Workgroup: Network Working Group
Internet-Draft: draft-ietf-bier-te-arch-12
Published: January 2022
Intended Status: Standards Track
Expires: 1 August 2022
Authors: T.T.E. Eckert, Ed. M.M. Menth
Futurewei University of Tuebingen
G.C. Cauchie
Bouygues Telecom

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, RFC8279) packets. 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" (BP). They indicate adjacencies of the network topology, as opposed to (non-TE) BIER in which BPs indicate "Bit-Forwarding Egress Routers" (BFER). A BIER-TE packets BitString therefore indicates the edges of the (loop-free) tree that the packet is forwarded across 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 "sub-domains" (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).

As it operates on the same per-packet stateless forwarding principles, BIER-TE can also be a good fit to support multicast path steering in "Segment Routing" (SR) networks.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents

at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 5 July 2022.

Copyright Notice

Table of Contents

1. Overview

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 in the expectation that the reader is familiar with these two documents.

BIER-TE introduces a new semantic for "bit positions" (BP). They indicate adjacencies of the network topology, as opposed to (non-TE) BIER in which BPs indicate "Bit-Forwarding Egress Routers" (BFER). A

BIER-TE packets BitString therefore indicates the edges of the (loop-free) tree that the packet is forwarded across by BIER-TE. With BIER-TE, the "Bit Index Forwarding Table" (BIFT) of each "Bit Forwarding Router" (BFR) is only populated with BP that are adjacent to the BFR in the BIER-TE Topology. Other BPs are empty in the BIFT. The BFR replicate and forwards BIER packets to adjacent BPs that are set in the packet. 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 sub-domains. 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).

As it operates on the same per-packet stateless forwarding principles, BIER-TE can also be a good fit to support multicast path steering in "Segment Routing" (SR) networks ([RFC8402]).

This document is structured as follows:

* Section 2 introduces BIER-TE with two forwarding examples, followed by an introduction of 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, 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 type of adjacencies and possible variations of BIER-TE forwarding pseudocode, and finally the mandatory and optional requirements.

* Section 5 describes operational considerations for the BIER-TE controller, foremost how the BIER-TE controller can optimize the use of BP by using specific type of BIER-TE adjacencies for different type of topological situations, but also how to assign bits to avoid loops and duplicates (which in BIER-TE does not come for free), and finally how "Set Identifier" (SI), "sub-domain" (SD) and BFR-ids can be managed by a BIER-TE controller, examples and summary.

* Appendix A concludes the technology specific sections of the document by further relating BIER-TE to SR.

Note that related work, [I-D.ietf-roll-ccast] 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 BitString 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 related work [ICC] describes it.

1.1. 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.

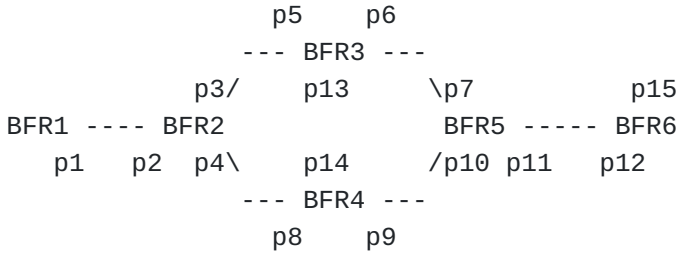
2. Introduction

2.1. Basic Examples

BIER-TE forwarding is best introduced with simple examples. These examples use formal terms defined later in the document (Figure 4), including forward_connected(), forward_routed() and local_decap().

BIER-TE Topology:

Diagram:



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

```
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

Consider the simple network in the above BIER-TE overview example picture with 6 BFRs. p1...p15 are the bit positions used. All BFRs can act as an ingress BFR (BFIR), BFR1, BFR3, BFR4 and BFR6 can also be BFERs. Forward_connected() is the name for adjacencies that are representing subnet adjacencies of the network. Local_decap() is the name of the adjacency to decapsulate BIER-TE packets and pass their payload to higher layer processing.

Assume 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 is showing the ability of the shown BIER-TE Topology to make the traffic pass across any possible path and be replicated where desired.

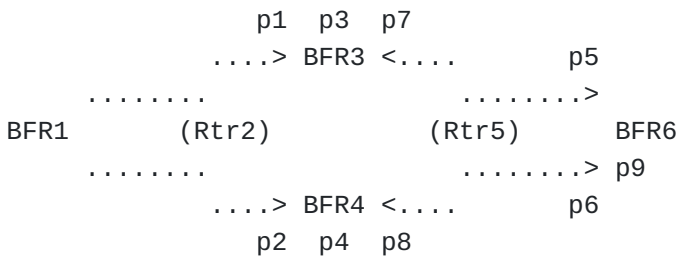
BIER-TE has various options to minimize BP assignments, many of which are based on out-of-band knowledge about the required multicast traffic paths and bandwidth consumption in the network, such as from pre-deployment planning.

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 BFIR but not BFER or transit BFR.

BIER-TE Topology:

Diagram:



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

```
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). From BFR1 via BFR4 to be received by BFR6, the BitString is (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).

2.2. 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 2.1. It is used to control where replication can or should happen and how to minimize the required number of BP for adjacencies.

The BIER-TE Topology consists of the BIFTs of all the BFR and can also be expressed as a directed graph where the edges are the adjacencies between the BFRs labelled with the BP used for the adjacency. Adjacencies are naturally unidirectional. 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.

2.3. 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 for 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 with needs for path steering but without desire for distributed routing protocols.

1. BIER-TE inherits the following aspects from BIER unchanged:
    1. The fundamental purpose of per-packet signaled replication and delivery via a BitString.
    2. The overall architecture consisting of three layers, flow overlay, BIER(-TE) layer and routing underlay.
    3. The supported encapsulations [RFC8296].
    4. The semantic of all [RFC8296] header elements used by the BIER-TE forwarding plane other than the semantic of the BP in the BitString.
    5. 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:
    1. In BIER, bits in the BitString of a BIER packet header indicate a BFER and bits in the BIFT indicate the BIER control plane 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.

2. In BIER, the implied reference option for the core part of the BIER layer control plane are the BIER extensions for distributed routing protocols. This includes ISIS/OSPF extensions for BIER, [RFC8401] and [RFC8444].

3. 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.

4. Assuming the reference options for the control plane, BIER-TE replaces in-network autonomous path calculation 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:

    1. "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.

    2. 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.

    3. 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.

    4. 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:

    1. The BIER/BIER-TE packet header needs to allow addressing 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 nights forwarding).

2. BIER-TE deployments will have to assign BFR-ids to BFRs and insert them into the BFIR-id field of BIER packet headers as BIER does, whenever the deployment uses (unchanged) components developed for BIER that use BFR-id, such as multicast flow overlays or BIER layer control plane elements. See also Section 5.3.3.

2.4. Accelerated/Hardware forwarding comparison

BIER-TE forwarding rules, especially the BitString parsing are designed to be as close as possible to those of BIER in 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 4.6), 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 being constituted from the same three layers as BIER: The "multicast flow overlay", the "BIER layer" and the "routing underlay". The following picture also shows how the "BIER layer" is constituted from the "BIER-TE forwarding plane" and the "BIER-TE control plane" represent by the "BIER-TE Controller".

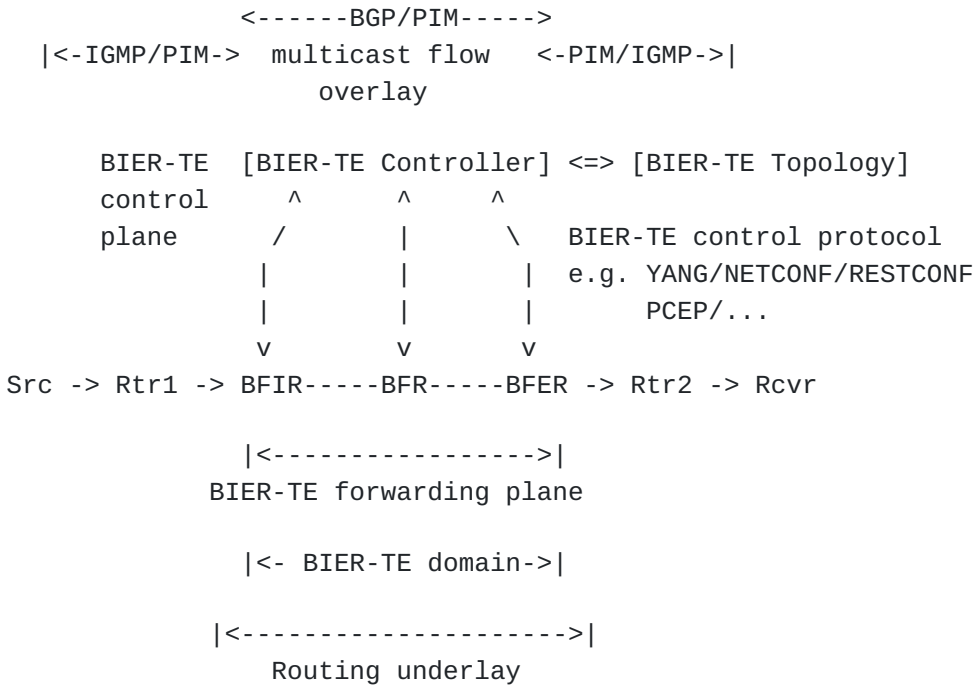


Figure 3: BIER-TE architecture

3.1. The Multicast Flow Overlay

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

When a BIER-TE controller is used, then 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 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 calculated BitStrings. See Section 3.2.1.2 and Section 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 ISIS/OSPF extensions for BIER, [RFC8401] and [RFC8444].

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

1. **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 a BIER-TE sub-domains: 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 controllers determinations about 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-id. Alternatively, BFIR-id 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.

2. **BIER-TE tree control:** During operations of the network, 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: BFIR 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 Section 3.2.1.2, Section 3.5 and Section 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 and 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

[RFC-Editor: the following text has three references to anchors topology-control, topology-control-1 and tree-control. Unfortunately, XMLv2 does not offer any tagging that reasonable references are generated (i had this problem already in RFCs last year. Please make sure there are useful-to-read cross-references in the RFC in these three places after you convert to XMLv3.]

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

- A BIER-TE controller.

- BFR data-models and protocols to communicate between controller 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 controller and BFIR 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 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 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 which 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 (Section 3.2, Paragraph 3, Item 2.2.1) 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, such as in 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 [I-D.ietf-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 BFER using different SIs/BitStrings. These topics are discussed in Section 5.

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

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

3.2.1.2. Engineered Trees via BitStrings

In BIER, the same set of BFER in a single sub-domain is always encoded as the same BitString. In BIER-TE, the BitString used to reach the same set of BFER in the same sub-domain can be different for different overlay flows because the BitString encodes the paths towards the BFER, so the BitStrings from different BFIR to the same set of BFER will often be different. Likewise, the BitString from the same BFIR to the same set of BFER 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 [I-D.ietf-bier-multicast-http-response] 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 re-use 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 link or nodes fail or recover in the topology, BIER-TE could quickly respond with FRR procedures such as [I-D.eckert-bier-te-

frr], 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

[RFC-editor Q: "is constituted from" / "consists of" / "composed from..." ???]

The BIER-TE Forwarding Plane is constituted from the following components:

1. On a BFIR, imposition of the BIER header for packets from overlay flows. This is driven by a combination of state established by the BIER-TE control plane and/or the multicast flow overlay as explained in Section 3.1.

2. On BFRs (including BFIR and BFER), 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 DSCP 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 that has one or more adjacencies in the BIFT, a copy is made according to the type of adjacencies for that BP in the BIFT. Before sending any copy, 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 packets from looping when the BitStrings erroneously includes a forwarding loop. When a forward_connected() adjacency has the "DoNotClear" (DNC) flag set, then this BP is re-set 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 (unicasted) 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 of 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 BFD. Providing these notifications is considered to be part of the routing underlay in this document.

3.5. Traffic Engineering Considerations

Traffic Engineering ([I-D.ietf-teas-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 of 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 to the BIER BIFT for (non-TE) BIER. It exists on every BFR running BIER-TE. For every BIER sub-domain (SD) in use for BIER-TE, it is a table as shown shown in Figure 4. 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 rows 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)	(FBM)	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)		...
...	...	...

BIER-TE Bit Index Forwarding Table (BIFT)

Figure 4: BIER-TE 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 re-use 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 but 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 the 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 type 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 which will cause the 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 non BIER-TE capable routers or to minimize the number of required BP 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 independent 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 Figure 4 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 allows to design 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 packet.

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 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 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 allows to also allocate non-overlapping label ranges for BIFT-id 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 BIFT-id is used which does not explicitly partition the BIFT-id space between BIER and BIER-TE, such as proposed for non-MPLS forwarding with [RFC8296] encapsulation in [I-D.ietf-bier-non-mpls-bift-encoding] revision 04, section 5, then 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 the same document does not statically encode BSL or SD into the BIFT-id, but allows for a mapping, and hence could provide for the same freedom as when MPLS is being used (same or different SD for BIER/BIER-TE).

forward_routed() requires an encapsulation that permits to direct 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 5, 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 5: 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 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 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 4.6), 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 6 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 (add/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 Section 5.1.5. The code therefore includes a loop over these adjacencies.

*The ECMP() adjacency is shown. 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 6: Complete BIER-TE Forwarding Pseudocode for required, recommended and optional functions

4.5. Basic BIER-TE Forwarding Example

[RFC Editor: remove this section.]

THIS SECTION TO BE REMOVED IN RFC BECAUSE IT WAS SUPERSEDED BY SECTION 1.1 EXAMPLE - IN CASE FINAL REVIES FIND A GOOD REASON TO KEEP IT, BUT DOESN'T SEEM TO SHOW IMPORTANT NEW STUFF OVER INITIAL EXAMPLES.

Step-by-step example of basic BIER-TE forwarding. This example does not use ECMP() or forward_routed() adjacencies nor does it try to minimize the number of required BitPositions for the topology.

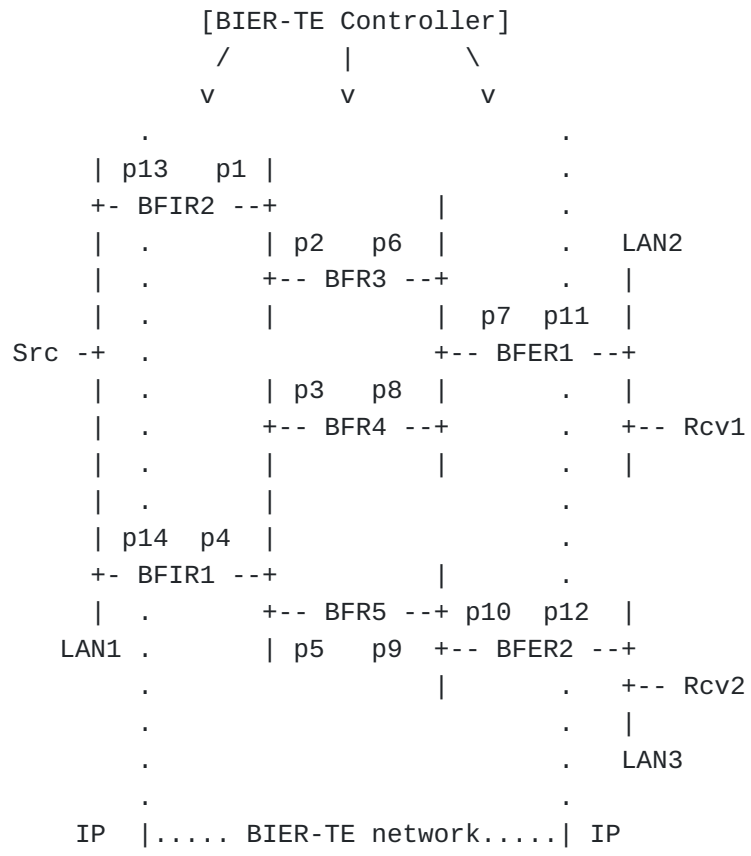


Figure 7: BIER-TE Forwarding Example

pXX indicate the BitPositions number assigned by the BIER-TE Controller to adjacencies in the BIER-TE topology. For example, p9 is the adjacency towards BFR5 on the LAN connecting to BFER2.

```
  BIFT BFIR2:
    p13: local_decap
     p2: forward_connected(BFR3)

  BIFT BFR3:
     p1: forward_connected(BFIR2)
     p7: forward_connected(BFER1)
     p8: forward_connected(BFR4)

  BIFT BFER1:
    p11: local_decap
     p6: forward_connected(BFR3)
     p8: forward_connected(BFR4)
```

Figure 8: BIER-TE Forwarding Example Adjacencies

...and so on.

For example, we assume that some multicast traffic seen on LAN1 needs to be sent via BIER-TE by BFIR2 towards Rcv1 and Rcv2. The BIER-TE Controller determines it wants it to pass this traffic across the following paths:

```
             -> BFER1 ---------------> Rcv1
BFIR2 -> BFR3
             -> BFR4 -> BFR5 -> BFER2 -> Rcv2
```

Figure 9: BIER-TE Forwarding Example Paths

These paths equal to the following BitString: p2, p5, p7, p8, p10, p11, p12.

This BitString is assigned by BFIR2 to the example multicast traffic received from LAN1.

Then BFIR2 forwards this multicast traffic with BIER-TE based on that BitString. The BIFT of BFIR2 has only p2 and p13 populated. Only p2 is in the BitString and this is an adjacency towards BFR3. BFIR2 therefore clears p2 in the BitString and sends a copy towards BFR2.

BFR3 sees a BitString of p5,p7,p8,p10,p11,p12. For those BPs, it has only adjacencies for p7,p8. It creates a copy of the packet to BFER1 (due to p7) and one to BFR4 (due to p8). It clears both p7 and p8 before sending.

BFER1 sees a BitString of p5,p10,p11,p12. For those BPs, it only has an adjacency for p11. p11 is a local_decap() adjacency installed by the BIER-TE Controller to receive a copy of the BIER packet -

dispose of the BIER header and pass the payload to IP multicast. IP multicast will then forward the packet out to LAN2 because it did receive PIM or IGMP joins on LAN2 for the traffic.

Further processing of the packet in BFR4, BFR5 and BFER2 accordingly.

4.6. BFR Requirements for BIER-TE forwarding

BFR that support BIER-TE and BIER MUST support configuration that enables BIER-TE instead of (non-TE) BIER forwarding rules for all BIFT of one or more BIER sub-domains. 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 to save bits 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 requirement, 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 requires therefore only one bit position.

5.1.2. BFER

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

5.1.3. Leaf BFERs

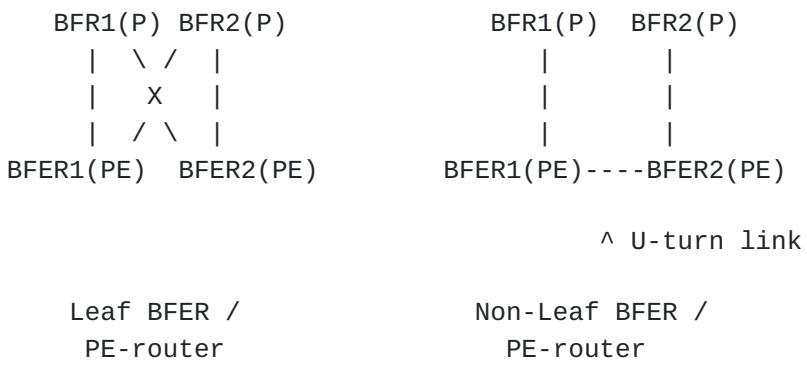


Figure 10: Leaf vs. non-Leaf BFER Example

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 in Figure 10: When BFER1/BFER2 are Non-Leaf BFER 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 in the left-hand picture are only guaranteed to be leaf-BFER by fitting routing configuration that prohibits transit traffic to pass through the BFERs, which is commonly applied in these topologies.

In most situations, leaf-BFER 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-BFER 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, such as in Section 5.1.4 or Section 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 leaf-BFER, and packets are therefore decapsulated potentially unnecessarily, this may still be appropriate if the decapsulated payload of the BIER-TE packet does indicate whether or not the packet needs 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 packet was 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 BFER are to receive and decapsulate a packet, then these two optimizations can not be used together with shared bit positions optimization for 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.

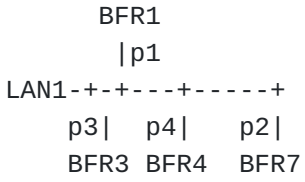


Figure 11: LAN Example

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 BFRs 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 spokes 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 12, 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.

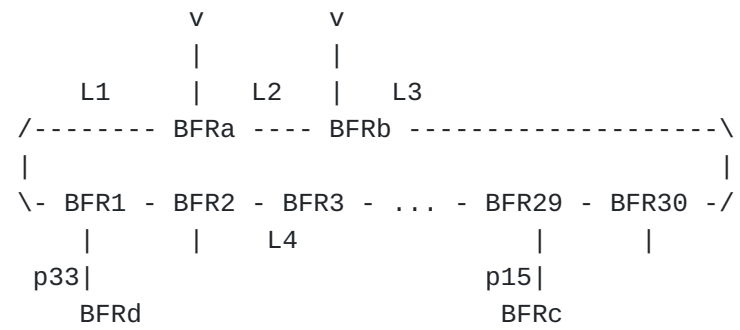


Figure 12: Ring Example

Note that this example only permits for packets intended to make it all the way around the ring to enter it at BFRa and BFRb, and 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 ingress ring BFR 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 ingress BFR 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)

[RFC-Editor: A reviewer (Lars Eggert) noted that the infinite "to use" in the following sentence is not correct. The same was also noted for several other similar instances. The following URL seems to indicate though that this is a per-case decision, which seems undefined: https://writingcenter.gmu.edu/guides/choosing-between-infinitive-and-gerund-to-do-or-doing. What exactly should be done about this ?].

An ECMP() adjacency allows to use just one BP to deliver packets to one one of N adjacencies instead of one BP for each adjacency. In the common example case Figure 13, a link-bundle of three links L1,L2,L3 connects BFR1 and BFR2, and only one BP is used instead of three BP to deliver packets from BFR1 to BFR2.

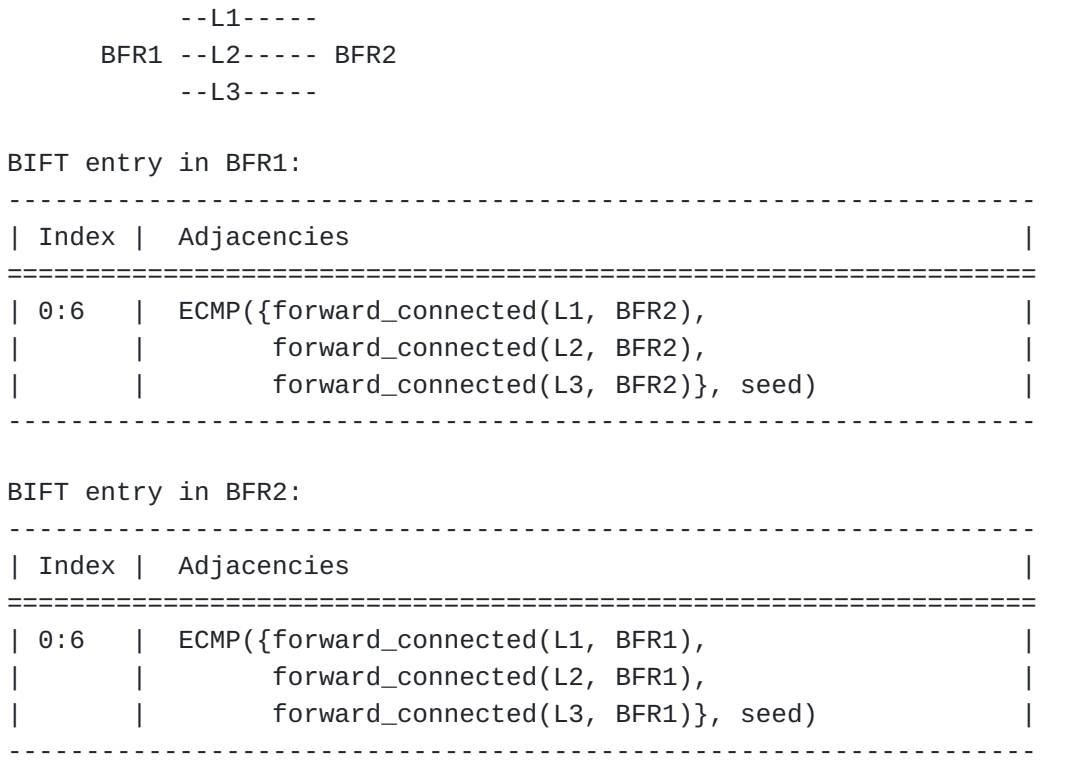


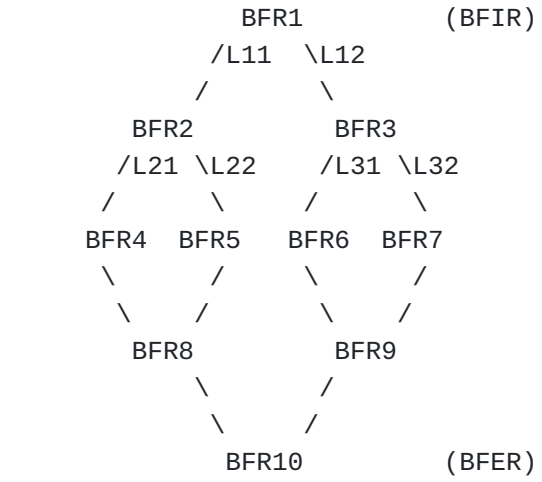
Figure 13: ECMP Example

This document does not standardize any ECMP algorithm because it is sufficient for implementations to document their freely chosen ECMP algorithm. Figure 14 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 14: ECMP algorithm Example

In the following example, 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, but to illustrate that ECMP can be used to share BPs not only across link bundles, but also across alternative paths across different transit BFR, and it explains the use of the seed parameter.



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 15: Polarization Example

Note that for the following discussion of ECMP, only the BIFT ECMP adjacencies on BFR1, BFR2, BFR3 are relevant. The re-use of BP across BFR in this example is further explained in Section 5.1.9 below.

With the setup of ECMP 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 performs again 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. And therefore L22 and BFR5 sees no traffic. Likewise for L31 and BFR6.

This issue in BFR2/BFR3 is called polarization. It results from the re-use 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 allow to operate BIER-TE across intermediate hop routers that do not support BIER-TE.

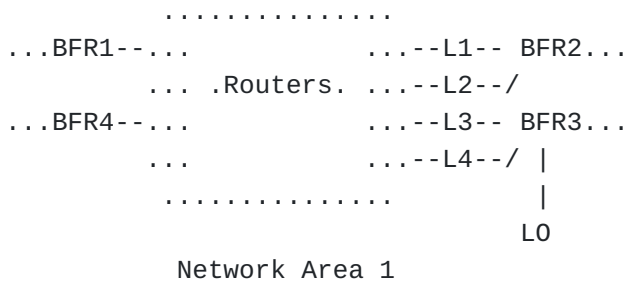


Figure 16: Forward Routed Adjacencies Example

Assume the requirement in Figure 16 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 three next segments: (1) BFR2 via link L1, (2) BFR2 via link L2, or (3) via BFR3 and then nor caring whether the packet is forwarded via L3 or L4.

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 non BIER-TE enabled nodes.

5.1.9. Reuse of bit positions (without DNC)

Bit positions can be re-used across multiple BFRs to minimize the number of BP 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 BP are cleared when passing a BFR with an adjacency for that BP, reuse of 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 BP is whether it allows to still achieve the desired Tree Engineering goals.

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. BP can be set across BFR that would (A) only occur across different paths or (B) across different branches of the same tree.

An example of (A) was given in Figure 15, 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 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 re-use of BP 0:7 between BFR2 and BFR3 is a case of (B)

where reuse of 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 where undesirable, this would be done by not setting BP 0:5 and BP 0:6 for the same packet. If the duplicates where desirable (e.g.: resilient transmission), the additional BP 0:10 would also not render additional value.

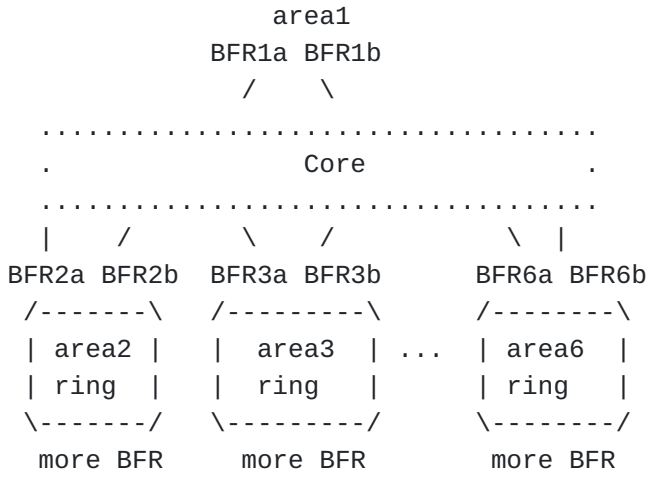


Figure 17: Reuse of BP

Reuse may also save BPs in larger topologies. Consider the topology shown in Figure 17. A BFIR/sender (e.g.: video headend) is attached to area 1, and area 2...6 contain receivers/BFER. Assume 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-BFER 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 BFR 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 BFR 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 allow to "tunnel" across non-BIER-TE capable routers and across BIER-TE capable routers where no traffic-steering or replications are required (Section 5.1.8).
- *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 scenario, this may limit the feasible path steering options (Section 5.1.9).

Note that the described list of optimizations is not exhaustive. Especially when the set of required path steering choices is limited and the set of possible subsets of BFERs that should be able to receive traffic is limited, further optimizations of BP are possible. 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 looping of packets. The only exception are adjacencies with DNC set.

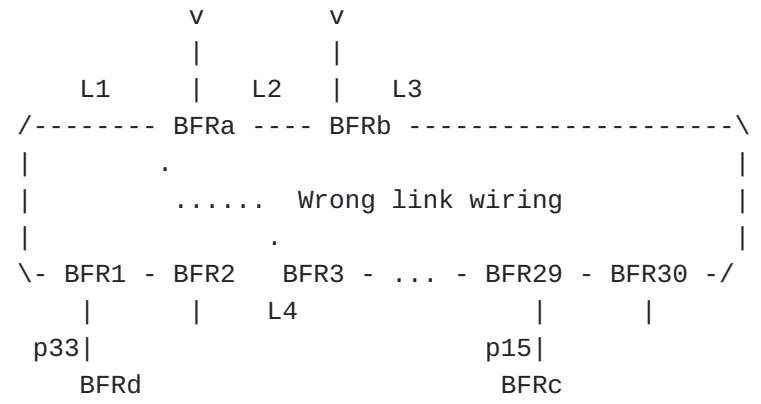


Figure 18: Miswired Ring Example

With DNC set, looping can happen. Consider in Figure 18 that link L4 from BFR3 is (inadvertently) plugged into the L1 interface of BFRa (instead of BFR2). This creates a loop where the rings clockwise bit position is never cleared for copies of the packets traveling clockwise around the ring.

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

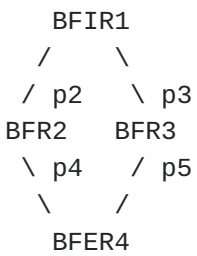


Figure 19: Duplicates Example

Duplicates happen when the graph expressed by a BitString is not a tree but redundantly connecting BFRs with each other. In Figure 19, a BitString of p2,p3,p4,p5 would result in duplicate packets to arrive on BFER4. The BIER-TE Controller must therefore ensure to only create BitStrings that are trees.

When links are incorrectly physically re-connected 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 re-addressing operations must be coordinated with the BIER-TE Controller.

5.3. Managing SI, sub-domains 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 sub-domains must be used. This section discusses how.

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

5.3.1. Why SI and sub-domains

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

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

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

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 sub-domain 0. Ideal replication efficiency for N BFER exists in a sub-domain if they are split over not more than ceiling(N/BitStringLength) SI.

If service instances justify additional BIER:SI state in the network, additional sub-domains will be used: BFIR/BFER are assigned

BFR-id in those sub-domains and each service instance is configured to use the most appropriate sub-domain. This results in improved replication efficiency for different services.

Even if creation of sub-domains and assignment of BFR-id to BFIR/BFER in those sub-domains is automated, it is not expected that individual service instances can deal with BFER in different sub-domains. A service instance may only support configuration of a single sub-domain 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 SI and sub-domain logically in the same way as they are used in BIER: All necessary BFIR/BFER 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 sub-domains that primarily exist to provide more efficient replication (and for BIER-TE desirable path steering) for different subsets of BFIR/BFER.

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 BFER 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 because it depends on the physical topology, and on the desire of the operator to allow for 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 that they will allow to express desirable path steering alternatives.

5.3.3. Assigning BFR-id with BIER-TE

BIER-TE forwarding does not use the BFR-id, nor does it require for the BFIR-id field of the BIER header to 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 in (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 BFR 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 performs 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 for flow overlay applications to operate independently of the controller whenever it needs to determine which subset of BFERs need to receive a particular packet.

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

Note that in either case (unlike in 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.: re-optimization 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 requires 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 identifier. 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 sub-domain 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-id in the same sub-domain. This allows higher replication efficiency for BIER because their BFR-id can be assigned sequentially, while the BitStrings for BIER-TE will have also 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 20. 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 BFER in the network, 4 copies need to be sent by the BFIR. On the BFIR it does not make a difference how the BFR-ids are allocated to BFER in the network, but for efficiency further down in the network it does make a difference.

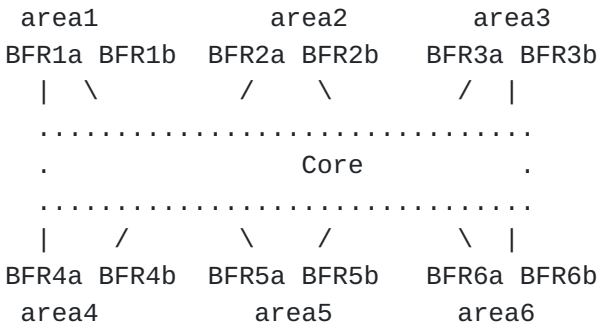


Figure 20: Scaling BIER-TE bits by reuse

With random allocation of BFR-id to BFER, each receiving area would (most likely) have to receive all 4 copies of the BIER packet because there would be BFR-id 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 BFER in an area would be given BFR-id with as few as possible different SIs. Each area would only have to forward one or two packets instead of 4.

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 sub-domain 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-id is just a derived set of identifiers from the operator/BIER-TE Controller as explained above.

Every time that different sub-topologies have overlap, bits need to be repeated across the BitStrings, increasing the overall amount of bits required across all BitString/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 SI and therefore reduces the number of BFER 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 20, 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 number of bits required for the topology in the area. In this example, 6 SIs, one per area.

In addition, we use 4 bits in each SI: bia, bib, bea, beb: (b)it (i)ngress (a), (b)it (i)ngress (b), (b)it (e)gress (a), (b)it (e)gress (b). These bits will be used to pass BIER packets from any BFIR via any combination of 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 BFR, 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 BFER 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 sub-domain. This allows to apply the mapping BFR-id = SI * BSL + BP. This allows to re-use the BIER architecture concept of BFR-id and therefore minimize 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 BFIR/BFER possible in a sub-domain is smaller than in BIER because BIER-TE uses additional bits for topology.

Sub-domains (SDs) in BIER-TE can be used like 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 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 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 can not 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 BFR 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 that a looping packet creates additional packet amplification through the misconfigured loop on the packet's second or further times around the loop, because all relevant adjacency bits would have been cleared on the first round through the loop. In 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, such as in embedded networks or in manufacturing networks during e.g. plant reworking/repairs. In these type 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 to provide additional layers of protection against configuration attacks, but would foremost protect the active production process from such configuration attacks.

7. IANA Considerations

This document requests no action by IANA.

8. Acknowledgements

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

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

9. Change log [RFC Editor: Please remove]

draft-ietf-bier-te-arch:

12:

AD review Alvaro Retana.

Various textual/editorial nits including adding () to all instances of forwarding adjacency name instances.

3.1 Added new paragraph outlining possible use of BGP as RR in BIER-TE controller as core of multicast flow overlay component of BIER-TE.

3.2 added xref's to relevant sections to the listed control plane points.

4.1 rewrote paragraphs of 4.1 leading up to Figure 4. to eliminate any confusion in how the BIFT work and how it compares to the notions in rfc8279, as well as better linking it to the Pseudocode.

Moved SR section into appendix.

TSV review Martin Duke.

Text/editorial nits.

4.4 improved text describing handling of F-BM.

RTGdir review Yingzhen Qu.

Various text/editorial nits.

Added notion that BitStrings represent loop free tree for packet to abstract and intro.

Various text nit and editorial improvements.

Fixed some BFR-id field -> BFIR-id field mistakes.

Capitalized NETCONF/RESTCONF/YANG, added RFC references.

Improved Figure 16 with explicitly two links into BFR3 and explanatory text.

Gen-ART review Robert Sparks.

Various textual nits, editorial improvements.

3.2 Introduced terms "BIER-TE topology control" and "BIER-TE tree control" for the two functional components of the control plane.

3.2.1 - 3.2 change introduces the open RFC-editor issue of appropriate xrfs (to be resolved by RFC-editor).

3.3 Rewrote last paragraph to better describe loop prevention through clearing of bits in BitString.

4.1 Fixed up text/formula describing mapping between bfr-id, SI:BP and SI,BSL and BP. Fix offset bug.

5.3.6.2 Improved description paragraph explaining overlap of topology for different SI.

5.3.7 Improved first summary paragraph.

7. Rephrased applicability statement of control plane protocol security considerations to BIER-TE security.

RTGDIR review Ines Robles.

Fixed up adjacencies in Example 2 and explanation text to be explicit about which BFR not only passes, but also receives the packet.

7. (security considerations). Added paragraph about forward_routed() and prohibiting BIER packet leaking in/out of domain.

IESG review Roman Danyliv (SEC).

Several textual/sentence nits/editorials.

IESG review Lars Eggert (GEN).

Various good editorial word fixed.

Pointer to non-false-positive bloom filter work that looks like it happened after our IETF discussions documented in this doc, so will not add it to doc, but here is URL for folks interested: https://ieeexplore.ieee.org/document/8486415.

Did not change "native" to a different word for inclusivity because of my worry there is no estavblished single replacement word, making reading/searching/understanding more difficult.

IESG review Martin Vigeureux (RTG).

Added back reference to RFC8402. Textual fixes.

IESG review Eric Kline (INT).

2.1 Fixed typo in BFR* explanations.

4.3 Added explanatio about MTU handling.

IESG review Eric Vyncke (INT).

Fixed up initial text to introduce various abbreviations.

2.4 refined wording to "with the _intent_ to easily build common forwarding planes...".

4.2.3 refined text about entropy in ECMP - now taken text from rfc8279.

IESG review Zaheduzzaman Sarker (TSV).

5.1.7 Refined text explaining documentation of ECMP algorithm.

5.3.6.2. fixed range of areas/SI over which to build the example large network BPs - removed explanation of the large network shown to be only used for sources in area 1 (IPTV), because it was a stale explanation.

IESG review Ben Kaduk (round 2):

4.4 Advanced pseudocode still had one wrong "~". Root cause seems to have been day 0 problem in pseudocode written for -01, "~" was inserted in the wrong one of two code lines. Also enhanced textual description and comments in pseudocode, changed variable name AdjacentBits to PktAdjacentBits to avoid confusion with AdjacentBits[SI].

5.1.3 Rewrote last two paragraphs explaining the sharing of bit positions for lead-BFER hopefully better. Also detailled how it interacts with other optimizations and the type of payload BIER-TE packets may carry.

4.4 (from Carsten Borman) changed spacing in pseudocode to be consistent. Fixed {VRF}, clarified pseudocode object syntax, typos.

11: IESG review Ben Kaduk, summary:

One discuss for bug in pseudocode. turned out to be one cahrcter typo.

Added (non-TE) prefix in places where BIER by itsels had to be better disambiguated.

enhanced text for hub-and-spoke to indicate we're only talking about hub to spoke traffic.

long list ot language fixes/improvement (nits). Thanks a lot!.

add suggestion to SHOULD use known confidentiality protocols between controller and BFR.

10: AD review Alvaro Retana, summary:

Note: rfcdiff shows more changes than actually exist because text moved around.

Summary:

1. restructuring: merged all controller sections under common controller ops main section, moved unfitting stuff out to other parts of doc. Split Intro section into Overview and Intro. Shortened Abstract, moved text into Overview, added sections overview.

2. enhanced/rewrote: 2.3 Comparison with -> Relationship to BIER-TE

3. enhanced/rewrote: 3.2 BIER-TE controller -> BIER-TE control plane, 3.2.1 BIER-TE controller, for consistency with rfc8279

4. additional subsections for Alvaros asks

5. added to: 3.3 BIER-TE forwarding plane (consistency with rfc8279)

6. Enhanced description of 4.3/encap considerations to better explain how BIER/BIER-TE can run together.

Notation: Markers (a),(b),... at end of points are references from the review discussion with Alvaro to the changes made.

Details:.

Throughout text: changed term spelling to rfc8279 - bit positions, sub-domain, ... (i).

Reset changed to clear, also DNR changed to DNC (Do Not Clear) (q).

Abstract: Shortened. Removed name explanation note (Tree Engineering), (a).

1. Introduction -> Overview: Moved important explanation paragraph from abstract to Introduction. Fixed text, (a).

Added bullet point list explanation of structure of document (e).

Renamed to Overview because that is now more factually correct.

1.1. Fixed bug in example adding bit p15.(l).

2. (New - Introduction): Moved section 1.1 - 1.3 (examples, comparison with BIER-TE) from Introduction into new "Overview" section. Primarily so that "requirements language" section (at end of Introduction) is not in middle of document after all the Introduction.

2.1 Removed discussion of encap, moved to 4.2.2 (m).

2.2 enhanced paragraph suggesting native/overlay topology types, also sugest type hybrid (n).

2.3 Overhauled comparison text BIER/BIER-TE, structured into common, different, not-required-by-te, integration-bier-bier-te. Changed title to "Relationship" to allow including last point. (f).

2.4 moved Hardware forwarding comparison section into section 2 to allow coalescing of sections into section 5 about the controller operations (hardware forwarding was in the middle of it, wrong place). Shortened/improved third paragraph by pointing to BIFT as deciding element for selection between BIER/BIER-TE. Removed notion of experimentation (this now targets standard) (g).

3. (Components): Aligned component name and descriptions better with RFC8279. Now describe exactly same three layers. BIER layer constituted from BIER-TE control plane and BIER-TE forwarding plane. BIER-TE controller is now simply component of BIER-TE control plane. (b).

3.1. shortened/improved paragraph explaining use of SI:BP instead of also bfr-id as index into BIFT, rewrote paragraph talking about reuse of BPs(o).

3.2. rewrote explanation of BIER-TE control plane in the style of RFC8729 Section 4.2 (BIER layer) with numbered points. Note that RFC8729 mixes control and forwarding plane bullet points (this doc does not). Merged text from old sections 2.2.1 and 2.2.3 into list. (b).

3.2.1. Expanded/improved explanation of BIER-TE Controller (b).

3.2.1.1. Added subsection for topology discovery and creation (d).

3.2.1.2. Added subsection for engineered BitStrings as key novel aspect not found in BIER. (X).

3.3. Added numbered list for components of BIER-TE forwarding plane (completing the comparable text from RFC8729 Section 4.2).

3.4 Alvaro does not mind additional example, fixed bugs.

3.5 Removed notion about using IGP BIER extensions for BIER-TE, such as BIFT address ranges. After -10 making use of BIFT clearer, it now looks to authors as if use of IGP extensions would not be beneficial, as long as we do need to use the BIER-TE controller, e.g. unlike in BIER, a BFR could not learn from the IGP information what traffic to send towards a particular BIFT-ID, but instead that is the core of what the controller needs to provide.

4.2.2 Improved text to explain requirement to identify BIER-TE in the tunnel encap and compress description of use-cases (m).

4.2.3 enhanced ECMP text (p).

4.3. rewrote most of Encapsulation Considerations to better explain to Alvaros question re sharing or not sharing SD via BIER/BIER-TE. Added reference to I-D.ietf-bier-non-mpls-bift-encoding as a very helpful example. (f).

4.3 Renamed title to "...Co-Existence with BIER" as this is what it is about and to help finding it from abstract/intro ("co-exist") (j).

4.4. Moved BIER-TE Forwarding Pseudocode here to coalesce text logically. Changed text to better compare with BIER pseudo forwarding code. Numerical list of how F-BM works for BIER-TE. Removed efficiency comparison with BIER (too difficult to provide sufficient justification, derails from focus of section) (j).

4.6. (Requirements) Restructured: Removed notion of "basic" BIER-TE forwarding, simply referring to it now as "mandatory" BIER-TE forwarding. Cleaned up text to have requirements for different adjacencies in different paragraphs. (c).

5. Created new main section "BIER-TE Controller operational considerations", coalesced old sections 4., 5., 7. into this new main section. No text changes. (k).

5.1.9 Added new separate picture instead of referring to a picture later in text, adjusted text (r).

5.3.2 Changed title to not include word "comparison" to avoid this being accounted against Alvaros concern about scattering comparison (IMHO text already has little comparison, so title was misleading) (h).

co-authors internal review:

4.4 Added xref to Figure 5.

5.2.1 Duplicated ring picture, added visuals for described miswiring (s).

5.2.2 replace "topology" with graph (wrong word).

5.3.3 rewrote explanation of how to map BFR-id to SI:BP and assign them, clarified BFR-id is option. Retitled to better explain scope of section.

5.3.4 Removed considerations in 5.3.4 for sharing BFR-id across BIER/BIER-TE (t), changed title to explain how BFIR/BIER-TE controller interactions need some form of identifying BFR but this does not have to be BFR-id.

7. Added new security considerations (u).

09: Incorporated fixes for feedback from Shepherd (Xuesong Geng).

Added references for Bloom Filters and Rate Controlled Service Disciplines.

1.1 Fixed numbering of example 1 topology explanation. Improved language on second example (less abbreviating to avoid confusion about meaning).

1.2 Improved explanation of BIER-TE topology, fixed terminology of graphs (BIER-TE topology is a directed graph where the edges are the adjacencies).

2.4 Fixed and amended routing underlay explanations: detailed why no need for BFER routing underlay routing protocol extensions, but potential to re-use BIER routing underlay routing protocol extensions for non-BFER related extensions.

3.1 Added explanation for VRF and its use in adjacencies.

08: Incorporated (with hopefully acceptable fixes) for Lou suggested section 2.5, TE considerations.

Fixes are primarily to the point to a) emphasize that BIER-TE does not depend on the routing underlay unless forward_routed() adjacencies are used, and b) that the allocation and tracking of resources does not explicitly have to be tied to BPs, because they are just steering labels. Instead, it would ideally come from per-hop resource management that can be maintained only via local accounting in the controller.

07: Further reworking text for Lou.

Renamed BIER-PE to BIER-TE standing for "Tree Engineering" after votes from BIER WG.

Removed section 1.1 (introduced by version 06) because not considered necessary in this doc by Lou (for framework doc).

Added [RFC editor pls. remove] Section to explain name change to future reviewers.

06: Concern by Lou Berger re. BIER-TE as full traffic engineering solution.

Changed title "Traffic Engineering" to "Path Engineering"

Added intro section of relationship BIER-PE to traffic engineering.

Changed "traffic engineering" term in text" to "path engineering", where appropriate

Other:

Shortened "BIER-TE Controller Host" to "BIER-TE Controller". Fixed up all instances of controller to do this.

05: Review Jeffrey Zhang.

Part 2:

4.3 added note about leaf-BFER being also a propery of routing setup.

4.7 Added missing details from example to avoid confusion with routed adjacencies, also compressed explanatory text and better justification why seed is explicitly configured by controller.

4.9 added section discussing generic reuse of BP methods.

4.10 added section summarizing BP optimizations of section 4.

6. Rewrote/compressed explanation of comparison BIER/BIER-TE forwarding difference. Explained benefit of BIER-TE per-BP forwarding being independent of forwarding for other BPs.

Part 1:

Explicitly ue forwarded_connected adjcency in ECMP adjcency examples to avoid confusion.

4.3 Add picture as example for leav vs. non-leaf BFR in topology. Improved description.

4.5 Exampe for traffic that can be broadcast -> for single BP in hub&spoke.

4.8.1 Simplified example picture for routed adjacency, explanatory text.

Review from Dirk Trossen:

Fixed up explanation of ICC paper vs. bloom filter.

04: spell check run.

Addded remaining fixes for Sandys (Zhang Zheng) review:

4.7 Enhance ECMP explanations:

example ECMP algorithm, highlight that doc does not standardize ECMP algorithm.

Review from Dirk Trossen:

1. Added mentioning of prior work for traffic engineered paths with bloom filters.

2. Changed title from layers to components and added "BIER-TE control plane" to "BIER-TE Controller" to make it clearer, what it does.

2.2.3. Added reference to I-D.ietf-bier-multicast-http-response as an example solution.

2.3. clarified sentence about resetting BPs before sending copies (also forgot to mention DNR here).

3.4. Added text saying this section will be removed unless IESG review finds enough redeeming value in this example given how -03 introduced section 1.1 with basic examples.

7.2. Removed explicit numbers 20%/80% for number of topology bits in BIER-TE, replaced with more vague (high/low) description, because we do not have good reference material Added text saying this section will be removed unless IESG review finds enough redeeming value in this example given how -03 introduced section 1.1 with basic examples.

many typos fixed. Thanks a lot.

03: Last call textual changes by authors to improve readability:

removed Wolfgang Braun as co-authors (as requested).

Improved abstract to be more explanatory. Removed mentioning of FRR (not concluded on so far).

Added new text into Introduction section because the text was too difficult to jump into (too many forward pointers). This primarily consists of examples and the early introduction of the BIER-TE Topology concept enabled by these examples.

Amended comparison to SR.

Changed syntax from [VRF] to {VRF} to indicate its optional and to make idnits happy.

Split references into normative / informative, added references.

02: Refresh after IETF104 discussion: changed intended status back to standard. Reasoning:

Tighter review of standards document == ensures arch will be better prepared for possible adoption by other WGs (e.g. DetNet) or std. bodies.

Requirement against the degree of existing implementations is self defined by the WG. BIER WG seems to think it is not necessary to apply multiple interoperating implementations against an architecture level document at this time to make it qualify to go to standards track. Also, the levels of support introduced in -01 rev. should allow all BIER forwarding engines to also be able to support the base level BIER-TE forwarding.

01: Added note comparing BIER and SR to also hopefully clarify BIER-TE vs. BIER comparison re. SR.

- added requirements section mandating only most basic BIER-TE forwarding features as MUST.

- reworked comparison with BIER forwarding section to only summarize and point to pseudocode section.

- reworked pseudocode section to have one pseudocode that mirrors the BIER forwarding pseudocode to make comparison easier and a second pseudocode that shows the complete set of BIER-TE forwarding options and simplification/optimization possible vs. BIER forwarding. Removed MyBitsOfInterest (was pure optimization).

- Added captions to pictures.

- Part of review feedback from Sandy (Zhang Zheng) integrated.

00: Changed target state to experimental (WG conclusion), updated references, mod auth association.

- Source now on https://www.github.com/toerless/bier-te-arch

- Please open issues on the github for change/improvement requests to the document - in addition to posting them on the list (bier@ietf.). Thanks!.

draft-eckert-bier-te-arch:

06: Added overview of forwarding differences between BIER, BIER-TE.

05: Author affiliation change only.

04: Added comparison to Live-Live and BFIR to FRR section (Eckert).

04: Removed FRR content into the new FRR draft [I-D.eckert-bier-te-frr] (Braun).

- Linked FRR information to new draft in Overview/Introduction

- Removed BTAFT/FRR from "Changes in the network topology"

- Linked new draft in "Link/Node Failures and Recovery"

- Removed FRR from "The BIER-TE Forwarding Layer"

- Moved FRR section to new draft

- Moved FRR parts of Pseudocode into new draft

- Left only non FRR parts

- removed FrrUpDown(..) and //FRR operations in ForwardBierTePacket(..)

- New draft contains FrrUpDown(..) and ForwardBierTePacket(Packet) from bier-arch-03

- Moved "BIER-TE and existing FRR to new draft

- Moved "BIER-TE and Segment Routing" section one level up

- Thus, removed "Further considerations" that only contained this section

- Added Changes for version 04

03: Updated the FRR section. Added examples for FRR key concepts. Added BIER-in-BIER tunneling as option for tunnels in backup paths. BIFT structure is expanded and contains an additional match field to support full node protection with BIER-TE FRR.

03: Updated FRR section. Explanation how BIER-in-BIER encapsulation provides P2MP protection for node failures even though the routing underlay does not provide P2MP.

02: Changed the definition of BIFT to be more inline with BIER. In revs. up to -01, the idea was that a BIFT has only entries for a single BitString, and every SI and sub-domain would be a separate BIFT. In BIER, each BIFT covers all SI. This is now also how we define it in BIER-TE.

02: Added Section 5.3 to explain the use of SI, sub-domains and BFR-id in BIER-TE and to give an example how to efficiently assign bits for a large topology requiring multiple SI.

02: Added further detailed for rings - how to support input from all ring nodes.

01: Fixed BFIR -> BFER for section 4.3.

01: Added explanation of SI, difference to BIER ECMP, consideration for Segment Routing, unicast FRR, considerations for encapsulation, explanations of BIER-TE Controller and CLI.

00: Initial version.

10. References

10.1. 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>.

10.2. Informative References

[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>.

[I-D.eckert-bier-te-frr] 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://www.ietf.org/archive/id/draft-eckert-bier-te-frr-03.txt>.

[I-D.ietf-bier-multicast-http-response] 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://www.ietf.org/archive/id/draft-ietf-bier-multicast-http-response-06.txt>.

[I-D.ietf-bier-non-mpls-bift-encoding] Wijnands, I., 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://www.ietf.org/archive/id/draft-ietf-bier-non-mpls-bift-encoding-04.txt>.

[I-D.ietf-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-04, 7 November 2021, <https://www.ietf.org/archive/id/draft-ietf-bier-te-yang-04.txt>.

[I-D.ietf-roll-ccast] 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://www.ietf.org/archive/id/draft-ietf-roll-ccast-01.txt>.

[I-D.ietf-teas-rfc3272bis]
Farrel, A., "Overview and Principles of Internet Traffic Engineering", Work in Progress, Internet-Draft, draft-ietf-teas-rfc3272bis-13, 8 November 2021, <https://www.ietf.org/archive/id/draft-ietf-teas-rfc3272bis-13.txt>.

[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, 2016, May 2016, <https://ieeexplore.ieee.org/document/7511036>.

[RCSD94] Zhang, H. and D. Domenico, "Rate-Controlled Service Disciplines", Journal of High-Speed Networks, 1994, May 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>.

Appendix A. BIER-TE and Segment Routing

SR (xref target="RFC8402"/>) aims to enable lightweight path steering via loose source routing. Compared to its more heavy-weight predecessor RSVP-TE, SR does for example not require per-path signaling to each of these hops.

BIER-TE supports the same design philosophy for multicast. Like in SR, it relies on source-routing - via the definition of a BitString. Like SR, it only requires to consider 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.

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 BFR(s) that have adjacencies for this BP in their BIFT. This 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 non-ingress BFR.

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 to be 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. What BIER does not allow is to indicate intermediate hops, or in terms of SR the ability to indicate a sequence of SID to reach the destination. This is what BIER-TE and its adjacency scoped BP enables.

Authors' Addresses

Toerless Eckert (editor)
Futurewei Technologies Inc.
2330 Central Expy
Santa Clara, 95050

United States of America

Email: tte+ietf@cs.fau.de

Michael Menth
University of Tuebingen

Email: menth@uni-tuebingen.de

Gregory Cauchie
Bouygues Telecom

Email: GCAUCHIE@bouyguestelecom.fr