Loser Buys
A Troublesome Retrospective

MARK BAINTER AND DAVE JOSEPHSEN

Mark Bainter is the Director of Global Support and Operations at Message Systems Inc., where he works with a talented team of sysadmins to provide support and outsourced monitoring and management for Message Systems' cross-channel messaging software. In his spare time he currently enjoys woodworking and tinkering with Arduino hardware.
mbainter+usenix@gmail.com

David Josephsen is the sometime book-authoring developer evangelist at Librato.com. His continuing mission: to help engineers worldwide close the feedback loop. dave-usenix@skeptech.org

Dave: Rik Farrow wants some troubleshooting stories for the next *;login* issue. Loser buys?

Mark: You sure about that, son? Need to check your balance first or anything?

Dave: Just for that, I'll let you go first.

Mark: Your funeral. First, a bit of background—the company I currently work for has a tool called "Gimli" [1], which you can link against your application and then use to ensure that it remains up and functioning, similar to "supervise." Gimli also does heartbeat monitoring, however, and if it crashes or has to be killed and restarted for being unresponsive, will take a snapshot of the stack, along with various details about the process for later review.

Dave: Nice. I bet that comes in handy.

Mark: Oh, it does. It's helped us resolve all sorts of weird issues that would otherwise require you to sit and watch and wait for the problem to happen. Best of all, you don't have any of the unpleasant side effects of some alternative methods of catching intermittent issues. Not to mention increasing the resilience of your application.

Anyway, a while back, in the middle of the night, I get an alert—the /var partition on one of the servers I manage is filling up quickly. That's definitely not good. Investigating, I find the disk is already nearly full, and in moments I find the culprit to be the log file for this Gimli process. The process it's managing is wedged, and Gimli is in a loop logging the same error over and over into the log, reporting that it has received a TERM signal and is terminating. That's really odd—I've never seen this failure condition before. I kill both processes, clean the Gimli log, and restart.

Reviewing the logs subsequently offers no clues as to what happened. There's no stack trace either. Curiouser and curiouser. I don't like unexplained activity like this, but it's the middle of the night and I'm at a dead end. I turn in.

The next morning it happens again. Same time. Now I have a hint. After I restore service, I start looking through the application's activity during that window, into the system logs, cron jobs, etc. It doesn't take me long to correlate the log rotation with the time window where this is occurring.

Dave: Ah, 4 a.m., when the logrotate bugs come out to play.

Mark: Exactly. This process that Gimli is monitoring is set up with a fairly standard daily rotation, followed by compression and then a postrotate instruction to send a HUP signal to force reopening of the logs.

I spin up a VM and start doing some testing and at first I can't reproduce the problem. I run the log rotation and everything works fine. Then it hits me. Some time back I made a modification to the logrotate script! By default we were not rotating the error log for this process, because it is almost never written to. This node, however, was throwing a lot of errors which another team had been investigating, so in the interim I had set up log rotation to keep it from filling the disk.

I add the path to the error log in the logrotate script on my virtual machine, rotate the logs, and sure enough, the rotate failed, and the log was filling up. Now I have a readily reproduc-

ible problem. Of course, this still doesn't explain the why, or the why now. After resetting the test, I do some tests with GDB, which is frustrating because the heartbeat method used by this app was sending a USR1 signal which kept causing GDB to stop.

Dave: You know, you can set a nostop pass signal handler in GDB to get around that [2].

Mark: Yeah, but at the time I wasn't aware of it, and there's another favored tool to reach for that could readily report those without being interrupted—namely, strace. In short order I discover that there are actually two HUP signals being sent in short succession.

Dave: Right. Logrotate would have sent one signal for each logfile unless you set the sharedscripts option.

Mark: Yeah, well, I didn't remember that when I set up the config, but I remembered it now. The full explanation here of what was happening requires some understanding around how the Gimli process interacts with the processes it monitors, so I'm going to gloss over some of that for the sake of not boring our readers. Basically, when Gimli saw the HUP come in, it created a new version of itself to take over monitoring the process, but the second HUP came in before that execve could complete. As a result, the two copies of Gimli would become confused, and continuously try to kill each other in a vicious loop, resulting in flooding the error log with the termination messages. Since neither would honor the TERM signal fully as a protective measure for a monitoring process, the loop never ended. Thankfully, more recent versions have addressed this weakness.

Dave: Heh, if you'd named it "Claudius" instead of "Gimli," it might have been more adept at fratricide. Okay, so now I understand the why, but I'm confused why it suddenly started happening. Wouldn't it have begun delivering a double HUP as soon as you first modified logrotate? Why didn't it happen right after you made the change?

Mark: That's the real kicker isn't it? Luckily, this was the easy part to figure out. When I first implemented it, as I said, this node was throwing a lot of errors, forcing me to implement the rotation. Since then, the problem causing those errors had been fixed, unbeknownst to me.

Dave: Oh! It was a race condition. When the log files had content, the time logrotate spent copying and compressing them would've given Gimli enough time to fork, so everything was fine. It was only when the log files were empty that the HUPs would win. Nice!

Mark: Got it in one!

Dave: Damn...that's a great story, and I'm not sure I can top it, but here's my favorite troubleshooting story. I like it because there's a bit of dramatic panache at the end.

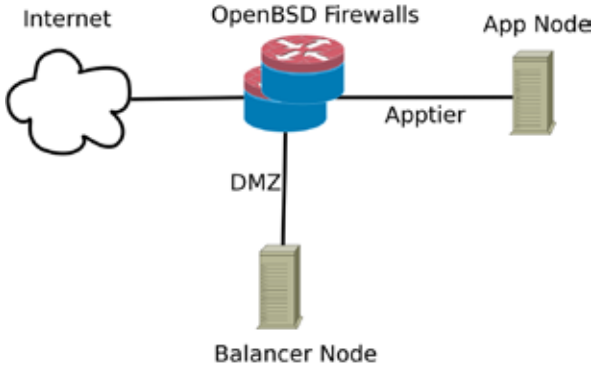

Figure 1: In Dave's troubleshooting conundrum, packets were disappearing somewhere between the load balancers and app servers. One of the pair of firewalls is a backup, using CARP to share state with the active firewall.

Anyway, we were having trouble with a Web application that we'd just put in production. The setup looked like Figure 1. The problem presented as some intermittent latency when using the app. Sometimes it worked fine, other times it was very slow, and still other times it didn't work at all. And this wasn't like, over the course of one hour it'd be slow and the next it'd work fine, this was like, one HTTP request might work fine while two others executing concurrently did not.

Mark: Sounds fun.

Dave: It wasn't. And for the first and only time in my professional career, when the developers started blaming "the network," it looked like they were actually right. Working our way down the stack we were pretty convinced that packets weren't getting to the application server. Somehow this network was eating packets, and obviously it wasn't any sort of ACL or filtering stuff because some requests were making it through just fine.

So here's the background you need to understand: the internal core routers were two OpenBSD systems running CARP (Common Address Redundancy Protocol) with pfsync. It's pfsync's job to replicate the firewall state table between the master and failover nodes, such that, if they ever fail over, the failover router will know about all the active and authorized connections. Without pfsync, the backup router would drop all the existing network connections and force them to re-handshake. We have a physical network port on each router configured specifically just for pfsync, and the two routers are directly connected to each other via a crossover cable on this port.

CARP creates a magical virtual network device whose status is shared between the two routers. CARP is what actually enables the backup router to detect and take over for a failed master router without the MAC or IP address changing for other network entities.

The balancers meanwhile operated using a multicast MAC address...

Mark: Was there IGMP?

Dave: No, and that's important. As you know, in the absence of IGMP the default behavior of a Cisco router is to broadcast the multicast packets to all live switchports in the same VLAN.

Mark: I think I see where this is going.

Dave: Shut up. Anyway, there I was, stumped. I could tcpdump the traffic as it came from the Internet to the balancers. That looked okay. But only a subset of the traffic was making it to the application tier nodes. It was maddening because I couldn't seem to narrow it down to any one device. Watching the firewalls carefully I could see that they weren't failing over. Watching the packet traces in the apptier, it looked like traffic worked fine for a while and then every so often, a good connection would just freeze. Eventually, when this happened the apptier node would send an RST and stuff would start working again. The balancers seemed to be getting traffic okay, but they were also freezing and RSTing every so often.

I had a bit of an ah-hah moment when I started looking at the packet traces on the failover firewall. It appeared to be getting a copy of all the multicast traffic that was destined for the balancers. This was odd because in CARP backup mode, the failover router isn't answering ARP on its CARP virtual devices, and should therefore not receive any of the traffic for those shared IPs.

Mark: Even if the failover was getting traffic, it shouldn't be routing it.

Dave: Exactly my thinking. Evidently the traffic was appearing on the backup firewall because in the absence of IGMP, the Cisco 3750 was broadcasting that traffic to all active switchports in the VLAN, including those of the failover router. But that traffic should be harmless anyway since the failover router would just drop it all on the floor. I was back to square one.

Mark: Or were you?

Dave: Or was . . . shut up. I stared at the rack a few minutes, trying to imagine every possible path a packet might take through this rather simple little network, and something interesting occurred to me when I imagined what might happen to me if I were one of those multicast packets that had been duplicated to the failover firewall. The interesting thing was that I would wind up in the inbound packet buffer on the firewall's DMZ port while the firewall checked its state table and ACLs. Our assumption that the traffic wouldn't be forwarded is based on the fact that the backup firewall wouldn't have a state table entry for the connection in question.

Mark: Right, the failover firewall would compare the source and destination addresses of the packet to its internal list of existing states, and then, not finding one, it would drop the packet.

Dave: Except OpenBSD's pfsync service replicates that state table between the master and failover CARP nodes. The failover router has every active state that the master does, and therefore DOES in fact have a state table entry that matches the packet. So there's an interoperability bug between Cisco and OpenBSD pfsync . . .

Mark: OpenBSD assumes the Cisco won't give it a packet it doesn't ARP for . . .

Dave: Yes, exactly, and Cisco assumes OpenBSD isn't going to forward a broadcast packet because it won't exist in its state table.

Mark: So why isn't there a broadcast loop that affects the master firewall node? Wouldn't the master also receive a copy of the multicast packet?

Dave: No, because the master firewall is the default gateway device for the network, so it's the switchport that originated the traffic, and will therefore not receive a copy of the broadcast.

Mark: Man, that's hairy. What happens when the failover node tries to route that packet?

Dave: I don't know exactly. It's undefined, but in that network, intermittent latency and lots of RST ensued.

Anyway, here's the best part. I have this big eureka moment, and jump up out of my chair excitedly describing my hypothesis to the other engineer who is working the problem with me. He thinks I might have it solved, but wonders out loud how I'm going to test whether I'm right or not. Not answering, I walk over to the rack and with as much showmanship as I can muster, ceremoniously rip out the pfsync cable connecting the two routers. TA-DAAA problem solved!

Mark: Nice, but what about clean failover?

Dave: It's not really an issue for us, because we usually use Pix's on the edge, but you could manually configure the switch ARP-table so they didn't broadcast, or you could use IGMP if possible, or, yeah, just run the firewalls without pfsync, which might bite you later on, but not very much. The network would "hiccup" whenever they failed over, and the users who did get an error could hit the reload button and everything would be fine.

Dave: Well, I think you probably won that one. I mean you had interprocess warfare and GDB!

Mark: Really? I kind of liked yours because I might one day try to run PF and mod_proxy_balancer with Cisco switches, and you probably just saved me a headache.

Dave: Well, Rik, we'll leave it to you. Who's buying?

Rik: Dave, you're buying. While both stories are good, Mark did a better job of explaining exactly what had gone wrong, as well as having more twists and turns. Your solution, breaking the con-

nection between firewalls, fixes the problem without telling us exactly what was going wrong. Not that figuring that out would be easy, as it likely lies in the IP stack of OpenBSD somewhere.

References

[1] https://github.com/MessageSystems/gimli.

[2] http://sourceware.org/gdb/onlinedocs/gdb/Signals.html.