Routing with Byzantine Robustness

Radia Perlman

Routing with Byzantine Robustness

Radia Perlman

SMLI TR-2005-146 September 2005

Abstract:

This paper describes how a network can continue to function in the presence of Byzantine failures. A *Byzantine* failure is one in which a node, instead of halting (as it would in a *fail-stop* failure), continues to operate, but incorrectly. It might lie about routing information, perform the routing algorithm itself flawlessly, but then fail to forward some class of packets correctly, or flood the network with garbage traffic.Our goal is to design a network so that as long as one nonfaulty path connects nonfaulty nodes A and B, they will be able to communicate, with some fair share of bandwidth, even if all the other components in the network are maximally malicious. We review work from 1988 that presented a network design that had that property, but required the network to be small enough so that every router could keep state proportional to n^2, where n is the total number of nodes in the network. This would work for a network of size on the order of a thousand nodes, but to build a large network, we need to introduce hierarchy.

This paper presents a new design, building on the original work, that works with hierarchical networks. This design not only defends against malicious routers, but because it guarantees fair allocation of resources, can mitigate against many other types of denial of service attacks.

Sun Labs
16 Network Circle
Menlo Park, CA 94025

email address:
radia.perlman@sun.com

Routing with Byzantine Robustness

Radia Perlman
radia.perlman@sun.com

Abstract

This paper describes how a network can continue to function in the presence of Byzantine failures. A *Byzantine* failure is one in which a node, instead of halting (as it would in a *fail-stop* failure), continues to operate, but incorrectly. It might lie about routing information, perform the routing algorithm itself flawlessly, but then fail to forward some class of packets correctly, or flood the network with garbage traffic. Our goal is to design a network so that as long as one nonfaulty path connects nonfaulty nodes A and B, they will be able to communicate, with some fair share of bandwidth, even if all the other components in the network are maximally malicious. We review work from 1988 that presented a network design that had that property, but required the network to be small enough so that every router could keep state proportional to n^2, where n is the total number of nodes in the network. This would work for a network of size on the order of a thousand nodes, but to build a large network, we need to introduce hierarchy. This paper presents a new design, building on the original work, that works with hierarchical networks. This design not only defends against malicious routers, but because it guarantees fair allocation of resources, can mitigate against many other types of denial of service attacks.

1. Introduction

This paper describes how to design a large, hierarchical network that guarantees packet delivery, with some fair share of bandwidth, even in the presence of Byzantine failures. A Byzantine failure is one in which a node misbehaves, rather than halting. For instance, a node might lie about connectivity, corrupt routing information from other nodes, flood the network with garbage traffic, or perform the routing protocol properly, but then fail to forward some types of traffic. Previous work, NPBR (Network Protocols with Byzantine Robustness) [15] has shown how to do this in a network which is small enough not to require hierarchy. However, NPBR does not extend in a straightforward way to a hierarchy. In this paper we show how to support hierarchy. It requires a different model than NPBR used for data packet delivery, a specific model of hierarchy designed to be supportive of Byzantine robustness, cryptographic protection of packets between endpoints of virtual links, and resource management designed so that routers need no more state than would be necessary to support their portion of the hierarchy.

Note that neither NPBR nor the design in this paper provides end-to-end guarantees other than that packets from source to destination will be delivered properly, with some fair share of network resources. It is possible that bogus or corrupted packets will be delivered to the destination, in addition to the real packets. End-to-end authentication and integrity must be provided by some other mechanism, such as Kerberos or a PKI-based protocol such as IPsec.

2. Review of NPBR

NPBR (Network Protocols with Byzantine Robustness)[15] was published in 1988. It presented a design for building a modest-sized network resilient to Byzantine failures. In this section we give an overview of NPBR, while noting (and sometimes fixing) some minor holes in NPBR as specified.

For simplicity of explanation, in this paper we assume a network in which all nodes are routers. The alternative, partitioning a network into routers and endnodes, is straightforward. Each router would be allocated some fair share of resources by NPBR, and the router would divide its share of resources fairly among the endnodes that it represents. So we will ignore endnodes in this paper, and use “node” and “router” interchangeably.

There are two basic forms of routing in NPBR.

- The first type is robust flooding, in which a packet generated by S is sent to all the other nodes. Although flooding could in theory be used for sending to a single destination, it is more suitable for information that should be sent to all the nodes. In particular, NPBR uses flooding for dissemination of routing information and public keys.
- The second type is data packet forwarding, in which the source selects and sets up a path to the destination, and the data packets from S to D follow that path.

NPBR uses the basic information (public keys, and routing topology information) disseminated through robust flooding to enable data packet forwarding. Using the link state information and public key information disseminated through robust flooding, a source node S chooses a path to the destination D, sets it up by creating state in the nodes along the path, (including reserving resources for the S-D conversation), and sends data on that path.

In section 2.1 we review NPBR's flooding, along with its uses. In section 2.2 we review NPBR's data packet delivery.

2.1 NPBR's Robust Flooding

NPBR flooding ensures that a source S's most recently generated packet will be delivered to each node D, provided that at least one correctly functioning path exists between S and D. If S issues a newer packet before D receives the current packet, then D might only receive the newer packet. At some point though, once S stops sending new packets, everyone (with a correctly functioning path to S) will receive the latest packet. Arbitrary Byzantine behavior of routers not on the path cannot prevent S's packet from being delivered to D in a reasonable amount of time.

To build up intuition for how NPBR achieves these goals, we start with traditional flooding.

Flooding is a routing algorithm in which each router R that receives a packet from neighbor N forwards the packet to each neighbor besides N. Flooding, in some theoretical sense, has the property that we want. A packet from S will reach D, provided S and D are connected by at least one correctly functioning path, regardless of arbitrary Byzantine behavior of routers and links not on that path, *provided the network has infinite capacity.*

Given that in reality networks do not have infinite capacity, the main issue in NPBR is assuring fair allocation of resources.

The finite resources in a network consist of:

- computation in the routers
- memory in the routers
- bandwidth on the links

We will solve the computation issue by requiring routers to have the capacity for processing all incoming packets at wire speed. This is possible, since the router has only a finite number of ports, and it can be engineered to have sufficient computational capacity to process all the packets that can be received on any of its attached links.

So we are left with enforcing fair allocation of memory and bandwidth.

We do that by requiring each router R to reserve a buffer for each possible source. To ensure that only a packet generated by the source occupies the buffer, the source digitally signs the packet. To prevent a malicious node from injecting old packets of S, competing with S's most recent packet, we also include a sequence number in the packet. The sequence number, like the rest of the packet, is protected by S's signature, so that no other node can forge a packet claiming to be from S, or modify any of the fields in the packet without detection.

The remaining finite resource is bandwidth. Each router enforces fairness on each of its outgoing links by ensuring that all the buffered packets have a share of the outgoing bandwidth. R will have n buffers (one for each source). R could constantly cycle through all these buffers, retransmitting each time that buffer's turn comes up, or R's neighbor could send acknowledgements for packets received, and R could just retransmit packets that have not yet been received by the neighbor.

Byzantine nodes are allowed to be arbitrarily malicious, but they cannot perform magic. Thus they cannot break cryptographic schemes.

Each packet contains a source address S, a signature, and a sequence number. If R receives a packet from S, R checks to see if S's signature is valid, and if so, whether the sequence number on the received packet is greater than the one R has stored for S in R's buffer reserved for S. If so, R overwrites S's buffer with the received packet, and floods the packet to all its neighbors.

Robust flooding is used by NPBR for two purposes: for a trusted node to inform all the routers of the public keys of all the other nodes, and for dissemination of link state information in a link state protocol.

2.1.1 Link state routing

Link state is a routing protocol in which each router first determines who its neighbors are (the state of its links), generates a *link state packet* asserting who it is and who its neighbors are, and floods the link state packet to all the other routers. Each router maintains a database consisting of the most recently generated link state packet from each other router. This information completely summarizes the topology of the network, and from that, each router can compute paths from itself to each other node. Examples of link state routing protocols are IS-IS, OSPF, and PNNI [14].

2.1.2 Distributing Ppublic keys

It would be onerous to require configuring every node with the public key of every other node. So instead, NPBR uses a trusted node, T, to securely inform the rest of the nodes of the public keys of the other nodes.

The trusted node T has a public key. T knows the public key of each node in the network, plus its own private key. Each router R must be configured with T's public key, as well as R's own private key. Because T's public key is known to all the nodes, T can use NPBR's flooding to distribute all the public keys of all the nodes.

To introduce a new node R into the network requires configuring R with T's public key, and configuring T with R's public key. Then T will flood an updated public key list so that all the other nodes can learn R's public key.

2.1.3 Byzantine failure of the trusted node

NPBR is resilient against Byzantine failures of the trusted node by allowing multiple trusted nodes. Rather than a voting scheme in which there would have to be at least 3 trusted nodes (to outvote the one that has failed), NPBR has each router devote resources to any public key announced by any trusted node, and an additional piece of configuration in each router consisting of the maximum number of routers in the area, so that no Byzantine trusted node can use up more than 1/k of the resources of the network (where k is the number of trusted nodes).

Note that a trusted node T cannot fail in any subtle way. If any router R notices that T is not advertising R's public key properly, R can raise an alert (through some out-of-band mechanism, because R cannot actually communicate if its public key is not known), and it will be easy to tell whether R or T is faulty. Or if there is more than one T, and they advertise different sets of keys, then any router will be able to detect that there is some problem with the trusted nodes.

2.2 NPBR's Data Packet Forwarding

There are two forms of data packet forwarding explored in NPBR.

- packet specified: In this form, the route is specified in the data packet, and the data packet (including the route) is signed by the source. To make this work, each router would need to reserve a buffer for each (source, destination) pair, each data packet would need to be digitally signed by the source and verified by the intermediate nodes, and the route would take up space in the header. Therefore, the second form is preferable:
- path setup: In this form, the route is set up by the source, using a digitally signed route setup packet, and then the source need not sign the data packets, and routers along the path are not burdened with verifying the source's signature when forwarding the data packets. Path setup is commonly used in connection-oriented network protocols such as ATM and MPLS [14].

Because each source has a complete link state database (within an area), each source S can choose a path to a destination D. In the path setup method, S chooses a path and then sends a cryptographically signed path setup packet specifying the chosen path, with the path setup packet following the specified path[1]. Each router R along the path specified by S verifies the signature in the route setup packet, and if S has not already used all the path state reserved at R for S, R remembers the expected input port and output port for packets from S to D, and reserves at least one buffer for the S-D flow. If R can reserve resources for n^2 flows, then R would never need to refuse to set up a flow, since it would be able to accommodate every (S,D) pair.

NPBR argues that an advantage of this form of data packet forwarding is that no cryptography is required for forwarding data packets. The argument is that if the path chosen is through correctly functioning routers and links, and as long as each router on the path checks that the packet is arriving on the expected input port, no Byzantine router off-path can disrupt the flow of packets on the path.

NPBR reserves resources (bandwidth and memory) for each flow that has been set up. Each router might reserve n^2 buffers for data packet forwarding; one for each possible (source, destination) pair (allowing each source to set up a flow through that router to each destination). Or it might reserve a fixed number, k, of buffers for data for a particular source S, and refuse to store more than k routes for S. Or it might reserve k buffers for each source, plus a pool of extra buffers that sources can use on an as-needed basis, if they are not already used. And of course, it could reserve j buffers per flow rather than one, if it had the resources.

2.2.1 Byzantine Links

It is possible for links to have Byzantine failures, where a link fails to forward packets properly, depending on factors such as addresses in the header, packet size, or bit patterns in the data. A link might pass all diagnostics, and yet in certain obscure cases not forward packets properly. This happens in practice without malicious errors. It is even more likely to happen when there are active components such as bridges or repeaters in the path between "neighbors".

NPBR considers the issue of a malicious node C that tricks honest nodes A and B into thinking they have a link to each other (an A-B link), by relaying messages between

1. Actually, NPBR recommends flooding the route setup packet for the S-D path throughout the area in order to garbage-collect any pre-existing S-D path state, but that is a detail orthogonal to the points in this paper.

them. It isn't enough for A and B to determine the fraction of traffic that is successfully transmitted on the link. If almost all the traffic gets through, but C is purposely throwing away one type of traffic -- say, traffic from source S, it would be nice for A and B to detect that. It isn't strictly necessary, since S can always choose a different path. However, the more Byzantine components in the network, the harder it will be for S to find a path.

NPBR discusses foiling the ability of malicious component C to distinguish between packets (and therefore cause Byzantine behavior of a link between two honest nodes), by having A and B foil traffic analysis of any intermediate component by employing techniques such as

- using link encryption on the link between A and B to hide header information and bit patterns in the data
- sending fixed-size packets (by padding or fragmenting as needed)

To avoid other forms of traffic analysis, NPBR should have mentioned sending traffic at a steady rate (regardless of whether there is any real data to send), and not sending traffic strictly round-robin (in case every nth packet is dropped), but rather randomizing the order of flows.

In order for A and B to employ encryption, they have to have a shared secret key. Fortuitously, NPBR securely distributes public keys, so A and B can do mutual authentication and establish a shared secret key using their public keys. This shared secret key, and any associated state is known as a *security association*.

As we will see later in this paper, we will extend this notion of establishing security associations between neighbors to *virtual links*, which in a hierarchy appears to be a single link to some nodes, but actually is a path.

3. Other Background

In this section we'll describe other work being done in routing security, and we'll also introduce hierarchical routing.

3.1 Other Approaches to Hardening Routing in the Literature

There have been many approaches to hardening the routing infrastructure. One approach is to have a shared key for all routers within some scope (such as an area or a domain), and to integrity-protect the routing packets with that key, e.g., [11], with some function such as HMAC [10]. This prevents outsiders from injecting or modifying routing messages, but does not protect against Byzantine failure of the nodes that know the key. Although this simple mechanism helps defend against a lot of practical attacks, the trusted insiders can still generate incorrect routing information, and can certainly fail to forward packets correctly.

A variant of this approach is neighbor-neighbor sharing of keys (pairwise, or link-wise on shared media), so that routing messages are protected as transmitted between neighbors [12]. This also does not protect against Byzantine failure of the routers. Another approach is to use public keys, so that the source of a routing message signs the message and no Byzantine node can modify it without being caught. Although this protects against routers corrupting other routers' routing messages, this does not guarantee that the routing message injected by node S is correct. S might advertise a link to B, and indeed such a link might exist, but S might fail to properly forward some types of traffic over that link.

Another class of papers works on improving computational overhead of digital signature mechanisms by substituting hash functions. [3], [4], [5].

Another approach is to harden the routing protocol against certain types of Byzantine failures of the trusted routers themselves. Recently this approach has been primarily focused on BGP (S-BGP [8], SoBGP [17], SPV [7]), although earlier methods have been simple measures such as the two-way check on links introduced in IS-IS adopted by OSPF (i.e., don't use a link between R1 and R2 in calculating paths unless both R1 and R2 report the link).

However, none of these approaches helps guarantee against Byzantine failure of packet *forwarding*. Although a protocol such as S-BGP can guarantee that a particular path exists, it does not mean that the path will actually work, since a router can perform the routing protocol correctly, but then fail to forward packets. Or it might forward some packets, but not others. A link that forwards for all sources but X might be 99.9% reliable overall, but this does not help X when its traffic is not getting delivered.

In [1], a mechanism is presented so that a source can isolate (to two candidates) which router along a particular non-functioning path is the culprit, but like NPBR, will not scale to a hierarchical network. Also, it assumes that packet loss is an unusual phenomenon. Without careful resource allocation to ensure all flows get a fair share, and backpressure, this is not a reasonable assumption. However, with the design we present in this paper, which partitions the work (so scaling is not an issue), ensures fair resource allocation, and applies backpressure so no packets are lost due to congestion, something similar to the design in [1] could be supplementally used to assist sources in building suspicion about routers in their area so they can look for paths that avoid those routers.

3.2 Hierarchical Routing

There is a limit to how much any routing protocol can scale in a flat space (where all routers have knowledge of the complete topology). In order to build bigger networks, a technique pretty much universally adopted by all network protocols is to impose hierarchy. If addresses are assigned appropriately, the network can be broken into regions we will refer to as *areas*. Routers inside an area know the complete topology of that area, but not details about the internals of any other area.

We will refer to routers that route between areas as *interarea routers*. In its role as an interarea router, router R would not know the inner structure of any of the areas: just the knowledge of the network consisting of interarea routers connected by links (where a "link" might be a virtual link across an area), and the location of the areas. R might also be a router in one or more areas, in which case R would know the structure of the area(s) in which R resides.

Although the technique of hiding the internals of an area from routers external to the area is implemented in all hierarchical routing protocols, there are differing strategies for what a router internal to an area sees of the interarea topology. Which choice is made by various routing protocols is a tradeoff between complexity of protocol, optimality of routes, and routing overhead.

We will argue that providing Byzantine robustness is best provided by certain forms of hierarchy. An example of a deployed model of hierarchy that would not work well for providing Byzantine robustness (as we describe in section 4.5.1) is:

- Global view: R is informed of all the interarea routers, and the links between them. This is similar to the original model in OSPF [12] (see Figure 1). Due to concepts such as stub areas, an OSPF network does not necessarily adhere to this model today. In the global view model, R's extra-area information would require state proportional to the total number of interarea routers and links. ($O(i^2)$), where i is the total number of interarea routers.)

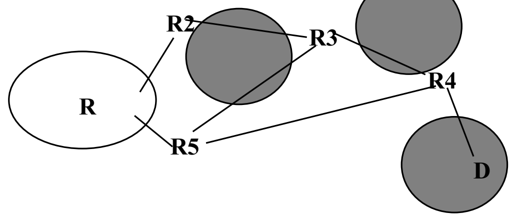

Figure 1. Global View Hierarchy

Two models that are much better suited for providing Byzantine robustness are:

- Local view: R only knows which of the routers in its own area are interarea routers. This is similar to the original model of IS-IS [14]. (see Figure 2). In this

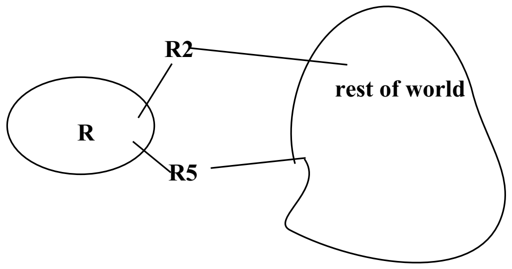

Figure 2. Local View Hierarchy

model, R would have (interarea) state proportional to the number of routers in its own area that are interarea routers (in Figure 2, this is only two pieces of information).

- Interarea cloud view: R knows about all the interarea routers, but sees the interarea network as fully connected; i.e., all interarea paths are seen by R as a single hop from one of the interarea routers in R's own area. (see Figure 3). In this model R has state proportional to

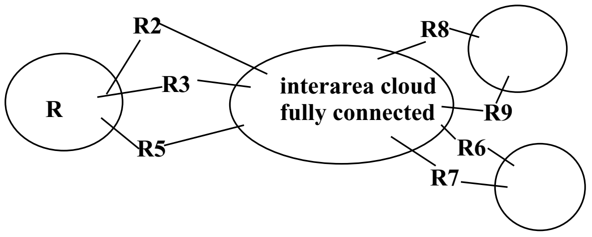

Figure 3. Interarea Cloud View

the number of interarea (border) routers in its own area times the total number of interarea routers. (In Figure 3, this is 3*4, because there are 3 interarea routers in R's area, and 4 interarea routers through which a path constructed by R might exit the interarea cloud into a destination area.)

3.3 Issues with Hierarchical NPBR

In keeping with the spirit of hierarchical routing, we would like to ensure that within an area, whether the area is the "interarea" network, or an area, the state that a router must reserve is at most proportaional to a^2, where a is the number of nodes within that area. A router might exist in multiple areas, or be an interarea router, but we'd like the state to just be additive for all that router's roles.

So, for instance, we cannot require that all the nodes in a hierarchical network know each other's public keys, or have buffers reserved for all the flows in the entire network.

What problems are there with trying to extend NPBR in a straightforward way, to a hierarchy?

3.3.1 Choosing paths

NPBR argues that only source-chosen paths can work because some intermediate router might forward properly for everyone else, but not forward packets from certain sources. Some research has attempted to build reputations by distributed gossiping, but it is exceedingly difficult to do this robustly, especially when a component is working well for everyone except for some source. Is the component that the source is complaining about genuinely bad or is the source evil, attempting to disparage the reputation of an honest node? Should A stop using R if R is working for A, even if B could prove R is discriminating against B?

In a large network, source-specified routes are problematic because there is such a large number of paths (exponential in the number of nodes). Using trial and error, or fault detection protocols such as in [1], can be very expensive if the paths are long.

The problem becomes even more intense, however, with hierarchy. The hierarchy hides the details of the internals of other areas, so the source S can choose a path through its own area to an exit router R1, and then specify a path consisting of interarea routers R2, R3, etc. Since the routing information (by definition of hierarchical routing) does not disclose the inner structure of external areas, it is up to R1 to choose an appropriate path across the next area (area A in Figure 4) to the next hop interarea router R2. The path across A from R1 to R2 might work well most of the time, but some router along the path might specifically corrupt packets from S.

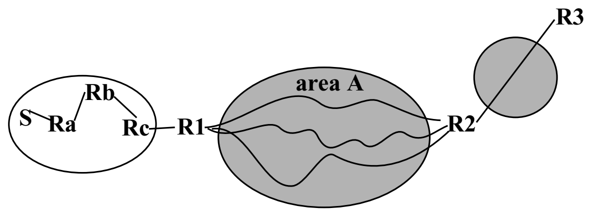

Figure 4. Hidden detail between R1 and R2

So not only does S have to choose an appropriate path, but it would be very difficult for S to cope with a Byzantine failure in the middle of a foreign area. S might choose a path consisting of honest routers, but the path through the area between R1 and R2 might have a Byzantine failure that manifests only on packets from S. So R1, which has chosen the path through its area to R2, might not be aware that there is anything wrong with the path.

Perhaps there might be a mechanism for S to complain to R1, but then R1 does not know whether S might be lying. Should R1 abandon what might be a perfectly functioning path for all other nodes based on a complaint for S? Presumably R1 is limited in the number of paths it can set up through the area. Should R1 be willing to calculate and set up a different path for each source? Or should R1 keep trying paths until it finds one for which no source complains?

3.3.2 State in interarea routers

Another issue with hierarchy is the state in the interarea routers. If each source chooses its own interarea path, then the interarea routers would need to keep state proportional to n^2, where n is the total number of nodes in the entire network, because they would potentially need to keep state (including a buffer) for each (S,D) flow. However, we would like to ensure that interarea routers only need state proportional to the number of links and nodes in the interarea network (or the square of the size of the area). If an interarea router additionally resides within an area, we would like its state to only additively increase with the amount of state necessary to be a resident of that area (a^2 for the number of nodes, a, in that area).

Why do the intermediate routers on a path need to keep state? For instance, instead of setting up a path, the source could include the path in the header of the packet. There are several reasons why creating state for the flow is preferable:

- To assure each flow receives a fair share of bandwidth.
- If the path was in the packet, in order to prevent a Byzantine node from impersonating the source and starving out all resources in the net, the data packets would have to be digitally signed by the source, and verified by the intermediate routers. This would be much more expensive than the NPBR method in which only the path setup packet needs to be signed by the source, and then data packets can be sent without cryptographic protection, or perhaps using secret key encryption hop by hop. Note that secret key encryption in hardware has kept pace with link speeds, but public key encryption has not.

3.3.3 Guaranteeing bandwidth for a flow, over a virtual link

In flat NPBR, when a source sets up a path, resources (at least one buffer) are reserved for that flow, in routers along the path. If a source sends packets too quickly, a later packet from that source might overwrite an earlier packet from that source. However, if the source stops sending new packets, the most recent packet will arrive at the destination.

However, in a hierarchy, we will need to use virtual links, such as the R1-R2 link in Figure 4. The routers inside area A will not know about all the S-D flows that are using that link. Instead, they will see only a single flow (the R1-R2 flow). If we use straightforward NPBR, then since R1 is sending traffic for many different flows over that virtual link, if R1 sends too quickly, traffic from one flow can be overwritten by another flow. We will no longer be able to ensure that the latest packet of any particular flow will ever make it to the destination without being overwritten by another flow on some virtual link.

As we will see later in the paper, we will solve this problem by requiring packet delivery to be reliable across every link, using backpressure to ensure that no flow sends more quickly than the network can accommodate, and to apply this backpressure in a careful way so that one slow flow will not slow down other flows.

4. Hierarchical Robust Routing

In this section we will present our scheme. In section 4.1, we summarize all the steps. In the remaining subsections of section 4, we go into more detail about each step.

4.1 Overview of Our Scheme

In this section we summarize all the steps, but the rationale for these steps and the details of how to implement them are in sections 4.2 through 4.8.

We will use the intra-area NPBR (as designed in 1988) to distribute public keys and link state information within each area (including the "area" consisting of interarea routers), so we can assume that each node within an area securely knows a public key for each other node in that area, and that each node has a link state database detailing the topology of its own area.

Given that all nodes within an area know each other's public keys securely, any pair of nodes within the area can do mutual authentication and agree upon a pair-wise shared key. There are many protocols they could use for this, for instance IPsec's IKE protocol.

To make interarea path creation tractable, routers will create virtual links and take responsibility for a portion of the path, and advertise the endpoint of a path as a neighbor. For example, in Figure 4, R1 would advertise that R2 is its neighbor within the interarea "area". "Taking responsibility" means that R1 must ensure that *every* packet it forwards to R2 over that virtual link is actually received by R2.

To provide reliability on the virtual link, R1 and R2 have to use a protocol in which packets are acknowledged, and retransmitted until R2 has received it properly *and has a buffer to store the packet*.

When a path is created, such as S-R1-R2-D, the highest-level components along the path (i.e., R1 and R2) must maintain a buffer pool for packets for that flow. At least one buffer must be reserved at R1 and R2 for the S-D flow. If R1-R2 is a virtual link, only R1 and R2 need to maintain state for the S-D flow. Components inside the R1-R2 virtual link just think there is a single flow (between R1 and R2). (Note that we will eliminate the necessity for R2 to know about and reserve buffers for the S-D flow later in this paper).

To prevent Byzantine components between R1 and R2 from discriminating against certain forms of traffic, R1 and R2 communicate across the virtual link through a cryptographically protected tunnel, for instance, by using IPsec with both encryption and integrity protection. This is merely applying NPBR's cryptographic defense against a Byzantine link (see section Figure 2.2.1) to a virtual link.

In theory all we'd really need to do is provide integrity protection (i.e., not encryption) between R1 and R2, together with an insistence on 100% reliability (though R1 might retransmit a lost packet a few times before assuming the path is not working). However, encryption foils the ability for a component to discriminate, eliminating all Byzantine behavior except losing packets at random.

To prevent interarea routers from needing to keep state for every (S,D) flow in the entire network, we will use the Interarea Cloud View of hierarchy as shown in Figure 3. This also reduces S's problem (in Figure 4) of route selection to selecting an honest egress router, R1, from its area, and an honest ingress router, R2, into D's area. If R1 and R2 are doing their job, they will ensure that the path they have set up between themselves is completely reliable.

We will also provide backpressure to ensure packets are not lost due to congestion. We must do this in such a way that a slow flow will not slow down other flows. This was not necessary in a flat network, since a source sending too quickly on a flow would only overwrite packets for its own flow. However, if R1 were to send too quickly to R2 over the virtual link, since the intermediate components only keep state for a single R1-R2 flow (and not for all the (S,D) pairs R1 is multiplexing over that virtual link), then packets for one flow might overwrite packets for another flow, causing it to be impossible to guarantee delivery for any flow.

We will require R1 and R2 to do the following in order to maintain a virtual link:

- establish a pair-wise secret key (a security association)
- cryptographically disguise and integrity protect (with the R1-R2 shared key) all traffic on that link

- employ an end-to-end reliable protocol across the virtual link, so that R1 retransmits each packet until R2 acknowledges it. If the virtual link's quality is not sufficiently high (too many packets need to get retransmitted), then R1 must choose a different path to R2.
- have two types of acknowledgments. In one, R2 assures R1 that the packet arrived intact, and that the virtual link is performing correctly. The second type of acknowledgement means that R2 actually has a buffer for a packet for that flow, and is willing to take responsibility for the packet's further progress. With these two forms of ack, a slow flow will not impede the progress of a different flow multiplexed over the same virtual link.

To prevent any node from needing more state than necessary to support its own area, we limit each source to a finite set of simultaneous connections it can *initiate* with nodes outside its area. We cannot place such a requirement on D to restrict the number of nodes that connect *to* D. Therefore we use an asymmetric buffer allocation strategy to avoid requiring D's border router to keep state about an unbounded number of inbound connections to D. Instead, we will arrange that the border router R1 at the connection initiator (S) is the only interarea router required to keep (S,D) state.

The requirement on R1 to keep (S,D) state for outbound connections from its area will not violate our state allocation budget for R1, since we will limit initiators to a constant number of simultaneous outgoing connections. This is explained further in section 4.5.3.

4.2 Establishing a Shared Secret Key with Each Peer

We will need nodes to create virtual links, together with security associations across those virtual links. A virtual link will appear as a single hop when constructing an interarea path. The design using the cloud view of hierarchy only requires building virtual links between peers, where a "peer" of R is a node in the same area as R, or if R is an interarea router, a peer of R is another interarea router. This is fortuitous, because we can assume (by using NPBR) that peers securely know each other's public keys.

A security association between X and Y is established by having X and Y perform mutual authentication (based on public keys that we can assume are securely distributed within each area by the NPBR mechanism), agree upon a shared secret key, and then send all traffic between each other on an encrypted (and integrity protected) tunnel. Once R1 and R2 know each other's public keys reliably, there are many existing protocols for doing so, such as IPsec's IKE protocol.

Secret key integrity checks and encryption are efficient enough to be done at wire speed at reasonable cost.

As we will see in subsequent sections, security associations are created between:

- nodes that think they are neighbors, to prevent any Byzantine behavior of the link between them (including having a node C between A and B trick A and B into thinking they are neighbors, and then having C cause the "link" to behave maliciously)
- each node S inside an area and each border router of that area (to expedite S communicating with nodes outside S's area)
- border routers of the same area, in order to advertise that they are neighbors within the interarea network (e.g., routers R1 and R2 in Figure 4.
- border routers in different areas, in order to facilitate the Interarea Cloud View of Figure 3.

Although this might seem like a lot of security associations, none of these violate our goal of having each router capable of supporting its portion of the hierarchy. We can limit the size of areas to what a router can support. There are deployed protocols that already require a^2 state within an area of size a, such as MPLS nets in which there are standing paths set up between every pair of routers.

4.3 Step 2: Making Links and Virtual Links Reliable

When R1 forwards traffic to R2, whether R1-R2 is a real physical link, or a virtual link consisting of a path through an area, R1 numbers and integrity-protects the packets. R2 acknowledges each packet, but that only means that the link (virtual or physical) has successfully transmitted the packet to R2.

R1 will be multiplexing lots of flows over its link to R2. Even though R2 might successfully receive a packet over this link, it does not mean R2 can accept the packet without overwriting a previous packet for that flow, since R2's allotment of buffers for that flow might currently be full.

In the example in Figure 5, ra and rb are on the path between R1 and R2, which are both border routers in area A. The two internal routers ra and rb have only created state for a single flow between R1 and R2, even though many (S,D) pairs are being multiplexed across that flow. (R1 and R2, however, will keep state about these flows, as we will see in later sections.)

The internal routers ra and rb must create backpressure so that R1 does not swamp the capacity of the R1-R2 virtual link, causing a packet for one (S,D) pair to overwrite a packet for another.

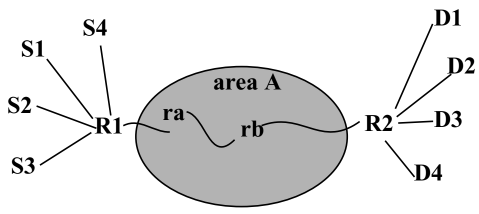

Figure 5. Many flows multiplexed over the R1-R2 virtual link, invisibly to ra and rb

There are two reasons for R2 to tell R1 that it has received a packet.

- So that R1 can monitor the quality of the virtual link and ensure that it is reliable enough. Given the cryptographic protection, the link can at worst lose packets at random. It cannot discriminate among packets. If too many packets require being retransmitted in order to make the R1-R2 virtual link reliable, R1 will establish a new path to R2 within the area. How R1 finds a reliable intraarea path is an orthogonal problem to the rest of the issues in this paper, and for example, it can use the fault diagnosis algorithm in [1] to choose routers it trusts.
- So that R2 can backpressure a particular (S,D) flow, making R1 maintain responsibility for the packet until R2's buffer for that (S,D) flow is free.

So R2 informs R1, in addition to whether R2 received packet #k correctly, whether R2 has a free buffer for the flow to which packet #k belongs, so that R2 can now take over responsibility for this packet. If R2 is backpressuring that flow (by refusing that packet), then R1 round robins through packets for other flows that also need to be forwarded across the virtual link. The design ensures that each flow has its own buffer pool at R1 and R2, even though intermediate components only see a single flow.

4.4 Backpressure

Our scheme does backpressure. Each "hop" in a path (where "hop" consists of the endpoints of virtual links, from the point of view of whoever set up the path) keeps at least one buffer for a flow, and refuses to accept a new packet until the next hop has accepted the previous packet.

With backpressure, it is natural to worry about deadlocks. A lot of work has been done on deadlock-free routing (e.g., [16]). With deadlock-free routing, backpressure can be applied in the presence of congestion rather than dropping packets. With ordinary routing, where there is no attempt to ensure that different flows do not form dependency cycles, the network can become deadlocked unless packets are dropped.

However, because of our scheme's buffer management, there is no need to worry about deadlock-free routes. There is no dependency between flows. Therefore, backpressure can be applied.

It is essential that congestion on one flow not slow down other flows multiplexed over the same virtual link. Therefore, we must have independent reliability for each flow. The definition of a "flow" is subtle, as we will describe, when we are discussing a hierarchy. When a component is an intermediary in a virtual link, say, between R1 and R2, it does not need to be aware of all the flows being multiplexed over that virtual link. It just sees a single flow, the R1-R2 flow.

A blocked flow travelling over the R1-R2 link will not impede progress of another flow using that link, since R2 does not backpressure the flow inside the virtual link. Instead, if R2 receives a packet for a blocked link, it throws it away, informing R1 that that particular packet, although received correctly over the virtual link, had no buffer available. The effect is that R2 backpressures that flow, but not other flows multiplexed over the R1-R2 virtual link.

Although R1 must ensure fair bandwidth allocation of the flows that are using the R1-R2 link, there is no reason why R1 cannot allow flows with more traffic to utilize bandwidth not needed by other flows (either because those flows have no traffic to send, or because they are being slowed down by congested links later on).

Because our scheme is inherently deadlock-free, we can use backpressure all the way to the source, so that the source will know the optimal rate at which it can transmit on each flow.

To summarize, the backpressure applied is at the highest level. Inside a virtual link between R1 and R2, there are resources for the one flow seen by internal components on that virtual link. They will forward packets with whatever resources are available to the R1-R2 flow, but a particular subflow seen by R1 and R2 will be backpressured at the R1-R2 level. Since R2 has independent buffer pools for each of the flows, one subflow will not be blocked because of a different blocked flow. And it is up to R1 to use the bandwidth available on the R1-R2 link fairly between flows it is multiplexing on that link.

4.5 Minimizing State

In this section we will describe the steps necessary to ensure we do not exceed our requirement that no node needs to keep state proportional to the entire size of the network: just state necessary to support the portion of hierarchy to which the node belongs. We will first describe how to save state in interarea routers (4.5.1), then for routers internal to an area (4.5.2). then border routers (4.5.3).

4.5.1 Reducing state in interarea routers

In this section we will explain why the Interarea Cloud View of hierarchy shown in Figure 3 saves state for interarea routers as compared with the Global View of hierarchy.

If we were to use the global view of hierarchy as shown in Figure 6, then S would specify S, ra, rb, rc, R1, R2, R3, R4, D as a path to D.

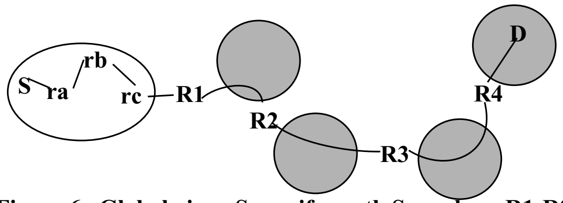

Figure 6. Global view: S specifes path S-ra-rb-rc-R1-R2-R3-R4-D

The routing information available to S consists of the link state information for its area (by which it concludes that a path to R1 would be S, ra, rb, rc, R1), and that the interarea path consists of R1, R2, R3, R4, D. S must trust R1 to maintain a virtual link to R2, R2 to maintain a virtual link to R3, R3 to maintain a virtual link to R4, and R4 to maintain a virtual link to D. Additionally, S must trust all the components S has named in its path (S, ra, rb, rc, R1, R2, R3, R4) to maintain state for the S-D flow (at least one buffer). Otherwise, if some intermediate component, say R2, were not keeping a buffer for the S-D flow, then another flow using the R2-R3 virtual link would either be held up indefinitely if the S-D flow were backpressured downstream, or the S-D packet might be overwritten by a packet from another flow.

So the problem with this style of hierarchical routing (where the source specifies the path of interarea routers) is that every S-D flow in the entire network can choose a different sequence of interarea routers. This places an unreasonable burden on the state in the interarea routers. In the worst case, each interarea router would have to maintain buffers for every (S,D) pair in the entire network.

To reduce the state necessary for intermediate interarea routers, we will instead use the Interarea Cloud view of hierarchy. Each pair of interarea routers will maintain a virtual link to each other. All interarea paths specified by sources will consist of a single hop: the exit router R1 from the source's area and the entrance router R4 into the destination's area, with the link between them being the virtual link R1-R4. If R1 and R4 are non-Byzantine, they will be responsible for finding a non-Byzantine path if they claim they have a virtual link.

By shrinking the interdomain portion of the path to R1-R4, the interarea routers R2 and R3 will only have to keep

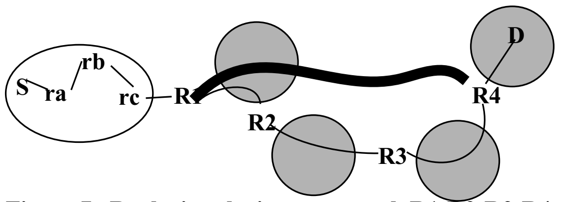

Figure 7. Replacing the interarea path R1-R2-R3-R4 with the single virtual link R1-R4

state about flows originating or destined to their own area, plus a flow consisting of a virtual link between any pair of interarea routers.

The R1-R4 virtual path might indeed consist of R1, R2, R3, R4, and indeed there might even be structure between the R1-R2 link (if the virtual link consists of a path across an area), but the components inside the R1-R2 link only need to keep state for one flow (the R1-R2 virtual link), regardless of the number of (S,D) flows multiplexed over that link. And R2 and R3 only need to keep state about the R1-R4 virtual link that they are part of, even though many flows might be multiplexed over the R1-R4 interarea path.

4.5.2 Reducing state in internal routers

If S's specified path to external destination D indeed consisted of S, ra, rb, rc, R1, R4, D, then ra, rb, and rc would have to keep track of the (S,D) flow. This is not a reasonable assumption because there may be too many external destinations. Internal routers should only need to keep state proportional to a^2, where a is the size of their own area.

We can reduce state in internal routers by having each source (that wants to communicate outside the area) maintain a virtual link to each exit router from the area. There is no reason why S would need to specify a different path to R1 in order to reach external destination D1, than in order to reach external destination D2, especially because once S establishes a virtual link to R1, S encrypts and tunnels all its traffic to R1 so that intermediate components along the S-R1 path cannot practice traffic discrimination. (see Figure 8).

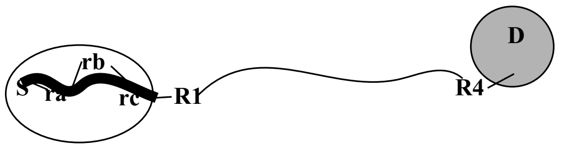

Figure 8. Replacing the intraarea portion, S-ra-rb-rc-R1 of the S-D path with the single virtual link S-R1

In this way, ra, rb, and rc need not keep state for the (S,D) flow, but only between S and R1, which consists of a pair of nodes within their own area, so it is within our state budget.

4.5.3 Reducing state in border routers

We still need to ensure that a border router's state budget is within our allotment. We have said that an interarea path between S and D will consist of three virtual links (S-R1, R1-R4, and R4-D. This would mean that R1 and R4 would need to maintain state about the (S,D) path. If there were no limit to how many D's S might be communicating with, this would exceed our state budget for R1.

It is reasonable for R1 to limit the number of simultaneous interarea connections S can *initiate*. If S needed to "simultaneously" talk to more nodes than the limit, S might be able to negotiate a higher limit with R1, or perform application-layer juggling of the connections so that the number that R1 has to simultaneously keep track of is within the limit. However, it is not reasonable for R1 to impose a limit on the number of connections S might receive from nodes outside the area. If there were such a limit, it would be too easy for there to be a DOS against S.

We will solve this problem by using asymmetric buffering strategies. If S initiates the connection, R1 will keep track of the (S,D) flow. However, since D did not initiate the connection, we will not require R4 to keep state for the (S,D) flow.

Instead we will require R4 to keep track of an (R1, D) flow. It is within R4's budget to keep track of a flow between every interarea router and every node within its area (this is at most i*a, where i is the number of interarea routers, and a is the number of nodes within R4's area).

The way it will work is when S requests setup of the path S-R1-R4-D, R1 checks to see if it already has the path "R1-R4-D" set up. If it does (because some other source within R1's area is already communicating with D through R1 and R4), then R1 merely adds local state for the S-D flow, but does not need to set up anything further. If it does not, then it sets up a path for R1-R4-D, creating state in R4 for an R1-D flow, but not an (S,D) flow. When a packet is sent on the (S,D) flow, the following high level steps occur:

- S sends the packet on the virtual link to R1. When R1 acks both that it was received properly over the virtual link, and that R1 has a buffer available for the (S,D) flow, S can stop retransmitting that packet to R1.
- R1 sends the packet on the virtual link to R4, ensuring that the (S,D) flow receives a fair share of the R1-R4 virtual link. R4 will have a buffer for the R1-D path. If that buffer is available (and the packet was received properly over the R1-R4 virtual link), R4 will ack the packet.
- R4 now sends the packet over its virtual link to D, ensuring that the R1-D packet gets its fair share of the virtual R4-D link, when competing with other (Ri, D) flows.

4.5.4 Return traffic on interarea flows

We have created one-way flows. In our example (see Figure 9), S has initiated a connection to D, and has set up the path S-R1-R2-D. Each of the links in that path is a virtual link.

R1 has state for the S-D flow. However, R2 only has state for an R1-D flow. If D attempted to return traffic to S by sending a packet to R2, R2 would not know that the traffic should go via R1, since R2 does not have any knowledge of S.

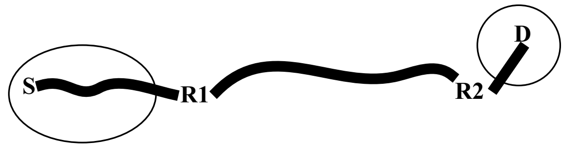

Figure 9. Path from S to D)

To make it work, R2 needs to pass along to D, when it delivers the packet to D, that the packet arrived via R1. The most convenient method of doing this is to add a field inside the encrypted information when R2 tunnels a packet to D on the R2-D virtual link, for passing such hints. D will also need to give a hint to R2 on return traffic to S, that R2 should send this packet to S via R1.

If there were no such field, it is still possible to give this hint using repeated encapsulation. The packet sent by S to D would have:

- an inner packet with source=S, destination=D
- inside a header with source=S, destination=R1

The packet from R1 to R2 would have:

- the inner packet unchanged, i.e., source=S, destination=D
- inside a header with source=R1, destination=D
- inside a header with source=R1, destination=R2

The packet from R2 to D would have

- the inner packet still unchanged, i.e., source=S, destination=D
- inside a header with source=R1, destination=D
- inside a header with source=R2, destination=D

A packet from D to S would have

- an inner packet with source=D, destination=S
- inside a header with source=D, destination=R1
- inside a header with source=D, destination=R2

When R2 receives this packet, it notes that the destination is R1. R2 has resources for a D-R1 flow (though not for a D-S flow). When R2 forwards this packet towards S, the packet would have

- an inner packet with source=D, destination=S
- inside a header with source=R2, destination=R1

4.6 Summarizing State Requirements

4.6.1 Nodes inside areas

A node S inside area A, of size a needs state for

- supporting its own area, which is at most a^2, assuming it is on the path for flows between every pair of nodes in its area
- the number of border routers in A times the number of potential ingress routers. This is probably less than a^2. However, we can avoid this requirement if when S wants to set up a path to D, S queries its chosen border router about potential ingress border routers into D's area, rather than having S at all times know about the n^2 virtual links in the fully connected interarea cloud (where n is the number of interarea border routers).

4.6.2 Interarea routers

Assuming there are n interarea routers, an interarea router that is not a border router needs state at most n^2, assuming it is on the path for virtual links between every pair of border routers.

4.6.3 Border routers

A border router needs state proportional to the number of nodes inside its area times the number of interarea routers, which is no more than the square of the larger of the number of interarea routers, or the number of nodes in its own area, so this requirement is within our state allocation budget.

4.7 Area partitions

Suppose D's area is partitioned, so that not all the interarea routers attached to D's area can reach D. We could employ the solution in [13], in order to repair the area partition, but it would be simpler if the router S chooses on its path to D (R4, in the path S, R1, R4, D) to politely refuse to make the connection by informing S that it cannot reach D. Then S can attempt making a connection using a different interarea router that also claims connectivity to D's area. In short, by allowing the source to specify both the interarea router that exits its area and the interarea that enters the destination area, this, as a side effect, solves the subnetwork partition problem.

If S's own area is partitioned, then S could also communicate with nodes in other partitions of its own area by specifying an egress area border router S can reach, and attempting communication to D through the area border routers S cannot reach within its area.

4.8 Using the "Local View" Hierarchy

In section 4.5.1 we explained why the Global View hierarchy in Figure 1 is not desirable for Byzantine robustness. Otherwise, if each S chose its own interarea path, each interarea router would need (S,D) state. Not only does avoiding the Global View hierarchy keep our state budget for interarea routers within budget, but it makes S's job of choosing a path much more tractable.

So instead we adopted the Interarea Cloud view of Figure 3. However, why can't we use the Local View hierarchy of Figure 2?

With the Cloud view of hierarchy, it is easy for each endpoint of a virtual link to establish a security association, because they are peers. S-R1 are both within the same area. R1-R4 are both interarea routers. R4-D are both in the same area.

If we were to adopt the Local View hierarchy, then the path would consist of two virtual links: S-R1, and R1-D. However, R1 and D are not peers. The NPBR mechanism we rely on does not guarantee that R1 and D will reliably know each other's public keys.

If R1 does not know a key for D, it cannot establish a security association and assure that the R1-D path is reliable.

If we really wanted to implement the Local View hierarchy, we could assume that S knows D's key through some sort of PKI. Since R1 and S share a key (they are peers), S could inform R1 of D's key. S could say to D "please establish a security association, and reliable path to D". Then S would only have to specify an egress router it trusts, and the destination, and it would be R1's responsibility to ensure that R1 chooses an honest ingress router R4 into D's area (and a reliable path of interarea virtual links to R4). R1 would have to receive acks from D, and ensure that all packets arrive safely at D.

4.9 Implementing This Today

We claim the design in this paper is practical to deploy, though figuring out how to phase it into the Internet with existing routers and protocols is a matter for future research. The robust flooding within an area takes a manageable amount of state, and a manageable amount of computation, since a public key signature verification is not that onerous, and nodes can be rate-limited in how often the generate new link state packets or public key announcements.

Once shared keys are established between “neighbors”, where neighbors might be physical neighbors or tunnel endpoints, route setup packets and data packets are cryptographically protected with secret keys.

5. Conclusions

This paper reviews the original work on Byzantinely robust routing, which worked in a flat network, and describes what is necessary in order to make Byzantinely robust routing work within a network large enough to require hierarchy. By “robust routing” we do not just ensure that the routing protocol operates properly, but that packets actually get delivered, even if some of the trusted routers perform the routing protocol correctly, and then handle data packets in a discriminatory manner.

A large part of the solution involves careful resource allocation to ensure that no flow gets starved, and to do this so that no matter how many flows are occurring, no router needs to keep state more than what would be required to support its portion of the hierarchy.

This form of routing is practical, and the protocol, cryptography, and resource allocation strategies in this paper would also solve many existing DOS problems.

6. Acknowledgements

Charlie Kaufman suggested many of the key ideas in this paper, and helped refine others.

7. Bibliography

1. B. Awerbuch, D. Holmer, C Nita-Rotaru, H. Rubens, “An on-demand secure routing protocol resilient to Byzantine Failures”, Proceedings of the ACM workshop on wireless security, 2002.
2. Baker, F., and Atkinson, R., “RIP-2 MD5 Authentication”, RFC 2082, 1997.
3. S. Cheung, “An Efficient Message Authentication Scheme for Link State Routing”, ACSAC 13, 1997.
4. M. Goodrich, “Leap-Frog Packet Packet Linking and Diverse Key Distributions for Improved Integrity in Network Broadcasts”, IEEE Symposium on Security and Privacy, 2005.
5. R.C. Hauser, T. Przygienda, and G. Tsudik, “Lowering Security Overhead in Link State Routing”, Computer Networks, 1999.
6. A. Heffernan, "Protection of BGP Sessions via the TCP MD5 Signature Option," RFC 2385, August 1998
7. Yih-Chun Hu, A. Perrig, M. Sirbu, “SPV: Secure Path Vector Routing for Secure BGP”, ACM SIGCOMM 2004.
8. S. Kent, C. Lynn, and K. Seo, "Secure Border Gateway Protocol (SBGP)," *IEEE Journal on Selected Areas in Communications,* Vol. 18, No. 4, April 2000.
9. Kent, S., Mikkelson, J., and Seo, K, “Secure Border Gateway Protocol (S-BGP) real world performance and deployment issues. NDSS 2000.
10. H. Krawczyk, M. Bellare, R. Canetti, “HMAC: Keyed-Hashing for Message Authentication”, RFC 2104.
11. T. Li, R. Atkinson, “Intermediate System to Intermediate System (IS-IS) Cryptographic Authentication, RFC 2567, July 2003.
12. Moy, J., “OSPF Version 2”, RFC 2328, 1998.
13. Perlman, R., “Hierarchical Routing and the Subnetwork Partition Problem”, Fifteenth Hawaiian International Conference on System Sciences, 1982.
14. Perlman, R., “Interconnections: Bridges, Routers, Switches, and Internetworking Protocols”, Addison Wesley, 1999.
15. Perlman, R., “Network Layer Protocols with Byzantine Robustness”, MIT LCS Tech report, 429, 1988.
16. M. Schroeder, A.Birrel, M. Burrows, H. Murray, R. Needham, T. Rodeheffer, “Autonet: A High-speed, Self-configuring LAN Using Pt-to-pt links”, IEEE Journal on Selected Areas in Communications, 1991.
17. White, R., “Securing BGP: soBGP”, Internet Protocol Journal, Volume 6, Issue 3, September 2003.

About the Author

Radia Perlman is a Distinguished Engineer at Sun Microsystems Laboratories. Her work has had a profound impact on the field of computer networking. She designed many of the algorithms that make link state routing protocols robust, efficient, and manageable; the spanning tree algorithm used by bridges/switches; and security innovations such as assured delete and blind decryption. She is the author of “Interconnection: Bridges, Routers, Switches, and Internetworking Protocols”, and coauthor of “Network Security: Private Communication in a Public World”, both of which are used by many universities as textbooks, and as references by engineers. She holds about 80 issued patents, a PhD in computer science from MIT, and an honorary doctorate from KTH. She was named 2004 Inventor of the Year by SVIPLA (Silicon Valley Intellectual Property Law Association).