Architecture and Performance of a Practical IP Fast Reroute Implementation

Ole Kristoffer Apeland* and Tarik Čičić†
* Simula Research Laboratory, Martin Linges vei 17, N-1331 Fornebu, Norway
Email: oleap@simula.no
† Department of Informatics, University of Oslo, Gaustadelléen 23, N-0371 Oslo, Norway
Email: tarikc@ifi.uio.no

***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 we present the architecture and evaluate performance of the first known practical implementation of IP FRR. The arcitecture 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 re-convergence time to a sub-second timeframe in a controlled environment [1], normally the re-convergence time is unacceptable for time-critical applications.

Several studies have shown that network failures happen frequently [2][3][4]. Furthermore, they show that a significant portion of these events have a temporary scope where 50% last less than one minute[4]. Ideally these transient failures should be handled without a global re-convergence. Due to stability concerns there is a limit on how fast it is possible to trigger a global re-convergence[5].

In recent years a research area, IP fast reroute, has provided IP based networks with the means to recover from node and link failures almost instantaneous. This is achieved by invoking pre-calculated repair paths in the node that detects the failure. A framework is being adopted by the IETF[6]. Several schemes that utilize such *local, proactive* recovery are known, and include "NotVia" [7], "Multiple Routing Configurations" [8] and "Failure Insensitive Routing" [9]. Since the backup path can be selected locally, the detecting node needs not inform other routers. For the short-lived failures this may suffice since the pre-calculated information may be used until the component is operational again. For permanent failures the recovery mechansims extend the period available to perform the time-consuming global failure signaling and path calculations. This should allow the disruption time to be reduced to the time needed to detect the failure and invoke the recovery rerouting, can be done in tens of milliseconds.

IP FRR schemes and methods have so far been studied using simulation and analytical models only. There is a need to better understand the architectural constraints and the performance yield of the IP FRR. The mechanisms generally impose a higher resource usage in routers, and if the performance limitations are exceeded it could lead to an unstable network where the effect of a single failure propagates and disrupts traffic otherwise unaffected. In this paper we describe an implementation of IP fast reroute in software based routers running GNU/Linux.

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)[8] 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.

Normally, MRC uses the default, non-restricted topology for routing. If a failure is detected, the detecting node selects a

"backup topology" where the next hop (failed) node is isolated. This way a path that bypasses the failure is implicitly selected.

The MRC concept development has undergone several iterations. The most recent version called "Relaxed MRC"(rMRC)[10] is both simpler and has improved performance compared to its predecessors. Typically, 3-6 backup topologies are sufficient to guarantee recovery from any link or node failure in the network.

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. This design allows new routing and forwarding functionality to be easily implemented. Furthermore, the components used are usually cheap and well documented, and there exists a large selection of open source software for networking application. In our implementation we 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.

The GNU/Linux operating system is a moving target as continuous development takes place in nearly all areas. We give a high level view of the current mechanisms used, as that are less likely too change in a radical way in the near future.

1) Routing engine: For the routing engine we selected Quagga [11], 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. A large number of common unicast IPv4 and IPv6 IGPs daemons are available. Quagga is configured by activating daemons according to the desired routing protocols. These daemons are then tied together with a daemon named zebra, responsible to update the kernel routing table and pass routing information between the IGP daemons. As Quagga operates at the application level and only instruct the forwarding process of the kernel it is hardware independent and very portable. An example Quagga configuration that runs OSPF[12] is depicted in Fig. 1.

2) The kernel forwarding engine: Several components are combined in order to provide the IP layer functionality in the Linux network stack. Here we give a basic understanding of the design and key components used; the routing policy database (RPDB), Netlink[13], and Netfilter [14].

The packets are processed and distributed into different internal paths depending on a variety of aspects. In particular this is determined by the individual packets origin and protocol. For example the locally generated packets are submitted to a different routing decision, i.e. the logic surrounding a

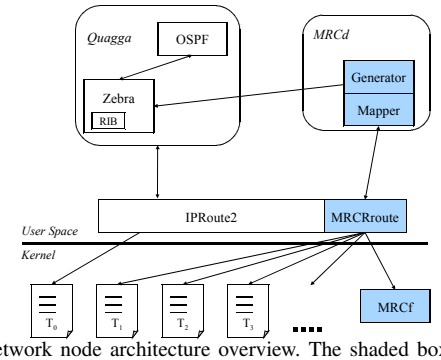

Fig. 1. Network node architecture overview. The shaded boxes implement the MRC functionality.

table lookup procedure, than packets received from a network interface. Due to optimizations, these differences are also reflected in the information available at the different processing paths and stages. E.g. the routing decision used for processing locally generated packets is not provided with an actual IP header, but operates on information that at a later stage will be used to generate the header.

The table lookup procedure in linux implements policy routing by using a set of rules and up to 255 different routing tables to extend the capabilities of the table lookup function. The collection of routing tables and rules is called the Routing Policy DataBase (RPDB). In the Linux kernel the rules are ordered by priority in a linear list that is used to evaluate both meta and header information of IP packets. Furthermore, the rules contain an action predicate commonly used to decide which routing table to consider when determining the next-hop. The rule matching and subsequent actions are performed in a linear manner until a route is selected or no more rules are left. The default behavior of Linux is to have three rules, matching all packets, that point to a local, main and default routing table. Additional user specified rules may for example be specified in order to tell the lookup procedure to consider specific routing tables based on which interface a packet arrived at.

Netlink is a general framework for message passing between single or multiple user-space processes and the kernel as well as intra-kernel communication. It is commonly used in the networking subsystem where it assists in tasks such as user-space management of routing tables and passing internal event notifications between the various sub components in the network stack.

Netfilter is a series of hooks located in the Linux network stack as shown in Fig. 2. Their placement allows packets to be intercepted at major points in the processing stages as well as at major forks or intersections in the path a packet may follow. Each hook maintains a chain of callback functions sorted based hook type and priority where various sub-processes in the network stack may register for additional packet processing. Netfilter is commonly in processes such as packet filtering and network address translation.

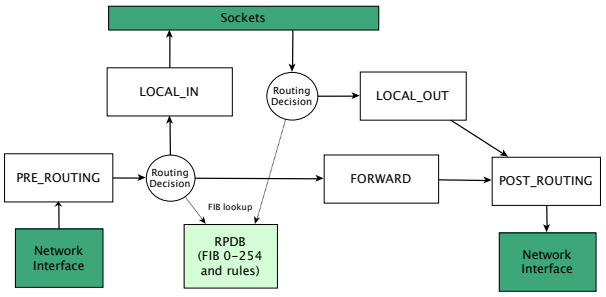

Fig. 2. A flow diagram showing placement of netfilter hooks (white boxes) and where in the process the routing decision takes place.

III. Design Challenges

The performance of a software router is governed by the capabilities of the hardware, and it will never be able to perform better than what is allowed by the weakest link in the forwarding chain. Since the routing and forwarding tasks are implemented in software all processes share the available resources. Several components may influence packet processing performance, including the cache, and computational resources provided by the CPU, the available memory and I/O bus bandwidth[15]. Deploying IP fast reroute mechanisms increases the resource usage, during both failure and normal operation situations.

IP fast reroute mechanisms use pre-calculated alternative paths to provide fast recovery. The mechanism we 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 we don't perceive additional memory requirements as a problem. However, the administrative overhead could potentially disrupt traffic if increased by a large factor. We choose to minimize RPDB administration overhead at the cost of using multiple routing tables. By doing this the size of routing table used during failfree 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, we 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.

A. MRC forwarding(MRCf)

In order to support IP fast reroute using MRC we extend the forwarding mechanism by adding two basic components to the linux forwarding engine; a general framework to support IP fast reroute mechanisms and an implementation of the MRC forwarding mechanisms that fits into this framework. The main components and their connections are shown in Fig. 3

1) Fast Reroute Framework: The framework is connected to the forwarding engine using the Netfilter framework. This allows the framework to register callback functions at several parts of the network stack. In general we 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.

We 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. However, when using IP fast reroute it could happen that a packet is recovered at a downstream router and that the recovery path of that packet includes the source router. Thus, in such an event the packet is moved from PRE_ROUTING, routed as though it was a locally generated packet, and subsequently released at the LOCAL_OUT hook.

Using netfilter hooks allows the framework to inspect which outgoing interface was intended to be used for each individual packet. It also makes it easy to toggle on or off both the framework and the recovery functionality, leaving IP fast reroute to be an option that may be easily toggled during operation.

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

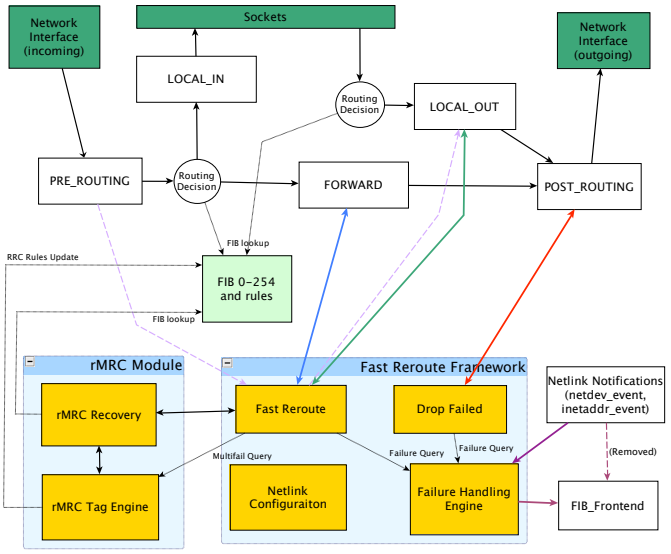

Fig. 3. An overview of the kernel modifications needed to implement the IP fast reroute framework and MRC module

The time it takes to respond to a failure is largely governed by the discovery time. Failures may be discovered actively using OSPF hello messages or standalone mechanisms such as the bidirectional forwarding detection protocol[16], or passively, by detecting loss of ethernet carrier. In this implementation we only implement detection trough listening to loss of carrier. The carrier loss detection time is determined by the specific hardware and driver capabilities. Some cards support notification of carrier status using interrupts while others may rely on the kernel to actively poll the status. Regardless of the method, they are generally able to detect an report such a loss at least within tens of milliseconds.

In the event of a failure the driver reports this using a netlink message. The default behavior within linux is to prune all routing tables entries containing the failed interface when such a message is received. To be able to identify the packets whos next-hop should have used the failed interface, we intercept the messages sent from the network card as they are delivered to the routing table subsystem. The messages are then either stored in a soft sate-waiting queue or, alternatively dropped, depending on the routing protocol used. For static routing the messages should be dropped. However, when using dynamic routing protocol such as OSPF, the message should be released after the re-convergance process, right before the new topology is reflected in the normal operation routing table. Furthermore, the messages are used by the framework to keep track of the interface status.

3) rMRC Tag engine: In order to signal to other routers that a packet has been recovered we tag the packet with a pre defined TOS value agreed upon by all routers. We use the bits 3, 4 and 5 in the TOS field, originally used to request low delay, high throughput and high reliability, respectively . This makes the signaling mechanism compatible with Differentiated Services but breaks backwards compatibility with with the ”DTR” bits as defined in RFC791[17]. MRC supports recovery from a single failure. Should a packet marked with a recovery TOS tag be received and use a failed interface it will be dropped.

In a real deployment the edges of the network should deploy a differentiated service in such a way that the values used for IP fast reroute signaling are unused for packets that have not been recovered. Failure to do this would lead to packets coded with ”DTR” bits to be forwarded according to a recovery topology. Furthermore, the presence of the recovery values is interpered as though the packet has been recovered once, forcing it to be dropped if it encounters another failure.

4) rMRC Recovery: In order to successfully recover a packet there are two main situations the implementation needs to cover; the initial recovery performed by the router adjacent to the failure, and subsequently the routers along the recovery path needs to be able to route the packet according to the selected recovery path. MRC is a recovery method that uses a multi topology approach to perform IP fast reroute. In our implementation we chose to use different tables to represent the different topologies. This approach reduces the processing time for routing table maintenance and table lookup for failure-free traffic, but increases the memory usage by a small amount and adds some more rules to the RPDB lookup procedure.

In order to forward a packet already recovered the RPDB is extended with rules allowing incoming packets marked with the predefined recovery TOS value to be forwarded according to the corresponding routing table.

The initial recovery is implemented using two new lookup functions, one for locally generated packets and one for packets arriving form external sources. Which one is used is determined by the source address of the packet. The lookup functions are provided by the framework and will generate a recovery cache entry from the resulting lookup. Furthermore, the cache entry obtained from the normal lookup is extended to contain a reference to the recovery TOS value.

V. Evaluation

We evaluate our 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.

We 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 we break connectivity between router 4 and 2. The link speed between all nodes is Gigabit ethernet running in a full-duplex mode.

Traffic is generated using Iperf[18], a tool designed to measure various metrics of a network. Iperf runs in a client/server

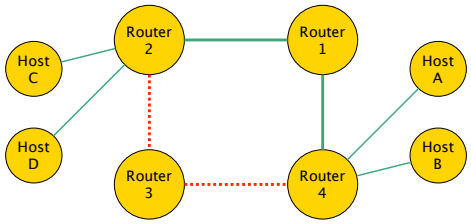

Fig. 4. A minimal topology consisting of a ring and two hosts.

configuration where the client generate traffic destined for the server using either TCP or UDP as the transport protocol. When using UDP the Iperf client sends packets with a predetermined payload at a constant bit rate determined by the requested payload and bandwidth. Each packet is given a sequence number and a time stamp that the server uses to calculate inter arrival times, loss and out-of-order delivery.

We choose to use the UDP protocol for measuring packet processing capabilities as it gives a best effort delivery without adapting to the actual network performance. This allows us to focus on the router and network performance rather than evaluating traffic engineering aspects of IPFRR schemes or the performance of specific TCP implementations.

When failing the link between router 4 and 2 the traffic from host A and B will follow the dotted path, traversing router 4, 3 and 2 before delivered to Host C and D, respectively. With this failure present, traffic from Host C or D will follow the path 2, 1, before being recovered using the path 2, 3, 4.

Sources of inaccuracy Due to clock resolution and drift the measurements may have some inaccuracies. Furthermore, routing traffic and running measurement tools consume considerable resources. Varying link quality due to external or internal interference may also add to the potential sources of inaccuracy. We use long test periods to get a high number of samples as a basis for our results. Furthermore, we try to smoothen the clock skew by using local NTP servers.

A. Bandwidth

In all tests we generate two simultaneous UDP streams. The streams flow from host A and B to C and D, respectively. We run each stream for 60 seconds. Since the streams are started from two separate hosts, we disregard the results from the first and last 500 ms of the measurement interval, and monitor that both the streams are started within this timeframe. Thus, data is collected for 59 seconds in 500ms intervals. In order to measure recovery throughput in the full period we temporarily disable OSPF during these tests.

The average maximum achieved bandwidth during normal and recovery operation, using different packet sizes is shown in Table I. The values are obtained from measurements where the generated bandwidth exceeded the link capabilities. Furthermore, the theoretical maximum throughput is included as a reference point.

In a failure free environment the routers are capable of delivering very close to maximum theoretical throughput.

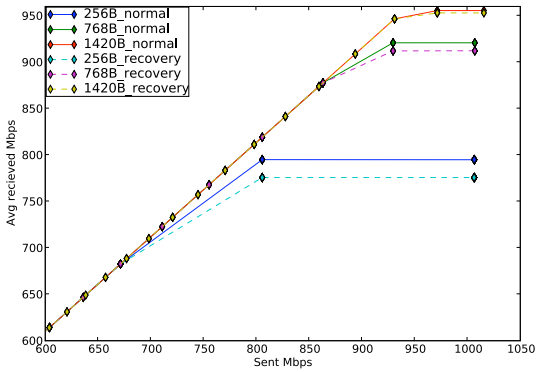

Fig. 5. Sustained bandwidth during recovery and normal operation.

During recovery the throughput delivered ranges between 97.5 and 99.7% of theoretical maximum bandwidth.

Fig. 5 show the average achieved throughput at increasingly higher bit rates. Again, the legend indicates payload size in bytes and the operational mode during the forwarding process. Furthermore, Fig 6 shows the average number of packet processed per second obtained from the same experiments. The achieved throughput increases in a linear manner until near link saturation, at which point the throughput obtained during normal operation exceeds recovery operation.

The difference in bandwidth observed near link saturation may originate from the number of memory I/O operations performed during the recovery process. The router is able to sustain a linear increase in packets processed per second when sending smaller size packets, which indicates that the computational resources available are sufficient. However, the signaling mechanism in the recovery process requires the packets to be copied when the TOS field is updated. Furthermore, the netfilter framework make another copy of the packet when the IP header is changed. Thus, the number of memory I/O operations due to an increased number of copy operations of the packet in conjunction with a large number of small packets could reduce the throughput.

TABLE I
TABLE SHOWING THE MEASURED AVERAGE THROUGHPUT IN MBPS DURING LINK SATURATION.

Payload	Theoretical max	Normal	Recovery
256B	795.0	795.0	775.7
512B	885.8	885.6	873.7
768B	920.9	920.9	912.2
1024B	939.4	939.5	932.6
1280B	951.0	950.8	945.3
1420B	955.6	955.6	953.0

Packet processing time in the IP layer In these tests the results were obtained using the netfilter framework. We have written a small kernel module that provide each packet with a time stamp when it arrives at the PRE_ROUTING hook. Furthermore, a time delta is computed when the packet leaves the POST_ROUTING hook.

Table II shows the processing time at the IP layer in nanoseconds. The results were obtained by logging the pro-

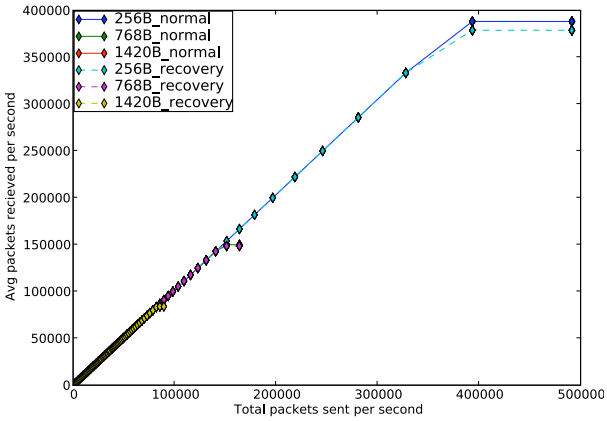

Fig. 6. Packets processed per second during recovery and normal operation

cessing time for the first 100.000 packet in a stream. We repeated the test varying the payload size and operational mode. Furthermore, MRC might need two table lookups when the destination node is the next-hop node and the link has failed. We tested the performance when of this mechanism by manually edit the MRC modules configuration.

The results show that the recovery look procedure increases the processing time by a factor of 2.3 - 3.4. However, the impact of performing additional table lookups does not seem to have a large effect on the processing time.

TABLE II
TABLE SHOWING PROCESSING TIME IN NANO SECONDS FOR VARIOUS OPERATIONAL MODES AND PACKET SIZES. "RECOVERY 2" INDICATES THAT TWO TABLE LOOKUPS WERE NEEDED TO RECOVER A PACKET.

Payload	mode	2.5th percentile	median	97.5th percentile
256B	normal	664	791	998
	recovery	1452	1846	2234
	recovery 2	1458	1861	2261
1420B	normal	562	689	908
	recovery	1497	1891	2276
	recovery 2	1440	1846	2225

VI. Conclusion and Future Work

In this paper we have presented an architecture for a practical IP fast reroute implementation using GNU/Linux. We have evaluated the performance of our implementation. Our tests shows that MRC / IP FRR increase the resource usage in the router. However, in our lab setup we 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. We expect that it is possible to gain some packet processing performance by using a zero copy approach to signaling. In this implementation the entire packet needs to be copied when updating the TOS field.

In this paper we do not consider the traffic engineering part of IP fast reroute, but assume that recovered traffic does not overload the links used to recover the traffic. However, it should be noted that the volume of the recovered traffic might lead to a congestion in other parts of the network. In case of a failure the recovered traffic may congest other outgoing interfaces of the router. In this case it is not clear which, if any, steps should be taken to combat this problem.

References

[1] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving sub-second 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. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 4, pp. 2307–2317, 2004.

[3] D. Watson, F. Jahanian, and C. Labovitz, "Experiences with monitoring ospf on a regional service provider network," *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*, pp. 204–213, 19-22 May 2003.

[4] 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.

[5] 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.

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

[7] 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.

[8] A. Kvalbein, A. F. Hansen, T. Cicic, S. Gjessing, and O. Lysne, "Fast ip network recovery using multiple routing configurations," in *INFOCOM 2006*, Z. Z. Arturo Azcorra, Joe Touch, Ed. Barcelona, Spain April 23 – 29: IEEE, 2006, pp. xx–yy. [Online]. Available: http://www.simula.no/departments/networks/.artifacts/infocom06final

[9] S. Lee, Y. Yu, S. Nelakuditi, Z.-L. Zhang, and C.-N. Chuah, "Proactive vs. reactive approaches to failure resilient routing," in *Proceedings INFOCOM*, Mar. 2004.

[10] T. Cicic, A. F. Hansen, A. Kvalbein, M. Hartmann, R. Martin, and M. Menth, "Relaxed multiple routing configurations for ip fast reroute," in *IEEE/IFIP Network Operations and Management Symposium (NOMS'08)*, 2008.

[11] "The quagga project," Online, 2008, http://www.quagga.net/.

[12] "Rfc 2328, ospf version 2," Online, April 1998, http://www.ietf.org/rfc/rfc2328.txt.

[13] "Rfc3549, linux netlink as an ip services protocol," Online, 2008, http://www.ietf.org/rfc/rfc3549.txt.

[14] "The netfilter.org project," Online, 2008, http://www.netfilter.org/.

[15] O.-I. Lepe-Aldama and J. Garcia-Vidal, "A performance model of a pc based ip software router," *Communications, 2002. ICC 2002. IEEE International Conference on*, vol. 2, pp. 1230–1235, 2002.

[16] "Bidirectional forwarding detection," February 2008, draft-ietf-bfd-base-08.txt draft-ietf-bfd-base-08.txt (Work In Progress).

[17] "Rfc791, internet protocol," Online, September 1981, http://www.ietf.org/rfc/rfc791.txt.

[18] "Iperf," Online, 2008, http://dast.nlanr.net/Projects/Iperf/.