Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2021

T. Eckert, Ed.
Futurewei
G. Cauchie
Bouygues Telecom
M. Menth
University of Tuebingen
Oct 30, 2020

Tree Engineering for Bit Index Explicit Replication (BIER-TE)
draft-ietf-bier-te-arch-09

Abstract

This memo introduces per-packet stateless strict and loose path steered replication and forwarding for Bit Index Explicit Replication packets (RFC8279). This is called BIER Tree Engineering (BIER-TE). BIER-TE can be used as a path steering mechanism in future Traffic Engineering solutions for BIER (BIER-TE).

BIER-TE leverages RFC8279 and extends it with a new semantic for bits in the bitstring. BIER-TE can leverage BIER forwarding engines with little or no changes.

In BIER, the BitPositions (BP) of the packets bitstring indicate BIER Forwarding Egress Routers (BFER), and hop-by-hop forwarding uses a Routing Underlay such as an IGP.

In BIER-TE, BitPositions indicate adjacencies. The BIFT of each BFR are only populated with BPs that are adjacent to the BFR in the BIER-TE topology. The BIER-TE topology can consist of layer 2 or remote (routed) adjacencies. The BFR then replicates and forwards BIER packets to those adjacencies. This results in the aforementioned strict and loose path steering and replications.

BIER-TE can co-exist with BIER forwarding in the same domain, for example by using separate BIER sub-domains. In the absence of routed adjacencies, BIER-TE does not require a BIER routing underlay, and can then be operated without requiring an Interior Gateway Routing protocol (IGP).

BIER-TE operates without explicit in-network tree-state and carries the multicast distribution tree in the packet header. It can therefore be a good fit to support multicast path steering in Segment Routing (SR) networks.

Name explanation

[RFC-editor: This section to be removed before publication.]

Explanation for name change from BIER-TE to mean "Traffic Engineering" to BIER-TE "Tree Engineering" in WG last-call (to benefit IETF/IESG reviewers):

This document started by calling itself BIER-TE, "Traffic Engineering" as it is a mode of BIER specifically beneficial for Traffic Engineering. It supports per-packet bitstring based policy steering and replication. BIER-TE technology itself does not provide a complete traffic engineering solution for BIER but would require combination with other technologies for a full BIER based TE solution, such as a PCE and queuing mechanisms to provide bandwidth and latency reservations. It is also not the only option to build a traffic engineering solution utilizing BIER, for example BIER trees could be steered through IGP metric engineering, such as through Flex-Topologies. The architecure for Traffic Engineering with either modes of BIER (BIER-TE/BIER) is intended to be defined in a separate document, most likely in TEAs WG.

Because the name of such an overall solution is intended to be BIER-TE, the expansion of BIER-TE was therefore changed to name this BIER mode "Tree Engineering", so the overall solution can be distinguished better from its tree building/engineering method without having to change the long time well-established abbreviation BIER-TE.

Status of This Memo

Table of Contents

1. Introduction

BIER-TE shares architecture, terminology and packet formats with BIER as described in [RFC8279] and [RFC8296]. This document describes BIER-TE in the expectation that the reader is familiar with these two documents.

In BIER-TE, BitPositions (BP) indicate adjacencies. The BIFT of each BFR is only populated with BP that are adjacent to the BFR in the BIER-TE Topology. Other BPs are left without adjacency. The BFR replicate and forwards BIER packets to adjacent BPs that are set in the packet. BPs are normally also reset upon forwarding to avoid duplicates and loops. This is detailed further below.

Note that related work, [I-D.ietf-roll-ccast] uses Bloom filters [Bloom70] to represent leaves or edges of the intended delivery tree.

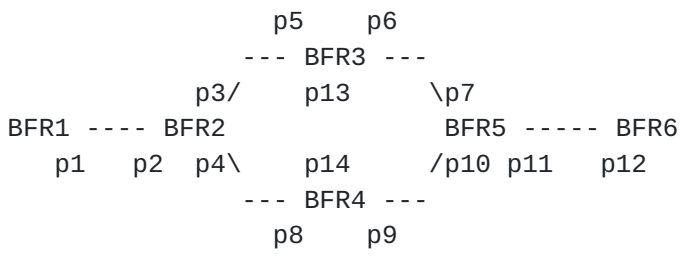
Bloom filters in general can support larger trees/topologies with fewer addressing bits than explicit bitstrings, but they introduce the heuristic risk of false positives and cannot reset bits in the bitstring during forwarding to avoid loops. For these reasons, BIER-TE uses explicit bitstrings like 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. Basic Examples

BIER-TE forwarding is best introduced with simple examples.

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
        p12 -> local_decap
```

Figure 1: BIER-TE basic example

Consider the simple network in the above BIER-TE overview example picture with 6 BFRs. p1...p14 are the BitPositions (BP) used. All BFRs can act as ingress BFR (BFIR), BFR1, BFR3, BFR4 and BFR6 can also be egress BFR (BFER). 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). When this packet is examined by BIER-TE on BFR1, the only BitPosition 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. p12 also makes BFR6 receive and decapsulate the packet.

To send in addition to BFR6 via BFR4 also a copy to BFR3, the bitstring needs to be (p2,p5,p8,p10,p12,p13). 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), 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.

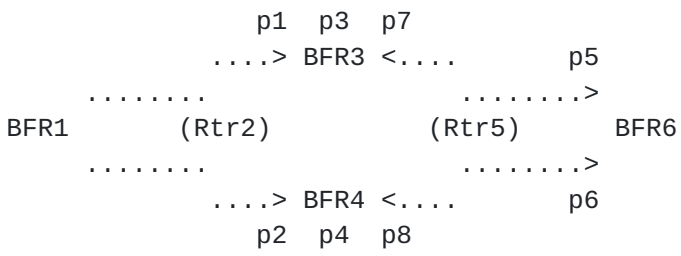
BIER-TE has various options to minimize BP assignments, many of which are based on assumptions about the required multicast traffic paths and bandwidth consumption in the network.

The following picture 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. Unicast tunneling of BIER-TE packets can leverage any feasible mechanism such as MPLS or IP, these encapsulations are out of scope of this document. To emphasize non-native forwarding of BIER-TE packets, these adjacencies are called "forward_routed", but 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:   p5  -> local_decap
        p6  -> local_decap
        p7  -> forward_routed to BFR3
        p8  -> forward_routed to BFR4
```

Figure 2: BIER-TE basic overlay example

To send a BIER-TE packet from BFR1 via BFR3 to BFR6, the bitstring is (p1,p5). From BFR1 via BFR4 to BFR6 it is (p2,p6). A packet from BFR1 to BFR3,BFR4 and from BFR3 to BFR6 uses (p1,p2,p3,p4,p5). A packet from BFR1 to BFR3,BFR4 and from BFR4 to BFR uses (p1,p2,p3,p4,p6). A packet from BFR1 to BFR4, and from BFR4 to BFR6 and from BFR6 to BFR3 uses (p2,p3,p4,p6,p7). A packet from BFR1 to BFR3, and from BFR3 to BFR6 and from BFR6 to BFR4 uses (p1,p3,p4,p5,p8).

1.2. BIER-TE Topology and adjacencies

The key new component in BIER-TE compared to BIER is the BIER-TE topology as introduced through the two examples in Section 1.1. 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 BIFT of all the BFR and can also be expressed as a directed graph where the edges are the adjacencies between the BFR 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 further down in the text.

If the BIER-TE topology represents the underlying (layer 2) topology of the network, this is called "native" BIER-TE as shown in the first example. This can be freely mixed with "overlay" BIER-TE, in "forward_routed" adjacencies are used.

1.3. Comparison with BIER

The key differences over BIER are:

- BIER-TE replaces in-network autonomous path calculation by explicit paths calculated by the BIER-TE Controller.
- In BIER-TE every BitPosition of the BitString of a BIER-TE packet indicates one or more adjacencies - instead of a BFER as in BIER.
- BIER-TE in each BFR has no routing table but only a BIER-TE Forwarding Table (BIFT) indexed by SI:BitPosition and populated with only those adjacencies to which the BFR should replicate packets to.

BIER-TE headers use the same format as BIER headers.

BIER-TE forwarding does not require/use the BFIR-ID. The BFIR-ID can still be useful though for coordinated BFIR/BFER functions, such as the context for upstream assigned labels for MPLS payloads in MVPN over BIER-TE.

If the BIER-TE domain is also running BIER, then the BFIR-ID in BIER-TE packets can be set to the same BFIR-ID as used with BIER packets.

If the BIER-TE domain is not running full BIER or does not want to reduce the need to allocate bits in BIER bitstrings for BFIR-ID values, then the allocation of BFIR-ID values in BIER-TE packets can be done through other mechanisms outside the scope of this document, as long as this is appropriately agreed upon between all BFIR/BFER.

1.4. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Components

End to end BIER-TE operations consists of four mayor components: The "Multicast Flow Overlay", the "BIER-TE control plane" consisting of the "BIER-TE Controller" and its signaling channels to the BFR, the "Routing Underlay" and the "BIER-TE forwarding layer". The Bier-TE Controller is the new architectural component in BIER-TE compared to BIER.

Picture 2: Components of BIER-TE

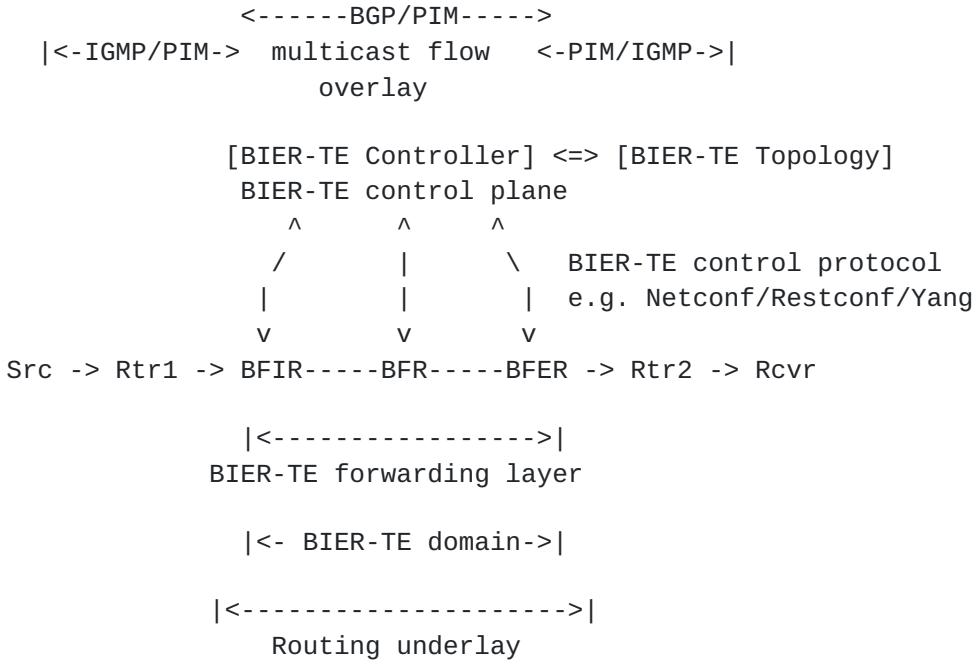


Figure 3: BIER-TE architecture

2.1. The Multicast Flow Overlay

The Multicast Flow Overlay operates as in BIER. See [RFC8279]. Instead of interacting with the BIER forwarding layer (as in BIER), it interacts with the BIER-TE Controller.

2.2. The BIER-TE Controller

The BIER-TE Controller is representing the control plane of BIER-TE. It communicates two sets of information with BFRs:

During initial provisioning or modifications of the network topology, the BIER-TE Controller discovers the network topology and creates the BIER-TE topology from it: determine which adjacencies are required/ desired and assign BitPositions to them. Then it signals the

resulting of BitPositions and their adjacencies to each BFR to set up their BIER-TE BIFTs.

During day-to-day operations of the network, the BIER-TE Controller signals to BFIRs what multicast flows are mapped to what BitStrings.

Communications between the BIER-TE Controller and BFRs is ideally via standardized protocols and data-models such as Netconf/Restconf/Yang. This is currently outside the scope of this document. Vendor-specific CLI on the BFRs is also a possible stopgap option (as in many other SDN solutions lacking definition of standardized data model).

For simplicity, the procedures of the BIER-TE Controller are described in this document as if it is a single, centralized automated entity, such as an SDN controller. It could equally be an operator setting up CLI on the BFRs. Distribution of the functions of the BIER-TE Controller is currently outside the scope of this document.

2.2.1. Assignment of BitPositions to adjacencies of the network topology

The BIER-TE Controller tracks the BFR topology of the BIER-TE domain. It determines what adjacencies require BitPositions so that BIER-TE explicit paths can be built through them as desired by operator policy.

The BIER-TE Controller then pushes the BitPositions/adjacencies to the BIFT of the BFRs, populating only those SI:BitPositions to the BIFT of each BFR to which that BFR should be able to send packets to
- adjacencies connecting to this BFR.

2.2.2. Changes in the network topology

If the network topology changes (not failure based) so that adjacencies that are assigned to BitPositions are no longer needed, the BIER-TE Controller can re-use those BitPositions for new adjacencies. First, these BitPositions 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.

2.2.3. Set up per-multicast flow BIER-TE state

The BIER-TE Controller interacts with the multicast flow overlay to determine what multicast flow needs to be sent by a BFIR to which set of BFER. It calculates the desired distribution tree across the BIER-TE domain based on algorithms outside the scope of this document

(e.g. CSFP, Steiner Tree, ...). It then pushes the calculated BitString into the BFIR.

See [I-D.ietf-bier-multicast-http-response] for a solution describing this interaction.

2.2.4. Link/Node Failures and Recovery

When link or nodes fail or recover in the topology, BIER-TE can quickly respond with the optional FRR procedures described in [I-D.eckert-bier-te-frr]. 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.

2.3. The BIER-TE Forwarding Layer

When the BIER-TE Forwarding Layer receives a packet, it simply looks up the BitPositions that are set in the BitString of the packet in the Bit Index Forwarding Table (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 resets all BP in the BitString of the packet for which the BFR has one or more adjacencies in the BIFT, except when the adjacency indicates "DoNotReset" (DNR, see Section 3.2.1). This is done to inhibit that packets can loop.

2.4. 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 3.2.2 for the adjacency definition.

BIER relies on the routing underlay to calculate paths towards BFER 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 bitsring. There is thus no need for BFER specific

routing underlay extensions to forward BIER packets with BIER-TE semantics.

BIER encapsulations may have BFER independent extensions in the routing underlay, such as the label range for BIER packets in the BIER over MPLS encapsulation ([RFC8296]). These BIER specific functions of the routing underlay are equally useable by BIER-TE. Alternatively, these 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.

2.5. 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 (BFR flow state) and BFR BIFT state. The mapping of user/IP traffic to specific BitStrings/BIER-TE flows is made based on policy. The specifics details of BIER-TE policies and how a controller uses such 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. 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 PCE 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 3.2.1 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 trafic and potential 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.

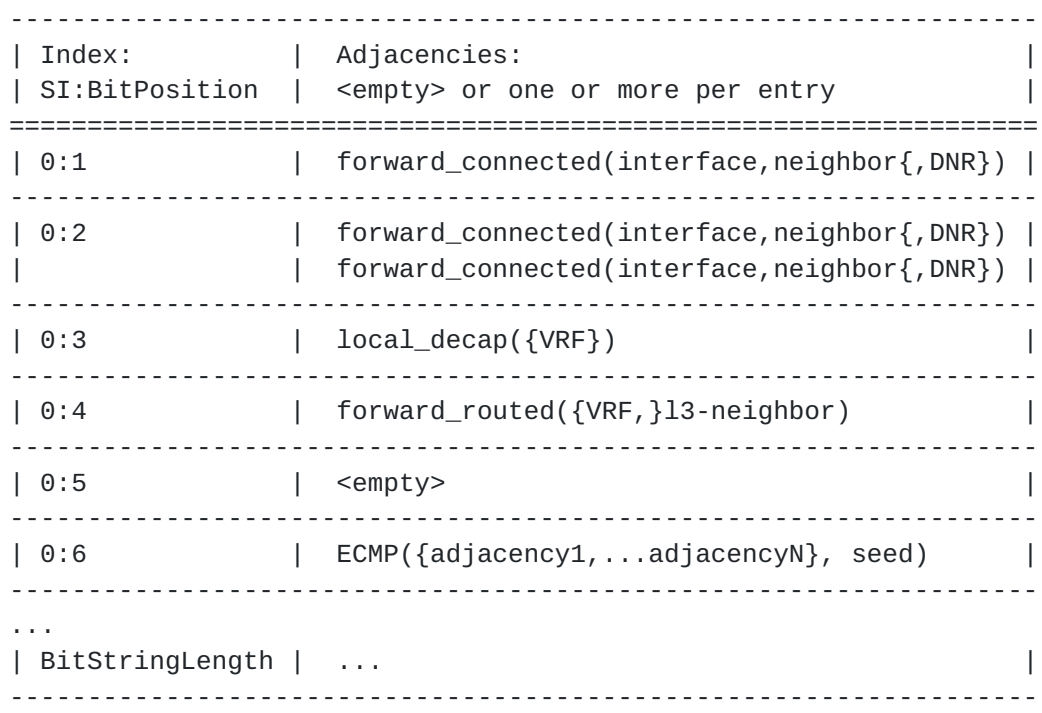
3. BIER-TE Forwarding

3.1. The Bit Index Forwarding Table (BIFT)

The Bit Index Forwarding Table (BIFT) exists in every BFR. For every subdomain in use, it is a table indexed by SI:BitPosition and is populated by the BIER-TE control plane. Each index can be empty or contain a list of one or more adjacencies.

BIER-TE can support multiple subdomains like BIER. Each one with a separate BIFT

In the BIER architecture, indices into the BIFT are explained to be both BFR-id and SI:BitString (BitPosition). This is because there is a 1:1 relationship between BFR-id and SI:BitString - every bit in every SI is/can be assigned to a BFIR/BFER. In BIER-TE there are more bits used in each BitString than there are BFIR/BFER assigned to the bitstring. This is because of the bits required to express the engineered path through the topology. The BIER-TE forwarding definitions do therefore not use the term BFR-id at all. Instead, BFR-ids are only used as required by routing underlay, flow overlay of BIER headers. Please refer to Section 7 for explanations how to deal with SI, subdomains and BFR-id in BIER-TE.



Bit Index Forwarding Table

Figure 4: BIFT adjacencies

The BIFT is programmed into the data plane of BFRs by the BIER-TE Controller and used to forward packets, according to the rules specified in the BIER-TE Forwarding Procedures.

Adjacencies for the same BP when populated in more than one BFR by the BIER-TE Controller does not have to have the same adjacencies. This is up to the BIER-TE Controller. BPs for p2p links are one case (see below).

{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.

3.2. Adjacency Types

3.2.1. 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 "DoNotReset" (DNR) set in the BIFT will not have the BitPosition for that adjacency reset when the BFR creates a copy for it. The BitPosition will still be reset for copies of the packet made towards other adjacencies. This can be used for example in ring topologies as explained below.

3.2.2. Forward Routed

A "forward_routed" adjacency is an adjacency towards a BFR that is not a forward_connected adjacency: towards a loopback address of a BFR or towards an interface address that is non-directly connected. Forward_routed packets are forwarded via the Routing Underlay.

If the Routing Underlay has multiple paths for a forward_routed adjacency, it will perform ECMP independent of BIER-TE for packets forwarded across a forward_routed adjacency. This is independent of BIER-TE ECMP described in Section 3.2.3.

If the Routing Underlay has FRR, it will perform FRR independent of BIER-TE for packets forwarded across a forward_routed adjacency.

3.2.3. ECMP

The ECMP mechanisms in BIER are tied to the BIER BIFT and are therefore not directly useable with BIER-TE. The following procedures describe ECMP for BIER-TE that we consider to be lightweight but also well manageable. It leverages the existing entropy parameter in the BIER header to keep packets of the flows on the same path and it introduces a "seed" parameter to allow for traffic flows to be polarized or randomized across multiple hops.

An "Equal Cost Multipath" (ECMP) adjacency has a list of two or more adjacencies included in it. It copies the BIER-TE to one of those adjacencies based on the ECMP hash calculation. The BIER-TE ECMP hash algorithm must select the same adjacency from that list for all packets with the same "entropy" value in the BIER-TE header if the same number of adjacencies and same seed are given as parameters. Further use of the seed parameter is explained below.

3.2.4. Local Decap

A "local_decap" adjacency passes a copy of the payload of the BIER-TE packet to the packets NextProto within the BFR (IPv4/IPv6, Ethernet,...). 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.

3.3. Encapsulation considerations

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

Because a BFR needs to interpret the BitString of a BIER-TE packet differently from a BIER packet, it is necessary to distinguish BIER from BIER-TE packets. This is subject to definitions in BIER encapsulation specifications.

MPLS encapsulation [RFC8296] for example assigns one label by which BFRs recognizes BIER packets for every (SI,subdomain) combination. If it is desirable that every subdomain can forward only BIER or BIER-TE packets, then the label allocation could stay the same, and only the forwarding model (BIER/BIER-TE) would have to be defined per subdomain. If it is desirable to support both BIER and BIER-TE forwarding in the same subdomain, then additional labels would need to be assigned for BIER-TE forwarding.

"forward_routed" requires an encapsulation permitting to unicast 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 (SI,subdomain) - and if necessary (see above) BIER-TE. 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).

3.4. Basic BIER-TE Forwarding Example

[RFC Editor: remove this section.]

THIS SECTION TO BE REMOVED IN RFC BECAUSE IT WAS SUPERCEEDED BY SECTION 1.1 EXAMPLE - UNLESS REVIEWERS CHIME IN AND EXPRESS DESIRE TO KEEP THIS ADDITIONAL EXAMPLE SECTION.

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

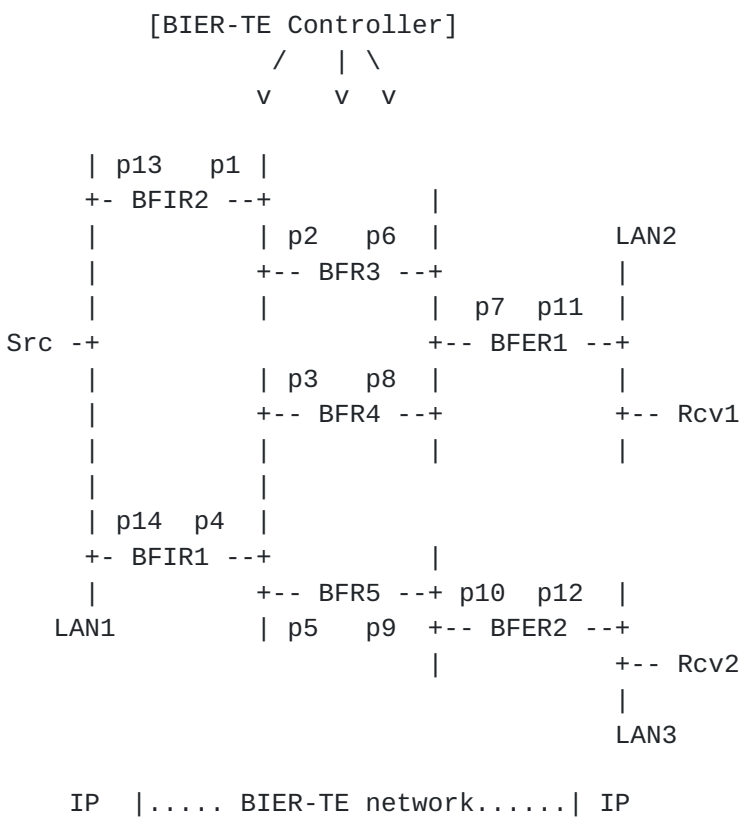


Figure 5: BIER-TE Forwarding Example

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 6: 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 7: 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 resets p2 in the BitString and sends a copy towards BFR2.

BFR3 sees a BitString of p5,p7,p8,p10,p11,p12. It is only interested in p1,p7,p8. It creates a copy of the packet to BFER1 (due to p7) and one to BFR4 (due to p8). It resets p7, p8 before sending.

BFER1 sees a BitString of p5,p10,p11,p12. It is only interested in p6,p7,p8,p11 and therefore considers only p11. p11 is a "local_decap" adjacency installed by the BIER-TE Controller because BFER1 should pass packets to IP multicast. The local_decap adjacency instructs BFER1 to create a copy, decapsulate it from the BIER header and pass it on to the NextProtocol, in this example 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.

3.5. Forwarding comparison with BIER

Forwarding of BIER-TE is designed to allow common forwarding hardware with BIER. In fact, one of the main goals of this document is to encourage the building of forwarding hardware that can not only support BIER, but also BIER-TE - to allow experimentation with BIER-TE and support building of BIER-TE control plane code.

The pseudocode in Section 6 shows how existing BIER/BIFT forwarding can be amended to support basic BIER-TE forwarding, by using BIER

BIFT's F-BM. Only the masking of bits due to avoid duplicates must be skipped when forwarding is for BIER-TE.

Whether to use BIER or BIER-TE forwarding can simply be a configured choice per subdomain and accordingly be set up by a BIER-TE Controller. The BIER packet encapsulation [RFC8296] too can be reused without changes except that the currently defined BIER-TE ECMP adjacency does not leverage the entropy field so that field would be unused when BIER-TE forwarding is used.

3.6. Requirements

Basic BIER-TE forwarding MUST support to configure Subdomains to use basic BIER-TE forwarding rules (instead of BIER). With basic BIER-TE forwarding, every bit MUST support to have zero or one adjacency. It MUST support the adjacency types forward_connected without DNR flag, forward_routed and local_decap. All other BIER-TE forwarding features are optional. These basic BIER-TE requirements make BIER-TE forwarding exactly the same as BIER forwarding with the exception of skipping the aforementioned F-BM masking on egress.

BIER-TE forwarding SHOULD support the DNR flag, as this is highly useful to save bits in rings (see Section 4.6).

BIER-TE forwarding MAY support more than one adjacency on a bit and ECMP adjacencies. The importance of ECMP adjacencies is unclear when traffic steering is used because it may be more desirable to explicitly steer traffic across non-ECMP paths to make per-path traffic calculation easier for BIER-TE Controllers. Having more than one adjacency for a bit allows further savings of bits in hub&spoke scenarios, but unlike rings it is less "natural" to flood traffic across multiple links unconditional. Both ECMP and multiple adjacencies are forwarding plane features that should be possible to support later when needed as they do not impact the basic BIER-TE replication loop. This is true because there is no inter-copy dependency through resetting of F-BM as in BIER.

4. BIER-TE Controller BitPosition Assignments

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

Because the size of the BitString is limiting the size of the BIER-TE domain, many of the options described exist to support larger topologies with fewer BitPositions (4.1, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8).

4.1. P2P Links

Each P2p link in the BIER-TE domain is assigned one unique BitPosition with a forward_connected adjacency pointing to the neighbor on the p2p link.

4.2. BFER

Every non-Leaf BFER is given a unique BitPosition with a local_decap adjacency.

4.3. Leaf BFERs

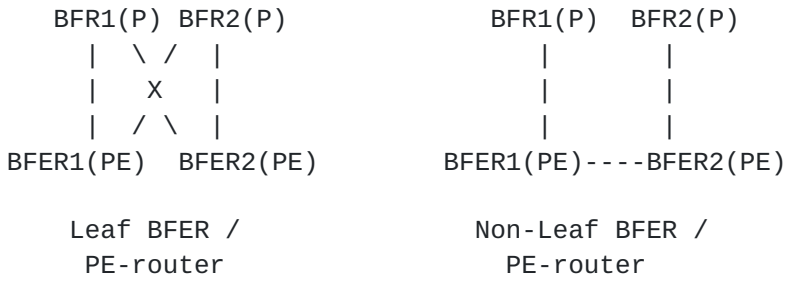


Figure 8: Leaf vs. non-Leaf BFER Example

Leaf BFERs are BFERs 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 PEs are spokes connected to 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 above picture: 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.

All leaf-BFER in a BIER-TE domain can share a single BitPosition. This is possible because the BitPosition for the adjacency to reach the BFER can be used to distinguish whether or not packets should reach the BFER.

This optimization will not work if an upstream interface of the BFER is using a BitPosition optimized as described in the following two sections (LAN, Hub and Spoke).

4.4. LANs

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

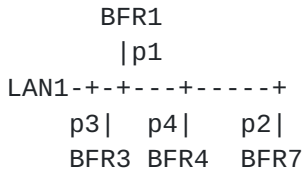


Figure 9: LAN Example

If Bandwidth on the LAN is not an issue and most BIER-TE traffic should be copied to all neighbors on a LAN, then BitPositions can be saved by assigning just a single BitPosition to the LAN and populating the BitPosition 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 LANs with this optimization because these BFRs would receive duplicates and forward those duplicates into the opposite LANs. Adjacencies of such BFRs into their LANs still need a separate BitPosition.

4.5. Hub and Spoke

In a setup with a hub and multiple spokes connected via separate p2p links to the hub, all p2p links can share the same BitPosition. The BitPosition 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 BitPosition optimization in LANs: Redundantly connected spokes need their own BitPositions.

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 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.

4.6. Rings

In L3 rings, instead of assigning a single BitPosition for every p2p link in the ring, it is possible to save BitPositions by setting the "Do Not Reset" (DNR) flag on forward_connected adjacencies.

For the rings shown in the following picture, a single BitPosition will suffice to forward traffic entering the ring at BFRa or BFRb all the way up to BFR1:

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

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

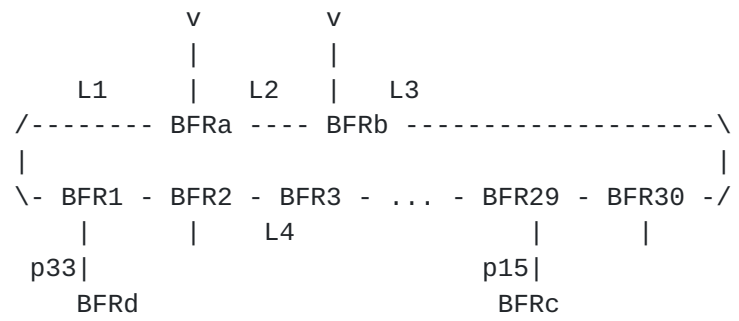


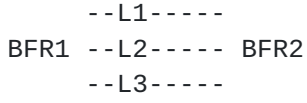
Figure 10: Ring Example

Note that this example only permits for packets to enter the ring 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 BitPositions. One for clockwise, one for 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 reset the counterclockwise BitPosition because the DNR bit only applies to the bit for which the replication is done. Likewise for the clockwise BitPosition for the counterclockwise copy. In result, the ring ingress BFR will send a copy in both directions, serving BFRs on either side of the ring up to L1.

4.7. Equal Cost MultiPath (ECMP)

The ECMP adjacency allows to use just one BP per link bundle between two BFRs instead of one BP for each p2p member link of that link bundle. In the following picture, one BP is used across L1,L2,L3.



BIFT entry in BFR1:

Index	Adjacencies
0:6	ECMP({forward_connected(L1, BFR2), forward_connected(L2, BFR2), forward_connected(L3, BFR2)}, seed)

BIFT entry in BFR2:

Index	Adjacencies
0:6	ECMP({forward_connected(L1, BFR1), forward_connected(L2, BFR1), forward_connected(L3, BFR1)}, seed)

Figure 11: ECMP Example

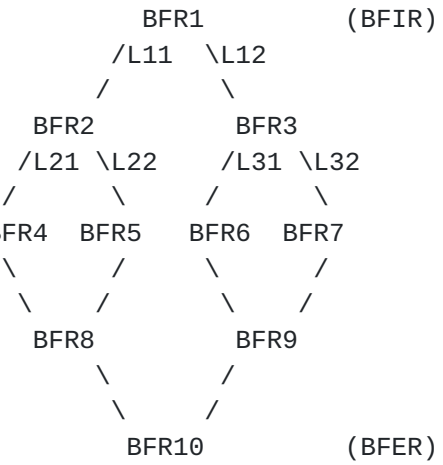
This document does not standardize any ECMP algorithm because it is sufficient for implementations to document their freely chosen ECMP algorithm. This allows the BIER-TE Controller to calculate ECMP paths and seeds. The following picture shows an example ECMP algorithm:

```
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 12: 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, 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 13: 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 4.9 below.

With the setup of ECMP in above topology, 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. The solutions chosen for BIER-TE to allow the BIER-TE Controller to explicitly set the seed maximizes the ability of the BIER-TE Controller to choose the seed, independent of such seed source that the BIER-TE Controller may not be able to control well, and even calculate optimized seeds for multi-hop cases.

4.8. Routed adjacencies

4.8.1. Reducing BitPositions

Routed adjacencies can reduce the number of BitPositions required when the path steering requirement is not hop-by-hop explicit path selection, but loose-hop selection. Routed adjacencies can also allow to operate BIER-TE across intermediate hop routers that do not support BIER-TE.

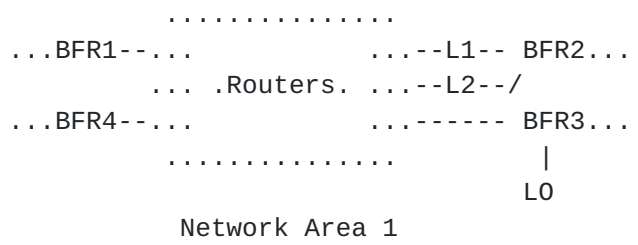


Figure 14: Routed Adjacencies Example

Assume the requirement in the above picture is to explicitly steer traffic flows that have arrived at BFR1 or BFR4 via a shortest 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, (3) via BFR3.

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

4.8.2. Supporting nodes without BIER-TE

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.

4.9. Reuse of BitPositions (without DNR)

BitPositions can be re-used across multiple BFR to minimize the number of BP needed. This happens when adjacencies on multiple BFR use the DNR flag as described above, but it can also be done for non-DNR adjacencies. This section only discussses this non-DNR case.

Because BP are reset after passing a BFR with an adjacency for that BP, reuse of BP across multiple BFR does not introduce any problems with duplicates or loops that do not also exist when every adjacency has a unique BP: Instead of setting one BP in a BitString that is reused in N-adjacencies, one would get the same or worse results if each of these adjacencies had a unique BP and all of them where set in the BitString. Instead, based on the case, BPs can be reused without limitation, or they introduce fewer path steering choices, or they do not work.

BP cannot be reused across two BFR that would need to be passed sequentially for some path: The first BFR will reset 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 13, where BP 0:7, BP 0:8 and BP 0:9 are each reused across multiple BFR 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.

Reuse may also save BPs in larger topologies. Consider the topology shown in Figure 17, but only the following explanations: 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 in before). These two BPs could be reused across the 5 areas. Packets would be replicated through other BPs 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.

4.10. 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 4.1).
- All leaf-BFER can share a single local_decap BP (Section 4.3).
- A LAN with N BFR needs at most N BP (one for each BFR). It only needs one BP for all those BFR tha are not redundanty connected to multiple LANs (Section 4.4).
- 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 4.5).
- 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 4.6).
- 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 4.7).
- 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 4.8).
- BP can generally be reused across nodes that do not need to be consecutive in paths, but depending on scenario, this may limit the feasible path steering options (Section 4.9).

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 BFER that should be able to receive traffic is limited, further optimizations of BP are possible. The hub & spoke optimization is a simple example of such traffic pattern dependent optimizations.

5. Avoiding duplicates and loops

5.1. Loops

Whenever BIER-TE creates a copy of a packet, the BitString of that copy will have all BitPositions cleared that are associated with

adjacencies on the BFR. This inhibits looping of packets. The only exception are adjacencies with DNR set.

With DNR set, looping can happen. Consider in the ring picture that link L4 from BFR3 is plugged into the L1 interface of BFRa. This creates a loop where the rings clockwise BitPosition is never reset 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 DNR 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 DNR flag set.

5.2. Duplicates

Duplicates happen when the topology of the BitString is not a tree but redundantly connecting BFRs with each other. The BIER-TE Controller must therefore ensure to only create BitStrings that are trees in the topology.

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.

6. BIER-TE Forwarding Pseudocode

The following simplified pseudocode for BIER-TE forwarding is using BIER forwarding pseudocode of [RFC8279], section 6.5 with the one modification necessary to support basic BIER-TE forwarding. Like the BIER pseudo forwarding code, for simplicity it does hide the details of the adjacency processing inside PacketSend() which can be forward_connected, forward_routed or local_decap.

```
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;
        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 15: Simplified BIER-TE Forwarding Pseudocode

The difference is that in BIER-TE, step [1] must not be performed, but is replaced with [2] (when the forwarding plane algorithm is implemented verbatim as shown above).

In BIER, the F-BM of a BP has all BP set that are meant to be forwarded via the same neighbor. It is used to reset those BP in the packet after the first copy to this neighbor has been made to inhibit multiple copies to the same neighbor.

In BIER-TE, the F-BM of a particular BP with an adjacency is the list of all BPs with an adjacency on this BFR except the particular BP itself if it has an adjacency with the DNR bit set. The F-BM is used to reset the F-BM BPs before creating copies.

In BIER, the order of BPs impacts the result of forwarding because of [1]. In BIER-TE, forwarding is not impacted by the order of BPs. It is therefore possible to further optimize forwarding than in BIER. For example, BIER-TE forwarding can be parallelized such that a parallel instance (such as an egres linecard) can process any subset of BPs without any considerations for the other BPs - and without any prior, cross-BP shared processing.

The above simplified pseudocode is elaborated further as follows:

- This pseudocode eliminates per-bit F-BM, therefore reducing state by BitStringLength^2*SI and eliminating the need for per-packet-copy masking operation except for adjacencies with DNR flag set:

    * AdjacentBits[SI] are bits with a non-empty list of adjacencies. This can be computed whenever the BIER-TE Controller updates the adjacencies.

    * Only the AdjacentBits need to be examined in the loop for packet copies.

    * The packets BitString is masked with those AdjacentBits on ingress to avoid packets looping.

- The code loops over the adjacencies because there may be more than one adjacency for a bit.

- When an adjacency has the DNR bit, the bit is set in the packet copy (to save bits in rings for example).

- The ECMP adjacency is shown. Its parameters are a ListOfAdjacencies from which one is picked.

- The forward_local, forward_routed, local_decap adjacencies are shown with their parameters.

```
void ForwardBitMaskPacket_withTE (Packet)
{
    SI=GetPacketSI(Packet);
    Offset=SI*BitStringLength;
    AdjacentBitstring = Packet->BitString &= ~AdjacentBits[SI];
    Packet->BitString &= AdjacentBits[SI];
    for (Index = GetFirstBitPosition(AdjacentBits); Index ;
         Index = GetNextBitPosition(AdjacentBits, Index)) {
        foreach adjacency BIFT[Index+Offset] {
            if(adjacency == ECMP(ListOfAdjacencies, seed) ) {
                I = ECMP_hash(sizeof(ListOfAdjacencies),
                              Packet->Entropy, seed);
                adjacency = ListOfAdjacencies[I];
            }
            PacketCopy = Copy(Packet);
            switch(adjacency) {
                case forward_connected(interface,neighbor,DNR):
                    if(DNR)
                        PacketCopy->BitString |= 2<<(Index-1);
                    SendToL2Unicast(PacketCopy,interface,neighbor);

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

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

Figure 16: BIER-TE Forwarding Pseudocode

7. Managing SI, subdomains and BFR-ids

When the number of bits required to represent the necessary hops in the topology and BFER exceeds the supported bitstring length, multiple SI and/or subdomains 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.

7.1. Why SI and sub-domains

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

For BIER and BIER-TE forwarding itself there is also no difference whether different SI and/or sub-domains are chosen, but SI and subdomain 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 subdomain 0 (unless there are > 64k BFIR/BFER).

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

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

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

To be able to easily reuse (and modify as little as possible) existing BIER procedures including flow-overlay and routing underlay, when BIER-TE forwarding is added, we therefore reuse SI and subdomain 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 SI as necessary (see below). Different services may use different subdomains that primarily exist to provide more efficient replication (and for BIER-TE desirable path steering) for different subsets of BFIR/BFER.

7.2. Bit assignment comparison BIER and BIER-TE

In BIER, bitstrings only need to carry bits for BFER, 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 steeering across every single hop (which requires more bits), or reducing the number of required bits by exploiting optimizations such as unicast (forward_route), ECMP or flood (DNR) 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 BFER in a BIFT:SI can range widely based on the size of the topology and the amount of alternative paths in it. The higher the percentage, 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.

7.3. Using BFR-id with BIER-TE

Because there is no 1:1 mapping between bits in the bitstring and BFER, BIER-TE cannot simply rely on the BIER 1:1 mapping between bits in a bitstring and BFR-id.

In BIER, automatic schemes could assign all possible BFR-ids sequentially to BFERs. This will not work in BIER-TE. In BIER-TE, the operator or BIER-TE Controller has to determine a BFR-id for each BFER in each required subdomain. The BFR-id may or may not have a relationship with a bit in the bitstring. Suggestions are detailed below. Once determined, the BFR-id can then be configured on the BFER and used by flow overlay, routing underlay and the BIER header almost the same as the BFR-id in BIER.

The one exception are application/flow-overlays that automatically calculate the bitstring(s) of BIER packets by converting BFR-id to bits. In BIER-TE, this operation can be done in two ways:

"Independent branches": For a given application or (set of) trees, the branches from a BFIR to every BFER are independent of the

branches to any other BFER. For example, shortest part trees have independent branches.

"Interdependent branches": When a BFER is added or deleted from a particular distribution tree, branches to other BFER still in the tree may need to change. Steiner tree are examples of dependent branch trees.

If "independent branches" are sufficient, the BIER-TE Controller can provide to such applications for every BFR-id a SI:bitstring with the BIER-TE bits for the branch towards that BFER. The application can then independently calculate the SI:bitstring for all desired BFER by OR'ing their bitstrings.

If "interdependent branches" are required, the application could call a BIER-TE Controller API with the list of required BFER-id and get the required bitstring back. Whenever the set of BFER-id changes, this is repeated.

Note that in either case (unlike in BIER), the bits in BIER-TE 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 bitstrings.

7.4. Assigning BFR-ids for BIER-TE

For a non-leaf BFER, there is usually a single bit k for that BFER with a local_decap() adjacency on the BFER. The BFR-id for such a BFER is therefore most easily the one it would have in BIER: SI * bitstring-length + k.

As explained earlier in the document, leaf BFERs do not need such a separate bit because the fact alone that the BIER-TE packet is forwarded to the leaf BFER indicates that the BFER should decapsulate it. Such a BFER will have one or more bits for the links leading only to it. The BFR-id could therefore most easily be the BFR-id derived from the lowest bit for those links.

These two rules are only recommendations for the operator or BIER-TE Controller assigning the BFR-ids. Any allocation scheme can be used, the BFR-ids just need to be unique across BFRs in each subdomain.

It is not currently determined if a single subdomain could or should be allowed to forward both 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 subdomain. 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 BIER-TE BFR-id.

B. BIER and BIER-TE share the same BFR-id. The BFR-id 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. Depending on topology, only the same 20%..80% of bits as possible for BIER-TE can be used for BIER.

7.5. Example bit allocations

7.5.1. With BIER

Consider a network setup with a bitstring length of 256 for a network topology as shown in the picture below. The network has 6 areas, each with ca. 170 BFR, connecting via a core with some larger (core) BFR. To address all BFER with BIER, 4 SI 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-id are allocated to BFER in the network, but for efficiency further down in the network it does make a difference.

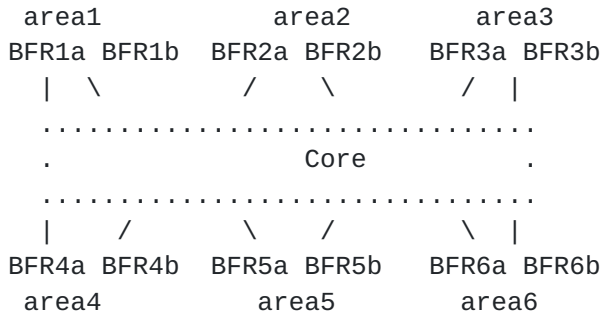


Figure 17: Scaling BIER-TE bits by reuse

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 SI 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-id are allocated intelligently, then all the BFER in an area would be given BFR-id with as few as possible different SI. 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 also easily go down over time when BFR-id are network wide allocated sequentially over time. An area that initially only has BFR-id in one SI might end up with many SI 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 BFR-id after network expansion. In this example one may consider to use 6 SI and assign one to each area.

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

7.5.2. With BIER-TE

In BIER-TE one needs to determine a subset of the physical topology and attached BFER 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 BFER, 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, random subsets of BFER are assigned to different SI. This is much worse than in BIER because it not only reduces replication efficiency with the same number of overall bits, but even further - because more bits are required due to duplication of bits for topology across multiple SI. Intelligent BFER to SI assignment and selecting specific "desired" subtopologies can minimize this problem.

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

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

In addition, we use 4 bits in each SI: bia, bib, bea, beb: bit ingress a, bit ingress b, bit egress a, bit egress 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 BFIR in an area j, 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 is populated with forward_routed(BFRka) and beb in BIFT:SI=k with forward_routed(BFRkb).

For BIER-TE forwarding of a packet to some subset of BFER 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 are 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.

7.6. Summary

BIER-TE can like BIER support multiple SI within a sub-domain to allow re-using the concept of BFR-id and therefore minimize BIER-TE specific functions in underlay routing, flow overlay methods and BIER headers.

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

Subdomains can in BIER-TE be used like in BIER to create more efficient replication to known subsets of BFER.

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

8. BIER-TE and Segment Routing

SR 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 BitPosition (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. 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 BitPositions for non-replicating hops, it is equally possible to use segment routing encapsulations (eg: MPLS label stacks) for the encapsulation of "forward_routed" adjacencies.

Note that 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 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.

Both BIER and BIER-TE allow BFIR to "opportunistically" copy packets to a set of desired BFER on a packet-by-packet basis. In BIER, this is done by OR'ing the BP for the desired BFER. In BIER-TE this can be done by OR'ing for each desired BFER a bitstring using the "independent branches" approach described in Section 7.3 and therefore also indicating the engineered path towards each desired BFER. This is the approach that [I-D.ietf-bier-multicast-http-response] relies on.

9. Security Considerations

The security considerations are the same as for BIER with the following differences:

BFR-ids and BFR-prefixes are not used in BIER-TE, nor are procedures for their distribution, so these are not attack vectors against BIER-TE.

10. IANA Considerations

This document requests no action by IANA.

11. Acknowledgements

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

12. Change log [RFC Editor: Please remove]

draft-ietf-bier-te-arch:

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

Added references for Bloom Filders 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: detailled why no need for BFER routing underlay routing protocol etensions, 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 http://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 subdomain 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 7 to explain the use of SI, subdomains 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.

13. References

13.1. Normative References

[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>.

13.2. Informative References

[Bloom70] Bloom, B., "Space/time trade-offs in hash coding with allowable errors", Comm. ACM 13(7):422-6, July 1970.

[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", draft-ietf-bier-multicast-http-response-04 (work in progress), July 2020.

[I-D.ietf-roll-ccast] Bergmann, O., Bormann, C., Gerdes, S., and H. Chen, "Constrained-Cast: Source-Routed Multicast for RPL", draft-ietf-roll-ccast-01 (work in progress), October 2017.

[I-D.ietf-teas-rfc3272bis] Farrel, A., "Overview and Principles of Internet Traffic Engineering", draft-ietf-teas-rfc3272bis-01 (work in progress), July 2020.

[ICC] Reed, M., 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>.

[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>.

Authors' Addresses

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

Email: tte+ietf@cs.fau.de

Gregory Cauchie
Bouygues Telecom

Email: GCAUCHIE@bouyguestelecom.fr

Michael Menth
University of Tuebingen

Email: menth@uni-tuebingen.de