PIM
Internet-Draft
Intended status: Experimental
Expires: 7 January 2026

T. Eckert, Ed.
Futurewei Technologies USA
M. Menth
S. Lindner
University of Tuebingen
Y. Liu
China Mobile

6 July 2025

Stateless Multicast Replication with Segment Routed Recursive Tree Structures (RTS)
draft-eckert-pim-rts-forwarding-03

Abstract

BIER provides stateless multicast in BIER domains using bitstrings to indicate receivers. BIER-TE extends BIER with tree engineering capabilities. Both suffer from scalability problems in large networks as bitstrings are of limited size so the BIER domains need to be subdivided using set identifiers so that possibly many packets need to be sent to reach all receivers of a multicast group within a subdomain.

This problem can be mitigated by encoding explicit multicast trees in packet headers with bitstrings that have only node-local significance. A drawback of this method is that any hop on the path needs to be encoded so that long paths consume lots of header space.

This document presents the idea of Segment Routed Recursive Tree Structures (RTS), a unifying approach in which a packet header representing a multicast distribution tree is constructed from a tree structure of vertices ("so called Recursive Units") that support replication to their next-hop neighbors either via local bitstrings or via sequence of next-hop neighbor identifiers called SIDs.

RTS, like RBS is intended to expand the applicability of deployment for stateless multicast replication beyond what BIER and BIER-TE support and expect: larger networks, less operational complexity, and utilization of more modern forwarding planes as those expected to be possible when BIER was designed (ca. 2010).

This document only specifies the forwarding plane but discusses possible architectural options, which are primarily determined through the future definition/mapping to encapsulation headers and controller-plane functions.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Please see {#version01} for a summary of changes over the prior version of this document.

This draft expands on prior experimental work called "Recursive BitString Structure" (RBS) for stateless multicast replication with source routed data structures in the header of multicast data packets. Its changes and enhancements over RBS are a result from further scalability analysis and further matching against different use cases. Its proposed design also includes Proof of Concept work on high-speed, limited functionality programmable forwarding plane engines in the P4 programming language.

RTS, like RBS is intended to expand the applicability of deployment for stateless multicast replication beyond what BIER and BIER-TE support and expect: larger networks, less operational setup complexity, and utilization of more flexible programmable forwarding planes as those expected to be possible when BIER was designed (ca. 2010).

Unlike RBS, RTS does not limit itself to a design that is only based on the use of local bitstrings but instead offers both bitstring and SID based addressing inside the recursive tree structure to support to allow more scalability for a wider range of use cases.

2. Overview

2.1. From BIER to RTS

2.1.1. Example topology and tree

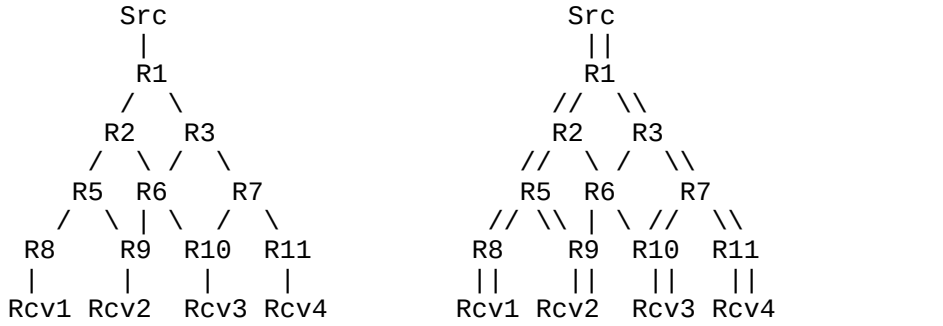


Figure 1: Example topology and tree

The following explanations use above example topology in Figure 1 on the left, and example tree on the right.

2.1.2. IP Multicast

Assume a multicast packet is originated by Src and needs to be replicated and forwarded to be received by Rcv1...Rcv4. In IP Multicast with PIM multicast routing, router R1...R11 will have so-called PIM multicast tree state, especially the intermediate routers R2...R7. Whenever an IP Multicast router has multiple upstream routers to choose from, then the path election is based on routing RPF, so the routing protocol on R9 would need to route Src via R5, and R10 would need to route Src via R7 to arrive at the tree shown in

the example.

2.1.3. BIER

In stateless multicast forwarding with Bit Index Explicit Replication (BIER), [RFC8279], a packet has a header with a bitstring, and each bit in the bitstring indicates one receiver side BIER router (BFER).

[R8:5 R9:9 R10:11 R11:17] =

00001000001000001000000000000000000000000

Figure 2: Example BIER bitstring

In Figure 2, the term [Ri:bi...] (i=5,9,10,11; bi=5,9,11,17) indicates the routers "Ri" that have their associated bit in the bitstring number "bi" set. In this example, the bitstring is assumed to be 42 bit long. The actual length of bitstring supported depends on the header, such as [RFC8296] and implementation. The assignment of routers to bits in this example is random.

With BIER, there is no tree state in R2...R7, but the packet is forwarded from R2 across these routers based on those "destination" bits bi and information of the hop-by-hop IP routing protocol, e.g.: IS-IS or OSPF. The intervening routers traversed therefore also solely depend on that routing protocols routing table, and as in IP multicast, there is no guarantee that the shown intermediate hops in the example picture are chosen if, as shown there are multiple equal cost paths (e.g.: src via R10->R6->R3 and R10->R7->R3).

The header and hence bitstring size is a limiting factor for BIER and any source-routing. When the network becomes larger, not all receiver side routers or all links in the topology can be expressed by this number of bits. A network with 10,000 receivers for example would require at least 40 different bitstrings of 256 bits to represent all receiver routers with separate bits. In addition, the packet header needs to indicate which of those 40 bitstrings is contained in the packet header.

When then receiver routers in close proximity in the topology are assigned to different bitstrings, then the path to these receivers will need to carry multiple copies of the same packet payload, because each copy is required to carry a different bitstring. In the worst case, even as few as 40 receivers may require still 40 separate copies, as if unicast was used - because each of the 40 bits is represented in a different bitstring.

2.1.4. BIER-TE

In BIER with Tree Engineering (BIER-TE), [RFC9262], the bits in the bitstring do not only indicate the receiver side routers, but also the intermediate links in the topology, hence allowing to explicitly "engineer" the tree, for purposes such as load-splitting or bandwidth guarantees on the tree.

[R1R2:4 R2R5:10 R5R8:15 R5R9:16 R1R3:25 R3R7:32 R7R10:39 R7R11:42]

000100000100001100000000100000010000001001

Figure 3: Example BIER-TE bitstring

In Figure 3, the list of [RxRy:bi...] indicates the set of bits needed to describe the tree in Figure 1, using the same notation as in Figure 2.

Each RxRy indicates one bit in the bitstring for the link Rx->Ry. The need to express every link in a topology as a separate bit makes scaling even more challenging and requiring more bitstrings to represent a network than BIER does, but in result of this representation, BIER-TE allows to explicitly steer copies along the engineered path, something required for services that provide traffic engineering, or when non-equal-cost load splitting is required (without strict guarantees).

2.1.5. RTS

With Recursive Tree Structure (RTS) encoding, the concept of steered forwarding from BIER-TE is modified to actually encode the tree structure in the header as opposed to just one single "flat" bitstring out of a large number of such bitstrings (in a large network). For the same tree as above, the structure in the header will logically look as follows.

```
   Syntax:
     RU  = SID { :[  NHi+ ] }
     NHi = SID
     SID = Ri

   Example tree with SID list on R1:
     R1 :[ R2 :[ R5 :[ R8   ,R9   ]], R3 :[R7 :[R10,  R11]]]

   Semantic:
     R1 replicates to neighbors R2, R3.
     R2 replicates to R5
     R3 replicates to R7
     ...

   Encoding structure:
     1 byte SID always followed by
     1 byte length of recursive structure length (":[" in example)
       If no recursive structure follows, length is 0.

   Example SID list serialization (decimal):

     R1 :[ R2 :[ R5 :[ R8   ,R9   ]], R3 :[ R7 :[R10,  R11 ]]]
      |  |  |  |  |  |  | |   | |      | |   | |   | |   | |
      v  v  v  v  v  v  v v   v v      v v   v v   v v   v v

       ..........SIDs according to above example..........
      |     |     |     |     |        |     |     |     |
     01 16 02 06 05 04 08 00 09 00    03 06 07 04 10 00 11 00
         |     |     |    |     |        |     |     |     |
         ......................Length fields................

   Tree with SID list on R2:
     R2 :[ R5 :[ R8   ,R9   ]]
```

Figure 4: Example RTS structure with SIDs

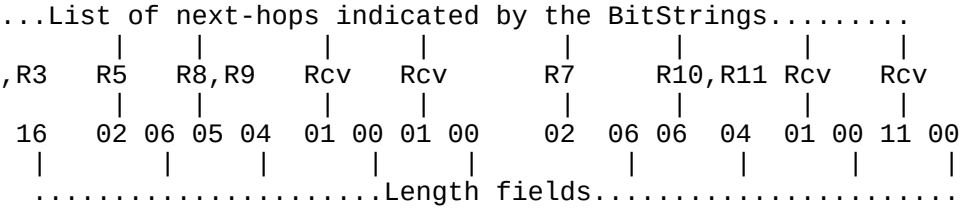
In the example the simplified RTS tree representation in Figure 4, Rx:[NH1,... NHn] indicates that Rx needs to replicate the packet to NH1, NH2 up to NHn. This [NH1,... NHn] list is called the SID-list. Each NH can again be a "recursive" structure Rx:[NH1’,...NHn’], such as R5, or a leaf, such as R8, R9, Ro10, R11.

A simplified RTS serialization of this structure for the packet header is also shown: Each router Ri is represented by am 8-bit SID i. The length of the following SID list, :[NHi,...NHn], is also encoded in one byte. If no SID list follows, it is 00.

When a packet copy is made for a next-hop, only the relevant part of the structure is kept in the header as shown for R2.

Example tree with bitstrings on R1:
BS1 :[BS2 :[BS5 :[BS8, BS9]], BS3 :[BS7 :[BS10, BS11]]]

Example bitstring serialization (decimal):



Example tree with bitstrings on R2:
BS2 :[BS5 :[BS8, BS9]]

Figure 5: Example RTS structure with bitstrings

Instead of enumerating for each router the list of next-hop neighbors by their number (SID), RTS can also use a bitstring on each router, resulting in a potentially more compact encoding. Scalability comparison of the two encoding options is discussed later in the document. Unlike BIER/BIER-TE bitstrings, each of these bitstring will be small, as it only needs to indicate the direct neighbors of the router for which the bitstring is intended.

In Figure 5, the example tree is shown with this bitstring encoding, also simplified over the actual RTS encoding. BSi indicates the bitstring for Ri as an 8-bit bitstring. On R8, R9, R10, R11 this bitstring has bit 1 set, which is indicating that these routers should receive ("Rcv") and decapsulate the packet.

2.1.6. Summary and Benefits of RTS

In BIER for large networks, even small number of receivers may not fit into a single packet header, such as aforementioned when having 10,000 receiver routers with a bitstring size of 256. BIER always requires to process the whole bitstring, bit-by-bit, so longer bitstrings may cause issues in the ability of routers to process them, even if the actual length of the bitstring would fit into processable packet header memory in the router.

In BIER-TE, these problems are even more pronounced because the bitstrings now need to also carry bits for the intermediate node hops, which are necessary whenever the path for a packet need to be

explicitly predetermined such as in traffic engineering and global network capacity optimization through non-equal cost load-balancing, which in unicast is also a prime reason for deployment of Segment Routing.

These scalability problems in BIER and BIER-TE can be reduced by intelligent allocation of bits to bitstrings, but this requires global coordination, and for best results good predictions of the most important required future multicast trees.

In RTS, no such network wide intelligent assignment of addresses is required, and any combination of receiver routers can be put into a single packet header as long as the maximum size of the header is not exceeded (including of course the intermediate nodes along the path).

Unlike Bier/BIER-TE, the RTS header can likely on many platforms be larger than a BIER/BIER-TE bitstring, because the router never needs to examine every bit in the header, but only the (local) bitstring or list of SIDs for this router itself and then for each copy to a neighbor, it only needs to copy the recursive structure for that neighbor. The only significant limit for RTS in processing is hence the maximum amount of bytes in a header that can be addressed.

3. Architecture

This version of the document does not specify an architecture for RTS.

The forwarding described in this document can allow different architectures, also depending on the encapsulation chosen. The following high-level architectural considerations and possible goals/ benefits apply:

(A) If embedding RTS in an IP or IPv6 source-routing extension header, RTS can provide source-routing to eliminate stateful (IP) Multicast hop-by-hop tree building protocols such as PIM. This can be specifically attractive in use cases that previously used end-to-end IP Multicast without a more complex P/PE architecture, such as enterprises, industrial and other non-SP networks.

(B) The encoding of the RTS multicast tree in the packet header makes it natural to think about RTS providing a multicast "Segment Routing" architecture style service with stateless replication segments: Each recursive structure is an RTS segment.

This too can be a very attractive type of architecture to support, especially for networks that already use MPLS or IPv6 Segment Routing for unicast. Nevertheless, RTS can also be beneficial in SP networks

not using unicast Segment Routing, and there are no dependencies for networks running RTS to also support unicast SR, other than sharing architecture concepts.

(C) RTS naturally aligns with many goals and benefits of BIER and even more so BIER-TE, which it could most easily supersede for better scalability and ease of operations.

In one possible option, the RTS header specified in this document could even replace the bitstring of the BIER [RFC8296] header, keeping all other aspects of BIER/BIER-TE reusable. In such an option, the architectural aspects of RTS would be derived and simplified from [RFC9262], similar to details described in [I-D.eckert-bier-cgm2-rbs-01].

4. Specification

4.1. RTS Encapsulation

Figure 6: RTS encapsulation

This document specifies the formatting and functionality of the "Recursive Tree Structure" (RTS) Header, which is assumed to be located in a packet between some Encap Header and some Next Proto / Payload.

The RTS header contains only elements to support replication to next-hops, not any element for forwarding to next-hop. This is left as a task for the Encap Header so that RTS can most easily be combined with potentially multiple alternative Encapsulation Header(s) for different type of network protocols or deployment use cases. Common Encap Headers will also require an Encap Header specific description of the total length of the RTS Header.

In a minimum (theoretical) example, RTS could be used on top of Ethernet with an ethertype of RTS+Payload, which indicates not only that an RTS Header follows, but also the type of the Next Proto Payload.

See the encap discussions in Section 5.2 for considerations regarding BIER or IPv6 extension headers as Encap Headers.

4.2. RTS Addressing

Addresses of next-hops to which RTS can replicate are called RTS Segment IDentifiers (SIDs). This is re-using the terminology established by [RFC8402] to be agnostic of the addressing of the routing underlay used for forwarding to next-hops and obtaining routing information for those routing underlay addresses. Specifying an encapsulation for RTS requires specifying how to map RTS SIDs to addresses of the addresses used by that (unicast) forwarding mechanism.

RTS SIDs are more accurately called RTS replication SIDs. They are assigned to RTS nodes. When a packet is directed to a particular RTS SID of an RTS node it means that that node needs then to process the RTS Header and perform replication according to it.

Using the SR terminology does not mean that RTS is constrained to be used with forwarding planes for which (unicast) SR mappings exist: IPv6 and MPLS, but it means that for other forwarding planes, mappings need to be defined. For example, when using RTS with [RFC8296] encapsulation, and hence BIER addressing, which is relying on 16-bit BFR-id addressing (especially the BFIR-id in the [RFC8296] header), then RTS SIDs need to map to these BFR-ids.

If instead RTS is to be deployed with (only) an IPv6 extension header as the Encap Header, then RTS SIDs need to be mapped to IPv6 SIDs.

This document uses three types of RTS SIDs to support three type of encoding of next-hops in an RTS Header: Global, Local and Local bitstring RTS SIDs.

All SIDs map to a unicast address or unicast SID of the node which the RTS SID addresses. This unicast address or SID is used in an Encap Header when sending an RTS packet to that node.

SIDs can be local or global in scope. All nodes only have one RTS SID table, and it is purely a matter or the semantic assigned to a SID whether it is local or global (as it is in SR).

There are two encoding lengths for SIDs, 10 and 18 bit. It may hence be beneficial to fit all local SID into the lower 10 bit of the SID address table so they can use the so-called short SID encoding (10 bit).

The network is expected to make SID information available to the creators of RTS headers so they can create one or more RTS headers to achieve the desired replication tree(s) for a payload. This includes:

* global SIDs and the nodes they map to.

* Semantic of local SIDs on each node to optimize RTS headers by use of local SIDs

* Capabilities of each node to understand which subset of the RTS syntax can be encoded in the so-called "Recursive Unit" for this node.

* For each node its "all-leaf-neighbors" list of global SIDs (see {#all-leaf-neighbors})

4.3. RTS Header

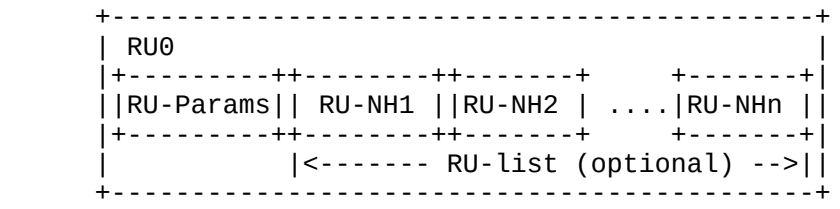


Figure 7: RTS Header

The RTS Header consists of a recursive structure of "Recursive Units" abbreviated "RU"s. The whole header itself is called RU0. Every RU is composed of RU-Params and an optional list of RU for next-hops called the RU-list.

4.3.1. RU-Params

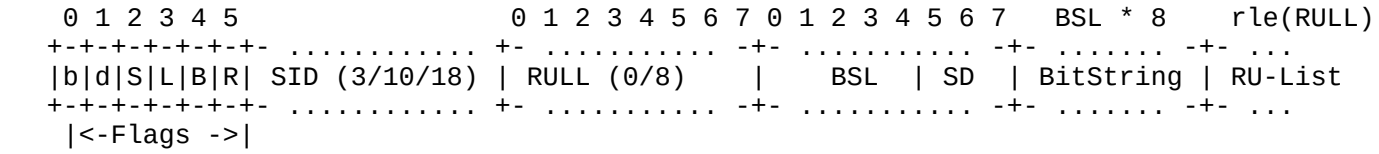


Figure 8: RU-Format

Flags:

b)roadcast: The node should broadcast copies of this packet to all direct leaf neighbors if set to 1, else not

d)eliver: The node should receive a copy of the packet itself if set to 1, else not.

S)ID: A SID is present if set to 1, else not. If no SID is present, the flags are followed by pad bits so that the following field will start at a byte boundary.

L)ength: The SID is "Long" (18 bit), else short (10 bit).

B)itString: If set to 1, a BitString is present as well as the BLE/SD fields, else all three fields are not present.

R)U-List: An RU-List and its length field, the RULL are present. If 0, both are not present.

SIDs:

As described by the S)ID and L)ength flags, when present, the SID can be 10 or 18 bit long. These are called "short" and "long" SIDs. They both address the same single SID table in the node. Short SIDs are just an encoding optimization. Any SID may be "local" or "global". A SID is "global" when it points to the same node consistently. It is "local" when each node has it’s own, potentially different meaning for it. Typically, SIDs fitting into a short SID will preferably be local SIDs such as those pointing to direct (subnet) neighbors.

BitString / BSL / SD:

The BitString Length field (BSL) indicates the length of the BitString field in bytes. It permits length of 0-32 bytes. SD is a SubDomain, and allows up to 8 different BitStrings for each length.

RU-List / RULL:

RU-List Length (RULL) is the length of the RU-List field. To allow RU-Lists longer than 256 bytes, the encoding of RULL uses a simple encoding: RULL values <= 127 indicate the length of RU-List in bytes, values >= 128 indicate a length of (127 + (RULL - 127) * 4). This allows RU-List length of up to 640 bytes at the expense of up to 3 padding bytes at the end of an RU-List required when encoding RU-Lists with length > 127 bytes.

Note: RULL is placed before BSL/SD and BitString so that the offset of RULL inside of RU is fixed and does not depend on the length of BitString (if present). This is beneficial because both SID and RULL need to be parsed by prior-hop nodes when this RU is in an RU-list. See the following explanations for more details.

4.3.2. RU-Params Parsing and Semantics

An RU representing some specific nodeRU is parsed up to two times. The first time, the RU may be present in the RU-List of the prior-hop node to nodeRU, and the second time it is parsed by nodeRU itself.

4.3.2.1. Bitstring replication to leaf nodes

Figure 9 shows the most simple example with bitstrings, the RU for a node that should use a local BitString to replicate to only lead nodes

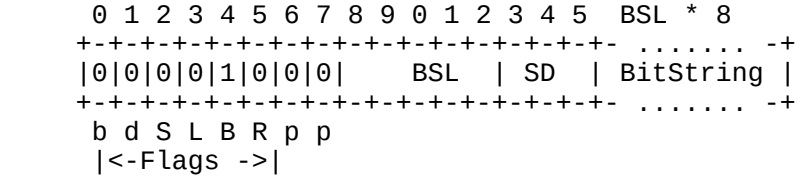


Figure 9: BitString replication to leaf nodes

We assume the node is just transit node, so the packet is neither to be broadcast to all leaf neighbors nor received, hence b=0, d=0.

The RU has no SID, so S=0, L=0, and hence the flags are followed by padding bits set to 0.

B=1 indicates a bitstring follows and R=0 indicates that the RU has no RU-list, and hence no RULL. If the node’s packet parser requires any non-required field to be present, then those would have to be included and set to 0.

Note that this RU can only be used if this is the the root of the tree or the prior hop node was also replicating to this node via a BitString. If the prior hop node was replicating via a SID-List, then this RU would include a SID, but that SID would be ignored, because it was only included for the benefit of the prior hop node.

BSL and SD indicate the BitString and its length.

When the node replicates to the bits in the BitString, it operates pretty much like BIER-TE, aka: each bit indicates an adjacency, most likely direct, subnet-local neighbors. For every bit, a copy is being made.

For the copy to a leaf neighbor, the RTS packet is rewritten to a simple form shown in Figure 10.

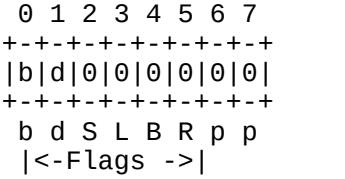


Figure 10: RU for leaf

The whole RTS header consists of a single by RU indicating simply whether the receiving node should b)roadcast and/or d)eliver the packet. Both bits are derived from the BIFT (Bit Index Forwarding Table) of the replicating node.

4.3.2.2. SID-list replication to leaf nodes

Replication to a list of SID on a node requires an RU on the node as shown in Figure 11.

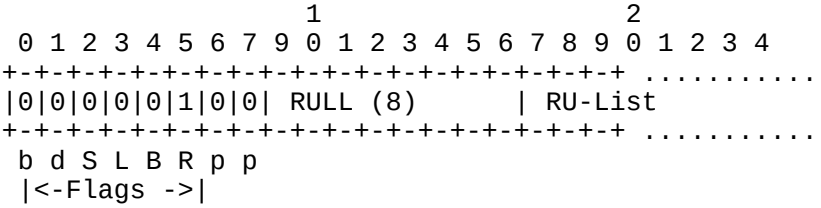


Figure 11: RU-List replication to leaf nodes

The SIDs of nodes to replicate to are always inside the neighbors RU, hence SID-list replication means replication with an RU-List and hence also an RULL field.

The b and d field for the replicating nodes RU itself determine only whether it should also broadcast and/or receive the packet, so their setting is irrelevant to the RU-List operation, so we assume in this example both are 0.

If the prior hop node was also replicating via an RU-List, this RU would also require a SID, but if this node is the root of the tree or the prior hop was replicating via a BitString, it does not require a SID. We show this simpler case. S=0, L=0 mean that instead of a SID, there are just two padding bits p before RULL.

Assume all the neighbors to replicate to are direct neighbors, so we encode local SIDs that fit into short, 10-bit long SIDs. In result, the RU-list is a sequence of 16-bit long RUs, one each for each neighbor to replicate to, as shown in Figure 12. And if the neighbor

was a couple of hops in the network topology away, one would use a global SID, in which case it likely might require a long SID encoding, and hence 24-bit for the RU.

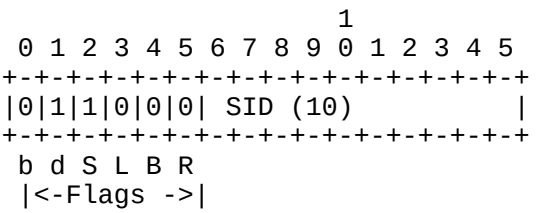


Figure 12: Leaf-node RU with SID

When the packet with the neighbors RU reaches that neighor, the RU is RU0, aka: the complete RTS header.

In this encoding, unlike the encoding of the prior version of this document, the SID itself carries no other semantic than the node it is targeted for.

When RTS is used (like BIER) as a standalone L3 network layer, this means that intermediate RTS capable, but not replicating nodes could simply unicast forward the packet to the node indicated by the RU0, similar to how loose hops operate in IPv6 with SRH. But that only works when the SID of RU0 is a global SID. So the replicating node may want to rewrite the SID in the RU to a global SID (including adjusting the length) when such a mode of operation is desirable. However, when operating RTS inside of IPv6, this loose-hop role would be served by an outer IPv6 header, and the RTS header would become another routing header. And the SID serves no value anymore, but can be replaced by a special value such as 0, or the RU be rewritten to not include a SID but just 2 bits of padding.

Likewise, if the SID used in the RU encoding is a local SID, then its semantic will likely be different on the receiving node, so if its needed for loose hop routing, it would need to be rewritten to a global SID. If used inside IPv6 or another network layer, local SIDs in RU0 should be removed or replaced by 0.

4.3.2.3. RU with SID-List and BitString

When would an RU require a SID and a BitString ? This is exactly the case when the prior-hop node was replicating based on RU-List, because this requires a SID in the next-hop RU, but that next-hop RU itself should replicate with a BitString. In this case, the RU for that node would include both a SID for the benefit of processing by the prior-hop node, and a BitString (including BSL/SD) for its own

replication.

As mentioned before, the SID itself has no relevance anymore after having been processed by the prior-hop replication engine, and might be removed in the packet copy to the target node, but in the packet header as encoded in the packet header for the root of the RTS tree, both SID and BitString would need to be in the RU.

4.3.2.4. BitString replication to non-leaf neighbors

When replicating to non-leaf neighbors via BitString, an RU-List parameter is necessary in the RU of the replicating node. Which raises the question, how the replicating node can know whether to replicate to a leaf or a non-leaf neighbor.

There are two options for this:

1. In the BIFT of the replicating node, each bit has a flag indicating whether or not the node indicated by the bit is always a leaf node or potentially also a non-leaf node. In this option, there needs to be an RU-list in the RU with an entry for each bit set in the BitString that indicates to not always be a lead node in the BIFT. Each such RU-list entry can be a simple 8-bit RU without SID or BitString when that neighbor is addressed without the need for a larger RU.

2. There is no such bit in the BIFT, then all neighbors always require an RU, which equally can be as short as 8-bit.

4.3.2.5. Functional subsets on individual nodes

Less capable forwarding engines may support only a subset of the encoding and then need to be able to discover from the flag-bits whether the RU destined for it contains an unsupported option and then stop processing the packet and raise an error, e.g.: ICMP.

One set of limitations that seems to be necessary to fit functionally limited forwarding engines could include all or a subset of the following:

o Only one size of SID can be supported. In this case, the long SID format should be supported, unless the type of device is known to always be leaf nodes in topologies, in which case only short SIDs may be supported.

o The variability of including an RU-list can not well be supported. In this case, the RU for such a node may need to always include a RUST field with a value of 0.

o The node can not support BitString replication for non-leaf neighbors. In this case, the RU with a BitString may not have the R-bit set and no RU-List. For common cases where this would be an issue, workarounds may be devised, for example, the node may have a local SID pointing back to it, so that the node needs to be encoded as 2 RU: The first RU uses RU-List replication, and one of the RU is for the local SID of the node itself, causing the packet to recirculate, and that RU would then use a BitString (without RU-List).

Because there are no "global" aspects to the parsing of RUs, it limitations on one type of node only have limited impact on the network-wide ability to construct tree.

4.3.3. Creating and Receiving copies

RTS relies on unicast forwarding procedures using the Encap Header(s) to receive packets and send copies of packets. Every copy of a packet created, except for those that are for local reception by a node, is sent towards a unicast address/SID according to the RTS SID it addresses.

The RU0 Flags are responsible for distinguishing the encoding of the following RU parameters but also provides the bits used for processing by so-called "leaves" of an RTS tree, where packets need to be delivered and/or broadcast to all "leaf" neighbors (where they are then delivered).

4.3.4. Creating copies because of RTS Header d=1

When d=1 is encountered in RU0, an (internal) copy of the packet is created in which the headers up to the RTS Header are disposed of according to the procedures specified for Encap Header(s) so that the Next Proto Payload after the RTS Header is processed.

4.3.5. Creating copies because of RTS Header b=1

When b=1 is set in RU0 flags, a list of unicast addresses/SIDs called the "all leaf neighbors" is used to create a separate copy of the packet for each element in that list. Each RTS node MAY have such a list.

For each packet copy made because of b=1, the RU0 for that neighbor is set with all Flags to 0 except for d=0. The RU0 for each such neighbor is thus 8 bit long. Typically, the "all-leaf-neighbors" list is (auto-)configured with the list of RTS L2 neighbors that are known to be leaves of the RTS domain.

5. Discussion

5.1. Encoding Efficiency vs. decoding challenges

One core challenge for RTS is encoding efficiency, aka: the size of the header required to address a particular number of leaves. Or with a defined maximum header size the maximum number of leaves that can be addressed.

One core encoding trick to increase encoding efficiency is to not indicate the semantic of variable fields but derive encoding from prior field or state knowledge.

One example for this is not to encode the length of a local bitstring but assume that this bitstring length is consistently known between the processing node and the node encoding the bitstring, such as ingress-PE, controller or sender application.

Consistent understanding of such control plane information to correctly encode packet headers already exists as a requirement in other headers, specifically MPLS and SR-MPLS and SRv6/SRH. The semantic of labels/SID is not necessarily global and static in nature but often also local and learned through control plane. If that control plane information is inconsistent, then encoding nodes may create incorrect headers that will be incorrectly processed.

For RTS, one additional challenge is when such consistent control plane information implies the length of fields, such as in the bitstring example. To reduce the problem space to what has been accepted in unicast solutions such as MPLS/SR, it may hence be necessary to explicitly include lengths for all variable lengths fields. But this is in the opinion of the authors an open issue to investigate further.

It may be possible and beneficial to instead of expanding the size of headers because of this issue, to look into control plane solutions to avoid this requirement. This could be based on the following mechanisms:

o Packets are forwarded only as L2 unicast with explicit addressing of destinations to prohibit hop-by-hop mis-delivery. Such as using L2 ethernet with globally unique L2 ethernet MAC destination addresses. This is what the solution currently assumes.

o Mechanisms to the control plane distributing the relevant information (such as lengths of bitstrings, semantics of SIDs) not only in an "eventual consistency" way, but in a "strict consistency" way. Aka: all nodes in the domain that can be addressed by RTS trees are known to have consistent control plane information relevant to the consistent encoding/decoding.

o Mapping this "strict consistency" encoding to a numeric "control plane version" value that can be carried as a new field in headers.

Such an approach may not be sufficient to answer all questions, such as how to change control plane information upon planned topology changes, but it should suffice to ensure that whoever encodes a packet can add the "control plane version" field to the header, and any node potentially parsing this header will have either consistent information or not accept the indicated "control plane version".

Short of deriving on such a solution, the encoding will become longer due to the need of explicitly including fields for the semantic of following fields. The encoding described has a range of cases where this option is already defined as an alternative.

Because of these additional, not currently standard control plane requirements, this version of the document includes now all variable aspects explicitly in the encoding.

5.2. Encapsulation considerations

5.2.1. Comparison with BIER header and forwarding

The RTS header is equivalent to the elements of a BIER/BIER-TE header required for BIER and BIER-TE replication.

(SI, SD, BSL, Entropy, Bitstring)

RTS currently does not specify an ECMP procedure to next-hop SIDs because it is part of the (unicast) forwarding to next-hops, but not to RTS replication.

Note that this is not the same set of header fields as [RFC8296], because that header contains more and different fields for additional functionality, which RTS would require to be in an Encap Header.

For the same reason, the RTS Header does also not include the [RFC8296] fields TC/DSCP for QoS, OAM, Proto (for next proto identification) and BFIR-id. Note that BFIR-id is not used by BIER forwarding either, but by BIER overlay-flow forwarding on BFIR and BFER.

Constraining the RTS header to only the necessary fields was chosen to make it most easy to combine it with any desirable encapsulation header.

RTS could use [RFC8296] as an Encap Header and BIER/[RFC8296] forwarding procedures, replacing only BIER bitstring replication to next-hop functionality with RTS replication.

In this case, the RTS Header could take the place of the bitstring field in the [RFC8296] header, using the next largest size allowed by BIER to fit the RTS header. SI would be unused, and SD could be used to run RTS, BIER and even BIER-TE in parallel through different values of SD, and all BIER forwarding procedures including ECMP to next-hop SIDs could be used in conjunction with RTS replication.

5.2.2. Comparison with IPv6 extension headers

The RTS header could be used as a payload of an an IPv6 extension header as similarly proposed for RBS in [I-D.eckert-msr6-rbs]. Note that the RTS header itself does not contain a simple length field that allows to completely skip across it. This is done because such functionality may not be required by all encapsulation headers / forwarding planes, or the format in which such a length is expected (unit) may be different for different forwarding planes. If required, such as when using the RTS header in an IPv6 extension header, then such a total-length field would have to be added to the Encap Header.

5.3. Encoding choices and complexity

5.3.1. SID vs bitstrings in recursive trees

The use of SID-lists in the encoding is a natural fit when the target tree is one that does not require replication on many of the hops through which it passes, such as when doing non-equal-cost load-splitting, such as in capacity optimization in service provider networks. In [RFC9262], Figure 2, such an example is called an "Overlay" (tree). In the SID list, each of the SID can easily be global, making it possible for a next-hop to be anywhere in the network. While it is possible to also use global SIDs in a bitstring, the decision to include any global (remote) SID as a bit in a bitstring introduces additional encoding size cost for every tree, and not only the ones that would need this bit. This is also the main issue of using such global SIDs in BIER-TE (where they are represented as forward_routed()) adjacencies.

When replicating to direct neighbors, SID lists may be efficient for sparse trees. In the RTS encoding, up to 127 direct neighbors could be encoded in 8 bit for each SID, so it is easy to compare the encoding efficiency to that of a bitstring. A router with 32 neighbors (assume leaf neighbors for simplicity) requires 32 bits to represent all possible neighbors, if 4 or fewer neighbors need to receive a copy, a SID-list encoding requires equal or fewer bytes to encode.

Use of the broadcast option is equally possible with SID-list or bitstrings. An initial scalability test with such an option was shown in slide 6 of [RBSatIETF115], but not included in any prior proposed encoding option; a better analysis of this option is subject to future work.

The ability of the RTS encoding to mix for the initial part(s) of a tree SID lists and then for leaves or tail parts of the tree bitstrings specifically addresses the common understanding that multicast trees typically are sparse at the root and may also be more "overlay" type, but then tend to become denser towards the leaves. Even if there is the opportunity to create more replications within the first hops of a tree, that approach may often not result in the most costs-effective ("steiner") trees.

5.3.2. Limited per-node functionality

The encoding proposed in this (version of the) draft is based on P4 development of a reference proof of concept called SEET+. It does not exactly follow the encoding, but attempts to generalize it to support both more flexible and more constrained forwarding platforms.

Because in a recursive tree an individual node only needs to parse one part of the tree that is designated for it, this type of encoding allows it to support different flexible forwarding engines in the same network: The recursive units that are to be parsed by a less flexible nodes forwarding engine simply can not use all the possible options, but only those options possible on that forwarding engine.

To then support such a mix of forwarding engines, the following architectural elements are necessary:

1. The control plane of every node needs to signal the subset of functionality so that the place where the control plane constructs an RTS will know what limitations it has.

2. The forwarding plane of nodes not supporting full functionality needs to be able to reliably recognize when the encoding utilizes a feature not supported, stop parsing/replicating the packet and raise an error (ICMP or similar) as necessary.

For example, some type of forwarding engines could have the following set of limitations.

The forwarding engine might not be possible to process the recursive bitstring structure because there could be insufficient code space for both recursive SID and recursive bitstring option. In this case the limitation would be that a bitstring type RU to be processed by this node does have to be a leaf which is not followed by a set of RU for downstream neighbors.

Consider such a node is located in a distribution ring serving as an aggregation node to a large number of lead neighbors. In this topological case, a local bitstring would be ideal to indicate which of the leaf neighbors the packet has to be replicated. However, the packet would equally need to be forwarded to the next ring neighbor, and that ring neighbor would need its own RU. Which would not be directly supportable on this type of node.

To handle this situation, such a limited functionality node would assign a local SID to itself and trees that pass through this node would need to encode first a SID type RU indicating the ring neighbor as well as the node itself. The RU for this nodes SID itself would then be a local bitstring RU. Likely the node would then also process the packet twice through so-called recirculation. In addition, this approach increases the side of the RTS encoding.

5.4. Differences over prior Recursive BitString (RBS) encodings proposal

The encoding for bitstrings proposed in this draft relies again on discarding of unnecessary RU instead of using offset pointers in the header to allow parsing only the relevant RU.

Discarding unnecessary RU has the benefit, that the total size of the header can be larger than if offset pointers where used. Forwarding engines have a maximum amount of header that they can inspect. With offset pointers, the furthest a node has to look into the RTS header is the actual size of the RTS header. With discarding of unnecessary RU, this maximum size for inspection can be significantly less than the maximum RTS header size. Consider the root of tree has two neighbors to copy to and both have equal size RU, then this root of the tree only needs to inspect up to the beginning of the second RU (the SID or bitstring in it).

6. Security considerations

TBD

7. Acknowledgments

The local bitstrings part of this work is based on the design published by Sheng Jiang, Xu Bing, Yan Shen, Meng Rui, Wan Junjie and Wang Chuang {jiangsheng|bing.xu|yanshen|mengrui|wanjunjie2|wangchuang }@huawei.com, see [CGM2Design]. Many thanks for Bing Xu (bing.xu@huawei.com) for editorial work on the prior variation of that work [I-D.xu-msr6-rbs].

8. Changelog

8.1. -00

This version was derived from merging the [SEET] and RBS proposals (called BEET in the [SEET] presentation) into a single encoding mechanism. SEET is a proposal for per-tree-hop replication with SID-Lists, RBS is a proposal for replication with per-hop local BitStrings. Both embody the same idea of Recursive Units to represent each hop in the tree, but the content of these recursive Units is different whether SID-Lists or BitStrings are used.

Because the processing of recursive structures are directly impacted and limited by the capabilities of forwarding planes, and because by the time of writing this -00 draft, only separate SEET and separate RBS reference Proof Of Concept implementations existed, this version does propose to support only separate trees: A tree only constructed from SID-Lists, or a Tree only supported from local BitStrings. Each packet needs to choose which format to use.

8.2. -01

After -00 version, reference Proof Of Concept work was done to investigate whether it was possible in a single forwarding plane to support trees that can have both SID-lists and local BitStrings. The main reason for this was that supporting both options as ships in the night turned out to be costing too much code space on limited forward plane engines.

In result, the reference limited forwarding plane engine could be made to support trees with a limited mix of SID-list replication and local BitStrings. Local BitStrings could only be used on hops prior to leaves, e.g.: no further local BitString replication was possible (single stage).

RTS -01 is now a proposal that generalizes on these PoC experiences by proposing a format for recursive units that allows to implement the limited functionality possible on limited capability forwarding engines, but that also allow to implement arbitrary mixing of per-hop use of SID-List and BitString replication on forwarding planes that are more capable. The encoding is also chosen so that nodes can easily recognize if the packet embodies an encoding option not supported by it and reject the packet.

In addition, the -00 version of RTS did embody a range of optimizations for shorter encoding length by avoiding packet header fields that contain information such as parameter length that could be assumed to be known by both the generator and consumer of the header element (RU). This has been removed in this version of the proposal and replaced by explicitly integrating all variable length field information as well as other interpretation semantics explicitly into the header encoding. This change is not necessarily the final conclusion of thi issue, but the document lays out what other control plane requirements would first have to be built to be able to have more compact encodings - and those requirements are not commonly support by the most widely used control planes.

8.3. -02

Refresh.

9. References

9.1. Normative References

[RFC6554] Hui, J., Vasseur, JP., Culler, D., and V. Manral, "An IPv6 Routing Header for Source Routes with the Routing Protocol for Low-Power and Lossy Networks (RPL)", RFC 6554, DOI 10.17487/RFC6554, March 2012, <https://www.rfc-editor.org/rfc/rfc6554>.

[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/rfc/rfc8200>.

[RFC8279] Wijnands, IJ., Ed., Rosen, E., Ed., Dolganow, A., Przygienda, T., and S. Aldrin, "Multicast Using Bit Index Explicit Replication (BIER)", RFC 8279, DOI 10.17487/RFC8279, November 2017, <https://www.rfc-editor.org/rfc/rfc8279>.

[RFC8296] Wijnands, IJ., Ed., Rosen, E., Ed., Dolganow, A., Tantsura, J., Aldrin, S., and I. Meilik, "Encapsulation for Bit Index Explicit Replication (BIER) in MPLS and Non-MPLS Networks", RFC 8296, DOI 10.17487/RFC8296, January 2018, <https://www.rfc-editor.org/rfc/rfc8296>.

[RFC8402] Filsfils, C., Ed., Previdi, S., Ed., Ginsberg, L., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", RFC 8402, DOI 10.17487/RFC8402, July 2018, <https://www.rfc-editor.org/rfc/rfc8402>.

[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/rfc/rfc8754>.

[RFC9262] Eckert, T., Ed., Menth, M., and G. Cauchie, "Tree Engineering for Bit Index Explicit Replication (BIER-TE)", RFC 9262, DOI 10.17487/RFC9262, October 2022, <https://www.rfc-editor.org/rfc/rfc9262>.

9.2. Informative References

[CGM2Design] Jiang, S., Xu, B. (Robin)., Shen, Y., Rui, M., Junjie, W., and W. Chuang, "Novel Multicast Protocol Proposal Introduction", 10 October 2021, <<https://github.com/BingXu1112/CGMM/blob/main/Novel%20Multicast%20Protocol%20Proposal%20Introduction.pptx>>.

[CGM2report] "Carrier Grade Minimalist Multicast CENI Networking Test Report", 1 August 2022, <<https://raw.githubusercontent.com/network2030/publications/main/CENI_Carrier_Grade_Minimalist_Multicast_Networking_Test_Report.pdf>>.

[I-D.eckert-bier-cgm2-rbs] Eckert, T. T. and B. Xu, "Carrier Grade Minimalist Multicast (CGM2) using Bit Index Explicit Replication (BIER) with Recursive BitString Structure (RBS) Addresses", Work in Progress, Internet-Draft, draft-eckert-bier-cgm2-rbs-01, 9 February 2022, <https://datatracker.ietf.org/doc/html/draft-eckert-bier-cgm2-rbs-01>.

[I-D.eckert-bier-cgm2-rbs-00]
Eckert, T. T., "Carrier Grade Minimalist Multicast (CGM2) using Bit Index Explicit Replication (BIER) with Recursive BitString Structure (RBS) Addresses", Work in Progress, Internet-Draft, draft-eckert-bier-cgm2-rbs-00, 25 October 2021, <https://datatracker.ietf.org/doc/html/draft-eckert-bier-cgm2-rbs-00>.

[I-D.eckert-bier-cgm2-rbs-01]
Eckert, T. T. and B. Xu, "Carrier Grade Minimalist Multicast (CGM2) using Bit Index Explicit Replication (BIER) with Recursive BitString Structure (RBS) Addresses", Work in Progress, Internet-Draft, draft-eckert-bier-cgm2-rbs-01, 9 February 2022, <https://datatracker.ietf.org/doc/html/draft-eckert-bier-cgm2-rbs-01>.

[I-D.eckert-bier-rbs]
Eckert, T. T., Menth, M., Geng, X., Zheng, X., Meng, R., and F. Li, "Recursive BitString Structure (RBS) Addresses for BIER and MSR6", Work in Progress, Internet-Draft, draft-eckert-bier-rbs-00, 24 October 2022, <https://datatracker.ietf.org/doc/html/draft-eckert-bier-rbs-00>.

[I-D.eckert-msr6-rbs]
Eckert, T. T., Geng, X., Zheng, X., Meng, R., and F. Li, "Recursive Bitstring Structure (RBS) for Multicast Source Routing over IPv6 (MSR6)", Work in Progress, Internet-Draft, draft-eckert-msr6-rbs-01, 24 October 2022, <https://datatracker.ietf.org/doc/html/draft-eckert-msr6-rbs-01>.

[I-D.xu-msr6-rbs]
Xu, B., Geng, X., and T. T. Eckert, "RBS(Recursive BitString Structure) for Multicast Source Routing over IPv6", Work in Progress, Internet-Draft, draft-xu-msr6-rbs-01, 30 March 2022, <https://datatracker.ietf.org/doc/html/draft-xu-msr6-rbs-01>.

[Menth20h] Merling, D., Lindner, S., and M. Menth, "P4-Based Implementation of BIER and BIER-FRR for Scalable and Resilient Multicast", IEEE in "Journal of Network and Computer Applications" (JNCA), vol. 196, Nov. 2020, preprint https://atlas.informatik.uni-tuebingen.de/~menth/papers/Menth20h.pdf, DOI 10.1016/j.jnca.2020.102764, n.d., <https://doi.org/10.1016/j.jnca.2020.102764>.

[Menth21] Merling, D., Lindner, S., and M. Menth, "Hardware-based Evaluation of Scalable and Resilient Multicast with BIER in P4", IEEE in "IEEE Access", <https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=6287639>, vol. 9, p. 34500 - 34514, March 2021, <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9361548>, n.d..

[Menth23] Merling, D., St端ber, T., and M. Menth, "Efficiency of BIER Multicast in Large Networks", IEEE accepted for "IEEE Transactions on Network and Service Managment", preprint <https://atlas.cs.uni-tuebingen.de/~menth/papers/Menth21-Sub-5.pdf>, n.d.. preprint

[Menth23f] Lindner, S., Merling, D., and M. Menth, "Learning Multicast Patterns for Efficient BIER Forwarding with P4", IEEE in "IEEE Transactions on Network and Service Managment", vol. 20, no. 2, June 2023, preprint https://atlas.cs.uni-tuebingen.de/~menth/papers/Menth22-Sub-2.pdf, n.d..

[RBSatIETF115] Eckert, T., Menth, M., Gend, X., Zhen, X., Meng, R., and F. Li, "RBS (Recursive BitString Structure) to improve scalability beyond BIER/BIER-TE, IETF115", November 2022, <<https://datatracker.ietf.org/meeting/115/materials/slides-115-bier-recursive-bitstring-structure-rbs-beyond-bierbier-te-00>>.

[RFC791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/rfc/rfc791>.

[SEET] Lindner, S., Menth, M., and T. Eckert, "P4 Tofino Implementation Experiences with Advanced Stateless Multicast Source Routing", 1 November 2023, <<https://datatracker.ietf.org/meeting/118/materials/slides-118-bier-02-ietf118-bier-p4-02.pdf>>.

Appendix A. Evolution to RTS

The following history review of RBS explains key aspects of the road towards RTS and how prior document work is included (or not) in this RTS work.

A.1. Research work on BIER

Initial experience implementation with implementation of BIER in PE was gained through "P4-Based Implementation of BIER and BIER-FRR for Scalable and Resilient Multicast", [Menth20h], from which experience was gained that processing of large BIER bitstring requires significantly complex programming for efficient forwarding, as described in "Learning Multicast Patterns for Efficient BIER Forwarding ith P4", [Menth23f]. Further evaluations where researched through "Hardware-based Evaluation of Scalable and Resilient Multicast with BIER in P4", [Menth21] and "Efficiency of BIER Multicast in Large Networks", [Menth23].

A.2. Initial RBS from CGM2

The initial, 2021 [I-D.eckert-bier-cgm2-rbs-00] introduces the concept of Recursive Bitstring Forwarding (RBS) in which a single bitstring in a source routing header for stateless multicast replication as introduced by BIER and re-used by BIER-TE is replaced by a recursive structure representing each node of a multicast tree and in each node the list of neighbors to which to replicate to is represented by a bitstring.

Routers processing this recursive structure do not need to process the whole structure, instead, they only need to examine their own local bitstring, and replicate copies to each of the neighbors for which a bit is set in the bitstring for this node. For each copy the recursive structure is rewritten so that only the remaining subtree behind the neighbor remains in the packet header. By only having to examine a "local" (and hence short) bitstring, RBS processing can arguably be simpler than that of BIER/BIER-TE. By discarding the parts of the tree structure not needed anymore, there is also no need to change bits in the bitstring as done in BIER/BIER-TE to avoid loops.

This initial version of RBS encoding is based on a design originally called "Carrier Grade Minimalist Multicast" (CGM2), and which started as a research project whose design is summarized in [CGM2Design]. A vendor high-speed router implementation proof-of-concept was done, as well as a wide-area proof-of-concept research network deployment, which was documented for the 2022 Nanjing "6th future Network Development Conference". An english translation of the report can be found at [CGM2report].

A.3. RBS scalability compared to BIER

The 2022 [I-D.eckert-bier-cgm2-rbs-01] version of the document adds topology and testing information about a simulation comparing RBS with BIER performance in a dense, high-speed network topology. It is showing that the number of replications required to reach an increasing number of receivers does grow slower with RBS than with BIER, because in BIER, it is necessary to send another packet copy from the source whenever receivers in a different Set Identifier Bitstring (SI) are required, whereas RBS requires to only create multiple copies of a packet at the source to reach more receivers whenever the RBS packet header size for one packet is exhausted. The results of this simulation are shown in slide 6 of [RBSatIETF115].

While RBS with its explicit description of the whole multicast tree structure seems immediately like (only) a replacement for BIER-TE, which does the same, but encodes it in a "flat"BIER bitstring (and incurring more severe scalability limitations because of this), this simulation shows that the RBS approach may also compete with BIER itself, even though this may initially look counter-intuitive because information not needed in the BIER encoding - intermediate hops - is encoded in RBS.

The scalability analysis also assumes one novel encoding optimization, indicating replication to all leaf neighbors on a node. This allow to even further compact the RBS encoding for dense trees, such as in applications like IPTV. Note that this optimization was not included in any of the RBS proposal specifications, but it is included in this RTS specification.This optimization leads to the actual reduction in packet copies sent for denser trees in the simulation results.

A.4. Discarding versus offset pointers

[I-D.eckert-bier-rbs] re-focusses the work of the prior [I-D.eckert-bier-cgm2-rbs] to focus only on the forwarding plane aspects, removing simulation results and architectural considerations beyond the forwarding plane.

It also proposes one then considered to be interesting alternative to the encoding. Instead of discarding unnecessary parts of the tree structure for every copy of a packet made along the tree, its forwarding machinery instead uses two offset pointers in the header to point to the relevant sub-structure for the next-hop, so that only a rewrite of these two pointers is needed. This replicates the offset-rewrite used in unicast source-routing headers such as in IP, [RFC791], or IPv6, [RFC6554] and [RFC8754].

Discussions about discarding vs. changing of offset since then seems to indicate that changing offsets may be beneficial for forwarders that can save memory bandwidth when not having to rewrite complete packet headers, such as specifically systems with so-called scatter-gather I/O, whereas discarding of data is more beneficial when forwards do have an equivalent of scatter/gather I/O, something which all modern high-speed routers seem to have, including the platform used for validation of the approach described in this document.

A.5. Encapsulations for IPv6-only networks

Whereas all initial RBS proposal did either not propose specific encapsulations for the RBS structure and/or discussed how to use RBS with the existing BIER encapsulation [RFC8296], the 2022 [I-D.xu-msr6-rbs] describes the encapsulation of RBS into an IPv6 extension header, in support of a forwarding plane where all packets on the wire are IPv6 packets, rewriting per-RBS-hop the destination IPv6 address of the outer IPv6 header like pre-existing unicast IPv6 stateless source routing solutions too ([RFC6554], [RFC8754]).

This approach was based on the express preference desire of IPv6 operators to have a common encapsulation of all packets on the wire for operation reasons ("IPv6 only network design") and to share a common source-routing mechanism operating on the principle of per-steering-hop IPv6 destination address rewrite.

[I-D.eckert-msr6-rbs] extends this approach by adding the offset-pointer rewrite of [I-D.eckert-bier-rbs] to the extension header to avoid any change in length of the extension header, but it also includes another, RBS independent field, the IPv6 multicast destination address to the extension header. Only this additional would allow for RBS with a single extension header to be a complete IPv6 multicast source-routing solution. BIER/BIER-TE or any encapsulation variations of RBS without such a header field would always require to carry a full IPv6 header as a payload to provide end-to-end IPv6 multicast service to applications.

A.6. SEET and BEET

To validate concepts for recursive trees, high-speed reference platform proof of concept validation was done for booth SID-list and local-bitstring based recursive trees in 2023. This is called in the research work SEET and BEET. See [SEET]. Paper to follow.

Contributors

Xuesong Geng
Huawei
China
Email: gengxuesong@huawei.com

Xiuli Zheng
Huawei
China
Email: zhengxiuli@huawei.com

Rui Meng
Huawei
China
Email: mengrui@huawei.com

Fengkai Li
Huawei
China
Email: lifengkai@huawei.com

Authors' Addresses

Toerless Eckert (editor)
Futurewei Technologies USA
2220 Central Expressway
Santa Clara, CA 95050
United States of America
Email: tte@cs.fau.de

Michael Menth
University of Tuebingen
Germany
Email: menth@uni-tuebingen.de

Steffen Lindner
University of Tuebingen
Germany
Email: steffen.lindner@uni-tuebingen.de

Yisong Liu
China Mobile
China
Email: liuyisong@chinamobile.com