Increasing Router Availability through Virtualization

Sergio Correia*†, Omar Cherkaoui*
*Laboratoire de téléinformatique et réseaux (LTIR)
Université du Québec à Montréal (UQAM)
Montréal, Canada QC H3C 3P8l
sergio.correia@netvirt.ca, cherkaoui.omar@uqam.ca

Joaquim Celestino Júnior†
†Graduate Program in Computer Science
State University of Ceara (UECE)
Fortaleza, Brazil 60.740–000
celestino@larces.uece.br

***Abstract*—Router high availability is an important challenge in network management. In fact, the breakdown of a router can be expensive for the network, as it can lead to congestion and to the loss of connections or packets – which, in turn, translates into loss of money. Vendors usually achieve carrier-grade availability by having standby routers ready to take over when a failure condition hits the active router(s). In such redundancy scheme, it is needed to duplicate practically every hardware component (e.g. the router switching processors and line cards), what makes the final high availability router very expensive. In this paper, we propose a redundancy model that tries to increase the overall router availability – by exploiting virtualization –, while also reducing the final device cost – by not having to duplicate every piece of hardware to obtain the desired availability level. The analytical results obtained show the proposed approach was able to achieve the intended goals.**

I. Introduction

Router high availability is an important challenge in network management, mostly because any problem in the routing/switching infrastructure results in downtime which, in turn, translates into loss of money. With the ever-growing appearance of Internet-based mission-critical applications, we experience nowadays for instance, there is a huge demand for more reliable and highly available systems.

A highly available system is one that is usable when the customer needs it [1], and the availability concept can be expressed mathematically in terms of two metrics: the *mean time between failures of a system* (MTBF) and the *mean time to repair a system* (MTTR). MTBF is a number that describes the number of hours (in average) between failures for a particular device or system, while MTTR is that amount of time (also in average) that elapses between the failure of a device/system and this device or system being restored to the proper working order [2]. Equation (1) expresses how the MTBF and MTTR concepts are mathematically related with the availability of a system.

$$Availability = \frac{MTBF}{MTBF + MTTR} \tag{1}$$

In order to achieve the demanded router availability levels, the industry typically uses redundancy schemes that require to duplicate practically every component of the system, such as the route processor and the switch fabrics, which results in a very expensive final system.

978-1-4577-1261-6/11/$26.00

Router virtualization strategies allow a single device to create multiple virtual or logical routers, and currently the research community and the industry is working on how to exploit virtualization to increase the overall availability of a router while reducing the costs at the same time.

This work proposes a redundancy model that tries to: (i) increase the overall router availability by exploiting virtualization; and (ii) reduce the final device costs, by not having to duplicate every piece of hardware to achieve the desired availability level.

The remainder of this paper is divided as follows: section II introduces the router architectures and the redundancy schemes used to achieve high availability; section III explains how existing (physical) routers are built having in mind protection; section IV introduces our slices (or virtual routers); section V explains the protection scheme used in the virtual slices; section VI evaluates a few router models taking into account the proposed slices; and finally, section VII concludes this work.

II. Routers and redundancy models

A. Router architectures

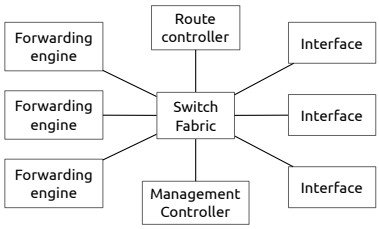
Regarding their architecture, routers are typically divided into two categories: *centralized* and *distributed* [3], as shown in Figure 1. The main difference between these two approaches to building routers is that, in the distributed architecture, we have the forwarding engine integrated into the interfaces themselves [3].

B. Redundancy models

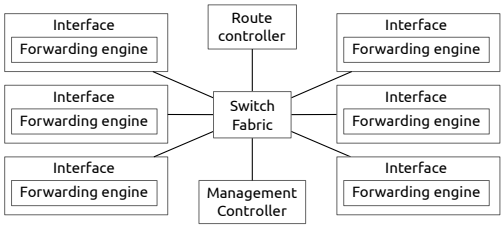
Regarding redundancy, the most common forms are the passive and active redundancy [4], each with advantages and drawbacks.

Passive redundancy (1:1, N:1): in this scheme, backup fabrics are installed in addition to the active ones, and when a failure condition happens, one of the backup fabrics take over automatically. This is the simplest approach to implement, but has the drawback that, when the switchover is made, any data in the active fabrics is lost.

Active redundancy (1+1): this scheme supports loss-less switchover to backup fabric devices and cards, but it is the most expensive to implement. This configuration features two sets of active fabrics carrying the same traffic, with only one set connected to the outputs. If a problem occurs, there is an automatic switchover to the redundant fabrics.



(a) centralized



(b) distributed

Fig. 1. Router architectures

III. Protection model in physical routers

Physical routers typically employ one of the redundancy schemes mentioned in section II in order to achieve high availability. As an example, Cisco ASR-9000 series [5] has full hardware (and software) redundancy: it has redundant RSPs (route switch processors), redundant switch fabrics, redundant power supplies, redundant fan trays and redundant Ethernet out-of-band management [6]. If on the one hand, such systems are able to service high availability, on the otherhand, the costs for duplicating every piece of hardware are considerably high.

Working with the N+M (N active routers, M standby routers) redundancy model, Tsai *et al.* [7] showed that, for up to 47 active routers, only one standby router is required to achieve a five-nine carrier-grade availability. They made use of a Continuous-Time Markov Chain (CTMC) and proposed an availability function that can be computed in time $O(N^3M^3 \log N)$ to determine the minimum number of standby routers in a high availability (HA) router needed to meet a desired availability ρ. They also showed, with their analytical results, that the failure detection and recovery rate δ is a key parameter for increasing the availability of a HA router, and having a goal to increase the failure detection and recovery rate, they also proposed a middleware to help reduce the takeover delay and meet the carrier-grade availability with five nines. They implemented their proposed HA-OSPF on a PC platform, using the 2+1 redundancy model, and compared their performance – more specifically, their *takeover delay* – against Cisco's ASR-1000 series and Juniper's MX series. Their results are summarized in Table I.

From Table I, we see their proposed solution performed slightly better than Cisco ASR-1000 series and Juniper MX series.

Understanding a bit how physical routers are built taking availability into account nowadays, in the next section, we introduce our *slices* or virtual routers.

TABLE I
Takeover delay from HA-OSPF, Cisco ASR-1000 series and Juniper MX series routers, taken from [7]

	Takeover delay
HA-OSPF	189ms
Cisco ASR-1000 series	about 200ms
Juniper MX series	300ms

IV. Virtual routers

In our redundancy model, we introduce our *slices* – virtual routers that exist in the domain of the line card. Each line card can create and manage several slices; a slice can protect another slice, similarly to the active redundancy model, so that if a slice fails, we have another slice ready to take over. By exploiting live migration of virtual routers [8]–[10], we are able to achieve stateful switchover in case a slice fails; another slice will be protecting it and can take over with no impact in forwarding traffic.

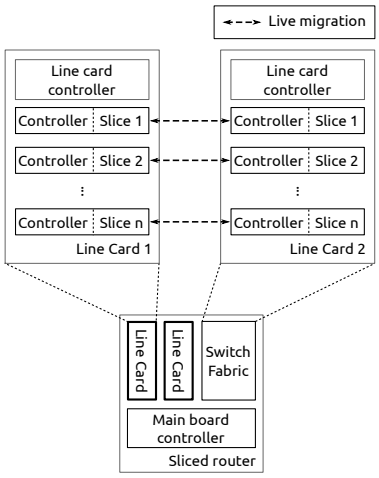


Fig. 2. High-level design of a router that supports several slices

Figure 2 shows the generic high-level design of a router that supports several slices. It is composed of *line cards*, *switch fabric* and a *main board controller* that manages and configures the router components (line cards, switch fabrics and interfaces) – an example of Main board controller could be Cisco IOS [11]). The line cards are composed of a *line card controller* – the *Slicer* – and slices, each of which is composed of a controller as well (an OS controller), which is responsible for communicating with the line card controller and for configuring the slices themselves. The *Slicer* can be installed on a hypervisor such as KVM [12] or XEN [13], to isolate the code (slices).

A. Estimating the MTTR of a slice (without considering redundancy)

The process of booting a slice (i.e. a virtual router) is similar to the process of booting a physical router. In fact, booting a slice starts with the hardware allocation, then the image loading and mounting, and finally, the configuration loading.

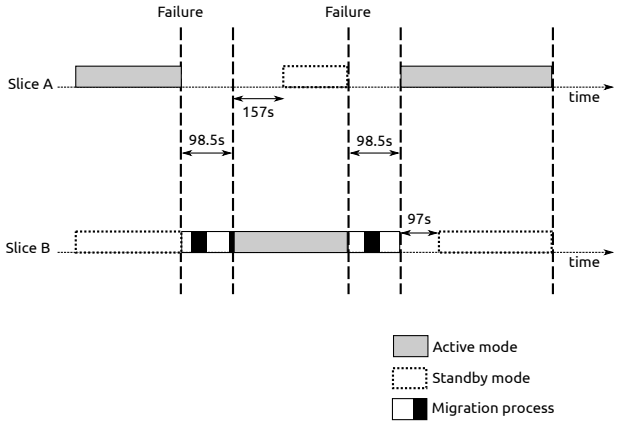


Fig. 3. Slice B protecting Slice A: live migration taking place after a failure happens with Slice A, then again when a failure happens with Slice B

The reboot sequence for a virtual router may be divided into five main steps, namely: (i) the *hardware allocation*; (ii) the *loading of the image*; (iii) the *loading of the configuration*; (iv) the *booting of the slice internal controller on the slicer hypervisor kernel*; and (v) the *restarting of a virtual machine booted on the hypervisor*. The time needed to perform these tasks will be modeled by five variables T_1, T_2, T_3, T_4 and T_5, and therefore, the total time needed to repair a silce is the sum of the individual times, as described in Equation (2).

$$T = \sum_{i=1}^{5} T_i \tag{2}$$

T_1 is the time needed for the hardware allocation, and in our case, it can be considered negligible, when compared to the time the other operations take. Hence, we have $T_1 = 0$

T_2 is the time needed to load the virtual router image. This image can be stored in the local disk of the same physical machine or in a remote storage device. Therefore, the time to load it depends basically on the size of the image file and the transfer rate. Table II gives the transfer rates for loading the virtual router image stored in different locations.

TABLE II
DATA TRANSFER RATES FOR LOADING A VIRTUAL ROUTER IMAGE STORED IN DIFFERENT LOCATIONS [14], [15]

Transfer Rate	Where the image is stored
13MB/s	local disk
35MB/s	NFS server
25MB/s	iSCSI server

Being S_{im} the image size and R_{im} its transfer rate – which depends on where the image is stored –, we can express T_2 as

$$T_2 = \frac{S_{im}}{R_{im}} \tag{3}$$

T_3 is the time to load the configuration, the time needed to reconfigure the router after a crash. We assume that the router configuration files are stored at two different places, in the global virtual router image, and also somewhere else, so that it allows us to avoid reloading the entire image in case of a configuration failure. Being S_{cf} the size of the configuration file and R_{cf} its transfer rate, we can express T_3 as

$$T_3 = \frac{S_{cf}}{R_{cf}} \tag{4}$$

T_4 is the time to boot the operating system on the slicer hypervisor kernel. In other words, it is the time needed to boot the Xen kernel, since we are using Xen in our experiments. In general, this time is fixed, and booting a normal Linux OS takes around 60 seconds. For example, with the Red Hat $AS4$ distribution, booting it takes about 58.9 seconds [16]. We measured the time needed for booting the Fedora 8 distribution on the Xen kernel, and we found it to be around 56 seconds. For our modeling, we are going to consider the time needed to start a Xen-Linux machine to be 60 seconds. Hence,

$$T_4 = 60s \tag{5}$$

T_5 is the time to restart a virtual machine booted on the hypervisor. That means to start the operating system of the virtual machine after loading the image. We measured this time for a Fedora 9 virtual machine booted on the QEMU hypervisor, and we found it to be around 18 seconds. We assume then, that the time to start a virtual machine on a hypervisor is equal to 20 seconds.

$$T_5 = 20s \tag{6}$$

Our modeling on the time it takes to repair a slice is described in Equation (7).

$$T = 80 + \frac{S_{im}}{R_{im}} + \frac{S_{cf}}{R_{cf}} \tag{7}$$

where, as indicated previously, S_{im} and S_{cf} represent the size of the image and the configuration file, respectively, and R_{im} and R_{cf} represent the transfer rate of the same image and configuration file, also respectively.

Table III resumes the time of router restoration for each type of failure.

We detail in what follows, examples of each failure type:

- Slice – OS bug
- Configuration – virtual interface address, Forwarding Information Base (FIB) corruption, access lists erased
- Slice controller – bugs
- Slicer – Xen-kernel error, Dom0 error

The restoration process: To restore a slice from a failure, we expect to do it on the same physical machine and the same domain, by rebooting it or reloading the most recent working configuration. In fact, if the failure happens with the routing information table, a corruption in the virtual interface address or in case of virtual machine failure (like DomU bugs), we can just restart the virtual machine by loading the most recent stored image. The slice will be still down until the reboot ends successfully. Figure 4 shows the restoration of two cases of failure: configuration loss and DomU bug.

TABLE III
RESTORATION TIME BY TYPE OF FAILURE

Failure type	Restoration Time
Slice	$T_{vm} = 20 + \frac{S_{im}}{R_{im}}$
Configuration	$T_{config} = \frac{S_{cf}}{R_{cf}}$
Slice controller	$T_{os} = 80 + \frac{S_{im}}{R_{im}}$
Slicer	$T_{hv} = 20 + \frac{S_{cf}}{R_{cf}}$

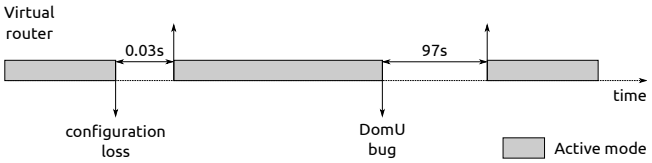


Fig. 4. Virtual router restoration with no redundancy

B. Estimating the restoration time of a slice (considering redundancy)

The restoration process: In some cases, we cannot recover the router on the same domain. When that happens, we exploit the concept of resource's redundancy to restore the failed domain in another one on the same hardware or another host. The restoration operation based on redundancy requires two routers – one is in active mode, and the other is in standby mode. When a failure is detected on the active router, its traffic will be migrated to the standby one, while the repair process is taking place. This migration process takes around 98.5 seconds, and leads to a downtime of 563ms [8]. After that, we expect to have a second domain started in the same or in a different host in *standby* mode, and it will be ready to replace the failed router. This process is shown in Figure 5

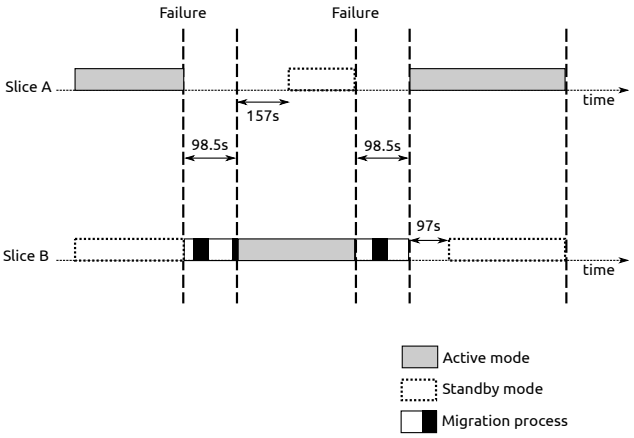


Fig. 5. Restoration process using redundancy

In Figure 5, we have *Slice 1*, which is working well until it suffers a hardware failure. After the failure is detected, the restoration process starts by migrating the configuration of *Slice 1* to *Slice 2*, which is in standby mode. After the migration operation ends, *Slice 2* will support all functionalities of *Slice 1*, while *Slice 1* goes to a reparation process. After the reparation process takes place, *Slice 2* completes his routing operations, and *Slice 1* is restarted in standby mode. When *Slice 2* is down, the same process happens again, but now *Slice 1* is the standby router.

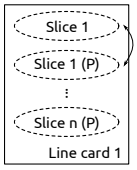
Reboot time function: The time expression of restoration using the redundancy concept is similar to restoration without redundancy approached in IV-A. The only difference in terms of restoration time is the time needed to migrate the router. Basing on statistics from the VROOM project [9], this time is equal to 98.5 seconds.

$$T_6 = 98.5s \tag{8}$$

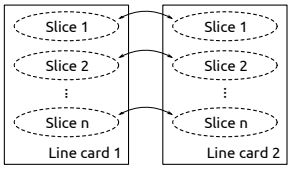
However, the migration operation causes a downtime of 563 ms [9].

V. PROTECTION MODEL FOR VIRTUAL ROUTERS

For virtual routers or slices, we can have two types of protection: (i) protection within the same line card; and (ii) protection in a different line card. Figure 6 shows these two protection approaches.



(a) Slice protection in a single line card



(b) Slice protection with more than one line card

Fig. 6. Slice protection schemes

A. Slice protection analysis

In this section, we are going to develop simple equations to model the availability of the line cards considering their slices. In this context, A_L represents the availability of the line card, A_S is the availability of the slice, n is the number of line cards and m, the numbers of slices *per line card*.

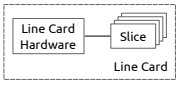


Fig. 7. Line card with slices

In Figure 7, we see a line card with m slices. Each of these slices is a virtual router, and hence, in this line card we have

now m parallel data paths. Equation (9) models the availability of each of these data paths.

$$A_{DP} = A_L \cdot A_S \tag{9}$$

In Equation (9), A_{DP} represents the availability of a data path. Since we have m data pahts (m slices), we have to consider the parallel availability of the data paths. Equation (10) calculates the global availability of a singe line card with m slices, represented by A_{SL}.

$$A_{SL} = 1 - (1 - A_{DP})^m \tag{10}$$

Combining Equations (9) and (10), we have the Equation (11), that represents the availability of a sliced line card (A_{SL}) in function of the regular availability of the line card and the availability of a slice.

$$A_{SL} = 1 - (1 - A_L \cdot A_S)^m \tag{11}$$

Applying Equation (11) to n sliced line card (each with m slices), we obtain Equation (12), that represents the availability of the parallel block of line cards.

$$A_{LC} = 1 - [1 - (1 - A_L \cdot A_S)^m]^n \tag{12}$$

VI. Numerical Evaluation

In this section, we analyze the availability of eight routers. They are based on a classical router model and are depicted in Figure 8. We are going to consider their regular availability as well as their availability using the proposed slice redundancy model.

Figure 8 shows the RBDs for the evaluated models, and Table IV lists the MTBF and MTTR for their components, which will be used in the availability calculations.

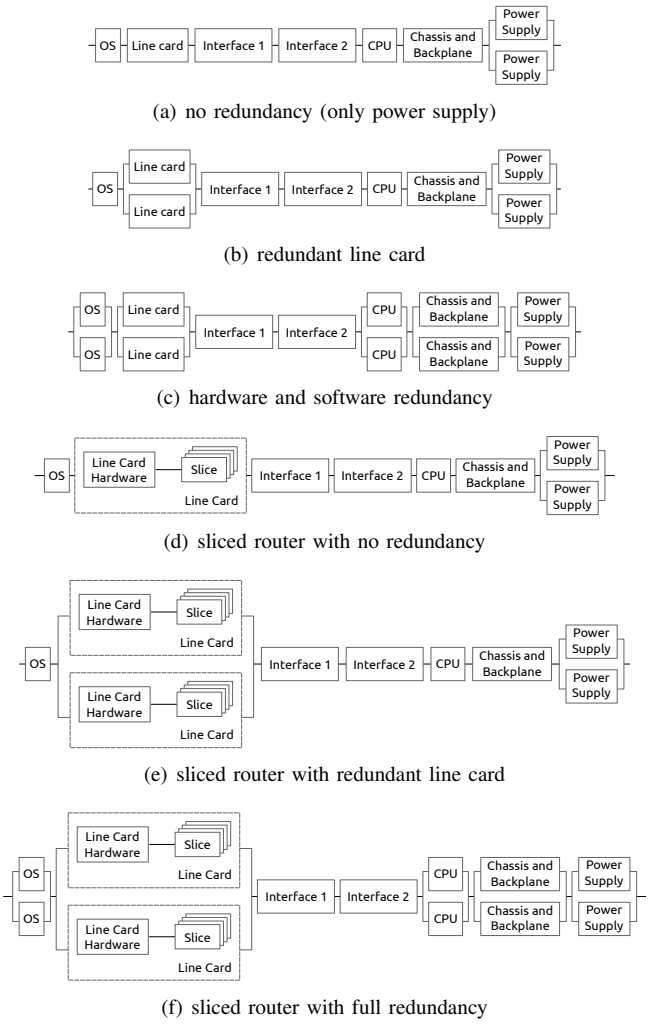
TABLE IV
MTBF and MTTR (both in hours) for the components of a generic router. Data from [17] and [18]

	MTBF	MTTR
Operating System	30000	0.1
Line Card	110000	4
Interface 1	1120000	4
Interface 2	600000	4
CPU	490000	4
Chassis and Backplane	460000	8
Power Supply	750000	4
Slice	7000	0.0275

A. Results

To calculate the availability of a line card considering their slices, we use Equation (12) and the data from Table IV, which has the MTBF and MTTR for a line card and a slice as well as for the other components of our router model.

The availability results are summarized in Table V and plotted in Figure 9, and as we can see, the proposed redundancy approach using slices was effective as in it could increase the overall router availability. Table VI shows that the proposed



(a) no redundancy (only power supply)

(b) redundant line card

(c) hardware and software redundancy

(d) sliced router with no redundancy

(e) sliced router with redundant line card

(f) sliced router with full redundancy

Fig. 8. RBDs for a classic router model considering redundancy and slicing

redundancy model achieved its goal, increasing the router availability without duplicating every hardware component. We can see that the Router depicted in Figure 8(a) achieved 99.996% service availability (with 4 slices), comparable to Router from Figure 8(b). The latter has a redundant line card, while the former does not.

We note the availability increases with the number of slices, up to a limit – when the global line card availability is close to 1.0. The results show that the sliced version of the router with low redundancy (Figures 8(a) and (d)) achieved higher availability than the regular (non-sliced) version of the intermediate redundancy router (Figure 8(b)), without requiring the extra line card the intermediate redundancy router has. We were then able to achieve higher availability at a cheaper price.

VII. Conclusion

This work proposed a redundancy model with slices (virtual routers) that aims to increase the overall router availability. In the proposed model, we have slices protecting other slices in case of failures, and with a simple analytical model, we

TABLE V
AVAILABILITY RESULTS FOR THE CLASSICAL ROUTER MODELS DEPICTED IN FIGURE 8

	No slicing		
	Router 8(a)	Router 8(b)	Router 8(c)
	0.999924514066016	0.999960873635270	0.999989760254930
	Slicing (n slices per line card)		
	Router 8(d)	Router 8(e)	Router 8(f)
1 Slice	0.999920605646188	0.999960873335756	0.999989759955407
2 Slices	0.999960873335756	0.999960874957436	0.999989761577135
3 Slices	0.999960874957371	0.999960874957436	0.999989761577135
4 Slices	0.999960874957436	0.999960874957436	0.999989761577135

TABLE VI
COMPARISONS OF THE ROUTER MODELS

	Low Redundancy	Intermediate Redundancy	Full Redundancy
Availability	0.99992	0.99996	0.99998
Availability with 4 slices	0.99996	0.99996	0.99998
Cost	High	Higher	Even higher
Duplicated units	Power Supply	Power Supply, line card	Power Supply, OS, Backplane, CPU, line card

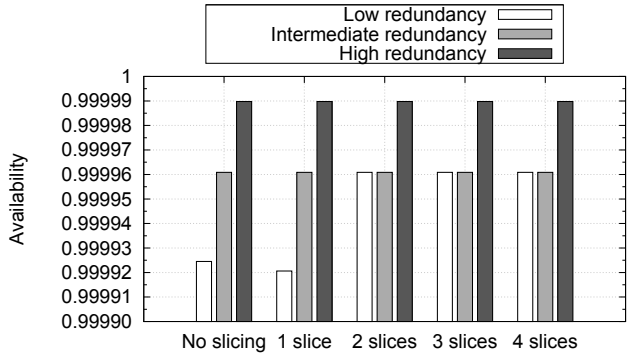


Fig. 9. Availability results for the router models evaluated

showed this scheme can achieve higher availability than a non-virtualized router. The proposed model is also interesting because it reduces the final costs of the network device, as we do not need full hardware redundancy, meaning we do not have to duplicate every piece of hardware to achieve higher availability degrees.

In the proposed approach, the availability increases with the number of slices, up to a limit, so the more slices we have, the higher the global line cards' availability will be, and therefore, the overall router availability will also be higher.

REFERENCES

[1] Cisco, "High availability initiative – Beat the downtime," 2001.

[2] C. Oggerino, *High availability network fundamentals: a practical guide to predicting network availability*. Cisco Press, 2001.

[3] H. Chao and B. Liu, *High performance switches and routers*. Wiley-IEEE Press, 2007.

[4] S. Stanley, "Packet Switch Chips," *Light Reading*, vol. 10, p. 17, 2002.

[5] Cisco, "Cisco ASR 9000 Series Aggregation Services Routers," 2011.

[6] ——, "Best practices to deploy high-availability in Service Provider Edge and Aggregation Architectures," 2010.

[7] C.-T. Tsai, R.-H. Jan, and K. Wang, "Optimal redundancy allocation for high availability routers," *International Journal of Communication Systems*, 2010.

[8] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 273–286.

[9] Y. Wang, J. van der Merwe, and J. Rexford, "VROOM: Virtual routers on the move," in *Proc. ACM SIGCOMM Workshop on Hot Topics in Networking*. Citeseer, 2007.

[10] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford, "Virtual routers on the move: live router migration as a network-management primitive," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 231–242, 2008.

[11] V. Bollapragada, R. White, and C. Murphy, *Inside Cisco IOS software architecture*. Cisco Systems, 2000.

[12] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the Linux virtual machine monitor," in *Proceedings of the Linux Symposium*, vol. 1, 2007, pp. 225–230.

[13] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*. ACM, 2003, pp. 164–177.

[14] L. Mirtain and J. Szpyrka, "Solution de stockage répartie sur les centres de recherche INRIA à base de serveurs de fichiers de type "NAS"."

[15] Y. Lu and D. Du, "Performance study of iSCSI-based storage subsystems," *Communications Magazine, IEEE*, vol. 41, no. 8, pp. 76–82, 2003.

[16] W. Huang, J. Liu, B. Abali, and D. Panda, "A case for high performance computing with virtual machines," in *Proceedings of the 20th annual international conference on Supercomputing*. ACM, 2006, pp. 125–134.

[17] "Best practices to deploy high-availability in Service Provider Edge and Aggregation Architectures," 2010.

[18] Cisco, "Cisco ASR 9000 Series Ethernet Line Cards," 2011.