Distributed Systems Performance on Resilient Overlay Networks

Kyle Vincent Moses* kmoses@cs.duke.edu
Alexandros-Stavros Iliopoulos† ailiop@cs.duke.edu

April 29, 2012

Abstract

A primary benefit of Resilient Overlay Networks (RONs) is their ability to greatly improve the reliability of Internet packet delivery by detecting and recovering from outages and path failures more quickly than current inter-domain routing protocols. They accomplish this feat at the application layer by reducing the size of the problem to only include member nodes. First generation RONs could support networks of approximately 50 nodes and recent improvements have increased this size to approximately 300 nodes. RON systems have been shown to reduce link outage during network re-convergence to around 18 seconds instead of several minutes as can often occur in BGP. Most distributed applications are sensitive to intermittent network outages, and often are used to carry sensitive real-time information. Both RON papers discuss VPNs and other time-sensitive applications, but their experiments only show how RONs impact route convergence in general terms, without showing the end impact for specific applications. We seek to confirm the claims of the RON papers and build a Ruby-based application interface to allow any application to use a RON for communication, as well as provide a realistic and easy to set up network environment that can be used as testing grounds for a wide range of experiments.

1 Introduction

Network security, at its core, is a rather dichotomous concept. The sole purpose of a network is communication; generally, the more open and trusting a network, the more easily it facilitates this communication. In contrast, the most secure system is one that is entirely isolated, yet what benefit can be obtained from such a device? In the end, any networked system is forced to sacrifice absolute security in order to allow productivity and growth.

In the past two decades, the Internet has seen exponential growth and has become the heart of global communications. This rapid growth has certainly been facilitated by technological advances but is also attributed to the open trust nature of the Internet Protocol suite. Countless tools have been developed and applied across the Internet to improve security in this period, but despite all these efforts the growth of the Internet has been accompanied by parallel growth in cyber-crime.

*M.S. candidate, department of Computer Science, Duke University.
†Ph.D. candidate, department of Computer Science, Duke University.

Hewlett-Packard recently funded a study of 50 large companies and found that the average cost of cyber-crime per company was approximately $6 million annually, with at least one successful attack per company per week [6]. Additionally, a series of high profile attacks against Sony's PlayStation Network last summer cost approximately $189 million [1]. In addition to these direct financial costs, attacks increasingly result in the exposure of privacy data and other sensitive information. Stolen privacy data is the driving factor in growing costs associated with identity theft [7]. Hacktivist groups (i.e. hackers with a social/political agenda), such as *Anonymous*, specifically target such information to undermine companies and groups they find to be at fault. Their most recent publicized attacks involved a radical neo-Nazi political group and a mining company in Finland [5]. In both cases *Anonymous* released individual names and other contact information encouraging harassment of members of both organizations.

In addition to global financial implications, cyber-attacks are increasingly being acknowledged as a serious threat by governments around the world including the US and China. Both countries have recently activated full military commands to deal with such threats and expect cyber-warfare to be a reality in the 21st century. As an example, the Stuxnet virus that disabled Iran's nuclear enrichment centers is widely speculated to have been developed and placed by foreign government agents [4].

These examples clearly show the costly nature of network attacks and demonstrate the need for effective security. Unfortunately, these examples also demonstrate that network attacks continue to occur with security systems in place. If a method of communication exists, it can be exploited for unintended use; and for every new security patch applied to a system a new exploit is discovered. In light of these continually emerging threats, how can an organization achieve acceptable levels of security?

Virtual Private Networks (VPNs) and network tunneling in general are crucial elements of such security for modern commerce and government. They allow organizations to exchange sensitive information over the Internet and other public networks with full confidence and in many ways have fueled the growth of the Internet and the Information Age. For proof, we need look no further than Netscape's development of SSL tunneling in 1994 and the explosive growth of e-commerce and sites such as Amazon.com over the last two decades.

Unfortunately, the story does not end with VPNs saving the world before bedtime! As discussed above, the financial success of e-commerce has provided plenty of motivation for criminals and like most other security measures numerous exploits against VPNs, tunneling, and encryption systems continue to be discovered and patched in a perpetual digital arms race.

One side effect of efforts to secure VPNs is a lowered tolerance to network instability. Well-designed modern VPNs quickly kill connections when faced with link outages or intermittent stability. This measure is designed to prevent man-in-the-middle attacks and other forms of eavesdropping or connection hijacking with the trade-off of artificially reducing availability. On a system such as OpenVPN, timeout parameters can be adjusted to prevent a connection from being killed, but traffic sent during this period will simply fail to arrive, just as it would without a VPN; as the

timeout interval becomes longer, more effort and care must be put into security.

In many cases this trade-off is merely an annoyance, but for organizations relying on sensitive real time information (e.g. international financial transactions, military operations, medical evacuations, etc.) every second counts. A three minute outage could literally be a matter of life and death in the latter two cases.

The goal of this project is to incorporate RON and VPN software in a realistic test environment and determine the exact impact of their combined use on VPN availability and performance.

2 Emulab: Testing network implementation

In order to explore and measure the comparative merits that are to be enjoyed by the employment of a RON, any experiments must be carried out in a network that resembles a realistic environment, particularly with respect to the routing processes that take place within the network. To that end, the Emulab environment [9] was chosen as the setting in which experimental network topologies would be developed, in lieu of access to an actual live network.

Two main design goals were set regarding the emulated network topology: It should (a) be easily scalable, extensible and customisable in order to facilitate experimentation for a broad range of applications, not limited to our specific case; and (b) rely on BGP to enable inter-AS routing. The latter goal can be thought as being relevant to the former one, since the primary functions of BGP is to facilitate scalability, but its main purpose is to ensure that the network behaves in a realistic way, slow convergence issues due to link outings, security issues, etc. included.

Consequently, one is led to install and configure software routers on the nodes of the emulated topology, since the preset routing schemes that can be defined through the Emulab `ns` syntax cannot support the flexibility of the desired behaviour. This, in conjunction with the aforementioned goal item a, defines another design goal: The network configuration must be done *automatically*—otherwise, the emulated network topology would scarcely serve as a convenient experimentation tool, particularly as the size of the network increases.

To meet these three requirements, we developed a *multi-AS network generator*. The generator is provided as a bundle of `.ns`, `AWK` and `C-shell` scripts that perform the generation, configuration and initialisation of the desired emulated network. All relevant code can be found at `https://github.com/ailiop/cps214/tree/master/code/multi-as`.

2.1 The network generator

The functionality of our network generator can be considered, roughly, to address two issues: (a) the network topology and (b) routing within the network.

Topology Using the network topology generator, we are able to easily instantiate a class of multi-AS network topologies. These topologies adhere to the following specification: Each AS is comprised by a full mesh of routers and a number of LANs; each LAN, in turn, is comprised by

a number of terminal (e.g. PC) nodes which share a single router as gateway. The inter-AS links must be explicitly set by the user, the reasoning being that one would not expect too many inter-AS connections to be defined in an experimental emulated network. The creation of an inter-AS link can be flagged in a `.txt` file, simply by specifying the two routers that will be linked by way of the ASNs and serial numbers within their respective ASes. It should also be noted that, although the intra-AS links are set to form a full mesh, a user may easily obtain custom topologies by explicitly removing any unwanted links, either by modifying the `.ns` script (e.g. by including a source file that deletes a number of links) or by dynamically killing the links after the network is instantiated.[1]

The topology can be scaled through a few parameters in the `.ns` script that control the number and size of ASes in the network, as well as the number and size of LANs within each AS. Additional parameters control the configuration (e.g. delay and bandwidth) of the inter-AS, intra-AS and LAN links, as well as a number of other elements of the network configuration.

Technically, the size of the network is limited to 8 routers and as many LANs per AS, 253 terminals per LAN, 256 ASes, and 16384 inter-AS connections, while it is easy to modify the network creation script to balance between the number of allowed ASes and that of routers (and LANs) per AS. However, much tighter limits will likely be imposed on the actual experiment by the testbed environment.

Routing After the network topology has been created and instantiated in Emulab, routing must be set up. To that end, we make use of the *Quagga* software routing suite,[2] and enable BGP routing over OSPF.

The routing configuration process is commenced as soon as a node is booted in the Emulab experiment. The first step is to link the kernel's interfaces (e.g. `eth3`) to the corresponding IP addresses (e.g. `192.168.0.3/30`); this cannot be done in advance, since the assignment of IP addresses to interfaces is random in the Emulab environment.[3]

Then, configuration files for the relevant Quagga daemons[4] are generated. The OSPF process is configured to assign a router's links to areas (currently all links are assigned to area 0) and to retrieve *connected* and BGP routes, giving higher precedence to the connected ones. The BGP process is configured to retrieve the OSPF routes and to set up BGP peering between the node and its immediate neighbours. The next step is, of course, to invoke the routing daemons and wait for convergence.

As far as the terminal nodes (i.e. the ones that form the various LANs, sans the respective gateways) are concerned, these need not be running routing software, as they will only act as sinks from the perspective of the OSPF/BPG routing processes. Hence, the set of IP prefixes that

[1] It should be noted that the latter approach should not be preferred as it would result in allocation of redundant nodes in the Emulab testbed.

[2] http://www.nongnu.org/quagga/

[3] The Emulab *control link* is an exception in that it is always attached to `eth0`. Furthermore, Emulab does provide a way to statically specify the interfaces of links, but this functionality is problematic and it greatly limits the nodes that can be used.

[4] Three daemons are needed: `zebra`, `ospfd` and `bgpd`.

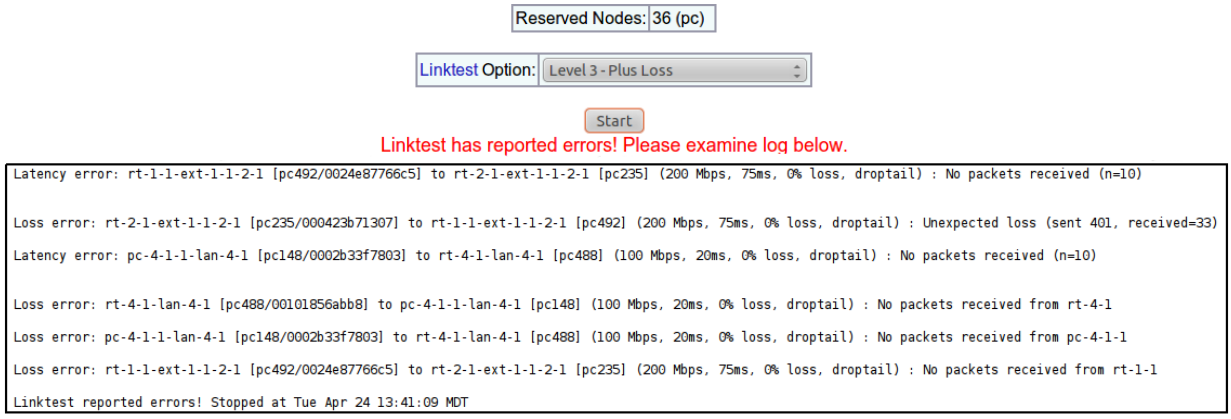

Figure 2.1: Emulab Linktest log for an experiment of 36 nodes. The link `ext-1-1-2-1` seems to exhibit a very high packet loss rate, whereas the link `lan-4-1` appears to be dead.

are associated with the emulated network are simply added to its default routing table with the corresponding LAN gateway router as the next hop.

It should be noted that, while router nodes must be booted with a Linux OS image that contains the Quagga package,[5] the terminal nodes may be booted with any Linux or Free-BSD image. This will likely allow for running most application-layer experiments with minimal configuration. A possible scenario would be for one to develop an experiment script that is supposed to run on the terminal nodes and simply have it be invoked from within `startup-pc.csh`.

Lastly, since experimentation is an iterative process and manually resetting the network to its initial state (perhaps after reconfiguring the routing, too) would be even more tedious than re-initiating the Emulab experiment, a script is provided that crawls among all nodes in the network and completely resets the node initialisation process.

2.2 The dark side of Emulab

Emulab has proved to provide a great environment for network emulation in a realistic way. Unfortunately, "realistic" also implies that a host of things might go wrong. Here we mention an error that might occur while swapping a large experiment in Emulab which would cause the network to exhibit aberrant routing behaviour, as its source might be hard to track down.

Figure 2.1 shows the Emulab Linktest log for an experiment that uses 36 nodes. It can be seen that a link appears to exhibit a very high packet loss rate and another one seems to be completely dead. However, that is not the case: Links that appear to be dead after running Linktest may still allow a packet to go through from time to time (e.g. once every one or two minutes); depending on the placement of such links in a topology, this could cause the BGP routes to be constantly changing.

[5] A ready option in the Emulab of the University of Utah is the custom image we created: `UBUNTU11-64-QUAGGA`.

2.3 An example network

Figure 2.2 shows a visual representation of a network topology with eight ASes, generated with our network generator script. It should be evident that the automatic routing configuration procedure is invaluable to facilitating experimentation with such large topologies. The figure was obtained using the Emulab visualisation tool.

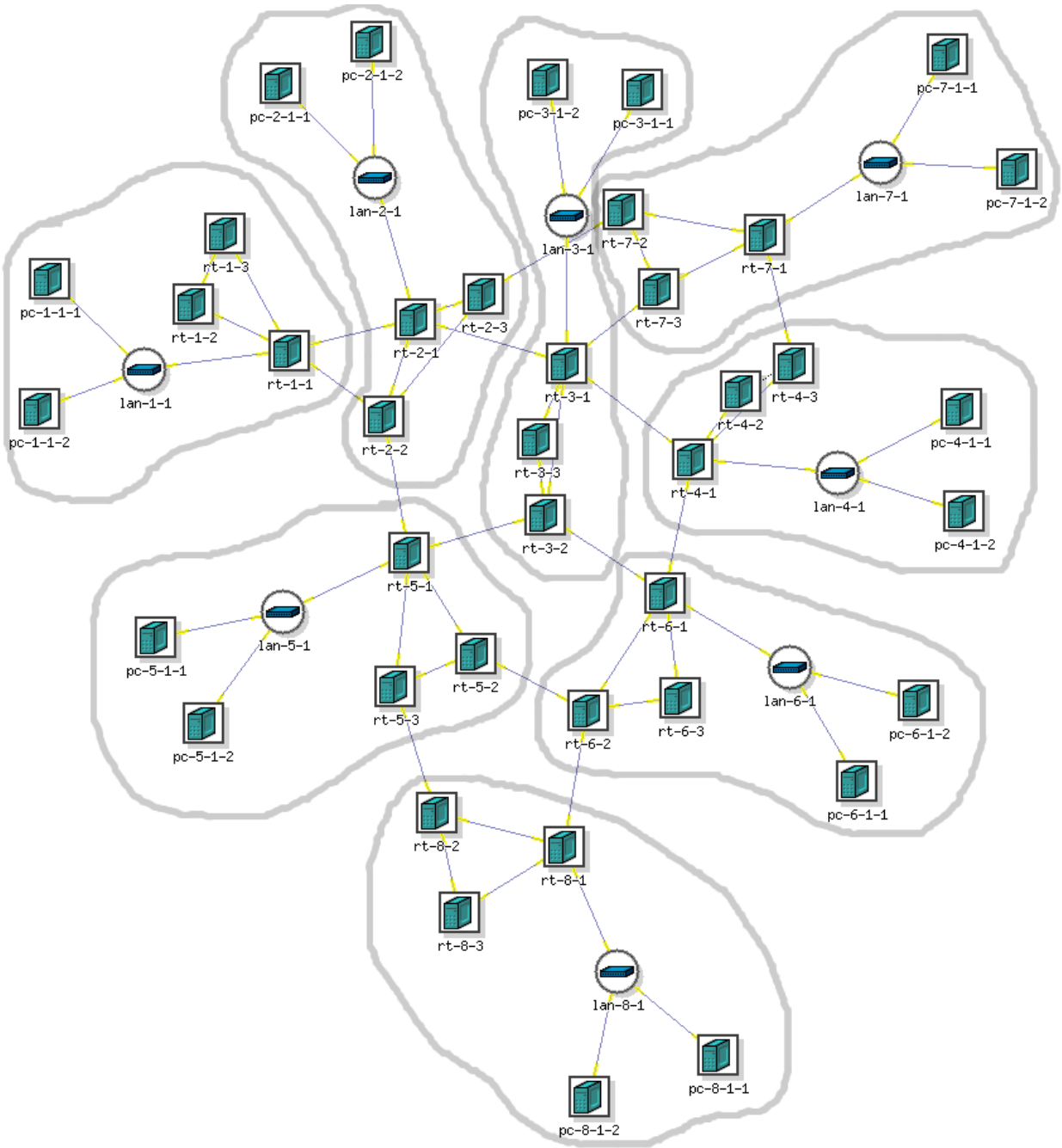

Figure 2.2: A network topology of 8 ASes, with 3 router nodes and 2 terminal nodes each. The gray borders delimit individual ASes.

3 Resilient Overlay Network (RON)

A Resilient Overlay Network is a communications architecture designed to improve the performance of distributed applications in the presence of Wide Area Network path degradation or outages. Most WANs are large and complex domains, the Internet being foremost among them. In order for routing to scale in such environments, trade-offs in precision must occur. Maintaining a link state database on a large WAN is nearly impossible, as simply passing the messages back and forth would quickly cripple most networks and WAN latencies would negatively impact the accuracy of any such database even if someone managed to sustain one. Border Gateway Protocol is the current standard for Internet routing and it divides the Internet into Autonomous Systems that heavily filter communications with each other, typically providing only summary link and network status updates instead of detailed information about internal routing. This filtering is a key aspect of the scalability of BGP, but along with other features of BGP it results in routing outages that can range from 30 to 180 seconds after a link failure [2]. Typically, the length of an AS path plays a large part in dictating the point in this range where a particular outage will fall; intuitively, a longer AS path will usually take more time to re-converge. This can be attributed to the potential for more conflicting updates in a highly meshed system like the Internet.

A RON is able to improve performance in these situations by reducing the scale of the network to include its member nodes only. This significant scale reduction allows the RON to recover the ability to maintain a full link state database when the underlying network protocols cannot. A RON is able to improve performance over BGP in these situations due to physical path redundancy and active probing/polling. Most Autonomous Systems are connected to several other systems and it is also rare for any system to only have a single link to any of its neighbors. BGP is slower to recognize such alternative paths because they are typically filtered out of communication between ASes [2].

The primary limitation of any RON is its size constraints. The original RON system was empirically tested to scale to approximately 50 nodes and a revision in the link state update system called Scalable RON improved this upper limit to approximately 300 nodes [8]. The scaling problem is the same fundamental issue faced by regular routing protocols such as BGP and no solutions for further scaling have been proposed to date. Interestingly, any significant improvement in scaling would likely have far-reaching implications for graph theory in general, not just networking. Fortunately, the scaling issue has little impact on our project, since 50 to 300 nodes is a more than adequate range for most VPNs and many other distributed systems.

3.1 Creating an Emulab-compatible RON image

Initially, we thought that creating a functional RON image for use with Emulab would be a simple and quick portion of our experiment. Unfortunately, our assumptions were incorrect and this phase of the project actually proved to be rather challenging. The original RON paper was published in October 2001 and the source code was maintained through 2005 by the authors, but after

that fell into disuse as research efforts at MIT and CMU shifted to other projects. In 2009, the RON research group published an additional paper titled Scalable Resilient Overlay Networks that improved functionality by allowing efficient link-state update in networks up to 300 nodes. The scalable RON code is also available, but is a completely separate implementation in Java and only seems to run as a simulation.

Since no fully functional source code was available, we elected to work on a basic port of the original RON code to move it from its original OS environment (Free-BSD 4.3) to a modern system (Free-BSD 9). We briefly considered utilizing Ubuntu Linux instead of Free-BSD, but RON relies on two kernel modules unique to Free-BSD: IP Firewall (`ipfw`) and Divert Sockets (`ipdivert`). Used in conjunction, these two modules allow raw IP packets to be captured at the interface level and diverted to any program on a specified listening port, which allowed RON to capture traffic that originated from other systems or from any network application. Similar functionality was briefly supported in Linux through IP Chains, but no support exists beyond Linux kernel 2.12 [3]. Since these modules were instrumental to RON, we decided to abandon the Linux port and continue using Free-BSD.

Even under Free-BSD, building an image proved challenging. We were briefly excited by the presence of an old RON image in the Emulab archives, but unfortunately the image was built on Free-BSD 4.3 and was too old to work on any machines currently available in the Emulab cluster. Additional challenges came from limited documentation and reliance on several old libraries including Berkeley DB3 and an MIT in-house library called `net-util`. Both libraries are no longer actively maintained, though we were able to find and compile old source code. These libraries and the RON source code included many deprecated/obsolete programming abstractions that we updated to work with the latest version of `gcc`. After a few weeks of work we had a basic Free-BSD image that would compile and complete all included unit tests successfully.

We briefly celebrated, and then quickly realized our image problems were not over! The additional issues we encountered were unique to the environment of Emulab. Every time we locally compiled Free-BSD's kernel with the IP Firewall and Divert Sockets modules there was no problem, but any attempt to uses these modules on an Emulab system would cause it to fail and become unresponsive immediately after booting. It wasn't even possible to log into the systems through Emulab's back door console system. After extensive troubleshooting, we learned that when IP Firewall is enabled it immediately blocks all network traffic by default! This feature was therefore blocking access from the Emulab control network and thereby preventing the successful application of all Emulab start-up scripts, which caused the control system to register a boot failure every time. The solution to this problem was to create a startup script that dynamically added both modules after Emulab's boot process and then immediately modified the Firewall rules to allow all traffic by default.

After this final phase of troubleshooting, we finally had a "fully functional" ported RON image that passed some basic connectivity testing, but failed to improve resiliency in our test network, as shown in the performance chart in section 4. We spent several weeks attempting to debug the errors,

but only found that the RON routing tables were failing to properly update in the presence of a routing failure. Even when all links were cut, a node would continue to claim it had direct routes to all other nodes in the RON. Our current intuition is that some of the code modifications made for the port might be generating some form of run-time error, but a lack of thorough documentation and code comments has made it very difficult to attempt meaningful troubleshooting, even with some help from the program authors.

Due to these difficulties with the RON image we decided to slightly alter our project and began development of a new simple RON system written entirely in Ruby.

3.2 Ruby-RON

We chose Ruby as our base language due to its intuitive high-level abstractions, extensive built-in libraries, and platform independence. The goal of our simple RON implementation is to provide applications with a simple messaging interface that will allow them to choose between their regular communications channels or a secure encrypted channel from the RON that supports one-hop indirection for resiliency. Our current implementation can be broken into four components: (a) Initialization/Membership, (b) Link-State Server, (c) Link-State Client, and (d) a Messaging Daemon

Currently, Ruby-RON is initialized from a local text file containing the IP addresses of all member nodes. This results in a static membership set, which would be maintained by an administrator. Some applications might find use for dynamic membership, but we removed this feature for simplicity. Additionally, static control is a desirable feature for a VPN system which is our current target application.

The Link-State Server is a simple TCP agent running in an independent thread that operates on each member node. When queried by any client process, the server provides an update message containing the link state of all its local connections. Currently, the Server reads its link state data directly from a Fibonacci-heap priority queue running in the local client process. This data structure is very efficient for routing, but look-ups for a specific key other than the current minimum are slow. One modification we are considering is using a separate array or hash to maintain a copy of local link-state and allow for more efficient look-ups on all keys without impacting routing performance.

The Link-State Client iteratively queries all other member nodes for their current link state tables and also performs probing to determine link state for the local node. This link state information is then fed dynamically into a hash of Fibonacci-heap priority queues. The hash contains one queue for each destination node. The end result is a simplified version of Dijkstra's algorithm that outputs the shortest current one- or two-hop paths. The Fibonacci heap is the most efficient data structure for Dijkstra's algorithm and minimizes computational overhead for our program. Limiting link-state data to one- and two-hop paths minimizes network traffic overhead and was shown by the original RON research to provide good improvements in fault tolerance and performance for most networks [2].

The messaging module of Ruby-RON is still in development and will utilize Ruby's built-in OpenSSL library to provide message encryption and traffic tunnels. Message forwarding will rely

on the routing tables maintained by the link-state client.

The code of our Ruby-RON implementation can be found at github.com/ailiop/cps214/tree/master/code/ruby-ron.

4 RON and Ruby-RON performance

The functionality and performance of the RON image and our simple Ruby-RON implementation where tested in the network shown in figure 4.1. To that end, we tried killing a link in the network, after it had reached its steady state, and monitored traffic between the terminal nodes to measure the perceived network outage time. The experiment was also repeated for the case where no RON was utilised.

As can be seen in figure 4.2, the simple lightweight Ruby application was able to provide a significant reduction in perceived outages and maximized resiliency in a large network with six RON nodes. Figure 4.2 also shows that our network topology was of sufficient size to generate significant BGP convergence delays during link failures. Finally, our testing revealed that the current port of the original RON code seems to provide no improvement in performance or fault tolerance.

As a disclaimer, we cannot sequester enough nodes in Emulab to fully support a scalability test for Ruby-RON; plus, some of the computational inefficiencies discussed in section 3 would likely cause a performance decrease and RON was reported to scale well for up to 50 nodes. Additionally, we still have doubts about the functionality of our current RON port and do not feel this performance reflects its actual capabilities, especially since it doesn't match a large amount of published test data.

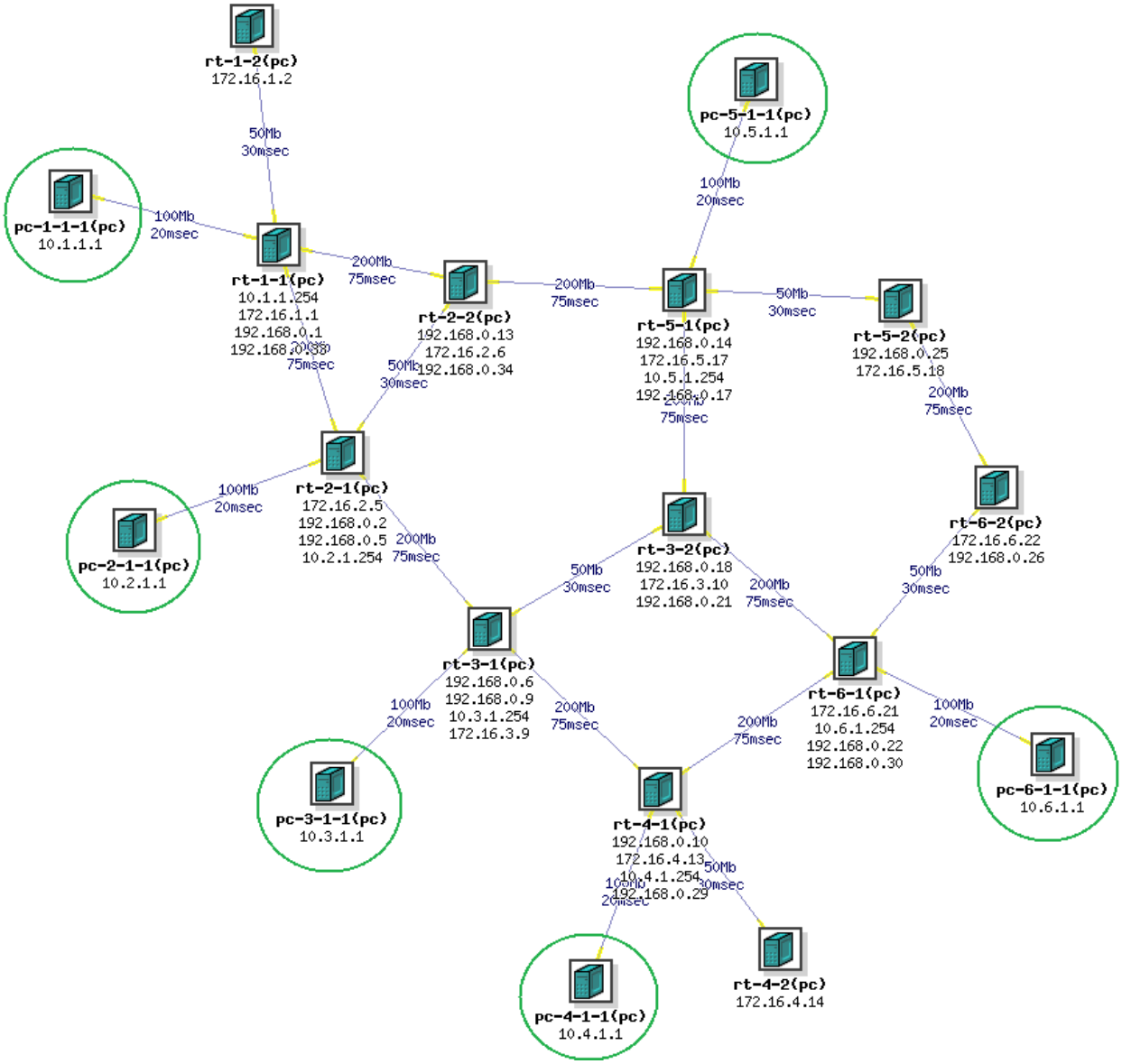

Figure 4.1: The testing network. The terminal nodes (i.e. the ones that form the overlay network) are circled.

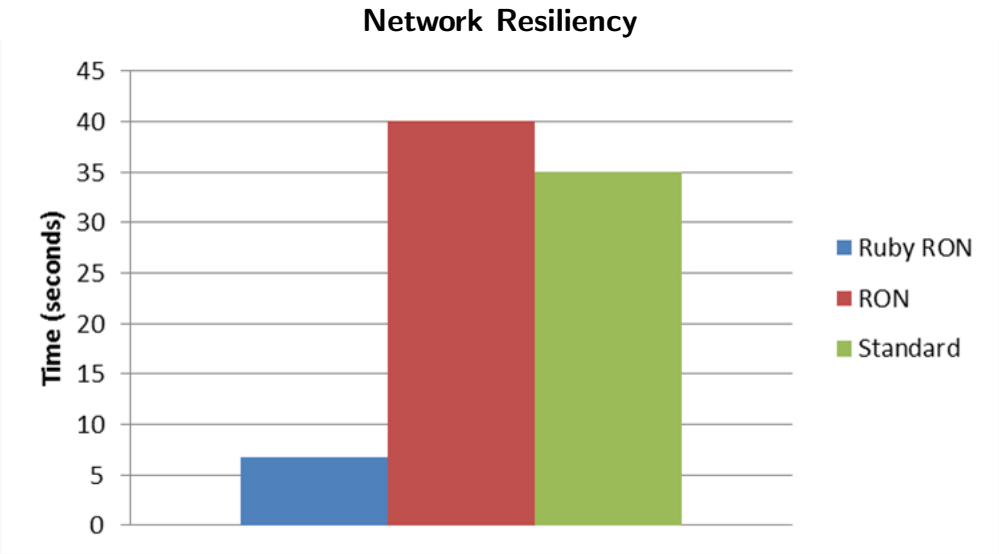

Figure 4.2: Average routing outage intervals after killing a link in the network in figure 4.1. "Standard" means that no overlay network scheme was employed.

5 Future Work and Conclusions

The results of our Ruby-RON experiment concur with those of both the RON and Scalable RON projects and show clear benefits for time-sensitive distributed applications using a functional RON. Interesting directions for future research include considering security implications of using a RON in a true peer-to-peer environment and exploring methods of routing updates that might avoid the lower bounds involving direct comparisons of paths. Additionally, the efficient algorithms used in SRON could be incorporated into Ruby-RON to provide a highly scalable, platform-independent architecture.

Furthermore, we feel that the network generator that came out of our work on this project is an important tool in its own right, as it can greatly simplify the process of emulating a realistic network environment. We hope that even in cases where the desired network does not perfectly correspond to the functionality that is provided by our implementation (e.g. one might need to run IS-IS within the ASes, instead of OSPF), it could still be easily extensible or modifiable to allow for the extra functionality. Future work on this part would include consolidating certain parts of the associated code,[6] and perhaps providing greater flexibility in its use.

References

[1] Doug Aamoth. PlayStation network attacks may cost Sony $170+ million. *Time Magazine*, May 2011. URL http://techland.time.com/2011/05/23/playstation-network-attacks-may-cost-sony-170-million/.

[6] This is planned to be done soon, in the hope that it might actually be useful to the community.

[2] David G. Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, SOSP '01, pages 131–145, October 2001. URL http://doi.acm.org/10.1145/502034.502048.

[3] Illia Baldine. Divert sockets mini-HOWTO, February 2000. URL http://www.faqs.org/docs/Linux-mini/Divert-Sockets-mini-HOWTO.html.

[4] William J. Broad, John Markoff, and David E. Sanger. Israeli test on worm called crucial in Iran nuclear delay. *The New York Times*, January 2011. URL http://www.nytimes.com/2011/01/16/world/middleeast/16stuxnet.html?pagewanted=all.

[5] Christopher Brook. Anonymous Finland wages war on mining, leaks 500,000 e-mails. *Kaspersky labs security news service*, November 2011. URL http://threatpost.com/en_us/blogs/anonymous-finland-wages-war-mining-leaks-500000-e-mails-111411.

[6] Ponemon Institute. Second annual cost of cyber crime study: Benchmark study of U.S. companies. Technical report, August 2011. URL http://www.arcsight.com/collateral/whitepapers/2011_Cost_of_Cyber_Crime_Study_August.pdf. Sponsored by ArcSight, an HP company.

[7] Jennifer Saranow Schultz. The rising cost of identity theft for consumers. *The New York Times*, February 2011. URL http://bucks.blogs.nytimes.com/2011/02/09/the-rising-cost-of-identity-theft-for-consumers/.

[8] David Sontag, Yang Zhang, Amar Phanishayee, David G. Andersen, and David Karger. Scaling All-Pairs overlay routing. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, pages 145–156, Rome, Italy, 2009. ACM Press. URL http://portal.acm.org/citation.cfm?doid=1658939.1658956.

[9] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. *ACM SIGOPS Operating Systems Review*, 36(SI):255–270, December 2002. URL http://portal.acm.org/citation.cfm?doid=844128.844152.