attacker starts the attack at 50s. We can see that the goodput across the router without the security features drop immediately once the attack starts. However, *with the security features enabled, we see no drop in the goodput across the router.*

E. Latency Impact

In Sec. VII-C, we stated that using store-and-forward routing, the latency is at least 12μs. Verification using the mod-exp would also require 2112×4 5 ns cycles or 42 μs total over the mod-exp's 4 stages. However, using cut-through routing, we reduce this greatly. The Ethernet MAC core takes 12 cycles to convert incoming raw frames to data the router can understand, and the same is true for outgoing data. The flow table has a two cycle pipelined lookup, and the crossbar takes two cycles. This gives us approximately 28 cycle latency, which is 224 ns at 125 MHz. We see no difference in latency under attack compared with normal conditions.

IX. Attack Defenses

RotoRouter's protocol and hardware are designed to prevent traditional Denial-of-Service attacks, but they open up potential attack surfaces. This section describes two possible attacks on RotoRouter and how we address them.

Connection Attack: Because establishing new connections is costly for our router, it could be vulnerable to a new kind of DoS attack where the attacker repeatedly sends packets for bogus new connections. To prevent such an attack, RotoRouter keeps a counter for invalid connection requests arriving through an input port per unit time. When this exceeds a set threshold, the router notes that it is under attack and shuts down the port to prevent further attacks. Since neither a proper endpoint nor a proper upstream RotoRouter will forward **any** bogus connection requests, the receipt of a large number of bogus connection requests is a definite indication of misbehavior. The network then treats this disabled port as a down link and reconfigures around it, using the path vector routing protocol [33]. This can be seen as the logical equivalent of TCP SYN flooding. More primitive attempts at TCP SYN flooding are not possible since all traffic in IBR must flow over established connections.

Rogue Routers: Our dependence on the router to protect the network poses an additional question: what should we do when one of our routers turns rogue?

We have identified three potential problems that could arise from rogue routers: (1) copying flows, (2) generating or forwarding bad traffic, and (3) misrouting the flows. The first is a non-problem as any information the router possesses is public information since the content of the flows are encrypted using IPsec. Rogue routers thus cannot leak any new information.

The second threat would be detected when a flow goes through another router. A good RotoRouter will detect that it is receiving flows that do not validate, and consequently will drop the packets. Furthermore, when the RotoRouter takes part in the IBR network, the reputation of the rogue router will be marked down as well.

The final problem, misrouting of flows, will be detected by failed end-to-end acknowledgments. If we can localize the misbehaving router (*e.g.* with a secure traceroute [28]), we can update the reachability graph using the path vector protocol to avoid it for future traffic. Another approach would be protocols robust to arbitrary (*Byzantine*) faults [31].

X. Scaling Requirements

Although the tested hardware uses 1 Gbps connections, the design is scalable to higher connection speeds, greater number of active flows, and routers with more ports. We sketch the feasibility and impact of scaling in this section. Since we use a standard control processor, line-card, and switch-fabric architecture, this separates port-scaling issues from line-card rate scaling, and we address each in turn.

Switching Fabric: Modern FPGAs easily support high speed switching fabrics in excess of 100 Gbps, *e.g.,*, the Grouped Crosspoint Queued switches provides switching throughput of 160 Gbps [8] when implemented on a Xilinx Virtex-6 (40 nm) FPGA. Newer generation FPGAs have even more I/O and switching capacity.

Line Cards: Aside from switching, logic and handling can be performed on a per port basis. The line card modules that must scale with the bandwidth are the flow table, the SHA-1 hash unit, and the mod-exp signature verifier. Each of these modules is used for verification of every packet, and thus scalability of the module is a necessary and sufficient condition for the line-card rate scalability. As we have shown (Sec. VIII-B), many modules can already run much faster than needed to support 1 Gbps.

The flow table is used once per packet to determine authorization and acquire the public key associated with the flow. With one flow table, we can easily service four 10 Gbps ports and perhaps four 100 Gbps ports. To scale up even further, we can replicate the flow table to one table per port. The flow table per port also has the advantage of supporting more active flows. To scale to even more active flows, each additional 1024 flows require 74 BRAMs. Supporting the 10,000 flows suggested by [3] will require 740 BRAMs, which modern FPGAs can accommodate.

We also require the hash computation to run at line rate since we need to compute SHA-1 for every packet. The simple SHA-1 core we currently use provides about 800 Mbps of throughput per core, allowing us to handle almost a gigabit throughput when replicated one per port. More complicated but faster implementation of SHA could support 10 Gbps connections per port [22]. For 100 Gbps and beyond, we can add multiple SHA-1 modules per port such that we can service multiple packets simultaneously. We used an SHA-1 hash for the prototype, but this can and should be replaced with a SHA-256 hash. A naive implementation of SHA-256 on FPGA has approximately 60% area overhead compared to SHA-1

The modular exponentiation (mod-exp) module is the one that requires the most work in order to scale to higher speeds. In order to get the mod-exp module to scale up to 10 Gbps or even 100 Gbps, there are two improvements that can be made. First, the design can be more heavily pipelined in order to increase the throughput. Second, multiple modules can be dedicated to each port in order to keep up with the rate at which packets arrive. Both throughput optimizations will require additional hardware resources.

Alternatively, it may make sense to use a different, less expensive public-key encryption scheme. Prior work shows that MQQ [16] can be decrypted at 400 Mbps in a Virtex 5 FPGA using only 7,000 LUTs [13]. Considering the need to only verify 160b of the packet, this will handle over 30 Gbps of traffic. The MQQ scheme also suggests lighter weight encryption for either a processor or FPGA, which would also address the host burden. Nonetheless, MQQ does not have the history of attention, analysis, and scrutiny of RSA.

Processors: For large systems, the flow setup load from the line cards may exceed the capabilities of a single processor. Multiple processors can be added, each serving a set of line cards. The processors may also serve as a shared L2 cache on connections for the flow tables.

XI. Future Work

We have characterized the throughput, resources, and goodput attack response, but additional development and analysis remains. For example, it is necessary to characterize the dynamic behavior of new flow setup, including latency, throughput impact, and rate limits. This will allow measurement and tuning of new protocol parameters such as the number of packets forwarded before receiving a validation and the rate of invalid packets allowed over a link. Further measurements of RotoRouter behavior in larger network settings will be valuable, particularly to understand the impact of the connection validation traffic.

We have shown that it is possible to realize an endpoint-authorized, decentralized traffic filtering router based around technologies readily available today. In this first existence proof of such a router, we integrate concrete technologies to use for the components (*i.e.* RSA signatures, SHA hashes, ESP packet encryption, IBR key management) and demonstrate reasonable performance for the integrated solution. Many of these components, such as ESP, are reusable in an IPv6 [9] implementation of ROTOROUTER. We also see several directions for improved ROTOROUTERs by replacing individual components within the architecture with current or future technologies (*e.g.*, a lighter-weight, public-key signing scheme, replacing SHA-1 with a more secure hash, *etc.*).

XII. CONCLUSIONS

Today's Internet remains vulnerable to DoS and DDoS attacks. Since DDoS attacks can choke off critical link bandwidths anywhere in the network, the problem cannot be solved by the endpoints alone. We show how routers can assist with DoS filtering inside the network while leaving the endpoint hosts in charge of the decisions about which traffic to accept and filter. As a result, we present a filtering scheme that is amenable to decentralized control of network packet filtering. While the ROTOROUTER requires more logic area and state than a minimal IPv4 router, our proof-of-concept router implementation shows that the area cost is not prohibitive, reasonable throughput is achievable, and the router is effective at mitigating DoS attacks. We were able to demonstrate a 4-port, 1000Base-T router in a 65 nm generation FPGA platform, the NetFPGA-10G, and we sketch a path to scale to higher ports, throughput, and flows on newer generation FPGAs.

ACKNOWLEDGMENT

This material is based upon work supported by the DARPA MRC program through the United States Air Force Research Laboratory (AFRL) under Contract No. FA8650-11-C-7189. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

REFERENCES

[1] NetFPGA reference router walkthrough. `https://github.com/NetFPGA/netfpga/wiki/ReferenceRouterWalkthrough`, 2013.
[2] NetFPGA web home. `http://netfpga.org`, 2013.
[3] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *Proc. ACM SIGCOMM IMC*, pages 267–280, 2010.
[4] M. Blott, J. Ellithorpe, N. McKeown, K. Vissers, and H. Zeng. FPGA research design platform fuels network advances. *Xilinx Xcell Journal*, pages 24–29, September 2010.
[5] Bluespec, Inc. Bluespec SystemVerilog.
[6] W. R. Cheswick and S. M. Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, 1994.
[7] D. D. Clark. The design philosophy of the DARPA Internet protocols. In *Proc. ACM SIGCOMM Conf.*, pages 106–114, 1988.
[8] Z. Dai and J. Zhu. Saturating the transceiver bandwidth: Switch fabric design on FPGAs. In *Proc. IEEE FCCM*, pages 67–75, 2012.
[9] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, December 1998.
[10] U. Dhawan and A. DeHon. Area-efficient near-associative memories on FPGAs. In *Proc. IEEE FCCM*, pages 191–200, 2013.
[11] D. Eastlake and P. Jones. US secure hash algorithm 1 (SHA1). RFC 3174, 2001.
[12] D. Eastlake 3rd. RSA/SHA-1 SIGs and RSA KEYs in the Domain Name System (DNS). RFC 3110 (Proposed Standard), May 2001.
[13] M. El-Hadedy, D. Gligoroski, and S. J. Knapskog. High performance implementation of a public key block cipher - MQQ, for FPGA platforms. In *Intl. Conf. on Reconfigurable Computing and FPGAs (ReConFig)*, pages 427–432, 2008.
[14] G. Frazier, Q. Duong, M. P. Wellman, and E. Petersen. Incentivizing responsible networking via introduction-based routing. In J. M. McCune, B. Balacheff, A. Perrig, A.-R. Sadeghi, A. Sasse, and Y. Beres, editors, *Trust and Trustworthy Computing*, volume 6740 of *Lecture Notes in Computer Science*, pages 277–293. Springer Berlin Heidelberg, 2011.
[15] G. Gibb, J. W. Lockwood, J. Naous, P. Hartke, and N. McKeown. NetFPGA: An open platform for teaching how to build gigabit-rate network switches and routers. *IEEE Trans. Educ.*, 51(3):364–369, 2008.
[16] D. Gligoroski, S. Markovski, and S. J. Knapskog. Multivariate quadratic trapdoor functions based on multivariate quadratic quasigroups. In *Proc. of the American Conf. on Applied Mathematics*, pages 44–49, 2008.
[17] P. Hartke. OpenCores SHA1, 2002.
[18] J. Ioannidis and S. M. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *Proc. NDSS*, 2002.
[19] Y. Ito, K. Nakano, and B. Song. The parallel FDFM processor core approach for CRT-based RSA decryption. *Intl. Journal of Networking and Computing*, 2(1):79–96, January 2012.
[20] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301, December 2005.
[21] X. Liu, A. Li, X. Yang, and D. Wetherall. Passport: Secure and Adoptable Source Authentication. In *Proc. NSDI*, 2008.
[22] Y. Liu, L. Wu, Y. Niu, X. Zhang, and Z. Gao. A high-speed SHA-1 IP core for 10 Gbps ethernet security processor. In *Intl. Conf. on Computational Intelligence and Security*, pages 237–241, 2012.
[23] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM CCR*, 38(2):69–74, Mar. 2008.
[24] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, pages 519–521, 1985.
[25] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown. Implementing an OpenFlow switch on the NetFPGA platform. In *ACM/IEEE ANCS*, 2008.
[26] P. Newman, G. Minshall, and T. Lyon. IP switching: ATM under IP. *IEEE/ACM Trans. on Networking*, 6(2):117–129, April 1998.
[27] S. Northcutt and J. Novak. *Network Intrusion Detection, Third Edition*. SAMS, 2002.
[28] V. N. Padmanabhan and D. R. Simon. Secure traceroute to detect faulty or malicious routing. In *Proc. ACM SIGCOMM*, August 2003.
[29] C. Partridge, P. P. Carvey, E. Burgess, I. Castineyra, T. Clarke, L. Graham, M. Hathaway, P. Herman, A. King, S. Kohalmi, T. Ma, J. Mcallen, T. Mendez, W. C. Milliken, R. Pettyjohn, J. Rokosz, J. Seeger, M. Sollins, S. Storch, B. Tober, G. D. Troxel, D. Waitzman, and S. Winterble. A 50-Gb/s IP router. *IEEE/ACM Trans. on Networking*, 6(3):237–248, June 1998.
[30] T. Peng, C. Leckie, and K. Ramamohanarao. Survey of network-based defense mechanisms countering the DoS and DDoS problems. *ACM Computing Surveys*, 39(1), 2007.
[31] R. Perlman. *Network layer protocols with Byzantine robustness*. PhD thesis, MIT, 1988.
[32] N. Perlroth. Attacks on 6 Banks Frustrate Customers. `http://www.nytimes.com/2012/10/01/business/cyberattacks-on-6-american-banks-frustrate-customers.html?_r=0`, September 30 2012.
[33] Y. Rekhter and P. Gross. Application of the Border Gateway Protocol in the Internet. RFC 1772 (Draft Standard), March 1995.
[34] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, Feb. 1978.
[35] G. Schelle and D. Grunwald. CUSP: a modular framework for high speed network applications on FPGAs. In *Proc. of the Intl. Symp. on FPGAs*, pages 246–257, 2005.
[36] Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124. *Virtex-5 Family Overview*, February 2009.
[37] A. Yaar, A. Perrig, and D. Song. StackPi: New packet marking and and filtering mechanisms for DDoS and IP spoofing defense. *IEEE JSAC*, 24(10):1853–1863, October 2006.
[38] L. Yang, R. Dantu, T. Anderson, and R. Gopal. Forwarding and Control Element Separation (ForCES) Framework. RFC 3746, 2004.
[39] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting network architecture. *SIGCOMM CCR*, 35(4):241–252, 2005.