Workgroup: BIER
Internet-Draft: draft-eckert-bier-rbs-00
Published: 24 October 2022
Intended Status: Experimental
Expires: 27 April 2023
A   T. Eckert, Ed.   M. Menth
uFuturewei Technologies USA   University of Tuebingen
t
h
o
r
s
:
X. Geng   X. Zheng
Huawei 2012 NT Lab   Huawei 2012 NT Lab
R. Meng   F. Li
Huawei 2012 NT Lab   Huawei 2012 NT Lab

Recursive BitString Structure (RBS) Addresses for BIER and MSR6

Abstract

This memo introduces a compact data-structure representation of multicast trees called "Recursive Bitstring Structure" (RBS) and its use for (stateless) forwarding of packets that include this structure in their header. It is intended as an alternative to "flat" bitstring addresses as used in BIER and BIER-TE or possible forwarding plane variations such as MSR6. RBS aims to improve performance and scalability over flat bitstrings and simplify operations.

Operations is simplified because RBS does not require the use, management and optimization of network-wide configured address spaces BIFT-ID and SI and because one common RBS mechanism can replace flat bitstring addresses for both shortest-path forwarding and tree engineered forwarding. It intends to improve performance by allowing multicast to sparse set of receivers in larger networks with fewer packets and it intends to improve scalability by requiring less BIFT state on routers.

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). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on 27 April 2023.

Copyright Notice

Copyright (c) 2022 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 (https://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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction
  1.1. BIER review
  1.2. BIER-TE review
  1.3. RBS introduction
2. RBS Network Architecture
  2.1. Controller centric
  2.2. Distributed/Decentralized
  2.3. Host-to-host
3. Overview
  3.1. Example
  3.2. RBS-BIFT
  3.3. RBS Address
  3.4. Processing on R1 in the example
4. Specification
  4.1. RBS Address
  4.2. RBS-BIFT
  4.3. RBS address creation
  4.4. Common RBS processing
  4.5. Receiving RBS packets
  4.6. Encapsulation considerations
  4.7. RBS forwarding Pseudocode
5. Using RBS with BIER and RFC8296 encapsulation
6. Using RBS with IPv6 extension header encapsulation
7. Security considerations
8. Acknowledgments
9. Changelog
10. References
  10.1. Normative References
  10.2. Informative References
Appendix A. High-speed implementation considerations
Appendix B. Complete RBS example
Appendix C. Replication efficiency performance considerations
  C.1. Reducing number of duplicate packet copies
  C.2. Statistical Analysis of performance gain
Authors' Addresses

1. Introduction

This memo introduces a compact data-structure representation of multicast trees called "Recursive Bitstring Structure" (RBS) and its use for (stateless) forwarding of packets that include this

structure in their header. It is intended as an alternative to "flat" bitstring addresses in BIER and BIER-TE or their possible variations such as MSR6. RBS aims to improve performance and scalability over flat bitstrings and simplify operations.

Operations is simplified because RBS does not require the use, management and optimization of network-wide configured address spaces BIFT-ID and SI and because one common RBS mechanism can replace flat bitstring addresses for both shortest-path forwarding and tree engineered forwarding.

This document calls the bitstring addressing used today in BIER and BIER-TE "flat" solely as simple to remember distinction to the "recursive" bitstring addressing used by RBS.

The document is structured as follows:

The introduction reviews the aspect of BIER and BIER-TE that RBS intends to improve on to then give an introduction to RBS.

The architecture section describes the models how RBS can integrate into comprehensive forwarding architectures such as those defined by BIER/BIER-TE.

The Overview section explains RBS address encoding and forwarding based on an example

The Specification section defines normative requirements of RBS including forwarding Pseudocode.

The section on using RBS with BIER and RFC8296 encapsulation describes proposed normative aspects when applying RBS to BIER.

Appendices discuss High-speed implementation considerations and current insight into how well RBS can help to reducing the number of of packet required to be sent with RBS.

1.1. BIER review

In BIER, each bit of a bitstring indicates a BIER egress router (BFER). When using [RFC8296] encapsulation, BitString can have a BitStringLength (BSL) of 2^N, N=6...12. Routers may not be able to support up to 2^12=496 long BitStrings. The most common BSL assumed to be supported is 256.

When a network has a number M of BFER, M >> BSL support by routers in the network, it it necessary to use multiple "sets" of BitStrings across the network to address all BFER. Each set has a SetIdentifier (SI). BFER are identified in BIER via their BFR-id which is (SI << N | 2^BP), where BP is the BitPosition, the bit in the BitString used for this BFER: the lower N bits of the BFR-id are the BP of the BFER and the high order bits the SI. In [RFC8279] this is also shown as (SI:BitString), where the BitString has only the BP of the BFER set.

When a network requires k SI to address all BFER, then a message that needs to be sent to k arbitrary BFER in the network may require to send as many as k BIER packets - when each of the k BFER has a different SI. The total number of packets required for any possible

set of receiver BFER is a stochastic matter. At best, BIER can reduce the number of packet required to reach M BFER to M/BSL.

Intelligent allocation of BFR-id can lead to a more efficient delivery of BIER traffic. Consider a network requiring k SI and random allocation of BFR-id so that every edge area of the network has at least one BFR in each of the k BFR. This makes it more likely that up to k BIER packets need to be sent into such an area to reach subsets of BFR in it. Compare this to an allocation that attempts to minimize the number of SI used by BFR in each individual area. This will result in fewer BIER packets required to reach subsets of BFR in such an area.

1.2. BIER-TE review

Whereas BIER relies on hop-by-hop routing to direct its traffic, Tree Engineering for BIER (BIER-TE, [RFC9262]) is based on explicit source routing by encoding the whole delivery tree in a BitString. This is done to support similar type of requirements as those that require explicit source routing in IP unicast, also called traffic steering, such as SRH in IPv6, but also multicast specific ones, such as lowest cost trees (so-called Steiner trees).

BIER-TE was designed to reuse the packet encodings of BIER and as much as feasible of the BIER forwarding plane. It therefore relies on the same type of flat BitStrings (and their addressing via SI) as BIER. In BIER-TE, each bit of a BitString indicates an adjacency. In the most simple case those adjacencies are subnet adjacent BFR for the edges of the multicast tree which are called forward_connected() in BIER-TE, and local_decap() adjacencies for the leaves of the tree - effectively its BFER.

Because BIER-TE needs to represent the whole delivery tree and not only its leaves/BFER in the BitString, intelligent and efficient allocation of BP is even more important than in BIER, and a significant number of BP in every SI must be allocated to transit hops of the network to allow defining BIER-TE trees across those transit hops. In large networks this may also lead to the need to allocate BP across multiple SI for the same transit hops and thus a much larger total number of BP required to represent a smaller number of BFER and transit hop adjacencies - and in result also more BIER-TE packets required for the same message to send to the larger number of different SI required.

1.3. RBS introduction

One way to understand the Recursive BitString Structure address is to think of it as an evolution of BIER-TE flat bitstrings. Every single BFR processing a BIER-TE bitstring only needs to look at a small subset of the BP in it: those BP that indicate adjacencies of this BFR. All the other bits are ignored because they are adjacencies on other BFR.

Consider we decompose a BIER-TE BitString into separate smaller bitstrings - one each for every BFR on the multicast tree that we want to encode. The BitString for each BFR now only needs to have a BP for each subnet adjacent (neighbor) BFR. And an equivalent to the

local_decap() BP to indicate to the BFR itself to be a leaf/BFER on the multicast tree itself.

With this step, RBS eliminates the complex optimization problems resulting from the flat BitStrings: There is no need anymore for a network wide SI address space and optimizing which adjacencies and BFR-id to put into which SI. There is no hard partitioning by SI: A tree can span an arbitrary subset of BFR. Just the total encoded size of the tree needs to fit into an appropriate maximum header field size. And if the target tree is too large, then it can arbitrarily be broken up into overlapping subtrees - all originating at the sender, but each only delivering to a subset of BFER small enough so the encoded tree fits into the target packet header size. And none of these optimization have to happen at network configuration time by seeding optimized BIFT, but it happens when building an RBS address on on ingress router or with the help of a controller.

The RBS encoding is called recursive, because it consists of such a local BitString for the first BFR of the tree (BFIR), followed by a sequence of RBS sub-trees, one for each adjacent BFR whose BP is set in the first BFR BitString. Whenever a packet is forwarded to such an adjacent BFR, the RBS addressing information is appropriately updated such that that BFR will only have to examine the local BitString for that BFR. In result, every BFR in RBS only has to examine - like in BIER and BIER-TE a single BitString. What really changes is that instead of clearing bits in a flat bitstring as done in BIER/BIER-TE, every hop advances the decoding offset into the RBS address structure to look at a different, small local BitString.

2. RBS Network Architecture

2.1. Controller centric

RBS may simply use the same network architecture as BIER-TE as shown in Figure 1, and operations of the Controller is significantly simplified because the complex allocation of BP across SI, especially the allocation of BP for transit adjacencies is eliminated.

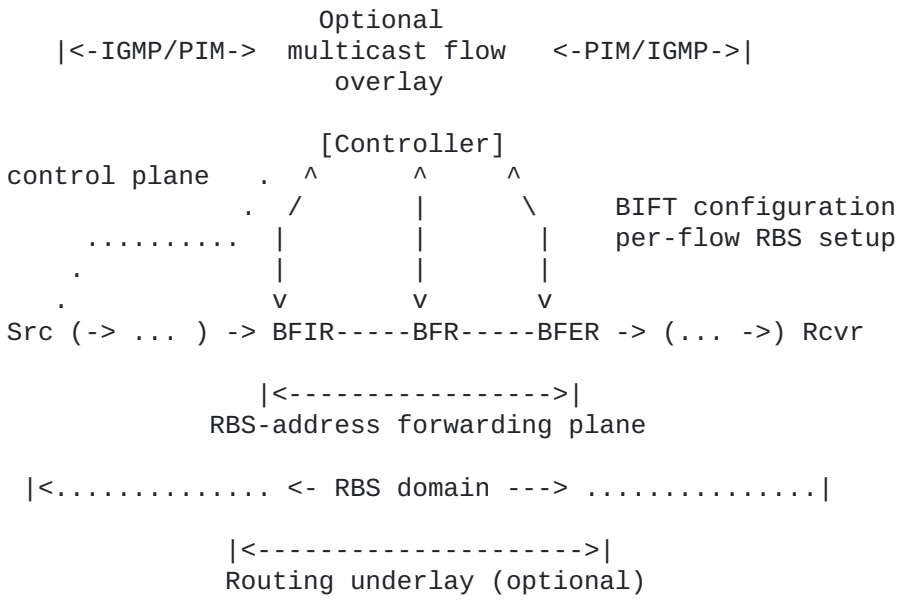


Figure 1: RBS Architecture with Controller

2.2. Distributed/Decentralized

Instead of a controller centric network architecture, RBS also lends itself to a distributed/decentralized model, similar to the typical deployment model of BIER, with extensions to a link-state IGP routing protocol.

Every BFR can automatically allocate its BFR neighbors and derive the size of its local BitString and allocation of BP to each neighbor from it. This is then signaled via appropriate link-state IGP extensions.

BFIR can derive not only the hop-by-hop paths towards BFER from this IGP information, but also the necessary local BitString for each BFR on a tree. In the most simple csae, these paths are the shorted-paths normally calculated by link-state IGP, but for traffic-engineering purposes, this can easily include all type of constrained path calculations.

It is this model that would be attractive, when there are no tree engineering / traffic engineering requirements, but RBS is simply used to replace flat bitstrings for BIER to simplify its operations and (depending on size / topology of network) improve its scale / performance.

2.3. Host-to-host

To eliminate the need for an IP Multicast flow overlays and allow utilization of benefits of bitstring addressing at the application level (e.g.: eliminating group membership management for the network), the RBS domain may extend all the way into Sender/Receiver hosts. This is possible with BIER/BIER-TE as well, but when the total number of sender/receiver hosts is for example a factor 10 larger than the number of BFR in BIER, then the elimination of the network wide SI/BP allocation issue of BIER/BIER-TE could help to make this model easier deployable with RBS than with BIER-TE/BIER.

To avoid dependencies against initial operating system level network stack upgrades on hosts, such deployment option could for example be introduced by tunneling the RBS packets over UDP to first-hop BFIR/ BFER as well as appropriate routing / controller plane extensions.

3. Overview

This section gives a more thorough run run-through of the life of a packet forwarded with an RBS address.

3.1. Example

Figure 2 shows the example network topology. R1 has network connections to R2, R3, R4, R5 (not shown) and R6. R3 and R4 have connections to R1, R7, R8, R9 and R10. R9 has connections to R3, R4, and further, not shown routers. For the purpose of explaining RBS, it is irrelevant whether those connections are separate L2 point-to-point links or adjacencies on shared LANs.

The example multicast tree encoded as an RBS address utilizing topology information as explained in Section 2 is R1 -> (R2, R3 -> (R7), R4 -> (R9 -> (...), R10), R6): The packet originates in the trees root R1, which needs to form the appropriate packet header with this trees RBS address and replicate the packet to R2, R3, R4 and R6. R2, R4 and R6 should receive the packet as domain egress routers. R3 should only replicate the packet to R7, and R7 should replicate the packet to R9 and R10. R10 should only receive the packet, R9 should receive the packet and further replicate it to further routers not shown.

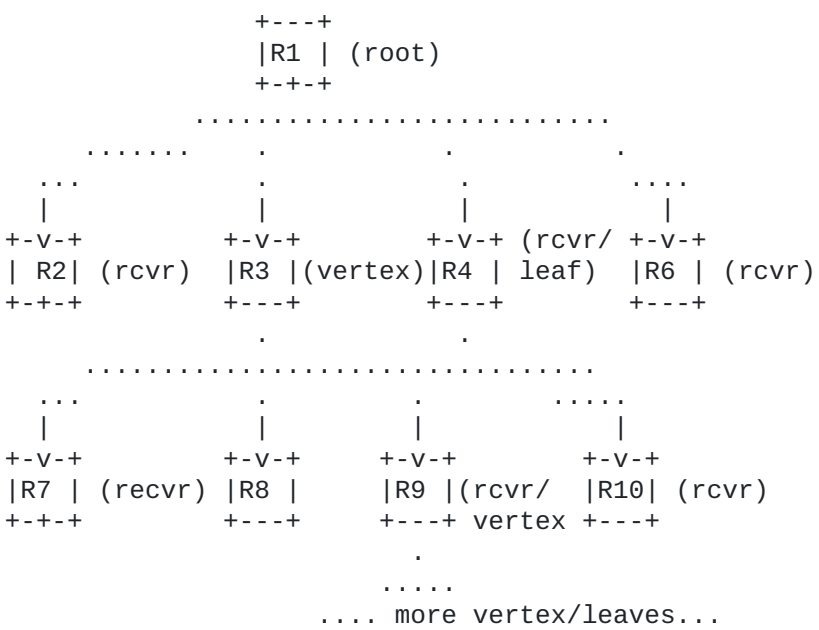


Figure 2: Example Topology/RBS tree

R7, R10 and some MSER behind R9. Given how R7, R8, R8, R10 and the router behind R9 can be reached via both either R3 and R4, this tree involves an explicit packet steering and replication (tree

engineering) choice of using R3 instead of R4 to reach R7, and R4 instead of R3 to reach R9, R10 (and routers below R9).

3.2. RBS-BIFT

Every router has an RBS "Bit Index Forwarding Table" (RBS-BIFT) that defines which BitPosition (BP) (1..N) indicates which adjacency. Figure 3, shows the example RBS-BIFT for R1.

BP	R Flag	Adjacency
1	0	receive
2	0	R2
3	1	R3
4	1	R4
5	0	R5
6	0	R6

Figure 3: BIFT on R1

The "receive" adjacency is the BP indicating that the packet is to be received by the router itself. The (R)ecursive flag indicates whether the adjacency when set in the BitString of an RBS address will have a subtree (Recursive Unit, see below) associated with it.

The absence of the R flag allows for more compact RBS encodings or adjacencies that for the purpose of RBS are not used for transit. In the example, R2, R5 and R6 are connected to R1 but also leaf router in the topology. Hence they have R=0 in the RBS-BIFT of R1.

3.3. RBS Address

The RBS address as shown in Figure 4 consists of RU-Length, RU-Offset and RecursiveUnit0. Depending on packet header encoding, these fields do not need to be encoded sequentially.

A RecursiveUnit (RU) is the unit of data processed by a particular router Rx on the multicast tree encoded by the RBS address. RU0 is the RU processed by the root of the tree. An RU consists of the BitString whose length is the length of the RBS-BIFT of Rx, followed by (N-1) AddressFields and N RUs. N is the number of BP set in BitString with R=1 flag set - e.g. which do need an RU in the RBS address. Each AddressField indicates the length of one RU. There are only N-1 AF for N RU because the length of the N'th RU can be derived by calculation, saving for every router on the tree one AF field, and therefore resulting in a more compact encoding.

RU-Offset indicates the offset of the current RU from the start of RU0. '$' in Figure 4 is the first bit of RU0, and a value of RU-Offset=0 indicates that the RU starts at '$' - and is therefore RU0.

For every copy of an RBS packet made by a router, RU-Offset and RU-Length are updated. None of the other fields of the RBS-Address are modified for RBS forwarding.

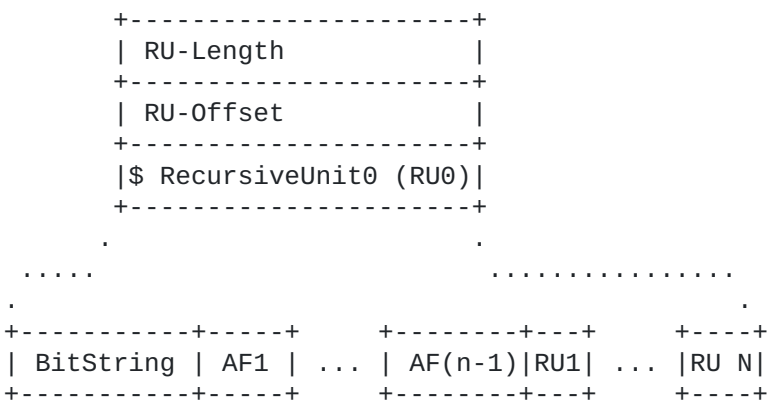


Figure 4: RBS Address

3.4. Processing on R1 in the example

In the example, the root of the tree is is R1, so the BitString of RU0 is as shown in Figure 5

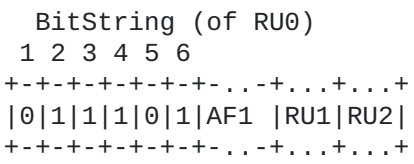


Figure 5: RU for R1 (RU0)

When RBS forwarding in a router processes the RBS address, the length of the BitString is known from the length of the RBS-BIFT. In the case of R1 it is therefore known to be 6 bits long.

Two of the BP set in the BitString, BP3 for R3 and for R4 have R=1 in the RBS-BIFT of R1, therefore (N-1)=1 AF field must follow and N=2 RU must follow in the RBS address for RU0 - one for R3, one for R4.

When R1 creates packet copies to R3 and R4, it will rewrite RU-Length and RU-Offset accordingly, so that RU-offset will point to RU1 for the packet towards R3 and to RU2 for the packet towards R4, and RU-Length indicates the length of RU1 or RU2.

This forwarding process repeats on every hop along the tree. When a packet copy is made on a BP with R=0, RU-Length is set to 0. When such a packet copy is received, it indicates that no further RU lookup is required, and the packet is only received - same as processing for a receive BU.

4. Specification

4.1. RBS Address

Any RBS router MUST support to parse its RU with AF entries that are 8 bit in size. Any RBS routers SHOULD support to decode a variable length AF encoding, where 0XXXXXXX (8-bit length AF field) is used to encode a 7-bit XXXXXXX (0..127) values, and where 1XXXXXXXXXXX is used to encode an 12-bit value XXXXXXXXXXX. All values indicate the size of an RU in bits, therefore allowing up to 4096 bit long RU.

An RBS router MUST support processing the BitString size of its configured RBS-BIFT (see Section 4.2).

RBS routers MUST support RU-Length and RU-Offset encodings of 12 bits.

4.2. RBS-BIFT

An Router must support for its RBS-BIFT to be configured with a number of entries ranging between min(k,1024), where k is an implementation specific number, no less than the number of physical interfaces on the router.

The leftmost bit in an RBS RU Bitstrings is RBS-BIFT entry 1.

The type of adjacencies to be supported depend on the encapsulation and are out of scope.

4.3. RBS address creation

Upon creation of the RBS header with an RBS-Address, RU-Length MUST be set to the length of RU0 and RU-offset is set to 0.

4.4. Common RBS processing

Whether a header with an RBS address is created/imposed on the root of an RBS tree or received from another RBS router, encapsulation independent processing of the packet by RBS forwarding is the same.

Parsing RBS-Address, generating copies and rewriting RU-Length and RU-Offset for each copy is formally described in Section 4.7

4.5. Receiving RBS packets

When a packet copy is received with RU-Length=0, the packet is "received" - it is passed up the stack to an appropriate receiving entity based on the encapsulation parameters.

When a packet copy is made for a receive BP, its RU-Length is set to 0 and the packet is processed as if it was received with RU-Length=0.

4.6. Encapsulation considerations

The total length of an RBS address is not included in the definition of an RBS address here. This length is assumed to be provided by

some other packet header field, because it is not required to parse an RBS address itself, but is only required to parse beyond an RBS address in a packet header by an RBS unaware parser. The field that carries RU0 may be larger (for example due to padding) than RU0 itself without problems for the RBS parsing/processing described here.

Additional forwarding rules may be established by specific encapsulations such as BIER OAM processing steps when using BIER with RFC8296 encapsulation.

Given how the processing of the RBS address describes a naturally loop-free rewrite operation, no further loop-prevention mechanism is required in packet processing with RBS addresses, but no harm is done if this is still performed (on appropriate header TTL fields independent of RBS).

4.7. RBS forwarding Pseudocode

The following RBS forwarding (derived from C language) pseudocode assumes all pointers (and de-referencing them) are using bit-accurate addresses so that of calculation of starting bit addresses of address fields and RU in RU0 can be shown with as simple code as if byte addressing for pointers was used. byte addressing of pointers was used. This is NOT supported by C language!

```
void ForwardRBS(Packet)
{
  // parse bit accurate addresses of RBS address fields in Packet into
  // RBS.{RULength,RUOffset,RU0}
  RBS = ParseRBSAddress(Packet);

  if(*(RBS.RULength) == 0) return ReceiveRBS(Packet);
  RU  = RBS.RU0 + *(RBS.RUOffset);
  RUL = *(RBS.RULength);

  BitStringA = RU
  AddressingField =  BitStringA + BIFT.entries;

  // [1] calculate number of R=1 BP set in BitString
  CopyBitString(*BitStringA, *RecursiveBits, BIFT.entries);
  And(*RecursiveBits,*BIFTRecursiveBits, BIFT.entries);
  N = CountBits(*RecursiveBits, BIFT.entries);

  // Start of first RecursiveUnit in RBS address
  // After AddressingField array with 8-bit length fields
  RecursiveUnit = AddressingField + (N - 1) * 8;

  RemainLength = *(RBS.RULength);
  Index = GetFirstBitPosition(*BitStringA);
  while (Index) {
    PacketCopy = Copy(Packet);
    RBSc = ParseRBSAddress(PacketCopy)
    if (BIFT.BP[Index].adjacency == receive)
      *(RBSc.RULength) = 0;
      ReceiveRBS(PacketCopy);
      next;
    }

    If (BIFT.BP[Index].recursive) {
      if(N == 1) {
        RecursiveUnitLength = RemainLength;
      } else {
        RecursiveUnitLength = *AddressingField;
        N--;
        AddressingField += 8;
        RemainLength -= RecursiveUnitLength;
        RemainLength -= 8; // 8 bit of AddressingField
      }
      *(RBSc.RUOffset) = RecursiveUnit - RU0
      *(RBSc.RULength) = RecursiveUnitLength
      RecursiveUnit += RecursiveUnitLength;
    } else {
      *(RBSc.RUOffset) = 0
      *(RBSc.RULength) = 0
      *(MSR6c.SegmentsLeft) -= 1
    }
    SendTo(PacketCopy, BIFT.BP[Index].adjacency)
    Index = GetNextBitPosition(*BitStringA, Index);
  }
}
```

Figure 6: RBS forwarding Pseudocode

Explanations for Figure 6.

ForwardRBS(Packet) processes the RBS address independent of its encapsulation. ParseRBSAddress(Packet) parses the header of Packet to create a list of bit-accurate pointers to the elements of an RBS address: RBS.{RULength,RUOffset,RU0}.

BitStringA is the address of the RBS address BitString in Packet. Other variables use names matching those from the packet header figures (without " -_").

The BFR local BIFT has a total number of BIFT.entries addressable BP 1...BIFTentries. The BitString therefore has BIFT.entries bits.

BIFT.RecursiveBits is a BitString pre-filled by the control plane with all the BP with the recursive flag set. This is constructed from the Recursive flag setting of the BP of the BIFT. The code starting at [1] therefore counts the number of recursive BP in the packets BitString.

Because the AddressingField does not have an entry for the last (potentially only) RecursiveUnit, its length has to be calculated By subtracting the length of the prior N-1 RecursiveUnits from RULength as received. This is done via variable RemainLength.

For every PacketCopy that is to be forwarded, the RU-Length and RU-Offset fields are updated. SendTo(packet,adjacency) takes care of any encapsulation/architecture specific further rewrites of the packet headers based on the adjcency of the BP.

5. Using RBS with BIER and RFC8296 encapsulation

RBS can be used in a BIER domain by introducing as a per-subdomain mode of forwarding, exactly the same as [RFC8279] (non-TE) BIER and [RFC9262] BIER-TE can co-exist in a BIER.

In BIER deployments, RBS can most easily replace BIER-TE, and using a centralized controller and therefore simplify and easier scale deployment of tree engineering. RBS should also be able to replace BIER in networks with link-state routing protocols and reduce the number of replicated packets in large networks. This requires as aforementioned the change from hop-by-hop routing to source-routing.

When using BIER, RBS routers are BFR, RBS ingress routers are BFIR, RBS egress routers are BFER. Routers may support multiple RBS-BIFT through different BIFT-ID or SI. This may be useful when specific constructs such as slices of the network are only allowed to use a subset of the adjacencies of the network.

The RBS address is encoded as a binary string concatenating {RULenth,RUOffset,RU0} into the BitString field in [RFC8296] packet headers. Without changes to [RFC8296], the length of this field has to be a power of 2 sized. The RBS address SHOULD be zero-padded to the size used.

In BIER, the BitStringLength (BSL) expects to indicate different BIFT. When using RBS addresses, it SHOULD be possible for all supported BSL to refer to the same RBS-BIFT, so that upon imposition

of an RBS-Address the smallest power of 2 BitString size can be used without duplication of BIFT state on routers.

SendTo() of RBS forwarding pseudocode Section 4.7 needs to take care of any BIER specific rewrites of the [RFC8296] BIER header, specifically the BIFT-id derived from the RBS-BIFT adjacency.

TBD: This description does not outline, how much of existing BIER IGP extensions could be reused with RBS and how.

6. Using RBS with IPv6 extension header encapsulation

Solutions for stateless IP multicast have been proposed to the IETF under the name Multicast Source Routing for IPv6 (MSR6). They are based on putting the stateless multicast structures into an IPv6 routing extension headers and using per-steering-hop rules according to or derived from [RFC8200], Section 4.4 rules. IPv6 forwarding") for source routing.

SendTo() of RBS forwarding pseudocode Section 4.7 needs to take care of any IPv6 specific rewrites of the IPv6 header (IPv6 Destination address) and IPv6 routing extension header.

For complete support of IPv6 multicast with [RFC8200] compliant source routing, it is necessary for the IPv6 routing extension header to not only carry the RBS address information but also an IPv6 multicast destination address field.

[I-D.eckert-msr6-rbs] is a proposed IPv6 extension header design for MSR6 using RBS that is supporting IPv6 multicast.

7. Security considerations

8. Acknowledgments

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 this work [I-D.xu-msr6-rbs].

9. Changelog

[RFC-editor: please remove]

This document is written in https://github.com/cabo/kramdown-rfc2629 markup language. This documents source is maintained at https://github.com/toerless/multicast-rbs, please provide feedback to the bier@ietf.org and/or msr6@ietf.org mailing list and submit an Issue to the GitHub.

This draft is derived from and supercedes [I-D.eckert-bier-cgm2-rbs] as follows:

* Removes larger architectural context (CGM2) and re-focusses on only RBS.

*Add explanation about possible distributed/decentralized control plane via link-state IGP to complement the central controller based approach.

*Define its procedures independent of specific architectures such as BIER wih RFC8296 encapsulation or proposed MSR encoding.

*Inherits the RBS specific improvements originally introduced with [I-D.eckert-msr6-rbs]. RU-Length and RU-Offset to avoid rewriting complete RBS address with the RU of the next hop and instead just updating these two indices when forwarding RBS address.

*Adds specific proposed encapsulation details for BIER.

10. References

10.1. Normative References

[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>.

[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/info/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/info/rfc8296>.

[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/info/rfc9262>.

10.2. 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>.

[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://

www.ietf.org/archive/id/draft-eckert-bier-cgm2-rbs-01.txt>.

[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-00, 11 July 2022, <https://www.ietf.org/archive/id/draft-eckert-msr6-rbs-00.txt>.

[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://www.ietf.org/archive/id/draft-xu-msr6-rbs-01.txt>.

Appendix A. High-speed implementation considerations

RBS was designed with high-speed, low-cost forwarding hardware and possible backward compatibility with potentially existing flat-bitstring look-up and replication hardware in mind.

Because RBS requires to only perform replication on each router on a single bitstring, it could be possible to reuse existing bitstring replication hardware, or design future hardware such that it supports BIER, BIER-TE and RBS bitstrings.

The calculations required to process an RBS header are the added complexity of processing RBS packets are the additional new cost of RBS. It has to be seen whether / how these are feasible at the low-end of high-speed forwarding plane hardware, especially P4. Further optimizations to reduce calculations are possible, but at the expense of compression of the RBS address.

RBS also minimizes write-cycles to packet memory by only requiring per-packet-copy rewrites of the RU-Length and RU-Offset fields. With mandatory encoding of 12 bits each, these are 24 bits to rewrite and should therefore be causing minimal cost with today's high-speed forwarding hardware.

Appendix B. Complete RBS example

TBD: Need to rewrite more elaborate multi-hop example from [I-D.eckert-bier-cgm2-rbs] with RU-Offset, RU-Length.

Appendix C. Replication efficiency performance considerations

This section discusses in more detail the number of packets required to reach a set of receivers when using flat bitstrings vs. RBS addresses. The first sub-section gives a hopefully simple to understand theoretical topology example, the second sub-section presents initial results of a real-world, large-network simulation.

C.1. Reducing number of duplicate packet copies

If the total size of an RBS encoded delivery tree is larger than a supported maximum RBS header size, then the controller simply needs to divide the tree into multiple subtrees, each only addressing a

part of the BFER (leaves) of the target tree and pruning any unnecessary branches.

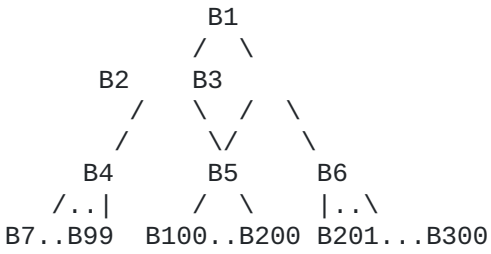


Figure 7: Simple Topology Example

Consider the simple topology in Figure 7 and a multicast packet that needs to reach all BFER B7...B300. Assume that the desired maximum RBM header size is such that a RBS address size of <= 256 bits is desired. The controller could create an RBS address B1=>B2=>B4=>(B7..B99), for a first packet, an RBS address B1=>B3=>B5=>(B100..B200) for a second packet and a third RBS address B1=>B3=>B6=>B201...B300.

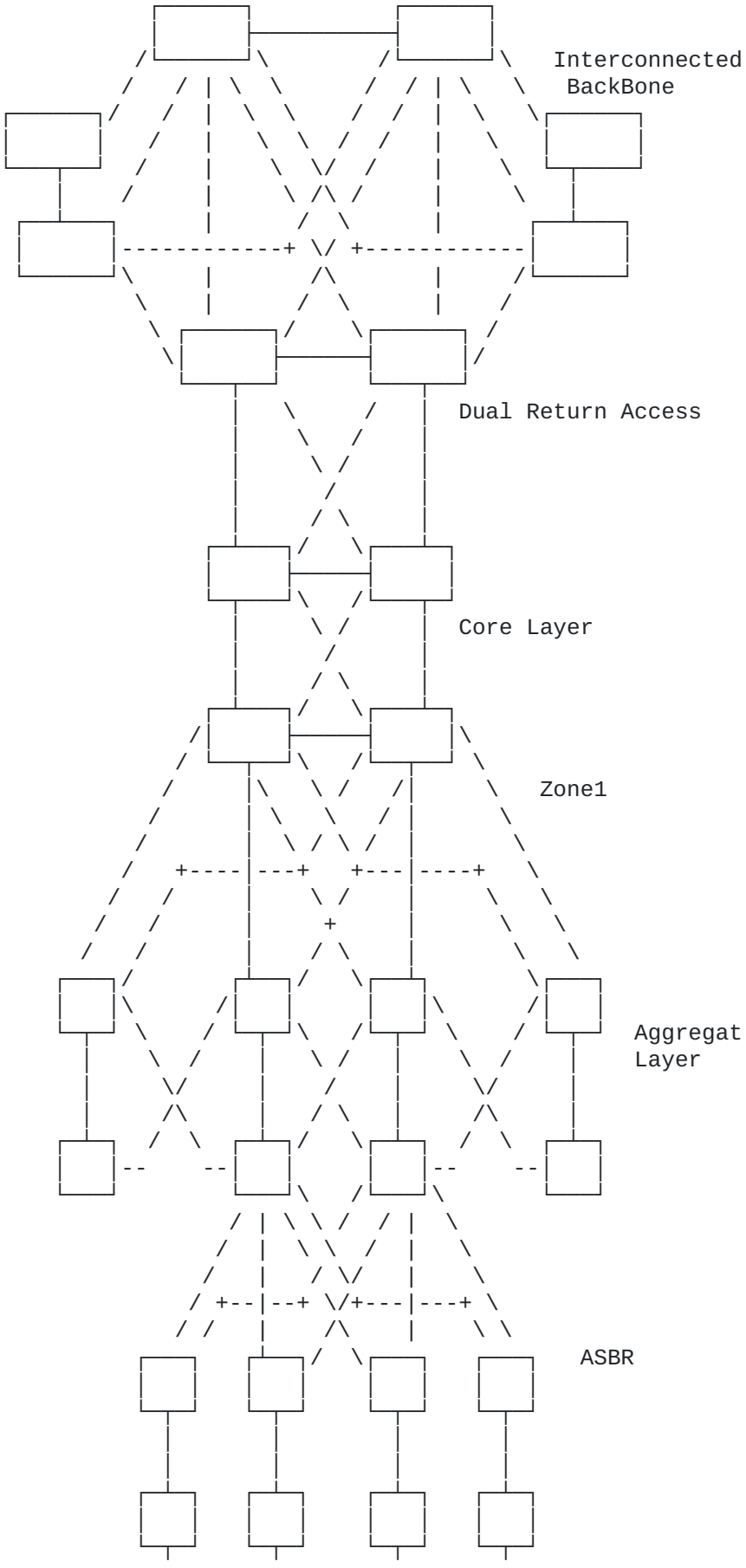
The elimination of larger BIFT state in BFR through multiple SI in BIER/BIER-TE does come at the expense of replicating initial hops of a tree in RBS addresses, such as in the example the encoding of B1=>B3 in the example.

Consider that the assignment of BFIR-ids with BIER in the above example is not carefully engineered. It is then easily possible that the BFR-ids for B7..B99 are not sequentially, but split over a larger BFIR-id space. If the same is true for all BFER, then it is possible that each of the three BFR B4,B5 and B6 has attached BFER from three different SI and one may need to send for example three multiple packets to B7 to address all BFER B7..B99 or to B5 to address all B100..B200 or B6 to address all B201...B300. These unnecessary duplicate packets across B4, B5 or B6 are because of the addressing principle in BIER and are not necessary in RBS, as long as the total length of an RBS address does not require it.

C.2. Statistical Analysis of performance gain

TBD: Comparison of number of packets/header sizes required in large real-world operator topology between BIER/BIER-TE and RBS. Analysis: Gain in dense topology

Topology description: 1. Typical topology of Beijing Mobile in China. 2. All zones dual homing access to backbone. 3. Core layer: 4 nodes full mesh connected 4. Aggregation layer: 8 nodes are divided into two layers, with full interconnection between the layers and dual homing access to the core layer on the upper layer. 5. Aggregation rings: 8 rings, 6 nodes per ring 6. Access rings: 3600 nodes, 18 nodes per ring


Interconnected
BackBone
Dual Return Access
Core Layer
Zone1
Aggregat
Layer
ASBR

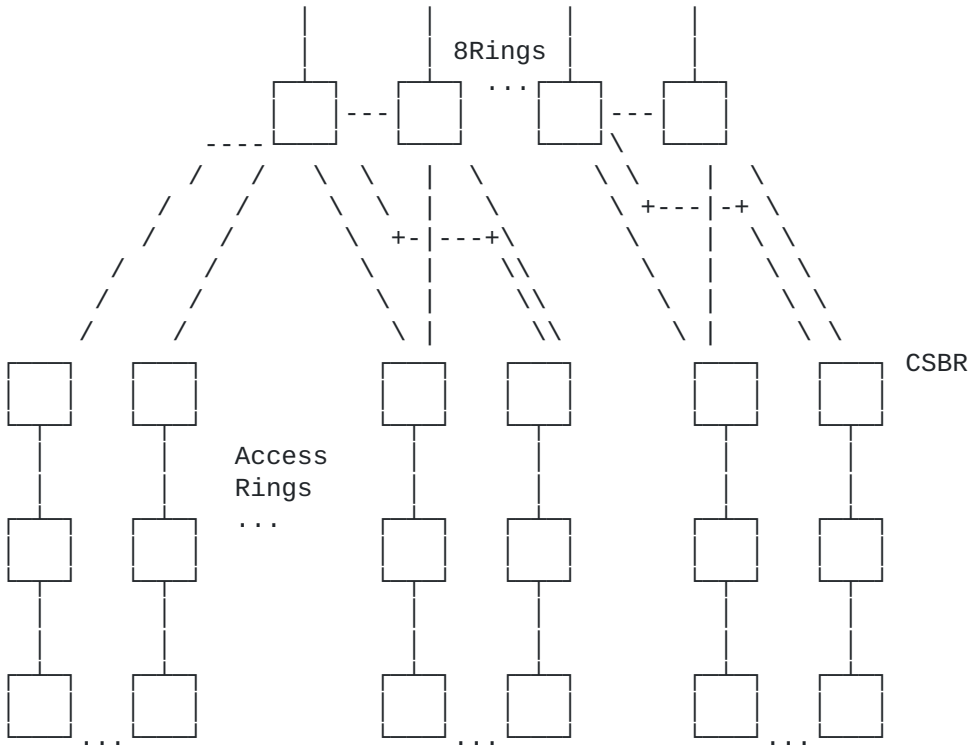

8Rings
...
CSBR
Access
Rings
...

Figure 8: Validation Topology

Comparison notes:

1. RBS: We randomly select egress points as group members, with the total number ranging from 10 to 28800 (for sake of simplicity, we assume merely one client per egress point). The egress points are randomly distributed in the topology with 10 runs for each value, showing the average result in our graphs as below. The total number of samples is 60

2. BIER: We divide the overall topology into 160 BIER domains, each of which includes 180 egress points, providing the total of 28000 egress points.

3. Simulation: In order to compare the BIER against the in-packet tree encoding mechanism, we limit the size of the header to 256 bits (the typical size of a BIER header).

Results are shown in the following image: https://user-images.githubusercontent.com/92767820/153332926-defe38e4-1b63-4b16-852f-feaae487d307.png

Conclusions:

1. BIER reaches its 160 packet replication limit at about 500 users, while the in-packet tree encoding reaching its limit of 125 replications at about 12000 users. And the following decrease of replications is caused by the use of node-local broadcast as a further optimization.

2. For the sake of comparison, the same 256-bit encapsulation limit is imposed on RBS, but we can completely break the 256-bit encapsulation limit, thus allowing the source to send fewer multicast streams.

3. RBS encoding performs significantly better than BIER in that it requires less packet replications and network bandwidth.

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

Xuesong Geng

Huawei 2012 NT Lab
China

Email: gengxuesong@huawei.com

Xiuli Zheng
Huawei 2012 NT Lab
China

Email: zhengxiuli@huawei.com

Rui Meng
Huawei 2012 NT Lab
China

Email: mengrui@huawei.com

Fengkai Li
Huawei 2012 NT Lab

Email: lifengkai@huawei.com