International Journal of Computer Science
Scholarly Peer Reviewed Research Journal - PRESS - OPEN ACCESS
ISSN: 2348-6600
http://www.ijcsjournal.com Volume 5, Issue 1, No 3, 2017 ISSN: 2348-6600
Reference ID: IJCS-166 PAGE NO: 1013-1017

Sri Vasavi College, Erode Self-Finance Wing 3rd February 2017
National Conference on Computer and Communication *NCCC'17*
http://www.srivasavi.ac.in/ nccc2017@gmail.com

ARCHITECTURE AND PERFORMANCE OF A PRACTICAL IP FAST REROUTE IMPLEMENTATION

J.Ragavi Priya
M.phil Research Scholar
R.Senthil Kumar
Head, Department of Computer Science
Sree Amman arts & science college, Erode.

ABSTRACT- IP Fast Reroute (IP FRR) denominates a set of methods and technologies for proactive, local recovery in IP networks. It has received much attention recently, in the research community and the IETF. Several proposals for IP FRR have been published by independent research groups, and main properties and trade-offs are understood. However, only simulation-based and analytical evaluations are known to be performed so far. In this paper I present the architecture and evaluate performance of the first known practical implementation of IP FRR. The architecture features a modular approach consisting of an IP FRR framework and exchangeable backup path calculation modules. The performance evaluation is based on an implementation of Multiple Routing Configurations. The implementation provide throughput that closely matches the achieved performance during normal operation.

I INTRODUCTION

The versatility of the Internet Protocol (IP) combined with the ubiquitious Internet deployment has lead to a migration of all types of network services to a single platform. These include VoIP, video conferencing and other real-time and interactive services, increasing the requirements for the reliability and availability of the network infrastructure. Normally, failures in IP networks are taken care of by mechanisms that exist in traditional IP routing protocols such as OSPF or IS-IS. These protocols identify failures as topology changes, and convey the information throughout the network using link-state advertisements. All routers re-calculate their routing paths, and populate their forwarding tables based on this information. Such global, reactive response is slow. Although a careful tuning of parameters can decrease the reconvergence time to a sub-second timeframe in a controlled environment, normally the re-convergence time is unacceptable for time-critical applications.

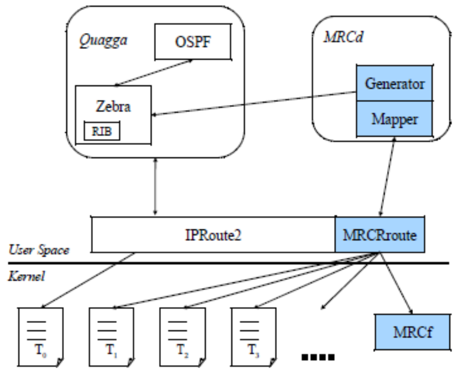

Figure 1: N/W node architecture overview, the shading part denotes MRC functionality.

This paper is organized as follows. Section II describes the IP fast reroute scheme used in this implementation and gives a short introduction to the basics of routing and forwarding using GNU/Linux. Section III gives an overview of the challenges the implementation needs to meet. Section IV gives a detailed description of the IP FRR architecture and implementation while section V gives an evaluation of the performance.

II BACKGROUND

A. MRC : Multiple Routing Configurations (MRC) is an IP Fast Recovery scheme based on observing that high link weights in a network with shortest-path routing can be used to “isolate” some nodes from forwarding. The isolated nodes are still functional data sources or destinations, but the shortest path algorithm will not select them as transit nodes. MRC requires that all nodes in the network are connected by a path that does not contain any isolated nodes.

B.Routing and forwarding using GNU/Linux : Software routers implement the forwarding and routing logic in software, and are generally built using normal PC class hardware. In our implementation I have used GNU/Linux as the foundation. GNU/Linux based software routers implement the traditional split between routing and forwarding; the forwarding engine and its configuration framework is found inside the Linux kernel while the routing protocols are implemented by software routing suites (i.e. applications) in the user space. Thus, software routing suites construct and maintain the routing table, while the kernel uses this information in order to decide and subsequently transmit the packet towards its next hop.

I give a high level view of the current mechanisms used, as that is less likely too changes in a radical way in the near future.

1)Routing engine: For the routing engine I selected Quagga [7], which is a popular software routing suite for GNU/Linux, BSD and Solaris. The software is based on a modular architecture where the routing service is provided by a set of daemons that implement different routing protocols.

2) The kernel forwarding engine: Several components are combined in order to provide the IP layer functionality in the Linux network stack. Here I give a basic understanding of the design and key components used; the routing policy database (RPDB), Netlink, and Net filter. A large number of common unicast IPv4 and IPv6 IGPs daemons are available.

III. DESIGN CHALLENGES

IP fast reroute mechanisms use pre-calculated alternative paths to provide fast recovery. The mechanism I use in our implementation has no upper bounds on how many backup topologies are required, but typically 3-6 are sufficient to guarantee recovery from any link or node failure in the network. The additional entries in the routing table increase the memory usage for the routing tables, the administration overhead, and the potential number of entries in the table lookup cache. In software routers memory is cheap and plentiful, and thus I don't perceive additional memory requirements as a problem. However, the administrative overhead could potentially disrupt traffic if increased by a large factor. I choose to minimize RPDB administration overhead at the cost of using multiple routing tables. By doing this the size of routing table used during fail free operation is kept at its original size and allows sequential scheduling of ordinary and recovery administration tasks.

In a failure situation IP fast reroute mechanisms increase the computational complexity for packet processing in routers adjacent to the failed network component. In order to determine if an IP packet should be recovered it must be submitted to an ordinary route resolve providing an outgoing interface. Subsequently, if the selected next-hop link has failed, the recovery procedure needs to perform it's own route resolve in order to determine the alternative outgoing interface. This leaves less resources available for other important routing tasks such as processing routing update LSAs, re-compute the paths, and updating the FIB. In order for routers perform recovery while retaining operational stability, the packet recovery process needs to be as fast as possible. Thus, I try to minimize the computational overhead in the packet recovery process.

IV. ARCHITECTURE AND IMPLEMENTATION

The architecture of our implementation is divided in two main parts; the MRC/IP FRR extensions to the kernel forwarding procedure, and the MRC extensions to the routing protocol running in userspace. An overview of the architecture is shown in Fig. 1. Our work extends the original architecture with four additional components; a backup configuration generator, a routing information processor, a user-space to kernel communication channel, and an extended forwarding engine.

1) Fast Reroute Framework: The framework is connected to the forwarding engine using the Net filter framework. This allows the framework to register callback functions at several
parts of the network stack. In general I only pick up packets after an ordinary route resolve; i.e. at the FORWARD hook and at the LOCAL OUT hook. At these points packets are available regardless of the path followed within the network stack and the selected outgoing interface is known. As a precaution the framework also register a hook at the POST ROUTING. At this stage no packets should be sent to a failed outgoing interface. This hook allows the framework to verify that packets sent to the outgoing buffer of any failed interface are dropped. I also register a special hook the PRE ROUTING. Normally, when a packet arrives at an interface it will be dropped if the source address in the IP header matches the local addresses of the router. This logic is used to avoid spoofing attacks or misconfigurations.

2) Failure handling engine: The main responsibility of the failure handling engine is to keep track of the failed interfaces. It maintains a local list which maps each network interface to

International Journal of Computer Science
Scholarly Peer Reviewed Research Journal - PRESS - OPEN ACCESS
ISSN: 2348-6600

http://www.ijcsjournal.com Volume 5, Issue 1, No 3, 2017 ISSN: 2348-6600
Reference ID: IJCS-166 PAGE NO: 1013-1017

Sri Vasavi College, Erode Self-Finance Wing 3rd February 2017
National Conference on Computer and Communication NCCC'17
http://www.srivasavi.ac.in/ nccc2017@gmail.com

an operational status. This is then used to verify if the outgoing interface selected by each individual packet is operational.

V. EVALUATION

I evaluate the implementation of MRC / IP FRR in terms of packet processing capabilities, i.e. the router's ability to sustain throughput during a failure. The machines used in the experiments are DELL 2950 servers with two dual core Xeon 5160 processors running at 3Ghz. They are equipped with 4096MB RAM and Intel Pro 1000PT (82571EB) network cards. Furthermore, they run a GNU/Linux operating system using linux kernel version 2.6.23.17 with our patch applied.

I use a basic ring topology consisting of four routers and four hosts as shown in Fig. 4. The routers run Quagga with OSPF where the link metrics are configured such that the green continuous lines are used in a failure-free environment. When testing failure operation I break connectivity between router 4 and 2. The link speed between all nodes is Gigabit ethernet running in a full-duplex mode.

VI. CONCLUSION AND FUTURE WORK

In this paper i have presented architecture for a practical IP fast reroute implementation using GNU/Linux. I have evaluated the performance of our implementation. Our test shows that MRC / IP FRR increase the resource usage in the router. However, in our lab setup I were still able to provide throughput that closely matches the performance during normal operation.Direction for future work include exploring the optimization aspects of the implementation. I expect that it is possible to gain some packet processing performance by using a zero copy approach to signalling. In this implementation the entire packet needs to be copied when updating the TOS field.

REFERENCES

[1] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving subsecond igp convergence in large ip networks," SIGCOMM Comput Commun. Rev., vol. 35, no. 3, pp. 35–44, 2005.

[2] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot, "Characterization of failures in an ip backbone," INFOCOM 2004. Tinty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, vol. 4, pp. 2307–2317, 2004.

[3] G. Iannaccone, C. nee Chuah, R. Mortier, S. Bhattacharyya, and C. Diot,
"Analysis of link failures in an ip backbone," in IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment. New York, NY, USA: ACM, 2002, pp. 237–242.

[4] A. Basu and J. Riecke, "Stability issues in ospf routing," in SIGCOMM
'01: Proceedings of the 2001 conference on Applications, technologies,architectures, and protocols for computer communications. New York, NY, USA: ACM, 2001, pp. 225–236.

[5] S. Bryant and M. Shand, "Ip fast reroute framework," IETF Internet Draft,draft-ietf-rtgwg-ipfrr-framework-08, February 2008, (Work in Progress).

[6] S. Bryant, M. Shand, and S. Previdi, "Ip fast reroute using not-via addresses," IETF Internet Draft, draft-ietf-rtgwg-ipfrr-notvia-addresses-02.txt, February 2008.

[7] "The quagga project," Online, 2008, http://www.quagga.net/. http://www.simula.no/departments/networks/.artifacts/infocom06finalhttp://www.simula.no/departments/networks/.artifacts/infocom06final Apr.-June 2005.