Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2015

T. Eckert
Cisco

March 5, 2015

Traffic Enginering for Bit Index Explicit Replication BIER-TE draft-eckert-bier-te-arch-00

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.wijnands-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

Copyright Notice

Table of Contents

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.wijnands-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 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.

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 BitPosition and populated with only those adjacencies to which the BFR should replicate packets to.

Currently, BIER-TE does not support BIER-sub-domains and it does not not use BFR-id or "Set Identifiers" (SI) in BIER-TE headers that share the same format as BIER headers.

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

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

                 [Bier-TE Controller Host]
                    ^      ^     ^
                   /       |      \   BIER-TE control protocol
                  |        |       |  eg.: Netconf/Restconf/Yang
                  v        v       v
    Src -> Rtr1 -> BFIR-----BFR-----BFER -> Rtr2 -> Rcvr

                 |---------------------->|
                 BIER-TE forwarding layer

                 |<- BIER-TE domain-->|

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

2.1. The Multicast Flow Overlay

The Multicast Flow Overlay operates as in BIER. See [I-D.wijnands-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 an offpath central host. It communicates via protocols such as Netconf/Restconf/Yang with BFRs. The protocols used between BFRs and the controller are outside the

scope of this document. This document is only concerned about the logic how a controller can assign BitPositions to the topology and BitStrings to BIER-TE packets:

During bring-up or modifications of the network topology, the controller needs to talk to all BFRs to assign BitPositions to adjacencies of the network topology. During day-to-day operations of the network it only needs to talks to BFIRs to install BitStrings for multicast flows.

These two tasks have the following steps:

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 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 (and BTAFT if FRR is supported, see below), then they can be repopulated, first into BIFT (and if FRR is supported BTAFT), then into 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 below. It can also more slowly react by recalculating the BitStrings of affected multicast flows. This reaction is slower than the FR 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.

If the BFR support BIER-TE FRR operations, then the BIER-TE forwarding layer will receive fast adjacency up/down notification uses the BIER-TE FRR Adjacency Table to modify the BitString of the packet before it performs BIER-TE forwarding. This is detailed in the FRR section.

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. It is a table indexed by BitPosition and is populated by the BIER-TE control plane. Each index can be empty or contain a list of one or more adjacencies.

Index	Adjacencies
1	forward_connected(interface,neighbor,DNR)
2	forward_connected(interface,neighbor,DNR) forward_connected(interface,neighbor,DNR)
3	local_decap([VRF])
4	forward_routed([VRF,]l3-neighbor)
5	<empty>
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

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. 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

```
               [Bier-Te Controller Host]
                       /   | \
                      v    v  v

           | p13   p1 |
           +- BFIR2 --+          |
           |          | p2   p6  |           LAN2
           |          +-- BFR3 --+           |
           |          |          |  p7  p11  |
      Src -+                     +-- BFER1 --+
           |          | p3   p8  |           |
           |          +-- BFR4 --+           +-- Rcv1
           |          |          |           |
           |          |
           | p14  p4  |
           +- BFIR1 --+          |
           |          +-- BFR5 --+ p10  p12  |
         LAN1         | p5   p9  +-- BFER2 --+
                                 |           +-- Rcv2
                                             |
                                             LAN3

          IP  |..... BIER-TE network......| IP
```

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 BFIRs

Leaf BFIRs are BFIRs where incoming BIER-TE packets never need to be forwarded to another BFR but are only sent to the BFIR to exit the BIER-TE domain. For example, in networks where PEs are spokes connected to P routers, those PEs are Leaf BFIRs unless there is a U-turn between two PEs.

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

This optimization will not work if an upstream interface of the BFIR 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 this BitPosition, therefore minimizing the chance to create loops.

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

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
6	ECMP({L1-to-BFR2,L2-to-BFR2,L3-to-BFR2}, seed)

BIFT entry in BFR2:

Index	Adjacencies
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.

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

     BIFT entry in BFR1:
     ------------------------------------------------------------------
     | 6     |  ECMP({L11-to-BFR2,L12-to-BFR3}, seed)                 |
     ------------------------------------------------------------------

     BIFT entry in BFR2:
     ------------------------------------------------------------------
     | 6     |  ECMP({L21-to-BFR4,L22-to-BFR5}, seed)                 |
     ------------------------------------------------------------------

     BIFT entry in BFR3:
     ------------------------------------------------------------------
     | 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 adjaceny on BFR1 simply needs to be set up with a different seed than the ECMP adjacncies 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

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 he 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 up with a forward_routed adjacency BitPosition towards a loopback address of BFR2 (or BFR3).

4.8.1. 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 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 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. FRR

FRR is an optional procedure. To leverage it, the BIER-TE controller host and BFRs need to support it. It does not have to be supported on all BFRs, but only those that are attached to a link/adjacency for which FRR support is required.

If BIER-TE FRR is supported by the BIER-TE controller host, then it needs to calculate the desired backup paths for link and/or node failures in the BIER-TE domain and download this information into the BIER-TE Adjacency FRR Table (BTAFT) of the BFRs. The BTAFT then drives FRR operations in the BIER-TE forwarding plane of that BFR.

6.1. The BIER-TE Adjacency FRR Table (BTAFT)

The BIER-TE IF FRR Table exists in every BFR that is supporting BIER-TE FRR procedures. It is indexed by FRR Adjacency Index. Associated with each FRR Adjacency Index is a ResetBitmask, AddBitmask and BitPosition.

FRR Adjacency Index	BitPosition	ResetBitmask	AddBitmask
1	5	..0010000	..11000000
...			

An FRR Adjacency is an adjacency that is used in the BIFT of the BFR. The BFR has to be able to determine whether the adjacency is up or down in less than 50msec. An FRR adjacency can be a forward_connected adjacency with fast L2 link state Up/Down state notifications or a forward_connected or forward_routed adjacency with a fast aliveness mechanism such as BFD. Details of those mechanism are outside the scope of this architecture.

The FRR Adjacency Index is the index that would be indicated on the fast Up/Down notifications to the BIER-TE forwarding plane

The BitPosition is the BP in the BIFT in which the FRR Adjacency is used

6.2. FRR in BIER-TE forwarding

The BIER-TE forwarding plane receives fast Up/Down notifications with the FRR Adjacency Index. From the BitPosition in the BTAFT entry, it remembers which BPs are currently affected (have a down adjacency).

When a packet is received, BIER-TE forwarding checks if it has affected BPs to which it would forward. If it does, it will remove the ResetBitmask bits from the packets BitString and add the AddBitmask bits to the packets BitString.

Afterwards, normal BIER-TE forwarding occurs, taking the modified BitString into account.

6.3. FRR in the BIER-TE Controller Host

The basic rules how the BIER-TE controller host would calculate ResetBitMask and AddBitmask are as follows:

1. The BIER-TE controller host has to determine whether a failure of the adjacency should be taken to indicate link or node failure. This is a policy decision.

2. The ResetBitmask has the BitPosition of the failed adjacency.

3. In the case of link protection, the AddBitmask are the segments forming a path from the BFR over to the BFR on the other end of the failed link.

4. In the case of node protection, the AddBitmask are the segments forming a tree from the BFR over to all necessary BFR downstream of the (assumed to be failed) BFR across the failed adjacency.

5. The ResetBitmask is extended with those segments that could lead to duplicate packets if the AddBitmask is added to possible BitStrings of packets using the failing BitPosition.

6.4. BIER-TE FRR Benefits

Compared to other FRR solutions, such as RSVP-TE/P2MP FRR, BIER-TE FRR has two key distinctions

- It maintains the goal of BIER-TE not to establish in-network per multicast traffic flow state. For that reason, the backup path/ trees are only tied to the topology but not to individual distribution trees.

- For the case of node failure, it allows to build a path engineered backup tree (4.) as opposed to only a set of p2p backup tunnels.

7. 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 an adjacency used for BIER-TE FRR changes state:

```
      global ResetBitMaskByBT[BitStringLength]
      global AddtBitMaskByBT[BitStringLength]
      global FRRaffectedBP

      void FrrUpDown(FrrAdjacencyIndex, UpDown)
      {
          global FRRAdjacenciesDown
          local Idx = FrrAdjacencyIndex

          if (UpDown == Up)
              FRRAdjacenciesDown &= ~ 2<<(FrrAdjacencyIndex-1)
          else
              FRRAdjacenciesDown |=   2<<(FrrAdjacencyIndex-1)

          for (Index = GetFirstBitPosition(FRRAdjacenciesDown); Index ;
              Index = GetNextBitPosition(FRRAdjacenciesDown, Index))

              local BP = BTAFT[Index].BitPosition
              FRRaffectedBP |= 2<<(Index)
              ResetBitMaskByBT[BP] |= BTAFT[Index].ResetBitMask
              AddBitMaskByBT[BP]   |= BTAFT[Index].AddBitMask
      }
```

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

          // FRR Operations
          // Note: this algorithm is not optimal yet for ECMP cases
          // it performs FRR replacement for all candidate ECMP paths

          local MyFRRBP = BitMask & FRRaffectedBP
          for (BP = GetFirstBitPosition(MyFRRNP); BP ;
               BP = GetNextBitPosition(MyFRRNP, BP))
              BitMask &= ~ResetBitMaskByBT[BP]
              BitMask |=  ResetBitMaskByBT[BT]

          // 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)
      }
```

8. 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.

9. IANA Considerations

This document requests no action by IANA.

10. Acknowledgements

The author would like to thank Ijsbrand Wijnands and Neale Ranns for their extensive review and suggestions.

11. Change log [RFC Editor: Please remove]

00: Initial version.

12. References

[I-D.wijnands-bier-architecture]
Wijnands, I., Rosen, E., Dolganow, A., Przygienda, T., and S. Aldrin, "Multicast using Bit Index Explicit Replication", draft-wijnands-bier-architecture-04 (work in progress), February 2015.

[I-D.wijnands-mpls-bier-encapsulation]
Wijnands, I., Rosen, E., Dolganow, A., Tantsura, J., and S. Aldrin, "Encapsulation for Bit Index Explicit Replication in MPLS Networks", draft-wijnands-mpls-bier-encapsulation-02 (work in progress), December 2014.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

Author's Address

Toerless Eckert
Cisco

Email: eckert@cisco.com