Literature Review: Deep Routing Simulation

Submitted in partial fulfilment
of the requirements of the degree of

Bachelor of Science (Honours)

of Rhodes University

Principal Investigator: Mr. A. Herbert

Grahamstown, South Africa
May 28, 2012

Introduction

Deep Routing Simulation is essentially the means of simulating routing traffic on a large scale network. Although many approaches have been taken to network simulation as a whole, this research takes a more direct approach by setting its scope mainly to packet routing simulation, primarily within the IPv4 scope, and related articles in this field of study.

The structures of a typical IPv4 header is shown below as defined by RFC 791 [22].

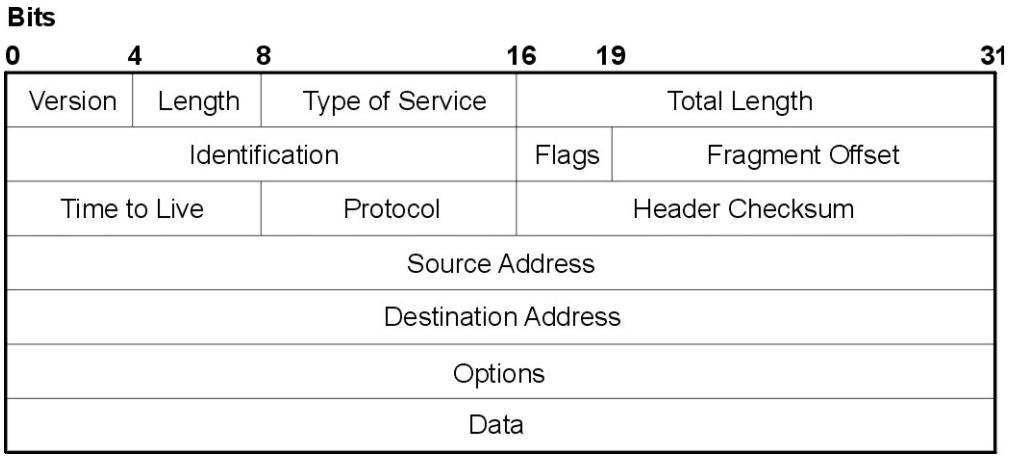

Figure 1: IPv4 Packet Structure

Aproaches to Network Simulation

Packet routing is simply the process of moving packets from a source host to a destination host through a network [16]. A route may simply be a direct connection to the destination host or it may involve a series of hops through routers, hubs and even load balancing systems.

Routing protocols are classified into three major classes which are interior gateway routing through link state routing protocols, interior gateway routing through path vector or distance vector protocols and exterior gateway routing. The purpose of a routing protocol is to prevent loops from forming when routing occurs within a network and if such loops occur, they should be able to break them up. They also deal with selecting the best routes around a network based on a predefined cost on each link which helps reduce latencies and traffic within a network [2].

As more hosts are added into the network so routing becomes more complex and the chance of a single node being connected to only one host becomes negligible. This makes error testing increasingly difficult as other systems that are not directly part of the system one may be testing can get affected by error and robustness testing on a point of the network. This can lead to loss of throughput, higher latency and even complete denial of service [17].

Network Simulation Software was developed to handle the needs of systems in development that required live network testing but was not yet ready to be opened to a network due to extraneous behavior that may affect the system or visa versa. Network Simulators could allow for creation of network traffic and complex routes without ever connecting to a real network. Some hardware implementations of network simulators are Netropy and Linktropy by Apposite Technologies [30] and products such as Hurricane, 8400E and 4XG Network Simulator by Packet Storm [8].

Network Simulation is also required at education and research levels as it is more secure and easier to control a network setup by an invigilator or institute than it is to use a real live network. This also reduces costs in which an institute no longer needs fund the required amount of hardware and space needed to create a network in which a class can be educated on. NS-3 was designed as a software implementation for this purpose and allows for scalability of the networks simulated size and also allows connection to real networks that exists [19].

Another specific that needs to be considered when designing Network Simulation Software is the protocols it has to support. Wireless has become increasingly popular in recent times and this area of research is still developing [1]. This has lead to development of many different wireless protocols, the IEEE 802.11 and 802.16 families of specifications, and as such requires different handles in simulation of these protocols.

Wired protocols are easier as most widely used protocols have become set in their implementation however, development over the years towards the medium in which packets are sent over have seen vast improvement. This has lead to benefits such as more reliable connections, less or no electromagnetic interference and higher bandwidth [21].

Complications in Design and Implementation

Bandwidth, latency and packet loss also come into play when designing and implementing a network simulator. A Network Simulator should be able to handle a wide spectrum of

connections to itself of which all may have separate distances and run of different mediums. This leads to different levels of packet loss, latencies and bandwidths. This is particularly noticeable when comparing a wired to wireless connection. Wireless connections typically have higher latencies and higher rates of packet loss than traditional wired connections [34] depending on distance and what mediums are between each wireless links. This means that the average statistical throughput in one environment can be completely different to another and so adds to the complexity in design and implementation of wireless protocol simulation.

Bandwidth speeds range from as low as 300 bps (bytes per second), as implemented by the ITU V.21 protocol, up to 159.25248 Gbps (Gigbits per second), as implemented by the OC-3072/STS-3072/STM-1024 protocol and currently implemented in todays networks. This speed is near imposible to simulate on publically available hardware and as such this makes simulation difficult. This is because not only does the system have to account for different synchronization speeds but also the fact that the faster the simulated connection the more hardware intensive it becomes. So in basics, implementing such handles on a network simulator increases the cost and complexity of the system as a whole while causing a greater level of abstraction which can lead to loss of base level features or complications in use of them.

There are approaches used to compensate for heavy load on Network Simulators. The first being the use of a cache, this is usually a server that stores recently or commonly accessed files and pages within a network in an attempt to speed up networks by handling page request directly instead of requesting the page/file again from the internet, host or network. This gets used to store pages and meta data on each node as to increase throughput by decreasing data requests. This has been seen to work and has been tested at network rates up to 100 Mbps (Megabits per second) but utilizing the same size caches on higher bandwidth tend to see the implementation of caches fail [20]. If one were to increase the cache servers to deal with higher bandwidth rates, this would lead to direct impact on the memory within the simulator and would result in thrashing which would lead to an overall decrease in performance.

Implementing cache policies in this case would increase overall performance as they are designed to increase performance by managing memory in a simple and logical manner. Traditional policies such as Least Recently Used (LRU), Round Robin (RR) and Least Frequently Used (LFU) lead to better memory management and prevent thrashing within the system [5], reducing memory requirements and allowing offloading to disk of lesser accessed entities.

The next approach is to improve the hardware running the simulator. A simple improvement would be to increase memory on the server, this would allow for greater sized caches at each node and larger more detailed routing tables to be stored. There has also been a proposed design by Wolf and Turner [33] for hardware implementation that runs multiple network processors, each with own cache and memory, which they suggest will overcome the hardware level bottlenecks created by conventional methods of Network Routing. This approach may be adapted into the hardware of a Network Simulator and should not be disregarded as useless in ones design of the overall system.

One may consider load sharing over multiple processors within the system as well, this will omit the need for task specific hardware and one can look into using low cost publically available hardware to solve the problem. Even though the quantity of hardware and space requirements of the simulator will most likely increase, this is unfortunately a trade off one may have to make.

However, one may only want to simulate packets being sent and received at end points of a system. This omits the need for simulating node hops within the network and as such reduces the actual load on the simulator. This is commonly implemented by receiving a packet from one end point and transmitting it to another end point without any intermediate processing of hops in the simulated network. Another addition made to this is generation of packets from pseudo hosts within the network. By having a single host or node generate traffic as if it were an entire network and hence create the activity of larger networks with lesser hardware requirements. This reduces the need for actual hosts and allows for testing of a specific host system that are designed to run on large networks [3].

Simulating delay on a network is something that is often over looked in design and implementation. The reason for this is that most of the time the actual results are not affected by the implementation of delay and so it is rather left out to cut down on time taken for development and time added during runtime of the tests. There are however uses for such features such as testing systems that require a certain levels of throughput and need to be able to handle delays. That being said, delay is very difficult to simulate accurately as it can be generated not only on time taken to travel through the medium used but also when processing, internally buffering and putting the data down onto the wire.

Another point that is raised by The VINT Project group is that protocols require extensive testing to ensure that they run under all conditions [6], this includes delay. This is not a problem with small networks as time taken for processing, buffering and transferring

the data over a medium is negligible. The problem is as the network becomes bigger and more wide spread that data takes longer to transfer between hosts and nodes. If a TCP (Transmission Control Protocol) like protocol were to implement resending of data that is not acknowledge within a time frame [15], this would work well on smaller networks but as a networks delay increases, there is no way to tell if the data is still being received or if it lost. Not implementing this correctly can lead to excess traffic in the network as data is resent when it not need be resent.

Like delay, packet loss is also a feature which is often omitted from network simulation software. The reasons are similar to that of delay as it adds time to the research project both at development of the software and runtime of it. This to is very purpose specific and mainly used in testing of systems and protocols that are sensitive to packet loss such as file servers or more specifically the FTP (File Transfer Protocol) and similar protocols.

Packet loss comes into play with systems that stream data, these systems must be able to recover from lost data without delaying the rest of the system as a whole. Protocols too must be able to deal with packet loss and as such must also have mechanisms built in to recover from such mishaps [6]. Examples of this are implementations of audio streaming and video streaming on frameworks such as HTML5 and Adobe Flash as they can actively stream media across the Internet and have mechanisms built in for packet loss detection and recovery.

BGPv4 (Border Gateway Protocol version 4) defined by RFC 4271 [25] is a protocol designed specifically to handle exchange of routing information between gateway hosts. Each gateway host stores a routing table that contains all known nodes to that gateway host. Along with each router is stored the address of that router, a list of all nodes that it is connected to and a cost that is used in determining the best path for a connection to made on [24]. This protocol is used mainly in inter-country and inter-continental routing and builds up the routing backbone for the Internet.

However, simulating a BGP is similar to simulating a routing node in a conventional network. This is made easier with the implementation of CIDR (Classless Inter-Domain Routing) defeined by RFC 1518 [23]. This makes use of a most significant portion and least significant portion in standard 32-bit IPv4 (Internet Protocol version 4) address. The most significant part defines a subnet, a portion of the internet to which a set of hosts belongs, and the least significant part is used to define the individual hosts. The major difference is the size of the routing table which is much larger. This however may lead to an decrease in memory usage as if one were to store all known connections for

all known nodes connected in the network to the current gateway host in the gateway hosts internal table, one could use this to direct a route for a packet rather than storing routing information by use of a separate table in each node within the network. In this way redundant data can be removed from the network. An example of redundant data would be simply shown by considering two connected nodes in a network. They each would store the connection to one another in their own routing tables, this means that there are two entries stored in total for one connection. If one were to store this data in a gateway host it would only keep one entry for the connection between the nodes and so reducing memory costs.

There are implementations of BGP available for use. The company bgpVista has produced a lightweight even driven BGP simulator named simBGP. This simulator allows for configuring of link delays, packet loss, processing delays, protocols to be run on the network as well as bandwidths. This along with being able to set up the network as one would require for testing purposes makes it a very well implemented simulator [4].

There are also simulators which require the ability to accept packets not addressed to it. Libpcap is a library which utilizes the promiscuous mode packet capture. Promiscuos mode packet capturing allows for packets addressed and not addressed to the host machine which it is running on to be captured, read and manipulated down to a byte level [14]. The process of capturing a packet of the wire that was not addressed to the host computer is known as "sniffing". This gives an easy to use interface onto any captured packets and as such one can easily use this library to discern between routing packets or not and as such simulate any response that one would want to.

The act of sniffing a packet however does have its problems. It becomes nearly impossible to sniff packets on a switched connection as the switch/router directs the relevant traffic to the destined host only and does not send it publically onto the network like conventional hubs. This means that a promiscuous mode enabled network adapter has nothing to sniff on the wire except its own packets that are directed at it. However data sent out can still be targeted at the host as the switch will still send the data to the destined address.

Libnet is particularly useful in doing just that. This library offers easy to use functions that allow for building a whole customized packet. This includes, source and destination IP, source and destination port, payload, checksum and even changing the bits that go in the zeros part of a TCP packet [31]. This can be useful if one wants to spoof route tracing packets and make a node or host believe that it is accessing a real time network.

The use of Libnet again is a good choice as it was designed as an attempt to standardize

the way in which protocols on a network are structured. With this in mind Libnet was created to be lightweight, simple and usable over many platforms [31]. This implies that its memory and CPU footprints are minimized and gives great functionality for little resource cost.

Using a combination of Libnet and Libpcap would make for an easy implementation of a network simulator with the ability to pick up packets off the wire and redirect them to where they need to go within the simulated network. This would be achieved by using Libpcap to retrieve a packet from the wire, break it down into its relevant parts that need to be analyzed. Then a decision can be made as whether the simulator should act on the packet received or not. If the simulator does act on the packet, in terms of generating more network traffic, it can use Libnet to create a packet that is relevant to the context of the packet received and generate a response to the host machine that the packet originated from if need be. The only drawback to this is due to the network adapter running in promiscuous mode, the more hosts connected to the network simulator, the more CPU intensive the process of sniffing and processing packets becomes.

Creation of Networks and Routing Data

A simple tool to trace routes around a network on a most platform is traceroute, the implementation for a Windows platformed host is tracert. This program is used by passing the it a series of options and then an IP or host name as an arguments. These options range from setting the hop limit between nodes to defining what protocol to use when preforming the traceroute such as an IP alternate to a standard UDP (User Datagram Protocol) packet approach such as defined in RFC 1393 [18]. Passing the command "traceroute -I -m 5 www.google.co.za" to a terminal would execute a route trace to "www.google.co.za" using traceroute where "-I" is an argument passed to make traceroute use ICMP echo packets and "-m 5" limits the maximum hop count to 5, if left out the default sets to 30.

These programs result in showing every hop in the route taken to along with delays and any IP address changes along the way. This becomes particularly useful when looking into recreating live networks, one can simply trace the routes to every known host machine on a network and then recreate the network using the routes returned. The Caida is an association that can aid in doing just that. They collect data about the Internet from average delays and packet loss to tracing routes, further more they make this data available for research on request [7]. This can be particularly useful in recreating a large

```
              :~$ sudo traceroute -I -m 5 www.google.co.za
traceroute to www.google.co.za (74.125.233.24), 5 hops max, 60 byte packets
 1  ict.gw.ru.ac.za (146.231.120.1)  1.164 ms  1.220 ms  1.222 ms
 2  ammcore-maincampus-0.net.ru.ac.za (146.231.2.29)  0.924 ms  0.956 ms  1.007
ms
 3  datacentres-1-ammcore.net.ru.ac.za (146.231.2.18)  1.120 ms  1.128 ms  1.128
 ms
 4  border-struben.net.ru.ac.za (146.231.0.2)  0.235 ms  0.245 ms  0.245 ms
 5  tenet.net.ru.ac.za (192.42.99.1)  8.515 ms  8.525 ms  8.526 ms
```

Figure 2: Use of traceroute with Options

portion of a network based on the Internet as a model which can lead into DoS (Denial of Service) attack testing and worst case scenario recovery tests in disaster simulations.

When creating the nodes of a simulated network with in the network simulator a general approach is to use an object orientated design to create nodes within the network, each with their own routing table of which nodes they are connected to [32]. As stated in Section 2 this becomes very memory taxing but as it turns out also takes up a lot of CPU time [32]. A solution for this is creating a over lying super routing table which contains the meta data and state and of every node in the simulated network. If this design is implemented with a database, such as SQL [9], it can also lead to decreases in both memory footprint, as unused nodes will be left on hard disk, and CPU usage as there is no longer a need to service all nodes in memory at once. The overheads of reading a node to memory though will have to be taken into consideration when calculating delays as access times on a hard disk are slower than access times to memory.

The IP (Internet Protocol) also needs to be factored in when creating a network as it is now becoming more and more common for a network to run a hybrid of IPv4 (Internet Protocol version 4) and IPv6 (Internet Protocol version 4). The IP handles routing around networks and more specifically the Internet. Every device that is connected to the Internet needs to have its own unique IP. This is because an IP address is used to locate the device within the network. If two devices were to have the same IP address on a network, this would cause a conflict when routing the packets to the correct host as the host which the packet is actually destined for could be iether one.

The main difference between IPv4 and IPv6 is the address space. IPv4 allowed 32 bits for an IP address, this allowed for 4294967296 (or 2^32) unique addresses [22]. At the time this was thought to provide more addresses than would ever be required until February

2011 when IPv4 address exhaustion occurred as a result of the IANA (International Assigned Numbers Authority) allocating the last available IPv4 blocks between the five RIRs (Regional Internet Registry) [27]. IPv6 was created int the late 90's to solve this problem by introducing a larger address space by allowing 128 bits for an unique address[10]. This is equivalent to 2^128 unique address which is 2^96 times larger than that of IPv4's address space.

As such, network simulators have implemented not only support for IPv6 but also for hybridized networks which consist of both types of addressing protocols. The National Science Foundation who implemented NS-3 manages this by defining each node as object, each node then extends from this object forming up its protocols that it supports, what IP family it belongs to and any other needed information [11, 12]. This allows for an object orientated design and communication between nodes on an object level until any sort of network simulation is required.

Detection of routing packets entering a simulated network is another consideration that is taken in design and implementation of a network simulator. If one relies on actual external hosts to feed information into a the simulated network, the actual system that simulates the network needs to be able to discern between which packets are directed to which simulated nodes or hosts within the system. This becomes more complex as the system on which the simulator runs does not accept any packets that are not addressed to it as each address is unique [29].

One can set a machine to accept packets not addressed to itself by setting the systems network device into promiscuous mode, this is not available on all network adapters though. The implication of this is a higher load on the CPU and memory of the simulating system as it has to process every packet visible on the wire but allows for capturing of packets that would not have been before.

This problem can be dealt with through scaling the simulator. As more nodes and connections are created to and within the simulator, so the need for more hardware to support the added load is required. This is not always the case however as Moore's law also comes into play. This was a simple observation made by Gordon E. Moore that every two years the number of transistors that can inexpensively be placed on an integrated circuit doubles [26]. This plays a big part in reducing the costs and space requirements needed to set up any system that needs to deal with heavy loads.

Even though this helps when scaling up a network simulator in terms of connections and nodes within the network, one can't go about doubling the size of the simulated network

every two years. This is mainly due again to the implementations of routing tables and the structure of the network leading to a O(n^2) complexity [28]. The result of this complexity means that every time one adds a node, the load cost for adding that single node gets larger and larger because it has to tie back into the routing table and structure of every other node. This means that if one implemented a global routing table containing all nodes in the network. For n nodes, each node would have to know about the other n-1 nodes.

If one were to take the approach of assigning each node their own routing table, one could also spawn a thread to handle that node. There will be no need for mutexes within each node as each node will have an instance in memory and would only deal with its own requests and responses. This conceptually is a very neat way to implement a network simulation of any scale as it removes global variables and focuses more on each node as a separate system rather than a network as whole. This also allows for adding of nodes very easily to the network as one can simply spawn a new thread, give it its connections and let it work out its routing table for itself.

Another key point to this approach is with the development of multi-cored systems, heavily threaded applications are becoming less strenuous to run. Also cheap commercial processors available to the public have features such as hyper-threading which allows multiple threads to on a single core. This, combined with multiple cores on one processor enhances, the ability for modern day inexpensive processors to deal with multiple thread based applications [13] and can be used to up scale network simulation cost effectively.

Research and development into simulation and more specifically network simulation has shown more and more use not only in research but also in testing and application in commercial and public sectors. As such, it is continuing to grow and with development not only in better algorithms but also in hardware, application specific hardware and low cost hardware with high availability will continue to push this field of research and will lead to more efficient and secure networks in the future.

Further more the approaches taken can be broken up into three main categories being hardware, software and hybrids which are a combination for both hardware and software. Hardware approaches, as produced by Apposite Technologies [**?**] and Packet Storm [8], are in most cases less cost effective but easier to set up as the hardware is designed to handle network simulation.

Software approaches are slower in execution due to the fact that the hardware in which the simulator runs on is not designed specifically for network simulation. Although this is a

more cost effective approach, it can lead to complicated software as configuring hardware which original intention was not for network simulation can get tedious. However, a software implementation does allow for modularization as it is far easier to pull out a a software module and stick in a new one than to go about designing a new piece of hardware to fill the required gap.

A hybrid implementation does have its advantages as well as its disadvantages. Being able to modularize software written in a language that compiles byte codes specifically optimized for a hardwares instruction set. This can always be used to offload tasks, such as checksum calculation, to hardware specific devices as to free up the core of the system. The disadvantages of this however are bottlenecks, specifically the time it takes to offload the data required for processing to another device may be unnoticable under light load but can cause major slow downs as the systems takes on a heavier load.

The need for routing simulation rather than emulation is that emulation deals with replicating a platform where simulation allows one to change features about a platform. This becomes useful as one can customize a simulation to help achieve result that by emulation would take longer periods of time or in worst case no results at all.

In closing, knowing about the route a packet takes with in a network is not knowledge to be taken lightly. Inefficient routing can lead to excess traffic and increased delays. Also this knowing about routes with in a network can help with redundancy, either creating redundant connections to increase reliability of a network or removing them where they hold no benefit. It can also help in selecting the best algorithm use within a network to increase efficiency. This is all due to the fact that networks are formed from links and built up from these links and if the links are inefficient in the way they communicate, the results compound into the structures built upon these links.

Bibliography

[1] Adachi, F. Wireless past and future-evolving mobile communications systems. *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences E Series A 84*, 1 (2001), 55–60.

[2] Baker, F. Rfc 1812. *Requirements for IP version 4* , 1995–06.

[3] Baumgartner, F., Scheidegger, M., and Braun, T. Simulating router-and domain characteristics. In *International Workshop on* (2004), vol. 117, pp. 139–145.

[4] BGPVista. bgpVista. Online, 2006 - 2009. Available from: `http://www.bgpvista.com/simbgp.php`.

[5] Bhuyan, L., and Wang, H. Execution-driven simulation of ip router architectures. In *Network Computing and Applications, 2001. NCA 2001. IEEE International Symposium on* (2001), IEEE, pp. 145–155.

[6] Breslau, L., Estrin, D., Fall, K., Floyd, S., Heidemann, J., Helmy, A., Huang, P., McCanne, S., Varadhan, K., Xu, Y., et al. Advances in network simulation. *Computer 33*, 5 (2000), 59–67.

[7] CAIDA. CAIDA Data - Overview of Datasets, Monitors, and Reports. Online, 2002 - 2012. Available from: `http://www.caida.org/data/overview/`.

[8] Communications, Inc. Network emulation with data rates up to 10 Gbps. Online, 2004-2012. site of packetstorm. Available from: `http://packetstorm.com/psc/psc.nsf/site/index`.

[9] Date, C., and Darwen, H. *A Guide to the SQL Standard*, vol. 3. Addison-Wesley Reading (Ma) et al., 1987.

[10] Deering, S. Internet protocol, version 6 (ipv6) specification.

[11] Foundation, N. S. IPv4 Class Defnition. Online, 2012. Available from: `http://www.nsnam.org/doxygen/classns3_1_1_ipv4.html`.

[12] Foundation, N. S. IPv6 Class Defnition. Online, 2012. Available from: `http://www.nsnam.org/doxygen/classns3_1_1_ipv6.html`.

[13] intel. How operating systems can do more and perform better. Online, 2012. Available from: `http://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html`.

[14] Jacobson, V., Leres, C., and McCanne, S. pcap-packet capture library. *UNIX man page* (2001).

[15] Kurose, J. F., and Ross, K. W. *Computer Networking*. PEARSON, 2010.

[16] Leighton, F., Maggs, B., and Rao, S. Packet routing and job-shop scheduling ino (congestion+ dilation) steps. *Combinatorica 14*, 2 (1994), 167–186.

[17] Mahajan, R., Spring, N., Wetherall, D., and Anderson, T. User-level internet path diagnosis. In *ACM SIGOPS Operating Systems Review* (2003), vol. 37, ACM, pp. 106–119.

[18] Malkin, G. Traceroute using an ip option.

[19] National Science Foundation. NS-3. Online, 2011 - 2012. Available from: `http://www.nsnam.org/`.

[20] Newman, P., Minshall, G., Lyon, T., and Huston, L. Ip switching and gigabit routers. *Communications Magazine, IEEE 35*, 1 (1997), 64–69.

[21] Popov, M. Everything convergeda flexible photonic home. In *Microwave Photonics, 2009. MWP'09. International Topical Meeting on* (2009), IEEE, pp. 1–4.

[22] Postel, J. Rfc 791: Internet protocol, 1981.

[23] Rekhter, Y. An architecture for ip address allotaft0n with cidr, 1993.

[24] Rekhter, Y., and Li, T. A border gateway protocol 4 (bgp-4).

[25] Rekhter, Y., Li, T., and Hares, S. Rfc 4271: Border gateway protocol 4, 2006.

[26] Schaller, R. Moore's law: past, present and future. *Spectrum, IEEE 34*, 6 (1997), 52–59.

[27] Smith, L., and Lipner, I. Free pool of ipv4 address space depleted. *Online (February 2011), February* (2011).

[28] Stone, A., DiBenedetto, S., Mills Strout, M., and Massey, D. Scalable simulation of complex network routing policies. In *Proceedings of the 7th ACM international conference on Computing frontiers* (2010), ACM, pp. 347–356.

[29] Tanenbaum, A. S. *Computer Networks*, international edition, third edition ed. Prentice-Hall, Inc., Upper Saddle River, New Jersey, USA, 1996.

[30] Technologies, A. Wan Emulation Made Easy. Online, 2012. Available from: `http://www.apposite-tech.com/index.html`.

[31] Wang, P. Libnet Documentation. Online, February 2003. Available from: `http://libnet.sourceforge.net/libnet.html`.

[32] Wang, S., Chou, C., Huang, C., Hwang, C., Yang, Z., Chiou, C., and Lin, C. The design and implementation of the nctuns 1.0 network simulator. *Computer networks 42*, 2 (2003), 175–197.

[33] Wolf, T., and Turner, J. Design issues for high-performance active routers. *Selected Areas in Communications, IEEE Journal on 19*, 3 (2001), 404–409.

[34] Zhao, J., and Govindan, R. Understanding packet delivery performance in dense wireless sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems* (2003), ACM, pp. 1–13.