A Kernel-Driven Framework for High Performance Internet Routing Simulation

Alan Herbert
Rhodes University
Department of Computer Science
Grahamstown, South Africa
Email: g09h1151@campus.ru.ac.za

Barry Irwin
Rhodes University
Department of Computer Science
Grahamstown, South Africa
Email: b.irwin@ru.ac.za

***Abstract*—The ability to provide the simulation of packets traversing an internet path is an integral part of providing realistic simulations for network training, and cyber defence exercises. This paper builds on previous work, and considers an in-kernel approach to solving the routing simulation problem. The in-kernel approach is anticipated to allow the framework to be able to achieve throughput rates of 1GB/s or higher using commodity hardware. Processes that run outside the context of the kernel of most operating system require context switching to access hardware and kernel modules. This leads to considerable delays in the processes, such as network simulators, that frequently access hardware such as hard disk accesses and network packet handling. To mitigate this problem, as experienced with earlier implementations, this research looks towards implementing a kernel module to handle network routing and simulation within a UNIX based system. This would remove delays incurred from context switching and allows for direct access to the hardware components of the host.**

This implementation evaluates the use of Work Queues within the Linux Kernel to schedule packets, and Radix Trees store routing information in the form of Nodes. The memory and CPU requirements are taken into consideration, and conclusions towards the trade off of resources for higher throughput and speeds are discussed.

Preliminary tests using this implementation method have achieved throughput rates in routing packets from core to end-point host at speeds up to 1 gigabit. This however, is subject to host load, network size and number of packets being routed simultaneously at the time of transaction. These factors are discussed and suggestions made towards further optimisations.

***Index Terms*—Packet routing, networks, simulation.**

I. Introduction

This builds on earlier Exploratory work that looks into high speed network simulation in software on commodity hardware [1].

A. Problem Statement

Routing simulation software implemented outside of kernel space on a UNIX system is subject to many process context switches when handling packet buffering during transmissions. This leads to major slowdowns as the simulation software often has to under go context switching to swap between user space, where data would be processed, and kernel space, where data would be sent and received, to transfer buffers to do each task.

B. Research Goals

The need to simulate is often a question that arises before implementation or further study of a topic. The simplest answer is that not every environment is ideal, as many implementations assume, or there are cases that are overlooked during design, implementation and testing phases of an implementation. These oversights are more easily brought to light through testing of systems in a pseudo-live environment such as a simulation [2].

This research looks into implementing a routing simulator that executes in kernel space. This will allow for a zero-copy architecture to be put in place and take advantage of data structures like work queues and radix-trees to allow for asynchrony and fast data access.

Throughput is also taken into account and this implementation looks to maximize this key feature of packet routing simulation. This leads to the need for pre-processing data before simulation, and storing results for re-use during simulation. The pre-processing is achieved through configuration script generation from traceroutes prior to the simulation instance, this is then loaded at run time. This is discussed in Section III-D of this paper.

Routing within this simulator will be done through the use of forwarding tables. These allow for a reduction in data access within the simulator, as well as a reduction in memory requirements of nodes within the simulated network. This is also explained in section III-C of this paper.

In this paper, the literature review is discussed first in Section II. This section involves explaining why simulation is important, how data is collected, performance trade-offs, and what packet routing actually involves.

In Section III, one can find the design taken in implementing this research and what considerations were taken into account when producing this routing simulator. This is followed by testing of the routing simulator in Section IV where throughput and memory usage are the primary focus of testing. The paper then concludes in Section V.

II. Literature Review

There is a need to understand what approaches have been taken, and research has been done in this field before moving onto the design and implementation of this routing simulator.

This is essential as there is little merit in reproducing results, or going down paths that have already been found to yield little success.

A. *Importance of Simulation*

Simulation is the process of creating a model that represents an implemented system, or system that is still to be implemented, and running tests against it to understand where its strengths and flaws exist [3].

As a system becomes more and more complex, so the probability of errors or oversights increases. These can have major repercussions and lead to failures of a product or a project. Also the average cases, as usually portrayed through graphs and spread sheets, don't always hold in real implementations [4].

An example would be in the case of an implemented server that is required to process ten clients a minute. The average request would appear every six seconds, and so implementation could be aimed at handling this. Now in real world application all ten request could happen in the first 10 seconds, this would cause the server to fail. A simulation run at this point before real world implementation would pick up this problem and lead to its mitigation.

With this in mind, one should beware of misconceptions that may arise because a system has undergone simulation. Just because a system has been simulated, does not mean it is error free. There are some errors that can only be found when a system has already gone live and so one must remain cautious [5].

B. *Packet Routing*

Packet routing is simply the process of moving packets from a source host to a destination host through a network [6]. 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. As networks grow through the addition of more hosts, so the number of connections a single node has grows; this leads to greater complexity within the networks, and so the chances of a single node being connected to only one other node or host decreases. This growth in complexity leads to the need for classification and more efficient algorithms to successfully route packets. Currently we classify routing protocols into three major categories. Interior gateway routing through works link state routing protocols, interior gateway routing through path vector or distance vector protocols, and exterior gateway routing. These combine to remove circular routes within networks, and find shortest paths from packet source to destination[7].

C. *Existing Approaches to Network Simulation*

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 [8] and Packet Storm [9], are in most cases less cost effective but easier to set up as the hardware is designed to handle network simulation. One can find cheaper or free alternatives in software or hybrid systems that exist.

Software approaches, such as NS-3 [10], 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 for which the original intention was not for network simulation can be tedious.

A hybrid implementation does have its advantages as well as its disadvantages. Being able to run hardware that works in parallel with the software implementation can be a major benefit. The disadvantages of this however, is bottlenecks. Specifically the time it takes to offload the data required for processing to another device. This may be unnoticeable under light load but can cause major slow downs as the systems takes on a heavier load.

D. *Collecting Routing Data*

A simple tool to enumerate routes around a network on most platforms is traceroute, the implementation for a Windows based host is tracert. This program is used by passing it a series of options, if any, and then an IP or host name as arguments. These options range from setting the hop limit between nodes to defining what protocol to use when performing the traceroute. The protocols range from using standard TCP [11], UDP [12] and ICMP [13] route tracing methods, to using IP as an alternate as defined in RFC 1393 [14]. UDP is the default protocol used with in the route tracing software on most platforms, and makes use of sending packets to ports that are expected to be closed [15]. This results in a ICMP message being returned about the ports status at which the destination is then marked as reached.

These programs result in showing every hop in the route from source to destination, including delays, name, 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.

CAIDA[1] is an association that can aid in doing just that. They collect data about the Internet from average delays and packet loss to existing routes. Furthermore, they make this data available for research on request [16]. This is particularly useful in recreating a large portion of a network based on the Internet as a model. This can lead into DoS (Denial of Service) attack testing and worst case scenario recovery tests in disaster simulations in an existing real world context.

E. *Performance*

The most resource intensive process that takes place within a routing simulator is the actual routing of packets. This involves searching for where the packet is destined, keeping track of

[1]http://www.caida.org/data

where it is currently and when it has reached the end of its life. This is a fairly trivial task in a single case, however there are many packets in flight at any time within a network. This leads to high CPU and Memory requirements from the host system [17].

There are two main methods in which packet routing is implemented, both with their own flaws. These methods use either a global routing table that stores every route, or forwarding tables at each individual node [17]. Global routing tables allow for a large portion of routing information to be pre-processed. This allows for requirements to be reduced in terms of CPU processing power and instead these results to take up more memory on the host system. Use of forwarding tables in nodes allow for data to be simplified and only forwarding information needs to be stored at each node. This method requires more CPU processing power as routes will need to be calculated at every hop in a route.

There are also restriction that have to be considered when addressing address space. The two main protocols used are IPv4 [18] and IPv6 [19]. The major difference between these two protocols is the unique address space. IPv4 has 2^{32} unique addresses, where IPv6 has 2^{128}; this open up 2^{96} more unique addresses for routing to nodes. When taking the sheer size of IPv6's address space and comparing it against a commodity 64-bit architecture based system, it is clear that there is no easy way to map 2^{128} into 2^{64} of addressable memory in a 64-bit architecture. IPv4's 2^{32} address space can be mapped on a 64-bit system though.

When bringing modern CPUs into account, we see the ever growing foothold that multi-core processors are making in commodity hardware and embedded devices, and even how it can improve networking performance [20]. The link is fairly simple when looking at how multi-core processors that can handle multiple processes in parallel can aid network route simulation. Many packets need to be routed in parallel, and so allowing multiple processes to run on a CPU allows the host system to achieve routing in parallel.

III. Design

A. Kernel Architecture

The use of kernel space over user space is primarily for speed up purposes only, however this method comes with increased risks in the event of a software flaw. User space has been shown to have slower process speeds in both memory-based web servers [21] and in simple mail box and message queues servers [22]. Both systems were run on UNIX based operating systems and made use of network, memory and CPU resources.

User space can be best described as a secure sand-box running within kernel space. In user space, a process has to make a request for resources to kernel space before they are allocated to the process in user space. This requires context switching of the process in user space when requesting a hardware resource to allow the kernel process that handles that resource to be queued on the CPU. What makes this worse is that another context switch is required to go back to the process that made the request in user space. This can happen several times per request [23].

The use of kernel space also allows for an implementation of zero-copy. This essentially means that on receiving or sending of a packet, the buffer requires no coping between kernel space and user space [24]. This is due to the fact that the implementation will be executed in kernel space only. As such, there is no need to pass buffers between the kernel and user space because of this.

Further motivation for a software driven approach within a kernel space context is that when compared to hardware approaches, software scales with time. Hardware only has the the processing power of when it was made, where as software has the processing power of the system in which it is run on. In time dedicated hardware becomes obsolete where as software can just be moved to a newer system.

B. Memory Structure

An effective memory structure is also required for node stores within the simulated network. These nodes hold information such as connections, delays, node IP and name. Memory use should be kept to minimum as there can be numerous nodes within a network, but access times should be as fast as possible too. For this a radix tree was taken into consideration as this system looks up nodes on a IP prefix.

This data structure has the advantage of only allocating memory when required, this allows this implementation to allocate and de-allocate nodes in memory only when required. The other benefit is fairly few memory accesses to reach the node requested for look up as searches are processed through IP prefixes [25]. Although not as fast as an array, which is O(1) in access complexity on standard sequentially processed programs, an array requires memory to be reserved for data structures that haven't even been, or will be instantiated.

A radix tree makes use of of prefix based tree search that in the worst case follows an O(n) complexity [26]. This is useful as IP is a prefix based addressing system and thus is suited to storage under a radix tree.

C. Routing through Asynchrony

The method in which a packet is routed in the simulated network can have a large effect on the hosts throughput ability. The need for a scalable architecture is required for the routing algorithm, allowing the structure of the nodes within the simulated network to closely resemble the Internet. Nodes acting as routers will make use of forwarding tables and keep lists of connections along with delays of each connection to ensure accurate implementation of links within the simulation.

Firstly the use of forwarding tables stored within each node serves as a memory reduction implementation. Links that are found to be similar, look identical under a shared subnet mask, are compressed into a single entry using the same link they were both destined for. This also aids in faster routing as there are less items to search through to find a packets destined outbound link.

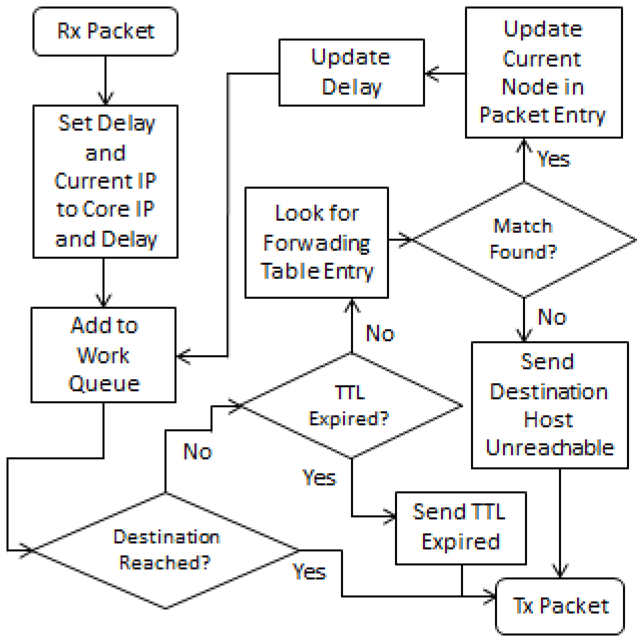
Rx Packet
Set Delay and Current IP to Core IP and Delay
Add to Work Queue
Update Delay
Update Current Node in Packet Entry
Yes
Look for Forwading Table Entry
Match Found?
No
Send Destination Host Unreachable
TTL Expired?
No
Yes
No
Send TTL Expired
Destination Reached?
Yes
Tx Packet

Fig. 1. Routing Logic in Routing Simulator

Delay is another key feature that this network simulation must keep in scope. A delay in a connection is effectively a time period in which no routing is being done, and instead the packet is waiting to be moved. This can be seen as a sleep to a threaded handler.

Rather than using threads, a higher throughput has been shown to come from asynchronous calls to the operating system [27]. The use of work queues allows for just this as it acts as an asynchronous scheduler [28]. One can simply put a routed packet with its status, such as current hop IP and headers, into a work queue data structure. Following this, one simply needs to set the item to be scheduled to asynchronously return from the work queue on the time inserted into work queue, plus the delay to the next hop.

This also removes the excess thread clutter as a thread is only spawned for packets routing when it reaches the head of the work queue. After an item gets put back into the work queue, or is emitted back onto the network, the thread is destroyed rather than put to sleep. This also allows for the delegating the task of scheduling to the kernel, thus keeping design focussed on how the routing is done rather than how to schedule a packet to be routed.

Figure 1 shows a simplification in the routing logic and the application of work queues within it. First we see the packet being brought off the wire by the network handler. As this is a core routing simulator, we are concerned with the routing from the top of the network outwards. This means that there is a delay applied to simulate the packet flow to the core of the network, real packet routing simulation continues from this point. The packet is given a delay to get to the core on receiving it into the simulation, this is then wrapped into a work queue structure, and put into the work queue and scheduled to reach the head of the queue after delay time passes.

We can see once a packet comes off the head of the queue, it has to be routed. This is done through looking up the current node's forwarding table where the packet currently resides. If a forwarding route is found for the packet, it is routed along that connection, else the packet is dropped with a relevant ICMP response to the source IP.

If the packet successfully reaches its destination IP, the sub process that looks up if the simulated node is connected to a real host starts. If a real host is found, the packet headers get modified so that the destination of packet emission reflects that of the real host. The real host then responds back into the simulation if it needs to, and routing continues.

D. *Configuration*

Ease of use is often an oversight in software implementations. In the case of simulation software, being able to generate a re-usable configuration should be included in design. This simulator reads in configuration scripts which contain each node in the network, their forwarding tables' data and simulated node to real world host bindings. This allows for quick configuration as well as fast repetition of testing environments.

The data used for generating reusable configurations should be readily available and easy to acquire. For this data is used from traceroute, as described in Section II-D, to collect and generate data from. The format returned form this program follows a logical ordering and can easily be processed into configuration data files.

These configuration files are simply loaded in through a single command that then reads in and calls all necessary functions to create the network depicted by the configuration. These functions can be called manually as well to allow for fine tuning to replicate specific events within a network.

IV. Tests and Results

These tests are aimed at finding the throughput of this routing simulator, as well as memory requirements to simulate a network. The hardware the system tested on consisted of a Intel Pentium Dual E2220 clocked at 2.40 gigahertz, 2 gigabytes of DDR2 RAM clocked at 800 megahertz, and a NetLink BCM57788 Gigabit Ethernet Controller. This processor was released July 2008[1] and so is fairly dated and so will give a good baseline for testing. If this configuration runs well, then modern hardware can be ensured to be effective with this packet routing software. Linux Ubuntu 12.04 (Precise Pangolin) was used in these tests, the kernel version was 3.2.0-23-generic at time of testing.

A. *Memory Usage*

The network in this routing simulator is buffered in memory. Thus the larger the simulated network gets, the more memory one requires. Memory allocation can be broken down into three main areas; memory allocated to creation of a nodes

[1]http://ark.intel.com/products/32430/Intel-Pentium-Processor-E2220-1M-Cache-2_40-GHz-800-MHz-FSB

within the network, memory allocated to forwarding tables within nodes, and finally memory allocated to packets that are being routed within the simulation.

Calculating the amount of memory needed for packets within the simulation is fairly trivial. This is done by taking the average packet throughput of the simulation in bits and dividing it by eight to get the requirement in bytes, from here the conversion is even easier to megabytes. Due to the nature of packet throughput being sporadic at best, getting a repeatable memory requirement reading is near impossible. The best solution, as for mentioned, is to simply work out an average and account for the overhead with an estimation.

$$Throughput = \frac{\sum(Packet\ Bit\ Size)}{Total\ Time}$$

$$Required\ Memory\ (Bytes) = \frac{Throughput + Overhead}{8}$$

The test for the memory usage of a node and forwarding table assignments is repeatable, and re-usable. This test aims to find the average memory requirements of a fixed size network with a varying number of forwarding table entries per node. The number of forwarding entries is based on an average number and is distributed as such.

The reason for varying the number of forwarding entries lies in the fact that the addition of an extra node in the simulator essentially raises a flag saying that the node exists. This is not too memory taxing. When considering adding forwarding information, where a single node can hold numerous entries, the memory consumption becomes a lot higher and so should be the focus of this test.

In Figure 2 we can see a growing requirement for memory as more forwarding entries are added to the tables of the 1000000 nodes. The trend is linear, as if one creates new nodes according to an average number of forwarding table entries, then the memory requirements for a new node with entries stays constant.

The average requirement for 5 forwarding entries added to each of the 1000000 nodes in the simulated network is 208,9 megabytes, this excludes the initial node creation which is 23,8 megabytes. If one were to use these averages to estimate an expected value on an internet level routing simulation. Using figures from the Internet System Consortium [29], we can estimate that there are close to 1 billion reachable nodes that are open to the Internet. Using this information to simulate every findable node on the internet, one would require around 741,9 gigabytes of memory. A number that considering the scale, is not too large, although expensive to acquire.

B. Throughput

As this is a packet routing simulator, the ability to receive, route and transmit as many packets as possible is a key function of this implementation. The sum of the throughput of all packets sent onto a network cannot exceed that of the total throughput of the routing simulator if it is to receive and route all packets introduced onto the network successfully.

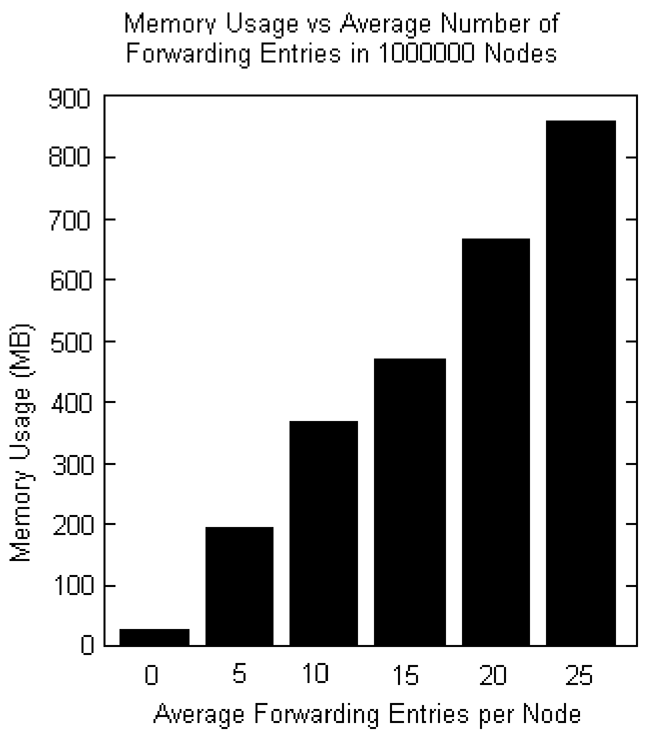

Fig. 2. Memory Usage vs Number of Average Forwarding Entries per Node within 1 000 000 Nodes

As the available hardware at time of testing could only sync at 1 gigabit, these tests aim to achieve this as a throughput. Furthermore, these tests aim to achieve this on supporting hardware, such as CPU and RAM, that can be found in a low-end computer found in store at the time of this testing.

The test entails, sending 128 megabytes (1 gigabit) of data into the simulator using TCP. This is then routed through the simulated network and transmitted to the host that is bound to the destined simulated node. The times taken to transmit this data is then recorded. The size of the data each packet is transferring is set to 1024 bytes (4096 bits), this excludes headers. To work out the throughput one simply divides 1 gigabit by the time taken to transfer it. This results in bits per second and so the routers throughput is calculated.

Figure 3 depicts the results from this test. One would expect the results to be consistent however, as the protocol used was TCP, there are missed packets that need to be accounted for along with other overheads such as packet headers. This brings in extra data into the simulation that is not accounted for in the original calculation; the headers for Ethernet IP and TCP alone being 14 bytes, 20 bytes and again 20 bytes respectively.

The average time taken to transmit 1 gigabit of data over all tests is 1,295 seconds. This brings the throughput of this routing simulator to 829,248 megabits per second. Taking this further and including the overheads for packet headers the total data transferred, without resent packets, is 1.053 gigabit. When reworking the throughput with this figure that includes the headers, one finds the throughput to be 872,869 megabits per second.

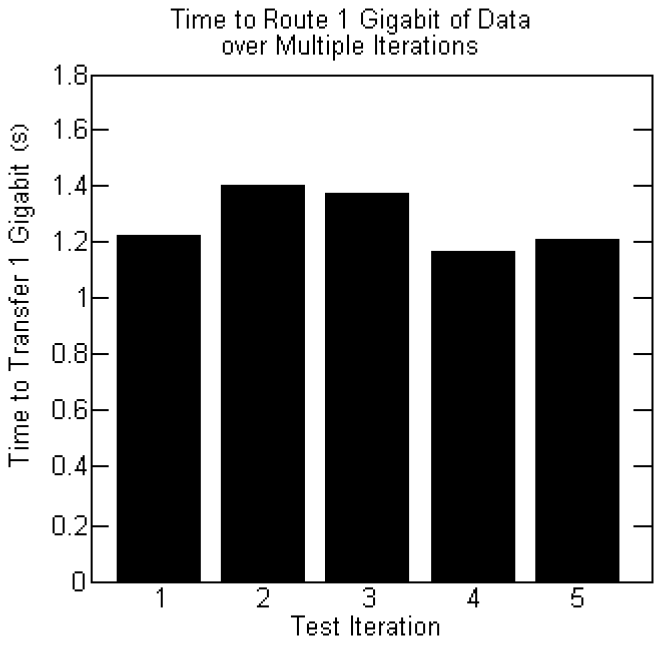

Fig. 3. Time Taken to Route 1 Gigabit Over Multiple Tests

These results prove quite satisfactory as this hardware is found within the entry-level specifications of computers at the time of these tests. This suggests strongly that running this routing simulator on high-end hardware should prove to achieve throughputs well over 1 gigabit per second.

V. Conclusion

This research looked towards the design and implementation of a kernel based packet routing simulator that could achieve throughput greater than 1 gigabit. This was done through the use of work queues to create a asynchronous environment, coupled with the use of radix-trees for fast memory accesses by reduction of memory accesses to lookup a node within the network.

In closing, this routing simulator shows that a kernel based software implementation of a packet routing simulator can achieve throughput of over 1 GBps when using entry-level to mid-ranged desktop computing hardware. This can also be done with a relatively low requirement on memory when considering the trade off of CPU time to memory in pre-calculated forwarding tables and network configuration.

VI. Acknowledgements

This research was supported in part by the CoE in Distributed Multimedia, THRIP & NRF SA (UID 75107). Additional funding was also received form CSIR/DPSS. The content is the responsibility of the author(s)and does not/do not necessarily reflect the views of the funders of Rhodes University.

References

[1] A. Herbert, “Narwhal: An ipv4 core routing simulator,” Rhodes University, Tech. Rep., 2012.

[2] Production Modelling. (2012) Why simulate? Accessed 10 April 2013. [Online]. Available: http://www.simulation.co.uk/solutions/why-simulate.html

[3] GoldSim Technology Group. (2010) What is simulation. Accessed 13 April 2012. [Online]. Available: http://www.goldsim.com/Web/Introduction/Simulation/

[4] TMN Simulation. (2011) What can simulation do for me? Accessed 13 April 2012. [Online]. Available: http://tmnsimulation.com.au/simulation/why-simulate/

[5] T. Santner, B. Williams, and W. Notz, *The design and analysis of computer experiments*. Springer, 2003.

[6] F. Leighton, B. Maggs, and S. Rao, “Packet routing and job-shop scheduling in O (Congestion+Dilation) steps,” *Combinatorica*, vol. 14, no. 2, pp. 167–186, 1994.

[7] F. Baker, “RFC 1812,” *Requirements for IP version*, vol. 4, pp. 1995–06, 1995.

[8] Apposite Technologies. (2011) Wan emulation made easy. Accessed 2 February 2012. [Online]. Available: http://www.apposite-tech.com/index.html

[9] Communications Inc. (2012) Network emulation with data rates up to 10 gbps. Accessed 28 February 2012. [Online]. Available: http://packetstorm.com/psc/psc.nsf/site/index

[10] National Science Foundation. (2012) Ns-3. Accessed 1 March 2012. [Online]. Available: http://www.nsnam.org/

[11] J. Postel, “Rfc 793: Transmission control protocol,” 1981.

[12] ——, “Rfc 768: User datagram protocol,” pp. 1–3, 1980.

[13] ——, “Rfc 792: Internet control message protocol,” 1981.

[14] G. Malkin, “RFC 1393: Traceroute using an IP option,” 1993.

[15] V. Jacobsen, “traceroute. man page, unix, 1989,” *See source code: ftp://ftp. ee. lbl. gov/traceroute. tar. gz, and NANOG traceroute source code: ftp://ftp. login. com/pub/software/traceroute*, 2001.

[16] CAIDA. (2012) Caida data - overview of datasets, monitors, and reports. Accessed 1 March 2012. [Online]. Available: http://www.caida.org/data/overview/

[17] S. Wang, C. Chou, C. Huang, C. Hwang, Z. Yang, C. Chiou, and C. Lin, “The design and implementation of the nctuns 1.0 network simulator,” *Computer networks*, vol. 42, no. 2, pp. 175–197, 2003.

[18] J. Postel, “Rfc 791: Internet protocol,” 1981.

[19] S. E. Deering, “Internet protocol, version 6 (ipv6) specification,” 1998.

[20] Intel. (2007) http://www.intel.com/content/www/us/en/ethernet-controllers/improving-network-performance-in-multi-core-systems-paper.html. Intel Corporation. [Online]. Available: http://www.intel.com/content/www/us/en/ethernet-controllers/improving-network-performance-in-multi-core-systems-paper.html

[21] P. Joubert, R. King, R. Neves, M. Russinovich, J. Tracey *et al.*, “High-performance memory-based web servers: Kernel and user-space performance,” in *Proceedings of the USENIX Annual Technical Conference*, 2001.

[22] J. H. Koh and B. W. Choi, “On benchmarking the predictability of real-time mechanisms in user and kernel spaces for real-time embedded linux,” in *Computer Applications for Security, Control and System Engineering*. Springer, 2012, pp. 205–212.

[23] The Linux Information Project. (2006, May) Context switch definition. The Linux Information Project. [Online]. Available: http://www.linfo.org/contextswitch.html

[24] D. Stancevic, “Zero copy i: user-mode perspective,” *Linux Journal*, vol. 2003, no. 105, p. 3, 2003.

[25] N. Askitis and R. Sinha, “Hat-trie: a cache-conscious trie-based data structure for strings,” in *Proceedings of the 30th Australasian conference on Computer science-Volume 62*. Australian Computer Society, Inc., 2007, pp. 97–105.

[26] D. R. Morrison, “Patriciapractical algorithm to retrieve information coded in alphanumeric,” *Journal of the ACM (JACM)*, vol. 15, no. 4, pp. 514–534, 1968.

[27] D. Kegel, “The c10k problem,” 2006.

[28] T. Heo and F. Mickler, “Concurrency managed workqueue,” Electronic, September 2010.

[29] Internet Systems Consortium. (2012, Jan) The isc domain survey. Internet Systems Consortium. [Online]. Available: http://www.isc.org/solutions/survey