Claremont Colleges
Scholarship @ Claremont

CMC Senior Theses | CMC Student Scholarship

2020

Optimizing Router Performance

Bradley Newton
Claremont McKenna College

Radon Rosborough
Harvey Mudd College

Miles President
Harvey Mudd College

Hakan Alpan
Harvey Mudd College

Follow this and additional works at: https://scholarship.claremont.edu/cmc_theses

 Part of the Digital Communications and Networking Commons

Recommended Citation

Newton, Bradley; Rosborough, Radon; President, Miles; and Alpan, Hakan, "Optimizing Router Performance" (2020). *CMC Senior Theses*. 2511.
https://scholarship.claremont.edu/cmc_theses/2511

This Open Access Senior Thesis is brought to you by Scholarship@Claremont. It has been accepted for inclusion in this collection by an authorized administrator. For more information, please contact scholarship@cuc.claremont.edu.

Computer Science Clinic

Final Report for
Juniper Networks

Optimizing Router Performance

April 23, 2020

Team Members

Hakan Alpan
Bradley Newton (Fall Project Manager)
Miles President
Radon Rosborough (Spring Project Manager)

Advisor

Geoff Kuenning

Liaison

Ron Bonica

Abstract

To support its development of networking hardware and software, Juniper Networks conducts research into enhancements to the protocols used on the Internet, in coordination with standards bodies such as the Internet Engineering Task Force. We helped Juniper Networks with two specific research objectives. The first was to design and implement an improved algorithm by which Internet hosts can establish the appropriate packet size to maximize bandwidth while avoiding packet fragmentation. We produced a working implementation of the improved algorithm in the Linux kernel. The second objective was to measure the effect of different Internet Protocol extension headers (specifically, Routing Header Type 0, the Segment Routing Header, and the Compressed Routing Header) on router performance. We produced code for running simple benchmarks locally, as well as a formal Internet Draft specifying the procedure so that it can be run by Juniper Networks on high-performance benchmarking hardware.

Contents

List of Figures

Acknowledgments

The Juniper Networks Clinic Team would like to thank both the Juniper Networks liaison Ron Bonica and our faculty advisor Geoff Kuenning for their help and advice throughout both projects.

Chapter 1

Introduction

In addition to its development of networking hardware and software, Juniper Networks proposes and contributes to the development of new standards for Internet protocols. There are many opportunities for improvements in the Internet Protocol, and since our work on the first such opportunity proceeded ahead of schedule, we subsequently advanced to work on a second opportunity. The next two sections detail each of these opportunities, and what our project contributed.

Chapter 2

Fragmentation project

New work on Internet protocols often occurs in response to limitations in existing protocols that are discovered after they are deployed in production systems. Our project addresses one such problem in the Internet Protocol, a feature called IP fragmentation that is now recognized as undesirable.

2.1 Background

To give some background on the issues addressed by our project, we'll first give an overview of IP fragmentation and why it is inefficient.

2.1.1 Maximum Transmission Unit

Internet paths consist of a source, a destination, and all the routers and links (e.g., wires or wireless connections) connecting them on that path. Because the Internet is composed of many devices that use different hardware, different links may have different constraints on how large a packet they can carry. This limit, a number of bytes, is called the maximum transmission unit (MTU) for that link. The link with the smallest MTU on a path sets the MTU for the entire path (called the path MTU, or PMTU), since any packet traversing the path must travel through that link. The Ethernet standard specifies that any link claiming to support Ethernet must have an MTU of at least 1500 bytes, but other communication protocols may have larger or smaller requirements.

2.1.2 IP fragmentation

When a packet with a size larger than the PMTU reaches a router that has an outgoing link with a smaller MTU, the router will *fragment* the packet. This means it creates a number of smaller packets, each of which has a new header and only a portion of the contents from the original packet. Each of the new packets is small enough that it can be transmitted individually along the link with the small MTU, and its header includes some extra information that allows the destination host to correctly reassemble the fragments. This process is outlined in Figure 2.1.

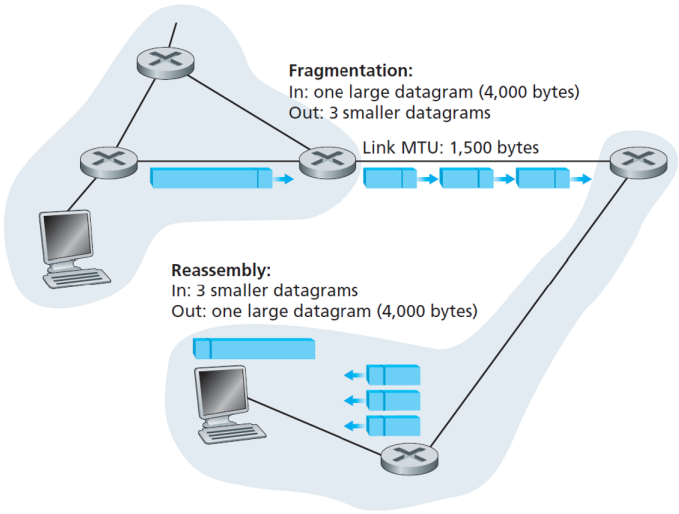

Figure 2.1 Fragmentation and reassembly in IPv4 (Kurose and Ross, 2013).

When the destination host receives all the fragments that made up the original packet, it reassembles them. If one or more of the fragments is missing, the destination host discards them all. This highlights one of the inefficiencies of fragmentation: how long should the destination host wait to see if a missing fragment will arrive? Not waiting long enough means that packets might be retransmitted unnecessarily, while waiting too long means higher latency and wasted memory. And under the Transmission Control Protocol (TCP), the most common protocol used for Internet traffic, a dropped fragment means the entire original packet needs to be retransmitted, leading to a more congested network.

In addition to incurring additional CPU time, memory overhead, and network congestion, the fragmentation protocol has security vulnerabilities

and does not interact well with more advanced networking features. These issues are outlined in detail in a current Internet Draft (Bonica et al., 2019).

2.1.3 Path MTU Discovery

Because of the performance problems and security risks we outlined above, it is desirable to avoid IP fragmentation whenever possible. This is precisely the goal of Path MTU Discovery (PMTUD). The PMTUD protocol allows a sender to determine the PMTU on the path to the destination before it starts transmitting packets. PMTUD works by sending an IP packet with the Don't Fragment (DF) bit set, which disallows fragmentation. If this packet reaches a link that is too small, the router sees that the DF bit is set and drops the packet, sending back an error message using the Internet Control Message Protocol (ICMP). This ICMP message contains the MTU of the link for which the packet was too large. When the sender receives this packet, it can adjust its estimate of the PMTU accordingly, and try again with another packet of this new size. This process is repeated until the PMTU estimate is small enough for a packet to successfully traverse the entire path to the destination without fragmentation.

2.2 Problem

As we saw in Section 2.1.2, IP fragmentation is generally undesirable. In version 6 of the Internet Protocol (IPv6), fragmentation by intermediate routers is no longer allowed. Even in IPv4, it is possible to disallow fragmentation by setting the DF bit on a packet. However, IPv4 is still widely deployed and some legacy technology and corner cases mean it is not always practical to disallow fragmentation entirely. Thus another solution is needed. Although PMTUD helps to deal with the fragmentation problem by trying to avoid sending packets that are too large, fragmentation can still happen. There are at least three reasons why PMTUD is not an adequate solution to the IP fragmentation problem:

1. The first issue is that PMTUD is slow. Although it avoids fragmentation, it increases latency on the delivery of some packets because messages must be sent back and forth synchronously before the packet exceeding the PMTU can be delivered successfully. With fragmentation, on the other hand, all packets that are sent can be delivered without a round trip, even if their processing is slower. PMTUD trades one performance problem for one that is potentially worse.

2. In general, Internet traffic between two hosts does not follow the same path for the entire duration of the connection. For example, the network topology may change as hosts are added or removed, or intermediate load balancers may redirect traffic depending on congestion. This means that PMTUD needs to be performed periodically throughout the connection; otherwise, the PMTU may change over time and lead to fragmentation.

3. Most problematically, PMTUD suffers from an issue known as blackholing. This occurs when an intermediate router fails to pass along the ICMP messages meant to alert the sender that fragmentation is required. Routers may do this for security reasons (since emitting less diagnostic information makes an attack on the network more difficult) or because they are misconfigured. In either case, if the ICMP message is never delivered, then the source will never be informed that the packet it sent was too large. Since PMTUD sets the DF bit in order to generate this ICMP message, the packet will be dropped. Under TCP, in which every data packet sent triggers an acknowledgement (ACK) packet in response, the source will notice from the absence of an ACK that its packet was not delivered. However, it will not know to split the data into smaller packets, so it will fall into an infinite loop sending the same packet over and over again. This failure condition is much worse than the degraded performance of fragmentation.

2.3 Our approach

Our project mitigates the problem of IP fragmentation by introducing a new *Packet Reassembled* ICMP message that the destination host can use to tell the source that fragmentation occurred and it should send smaller packets in the future. Specifically, we updated the Linux kernel IP stack in these two ways:

1. When the IP layer reassembles a packet out of several fragments (meaning that fragmentation occurred at some intermediate host during routing), it will send an ICMP message back to the source indicating that the packet was fragmented.

2. When the IP layer receives this ICMP message, it will update its estimate of the PMTU so that it sends smaller packets that will not lead to future fragmentation.

This procedure has several key advantages over PMTUD. The first is that it has a lower latency than PMTUD because packets never need to be retransmitted synchronously. In essence, the PMTU is "discovered" asynchronously, in parallel with continuous data transmission. If the network topology changes over the life of a connection and the PMTU decreases, the sending host will be alerted on the first instance of fragmentation and can reduce its PMTU estimate accordingly, but this discovery does not interrupt data transmission.

A second advantage is how our solution handles *increases* in the PMTU. Notice that if the PMTU increases, then this does not produce an ICMP message, so the source does not notice. The solution to this problem is to periodically send a larger packet to probe for an increased PMTU. Under PMTUD, deciding how often to probe is a difficult tradeoff, because each probe will result in a synchronous hiccup in data transmission. But with our solution, even very frequent probes will not interrupt the data stream.

The most important advantage of our approach, however, is that it mitigates the blackholing problem because fragmentation is still enabled. If an intermediate router fails to forward the ICMP Packet Reassembled message, packets will be fragmented as normal and will still be delivered successfully, albeit somewhat less efficiently.

2.4 Development environment

Building the Linux kernel from source is a complex process. As such, one of the deliverables of our project is a fast and maintainable way to test kernel changes that pertain to the networking stack. We used a standard piece of software called Vagrant (HashiCorp, n.d.) to provision and manage a fleet of virtual machines on a virtual network. Because of this, our first step was generating a configuration for the Linux kernel that would allow it to boot in this setup, and modifying this configuration to optimize build time. We abstracted these and other tasks related to compilation, installation, and network management into a set of scripts that can be used by any developer to quickly reproduce our setup. This work can be reused in future projects requiring kernel development, saving setup time.

2.5 Implementation

The code for the project consists of three parts: the definition of the new ICMP message, the behavior for when a host receives a fragmented packet,

and the behavior for when a host receives our ICMP message (after having sent a packet that was fragmented). We did not need to implement periodic discovery of PMTU increases because this code already exists in the Linux kernel and our implementation makes use of it implicitly.

Our code is available in a public GitHub repository at https://github.com/raxod502/juniper-tools. Our fork of the Linux kernel is at https://github.com/raxod502/juniper-linux.

2.5.1 New ICMP message

When a host receives and reassembles a fragmented packet, it delivers our Packet Reassembled ICMP message back to the sender. This message includes the size of the largest fragment that was received, which serves as an estimate of the PMTU from the sender to the receiver. As with all ICMP messages, it also contains the initial contents of the packet that caused the ICMP message (i.e., the reassembled packet) as well as the length of that packet measured in 32-bit words. The layout of the message is shown in Figure 2.2.

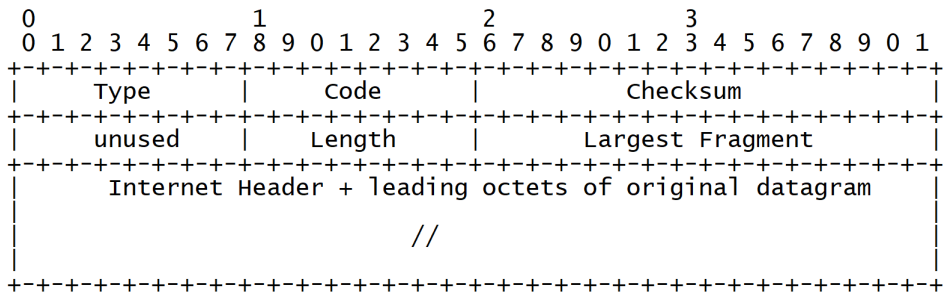

Figure 2.2 Format for the ICMP Packet Reassembled message.

Initially, our ICMP message used its `code` field to indicate whether the fragment reassembly was successful or encountered an error. However, upon further reflection and discussion with our liaison, we decided that no ICMP message should be sent when reassembly fails because in that case, we can't guarantee that the largest fragment received is representative of the MTU. Therefore, in our final implementation, the `code` field is always zeroed out.

2.5.2 Response to a fragmented packet

The code to respond to a fragmented packet and send the ICMP message is mostly uninteresting except for the fact that our Packet Reassembled message is not sent in response to fragmented ICMP messages. If it was, and the new message traversed the same link that caused the fragmentation, then it too would be fragmented, which would cause the original sender to send a Packet Reassembled message back in response. This would lead to a continuous cycle of fragmented Packet Reassembled messages between the two hosts.

2.5.3 Response to ICMP Packet Reassembled

The code responding to our ICMP message closely follows the structure of the existing kernel code. There is already an ICMP message called Fragmentation Needed that is sent when a packet is dropped because it required fragmentation, but its Don't Fragment option was enabled. Because this message is an error rather than a diagnostic, and is only sent when a packet is dropped by an intermediate router rather than when a packet is successfully reassembled at the destination, it differs from our ICMP Packet Reassembled message, and does not present a solution to the problem our project solves. However, the behavior when an ICMP Fragmentation Needed message is *received* is virtually identical to the behavior when an ICMP Packet Reassembled message is received: upon receiving the message, we call the handler corresponding to the protocol of the packet that was fragmented, and carry out the PMTU estimate update within that handler using the MTU information included in the ICMP message.

2.6 Testing

We have two different virtual network setups for testing our implementation. The first includes only a source host and a destination host on the same network. The second includes the source and destination hosts on different networks along with two intermediate routers on the path between them. This additional configuration allowed us to ensure that our implementation worked properly even when neither the source nor the destination was attached to the link constraining the MTU.

Throughout testing, we used the packet sniffing tool Wireshark to examine the structure of the packets we sent. Our testing procedure was as follows:

1. Disable PMTUD and check that there is no data in the PMTU cache on the source host, and if otherwise, clear it.

2. Establish a TCP connection between the source and destination hosts.

3. Reduce the MTU of the intermediate link (between the source and destination hosts in the first configuration and between the intermediate routers in the second configuration). Note that this step is done after establishing the TCP connection since TCP estimates the PMTU during its initial handshake. This MTU reduction simulates a change in the network topology during the lifetime of the connection.

4. Send a chunk of TCP data with a size exceeding the MTU of the intermediate link from the source to the destination. Observe in Wireshark that this packet is fragmented and that our ICMP message is sent in response. Also observe that there is now data in the source host's PMTU cache.

5. Send another chunk of TCP data of the same size. Observe in Wireshark that TCP breaks the data up into segments of sizes less than the MTU before it can reach the IP layer, thereby preventing fragmentation.

2.7 Future work

Because our project implements a new Internet standard, its next phase is adoption within the Internet community and in the Linux kernel. We worked with Juniper Networks to promote ICMP Packet Reassembled as a standard, which involved conversing with influential Internet Area Working Group members about the merit of ICMP Packet Reassembled as a good solution to the IPv4 fragmentation problem and encouraging Linux kernel developers to incorporate our change into newer releases of the kernel. We expect that the relative simplicity of our code changes will be a key argument in favor of their adoption, given the relative importance of the problem that they solve.

Chapter 3

Benchmarking project

In our second project, we helped Juniper Networks measure the impact of certain extensions to the Internet Protocol version 6 (IPv6) that implement a feature called source routing. We produced code that can be used to run basic tests locally, together with a formal procedure in Internet Draft format specifying the procedure so that Juniper Networks can collect accurate data using dedicated benchmarking hardware. With this data, Juniper Networks will be able to ascertain whether replacing the existing Segment Routing Header with their proposed Compressed Routing Header will improve router performance.

3.1 Background

Under IPv6, each packet has a standard set of headers. In addition to these, a packet may optionally include *extension headers*, which provide features that are not needed by every packet. One application of extension headers is source routing, which is a procedure by which the sender of a packet can specify information about how it should be routed. Source routing can be used to optimize the performance of a network; this application is called *traffic engineering*. Source routing is especially useful in combination with *software-defined networking*, a technology allowing some parts of network behavior to be controlled by a centralized source that can determine the flow of packets through the network in a holistic manner that optimizes overall performance.

The original version of IPv6 included support for source routing by means of the Routing Header Type 0 (RH0). The RH0 extension header, shown in Figure 3.1, includes a list of IPv6 addresses that a packet must be

routed through before reaching its destination. When the RH0 header is included, the packet is routed (using standard protocols) to the first listed address, whereupon the header is adjusted to indicate that it should be routed to the next address, and so on. Unfortunately, RH0 could be exploited to perform an effective Denial of Service (DoS) attack, as documented in RFC 4942 (Davies et al., 2007), so it was deprecated in 2007 by RFC 5095 (Abley et al., 2007), and support for it has since been removed from almost all IPv6 hosts on the Internet.

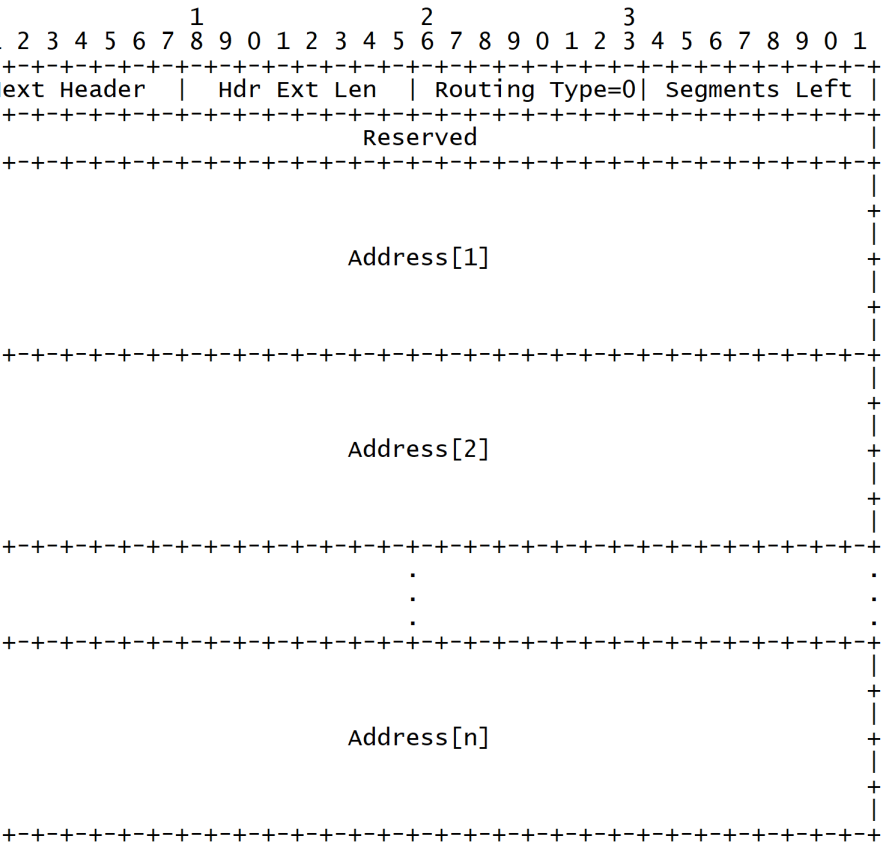

Figure 3.1 Format of Routing Header Type 0 as specified in RFC 2460 (Deering and Hinden, 1998).

However, source routing is still a desirable use case for IPv6, so a new extension header, called the Segment Routing Header (SRH), was proposed to replace RH0. This header was specified by RFC 8754 (Filsfils et al., 2020) and is shown in Figure 3.2. Although SRH has many more features than

RH0, its core functionality is similar; that is to say, it is possible to specify a list of IPv6 addresses that a packet must visit before it is delivered to its destination.

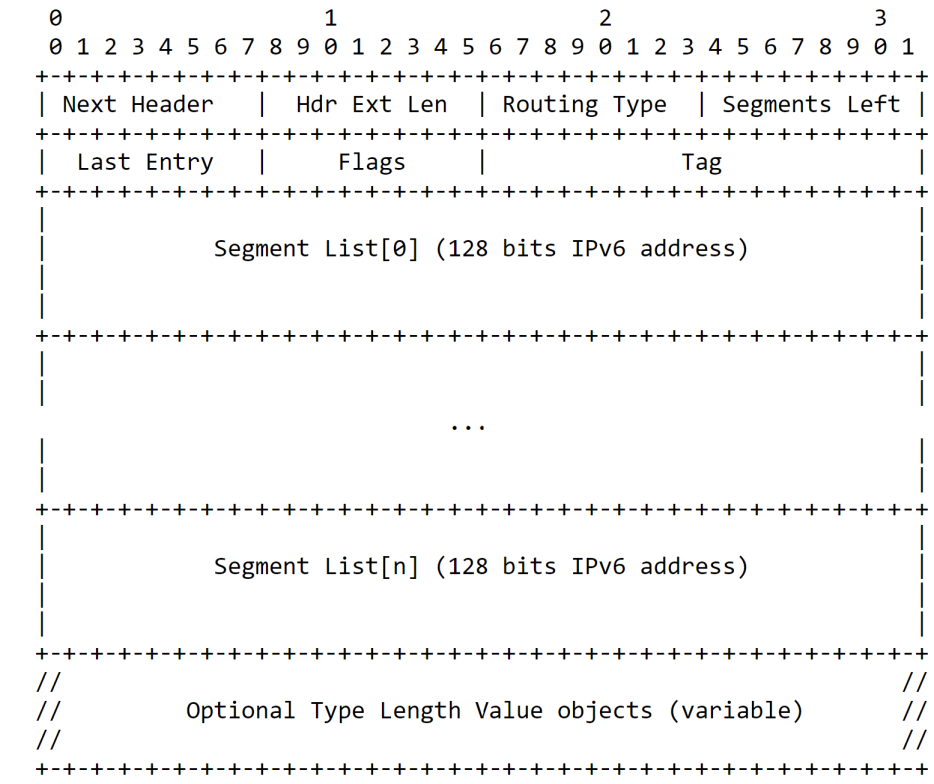

Figure 3.2 Format of Segment Routing Header as specified in RFC 8754.

One problem with SRH (and RH0) is that headers may grow to be very large. This is because IPv6 addresses are 128 bits long, and many of them may be included in an SRH (or RH0) header for practical implementations of source routing. To address this concern, Juniper Networks has proposed an alternative to SRH called the Compressed Routing Header (CRH). This header is specified by the CRH Internet Draft (Bonica et al., 2020) and is shown in Figures 3.3 and 3.4. The CRH does not specify intermediate hosts with 128-bit IPv6 addresses, instead using 16-bit or 32-bit identifiers that can be translated by routers into full IPv6 addresses. It is the job of the intermediate routers to maintain lookup tables for the CRH identifiers, which in many cases means there must be some infrastructure which is capable of informing the intermediate routers of how to populate their

lookup tables. (That infrastructure is beyond the scope of our project.)

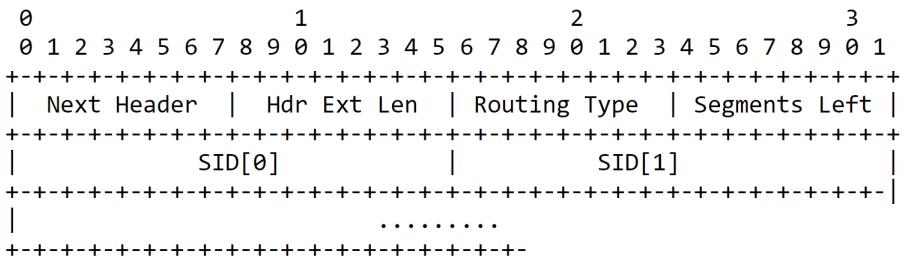

Figure 3.3 Format of Compressed Routing Header with 16-bit identifiers ((Bonica et al., 2020)). For CRH-16, the Routing Type is 5 (pending IANA assignment).

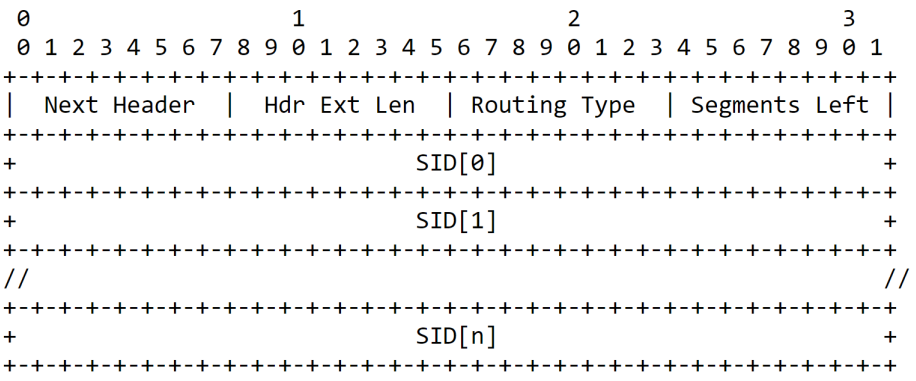

Figure 3.4 Format of Compressed Routing Header with 32-bit identifiers ((Bonica et al., 2020)). For CRH-32, the Routing Type is 6 (pending IANA assignment).

3.2 Problem

The differences in how source routing extension headers identify intermediate nodes cause variations in how quickly a packet is processed on its way to its destination. Since routers have to copy packets into memory before processing them, the larger RH0 and SRH packets incur extra overhead that increases as the number of intermediate nodes increases. The lookup table that CRH employs is also expensive, but it is a fixed cost that is independent of the number of intermediate nodes.

The time it takes for a packet to travel from source to destination is often a point of interest for optimization and comparison with different

implementations. As such, Juniper Networks has asked us to construct a framework for an experiment that examines this tradeoff by exploring packet throughput relative to the type of routing header and the number of intermediate nodes.

3.3 Our approach

Juniper Networks wanted us to design a procedure to test the performance of their routers. When designing and testing our experiment, we wanted to replicate such a setup, that is, a tester machine sending packets to a router and measuring the throughput. However, for a number of reasons, we were unable to use real routers or real machines in our setup.

We needed to test three different routing headers in different stages of development. As mentioned previously, RH0 was deprecated in 2007, so finding a router supporting RH0 was challenging. Also, CRH is currently being developed by Juniper Networks and has not yet reached standardization, so there is no available router that can support the processing of CRH. Juniper Networks supplied us with a Linux kernel image that implements CRH processing, so we used a virtual machine (VM) running that version of Linux as a router. Since we had to a use a VM for the router processing CRH, we used VMs for the other headers as well. To process RH0, we found an Ubuntu 6 image from 2006, when RH0 was still supported, and used another VM running that version of Linux. We did not have to use a separate VM for the SRH header since the Linux image that Juniper Networks provided supported that.

We encountered other problems when testing router throughput. One problem in testing the throughput of a router is that we need a machine that sends packets to a router at a very fast and consistent rate. Regular machines cannot do that, since the operating system decides exactly when to send a packet, which causes significant variance. Juniper Networks has Ixia machines they can use for that purpose, but those are too expensive to use in our Clinic project. Since we realized we could not reach the accuracy we desired without proper hardware, we opted to use a virtual machine for the tester device as well.

To get results from our tests, we needed to send packets fast enough to overrun the router VMs under test. However, this proved to be challenging because the sender VM simply could not send packets fast enoguh. We initially thought that we would be able to overcome this issue by reducing the processor speed of the router VMs we were using, but we were unable to

do so, so we used the traffic-control features of the Linux kernel to impose a manual limit on the VMs' interface bandwidths. Clearly this means we are not measuring the true performance of routing header processing, but that was not our goal in local testing. The interface-bandwidth constraint works well enough to validate our benchmarking setup.

Most routers would likely behave similarly to the interface-constrained router VMs we set up, because the limiting factor in practice is likely to be interface bandwidth rather than actual processing time. However, Juniper Networks also produces heavier-duty routers which can process a very large interface bandwidth, and our test procedure would likely produce more interesting data for those routers, since the extra CPU time incurred by CRH lookups would become non-negligible in comparison to the interface bandwidth constraint.

Overall, our setup consists of the tester VM sending packets to the router VMs, which process those packets and send them back. Although we knew that we would not be able to get accurate data with this setup, we were still able to design a procedure that would yield useful data in a real setup. Since we want to compare the efficiency of the routing headers, the procedure we came up with runs throughput tests in accordance with RFC 2544 (Bradner and McQuaid, 1999) for each of the headers, with the number of addresses in the header ranging from one to fifteen. In an actual setup, we expect CRH to be less efficient at first, and more efficient later, as the number of addresses included in the header increases. We used Python scripts to run these tests in our local setup and to plot the graphs comparing the headers.

After running tests, we wrote an Internet Draft, included in Appendix B, outlining our procedure that tests routers on their performance of processing different routing headers. This Internet Draft could be used by Juniper Networks, or anyone wanting to benchmark their routers, as it is available to the public.

3.4 Testing

Our implementation for local testing is "quick and dirty," since we do not expect to obtain meaningful data from this experiment. The code is available in a public GitHub repository at https://github.com/raxod502/juniper-tools.

We run the tester machine and the routers as VMs using VirtualBox, and provision them with the appropriate kernel and software using Vagrant and associated scripts. To measure throughput, we use a variation of binary search to implement the specification for throughput measurement outlined

in RFC 2544. Briefly, RFC 2544 states that to measure the throughput of a router, one should send a sequence of packets with a given time interval between each packet, and ascertain the minimum interval (and thus maximum throughput) that the router can tolerate without dropping any packets. Since we wanted to perform a large number of tests automatically, we needed a fast way to determine the minimum interval. Our algorithm is a variation of binary search that increases the inter-packet interval when the router drops packets and decreases it when no packets are lost. As the algorithm proceeds, it makes smaller and smaller changes to the interval until it can set a bound on the difference between the current and minimum possible intervals. Packets sent with this interval are then used to measure the throughput of the router, and the results of five tests are averaged together. A sample run of this algorithm is graphed in Figure 3.5.

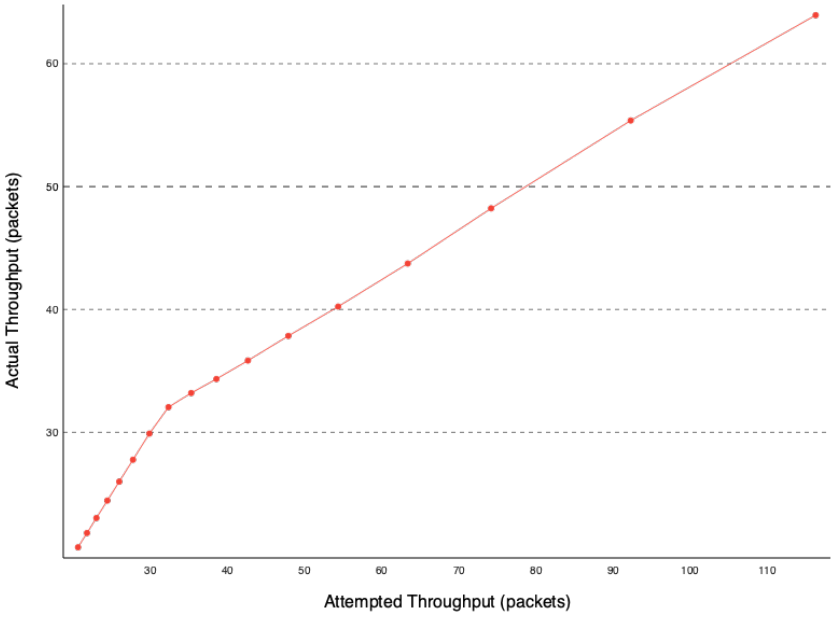

Figure 3.5 Actual throughput resulting from attempted throughput for rh0.

For each of the three types of extension headers (RH0, SRH, and CRH), we measured throughput as a function of the number of IPv6 addresses (or, for CRH, short identifiers) included in the header, from one to fifteen. Because of the way in which we forced our router VMs to drop packets, the resulting bandwidth (measured in packets per second) is inversely propor-

tional to packet size. As expected, we observed that CRH packets achieve a higher throughput than SRH and RH0 packets, due to the use of shorter identifiers rather than full 128-bit IPv6 addresses. SRH has lower throughput than RH0 because the header is larger by a fixed size, even though addresses are the same length. Furthermore, including more intermediate hosts in the header results in a lower throughput, due to the increased header size. The results of these tests are shown in Figures 3.6 and 3.7.

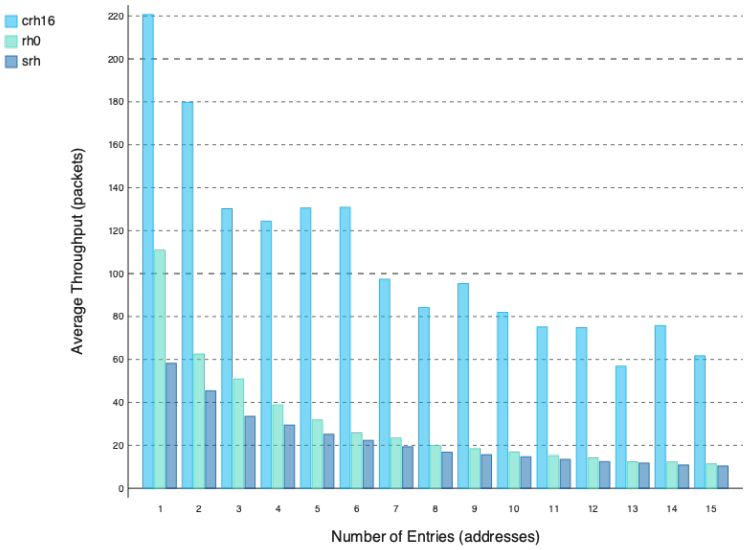

Figure 3.6 Average throughput for different header types.

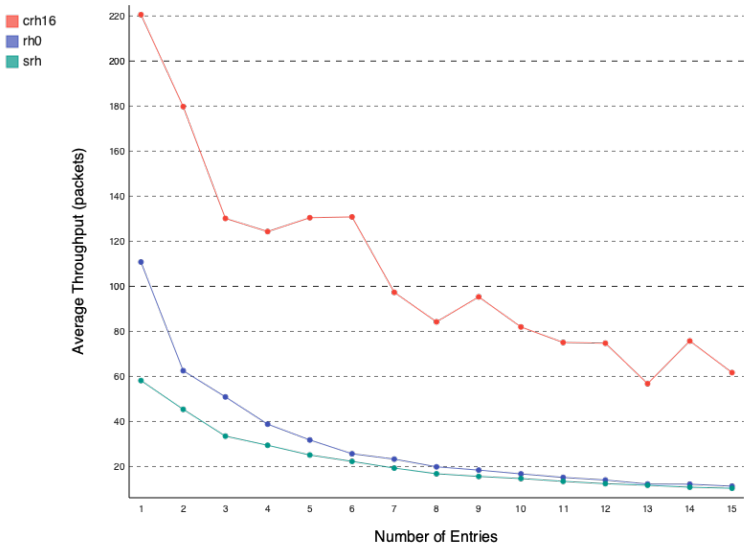

Figure 3.7 Average throughput for different numbers of entries.

3.5 Future Work

The next step for this project will be the actual data collection carried out by Juniper Networks using our experimental procedure and their specialized hardware. This data will allow Juniper Networks to evaluate the performance of their Compressed Routing Header in comparison with existing types of routing headers and contribute toward its adoption as an Internet standard.

Appendix A

Linux kernel patch

Here we include the contents of our Linux kernel patch.

```
diff --git a/include/uapi/linux/icmp.h b/include/uapi/linux/icmp.h
index 5589eeb791ca..ddaef4521e02 100644
--- a/include/uapi/linux/icmp.h
+++ b/include/uapi/linux/icmp.h
@@ -33,7 +33,8 @@
 #define ICMP_INFO_REPLY        16      /* Information Reply            */
 #define ICMP_ADDRESS           17      /* Address Mask Request         */
 #define ICMP_ADDRESSREPLY      18      /* Address Mask Reply           */
-#define NR_ICMP_TYPES          18
+#define ICMP_PKT_REASM         253     /* Report Packet Reassembly     */
+#define NR_ICMP_TYPES          253


 /* Codes for UNREACH. */
@@ -80,6 +81,11 @@ struct icmphdr {
                __be16  __unused;
                __be16  mtu;
        } frag;
+       struct {
+               __u8    __unused;
+               __u8    orig_dg_len;
+               __be16  mtu;
+       } reasm;
        __u8    reserved[4];
    } un;
 };
diff --git a/net/ipv4/icmp.c b/net/ipv4/icmp.c
index 92b3d2d1139e..76df9aafe32b 100644
--- a/net/ipv4/icmp.c
+++ b/net/ipv4/icmp.c
@@ -853,6 +853,9 @@ static bool icmp_unreach(struct sk_buff *skb)
                if (icmph->code == ICMP_EXC_FRAGTIME)
                        goto out;
                break;
+       case ICMP_PKT_REASM:
+               info = ntohs(icmph->un.reasm.mtu);
+               break;
        }

        /*
@@ -1099,7 +1102,8 @@ int icmp_err(struct sk_buff *skb, u32 info)
                return 0;
        }

-       if (type == ICMP_DEST_UNREACH && code == ICMP_FRAG_NEEDED)
+       if ((type == ICMP_DEST_UNREACH && code == ICMP_FRAG_NEEDED) ||
+           type == ICMP_PKT_REASM)
                ipv4_update_pmtu(skb, net, info, 0, IPPROTO_ICMP);
        else if (type == ICMP_REDIRECT)
```

```
                ipv4_redirect(skb, net, 0, IPPROTO_ICMP);
@@ -1179,6 +1183,9 @@ static const struct icmp_control icmp_pointers[NR_ICMP_TYPES + 1] = {
        [ICMP_ADDRESSREPLY] = {
                .handler = icmp_discard,
        },
+       [ICMP_PKT_REASM] = {
+               .handler = icmp_unreach,
+       },
 };

 static void __net_exit icmp_sk_exit(struct net *net)
diff --git a/net/ipv4/ip_fragment.c b/net/ipv4/ip_fragment.c
index cf2b0a6a3337..e5a50d8874d1 100644
--- a/net/ipv4/ip_fragment.c
+++ b/net/ipv4/ip_fragment.c
@@ -404,6 +404,8 @@ static int ip_frag_reasm(struct ipq *qp, struct sk_buff *skb,
        void *reasm_data;
        int len, err;
        u8 ecn;
+       u_int16_t orig_dg_len;
+       u_int32_t icmp_info;

        ipq_kill(qp);

@@ -449,6 +451,18 @@ static int ip_frag_reasm(struct ipq *qp, struct sk_buff *skb,

        ip_send_check(iph);

+       /* Original datagram length in 32-bit words,
+        * up to 576 - 8 = 568 bytes (568 / 4 = 142 32-bit words) */
+       orig_dg_len = len > 568 ? 142 : (len - 1) / 4 + 1;
+       icmp_info = (orig_dg_len << 16) + IPCB(skb)->frag_max_size;
+
+       /* skb has no dst, perform route lookup again */
+       err = ip_route_input_noref(skb, iph->daddr, iph->saddr,
+                                  iph->tos, skb->dev);
+
+       if (iph->protocol != IPPROTO_ICMP)
+               icmp_send(skb, ICMP_PKT_REASM, 0, htonl(icmp_info));
+
        __IP_INC_STATS(net, IPSTATS_MIB_REASMOKS);
        qp->q.rb_fragments = RB_ROOT;
        qp->q.fragments_tail = NULL;
diff --git a/net/ipv4/ping.c b/net/ipv4/ping.c
index 9d24ef5c5d8f..67d6d83bb33f 100644
--- a/net/ipv4/ping.c
+++ b/net/ipv4/ping.c
@@ -536,7 +536,9 @@ void ping_err(struct sk_buff *skb, int offset, u32 info)
                        harderr = 1;
                        break;
                case ICMP_DEST_UNREACH:
-                       if (code == ICMP_FRAG_NEEDED) { /* Path MTU discovery */
+               case ICMP_PKT_REASM:
+                       /* Path MTU discovery */
+                       if (type == ICMP_PKT_REASM || code == ICMP_FRAG_NEEDED) {
                                ipv4_sk_update_pmtu(skb, sk, info);
                                if (inet_sock->pmtudisc != IP_PMTUDISC_DONT) {
                                        err = EMSGSIZE;
diff --git a/net/ipv4/raw.c b/net/ipv4/raw.c
index 40a6abbc9cf6..a6b7a2c2f4a2 100644
--- a/net/ipv4/raw.c
+++ b/net/ipv4/raw.c
@@ -230,7 +230,8 @@ static void raw_err(struct sock *sk, struct sk_buff *skb, u32 info)
        int err = 0;
        int harderr = 0;

-       if (type == ICMP_DEST_UNREACH && code == ICMP_FRAG_NEEDED)
+       if ((type == ICMP_DEST_UNREACH && code == ICMP_FRAG_NEEDED) ||
+           type == ICMP_PKT_REASM)
                ipv4_sk_update_pmtu(skb, sk, info);
        else if (type == ICMP_REDIRECT) {
                ipv4_sk_redirect(skb, sk);
diff --git a/net/ipv4/tcp_ipv4.c b/net/ipv4/tcp_ipv4.c
index cfa81190a1b1..b1f9718d1095 100644
--- a/net/ipv4/tcp_ipv4.c
+++ b/net/ipv4/tcp_ipv4.c
```

```
@@ -498,10 +498,12 @@ int tcp_v4_err(struct sk_buff *icmp_skb, u32 info)
                err = EPROTO;
                break;
        case ICMP_DEST_UNREACH:
-               if (code > NR_ICMP_UNREACH)
+       case ICMP_PKT_REASM:
+               if (type == ICMP_DEST_UNREACH && code > NR_ICMP_UNREACH)
                        goto out;

-               if (code == ICMP_FRAG_NEEDED) { /* PMTU discovery (RFC1191) */
+               /* PMTU discovery (RFC1191) */
+               if (type == ICMP_PKT_REASM || code == ICMP_FRAG_NEEDED) {
                        /* We are not interested in TCP_LISTEN and open_requests
                         * (SYN-ACKs send out by Linux are always <576bytes so
                         * they should go through unfragmented).
diff --git a/net/ipv4/udp.c b/net/ipv4/udp.c
index 665f26e32d77..bdc2bfa68df4 100644
--- a/net/ipv4/udp.c
+++ b/net/ipv4/udp.c
@@ -709,6 +709,9 @@ int __udp4_lib_err(struct sk_buff *skb, u32 info, struct udp_table *udptable)
        case ICMP_REDIRECT:
                ipv4_sk_redirect(skb, sk);
                goto out;
+       case ICMP_PKT_REASM:
+               ipv4_sk_update_pmtu(skb, sk, info);
+               goto out;
        }

        /*
```

Appendix B

Benchmarking Methodology for IPv6 Routing Extension Headers

On the following pages, we include the contents of our Internet Draft.

Independent
Internet-Draft
Intended status: Informational
Expires: October 21, 2020

Hakan Alpan
Bradley Newton
Miles President
Radon Rosborough
Harvey Mudd College

May 2020

Benchmarking Methodology for IPv6 Routing Extension Headers
draft-clinic-ipv6-ext-hdr-bench-method-00

Abstract

This document specifies a test procedure which should be used to evaluate the performance characteristics of a network interconnection device which processes IPv6 routing extension headers. The results of the test procedure can be used to compare the performance of the Compressed Routing Header (CRH) with the performance of other routing extension headers and with the performance of packets which do not include routing extension headers. The routing extension headers which may be compared with the CRH using the test procedure are the Segment Routing Header (SRH) and Routing Header Type 0 (RH0).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

IPv6 [RFC8200] source nodes use routing extension headers to specify the path that packets follow to reach their destination. The first routing extension header to be defined was Routing Header Type 0 (RH0) [RFC2460]. This header was deprecated [RFC5095] and removed from current IPv6 implementations because it introduced security vulnerabilities.

Two replacements to RH0 have been proposed, the Segment Routing Header (SRH) [RFC8754] and the Compressed Routing Header (CRH) [I-D.draft-bonica-6man-comp-rtg-hdr]. Both of these routing extension headers provide a superset of the functionality that was previously provided by RH0, and both address the security vulnerabilities of RH0.

Both RH0 and the SRH specify intermediate nodes in the routing extension header as a list of 128-bit IPv6 addresses. The disadvantage of this is that routing headers may become very large, which may impose data transmission overhead and degrade router performance (see section 1 of [I-D.draft-bonica-6man-comp-rtg-hdr]). For this reason, in the CRH, intermediate nodes are specified using 16-bit or 32-bit short identifiers which are mapped to IPv6 addresses by intermediate routers.

For a given router, it is possible that either the SRH or the CRH would result in better performance. Processing a packet which uses the SRH requires the router to copy a larger header; however, processing a packet which uses the CRH requires the router to perform a lookup to translate the short identifier into an IPv6 address.

This document defines a procedure that can be used to compare the performance of the CRH against other routing extension headers, namely: the SRH, RH0, and packets without routing extension headers.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Test Procedure

The performance characteristics of routing extension headers on a given device under test (DUT) SHOULD be measured following the guidelines in [RFC2544], except as specified in the following sections. The set of tests that is run SHOULD include a throughput test, and MAY also include other tests that are specified in [RFC2544].

3.1. DUT Setup

The DUT to be tested MUST be able to process each of the routing extension headers whose performance will be compared. To get the most useful results, both the CRH and the SRH SHOULD be included. If possible, both 16-bit and 32-bit versions of the CRH SHOULD be included. RH0 and packets without a routing extension header MAY be included as well for comparison.

The CRH has limited support in current IPv6 implementations, so the requirement to support the CRH is likely to be the most difficult to fulfill. Juniper Networks has produced implementations of the CRH in the Linux kernel and in the MX-series router (see section 11 of [I-D.draft-bonica-6man-comp-rtg-hdr]). However, these implementations currently support only the 16-bit version of the CRH.

If the CRH is included in tests, then the router MUST have at least one SID configured to map to the tester's IP address. This SID MUST be used in the CRH to cause the router to forward the packet back to the tester (or receiver, if separate transmitting and receiving devices are used).

As per [RFC2544], configuration changes MUST NOT be made to the router between different tests.

3.2. Independent Variables

The performance characteristics of routing extension header processing may be affected by several factors, which SHOULD be used as independent variables in the test procedure:

- The type of routing extension header in use (the CRH, the SRH, RH0, or none).
- For the CRH, whether 16-bit or 32-bit short identifiers are used.
- For the CRH, the SRH, and RH0, the number of addresses (or, for the CRH, short identifiers) specified in the header. This variable SHOULD range at least from 1 to 15, but MAY include higher values if desired.
- The number of data bytes included in the packets that are sent. This variable SHOULD take on the same set of values for each permutation of the other independent variables. See the discussion of frame sizes below.

Each test SHOULD be run for every possible combination of the independent variables.

3.3. Header Contents

No extension headers should be used except for the routing extension headers being tested. Only one extension header at a time should be used.

The next segment in the SRH and RH0 MUST be the IP address of the tester (or, when using separate transmitting and receiving devices, the receiver). The next segment in the CRH MUST be an SID that the DUT has been configured to map to the IP address of the tester (or receiver). This configuration MUST be done before starting any tests.

Apart from the next segment for the SRH and RH0, the IP addresses used in the CRH, the SRH, and RH0 should be selected randomly as outlined in appendix C of [RFC2544] from the ranges reserved for this purpose by IANA.

3.4. Frame Sizes

The performance characteristics of routing extension headers may vary depending on frame size. Section 9 of [RFC2544] provides guidelines for selecting frame sizes. However, different routing extension headers use different amounts of space to encode the same information. In particular, the CRH uses less space to encode information about intermediate nodes than the SRH and RH0. For this reason, a fair comparison between two routing extension headers uses the same payload size for each rather than the same frame size for each.

The set of payload sizes for the tests SHOULD be chosen so that the resulting set of frame sizes for each routing extension header and each number of addresses follows the guidelines set out in [RFC2544] as closely as possible.

4. IANA Considerations

No IANA actions required.

5. Security Considerations

No security considerations.

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <https://www.rfc-editor.org/info/rfc2460>.

[RFC2544] Bradner, S. and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices", RFC 2544, DOI 10.17487/RFC2544, March 1999, <https://www.rfc-editor.org/info/rfc2544>.

[RFC5095] Abley, J., Savola, P., and G. Neville-Neil, "Deprecation of Type 0 Routing Headers in IPv6", RFC 5095, DOI 10.17487/RFC5095, December 2007, <https://www.rfc-editor.org/info/rfc5095>.

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

[RFC8754] Filsfils, C., Ed., Dukes, D., Ed., Previdi, S., Leddy, J., Matsushima, S., and D. Voyer, "IPv6 Segment Routing Header (SRH)", RFC 8754, DOI 10.17487/RFC8754, March 2020, <https://www.rfc-editor.org/info/rfc8754>.

6.2. Informative References

[I-D.draft-bonica-6man-comp-rtg-hdr] Bonica, R., Kamite, Y., Niwa, T., Alston, A., and L. Jalil, "The IPv6 Compressed Routing Header (CRH)", draft-bonica-6man-comp-rtg-hdr-14 (work in progress), April 2020.

Acknowledgements

The authors would like to thank Ron Bonica and Geoff Kuenning for their comments and suggestions that improved this document.

Authors' Addresses

Hakan Alpan
Harvey Mudd College

EMail: halpan@hmc.edu

Bradley Newton
Harvey Mudd College

EMail: bnewton@hmc.edu

Miles President
Harvey Mudd College

EMail: mpresident@hmc.edu

Radon Rosborough
Harvey Mudd College

EMail: rrosborough@hmc.edu

Bibliography

Abley, J., P. Savola, and G. Neville-Neil. 2007. Deprecation of type 0 routing headers in IPv6. RFC 5095, RFC Editor.

Bonica, Ron, Fred Baker, Geoff Huston, Robert Hinden, Ole Troan, and Fernando Gont. 2019. IP fragmentation considered fragile. Internet Draft draft-ietf-intarea-frag-fragile-16, IETF Secretariat. URL http://www.ietf.org/internet-drafts/draft-ietf-intarea-frag-fragile-16.txt.

Bonica, Ron, Yuji Kamite, Tomonobu Niwa, Andrew Alston, and Luay Jalil. 2020. The IPv6 compressed routing header (CRH). Internet-Draft draft-bonica-6man-comp-rtg-hdr-13, IETF Secretariat.

Bradner, Scott, and Jim McQuaid. 1999. Benchmarking methodology for network interconnect devices. RFC 2544, RFC Editor. URL http://www.rfc-editor.org/rfc/rfc2544.txt.

Davies, E., S. Krishnan, and P. Savola. 2007. IPv6 transition/co-existence security considerations. RFC 4942, RFC Editor.

Deering, Stephen E., and Robert M. Hinden. 1998. Internet protocol, version 6 (IPv6) specification. RFC 2460, RFC Editor. URL http://www.rfc-editor.org/rfc/rfc2460.txt.

Filsfils, C., D. Dukes, S. Previdi, J. Leddy, S. Matsushima, and D. Voyer. 2020. IPv6 segment routing header (SRH). RFC 8754, RFC Editor.

HashiCorp. n.d. Vagrant. URL https://www.vagrantup.com. Accessed: 2020-04-10.

Kurose, James F., and Keith W. Ross. 2013. *Computer Networking: A Top-Down Approach*. Pearson Education, 6th ed.