

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: January 9, 2017

T. Eckert  
Cisco Systems, Inc.  
G. Cauchie  
Bouygues Telecom  
W. Braun  
M. Menth  
University of Tuebingen  
July 8, 2016

**Traffic Engineering for Bit Index Explicit Replication BIER-TE**  
**draft-eckert-bier-te-arch-04**

**Abstract**

This document proposes an architecture for BIER-TE: Traffic Engineering for Bit Index Explicit Replication (BIER).

BIER-TE shares part of its architecture with BIER as described in [[I-D.ietf-bier-architecture](#)]. It also proposes to share the packet format with BIER.

BIER-TE forwards and replicates packets like BIER based on a BitString in the packet header but it does not require an IGP. It does support traffic engineering by explicit hop-by-hop forwarding and loose hop forwarding of packets. It does support Fast ReRoute (FRR) for link and node protection and incremental deployment. Because BIER-TE like BIER operates without explicit in-network tree-building but also supports traffic engineering, it is more similar to SR than RSVP-TE.

**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 <http://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 January 9, 2017.

## Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](http://trustee.ietf.org/license-info) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

<a href="#">1.</a>	<a href="#">Introduction</a>	<a href="#">3</a>
<a href="#">1.1.</a>	<a href="#">Overview</a>	<a href="#">3</a>
<a href="#">1.2.</a>	<a href="#">Requirements Language</a>	<a href="#">4</a>
<a href="#">2.</a>	<a href="#">Layering</a>	<a href="#">4</a>
<a href="#">2.1.</a>	<a href="#">The Multicast Flow Overlay</a>	<a href="#">5</a>
<a href="#">2.2.</a>	<a href="#">The BIER-TE Controller Host</a>	<a href="#">5</a>
<a href="#">2.2.1.</a>	<a href="#">Assignment of BitPositions to adjacencies of the network topology</a>	<a href="#">6</a>
<a href="#">2.2.2.</a>	<a href="#">Changes in the network topology</a>	<a href="#">6</a>
<a href="#">2.2.3.</a>	<a href="#">Set up per-multicast flow BIER-TE state</a>	<a href="#">6</a>
<a href="#">2.2.4.</a>	<a href="#">Link/Node Failures and Recovery</a>	<a href="#">6</a>
<a href="#">2.3.</a>	<a href="#">The BIER-TE Forwarding Layer</a>	<a href="#">7</a>
<a href="#">2.4.</a>	<a href="#">The Routing Underlay</a>	<a href="#">7</a>
<a href="#">3.</a>	<a href="#">BIER-TE Forwarding</a>	<a href="#">7</a>
<a href="#">3.1.</a>	<a href="#">The Bit Index Forwarding Table (BIFT)</a>	<a href="#">7</a>
<a href="#">3.2.</a>	<a href="#">Adjacency Types</a>	<a href="#">8</a>
<a href="#">3.2.1.</a>	<a href="#">Forward Connected</a>	<a href="#">8</a>
<a href="#">3.2.2.</a>	<a href="#">Forward Routed</a>	<a href="#">9</a>
<a href="#">3.2.3.</a>	<a href="#">ECMP</a>	<a href="#">9</a>
<a href="#">3.2.4.</a>	<a href="#">Local Decap</a>	<a href="#">9</a>
<a href="#">3.3.</a>	<a href="#">Encapsulation considerations</a>	<a href="#">10</a>
<a href="#">3.4.</a>	<a href="#">Basic BIER-TE Forwarding Example</a>	<a href="#">10</a>
<a href="#">4.</a>	<a href="#">BIER-TE Controller Host BitPosition Assignments</a>	<a href="#">12</a>
<a href="#">4.1.</a>	<a href="#">P2P Links</a>	<a href="#">12</a>
<a href="#">4.2.</a>	<a href="#">BFER</a>	<a href="#">13</a>
<a href="#">4.3.</a>	<a href="#">Leaf BFERs</a>	<a href="#">13</a>
<a href="#">4.4.</a>	<a href="#">LANs</a>	<a href="#">13</a>
<a href="#">4.5.</a>	<a href="#">Hub and Spoke</a>	<a href="#">14</a>
<a href="#">4.6.</a>	<a href="#">Rings</a>	<a href="#">14</a>
<a href="#">4.7.</a>	<a href="#">Equal Cost MultiPath (ECMP)</a>	<a href="#">15</a>
<a href="#">4.8.</a>	<a href="#">Routed adjacencies</a>	<a href="#">17</a>



<a href="#">4.8.1.</a>	Reducing BitPositions . . . . .	<a href="#">17</a>
<a href="#">4.8.2.</a>	Supporting nodes without BIER-TE . . . . .	<a href="#">17</a>
<a href="#">5.</a>	Avoiding loops and duplicates . . . . .	<a href="#">17</a>
<a href="#">5.1.</a>	Loops . . . . .	<a href="#">17</a>
<a href="#">5.2.</a>	Duplicates . . . . .	<a href="#">18</a>
<a href="#">6.</a>	BIER-TE Forwarding Pseudocode . . . . .	<a href="#">18</a>
<a href="#">7.</a>	Managing SI, subdomains and BFR-ids . . . . .	<a href="#">19</a>
<a href="#">7.1.</a>	Why SI and sub-domains . . . . .	<a href="#">20</a>
<a href="#">7.2.</a>	Bit assignment comparison BIER and BIER-TE . . . . .	<a href="#">21</a>
<a href="#">7.3.</a>	Using BFR-id with BIER-TE . . . . .	<a href="#">21</a>
<a href="#">7.4.</a>	Assigning BFR-ids for BIER-TE . . . . .	<a href="#">22</a>
<a href="#">7.5.</a>	Example bit allocations . . . . .	<a href="#">23</a>
<a href="#">7.5.1.</a>	With BIER . . . . .	<a href="#">23</a>
<a href="#">7.5.2.</a>	With BIER-TE . . . . .	<a href="#">24</a>
<a href="#">7.6.</a>	Summary . . . . .	<a href="#">25</a>
<a href="#">8.</a>	BIER-TE and Segment Routing . . . . .	<a href="#">25</a>
<a href="#">9.</a>	Security Considerations . . . . .	<a href="#">26</a>
<a href="#">10.</a>	IANA Considerations . . . . .	<a href="#">26</a>
<a href="#">11.</a>	Acknowledgements . . . . .	<a href="#">26</a>
<a href="#">12.</a>	Change log [RFC Editor: Please remove] . . . . .	<a href="#">26</a>
<a href="#">13.</a>	References . . . . .	<a href="#">27</a>
	Authors' Addresses . . . . .	<a href="#">28</a>

## [1.](#) Introduction

### [1.1.](#) Overview

This document specifies the architecture for BIER-TE: traffic engineering for Bit Index Explicit Replication BIER.

BIER-TE shares architecture and packet formats with BIER as described in [[I-D.ietf-bier-architecture](#)].

BIER-TE forwards and replicates packets like BIER based on a BitString in the packet header but it does not require an IGP. It does support traffic engineering by explicit hop-by-hop forwarding and loose hop forwarding of packets. It does support incremental deployment and a Fast ReRoute (FRR) extension for link and node protection is given in [[I-D.eckert-bier-te-frr](#)]. Because BIER-TE like BIER operates without explicit in-network tree-building but also supports traffic engineering, it is more similar to Segment Routing (SR) than RSVP-TE.

The key differences over BIER are:

- o BIER-TE replaces in-network autonomous path calculation by explicit paths calculated offpath by the BIER-TE controller host.



- o 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.
- o 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.

## **1.2. Requirements Language**

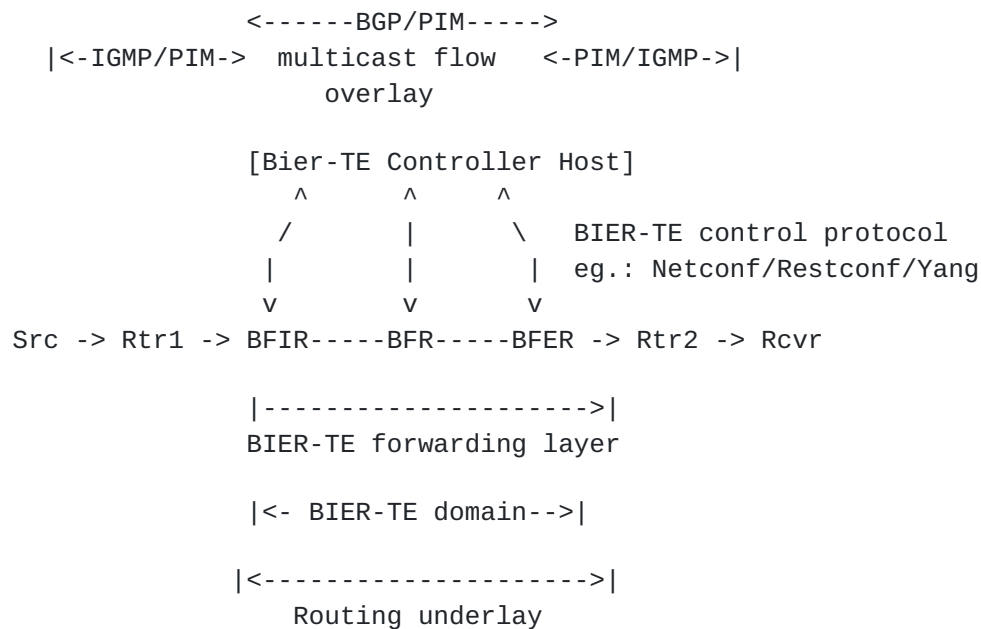
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#) [[RFC2119](#)].

## **2. Layering**

End to end BIER-TE operations consists of four components: The "Multicast Flow Overlay", the "BIER-TE Controller Host", the "Routing Underlay" and the "BIER-TE forwarding layer".



Picture 2: Layers of BIER-TE



### 2.1. The Multicast Flow Overlay

The Multicast Flow Overlay operates as in BIER. See [\[I-D.ietf-bier-architecture\]](#). Instead of interacting with the BIER layer, it interacts with the BIER-TE Controller Host

### 2.2. The BIER-TE Controller Host

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

During bring-up or modifications of the network topology, the controller discovers the network topology, assigns BitPositions to adjacencies and signals the resulting mapping of BitPositions to adjacencies to each BFR connecting to the adjacency.

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

Communications between the BIER-TE controller host to BFRs is ideally via standardized protocols and data-models such as Netconf/Retconf/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 host 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 host is currently outside the scope of this document.

#### **2.2.1. Assignment of BitPositions to adjacencies of the network topology**

The BIER-TE controller host 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 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 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 host tracks 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 (eg.: CSFP, Steiner Tree,...). It then pushes the calculated BitString into the BFIR.

#### **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 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 host. 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 BitPositions in the BitString of the packet to which it can create a copy. This is done to inhibit that packets can loop.

### **2.4. The Routing Underlay**

BIER-TE is sending BIER packets to directly connected BIER-TE neighbors as L2 (unicasted) BIER packets without requiring a routing underlay. BIER-TE forwarding uses the Routing underlay for forward\_routed adjacencies which copy BIER-TE packets to not-directly-connected BFRs (see below for adjacency definitions).

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, eg.: from BFD. Providing these notifications is considered to be part of the routing underlay in this document.

## **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 (traffic 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.

Index:	Adjacencies:	
SI:BitPosition	<empty> or one or more per entry	
=====		
0:1	forward_connected(interface,neighbor,DNR)	
0:2	forward_connected(interface,neighbor,DNR)	
	forward_connected(interface,neighbor,DNR)	
-----		
0:3	local_decap([VRF])	
-----		
0:4	forward_routed([VRF,]l3-neighbor)	
-----		
0:5	<empty>	
-----		
0:6	ECMP({adjacency1,...adjacencyN}, seed)	
-----		
...		
BitStringLength	...	
-----		

Bit Index Forwarding Table

The BIFT is programmed into the data plane of BFRs by the BIER-TE controller host 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 controller do not have to have the same adjacencies. This is up to the controller. BPs for p2p links are one case (see below).

## [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. The 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.

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 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 engineering traffic 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 [[I-D.ietf-bier-mpls-encapsulation](#)] 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 tunneling (IP in IP, LISP, GRE) would be required.

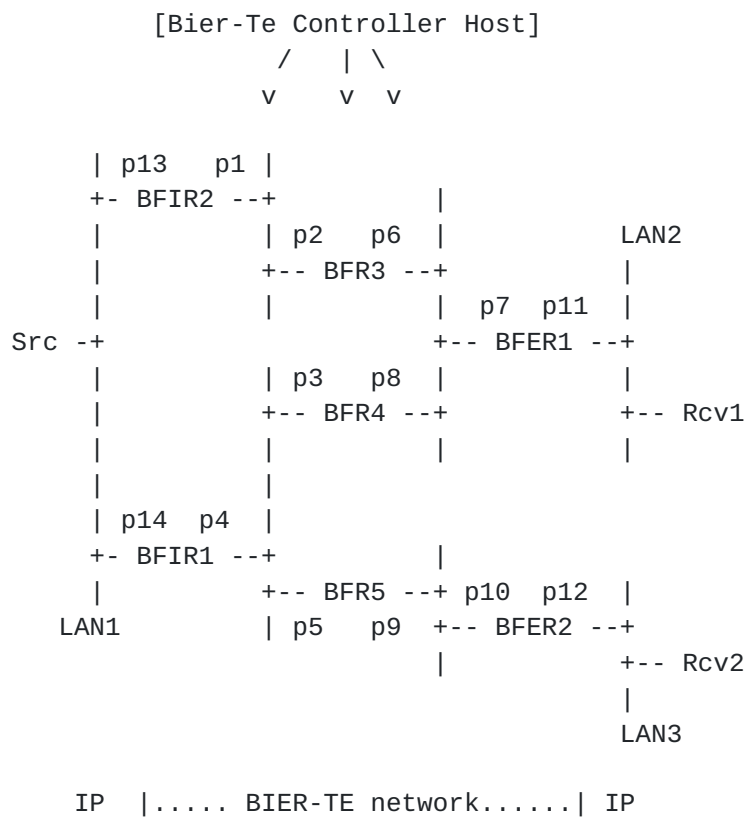
The encapsulation used for "forward\_routed" adjacencies can equally support existing advanced adjacency information such as "loose source routes" via eg: MPLS label stacks or appropriate header extensions (eg: for IPv6).

### **3.4. Basic BIER-TE Forwarding Example**

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.



### Picture 1: Forwarding Example



pXX indicate the BitPositions number assigned by the BIER-TE controller host to adjacencies in the BIER-TE topology. For example, p9 is the adjacency towards BFR9 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)
```

...and so on.

Traffic needs to flow from BFIR2 towards Rcv1, Rcv2. The controller determines it wants it to pass across the following paths:



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

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

This BitString is set up in BFIR2. Multicast packets arriving at BFIR2 from Src are assigned this BitString.

BFIR2 forwards based on that BitString. It has p2 and p13 populated. Only p13 is in BitString which has an adjacency towards BFR3. BFIR2 resets p2 in 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 host 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.

#### **4. BIER-TE Controller Host BitPosition Assignments**

This section describes how the BIER-TE controller host 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 BFER is given a unique BitPosition with a local\_decap adjacency.

#### [4.3.](#) Leaf BFERs

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.

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.

```

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

```

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 face 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 hubs 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.

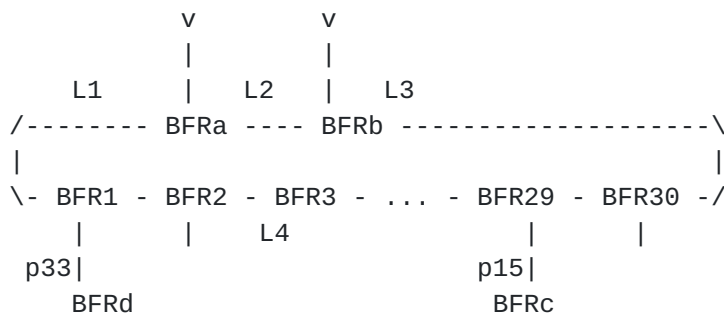
#### 4.6. Rings

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.



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, eg: 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 and BFR1/BFR2 have for the BP

```

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

```

BIFT entry in BFR1:

```

-----
| Index | Adjacencies                                     |
=====
| 0:6   | ECMP({L1-to-BFR2,L2-to-BFR2,L3-to-BFR2}, seed) |
-----

```

BIFT entry in BFR2:

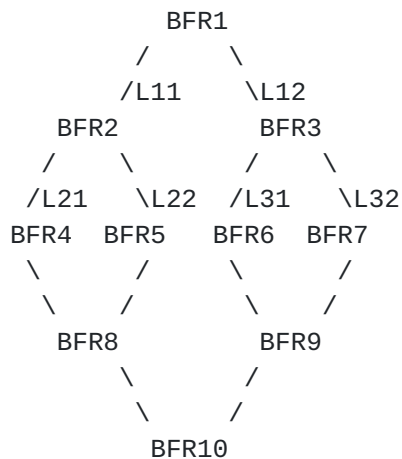
```

-----
| Index | Adjacencies                                     |
=====
| 0:6   | ECMP({L1-to-BFR1,L2-to-BFR1,L3-to-BFR1}, seed) |
-----

```

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 mean 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({L11-to-BFR2,L12-to-BFR3}, seed)  |
-----

```

BIFT entry in BFR2:

```

-----
| 0:6   |  ECMP({L21-to-BFR4,L22-to-BFR5}, seed)  |
-----

```

BIFT entry in BFR3:

```

-----
| 0:6   |  ECMP({L31-to-BFR6,L32-to-BFR7}, seed)  |
-----

```

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 a list of 2 adjacencies: link L11-to-BFR2. When forwarding in BFR2 performs again an ECMP with two adjacencies on that subset of traffic, then it will again select the first of its two adjacencies to it: L21-to-BFR4. And therefore L22 and BFR5 sees no traffic.

To resolve this issue, the ECMP adjacency on BFR1 simply needs to be set up with a different seed than the ECMP adjacencies on BFR2/BFR3

This issue is called polarization. It depends on the ECMP hash. It is possible to build ECMP that does not have polarization, for example by taking entropy from the actual adjacency members into account, but that can make it harder to achieve evenly balanced load-splitting on all BFR without making the ECMP hash algorithm potentially too complex for fast forwarding in the BFRs.



## 4.8. Routed adjacencies

### 4.8.1. Reducing BitPositions

Routed adjacencies can reduce the number of BitPositions required when the traffic engineering requirement is not hop-by-hop explicit path selection, but loose-hop selection.

```

.....
BFR1--... Redundant ...--L1-- BFR2... Redundant ...--
 \--... Network  ....-L2--/   ... Network  ....
BFR4--... Segment 1 ....-L3-- BFR3... Segment 2 ....
.....

```

Assume the requirement in above network is to explicitly engineer paths such that specific traffic flows are passed from segment 1 to segment 2 via link L1 (or via L2 or via L3).

To achieve this, BFR1 and BFR4 are set up with a forward\_routed adjacency BitPosition towards an address of BFR2 on link L1 (or link L2 BFR3 via L3).

For paths to be engineered through a specific node BFR2 (or BFR3), BFR1 and BFR4 are set up with a forward\_routed adjacency BitPosition towards a loopback address of BFR2 (or 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.

## 5. Avoiding loops and duplicates

### 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 in 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 destination address of the adjacency (eg.: 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 connects BFRs with each other. The controller must therefore ensure to only create BitStrings that are trees in the topology.

When links are incorrectly physically re-connected before the 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 controller.

## 6. BIER-TE Forwarding Pseudocode

The following sections of Pseudocode are meant to illustrate the BIER-TE forwarding plane. This code is not meant to be normative but to serve both as a potentially easier to read and more precise representation of the forwarding functionality and to illustrate how simple BIER-TE forwarding is and that it can be efficiently be implemented.

The following procedure is executed on a BFR whenever the BIFT is changed by the BIER-TE controller host:

```
global MyBitsOfInterest

void BIFTChanged()
{
    for (Index = 0; Index++ ; Index <= BitStringLength)
        if(BIFT[Index] != <empty>)
            MyBitsOfInterest |= 2<<(Index-1)
}
```

The following procedure is executed whenever a BIER-TE packet is to be forwarded:



```

void ForwardBierTePacket (Packet)
{
    // We calculate in BitMask the subset of BPs of the BitString
    // for which we have adjacencies. This is purely an
    // optimization to avoid to replicate for every BP
    // set in BitString only to discover that for most of them,
    // the BIFT has no adjacency.

    local BitMask = Packet->BitString
    Packet->BitString &= ~MyBitsOfInterest
    BitMask &= MyBitsOfInterest

    // Replication
    for (Index = GetFirstBitPosition(BitMask); Index ;
        Index = GetNextBitPosition(BitMask, Index))
        foreach adjacency BIFT[Index]

            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)
}

```

## 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-id 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) bitstring. 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  $\text{ceiling}(N/\text{bitstring-length})$  SI.

If service instances justify additional BIER:SI state in the network, additional subdomains will be used: BFIR/BFER are assigned BFR-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 traffic engineering) 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 lead 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 traffic engineering 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 - eg: parts where different trees do not need to take different paths due to traffic-engineering reasons.

The total number of bits to describe the topology in a BIFT:SI can therefore easily be as low as 20% or as high as 80%. The higher the percentage, the higher the likelihood, that those topology bits are not just BIER-TE overhead without additional benefit, but instead they will allow to express the desired traffic-engineering 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 can not 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 host 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 path 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 host 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 host 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 load by other traffic (as part of traffic engineering goals). Interactions between such BFIR applications and the BIER-TE controller host do therefore need to support dynamic updates to the bitstrings.

#### **7.4. Assigning BFR-ids for BIER-TE**

For 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:  $SI * \text{bitstring-length} + k$ .

As explained earlier in the document, leaf BFER 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:



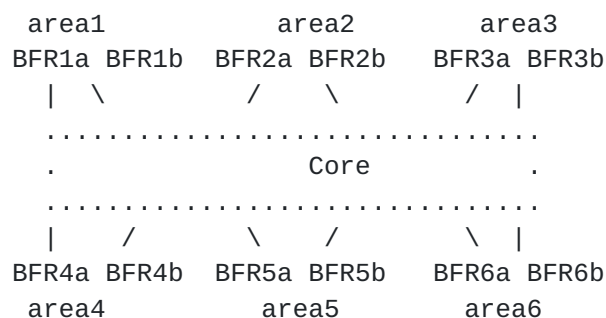
A. BIER and BIER-TE have different BFR-id in the same subdomain. This allows higher replication efficiency for BIER because their BFR-id can be assigned sequentially, while the bitstrings for BIER-TE will have also the additional bits for the topology. There is no relationship between a BFR BIER BFR-id 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. 180 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.



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 engineering 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.

### **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**

Segment Routing aims to achieve lightweight path engineering via loose source routing. Compared for example to RSVP-TE, it does 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.

Instead of defining BitPositions for non-replicating hops, it is equally possible to use segment routing encapsulations (eg: MPLS label stacks) for "forward\_routed" adjacencies.





## **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 and Neale Ranns for their extensive review and suggestions.

## **12. Change log [RFC Editor: Please remove]**

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 host and CLI.

00: Initial version.

## **[13.](#) References**

[I-D.ietf-bier-architecture]

Wijnands, I., Rosen, E., Dolganow, A., Przygienda, T., and S. Aldrin, "Multicast using Bit Index Explicit Replication", [draft-ietf-bier-architecture-03](#) (work in progress), January 2016.



[I-D.ietf-bier-mpls-encapsulation]

Wijnands, I., Rosen, E., Dolganow, A., Tantsura, J., and  
S. Aldrin, "Encapsulation for Bit Index Explicit  
Replication in MPLS Networks", [draft-ietf-bier-mpls-  
encapsulation-04](#) (work in progress), April 2016.

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

Authors' Addresses

Toerless Eckert  
Cisco Systems, Inc.

Email: [eckert@cisco.com](mailto:eckert@cisco.com)

Gregory Cauchie  
Bouygues Telecom

Email: [GCAUCHIE@bouyguestelecom.fr](mailto:GCAUCHIE@bouyguestelecom.fr)

Wolfgang Braun  
University of Tuebingen

Email: [wolfgang.braun@uni-tuebingen.de](mailto:wolfgang.braun@uni-tuebingen.de)

Michael Menth  
University of Tuebingen

Email: [menth@uni-tuebingen.de](mailto:menth@uni-tuebingen.de)

