Experimenting with Bit Index Explicit Replication

Anonymous Author(s)

ABSTRACT

Bit Index Explicit Replication (BIER) is a recent multicast architecture that solves several problems with deployed IP multicast protocols. BIER embeds an implicit multicast tree representation inside each transmitted packet. With this new mechanism, it becomes possible to reconsider multicast applications. However, no open-source implementation of BIER can be found. This paper presents our open-source implementation of the BIER forwarding mechanism and a companion socket-like API. Additionally, we show simulations of the implementation with ns-3 DCE.

1 INTRODUCTION

Multicast protocols enable efficient one-to-many communication as packets are duplicated in the network only when needed, thus ensuring that at most a single copy of a same packet will reach each router. IP Multicast, the current deployment of this architecture, suffers from scalability issues [3]. One of them concerns the states that each node of a multicast tree must keep as the tree is explicitly built. This memory consumption linearly grows with the number of multicast groups in a network, thus becoming a problem for large-scale multicast deployments.

The IETF standardized the Bit Index Explicit Replication (BIER) [7], a new source-routed multicast transmission mechanism. Each BIER packet embeds a *bitstring*, where each bit uniquely represents an egress router in the network. A bit set indicates that the corresponding router should receive the packet. Based on the *bitstring*, forwarding nodes rely on their Bit Index Forwarding Table (BIFT) to choose the output interface(s) for the packet (and its potential copies), thus implicitly building the multicast tree. The table is constructed statically or following the Interior Gateway Protocol (IGP). In opposition to IP multicast, the intermediate routers need constant memory state, independently of the number of multicast groups in the network.

BIER opens-up possibilities for easier deployment of multicast applications. New transport protocols can be built atop this mechanism and benefit from its source-routed architecture. Other research problems could also be reconsidered, such as reliable multicast and multicast congestion control algorithms using the traffic-engineering extensions for BIER (BIER-TE) [4]. Recent works already explore that direction. However, they either rely on simulation models [2], P4 designs [6], or non-open-source implementation [5].

Open-source implementations of protocols help to drive research and evaluate new designs. In this paper, we present our open-source implementation of the BIER data-plane. Its objective is to provide to the research community a building block to design and experiment protocols on top of this new multicast architecture. It leverages the C standard library to represent a BIER data-plane daemon. As no formal API has been defined by the IETF to communicate with BIER, we propose and implement a socket-like API to interact with our daemon. The BIER packets are forwarded using IPv4/IPv6 tunnels [9] to accommodate existing networks that do not natively support the architecture. We also provide an implementation of the BIER-TE draft [4]. To demonstrate it, we simulate a small network topology with ns-3 DCE and our daemon. The implementation essentially focuses on source-specific multicast, with (S,G) representing the multicast **S**ource and **G**roup addresses pair. Section 2 presents the BIER data-plane implementation and the API to communicate with the daemon. The ns-3 DCE simulation is detailed in Section 3. Section 4 concludes this paper. The data-plane uses ∼1500 SLOC and the API uses 400 SLOC.

2 IMPLEMENTATION

We design our daemon to work on existing Linux network stacks. As such, BIER packets are encapsulated in IPv4/IPv6 tunnels [9] between two BIER Forwarding Routers (BFR). The destination of the tunnel headers is the address of the *next BFR* that will process the packet. The *next BFR* is found by inspecting the packet *bitstring*. For each bit set in the *bitstring*, the daemon queries its BIFT, and erases the potential other bits that are mapped to the same outgoing interface. Finally, it sends to the *next BFR* a copy of the modified packet.

Each router running the daemon listens to incoming BIER packets using a raw IP socket. This socket is also used to forward the packet copies after processing.

The implementation supports the BIER-TE extension [4] for traffic engineering purposes. With BIER-TE, each link (or IP tunnel) between two BFRs is also assigned a bit in the *bitstring*. To forward a packet, a BFR now inspects the bits corresponding to its outgoing interfaces. BIER-TE offers strong traffic-engineering capabilities, at the cost that the source must explicitly encode the entire path towards the BIER egress routers of the multicast flow.

The two forwarding mechanisms are distinct and have different outcomes. We leverage the BIFT-id of the BIER header [8] to let the data-plane daemon understand which procedure to use when it processes a BIER packet.

The data-plane daemon runs in its own process to forward and create BIER packets. However, a BIER Forwarding Router may also act as an egress router, meaning that it could receive packets destined to its connected hosts. These packets should leave the BIER data-plane and be forwarded to an upper layer. Additionally, a multicast source must be able to send packets towards its receivers through the BIER data-plane.

To this end, we designed a socket-like API to let upper-layer applications, e.g., multicast transport protocols, communicate with the BIER daemon. As the daemon runs in its own process, the channel uses datagram UNIX sockets in both directions. The API (Listing 1) mimics the socket API.

```
ssize_t sendto_bier(int socket, const void *buf,
    size_t len, const struct sockaddr *dest_addr,
    socklen_t addrlen, uint16_t proto, bier_info_t
     *bier_info);
ssize_t recvfrom_bier(int socket, void *buf,
    size_t len, struct sockaddr *src_addr,
    socklen_t *addrlen, bier_info_t *bier_info);
int bind_bier(int socket, const struct sockaddr_un
     *bier_sock_path, bier_bind_t *bind_to);
```

Listing 1: Socket-like API.

`sendto_bier` forwards to the daemon the payload `buf` of length `len`. The UNIX path to the data-plane daemon is represented by `dest_addr`, of length `addrlen`. The `proto` value specifies the protocol of the payload, as defined in [7]. The last parameter, `bier_info`, is a new structure containing (i) the BIFT-id that a BFR uses to forward the packet, (ii) the *bitstring* of the packet and (iii) the *bitstring* length [8].

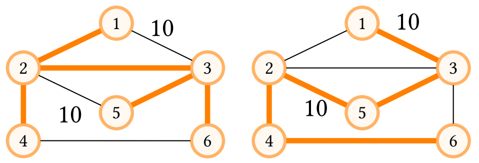
`recvfrom_bier` reads packets received from the data-plane daemon in `buf`, of capacity `len`. The `src_addr` is the address of the upstream BFR. `bier_info` carries the position in the *bitstring* of the upstream BIER router.

`bind_bier` informs BIER of the interest of an application in a multicast flow. `bier_sock_path` is again the UNIX path to communicate with the daemon. `bier_to` is yet another structure that specifies the (S,G) pair.

Similarly, `unbind_bier` (same signature as `bind_bier`) communicates to the daemon the intent to leave the multicast group. Upon reception of these messages, the BIER data-plane forwards to the BIER ingress router a BIER packet containing the multicast join/leave message.

3 SIMULATING BIER

We demonstrate our implementation using ns-3. We wrote a wrapper to simulate our daemon in ns-3 without source code change [1]. For illustration, we simulate a small network of 6 routers. The topology is represented in Figure 1. All links have a cost of 1, except `0-2` and `1-4` with a cost of 10. For simplification, we simulate hosts directly on the nodes and compute BIFTs beforehand using a shortest-path algorithm. Our host receivers and senders leverage our socket-like API. Node 0 simulates a multicast sender, waiting for all other nodes (the receivers) to send a multicast join request. We run



(a) BIER tree. **(b) BIER-TE tree.**

Figure 1: Small network topology with 6 nodes.

the simulation with BIER and BIER-TE, and collect packet captures from each interface of all routers. The multicast trees constructed by BIER are represented in bold orange by analyzing these packet captures.

With BIER, the multicast tree is implicitly built using the shortest path towards its leaves. As expected, the flow uses the shortest path to reach each receiver (Figure 1a). BIER-TE [4] enables the source to specify the entire multicast tree for traffic engineering purposes. Once the receivers join the multicast flow, we set the *bitstring* so that the multicast tree uses the two links with a cost of 10. Figure 1b shows that the path is correctly enforced despite the higher link costs.

4 CONCLUSION

This paper presented our open-source implementation of the Bit Index Explicit Replication (BIER) forwarding mechanism [7]. We provide our BIER implementation and a simple socket-like API to communicate with our daemon. This implementation can be deployed in networks that do not support BIER by leveraging IP tunnels to carry the BIER packets [9]. With this mechanism, several research problems related to multicast transport protocols can be tackled more easily, such as reliable communication, multicast congestion control and scalable deployment. We hope that this work will stimulate the research community into new efforts in developing multicast protocols above BIER.

REFERENCES

[1] Anonymous. [n. d.]. Anonymized source code.
[2] Yoann Desmouceaux et al. 2018. Reliable multicast with BIER. *Journal of Communications and Networks* 20, 2 (2018), 182–197.
[3] Christophe Diot et al. 2000. Deployment issues for the IP multicast service and architecture. *IEEE network* 14, 1 (2000), 78–88.
[4] Toerless Eckert et al. 2022. *Tree Engineering for Bit Index Explicit Replication (BIER-TE).* Internet-Draft draft-ietf-bier-te-arch-13. Internet Engineering Task Force. Work in Progress.
[5] Alessio Giorgetti et al. 2017. Bit Index Explicit Replication (BIER) multicasting in transport networks. In *ONDM 2017.* IEEE, 1–5.
[6] Daniel Merling et al. 2021. Hardware-based evaluation of scalable and resilient multicast with bier in p4. *IEEE Access* 9 (2021), 34500–34514.
[7] IJsbrand Wijnands et al. 2017. Multicast Using Bit Index Explicit Replication (BIER). RFC 8279. (Nov. 2017).
[8] IJsbrand Wijnands et al. 2018. Encapsulation for Bit Index Explicit Replication (BIER) in MPLS and Non-MPLS Networks. RFC 8296. (2018).
[9] Zheng Zhang et al. 2022. *Supporting BIER in IPv6 Networks (BIERin6).* Internet-Draft draft-ietf-bier-bierin6-05. IETF. WIP.