

Implementation of a Load Balancing Technique for OSPF

Nicholas Ng[†] Ashwin Sridharan[†] Roch Guérin[†]

Nicholas Ng (ngn@seas.upenn.edu)

Ashwin Sridharan (ashwin@ee.upenn.edu)

Roch Guérin (guerin@ee.upenn.edu)

[†] Dept. Elec. Eng., U. Pennsylvania, 200 South 33rd Street, Philadelphia, PA 19104.

This work was supported by NSF Grant No. ANI-9902943.

I. INTRODUCTION

The main motivation of the project is to implement a technique to improve the load balancing ability of OSPF. Current OSPF[3] networks perform the Shortest Path First Algorithm where link state packets are swapped with neighbors to determine the topology of the network. The shortest path algorithm is then run to compute next hops for each destination. Current OSPF implementations have the ability to compute Equal Cost Multiple Paths (ECMP) resulting in multiple equal cost next hops to a destination. The primary advantage of such an ability is that traffic headed towards the destination may be balanced across the next hops. However, present day routers only split traffic equally over *all* equal cost next hops resulting in a very coarse grained load profile across the next hops. Our goal is to provide a mechanism that improves the ability of a router to balance traffic better across equal cost next hops *without* making any changes to the way a router handles packets on the data path. The mechanism basically consists of *removing* selected next hops for a set of prefixes from the forwarding table.

The concept was derived from the paper *Achieving Near-Optimal Traffic Engineering Solutions for Current OSPF/IS-IS Networks*[4], where it states that by pruning congested next hops for each route, we can optimize throughput. The methods for finding the correct next hops to remove such that traffic is optimized is given within the paper.

An important consideration however, is that the modifications should not affect current protocols and systems already in place. Zebra[1], a GNU licensed routing software suite written primarily by Kunihiro Ishiguro, provides a good platform from which modifications could be made since it is open source and free of charge. The software is also widely used and considered stable.

II. BACKGROUND

The Zebra software suite is a modular package that serves as a dedicated router. The three major daemons that run under Zebra are RIP, OSPF and BGP daemons. They all report routes to the Zebra daemon, which then communicates them to the operating system routing table. Since the protocols and their respective daemons are all modular, existing protocols can be updated and new protocols can be added without disrupting the other functioning protocols. Each protocol has its own configuration file and VTY command line interface which allows a user to modify and control all parameters of the daemon. In addition, the Zebra daemon itself also has a command line interface for dealing with Zebra daemon specific commands.

The protocol daemons are responsible for communicating with other daemons on the network to find

the topology of the network. Once a route is found and confirmed, it will be added to its own route table, which is evident when a user logs into the VTY for that daemon and requests the routing table. It will then propagate the route to the Zebra daemon.

The Zebra daemon is a layer of abstraction between each protocol and the operating system routing table. It maintains a table of the routes and the protocol that sent the route to it. The Zebra daemon will then update the operating system routing table with the new route.

Our goal is to improve the load handling abilities of OSPF, thus the OSPF daemon was directly modified. It could not be determined whether modifications to the Zebra daemon itself would corrupt the routing tables of other protocols running under Zebra. Thus, internal functions are written into the OSPF daemon to allow the deletion of a next hop from a route given the specified prefix-nexthop combination.

To allow access to the functions, new OSPF daemon VTY commands are installed into the system. These VTY commands accepts a prefix and nexthop as arguments, then it passes them along to the internal deletion function. The internal function re-adds the route to Zebra to overwrite the old route and update the manipulated set of next hops.

Expect [2] scripts will help streamline the mass deletion of next hops. The scripts simply log on to the OSPF daemon and interacts with it. This also allows a user to delete a set of next hops at a remote server from a list that is located locally.

With this new set of functions within the OSPF daemon and the simple expect scripts, we can easily manipulate the set of next hops for each route. This would allow us to reach close to optimal traffic throughput.

III. OSPF VTY FUNCTIONS FOR DELETING NEXT HOPS

This section details the final usable functions. In order to use these commands, log into the OSPF daemon using the command “telnet server.ip.address port”. After entering the correct password and receiving the command prompt, enter the commands below. The Zebra VTY command line features such as “?” help and tab auto-complete also work with these new functions.

A. *ospf delnexthop A.B.C.D/M E.F.G.H*

Usage When prompted with “>” at the OSPF daemon command line, type the command above replacing A.B.C.D/M with the destination prefix desired and E.F.G.H with the nexthop. A.B.C.D/M can also be replaced with an IP address. The function will then search for the specified prefix. If it is

found, then it looks for the specified nexthop within its set of next hops. This nexthop will be removed if it is found and is not the last nexthop remaining. The function will then return control to the user.

Implementation This command was added to the OSPF daemon VTY command line. The file modified was `ospfd/ospf_vty.c`. The function first checks for the validity of the provided prefix and nexthop. It also checks whether OSPF and the required tables are ready. Next it tries the deletion using the function “`ospf_nexthop_remove`” within `ospf/ospf_route.c`. `ospf_nexthop_remove` first looks up the prefix using “`route_node_lookup`”, a function from the Zebra library which searches the specified table for the prefix. Then it takes the route found, and search through its list of next hops. Once found, the nexthop is deleted, and the route is re-added to zebra. This updates the Zebra daemon which will in turn update the operating system.

B. `ospf delnexthopfile FILENAME`

Usage When prompted with “>” at the OSPF daemon command line, type the command above replacing `FILENAME` with the *full path* of the file desired. The file must be line separated combinations of prefix and nexthop. There should be a single whitespace between the prefix and nexthop. The function will then search for the specified prefix in the file. If it is found, then it looks for the specified nexthop within its set of next hops. This nexthop will be removed if it is found and is not the last nexthop remaining. The function will continue to attempt to delete each prefix/nexthop pair until the end of the file is reached. It will then return control to the user. The results of the deletions will not be shown on screen but it will be logged.

Below is an example of the format of the file:

```
192.168.1.0/24 192.168.2.2
192.168.5.0/24 192.168.2.2
```

Fig. 1. Sample File Format

Implementation This command was also added to the OSPF daemon vty command line. The file modified was `ospfd/ospf_vty.c`. The function checks whether OSPF and the required tables are ready, then attempts to open the file to read. A line is read in a while loop, where it retrieves the prefix/nexthop combination, then it tries the deletion using the function “`ospf_nexthop_remove`” as described above. The loop continues until it reaches end of file.

IV. EXPERIMENTS

To test the functions above, a three machine testbed was set-up as shown in Figure 2. All three machines are running RedHat 7.3 (Linux Kernel 2.4) with Zebra 0.93a installed. Each machine first starts the Zebra daemon, then the OSPF daemon. Below is an image of how the machines are organized:

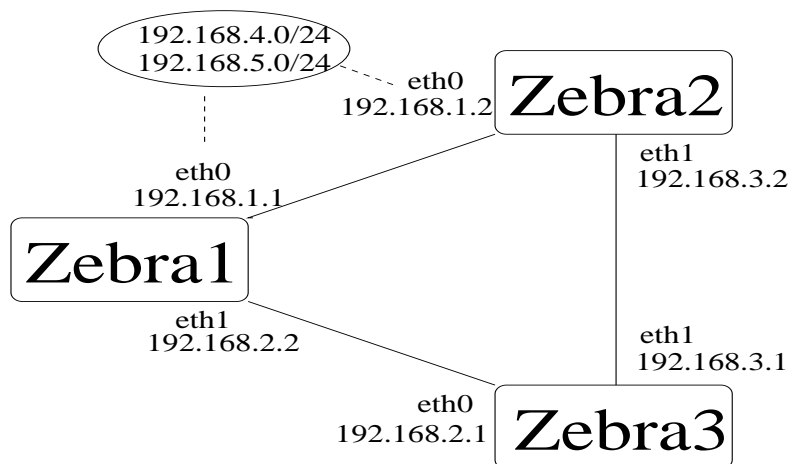


Fig. 2. Figure Representing TestBed used in Experiments

From above, Zebra1 connects to Zebra2 in the network 192.168.1.0/24, Zebra1 connects to Zebra3 in the network 192.168.2.0/24 and Zebra2 connects to Zebra3 in the network 192.168.3.0/24. Two additional false static routes were added to each of Zebra1 and Zebra2 which were 192.168.4.0/24 and 192.168.5.0/24.

When the OSPF daemon starts on Zebra1 and Zebra2, they both show that they can reach 192.168.1.0/24, 192.168.4.0/24 and 192.168.15.0/24 directly. Zebra3, however, show that these routes can be reached through either Zebra1 and Zebra2. An example of Zebra3 OSPF routing table can be seen in Figure 3.

```

===== OSPF network routing table =====
N    192.168.1.0/24      [20] area: 0.0.0.0
                                via 192.168.2.2, eth0
                                via 192.168.3.2, eth1
N    192.168.2.0/24      [10] area: 0.0.0.0
                                directly attached to eth0
N    192.168.3.0/24      [10] area: 0.0.0.0
                                directly attached to eth1

===== OSPF external routing table =====
N E2 192.168.4.0/24      [20/20] tag: 0
                                via 192.168.2.2, eth0
                                via 192.168.3.2, eth1
N E2 192.168.5.0/24      [20/20] tag: 0
                                via 192.168.2.2, eth0
                                via 192.168.3.2, eth1

```

Fig. 3. Routing Table of Zebra3 Before Deletion

If a nexthop, say 192.168.2.2, is congested, we can delete that nexthop for route 192.168.4.0/24. A user can then log into the OSPF daemon VTY and run the command “ospf delnexthop 192.168.4.0/24 192.168.2.2”. This will eliminate the Zebra1 nexthop for the route 192.168.4.0/24 and thus all traffic is forced into the remaining routes, namely Zebra2 in this case. If we go ahead and run “ospf delnexthop file” using the file from 1, we can route all traffic through Zebra2. The resulting Zebra3 OSPF routing table can be seen in Figure 4

V. MASS DELETION USING EXPECT SCRIPTS

This section describes the expect scripts written to facilitate the manipulation of the set of next hops from a remote location. The scripts were first generated using autoexpect. Since the scripts are quite simple to follow, only the syntax for usage is provided. The expect package needs to be installed on the operating for these commands to function.

```

===== OSPF network routing table =====
N    192.168.1.0/24      [20] area: 0.0.0.0
                                via 192.168.3.2, eth1
N    192.168.2.0/24      [10] area: 0.0.0.0
                                directly attached to eth0
N    192.168.3.0/24      [10] area: 0.0.0.0
                                directly attached to eth1

===== OSPF external routing table =====
N E2 192.168.4.0/24      [20/20] tag: 0
                                via 192.168.3.2, eth1
N E2 192.168.5.0/24      [20/20] tag: 0
                                via 192.168.3.2, eth1

```

Fig. 4. Routing Table of Zebra3 After Deletion

A. *ospfdelnh*

Usage ospfdelnh HOST PORT PASSWORD.

To delete a single nexthop, one needs to modify the file and enter the command exactly as one would within the OSPF daemon vty command line. One can also simply copy and paste send/expect to chain more commands.

B. *ospfdelnhfile*

Usage ospfdelnhfile FILE

To delete a set of prefix/nexthop combinations from a local file, run “ospfdelnhfile FILENAME”, replacing the FILENAME with the relative or full path. The script takes in the filename and checks whether it exists. In the loop, it reads each line and appends “ospf delnnextthopfile” in front of it, then sent to the OSPF daemon. It repeats until end of the local file is reached. The file it requires is exactly the same one used for “ospf delnnextthop file”

VI. CONCLUSION

We can now easily manipulate the set of next hops for each route in the routing table. With this set of tools, we have implemented the methodology presented in [4] and configure the traffic pattern for each route, thus optimizing throughput.

REFERENCES

- [1] Kunihiro Ishiguro. ZEBRA: Gnu zebra routing software 0.93a (software). <http://www.zebra.org>, July 2002.
- [2] Don Libes. EXPECT: Expect scripts (software). <http://expect.nist.gov>, July 2003.
- [3] J. Moy. OSPF Version 2. Request For Comments (Standard) RFC 2328, Internet Engineering Task Force, April 1998.
- [4] Ashwin Sridharan, Roch Guerin, and Christophe Diot. Achieving Near Optimal Traffic Engineering Solutions in Current OSPF/ISIS Networks . In *Proceedings of INFOCOM'2003*, San Francisco, CA, April 2003.