Bidirectional Forwarding Detection (BFD) implementation and support in OpenBSD

Peter Hessler
phessler@openbsd.org
OpenBSD

1 Introduction

Internet links fail. This is a truism as old as Internet links. When a link fails, traffic gets dropped until the failure is detected and traffic can be re-routed. Detection of failures can be quite tricky however, since they are not always directly visible. Most systems use link state or a form of keep-alives for detection of failures. Link state detection does not help when there are active devices between a router and the other system, such as a switch or long distance links which use MPLS. The in-protocol BGP timers can also be quite long (commonly 90 seconds) which is a lot of traffic when one are sending 10Gbps or even faster rates.

Bidirectional Forwarding Detection (BFD)[1] is a protocol that allows detecting faults in links or routes. This is similar to GRE keep-alives, but is actually supported on real routers. Contrary to traditional link-state detection, BFD works on the next-hop IP address; so one can detect failures of some peers that do not affect the link state.

BFD is a protocol that exists outside of existing routing protocols, but can communicate the status to all protocols. This allows for a single keep-alive to detect the health of a single link, without having to depend on a keep-alive in each and every protocol being used. As this is part of the ”parent” interface, this does not introduce another layer in the network configuration. And since the link-state is only per next-hop IP, one can mix and match BFD and non-BFD neighbours on the same interface. This is extremely useful for routers connected to an Internet Exchange Point, which can have hundreds of peers spread over 10 or more physical locations.

This paper discusses of the implementation of the BFD protocol for OpenBSD, problems discovered in both the protocol and network stack, use cases and production experience.

1.1 Before BFD

In general, Ethernet is supported over two types of physical links, one is copper and the other is fiber. When the connection is established between two physical devices, power is applied and either an electrical current is sent over copper, or light is sent over a fiber link. While the link-state is based upon receiving the current or light, it does not indicate that the remote side is configured, or is capabable of processing traffic received. Unlike other link protocols, Ethernet does not communicate the availablity of services over this link. As such, detecting failures is dependent on protocols that are layered upon Ethernet. Even with the limitations of positivly stating the link is available, the loss of link-state is a guarentee that the link is not usable. When the link-state of an interface goes down, routing protocol daemons may decide to mark that interface as ”down” and to do whatever the protocol may require. However, this depends on the link-state being fully in-sync with the availabilty of the remote peer. In an IXP[2] environment, switches are guarenteed to be in path which will prevent the link-state to a neighbor from being seen as down.

BGP[3], for example, sends a KEEPALIVE message to ensure the connection is still live. Three KEEPALIVES must fail to be received before the neighbor is marked as down. These messages are traditionally sent every 30 seconds, and so requires 60 to 90 seconds before an outage can be detected. At the fastest, BGP may only send KEEPALIVE messages every second, so outages still require 3 seconds to be detected. As an example, when a link has a sustained traffic flow of 10Gbps, the loss can be up to 900 Gb of data which is an unacceptable amount of traffic to be lost due to a faulty link. Additionally, voice calls can suffer between 60 and 90 seconds of silence which will cause users to think the call is lost.

OSPF[4] keepalives may be sent as fast as 50 milliseconds, but still require no response for 1 full second to detect the failure of an OSPF speaker. An astute reader will also notice that both protocols need to send their own keep-alive traffic, which is wasteful.

Other protocols have their own methods, which must be independently monitored, transmitted, and received.

2 BFD Protocol

The BFD protocol is relatively simple. If replies are not received with a negotiated amount of time, the link is declared dead and is not used. It is generally encapsulated in a UDP packet, but may be encapsulated in any other protocol, that is sent from one system to a direct neighbor. This neighbor copies some values from the packet that is sent, then a reply is sent.

As BFD checks the forwarding plane of a device, it is intended to be sent, and checked, from the forwarding engine. This ensures that if the system cannot forward packets, then it no longer sends BFD packets.

There are two types of BFD packets.

`Control`: Packets that are encapsulated appropriately to the environment, and meets the Mandatory format as specified in RFC 5880. Control packets may have one of two modes.
`Async`: Control packets are sent on a timer.
`Demand`: The decision to send control packets is implementation specific. A common method is to monitor the `received packets` counter. If no packets are received during that timer, then a control packet is sent.

`Echo`: The sender sends any information it wants, and the receiving system is supposed to copy it and send the information back without modification.

The timers used are in microseconds, which is one millionth of a second. A multiplier is used to detect failures to receive, which may be as low as 1. These two features, when configured to their most aggressive, means a link failure can be detected in as little as two microseconds.

There is an optional Authentication section, which is intended to allow the receiving system to determine the validity of the received packet. As we will see later, the Authentication section is problematic, so we do not implement it.

2.1 BFD: Control Packets

BFD Control packets are fixed binary format, sent in network byte order.

`Version`: a version number

`Diagnostic Code`: reason for the last change in session state.

`State`: the current state of the session.

`Flags`: Poll, Final, Control Plane Independent, Authentication Present, Demand, Multipoint. These indicate various flags for this packet in the session. Certain flags may not be combined, and other flags have other restrictions.

`Length`: the byte length of the BFD Control packet.

`My Discriminator`: A unique value generated by the transmitting system used to demultiplex multiple sessions between the same pair of systems.

`Your Discriminator`: The value received from a remote system.

`Desired Minimum TX Interval`: Minimum interval that we would like to use, in microseconds.

`Required Minimum RX Interval`: A remote system must send control packets faster than this value, in microseconds.

`Required Min Echo RX Interval`: If a system supports echo packets, this is the minimum interval a remote system may send, in microseconds.

2.2 BFD Authentication

Authentication in BFD is simplistic. In the optional header, there is normal boilerplate, then a sequence number and the auth hash. A pre-shared-secret is placed in the area for the hash, then the entire packet is hashed according the Authentication Type. All hashing methods have a normal mode, and a mode called "meticulous". In normal mode, the sequence key is incremented occasionally. In meticulous mode, the sequence key must be incremented for each packet.

If authentication is supported implementations are required to support both modes for SHA1. The Authentication features of BFD have been critisized in RFC 6039 and RFC 7492. No higher quality hashing algorithms are standardized at this time.

3 Using BFD

For this section, I use 203.0.113.1 as my IP address, 203.0.113.9 as my neighbor's IP address. You may use any IPv4 or IPv6 address in place of those.

Enabling BFD on OpenBSD is currently a single command: `route -n change 203.0.113.9 -bfd`. At this moment, all options cannot be changed from the defaults. However, we can negotiate most options with a neighbor.

Showing the current configurations is available using `route -n get 203.0.113.9`, and state changes are sent over the routing socket, so are visible in router daemons or in `route -n monitor`. Detailed information for the output is available with the `-bfd` flag as shown below.

```
$ route -n get 203.0.113.9 -bfd
   route to: 203.0.113.9
destination: 203.0.113.9
       mask: 255.255.255.255
  interface: em1
 if address: 203.0.113.1
   priority: 4 (connected)
      flags: <UP,HOST,DONE,LLINFO,CLONED,BFD>
        BFD: async state up remote up laststate down error 0
             diag none remote neighbor-down
             discr 186919089 remote 55
             uptime 05d 2h07m29s
             mintx 1000000 minrx 1000000 minecho 0 multiplier 3
     use       mtu    expire
   83923         0       229
sockaddrs: <DST,GATEWAY,NETMASK,IFP,IFA>
```

The BFD state is communicated via the routing socket to allow userland utilities to make decisions, but currently does not change the UP or DOWN state of the route itself.

OpenBSD is able to successfully negotiate and keep sessions for at least 7 days against several vendors including Juniper, Cisco, Force10, and Extreme Networks.

4 OpenBGPD

OpenBGPD[5] currently has simple support for BFD. At the time of publication, it knows when a BGP neighbor is monitored with BFD, and will immediately mark the nexthop IP address as DOWN when told by BFD.

5 Future Work

Future work on this topic include implementing multi-path sessions, Seamless BFD[6] , and extending the integration into OpenBSD's networking daemons such as ldpd, ospfd, eigrpd, etc. Implementation of the BFD protocol encapsulated in ospf, vxlan, ldp, and other protocols is also desirable.

An expired Internet Draft `draft-ietf-idr-rs-bfd`[7] introduces automagic configuration of BFD between parties allowing for stronger resilience when there are many potential neighbouring networks without the overhead of manual configuration. It was influencial in my interest in this technology, so it will be re-evaluted and possibly implemented.

6 Availability

Unless otherwise noted, the implementation described in this paper is available in OpenBSD -current, and will be part of the 6.1 release, currently expected in May 2017.

Notes

[1] RFC5880, D. Katz, and D. Ward, "Bidirectional Forwarding Detection (BFD)", June 2010
[2] Internet eXchange Point
[3] RFC4271, Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)", January 2006

[4] RFC1247, J. Moy, ”OSPF Version 2”, July 1991

[5] bgpd(8), bgpd - Border Gateway Protocol daemon, OpenBSD manual pages

[6] RFC7880, C. Pignataro, D. Ward, N. Akiya, M. Bhatia, and S. Pallagatti, ”Seamless Bidirectional Forwarding Detection (S-BFD)”, July 2016

[7] draft-ymbk-idr-rs-bfd, R. Bush, J. Haas, J. Scudder, A. Nipper, and T. King, ”Making Route Servers Aware of Data Link Failures at IXPs”, July 2015