Stream: Internet Engineering Task Force (IETF)
RFC: 0000
Category: Standards Track
Published: June 2022
ISSN: 2070-1721
Authors: T.T.E. Eckert, Ed. *Futurewei* M.M. Menth *University of Tuebingen* G.C. Cauchie *KOEVOO*

RFC 0000
Tree Engineering for Bit Index Explicit Replication (BIER-TE)

Abstract

This memo describes per-packet stateless strict and loose path steered replication and forwarding for "Bit Index Explicit Replication" (BIER) packets (RFC 8279). It is called "BIER Tree Engineering" (BIER-TE) and is intended to be used as the path steering mechanism for Traffic Engineering with BIER.

BIER-TE introduces a new semantic for "bit positions" (BPs). These BPs indicate adjacencies of the network topology, as opposed to (non-TE) BIER in which BPs indicate "Bit-Forwarding Egress Routers" (BFERs). A BIER-TE "packets BitString" therefore indicates the edges of the (loop-free) tree across which the packets are forwarded by BIER-TE. BIER-TE can leverage BIER forwarding engines with little changes. Co-existence of BIER and BIER-TE forwarding in the same domain is possible -- for example, by using separate BIER "subdomains" (SDs). Except for the optional routed adjacencies, BIER-TE does not require a BIER routing underlay and can therefore operate without depending on an "Interior Gateway Routing protocol" (IGP).

Status of This Memo

Copyright Notice

Table of Contents

1. Overview

Tree Engineering for Bit Index Explicit Replication (BIER-TE) is based on the (non-TE) BIER architecture, terminology and packet formats as described in [RFC8279] and [RFC8296]. This document describes BIER-TE, with the expectation that the reader is familiar with these two documents.

BIER-TE introduces a new semantic for "bit positions" (BPs). These BPs indicate adjacencies of the network topology, as opposed to (non-TE) BIER in which BPs indicate "Bit-Forwarding Egress Routers" (BFERs). A BIER-TE "packets BitString" therefore indicates the edges of the (loop-free) tree across which the packets are forwarded by BIER-TE. With BIER-TE, the "Bit Index Forwarding Table" (BIFT) of each "Bit-Forwarding Router" (BFR) is only populated with BPs that are adjacent to the BFR in the BIER-TE Topology. Other BPs are empty in the BIFT. The BFR replicates and forwards BIER packets to adjacent BPs that are set in the packets. BPs are normally also cleared upon forwarding to avoid duplicates and loops.

BIER-TE can leverage BIER forwarding engines with little or no changes. It can also co-exist with BIER forwarding in the same domain, for example, by using separate BIER subdomains. Except for the optional routed adjacencies, BIER-TE does not require a BIER routing underlay and can therefore operate without depending on an "Interior Gateway Routing protocol" (IGP).

This document is structured as follows:

- Section 2 introduces BIER-TE with two forwarding examples, followed by an introduction to the new concepts of the BIER-TE (overlay) topology, and finally a summary of the relationship between BIER and BIER-TE and a discussion of accelerated hardware forwarding.
- Section 3 describes the components of the BIER-TE architecture, Flow overlay, the BIER-TE layer with the BIER-TE control plane (including the BIER-TE controller) and BIER-TE forwarding plane, and the routing underlay.
- Section 4 specifies the behavior of the BIER-TE forwarding plane with the different types of adjacencies and possible variations of BIER-TE forwarding pseudocode, and finally the mandatory and optional procedures.
- Section 5 describes operational considerations for the BIER-TE controller, primarily how the BIER-TE controller can optimize the use of BPs by using specific types of BIER-TE adjacencies for different types of topological situations. It also describes how to assign bits to avoid loops and duplicates (which, in BIER-TE, does not come for free). Finally, it discusses how "Set Identifiers" (SIs), "subdomains" (SDs), and BFR-ids can be managed by a BIER-TE controller; examples and a summary are provided.
- Appendix A concludes the technology-specific sections of this document by further relating BIER-TE to Segment Routing (SR).

Note that related work [CONSTRAINED-CAST] uses Bloom filters [Bloom70] to represent leaves or edges of the intended delivery tree. Bloom filters in general can support larger trees/topologies with fewer addressing bits than explicit BitStrings, but they introduce the heuristic risk of false positives and cannot clear bits in the BitStrings during forwarding to avoid loops. For these reasons, BIER-TE uses explicit BitStrings like BIER. The explicit BitStrings of BIER-TE can also be seen as a special type of Bloom filter, and this is how other related work [ICC] describes it.

2. Introduction

2.1. Requirements Language

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.2. Basic Examples

BIER-TE forwarding is best introduced with simple examples. These examples use formal terms defined later in this document (Table 1 in Section 4.1), including forward_connected(), forward_routed(), and local_decap().

Consider the simple network in the BIER-TE overview example shown in Figure 1, with 6 BFRs. p1...p15 are the bit positions used. All BFRs can act as a Bit-Forwarding Ingress Router (BFIR); BFR1, BFR3, BFR4, and BFR6 can also be BFERs. "Forward_connected()" is the name used for adjacencies that represent subnet adjacencies of the network. "Local_decap()" is the name used for the adjacency that decapsulates BIER-TE packets and passes their payload to higher-layer processing.

```
BIER-TE Topology:

   Diagram:

                     p5    p6
                   --- BFR3 ---
                p3/    p13     \p7          p15
   BFR1 ---- BFR2               BFR5 ----- BFR6
      p1   p2  p4\     p14     /p10 p11   p12
                   --- BFR4 ---
                     p8    p9

   (simplified) BIER-TE Bit Index Forwarding Tables (BIFTs):

   BFR1:   p1  -> local_decap()
           p2  -> forward_connected() to BFR2

   BFR2:   p1  -> forward_connected() to BFR1
           p5  -> forward_connected() to BFR3
           p8  -> forward_connected() to BFR4

   BFR3:   p3  -> forward_connected() to BFR2
           p7  -> forward_connected() to BFR5
           p13 -> local_decap()

   BFR4:   p4  -> forward_connected() to BFR2
           p10 -> forward_connected() to BFR5
           p14 -> local_decap()

   BFR5:   p6  -> forward_connected() to BFR3
           p9  -> forward_connected() to BFR4
           p12 -> forward_connected() to BFR6

   BFR6:   p11 -> forward_connected() to BFR5
           p15 -> local_decap()
```

Figure 1: BIER-TE Basic Example

Assume that a packet from BFR1 should be sent via BFR4 to BFR6. This requires a BitString (p2,p8,p10,p12,p15). When this packet is examined by BIER-TE on BFR1, the only bit position from the BitString that is also set in the BIFT is p2. This will cause BFR1 to send the only copy of the packet to BFR2. Similarly, BFR2 will forward to BFR4 because of p8, BFR4 to BFR5 because of p10, and BFR5 to BFR6 because of p12. p15 finally makes BFR6 receive and decapsulate the packet.

To send a copy to BFR6 via BFR4 and also a copy to BFR3, the BitString needs to be (p2,p5,p8,p10,p12,p13,p15). When this packet is examined by BFR2, p5 causes one copy to be sent to BFR3 and p8 one copy to BFR4. When BFR3 receives the packet, p13 will cause it to receive and decapsulate the packet.

If instead the BitString was (p2,p6,p8,p10,p12,p13,p15), the packet would be copied by BFR5 towards BFR3 because of p6 instead of being copied by BFR2 to BFR3 because of p5 in the prior case. This demonstrates the ability of the BIER-TE Topology, as shown in Figure 1, to make the traffic pass across any possible path and be replicated where desired.

BIER-TE has various options to minimize BP assignments, many of which are based on out-of-band knowledge about the required multicast traffic paths and bandwidth consumption in the network, e.g., from predeployment planning.

Figure 2 shows a modified example, in which Rtr2 and Rtr5 are assumed not to support BIER-TE, so traffic has to be unicast encapsulated across them. To emphasize non-L2, but routed/tunneled forwarding of BIER-TE packets, these adjacencies are called "forward_routed". Otherwise, there is no difference in their processing over the aforementioned forward_connected() adjacencies.

In addition, bits are saved in the following example by assuming that BFR1 only needs to be a BFIR -- not a BFER or a transit BFR.

```
BIER-TE Topology:

   Diagram:

                    p1   p3   p7
                 ....> BFR3 <....       p5
        ........                 ........>
   BFR1       (Rtr2)          (Rtr5)      BFR6
        ........                 ........> p9
                 ....> BFR4 <....       p6
                    p2   p4   p8

   (simplified) BIER-TE Bit Index Forwarding Tables (BIFTs):

   BFR1:   p1  -> forward_routed() to BFR3
           p2  -> forward_routed() to BFR4

   BFR3:   p3  -> local_decap()
           p5  -> forward_routed() to BFR6

   BFR4:   p4  -> local_decap()
           p6  -> forward_routed() to BFR6

   BFR6:   p7  -> forward_routed() to BFR3
           p8  -> forward_routed() to BFR4
           p9  -> local_decap()
```

Figure 2: BIER-TE Basic Overlay Example

To send a BIER-TE packet from BFR1 via BFR3 to be received by BFR6, the BitString is (p1,p5,p9). A packet from BFR1 via BFR4 to be received by BFR6 uses the BitString (p2,p6,p9). A packet from BFR1 to be received by BFR3,BFR4 and from BFR3 to be received by BFR6 uses (p1,p2,p3,p4,p5,p9). A packet from BFR1 to be received by BFR3,BFR4 and from BFR4 to be received by BFR6 uses (p1,p2,p3,p4,p6,p9). A packet from BFR1 to be received by BFR4, and from BFR4 to be received by

BFR6 and from there to be received by BFR3 uses (p2,p3,p4,p6,p7,p9). A packet from BFR1 to be received by BFR3, and from BFR3 to be received by BFR6 there to be received by BFR4 uses (p1,p3,p4,p5,p8,p9).

2.3. BIER-TE Topology and Adjacencies

The key new component in BIER-TE compared to (non-TE) BIER is the BIER-TE topology as introduced through the two examples in Section 2.2. It is used to control where replication can or should happen and how to minimize the required number of BPs for adjacencies.

The BIER-TE Topology consists of the BIFTs of all the BFRs and can also be expressed as a directed graph where the edges are the adjacencies between the BFRs labeled with the BP used for the adjacency. Adjacencies are naturally unidirectional. A BP can be reused across multiple adjacencies as long as this does not lead to undesired duplicates or loops as explained in Section 5.2.

If the BIER-TE topology represents (a subset of) the underlying (Layer 2) topology of the network as shown in the first example, this may be called a "native" BIER-TE topology. A topology consisting only of "forward_routed" adjacencies as shown in the second example may be called an "overlay" BIER-TE topology. A BIER-TE topology with both forward_connected() and forward_routed() adjacencies may be called a "hybrid" BIER-TE topology.

2.4. Relationship to BIER

BIER-TE is designed so that its forwarding plane is a simple extension to the (non-TE) BIER forwarding plane, hence allowing it to be added to BIER deployments where it can be beneficial.

BIER-TE is also intended as an option to expand the BIER architecture into deployments where (non-TE) BIER may not be the best fit, such as statically provisioned networks that need path steering but do not want distributed routing protocols.

1. BIER-TE inherits the following aspects from BIER unchanged:
   a. The fundamental purpose of per-packet signaled replication and delivery via a BitString.
   b. The overall architecture consisting of three layers: flow overlay, BIER(-TE) layer, and routing underlay.
   c. The supported encapsulations [RFC8296].
   d. The semantic of all BIER header elements [RFC8296] used by the BIER-TE forwarding plane other than the semantic of the BP in the BitString.
   e. The BIER forwarding plane, except for how bits have to be cleared during replication.
2. BIER-TE has the following key changes with respect to BIER:
   a. In BIER, bits in the BitString of a BIER packet header indicate a BFER and bits in the BIFT indicate the BIER control plane's calculated next hop toward that BFER. In BIER-TE, a bit in the BitString of a BIER packet header indicates an adjacency in the BIER-TE topology, and only the BFR that is the upstream of that adjacency has its BP populated with the adjacency in its BIFT.

   b. In BIER, the implied reference options for the core part of the BIER layer control plane are the BIER extensions for distributed routing protocols. These include IS-IS and OSPF extensions for BIER, as specified in [RFC8401] and [RFC8444], respectively.
   c. The reference option for the core part of the BIER-TE control plane is the BIER-TE controller. Nevertheless, both the BIER and BIER-TE BIFTs forwarding plane state could equally be populated by any mechanism.
   d. Assuming the reference options for the control plane, BIER-TE replaces in-network autonomous path calculations by explicit paths calculated by the BIER-TE controller.

3. The following elements/functions described in the BIER architecture are not required by the BIER-TE architecture:
   a. "Bit Index Routing Tables" (BIRTs) are not required on BFRs for BIER-TE when using a BIER-TE controller because the controller can directly populate the BIFTs. In BIER, BIRTs are populated by the distributed routing protocol support for BIER, allowing BFRs to populate their BIFTs locally from their BIRTs. Other BIER-TE control plane or management plane options may introduce requirements for BIRTs for BIER-TE BFRs.
   b. The BIER-TE layer forwarding plane does not require BFRs to have a unique BP and therefore also no unique BFR-id. See Section 5.1.3.
   c. Identification of BFRs by the BIER-TE control plane is outside the scope of this specification. Whereas the BIER control plane uses BFR-ids in its BFR-to-BFR signaling, a BIER-TE controller may choose any form of identification deemed appropriate.
   d. BIER-TE forwarding does not require the BFIR-id field of the BIER packet header.

4. Co-existence of BIER and BIER-TE in the same network requires the following:
   a. The BIER/BIER-TE packet header needs to allow the addressing of both BIER and BIER-TE BIFTs. Depending on the encapsulation option, the same SD may or may not be reusable across BIER and BIER-TE. See Section 4.3. In either case, a packet is always only forwarded end to end via BIER or via BIER-TE ("ships in the night" forwarding).
   b. BIER-TE deployments will have to assign BFR-ids to BFRs and insert them into the BFIR-id field of BIER packet headers, as does BIER, whenever the deployment uses (unchanged) components developed for BIER that use BFR-ids, such as multicast flow overlays or BIER layer control plane elements. See also Section 5.3.3.

2.5. Accelerated/Hardware Forwarding Comparison

BIER-TE forwarding rules, especially the parsing of BitStrings, are designed to be as close as possible to those of BIER, with the expectation that this eases the programming of BIER-TE forwarding code and/or BIER-TE forwarding hardware on platforms supporting BIER. The pseudocode in Section 4.4 shows how existing (non-TE) BIER/BIFT forwarding can be modified to support the required BIER-TE forwarding functionality (Section 4.5), by using BIER BIFT's "Forwarding Bit Mask" (F-BM): only the clearing of bits to avoid duplicate packets to a BFR's neighbor is skipped in BIER-TE forwarding because it is not necessary and could not be done when using BIER F-BM.

Whether to use BIER or BIER-TE forwarding is simply a choice of the mode of the BIFT indicated by the packet (BIER or BIER-TE BIFT). This is determined by the BFR configuration for the encapsulation; see Section 4.3.

3. Components

BIER-TE can be thought of as being composed of the same three layers as BIER: The "multicast flow overlay", the "BIER layer", and the "routing underlay". Figure 3 also shows how the "BIER layer" is composed of the "BIER-TE forwarding plane" and the "BIER-TE control plane" as represented by the "BIER-TE Controller".

```
                 <------BGP/PIM----->
    |<-IGMP/PIM->  multicast flow   <-PIM/IGMP->|
                      overlay

        BIER-TE  [BIER-TE Controller] <=> [BIER-TE Topology]
        control     ^      ^     ^
        plane      /       |      \   BIER-TE control protocol
                  |        |       |  (e.g., YANG/NETCONF/RESTCONF
                  |        |       |       PCEP/...)
                  v        v       v
  Src -> Rtr1 -> BFIR-----BFR-----BFER -> Rtr2 -> Rcvr

                 |<----------------->|
               BIER-TE forwarding plane

                 |<- BIER-TE domain->|

               |<--------------------->|
                   Routing underlay
```

Figure 3: BIER-TE Architecture

3.1. The Multicast Flow Overlay

The Multicast Flow Overlay has the same role as that described for BIER in [RFC8279], Section 4.3. See also Section 3.2.1.2.

When a BIER-TE controller is used, the signaling for the Multicast Flow Overlay may also be preferred to operate through a central point of control. For BGP-based overlay flow services such as "Multicast VPN Using Bit Index Explicit Replication (BIER)" [RFC8556], this can be achieved by making the BIER-TE controller operate as a BGP Route Reflector [RFC4456] and combining it with signaling through BGP or a different protocol for the BIER-TE controller's calculated BitStrings. See Sections 3.2.1.2 and 5.3.4.

3.2. The BIER-TE Control Plane

In the (non-TE) BIER architecture [RFC8279], the BIER control plane is not explicitly separated from the BIER forwarding plane, but instead their functions are summarized together in Section 4.2. Example standardized options for the BIER control plane include IS-IS and OSPF extensions for BIER, as specified in [RFC8401] and [RFC8444], respectively.

For BIER-TE, the control plane includes, at a minimum, the following functionality.

BIER-TE topology control: During initial provisioning of the network and/or during modifications of its topology and/or services, the protocols and/or procedures to establish BIER-TE BIFTs:

1. Determine the desired BIER-TE topology for BIER-TE subdomains: the native and/or overlay adjacencies that are assigned to BPs. Topology discovery is discussed in Section 3.2.1.1, and the various aspects of the BIER-TE controller's determinations regarding the topology are discussed throughout Section 5.
2. Determine the per-BFR BIFT from the BIER-TE topology. This is achieved by simply extracting the adjacencies of the BFR from the BIER-TE topology and populating the BFRs BIFT with them.
3. Optionally assign BFR-ids to BFIRs for later insertion into BIER headers on BFIRs as BFIR-ids. Alternatively, BFIR-ids in BIER packet headers may be managed solely by the flow overlay layer and/or be unused. This is discussed in Section 5.3.3.
4. Install/update the BIFTs into the BFRs and, optionally, BFR-ids into BFIRs. This is discussed in Section 3.2.1.1.

BIER-TE tree control: During network operations, protocols and/or procedures to support creation/change/removal of overlay flows on BFIRs:

1. Process the BIER-TE requirements for the multicast overlay flow: BFIRs and BFERs of the flow as well as policies for the path selection of the flow. This is discussed in Section 3.5.
2. Determine the BitStrings and, optionally, Entropy. This is discussed in Sections 3.2.1.2, 3.5, and 5.3.4.
3. Install state on the BFIR to impose the desired BIER packet header(s) for packets of the overlay flow. Different aspects of this, as well as the next point, are discussed throughout Section 3.2.1 and in Section 4.3, but the main responsibility of these two points is with the Multicast Flow Overlay (Section 3.1), which is architecturally inherited from BIER.
4. Install the necessary state on the BFERs to decapsulate the BIER packet header and properly dispatch its payload.

3.2.1. The BIER-TE Controller

This architecture describes the BIER-TE control plane, as shown in Figure 3, as consisting of:

- A BIER-TE controller.

- BFR data models and protocols to communicate between controllers and BFRs in support of BIER-TE topology control (Section 3.2), such as YANG/NETCONF/RESTCONF [RFC7950] [RFC6241] [RFC8040].
- BFR data models and protocols to communicate between controllers and BFIRs in support of BIER-TE tree control (Section 3.2), such as BIER-TE extensions for [RFC5440].

The single, centralized BIER-TE controller is used in this document as the reference option for the BIER-TE control plane, but other options are equally feasible. The BIER-TE control plane could equally be implemented without automated configuration/protocols, by an operator via a CLI on the BFRs. In that case, operator-configured local policy on the BFIR would have to determine how to set the appropriate BIER header fields. The BIER-TE control plane could also be decentralized and/or distributed, but this document does not consider any additional protocols and/or procedures that would then be necessary to coordinate its (distributed/decentralized) entities to achieve the above-described functionality.

3.2.1.1. BIER-TE Topology Discovery and Creation

The first item of BIER-TE topology control includes network topology discovery and BIER-TE topology creation. The latter describes the process by which a Controller determines which routers are to be configured as BFRs and the adjacencies between them.

In statically managed networks, e.g., industrial environments, both discovery and creation can be a manual/offline process.

In other networks, topology discovery may rely on protocols including extending a "Link-State-Protocol" based IGP into the BIER-TE controller itself, [RFC7752] (BGP-LS) or [RFC8345] (YANG topology) as well as BIER-TE-specific methods -- for example, via [BIER-TE-YANG]. These options are non-exhaustive.

Dynamic creation of the BIER-TE topology can be as easy as mapping the network topology 1:1 to the BIER-TE topology by assigning a BP for every network subnet adjacency. In larger networks, it likely involves more complex policy and optimization decisions, including how to minimize the number of BPs required and how to assign BPs across different BitStrings to minimize the number of duplicate packets across links when delivering an overlay flow to BFERs using different SIs/BitStrings. These topics are discussed in Section 5.

When the BIER-TE topology is determined, the BIER-TE Controller will push the BitPositions/adjacencies to the BIFT of the BFRs. On each BFR, only those SI:BitPositions that are adjacencies to other BFRs in the BIER-TE topology are populated.

Communications between the BIER-TE Controller and BFRs for both BIER-TE topology control and BIER-TE tree control are ideally via standardized protocols and data models such as NETCONF/RESTCONF/YANG/PCEP. A vendor-specific CLI on the BFRs is also an option (as in many other Software-Defined Network (SDN) solutions lacking definitions of standardized data models).

3.2.1.2. Engineered Trees via BitStrings

In BIER, the same set of BFERs in a single subdomain is always encoded as the same BitString. In BIER-TE, the BitString used to reach the same set of BFERs in the same subdomain can be different for different overlay flows because the BitString encodes the paths towards the BFERs, so the BitStrings from different BFIRs to the same set of BFERs will often be different. Likewise, the BitString from the same BFIR to the same set of BFERs can be different for different overlay flows for policy reasons such as shortest path trees, Steiner trees (minimum-cost trees), diverse path trees for redundancy, and so on.

See also [BIER-MCAST-OVERLAY] for an application leveraging BIER-TE engineered trees.

3.2.1.3. Changes in the Network Topology

If the network topology changes (not failure based) so that adjacencies that are assigned to bit positions are no longer needed, the BIER-TE Controller can reuse those bit positions for new adjacencies. First, these bit positions need to be removed from any BFIR flow state and BFR BIFT state. Then, they can be repopulated, first into BIFT and then into the BFIR.

3.2.1.4. Link/Node Failures and Recovery

When links or nodes fail or recover in the topology, BIER-TE could quickly respond with Fast Reroute (FRR) procedures such as those described in [BIER-TE-PROTECTION], the details of which are out of scope for this document. It can also more slowly react by recalculating the BitStrings of affected multicast flows. This reaction is slower than the FRR procedure because the BIER-TE Controller needs to receive link/node up/down indications, recalculate the desired BitStrings, and push them down into the BFIRs. With FRR, this is all performed locally on a BFR receiving the adjacency up/down notification.

3.3. The BIER-TE Forwarding Plane

The BIER-TE forwarding plane consists of the following components:

1. On a BFIR, imposition of the BIER header for packets from overlay flows. This is driven by state established by the BIER-TE control plane, the multicast flow overlay as explained in Section 3.1, or a combination of both.
2. On BFRs (including BFIRs and BFERs), forwarding/replication of BIER packets according to their SD, SI, "BitStringLength" (BSL), BitString, and, optionally, Entropy fields as explained in Section 4. Processing of other BIER header fields, such as the Differentiated Services Code Point (DSCP) field, is outside the scope of this document.
3. On BFERs, removal of the BIER header and dispatching of the payload according to state created by the BIER-TE control plane and/or overlay layer.

When the BIER-TE forwarding plane receives a packet, it simply looks up the bit positions that are set in the BitString of the packet in the BIFT that was populated by the BIER-TE Controller. For every BP that is set in the BitString and has one or more adjacencies in the BIFT, a copy is made according to the type of adjacency for that BP in the BIFT. Before sending any copies, the BFR clears all BPs in the BitString of the packet for which the BFR has one or more adjacencies in the

BIFT. Clearing these bits inhibits the looping of packets when the BitStrings erroneously include a forwarding loop. When a forward_connected() adjacency has the "DoNotClear" (DNC) flag set, this BP is reset for the packet copied to that adjacency. See Section 4.2.1.

3.4. The Routing Underlay

For forward_connected() adjacencies, BIER-TE is sending BIER packets to directly connected BIER-TE neighbors as L2 (unicast) BIER packets without requiring a routing underlay. For forward_routed() adjacencies, BIER-TE forwarding encapsulates a copy of the BIER packet so that it can be delivered by the forwarding plane of the routing underlay to the routable destination address indicated in the adjacency. See Section 4.2.2 for the adjacency definition.

BIER relies on the routing underlay to calculate paths towards BFERs and derive next-hop BFR adjacencies for those paths. This commonly relies on BIER-specific extensions to the routing protocols of the routing underlay but may also be established by a controller. In BIER-TE, the next hops for a packet are determined by the BitString through the BIER-TE Controller-established adjacencies on the BFR for the BPs of the BitString. There is thus no need for BFR-specific routing underlay extensions to forward BIER packets with BIER-TE semantics.

Encapsulation parameters can be provisioned by the BIER-TE controller into the forward_connected() or forward_routed() adjacencies directly without relying on a routing underlay.

If the BFR intends to support FRR for BIER-TE, then the BIER-TE forwarding plane needs to receive fast adjacency up/down notifications: link up/down or neighbor up/down, e.g., from Bidirectional Forwarding Detection (BFD). Providing these notifications is considered to be part of the routing underlay in this document.

3.5. Traffic Engineering Considerations

Traffic Engineering [TE-RFC3272bis] provides performance optimization of operational IP networks while utilizing network resources economically and reliably. The key elements needed to effect TE are policy, path steering, and resource management. These elements require support at the control/controller level and within the forwarding plane.

Policy decisions are made within the BIER-TE control plane, i.e., within BIER-TE Controllers. Controllers use policy when composing BitStrings and BFR BIFT state. The mapping of user/IP traffic to specific BitStrings/BIER-TE flows is made based on policy. The specific details of BIER-TE policies and how a controller uses them are out of scope for this document.

Path steering is supported via the definition of a BitString. BitStrings used in BIER-TE are composed based on policy and resource management considerations. For example, when composing BIER-TE BitStrings, a Controller must take into account the resources available at each BFR and for each BP when it is providing congestion-loss-free services such as Rate-Controlled Service Disciplines [RCSD94]. Resource availability could be provided for example via routing protocol information, but may also be obtained via a BIER-TE control protocol such as NETCONF or any other protocol commonly used by a Controller to understand the resources of the network it operates on. The resource usage of the BIER-TE traffic admitted by the BIER-TE controller can be solely tracked on the BIER-TE Controller based on local accounting as long as no forward_routed() adjacencies are used (see Section 4.2.1 for the definition of forward_routed() adjacencies). When forward_routed() adjacencies are used, the paths selected by the underlying routing protocol need to be tracked as well.

Resource management has implications on the forwarding plane beyond the BIER-TE defined steering of packets. This includes allocation of buffers to guarantee the worst-case requirements of admitted RCSD traffic and potentially policing and/or rate-shaping mechanisms, typically done via various forms of queuing. This level of resource control, while optional, is important in networks that wish to support congestion management policies to control or regulate the offered traffic to deliver different levels of service and alleviate congestion problems, or those networks that wish to control latencies experienced by specific traffic flows.

4. BIER-TE Forwarding

4.1. The BIER-TE Bit Index Forwarding Table (BIFT)

The BIER-TE BIFT is the equivalent of the BIER BIFT for (non-TE) BIER. It exists on every BFR running BIER-TE. For every BIER subdomain (SD) in use for BIER-TE, it is a table, as shown in Table 1. That example BIFT assumes a BSL of 8 bit positions (BPs) in the packets BitString. As in [RFC8279], this BSL is purely used for the example and not a BIER/BIER-TE supported BSL (minimum BSL is 64).

A BIER-TE BIFT compares to a BIER BIFT as shown in [RFC8279] as follows.

In both BIER and BIER-TE, BIFT rows/entries are indexed in their respective BIER pseudocode ([RFC8279], Section 6.5) and BIER-TE pseudocode (Section 4.4) by the BIFT-index derived from the packets SI, BSL and the one bit position of the packets BitString (BP) addressing the BIFT row: BIFT-index = SI * BSL + BP - 1. BP within a BitString are numbered from 1 to BSL, hence the - 1 offset when converting to a BIFT-index. This document also uses the notion SI:BP to indicate BIFT rows, [RFC8279] uses the equivalent notion SI:BitString, where the BitString is filled with only the BP for the BIFT row.

In BIER, each BIFT-index addresses one BFER by its BFR-id = BIFT-index + 1 and is populated on each BFR with the next-hop "BFR Neighbor" (BFR-NBR) towards that BFER.

In BIER-TE, each BIFT-index, and therefore SI:BP, indicates one or more adjacencies between BFRs in the topology and is only populated with those adjacencies forwarding entries on the BFR that is the upstream for these adjacencies. The BIFT entry are empty on all other BFRs.

In BIER, each BIFT row also requires a "Forwarding Bit Mask" (F-BM) entry for BIER forwarding rules. In BIER-TE forwarding, F-BM is not required, but can be used when implementing BIER-TE on forwarding hardware derived from BIER forwarding, that must use F-BM. This is discussed in the first BIER-TE forwarding pseudocode in Section 4.4.

BIFT-index (SI:BP)	(F-BM)	Adjacencies: <empty> or one or more per entry
	BIFT indices for Packets with SI=0	
0 (0:1)	...	forward_connected(interface,neighbor{,DNC})
1 (0:2)	...	forward_connected(interface,neighbor{,DNC})
	...	forward_connected(interface,neighbor{,DNC})
...	...	...
4 (0:5)	...	local_decap({VRF})
5 (0:6)	...	forward_routed({VRF,}l3-neighbor)
6 (0:7)	...	<empty>
7 (0:8)	...	ECMP((adjacency1,...adjacencyN){,seed})
	BIFT indices for BitString/Packet with SI=1	
9 (1:1) ...	 ...	... ...

Table 1: BIER-TE Bit Index Forwarding Table (BIFT) with Different Adjacencies

The BIFT is configured for the BIER-TE data plane of a BFR by the BIER-TE Controller through an appropriate protocol and data model. The BIFT is then used to forward packets, according to the rules specified in the BIER-TE Forwarding Procedures.

Note that a BIFT index (SI:BP) may be populated in the BIFT of more than one BFR to save BPs. See Section 5.1.6 for an example of how a BIER-TE controller could assign BPs to (logical) adjacencies shared across multiple BFRs, Section 5.1.3 for an example of assigning the same BP to different adjacencies, and Section 5.1.9 for general guidelines regarding the reuse of BPs across different adjacencies.

{VRF} indicates the Virtual Routing and Forwarding context into which the BIER payload is to be delivered. This is optional and depends on the multicast flow overlay.

4.2. Adjacency Types

4.2.1. Forward Connected

A "forward_connected()" adjacency is towards a directly connected BFR neighbor using an interface address of that BFR on the connecting interface. A forward_connected() adjacency does not route packets; only L2 forwards them to the neighbor.

Packets sent to an adjacency with "DoNotClear" (DNC) set in the BIFT **MUST NOT** have the bit position for that adjacency cleared when the BFR creates a copy for it. The bit position will still be cleared for copies of a packet made towards other adjacencies. This can be used for example in ring topologies as explained in Section 5.1.6.

For protection against loops from misconfiguration (see Section 5.2.1), DNC is only permissible for forward_connected() adjacencies. No need or benefit of DNC for other types of adjacencies was identified and their risk was not analyzed.

4.2.2. Forward Routed

A "forward_routed()" adjacency is an adjacency towards a BFR that uses a (tunneling) encapsulation that will cause a packet to be forwarded by the routing underlay toward the adjacent BFR. This can leverage any feasible encapsulation, such as MPLS or tunneling over IP/IPv6, as long as the BIER-TE packet can be identified as a payload. This identification can either rely on the BIER/BIER-TE co-existence mechanisms described in Section 4.3, or by explicit support for a BIER-TE payload type in the tunneling encapsulation.

forward_routed() adjacencies are necessary to pass BIER-TE traffic across routers that are not BIER-TE capable or to minimize the number of required BPs by tunneling over (BIER-TE capable) routers on which neither replication nor path steering is desired, or simply to leverage path redundancy and FRR of the routing underlay towards the next BFR. They may also be useful to a multi-subnet adjacent BFR to leverage the routing underlay ECMP independently of BIER-TE ECMP (Section 4.2.3).

4.2.3. ECMP

(non-TE) BIER ECMP is tied to the BIER BIFT processing semantic and is therefore not directly usable with BIER-TE.

A BIER-TE "Equal-Cost Multipath" (ECMP()) adjacency as shown in Table 1 for BIFT-index 7 has a list of two or more non-ECMP adjacencies as parameters and an optional seed parameter. When a BIER-TE packet is copied onto such an ECMP() adjacency, an implementation-specific so-called hash function will select one out of the list's adjacencies to which the packet is forwarded. If the packet's encapsulation contains an entropy field, the entropy field **SHOULD** be respected; two packets with the same value of the entropy field **SHOULD** be sent on the same adjacency. The seed parameter permits the design of hash functions that are easy to implement at high speed without running into polarization issues across multiple consecutive ECMP hops. See Section 5.1.7 for more explanations.

4.2.4. Local Decap(sulation)

A "local_decap()" adjacency passes a copy of the payload of the BIER-TE packet to the protocol ("NextProto") within the BFR (IPv4/IPv6, Ethernet,...) responsible for that payload according to the packet header fields. A local_decap() adjacency turns the BFR into a BFER for matching packets. Local_decap() adjacencies require the BFER to support routing or switching for NextProto to determine how to further process the packets.

4.3. Encapsulation / Co-existence with BIER

Specifications for BIER-TE encapsulation are outside the scope of this document. This section gives explanations and guidelines.

Like [RFC8279], handling of "Maximum Transmission Unit" (MTU) limitations is outside the scope of this document and instead part of the BIER-TE packet encapsulation and/or flow overlay. See for example [RFC8296], Section 3. It applies equally to BIER-TE as it does to BIER.

Because a BFR needs to interpret the BitString of a BIER-TE packet differently from a (non-TE) BIER packet, it is necessary to distinguish BIER packets from BIER-TE packets. In the BIER encapsulation [RFC8296], the BIFT-id field of the packet indicates the BIFT of the packet. BIER and BIER-TE can therefore be run simultaneously, when the BIFT-id address space is shared across a BIER BIFT and BIER-TE BIFT. Partitioning the BIFT-id address space is subject to BIER-TE / BIER control plane procedures.

When [RFC8296] is used for BIER with MPLS, BIFT-id address ranges can be dynamically allocated from MPLS label space only for the set of actually used SD:BSL BIFT. This also permits the allocation of non-overlapping label ranges for BIFT-ids that are to be used with BIER-TE BIFTs.

With MPLS, it is also possible to reuse the same SD space for both BIER-TE and BIER, so that the same SD has both a BIER BIFT with a corresponding range of BIFT-ids and disjoint BIER-TE BIFTs with a non-overlapping range of BIFT-ids.

When a fixed mapping from BSL, SD, and SI to a BIFT-id is used, which does not explicitly partition the BIFT-id space between BIER and BIER-TE -- for example, as proposed for non-MPLS forwarding with BIER encapsulation [RFC8296] in [NON-MPLS-BIER-ENCODING], Section 5 -- it is necessary to allocate disjoint SDs to BIER and BIER-TE BIFTs so that both can be addressed by the BIFT-ids. The encoding proposed in Section 6 of [NON-MPLS-BIER-ENCODING] does not statically encode BSL or SD into the BIFT-id, but the encoding permits a mapping and hence could provide the same freedom as when MPLS is being used (same or different SD for BIER/BIER-TE).

forward_routed() requires an encapsulation that permits directing unicast encapsulated BIER-TE packets to a specific interface address on a target BFR. With MPLS encapsulation, this can simply be done via a label stack with that addresses label as the top label -- followed by the label assigned to the (BSL,SD,SI) BitString. With non-MPLS encapsulation, some form of IP encapsulation would be required (for example IP/GRE).

The encapsulation used for forward_routed() adjacencies can equally support existing advanced adjacency information such as "loose source routes" via e.g. MPLS label stacks or appropriate header extensions (e.g., for IPv6).

4.4. BIER-TE Forwarding Pseudocode

The following pseudocode (Figure 4) for BIER-TE forwarding is based on the (non-TE) BIER forwarding pseudocode of [RFC8279], Section 6.5 with one modification.

```
    void ForwardBitMaskPacket_withTE (Packet)
    {
        SI=GetPacketSI(Packet);
        Offset=SI*BitStringLength;
        for (Index = GetFirstBitPosition(Packet->BitString); Index ;
             Index = GetNextBitPosition(Packet->BitString, Index)) {
            F-BM = BIFT[Index+Offset]->F-BM;
            if (!F-BM) continue;                                [3]
            BFR-NBR = BIFT[Index+Offset]->BFR-NBR;
            PacketCopy = Copy(Packet);
            PacketCopy->BitString &= F-BM;                      [2]
            PacketSend(PacketCopy, BFR-NBR);
            // The following must not be done for BIER-TE:
            // Packet->BitString &= ~F-BM;                      [1]
        }
    }
```

Figure 4: BIER-TE Forwarding Pseudocode for Required Functions, Based on BIER Pseudocode

In step [2], the F-BM is used to clear bit(s) in PacketCopy. This step exists in both BIER and BIER-TE, but the F-BMs need to be populated differently for BIER-TE than for BIER for the desired clearing.

In BIER, multiple bits of a BitString can have the same BFR-NBR. When a received packets BitString has more than one of those bits set, the BIER replication logic has to avoid that more than one PacketCopy is sent to that BFR-NBR ([1]). Likewise, the PacketCopy sent to a BFR-NBR must clear all bits in its BitString that are not routed across a BFR-NBR. This protects against BIER replication on any possible further BFR to create duplicates ([2]).

To solve both [1] and [2] for BIER, the F-BM of each bit index needs to have all bits set that this BFR wants to route across a BFR-NBR. [2] clears all other bits in PacketCopy->BitString, and [1] clears those bits from Packet->BitString after the first PacketCopy.

In BIER-TE, a BFR-NBR in this pseudocode is an adjacency, forward_connected(), forward_routed() or local_decap(). There is no need for [2] to suppress duplicates in the way BIER does because in general, different BP would never have the same adjacency. If a BIER-TE controller actually finds some optimization in which this would be desirable, then the controller is also responsible to ensure that only one of those bits is set in any Packet->BitString, unless the controller explicitly wants for duplicates to be created.

The following points describe how the forwarding bit mask (F-BM) for each BP is configured in the BIFT and how this impacts the BitString of the packet being processed with that BIFT:

1. The F-BMs of all BIFT BPs without an adjacency have all their bits clear. This will cause [3] to skip further processing of such a BP.
2. All BIFT BPs with an adjacency (with DNC flag clear) have an F-BM that has only those BPs set for which this BFR does not have an adjacency. This causes [2] to clear all bits from PacketCopy->BitString for which this BFR does have an adjacency.
3. [1] is not performed for BIER-TE. All bit clearing required by BIER-TE is performed by [2].

This Forwarding Pseudocode can support the required BIER-TE forwarding functions (see Section 4.5), forward_connected(), forward_routed() and local_decap(), but not the recommended functions DNC flag and multiple adjacencies per bit nor the optional function, ECMP() adjacencies. The DNC flag cannot be supported when using only [1] to mask bits.

The modified and expanded Forwarding Pseudocode in Figure 5 specifies how to support all BIER-TE forwarding functions (required, recommended and optional):

- This pseudocode eliminates per-bit F-BM, therefore reducing the size of BIFT state by BSL^2*SI and eliminating the need for per-packet-copy BitString masking operations except for adjacencies with the DNC flag set:
  - AdjacentBits[SI] are bit positions with a non-empty list of adjacencies in this BFR BIFT. This can be computed whenever the BIER-TE Controller updates (adds/removes) adjacencies in the BIFT.
  - The BFR needs to create packet copies for these adjacent bits when they are set in the packets BitString. This set of bits is calculated in PktAdjacentBits.
  - All bit positions to which the BFR creates copies have to be cleared in packet copies to avoid loops. This is done by masking the BitString of the packet with ~AdjacentBits[SI]. When an adjacency has DNC set, this bit position is set again only for the packet copy towards that bit position.
- BIFT entries may contain more than one adjacency in support of specific configurations such as a hub and multiple spokes (Section 5.1.5). The code therefore includes a loop over these adjacencies.
- The ECMP() adjacency is also shown in the figure. Its parameters are a seed and a ListOfAdjacencies from which one is picked.
- The forward_connected(), forward_routed(), local_decap() adjacencies are shown with their parameters.

```
void ForwardBitMaskPacket_withTE (Packet)
{
    SI = GetPacketSI(Packet);
    Offset = SI * BitStringLength;
    // Determine adjacent bits in the packets BitString
    PktAdjacentBits = Packet->BitString & AdjacentBits[SI];

    // Clear adjacent bits in Packet header to avoid loops
    Packet->BitString &= ~AdjacentBits[SI];

    // Loop over PktAdjacentBits to create packet copies
    for (Index = GetFirstBitPosition(PktAdjacentBits); Index ;
         Index = GetNextBitPosition(PktAdjacentBits, Index)) {
        for adjacency in BIFT[Index+Offset]->Adjacencies {
            if(adjacency.type == ECMP(ListOfAdjacencies,seed) ) {
                I = ECMP_hash(sizeof(ListOfAdjacencies),
                              Packet->Entropy,seed);
                adjacency = ListOfAdjacencies[I];
            }
            PacketCopy = Copy(Packet);
            switch(adjacency.type) {
                case forward_connected(interface,neighbor,DNC):
                    if(DNC)
                        PacketCopy->BitString |= 1<<(Index-1);
                    SendToL2Unicast(PacketCopy,interface,neighbor);

                case forward_routed({VRF,}l3-neighbor):
                    SendToL3(PacketCopy,{VRF,}l3-neighbor);

                case local_decap({VRF},neighbor):
                    DecapBierHeader(PacketCopy);
                    PassTo(PacketCopy,{VRF,}Packet->NextProto);
            }
        }
    }
}
```

Figure 5: Complete BIER-TE Forwarding Pseudocode for Required, Recommended, and Optional Functions

4.5. BFR Requirements for BIER-TE Forwarding

BFRs that support BIER-TE and BIER **MUST** support configuration that enables BIER-TE instead of (non-TE) BIER forwarding rules for all BIFTs of one or more BIER subdomains. Every BP in a BIER-TE BIFT **MUST** support to have zero or one adjacency. BIER-TE forwarding **MUST** support the adjacency types forward_connected() with the DNC flag not set, forward_routed() and local_decap(). As explained in Section 4.4, these required BIER-TE forwarding functions can be implemented via the same Forwarding Pseudocode as BIER forwarding except for one modification (skipping one masking with F-BM).

BIER-TE forwarding **SHOULD** support forward_connected() adjacencies with a set DNC flag, as this is highly useful to save bits in rings (see Section 5.1.6).

BIER-TE forwarding **SHOULD** support more than one adjacency on a bit. This allows bits to be saved in hub-and-spoke scenarios (see Section 5.1.5).

BIER-TE forwarding **MAY** support ECMP() adjacencies to save bits in ECMP scenarios; see Section 5.1.7 for an example. This is an optional procedure, because for ECMP deployments using BIER-TE one can also leverage ECMP of the routing underlay via forwarded_routed adjacencies and/or might prefer to have more explicit control of the path chosen via explicit BP/adjacencies for each ECMP path alternative.

5. BIER-TE Controller Operational Considerations

5.1. Bit Position Assignments

This section describes how the BIER-TE Controller can use the different BIER-TE adjacency types to define the bit positions of a BIER-TE domain.

Because the size of the BitString limits the size of the BIER-TE domain, many of the options described exist to support larger topologies with fewer bit positions.

5.1.1. P2P Links

On a P2P link that connects two BFRs, the same bit position can be used on both BFRs for the adjacency to the neighboring BFR. A P2P link therefore requires only one bit position.

5.1.2. BFERs

Every non-leaf BFER is given a unique bit position with a local_decap() adjacency.

5.1.3. Leaf BFERs

A leaf BFER is one where incoming BIER-TE packets never need to be forwarded to another BFR but are only sent to the BFER to exit the BIER-TE domain. For example, in networks where Provider Edge (PE) router are spokes connected to Provider (P) routers, those PEs are Leaf BFERs unless there is a U-turn between two PEs.

Consider how redundant disjoint traffic can reach BFER1/BFER2 as shown in Figure 6: when BFER1/BFER2 are non-leaf BFERs as shown on the right-hand side, one traffic copy would be forwarded to BFER1 from BFR1, but the other one could only reach BFER1 via BFER2, which makes BFER2 a non-leaf BFER. Likewise, BFER1 is a non-leaf BFER when forwarding traffic to BFER2. Note that the BFERs on the left-hand side of the figure are only guaranteed to be leaf-BFERs by fitting routing configuration that prohibits transit traffic to pass through the BFERs, which is commonly applied in these topologies.

```
        BFR1(P) BFR2(P)              BFR1(P)  BFR2(P)
          |  \ /  |                     |        |
          |   X   |                     |        |
          |  / \  |                     |        |
     BFER1(PE)  BFER2(PE)        BFER1(PE)----BFER2(PE)

                                          ^ U-turn link

         Leaf BFER /                Non-Leaf BFER /
          PE-router                   PE-router
```

Figure 6: Leaf vs. Non-Leaf BFER Example

In most situations, leaf-BFERs that are to be addressed via the same BitString can share a single bit position for their local_decap() adjacency in that BitString and therefore save bit positions. On a non-leaf BFER, a received BIER-TE packet may only need to transit the BFER or it may need to also be decapsulated. Whether or not to decapsulate the packet therefore needs to be indicated by a unique bit position populated only on the BIFT of this BFER with a local_decap() adjacency. On a leaf-BFER, packets never need to pass through; any packet received is therefore usually intended to be decapsulated. This can be expressed by a single, shared bit position that is populated with a local_decap() adjacency on all leaf-BFERs addressed by the BitString.

The possible exception from this leaf-BFER bit position optimization can be cases where the bit position on the prior BIER-TE BFR (which created the packet copy for the leaf-BFER in question) is populated with multiple adjacencies as an optimization -- for example, as described in Sections 5.1.4 and 5.1.5. With either of these two optimizations, the sender of the packet could only control explicitly whether the packet was to be decapsulated on the leaf-BFER in question, if the leaf-BFER has a unique bit position for its local_decap() adjacency.

However, if the bit position is shared across a leaf-BFER and packets are therefore decapsulated potentially unnecessarily, this may still be appropriate if the decapsulated payload of the BIER-TE packet indicates whether or not the packets need to be further processed/received. This is typically true, for example, if the payload is IP multicast because IP multicast on a BFER would know the membership state of the IP multicast payload and be able to discard it if the packets were delivered unnecessarily by the BIER-TE layer. If the payload has no such membership indication and the BFIR wants to have explicit control about which BFERs are to receive and decapsulate a packet, then these two optimizations cannot be used together with shared bit positions optimization for a leaf-BFER.

5.1.4. LANs

In a LAN, the adjacency to each neighboring BFR is given a unique bit position. The adjacency of this bit position is a forward_connected() adjacency towards the BFR and this bit position is populated into the BIFT of all the other BFRs on that LAN.

```
                     BFR1
                       |p1
                LAN1-+-+---+-----+
                       p3|   p4|    p2|
                       BFR3  BFR4   BFR7
```

Figure 7: LAN Example

If bandwidth on the LAN is not an issue and most BIER-TE traffic should be copied to all neighbors on a LAN, then bit positions can be saved by assigning just a single bit position to the LAN and populating the bit position of the BIFTs of each BFR on the LAN with a list of forward_connected() adjacencies to all other neighbors on the LAN.

This optimization does not work in the case of BFRs redundantly connected to more than one LAN with this optimization because these BFRs would receive duplicates and forward those duplicates into the opposite LANs. Adjacencies of such BFRs into their LAN still need a separate bit position.

5.1.5. Hub and Spoke

In a setup with a hub and multiple spokes connected via separate p2p links to the hub, all p2p adjacencies from the hub to the spoke's links can share the same bit position. The bit position on the hub's BIFT is set up with a list of forward_connected() adjacencies, one for each spoke.

This option is similar to the bit position optimization in LANs: redundantly connected spokes need their own bit positions, unless they are themselves Leaf-BFER.

This type of optimized BP could be used for example when all traffic is "broadcast" traffic (very dense receiver set) such as live-TV or many-to-many telemetry, including situation-awareness (SA). This BP optimization can then be used to explicitly steer different traffic flows across different ECMP paths in data-center or broadband-aggregation networks with minimal use of BPs.

5.1.6. Rings

In L3 rings, instead of assigning a single bit position for every p2p link in the ring, it is possible to save bit positions by setting the "DoNotClear" (DNC) flag on forward_connected() adjacencies.

For the rings shown in Figure 8, a single bit position will suffice to forward traffic entering the ring at BFRa or BFRb all the way up to BFR1:

On BFRa, BFRb, BFR30,... BFR3, the bit position is populated with a forward_connected() adjacency pointing to the clockwise neighbor on the ring and with DNC set. On BFR2, the adjacency also points to the clockwise neighbor BFR1, but without DNC set.

Handling DNC this way ensures that copies forwarded from any BFR in the ring to a BFR outside the ring will not have the ring bit position set, therefore minimizing the chance to create loops.

```
                 v         v
                 |         |
        L1       |   L2    |    L3
    /-------- BFRa ---- BFRb --------------------\
    |                                            |
    \- BFR1 - BFR2 - BFR3 - ... - BFR29 - BFR30 -/
        |       |    L4              |       |
     p33|                         p15|
        BFRd                          BFRc
```

Figure 8: Ring Example

Note that this example only permits packets intended to make it all the way around the ring to enter it at BFRa and BFRb. Note also that packets will always travel clockwise. If packets should be allowed to enter the ring at any ring BFR, then one would have to use two ring bit positions, one for each direction: clockwise and counterclockwise.

Both would be set up to stop rotating on the same link, e.g., L1. When the ring BFIR creates the clockwise copy, it will clear the counterclockwise bit position because the DNC bit only applies to the bit for which the replication is done (likewise for the clockwise bit position for the counterclockwise copy). As a result, the ring BFIR will send a copy in both directions, serving BFRs on either side of the ring up to L1.

5.1.7. Equal-Cost Multipath (ECMP)

An ECMP() adjacency allows the use of just one BP to deliver packets to one of N adjacencies instead of one BP for each adjacency. In the common example case (Figure 9), a link-bundle of three links L1,L2,L3 connects BFR1 and BFR2, and only one BP is used instead of three BPs to deliver packets from BFR1 to BFR2.

```
             --L1-----
        BFR1 --L2----- BFR2
             --L3-----

   BIFT entry in BFR1:
   ------------------------------------------------------------------
   | Index |  Adjacencies                                           |
   ==================================================================
   | 0:6   |  ECMP({forward_connected(L1, BFR2),                    |
   |       |        forward_connected(L2, BFR2),                    |
   |       |        forward_connected(L3, BFR2)}, seed)             |
   ------------------------------------------------------------------

   BIFT entry in BFR2:
   ------------------------------------------------------------------
   | Index |  Adjacencies                                           |
   ==================================================================
   | 0:6   |  ECMP({forward_connected(L1, BFR1),                    |
   |       |        forward_connected(L2, BFR1),                    |
   |       |        forward_connected(L3, BFR1)}, seed)             |
   ------------------------------------------------------------------
```

Figure 9: ECMP Example

This document does not standardize any ECMP algorithm because it is sufficient for implementations to document their freely chosen ECMP algorithm. Figure 10 shows an example ECMP algorithm and would double as its documentation: A BIER-TE controller could determine which adjacency is chosen based on the seed and adjacencies parameters and the packet entropy.

```
    forward(packet, ECMP(adj(0), adj(1),... adj(N-1), seed)):
       i = (packet(bier-header-entropy) XOR seed) % N
       forward packet to adj(i)
```

Figure 10: ECMP Algorithm Example

In the example shown in Figure 11, all traffic from BFR1 towards BFR10 is intended to be ECMP load split equally across the topology. This example is not meant as a likely setup; rather, it illustrates that ECMP can be used to share BPs not only across link bundles but also across alternative paths across different transit BFRs, and it explains the use of the seed parameter.

```
                BFR1          (BFIR)
              /L11  \L12
             /        \
         BFR2          BFR3
        /L21 \L22     /L31 \L32
       /       \     /       \
     BFR4   BFR5   BFR6   BFR7
       \       /     \       /
        \     /       \     /
         BFR8          BFR9
             \        /
              \      /
               BFR10         (BFER)

BIFT entry in BFR1:
------------------------------------------------------------------
| 0:6   |  ECMP({forward_connected(L11, BFR2),                    |
|       |        forward_connected(L12, BFR3)}, seed1)            |
------------------------------------------------------------------

BIFT entry in BFR2:
------------------------------------------------------------------
| 0:7   |  ECMP({forward_connected(L21, BFR4),                    |
|       |        forward_connected(L22, BFR5)}, seed1)            |
------------------------------------------------------------------

BIFT entry in BFR3:
------------------------------------------------------------------
| 0:7   |  ECMP({forward_connected(L31, BFR6),                    |
|       |        forward_connected(L32, BFR7)}, seed1)            |
------------------------------------------------------------------

BIFT entry in BFR4, BFR5:
------------------------------------------------------------------
| 0:8   |  forward_connected(Lxx, BFR8)  |xx differs on BFR4/BFR5|
------------------------------------------------------------------

BIFT entry in BFR6, BFR7:
------------------------------------------------------------------
| 0:8   |  forward_connected(Lxx, BFR9)  |xx differs on BFR6/BFR7|
------------------------------------------------------------------

BIFT entry in BFR8, BFR9:
------------------------------------------------------------------
| 0:9   |  forward_connected(Lxx, BFR10) |xx differs on BFR8/BFR9|
------------------------------------------------------------------
```

Figure 11: Polarization Example

Note that for the following discussion of ECMP, only the BIFT ECMP adjacencies on BFR1, BFR2, and BFR3 are relevant. The reuse of BPs across BFRs in this example is further explained in Section 5.1.9 below.

With the ECMP setup shown in the topology above, traffic would not be equally load-split. Instead, links L22 and L31 would see no traffic at all: BFR2 will only see traffic from BFR1 for which the ECMP hash in BFR1 selected the first adjacency in the list of 2 adjacencies given as parameters to the ECMP. It is link L11-to-BFR2. BFR2 again performs ECMP with two adjacencies on that subset of traffic using the same seed1, and will therefore again select the first of its two adjacencies: L21-to-BFR4; therefore, L22 and BFR5 see no traffic (likewise for L31 and BFR6).

This issue in BFR2/BFR3 is called "polarization". It results from the reuse of the same hash function across multiple consecutive hops in topologies like these. To resolve this issue, the ECMP() adjacency on BFR1 can be set up with a different seed2 than the ECMP() adjacencies on BFR2/BFR3. BFR2/BFR3 can use the same hash because packets will not sequentially pass across both of them. Therefore, they can also use the same BP 0:7.

Note that ECMP solutions outside of BIER often hide the seed by auto-selecting it from local entropy such as unique local or next-hop identifiers. Allowing the BIER-TE Controller to explicitly set the seed gives the ability for it to control same/different path selection across multiple consecutive ECMP hops.

5.1.8. Forward Routed Adjacencies

5.1.8.1. Reducing Bit Positions

Forward_routed() adjacencies can reduce the number of bit positions required when the path steering requirement is not hop-by-hop explicit path selection, but loose-hop selection. Forward_routed() adjacencies can also permit BIER-TE operation across intermediate-hop routers that do not support BIER-TE.

Assume that the requirement in Figure 12 is to explicitly steer traffic flows that have arrived at BFR1 or BFR4 via a path in the routing underlay "Network Area 1" to one of the following next three segments: (1) BFR2 via link L1, (2) BFR2 via link L2, or (3) via BFR3 and then not caring whether the packet is forwarded via L3 or L4.

```
                 ...............
      ...BFR1--...             ...--L1-- BFR2...
                ... .Routers. ...--L2--/
      ...BFR4--...             ...--L3-- BFR3...
                ...            ...--L4--/ |
                 ...............          |
                                         LO
                  Network Area 1
```

Figure 12: Forward Routed Adjacencies Example

To enable this, both BFR1 and BFR4 are set up with a forward_routed adjacency bit position towards an address of BFR2 on link L1, another forward_routed() bit position towards an address of BFR2 on link L2 and a third forward_routed() bit position towards a node address LO of BFR3.

5.1.8.2. Supporting Nodes without BIER-TE

Forward_routed() adjacencies also enable incremental deployment of BIER-TE. Only the nodes through which BIER-TE traffic needs to be steered -- with or without replication -- need to support BIER-TE. Where they are not directly connected to each other, forward_routed adjacencies are used to pass over nodes that are not BIER-TE enabled.

5.1.9. Reuse of Bit Positions (without DNC)

BPs can be reused across multiple BFRs to minimize the number of BPs needed. This happens when adjacencies on multiple BFRs use the DNC flag as described above, but it can also be done for non-DNC adjacencies. This section only discusses this non-DNC case.

Because a BP is cleared when passing a BFR with an adjacency for that BP, reusing a BP across multiple BFRs does not introduce any problems with duplicates or loops that do not also exist when every adjacency has a unique BP. Instead, the challenge when reusing a BP is whether it still allows the achievement of the desired Tree Engineering goals.

A BP cannot be reused across two BFRs that would need to be passed sequentially for some path: The first BFR will clear the BP, so those paths cannot be built. A BP can be set across BFRs that would occur (A) only across different paths or (B) across different branches of the same tree.

An example of (A) was given in Figure 11, where BP 0:7, BP 0:8, and BP 0:9 are each reused across multiple BFRs because a single packet/path would never be able to reach more than one BFR sharing the same BP.

Assume that the example was changed: BFR1 has no ECMP() adjacency for BP 0:6, but instead BP 0:5 with forward_connected() to BFR2 and BP 0:6 with forward_connected() to BFR3. Packets with both BP 0:5 and BP 0:6 would now be able to reach both BFR2 and BFR3, and the still-existing reuse of BP 0:7 between BFR2 and BFR3 is a case of (B) where the reuse of a BP is perfect because it does not limit the set of useful path choices:

If instead of reusing BP 0:7 BFR3 used a separate BP 0:10 for its ECMP() adjacency, no useful additional path steering options would be enabled. If duplicates at BFR10 were undesirable, this would be done by not setting BP 0:5 and BP 0:6 for the same packet. If the duplicates were desirable (e.g., resilient transmission), the additional BP 0:10 would also not render additional value.

Reuse may also save BPs in larger topologies. Consider the topology shown in Figure 13:

```
                     area1
                  BFR1a  BFR1b
                    /      \
         .....................................
         .                 Core              .
         .....................................
         |    /        \      /          \    |
       BFR2a BFR2b   BFR3a BFR3b       BFR6a BFR6b
        /-------\    /---------\       /--------\
        | area2 |    |  area3  | ...   | area6  |
        | ring  |    |  ring   |       | ring   |
        \-------/    \---------/       \--------/
        more BFRs    more BFRs          more BFRs
```

Figure 13: Reuse of BPs

A BFIR/sender (e.g., video headend) is attached to area 1, and area 2...6 contain receivers/BFERs. Assume that each area had a distribution ring, each with two BPs to indicate the direction (as explained before). These two BPs could be reused across the 5 areas. Packets would be replicated through other BPs for the Core to the desired subset of areas, and once a packet copy reaches the ring of the area, the two ring BPs come into play. This reuse is a case of (B), but it limits the topology choices: Packets can only flow around the same direction in the rings of all areas. This may or may not be acceptable based on the desired path steering options: if resilient transmission is the path engineering goal, then it is likely a good optimization; if the bandwidth of each ring was to be optimized separately, it would not be a good limitation.

5.1.10. Summary of BP Optimizations

This section reviewed a range of techniques by which a BIER-TE Controller can create a BIER-TE topology in a way that minimizes the number of necessary BPs.

Without any optimization, a BIER-TE Controller would attempt to map the network subnet topology 1:1 into the BIER-TE topology and every subnet adjacent neighbor requires a forward_connected() BP and every BFER requires a local_decap() BP.

The optimizations described are then as follows:

- P2P links require only one BP (Section 5.1.1).
- All leaf-BFERs can share a single local_decap() BP (Section 5.1.3).
- A LAN with N BFR needs at most N BP (one for each BFR). It only needs one BP for all those BFRs that are not redundantly connected to multiple LANs (Section 5.1.4).
- A hub with p2p connections to multiple non-leaf-BFER spokes can share one BP to all spokes if traffic can be flooded to all spokes, e.g., because of no bandwidth concerns or dense receiver sets (Section 5.1.5).
- Rings of BFRs can be built with just two BP (one for each direction) except for BFR with multiple ring connections -- similar to LANs (Section 5.1.6).
- ECMP() adjacencies to N neighbors can replace N BP with 1 BP. Multihop ECMP can avoid polarization through different seeds of the ECMP algorithm (Section 5.1.7).

- Forward_routed() adjacencies permit "tunneling" across routers that are either BIER-TE capable or not BIER-TE capable where no traffic steering or replications are required (Section 5.1.8).
- A BP can generally be reused across a set of nodes where it can be guaranteed that no path will ever need to traverse more than one node of the set. Depending on the scenario, this may limit the feasible path steering options (Section 5.1.9).

Note that the described list of optimizations is not exhaustive. Further optimizations of BPs are possible, especially when both the set of required path steering choices and the set of possible subsets of BFERs that should be able to receive traffic are limited. The hub-and-spoke optimization is a simple example of such traffic-pattern-dependent optimizations.

5.2. Avoiding Duplicates and Loops

5.2.1. Loops

Whenever BIER-TE creates a copy of a packet, the BitString of that copy will have all bit positions cleared that are associated with adjacencies on the BFR. This inhibits the looping of packets. The only exceptions are adjacencies with DNC set.

With DNC set, looping can happen. Consider in Figure 14 that link L4 from BFR3 is (inadvertently) plugged into the L1 interface of BFRa (instead of BFR2). This creates a loop where the ring's clockwise bit position is never cleared for copies of the packets traveling clockwise around the ring.

```
               v         v
               |         |
        L1     |   L2    |  L3
    /-------- BFRa ---- BFRb ----------------------\
    |        .                                      |
    |         ......  Wrong link wiring             |
    |               .                               |
    \- BFR1 - BFR2   BFR3 - ... - BFR29 - BFR30 -/
        |      |    L4            |       |
     p33|                      p15|
        BFRd                       BFRc
```

Figure 14: Miswired Ring Example

To inhibit looping in the face of such physical misconfiguration, only forward_connected() adjacencies are permitted to have DNC set, and the link layer port unique unicast destination address of the adjacency (e.g. MAC address) protects against closing the loop. Link layers without port unique link layer addresses should not be used with the DNC flag set.

5.2.2. Duplicates

Duplicates happen when the graph expressed by a BitString is not a tree but is redundantly connecting BFRs with each other. In Figure 15, a BitString of p2,p3,p4,p5 would result in duplicate packets arriving on BFER4. The BIER-TE Controller must therefore ensure that only BitStrings that are trees are created.

```
                  BFIR1
                 /     \
                / p2    \ p3
               BFR2    BFR3
                \ p4    / p5
                 \     /
                  BFER4
```

Figure 15: Duplicates Example

When links are incorrectly physically reconnected before the BIER-TE Controller updates BitStrings in BFIRs, duplicates can happen. Like loops, these can be inhibited by link layer addressing in forward_connected() adjacencies.

If interface or loopback addresses used in forward_routed() adjacencies are moved from one BFR to another, duplicates can equally happen. Such readdressing operations must be coordinated with the BIER-TE Controller.

5.3. Managing SIs, Subdomains, and BFR-ids

When the number of bits required to represent the necessary hops in the topology and BFER exceeds the supported BitStringLength (BSL), multiple SIs and/or subdomains must be used. This section discusses how this is accomplished.

BIER-TE forwarding does not require the concept of BFR-ids, but routing underlay, flow overlay, and BIER headers may. This section also discusses how BFR-ids can be assigned to BFIRs/BFERs for BIER-TE.

5.3.1. Why SIs and Subdomains?

For (non-TE) BIER and BIER-TE forwarding, the most important result of using multiple SIs and/or subdomains is the same: packets that need to be sent to BFERs in different SIs or subdomains require different BIER packets, each one with a BitString for a different (SI,subdomain) combination. Each such BitString uses one BSL sized SI block in the BIFT of the subdomain. We call this a BIFT:SI (block).

For BIER and BIER-TE forwarding, there is also no difference whether different SIs and/or subdomains are chosen, but SIs and subdomains have different purposes in the BIER architecture shared by BIER-TE. This impacts how operators are managing them and how flow overlays in particular will likely use them.

By default, every possible BFIR/BFER in a BIER network would likely be given a BFR-id in subdomain 0 (unless there are > 64k BFIRs/BFERs).

If there are different flow services (or service instances) requiring replication to different subsets of BFERs, then it will likely not be possible to achieve the best replication efficiency for all of these service instances via subdomain 0. Ideal replication efficiency for N BFER exists in a subdomain if they are split over not more than ceiling(N/BitStringLength) SI.

If service instances justify additional BIER:SI state in the network, additional subdomains will be used: BFIRs/BFERs are assigned BFR-ids in those subdomains and each service instance is configured to use the most appropriate subdomain. This results in improved replication efficiency for different services.

Even if creation of subdomains and assignment of BFR-ids to BFIRs/BFERs in those subdomains is automated, it is not expected that individual service instances can deal with BFERs in different subdomains. A service instance may only support configuration of a single subdomain it should rely on.

To be able to easily reuse (and modify as little as possible) existing BIER procedures, including flow overlay and routing underlay, when BIER-TE forwarding is added, we therefore reuse SIs and subdomains logically in the same way as they are used in BIER: all necessary BFIRs/BFERs for a service use a single BIER-TE BIFT and are split across as many SIs as necessary (see Section 5.3.2). Different services may use different subdomains that primarily exist to provide more efficient replication (and, for BIER-TE, desirable path steering) for different subsets of BFIRs/BFERs.

5.3.2. Assigning Bits for the BIER-TE Topology

In BIER, BitStrings only need to carry bits for BFERs, which leads to the model that BFR-ids map 1:1 to each bit in a BitString.

In BIER-TE, BitStrings need to carry bits to indicate not only the receiving BFER but also the intermediate hops/links across which the packet must be sent. The maximum number of BFERs that can be supported in a single BitString or BIFT:SI depends on the number of bits necessary to represent the desired topology between them.

"Desired" topology means that it depends on the physical topology and the operator's desire to permit explicit path steering across every single hop (which requires more bits), or reducing the number of required bits by exploiting optimizations such as unicast (forward_routed()), ECMP(), or flood (DNC) over "uninteresting" sub-parts of the topology, e.g., parts where different trees do not need to take different paths due to path steering reasons.

The total number of bits to describe the topology vs. the number of BFERs in a BIFT:SI can range widely based on the size of the topology and the amount of alternative paths in it. In a BIER-TE topology crafted by a BIER-TE expert, the higher the percentage of non-BFER bits, the higher the likelihood that those topology bits are not just BIER-TE overhead without additional benefit but instead will allow the expression of desirable path steering alternatives.

5.3.3. Assigning BFR-ids with BIER-TE

BIER-TE forwarding does not use the BFR-id, nor does it require that the BFIR-id field of the BIER header be set to a particular value. However, other parts of a BIER-TE deployment may need a BFR-id, specifically multicast flow overlay signaling and multicast flow overlay packet disposition, and in that case BFRs need to also have BFR-ids for BIER-TE SDs.

For example, for BIER overlay signaling, BFIRs need to have a BFR-id, because this BFIR BFR-id is carried in the BFIR-id field of the BIER header to indicate to the overlay signaling on the receiving BFER which BFIR originated the packet.

In BIER, BFR-id = SI * BSL + BP, such that the SI and BP of a BFER can be calculated from the BFR-id and vice versa. This also means that every BFR with a BFR-id has a reserved BP in an SI, even if that is not necessary for BIER forwarding, because the BFR may never be a BFER but only a BFIR.

In BIER-TE, for a non-leaf BFER, there is usually a single BP for that BFER with a local_decap() adjacency on the BFER. The BFR-id for such a BFER can therefore be determined using the same procedure as that used for (non-TE) BIER: BFR-id = SI * BSL + BP.

As explained in Section 5.1.3, leaf BFERs do not need such a unique local_decap() adjacency. Likewise, BFIRs that are not also BFERs may not have a unique local_decap() adjacency either. For all those BFIRs and (leaf) BFERs, the controller needs to determine unique BFR-ids that do not collide with the BFR-ids derived from the non-leaf BFER local_decap() BPs.

While this document defines no requirements on how to allocate such BFR-id, a simple option is to derive it from the (SI,BP) of an adjacency that is unique to the BFR in question. For a BFIR, this can be the first adjacency only populated on this BFIR, for a leaf-BFER, this could be the first BP with an adjacency towards that BFER.

5.3.4. Mapping from BFRs to BitStrings with BIER-TE

In BIER, applications of the flow overlay on a BFIR can calculate the (SI,BP) of a BFER from the BFR-id of the BFER and can therefore easily determine the BitStrings for a BIER packet to a set of BFERs with known BFR-ids.

In BIER-TE, this mapping needs to be equally supported for flow overlays. This section outlines two core options, based on what type of Tree Engineering the BIER-TE controller needs to perform for a particular application.

"Independent branches": For a given flow overlay instance, the branches from a BFIR to every BFER are calculated by the BIER-TE controller to be independent of the branches to any other BFER. Shortest path trees are the most common examples of trees with independent branches.

"Interdependent branches": When a BFER is added or deleted from a particular distribution tree, the BIER-TE controller has to recalculate the branches to other BFER, because they may need to change. Steiner trees are examples of interdependent branch trees.

If "independent branches" are used, the BIER-TE Controller can signal to the BFIR flow overlay for every BFER an SI:BitString that represents the branch to that BFER. The flow overlay on the BIFR can then, independently of the controller, calculate the SI:BitString for all desired BFERs by OR'ing their BitStrings. This allows flow overlay applications to operate independently of the controller whenever it needs to determine which subset of BFERs needs to receive a particular packet.

If "interdependent branches" are required, the application would need to query the SI:BitString for a given set of BFERs whenever the set changes.

Note that in either case (unlike the scenario for BIER), the bits may need to change upon link/node failure/recovery, network expansion and network resource consumption by other traffic as part of Traffic Engineering goals (e.g., reoptimization of lower-priority traffic flows). Interactions between such BFIR applications and the BIER-TE Controller do therefore need to support dynamic updates to the SI:BitStrings.

Communications between the BFIR flow overlay and the BIER-TE controller require some way to identify the BFER. If BFR-ids are used in the deployment, as outlined in Section 5.3.3, then those are the natural BFR identifiers. If BFR-ids are not used, then any other unique identifier, such as the BFR-prefix of the BFR [RFC8279], could be used.

5.3.5. Assigning BFR-ids for BIER-TE

It is not currently determined if a single subdomain could or should be allowed to forward both (non-TE) BIER and BIER-TE packets. If this should be supported, there are two options:

A. BIER and BIER-TE have different BFR-ids in the same subdomain. This allows higher replication efficiency for BIER because their BFR-ids can be assigned sequentially, while the BitStrings for BIER-TE will also have the additional bits for the topology. There is no relationship between a BFR BIER BFR-id and its BIER-TE BFR-id.
B. BIER and BIER-TE share the same BFR-id. The BFR-ids are assigned as explained above for BIER-TE and simply reused for BIER. The replication efficiency for BIER will be as low as that for BIER-TE in this approach.

5.3.6. Example Bit Allocations

5.3.6.1. With BIER

Consider a network setup with a BSL of 256 for a network topology as shown in Figure 16. The network has 6 areas, each with 170 BFERs, connecting via a core with 4 (core) BFRs. To address all BFERs with BIER, 4 SIs are required. To send a BIER packet to all BFERs in the network, 4 copies need to be sent by the BFIR. On the BFIR, it does not matter how the BFR-ids are allocated to BFERs in the network, but for efficiency further down in the network it does matter.

```
            area1           area2        area3
           BFR1a BFR1b  BFR2a BFR2b   BFR3a BFR3b
             |   \         /     \        /   |
             ................................
             .                Core           .
             ................................
             |     /       \     /        \   |
           BFR4a BFR4b  BFR5a BFR5b   BFR6a BFR6b
            area4          area5        area6
```

Figure 16: Scaling BIER-TE Bits by Reuse

With random allocation of BFR-ids to BFERs, each receiving area would (most likely) have to receive all 4 copies of the BIER packet because there would be BFR-ids for each of the 4 SIs in each of the areas. Only further towards each BFER would this duplication subside -- when each of the 4 trees runs out of branches.

If BFR-ids are allocated intelligently, then all the BFERs in an area would be given BFR-ids with as few different SIs as possible. Each area would only have to forward one or two packets instead of four.

Given how networks can grow over time, replication efficiency in an area will then also go down over time when BFR-ids are only allocated sequentially, network wide. An area that initially only has BFR-id in one SI might end up with many SIs over a longer period of growth. Allocating SIs to areas with initially sufficiently many spare bits for growths can help to alleviate this issue. Or renumber BFERs after network expansion. In this example one may consider to use 6 SIs and assign one to each area.

This example shows that intelligent BFR-id allocation within at least subdomain 0 can even be helpful or even necessary in BIER.

5.3.6.2. With BIER-TE

In BIER-TE one needs to determine a subset of the physical topology and attached BFERs so that the "desired" representation of this topology and the BFER fit into a single BitString. This process needs to be repeated until the whole topology is covered.

Once bits/SIs are assigned to topology and BFERs, BFR-ids are just a derived set of identifiers from the operator/BIER-TE Controller as explained above.

Every time that different subtopologies have overlap, bits need to be repeated across the BitStrings, increasing the overall amount of bits required across all BitStrings/SIs. In the worst case, one assigns random subsets of BFERs to different SIs. This will result in an outcome much worse than in (non-TE) BIER: it maximizes the amount of unnecessary topology overlap across SIs and therefore reduces the number of BFERs that can be reached across each individual SI. Intelligent BFER-to-SI assignment and selecting specific "desired" subtopologies can minimize this problem.

To set up BIER-TE efficiently for the topology of Figure 16, the following bit allocation method can be used. This method can easily be expanded to other, similarly structured larger topologies.

Each area is allocated one or more SIs, depending on the number of future expected BFERs and the number of bits required for the topology in the area. In this example, 6 SIs are used, one per area.

In addition, we use 4 bits in each SI:

bia: (b)it (i)ngress (a)

bib: (b)it (i)ngress (b)

bea: (b)it (e)gress (a)

beb: (b)it (e)gress (b)

These bits will be used to pass BIER packets from any BFIR via any combination of an ingress area a/b BFR and egress area a/b BFR into a specific target area. These bits are then set up with the right forward_routed() adjacencies on the BFIR and area edge BFR:

On all BFIRs in an area, j|j=1...6, bia in each BIFT:SI is populated with the same forward_routed(BFRja) and bib with forward_routed(BFRjb). On all area edge BFRs, bea in BIFT:SI=k|k=1...6 is populated with forward_routed(BFRka) and beb in BIFT:SI=k with forward_routed(BFRkb).

For BIER-TE forwarding of a packet to a subset of BFERs across all areas, a BFIR would create at most 6 copies, with SI=1...SI=6. In each packet, the bits indicate bits for topology and BFER in that topology plus the four bits to indicate whether to pass this packet via the ingress area a or b border BFR and the egress area a or b border BFR, therefore allowing path steering for those two "unicast" legs: 1) BFIR to ingress area edge and 2) core to egress area edge. Replication only happens inside the egress areas. For BFERs in the same area as in the BFIR, these four bits are not used.

5.3.7. Summary

BIER-TE can, like BIER, support multiple SIs within a subdomain. This allows application of the mapping BFR-id = SI * BSL + BP. This also permits the reuse of the BIER architecture concept of BFR-ids and, therefore, minimization of BIER-TE-specific functions in possible BIER layer control plane mechanisms with BIER-TE, including flow overlay methods and BIER header fields.

The number of BFIRs/BFERs possible in a subdomain is smaller than in BIER because BIER-TE uses additional bits for topology.

Subdomains (SDs) in BIER-TE can be used as they are in BIER to create more efficient replication to known subsets of BFERs.

Assigning bits for BFERs intelligently into the right SI is more important in BIER-TE than in BIER because of replication efficiency and the overall amount of bits required.

6. Security Considerations

If [RFC8296] is used, BIER-TE shares its security considerations.

BIER-TE shares the security considerations of BIER, [RFC8279], with the following overriding or additional considerations.

BIER-TE forwarding explicitly supports unicast "tunneling" of BIER packets via forward_routed() adjacencies. The BIER domain security model is based on a subset of interfaces on a BFR that connect to other BFRs of the same BIER domain. For BIER-TE, this security model equally applies to such unicast "tunneled" BIER packets. This does not only include the need to filter received unicast "tunneled" BIER packets to prohibit the injection of such "tunneled" BIER packets from outside the BIER domain, but also prohibiting forward_routed() adjacencies to leak BIER packets from the BIER domain. It **SHOULD** be possible to configure interfaces to be part of a BIER domain solely for sending and receiving of unicast "tunneled" BIER packets even if the interface cannot send/receive BIER encapsulated packets.

In BIER, the standardized methods for the routing underlays are IGPs with extensions to distribute BFR-ids and BFR-prefixes. [RFC8401] specifies the extensions for IS-IS, and [RFC8444] specifies the extensions for OSPF. Attacking the protocols for the BIER routing underlay or (non-TE) BIER layer control plane, or impairment of any BFR in a domain may lead to successful attacks against the results of the routing protocol, enabling DoS attacks against paths or the addressing (BFR-id, BFR-prefixes) used by BIER.

The reference model for the BIER-TE layer control plane is a BIER-TE controller. When such a controller is used, impairment of an individual BFR in a domain causes no impairment of the BIER-TE control plane on other BFRs. If a routing protocol is used to support forward_routed() adjacencies, then this is still an attack vector as in BIER, but only for BIER-TE forward_routed() adjacencies and not other adjacencies.

Whereas IGP routing protocols are most often not well secured through cryptographic authentication and confidentiality, communications between controllers and routers such as those to be considered for the BIER-TE controller / control plane can be, and are, much more commonly secured with those security properties -- for example, by using Secure Shell (SSH) [RFC4253] for NETCONF [RFC6242]; or via Transport Layer Security (TLS), such as [RFC8253] for PCEP [RFC5440] or [RFC7589] for NETCONF. BIER-TE controllers **SHOULD** use security equal to or better than these mechanisms.

When any of these security mechanisms/protocols are used for communications between a BIER-TE controller and BFRs, their security considerations apply to BIER-TE. In addition, the security considerations of PCE, [RFC4655] apply.

The most important attack vector in BIER-TE is misconfiguration, either on the BFRs themselves or via the BIER-TE controller. Forwarding entries with DNC could be set up to create persistent loops, in which packets only expire because of TTL. To minimize the impact of such attacks (or, more likely, unintentional misconfiguration by operators and/or bad BIER-TE controller software), the BIER-TE forwarding rules are defined to be as strict in clearing bits as possible. The clearing of all bits with an adjacency on a BFR prohibits a looping packet from creating additional packet amplification through the misconfigured loop on the packet's second time or subsequent times around the loop, because all relevant adjacency bits would have been cleared on the first round through the loop. As a result, BIER-TE has the same degree of looping packets as possible with unintentional or malicious loops in the routing underlay with BIER or even with unicast traffic.

Deployments where BIER-TE would likely be beneficial may include operational models where actual configuration changes from the controller are only required during non-production phases of the network's life-cycle, e.g., in embedded networks or manufacturing networks during, for example, plant reworking/repairs. In these types of deployments, configuration changes could be locked out when the network is in production state and could only be (re-)enabled through reverting the network/installation into non-production state. Such security designs would not only allow a deployment to provide additional layers of protection against configuration attacks but would, first and foremost, protect the active production process from such configuration attacks.

7. IANA Considerations

This document requests no action by IANA.

8. References

8.1. Normative References

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[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>.

8.2. Informative References

[BIER-MCAST-OVERLAY]

Trossen, D., Rahman, A., Wang, C., and T. Eckert, "Applicability of BIER Multicast Overlay for Adaptive Streaming Services", Work in Progress, Internet-Draft, draft-ietf-bier-multicast-http-response-06, 10 July 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-bier-multicast-http-response-06>.

[BIER-TE-PROTECTION] Eckert, T., Cauchie, G., Braun, W., and M. Menth, "Protection Methods for BIER-TE", Work in Progress, Internet-Draft, draft-eckert-bier-te-frr-03, 5 March 2018, <https://datatracker.ietf.org/doc/html/draft-eckert-bier-te-frr-03>.

[BIER-TE-YANG] Zhang, Z., Wang, C., Chen, R., Hu, F., Sivakumar, M., and H. Chen, "A YANG data model for Tree Engineering for Bit Index Explicit Replication (BIER-TE)", Work in Progress, Internet-Draft, draft-ietf-bier-te-yang-05, 1 May 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-bier-te-yang-05>.

[Bloom70] Bloom, B. H., "Space/time trade-offs in hash coding with allowable errors", Comm. ACM 13(7):422-6, July 1970, <https://dl.acm.org/doi/10.1145/362686.362692>.

[CONSTRAINED-CAST] Bergmann, O., Bormann, C., Gerdes, S., and H. Chen, "Constrained-Cast: Source-Routed Multicast for RPL", Work in Progress, Internet-Draft, draft-ietf-roll-ccast-01, 30 October 2017, <https://datatracker.ietf.org/doc/html/draft-ietf-roll-ccast-01>.

[ICC] Reed, M. J., Al-Naday, M., Thomos, N., Trossen, D., Petropoulos, G., and S. Spirou, "Stateless multicast switching in software defined networks", IEEE International Conference on Communications (ICC), Kuala Lumpur, Malaysia, May 2016, <https://ieeexplore.ieee.org/document/7511036>.

[NON-MPLS-BIER-ENCODING] Wijnands, IJ., Mishra, M., Xu, X., and H. Bidgoli, "An Optional Encoding of the BIFT-id Field in the non-MPLS BIER Encapsulation", Work in Progress, Internet-Draft, draft-ietf-bier-non-mpls-bift-encoding-04, 30 May 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-bier-non-mpls-bift-encoding-04>.

[RCSD94] Zhang, H. and D. Ferrari, "Rate-Controlled Service Disciplines", Journal of High-Speed Networks, Volume 3, Issue 4, pp. 389-412, October 1994, <https://dl.acm.org/doi/10.5555/2692227.2692232>.

[RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <https://www.rfc-editor.org/info/rfc4253>.

[RFC4456] Bates, T., Chen, E., and R. Chandra, "BGP Route Reflection: An Alternative to Full Mesh Internal BGP (IBGP)", RFC 4456, DOI 10.17487/RFC4456, April 2006, <https://www.rfc-editor.org/info/rfc4456>.

[RFC4655] Farrel, A., Vasseur, J.-P., and J. Ash, "A Path Computation Element (PCE)-Based Architecture", RFC 4655, DOI 10.17487/RFC4655, August 2006, <https://www.rfc-editor.org/info/rfc4655>.

[RFC5440] Vasseur, JP., Ed. and JL. Le Roux, Ed., "Path Computation Element (PCE) Communication Protocol (PCEP)", RFC 5440, DOI 10.17487/RFC5440, March 2009, <https://www.rfc-editor.org/info/rfc5440>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC7589] Badra, M., Luchuk, A., and J. Schoenwaelder, "Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication", RFC 7589, DOI 10.17487/RFC7589, June 2015, <https://www.rfc-editor.org/info/rfc7589>.

[RFC7752] Gredler, H., Ed., Medved, J., Previdi, S., Farrel, A., and S. Ray, "North-Bound Distribution of Link-State and Traffic Engineering (TE) Information Using BGP", RFC 7752, DOI 10.17487/RFC7752, March 2016, <https://www.rfc-editor.org/info/rfc7752>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC7988] Rosen, E., Ed., Subramanian, K., and Z. Zhang, "Ingress Replication Tunnels in Multicast VPN", RFC 7988, DOI 10.17487/RFC7988, October 2016, <https://www.rfc-editor.org/info/rfc7988>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8253] Lopez, D., Gonzalez de Dios, O., Wu, Q., and D. Dhody, "PCEPS: Usage of TLS to Provide a Secure Transport for the Path Computation Element Communication Protocol (PCEP)", RFC 8253, DOI 10.17487/RFC8253, October 2017, <https://www.rfc-editor.org/info/rfc8253>.

[RFC8345] Clemm, A., Medved, J., Varga, R., Bahadur, N., Ananthakrishnan, H., and X. Liu, "A YANG Data Model for Network Topologies", RFC 8345, DOI 10.17487/RFC8345, March 2018, <https://www.rfc-editor.org/info/rfc8345>.

[RFC8401] Ginsberg, L., Ed., Przygienda, T., Aldrin, S., and Z. Zhang, "Bit Index Explicit Replication (BIER) Support via IS-IS", RFC 8401, DOI 10.17487/RFC8401, June 2018, <https://www.rfc-editor.org/info/rfc8401>.

[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/info/rfc8402>.

[RFC8444] Psenak, P., Ed., Kumar, N., Wijnands, IJ., Dolganow, A., Przygienda, T., Zhang, J., and S. Aldrin, "OSPFv2 Extensions for Bit Index Explicit Replication (BIER)", RFC 8444, DOI 10.17487/RFC8444, November 2018, <https://www.rfc-editor.org/info/rfc8444>.

[RFC8556] Rosen, E., Ed., Sivakumar, M., Przygienda, T., Aldrin, S., and A. Dolganow, "Multicast VPN Using Bit Index Explicit Replication (BIER)", RFC 8556, DOI 10.17487/RFC8556, April 2019, <https://www.rfc-editor.org/info/rfc8556>.

[TE-RFC3272bis] Farrel, A., Ed., "Overview and Principles of Internet Traffic Engineering", Work in Progress, Internet-Draft, draft-ietf-teas-rfc3272bis-16, 24 March 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-teas-rfc3272bis-16>.

Appendix A. BIER-TE and Segment Routing (SR)

SR [RFC8402] aims to enable lightweight path steering via loose source routing. For example, compared to its more heavyweight predecessor, RSVP-TE, SR does not require per-path signaling to each of these hops.

BIER-TE supports the same design philosophy for multicast. As is the case for SR, it relies on source routing -- via the definition of a BitString. Like SR, it only requires consideration of the "hops" on which either replication has to happen, or across which the traffic should be steered (even without replication). Any other hops can be skipped via the use of routed adjacencies.

A BIER-TE bit position (BP) can be understood as the BIER-TE equivalent of "forwarding segments" in SR, but they have a different scope than SR forwarding segments. Whereas forwarding segments in SR are global or local, BPs in BIER-TE have a scope that is the group of one or more BFRs that have adjacencies for this BP in their BIFTs. These segments can be called "adjacency-scoped" forwarding segments.

Adjacency scope could be global, but then every BFR would need an adjacency for this BP, for example a forward_routed() adjacency with encapsulation to the global SR SID of the destination. Such a BP would always result in ingress replication, though (as in [RFC7988]). The first BFR encountering this BP would directly replicate to it. Only by using non-global adjacency scope for BPs can traffic be steered and replicated on a non-BFIR.

SR can naturally be combined with BIER-TE and help to optimize it. For example, instead of defining bit positions for non-replicating hops, it is equally possible to use segment routing encapsulations (e.g. SR-MPLS label stacks) for the encapsulation of "forward_routed" adjacencies.

Note that (non-TE) BIER itself can also be seen as being similar to SR. BIER BPs act as global destination Node-SIDs, and the BIER BitString is simply a highly optimized mechanism to indicate multiple such SIDs and let the network take care of effectively replicating the packet hop by hop to each destination Node-SID. However, BIER does not allow the indication of intermediate hops or, in terms of SR, the ability to indicate a sequence of SIDs to reach the destination. This is what BIER-TE and its adjacency-scoped BP enable.

Acknowledgements

The authors would like to thank Greg Shepherd, IJsbrand Wijnands, Neale Ranns, Dirk Trossen, Sandy Zheng, Lou Berger, Jeffrey Zhang, Carsten Bormann, and Wolfgang Braun for their reviews and suggestions.

Special thanks to Xuesong Geng for shepherding this document and for IESG review/suggestions by Alvaro Retana (responsible AD/RTG), Benjamin Kaduk (SEC), Tommy Pauly (TSV), Zaheduzzaman Sarker (TSV), Éric Vyncke (INT), Martin Vigoureux (RTG), Robert Wilton (OPS), Erik Kline (INT), Lars Eggert (GEN), Roman Danyliw (SEC), Ines Robles (RTGDIR), Robert Sparks (Gen-ART), Yingzhen Qu (RTGdir), and Martin Duke (TSV).

Authors' Addresses

Toerless Eckert (EDITOR)
Futurewei Technologies Inc.
2330 Central Expy
Santa Clara, CA 95050
United States of America
Email: tte+ietf@cs.fau.de

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

Gregory Cauchie
KOEVOO
Email: gregory@koevoo.tech