LARGE SCALE NETWORK PROTOCOL EMULATION ON COMMODITY CLOUD

A Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Anirup Dutta

December 2013

LARGE SCALE NETWORK PROTOCOL EMULATION ON COMMODITY CLOUD

Anirup Dutta

I would like to thank Dr. Omprakash Gnawali for continous support in my research and studies. I thank my fellow lab mates at the Networked Systems Laboratory for the stimulating discussions and the fun we had in the last 2 years. Last but not the least, I would like to thank my parents for supporting me through the difficult times.

LARGE SCALE NETWORK PROTOCOL EMULATION ON COMMODITY CLOUD

An Abstract of a Thesis
Presented to
the Faculty of the Department of Computer Science
University of Houston

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

By
Anirup Dutta
December 2013

Abstract

Network emulation allows us to evaluate network protocol implementations, typically in higher fidelity than simulations. Most emulators allow execution of code in the platform or environment similar to the target platform on which the protocol will be deployed. This advantage comes at a cost. Emulation often requires much larger IO or computational resources than simulations. As a result, it is common to see some research projects doing simulations with up to hundred thousand nodes while emulations typically scale up to a few hundred nodes. In this thesis, we present CloudNet, a network protocol emulation platform that leverages the commodity cloud computing service to scale emulations to thousands of nodes. CloudNet uses a lightweight virtualization technique called LXC containers to emulate a single node. The network protocol code and the protocol state for each node is maintained in its respective container. CloudNet then uses properties of the network topology to determine where to place these containers among many physical machines, researchers might rent on the cloud service. CloudNet's careful mapping of nodes to the containers makes network performance more predictable and suitable for emulation even on a shared commodity cloud, which was previously thought to be unsuitable for serious network emulation. Through extensive experiments, we establish that CloudNet is scalable to thousand node networks while providing accurate emulation results.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The networking research community has a long history of building tools and methodologies to evaluate network protocols. Theoretical analysis and simulations are the most common way to evaluate protocols in the community. Theoretical analysis allows us to understand the protocols at the algorithmic level. We can implement the key ideas of the protocol in a simulator and study the protocol by injecting a bit of realism, for example, wireless link model and protocol state machines [12, 18, 16].

Emulation takes this study of protocols a step further. We can evaluate the actual implementation of the protocol, often in the same run time environment as the deployment target. This realism comes at a cost. Emulation often is hardware intensive. The protocol implementations often contain far more features than the original algorithm on which the protocol is based. An emulator must evaluate the entire implementation, not just the core idea behind the protocol. Many implementations assume exclusive access to system resources such as radio or the link. This means the number of machines in the emulator has to be the same as the number of machines in the target deployment. Despite these disadvantages, emulations are widely used because they produce results that are similar to the results on the testbeds, and the target deployments.

Recent advances in node virtualization makes it possible to emulate a large number of nodes in a single physical machine. We can run the protocol code for each node in its own virtual machine. We can configure the networking between the virtual machines using open source tools in the host operating system. We can thus emulate several nodes in a single machine thereby enabling emulation of a large network on a small network of physical machines. This technique however does not scale to very large networks because each physical machine cannot host more than a few tens of virtual machines while providing acceptable performance.

Lightweight virtualization can help scale these emulations to very large networks. In light weight virtualization, *containers* are the unit of virtualization. The protocol code for each node runs in its own container. Each physical machine can host several hundred *containers.* Mininet [11, 6] is a recent effort that uses this approach and shows how to emulate a network of several hundred nodes in a single machine. However, this solution does not scale beyond the number of containers that can be run in a single machine.

In this thesis, we present CloudNet, a network emulator that can emulate networks with thousands of nodes. Our work leverages two technology trends to make this possible. First, CloudNet uses machines provided by the commodity cloud service. The researchers can perform experiments on a large number of machines for the small expense of renting machines from cloud service providers such as Amazon, etc. Second, CloudNet leverages lightweight virtualization technique to further scale the size of the emulation.

We are not the first to propose using cloud services to scale network emulation. The lack of consistent performance in the shared cloud environment and the challenge it presents to network experiments have been identified as the key challenge in building an emulator in the Cloud [19]. In a shared cloud environment such as Amazon EC2, it is possible for one of our emulation instances to share the same physical host with an instance running a web server of a popular website. When the other instances in the same physical machine demand more resources, naturally fewer resources are available to our emulation instances. To address this challenge, CloudNet only uses Cluster instances, which have ample computation and communication resources and come with loose guarantees about resource allocation. This is in contrast to earlier work that used micro or small instances, which make large emulations affordable but come with the inconsistent and unpredictable performance.

While using Cluster instances with ample resources addresses the performance consistency to some extent, it becomes prohibitively expensive to emulate large networks. CloudNet uses light weight virtualization in each Cluster instance to economically scale the emulation while providing more consistent networking performance. Thus, CloudNet becomes as economical as emulation with micro or small instances but with better performance consistency.

In this thesis, we make these contributions:

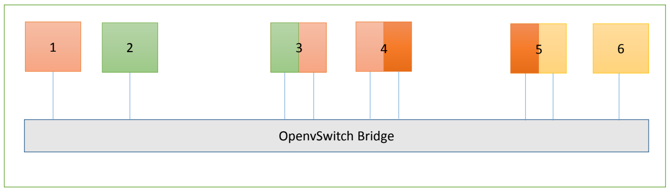


Figure 1.1: CloudNet emulation of a 6 node network. The different colors represent different networks. If we wish to send packets from 2 to 6 we need to do it via 3,4 and 5.

- We show how to achieve a partially consistent networking performance for network emulation in a completely uncontrolled shared cloud environment like Amazon EC2. Such consistency is critical to the development of a correct emulator.
- We present the design and implementation of an emulator that economically scales to thousands of nodes.
- We present experiments to evaluate the correctness of the such large scale emulators. We use those experiments to evaluate CloudNet.
- We present, to our knowledge, the first emulation of DTN protocol on a 1000 node network as a case study that exercises the capabilities of CloudNet.

Chapter 2

Design

In this section, we present the design of CloudNet, which meets these design goals:

- It should scale to thousand node networks and beyond.
- It should be affordable even to a small research group.
- It should provide consistent network performance.

We now describe the design that meets these requirements.

2.1 Network Nodes

CloudNet uses lightweight virtualization mechanism called Linux Containers (LXC) where each container represents a single node. The network protocol code for that node runs in its respective container. We used LXC for launching the containers in the machine. LXC uses Linux kernel mechanisms to provide lightweight virtual system with full resource isolation and resource control for running application or a system [14][4].

In short, LXC containers are stripped down virtual machines which share the same kernel as the host operating system. This reduces the virtualization overhead considerably. It is possible to run hundreds of LXC containers in a single machine. Resource isolation in LXC containers is achieved using control groups. It is possible to isolate the amount of memory and CPU cycles among different containers so that we can ensure fidelity among the different containers or design experiments with heterogeneous nodes. Thus, because of their lightweight nature with sufficient resource isolation, LXC containers are a good choice in node representation in our goal of designing a scalable emulator.

2.2 Network Connectivity

Each LXC container represents a node in the network we wish to emulate. To emulate links connecting different nodes in the network, we need to provide network connectivity between LXC containers.

In our design, LXC container has at least two network interfaces. One network interface is connected to the default LXC bridge and the other network interfaces are connected to the OpenvSwitch bridge. OpenvSwitch [15] is a software switch which can be used to connect virtual machines. It has been designed to be highly scalable, robust and extensible. OpenvSwitch supports many standard protocols and management interfaces. It allows QOS (Quality of Service) for each virtual machine interface. It supports openflow protocol as well as many tunneling protocols like GRE (Generic Routing Encapsulation), IPSEC (Internet Protocol Security), etc. The default LXC bridge and the OpenvSwitch bridge are not connected. We perform NAT (Network Address Translation) out of the box for the LXC containers through the LXC bridge and we create the logical network for emulation experiments using OpenvSwitch. CloudNet provides two ways to define the topology of the emulated network.

2.2.1 Creating Topology Using OpenFlow

This approach of defining emulation topology using OpenFlow [13] is borrowed from Mininet [11]. Network switches perform two main tasks; first they decide where to forward the packets and then they forward the packets based on the decision. Openflow separates the two responsibilities of a switch. A centralized software controller makes decision on how to forward the packet. When there is a link between the nodes in the topology we wish to emulate, the controller allows forwarding between those nodes thereby *creating* an emulated link. We attach an OpenFlow controller to OpenvSwitch with custom rules on how OpenvSwitch must forward the packets.

2.2.2 Creating Topology Using Subnets

We can use subnets to provide or limit connectivity between different nodes in the emulated networks. We use the LXC bridge as the default gateway in our containers.

We setup routing rules such that the packets meant for destination addresses belonging to the same subnets as the network interfaces of the containers will go through the OpenvSwitch bridge. Any packet meant for destinations other than those belonging to the same subnets as container's interfaces are dropped. Thus, if we need to provide a link between two nodes, together with our routing rules, we assign them ip addresses in the same subnet.

Although this approach is simpler to implement and configure, it is non trivial to determine the number of subnets required to emulate a large and complex network. Furthermore, we also need to determine the membership of each subnet. We cast the problem of finding the subnets and their memberships in the network as a problem of finding maximal cliques in the graph that describes the network we wish to emulate. Every maximal clique constitutes a subnet. The nodes belonging to the same clique are members of the same subnet. If a node is a member of multiple cliques, then it will be member of multiple subnets, which also means that it will have multiple virtual interfaces connected to OpenvSwitch. We find the maximal cliques in a graph using the Bron Kerbosch algorithm [1] and we used a software library called ipgraph [2] which implements the Bron Kerbosch algorithm.

2.3 Link QoS

In many network emulations, we wish to use links that have different loss rates or delays. For example, in DTN (Delay Tolerant Networking) [3] emulations, we might want to emulate links with propagation delay of several seconds or even longer. We use Netem and Tc to constrain the bandwidth and delay of a link.

2.4 Distributing Nodes Across Instances

The fundamental limitation of Mininet like approach is that it requires all the emulated nodes to be in the same physical node. Emulating a network with thousands of nodes will require thousands of containers, which will not fit in a single machine. Now, we describe how CloudNet distributes the containers across multiple machines thereby enabling emulation of network with thousands of nodes while providing consistent network performance.

We run OpenvSwitch in each of the instances. All the LXC containers in each instance is connected to OpenvSwitch bridge on that instance. We used GRE tunnels to connect the OpenvSwitch bridges on the different instances. GRE tunnels can encapsulate a wide variety of network layer protocols inside virtual point to point links. Connecting the bridges using GRE tunnels in Amazon EC2 requires using public IP addresses for the instances.[1] CloudNet requires only a small number of Cluster Instances to emulate even a large network, and only the Instance needs public IP address, so public IP addressing provisioning for CloudNet is not a problem.

We use NTP (Network Time Protocol) to synchronize the time across the instances. The synchronized time is essential to the functioning of networked applications and protocols that rely on timestamps for timeouts and other types of timing

[1]One possible reason is that the visibility of the private IP addresses is limited to a certain portion of the EC2 cloud.

based control.

2.5 Choice of Instance Type

Instance type selection is a critical design decision. We can use a large number of small instances or a small number of large instances and virtualize each instance using lightweight containers. Previous work has shown that virtualization in Amazon EC2 causes highly unreliable network performance [19]. However in such work, researchers used small instances. Such inconsistent performance would not meet CloudNet's requirement of *correct* emulation. CloudNet uses a smaller number of instances with more resources in Amazon EC2, then uses LXC containers to decrease the number of instances required to emulate a large network. For example, cluster instances have 60.5 GB of RAM with 88 ECUs (EC2 Compute Unit). These instances can be launched together in a single placement group to assure high and consistent network performance between the instances. To emulate a 1000 node network, with 200 containers per node, we only need five instances. Because we control the execution environment of the containers in each instance, we can provide more consistent network performance across the emulated nodes compared to running the nodes directly on top of Amazon EC2 and leaving it to Amazon to provision the network. Furthermore, there is high and consistent network connectivity between cluster instances. Thus, the emulated nodes within these instances achieve consistent network performance.

2.6 Container Fidelity

One of the most important challenges while performing any networking experiment is that we should maintain fidelity between the different nodes. Cloudnet uses a linux kernel feature called cgroup to to limit and isolate resource usage for the different LXC containers so that each container would only use the amount of resource available to it. Mininet uses a similar methodology and has found that cgroups work pretty well in isolating resources.

2.7 Difference from Mininet

CloudNet borrows the technique of using lightweight virtualization to emulate a node from Mininet. However, Mininet can run only on a single machine, limiting its scalability to the number of containers one machine can host. CloudNet, on the other hand, carefully places the containers on different instances using the maximal clique organization, and connecting them with the topology defined by the user, thereby scaling beyond what a single machine can run.

Chapter 3

Evaluation

In this section, we study how CloudNet satisfies its design goals. We use tools such as iperf to validate the correctness of CloudNet and DTN[3] as a case study to evaluate the usefulness of CloudNet.

3.1 Network Performance in Amazon EC2

Correctness is the most important requirement of any network emulator. To evaluate CloudNet's correctness, we create a network topology using CloudNet and run iperf over those topologies to measure bandwidth and jitter. If CloudNet emulation is correct and consistent, we expect the results of measurements in the experiment to match the specified properties of the topology. Otherwise, CloudNet results are not correct. In addition, we compare the correctness of emulation done on CloudNet vs directly on top of Amazon EC2. A network emulation directly on Amazon EC2 uses

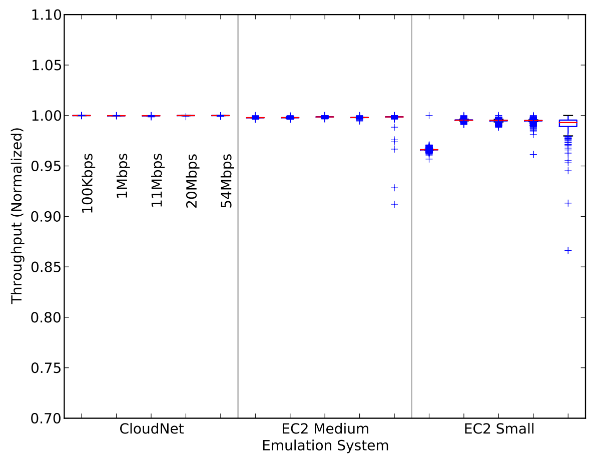


Figure 3.1: Boxplot of iperf throughput normalized by the nominal throughput for the setup varying from 100 Kbps to 54 Mbps.

an instance to represent a node and uses virtual interfaces to connect the instances to create the user specified network topology. This experiment will help us understand if CloudNet provides any advantage over running emulation directly on top of Amazon EC2. In each experiment, we created a small network of 10 nodes over 9 hops and made 50 bandwidth and jitter measurements using iperf. The experiments were performed multiple times a day over five days (to understand if performance consistency holds at different times of the day and different days of the week).

Figure 3.1 shows throughput achieved by iperf when doing emulation with CloudNet versus using Amazon EC2 to do those emulations. The first observation is that iperf achieved constant normalized throughput across multiple experiments with 50 iterations each for all throughput settings. Thus, the measured throughput matches the ground truth. The second observation is that emulation directly on top of Amazon EC2 medium instances are less consistent (despite much higher instance rental

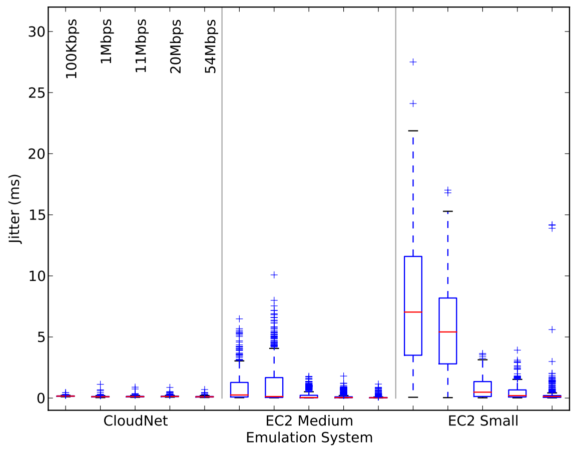


Figure 3.2: Boxplot of jitter in path latency measured during experiments for setups varying from 100 Kbps to 54 Mbps.

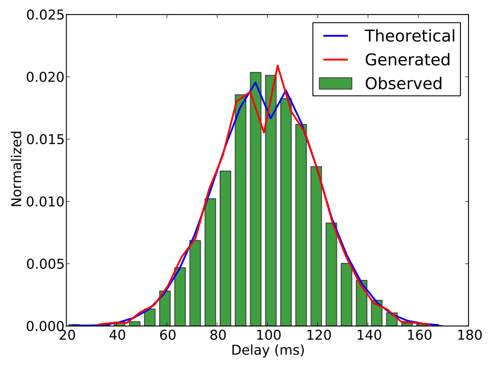
cost as we show later). The results are worse (and unacceptable) with Amazon EC2 small instances. Thus, we conclude that throughput available to emulation on CloudNet is consistent and correct.

Figure 3.2 shows the jitter in path latency measured during the experiments. The topologies used in these experiments were specified to have no jitter. Thus any considerable jitter could lead to inaccurate emulation results. We observe that the jitter with CloudNet is smaller than emulation that uses Amazon EC2 Medium instances, and significantly smaller than the emulation that uses Amazon EC2 Small instances. Thus, we find that latency measurements on CloudNet emulations are close to the ground truth.

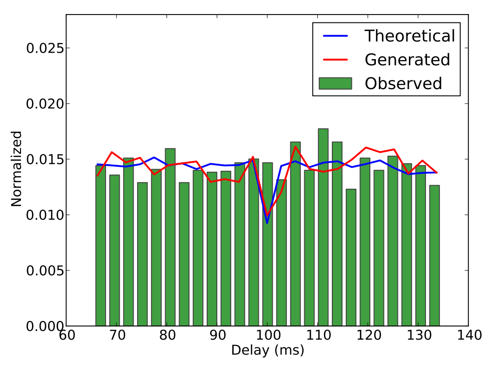
3.2 Correctness of Network Impairments in Amazon EC2

One of the features of Netem is that it can introduce random delay in the network links which can optionally be correlated. The default installation of Tc and Netem on Ubuntu 12.04 comes with 3 mathematical distribution tables which can be used to introduce variable delays in the links. These distribution tables are normal, pareto and paretonormal respectively. We can also make our own distribution table based on some experimental data. To test this feature we created our own uniform distribution table based on a uniformly distributed data for the experiments.

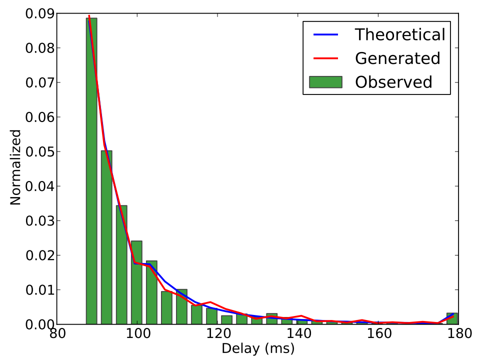
To evaluate the effectiveness of Netem in introducing variable delays, we performed our experiments in a 2 step process. In the first step we setup a 2 node, 1 hop network using CloudNet. We performed experiments where variable delay was introduced in the link based on the different mathematical distributions. In all our experiments we used 100 ms as our mean(μ) delay and 20 ms as the standard deviation(σ). We sent UDP packets between the 2 nodes and calculated the delay for each packet. We collected more than 2000 packet delay values from each experiment. Our hypothesis is that the distribution of delays from each of the experiment should match the behavior of the underlying distribution used for introducing delay in the link. We use the distribution tables and the algorithm implemented by Netem to generate the same number of delay samples as the experiments. We obtain the theoretical distribution plot by generating more than million points using the distribution tables and the algorithm implemented by Netem.



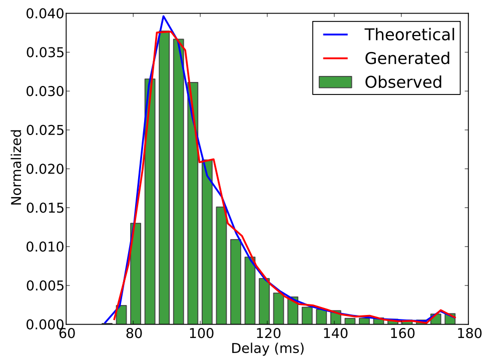
Normal Distribution.



Uniform Distribution.



Pareto Distribution.



Pareto Normal Distribution.

Figure 3.3: Theoretical vs Generated vs Observed plot of delays in a 1 hop network where in each link, delay was introduced based on an underlying distribution.

We compare the result obtained from experiments with the generated sample data and calculate the earth movers distance between the observed and generated delays. Figure 3.3 shows the results from the experiment which compares observed, generated and theoretical data. We see that the plots of the observed delay values confirm to the type of distribution used for introducing delay with less than 0.6 percent error as shown in Table 3.1.

In the next step we used CloudNet to setup a 10 node, 9 hop network. We

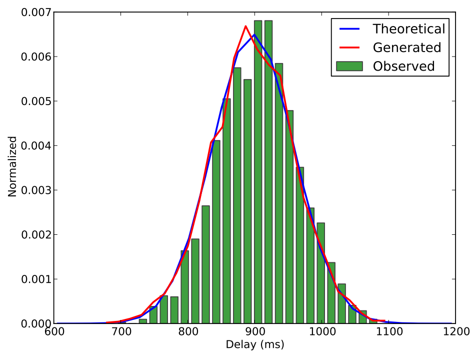
	Normal	Uniform	Pareto	Pareto Normal
1 Hop - EMD	0.6	0.08	0.36	0.04
Error (%)	0.6	0.08	0.36	0.04
9 Hops - EMD	7.29	6.90	7.62	7.8
Error (%)	0.81	0.76	0.84	0.86

Table 3.1: Earth movers distance between theoretical and observed values.

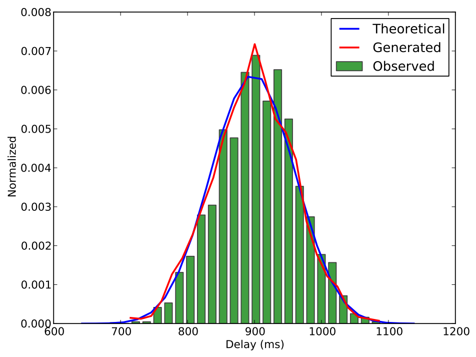
performed experiments similar to what we performed in the previous step. We introduced delay at each link which varied based on an underlying distribution. We sent UDP packets between the farthest nodes. Hence the path length for a single packet transmission was 9 hops. Once again, our hypothesis is that the distribution of delays from each experiment should match the behavior of the underlying distributions used for introducing delay in the links. We compare the result obtained with the generated data and calculate the earth movers distance as shown in Table 3.1 between the observed and generated delays. Figure 3.4 shows the results for the experiment.

We choose experiments where normal distribution was the underlying distribution for varying packet delay in the links for our additional analysis. It is a mathematically known fact that sum of independent normal distributions is also a normal distribution. We performed QQ tests on the observed delay values for the 1 hop and 9 hop experiments to determine if the delays were distributed normally. Figure 3.5 shows the results from the QQ tests.

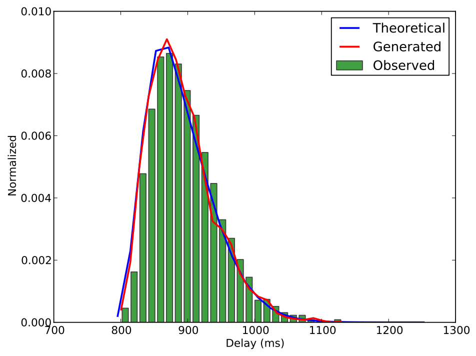
Next we perform the mean test for the observed delay values from each experiment. The mean in case of the 9 hop network should be approximately be equal to 9 times the mean for 1 hop network. We see that all the graphs confirm to that



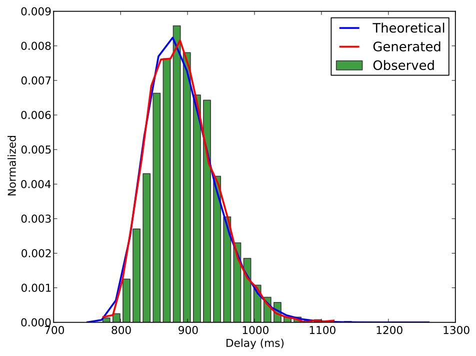
Normal Distribution.



Uniform Distribution.



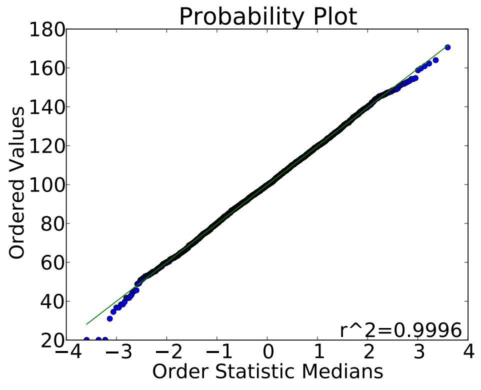
Pareto Distribution.



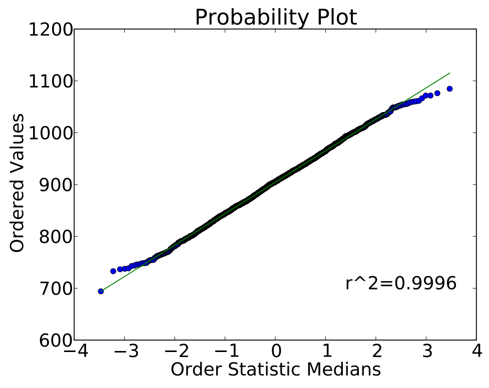
Pareto Normal Distribution.

Figure 3.4: Theoretical vs Generated vs Observed plot of delays in a 9 hop network where in each link, delay was introduced based on an underlying distribution.

characteristic with less than 0.77 percent error as shown in Table 3.2. Through the experiments so far, we find that CloudNet can provide controlled and correct throughput and latency topology to network emulations that run on top of CloudNet.



1 hop



9 Hops

Figure 3.5: QQ plot of normally distributed delays. We performed the normality test on the delay values. P value for 1 hop is 0.22 and for 9 hops it is 0.23.

Distribution	1 Hop Delay (sec)	9 Hop Delay (sec)	9 Hop Delay / 1 Hop Delay	Error (%)
Normal	0.0998	0.9045	9.06	0.66
Uniform	0.1002	0.9044	9.02	0.22
Pareto	0.0994	0.8984	9.03	0.33
Pareto Normal	0.0992	0.9004	9.07	0.77

Table 3.2: Latency analysis for the different distributions.

3.3 Storage I/O

Networking protocols in general don't need high storage I/O since they use memory for storing and retrieving data. However recent protocols like DTN [3] may require high storage I/O. DTN protocol allows communication between heterogeneous networks. DTN uses store and forward approach where data bundles are stored at every intermediate hop before being forwarded to the next hop. In some cases these bundles may be so large in size that we can't save them in the memory and will need to use the disk to save them. We wanted to evaluate the performance of CloudNet

	cr1.8xlarge	cc2.8xlarge
Price	\$3.1 per Hour	\$2.4 per Hour
	cr1.8xlarge (MB/sec)	cc2.8xlarge (MB/sec)
Initial Write	64	34
Re write	205	127
Read	1181	1101
Re Read	1055	1014

Table 3.3: Storage I/O for cr1.8xlarge and cc2.8xlarge instances.

under scenarios where Storage I/O is important. For example if we have a certain DTN topology, where there is a relay node in the middle. There are 50 flows going through that node and for each flow the throughput is 50 MB/s. Storage could be a bottleneck in such a scenario because DTN nodes would need to store bundles before relaying to ensure reliability. We performed an experiment to determine which Cluster Instance should we choose to launch in EC2 for CloudNet based on storage I/O requirements. We chose two kind of cluster instances provided by Amazon for our evaluation, named cc2.8xlarge and cr1.8xlarge respectively. The difference between them is that cr1.8xlarge comes with free SSD storage while cc2.8xlarge come with regular storage. We started 50 independent processes in each instance simultaneously and tried to measure I/O performance. We used iozone [22] for measuring the I/O performance. The results from the tests are shown in the Table 3.3. We see that if we want our DTN application to be able to handle 50 flows at a rate greater than 50MB/sec, we need to use cr1.8xlarge instance. For the tests each process wrote or read a file of size 10M, in blocks of 4096 bytes. The file system was formatted as ext4.

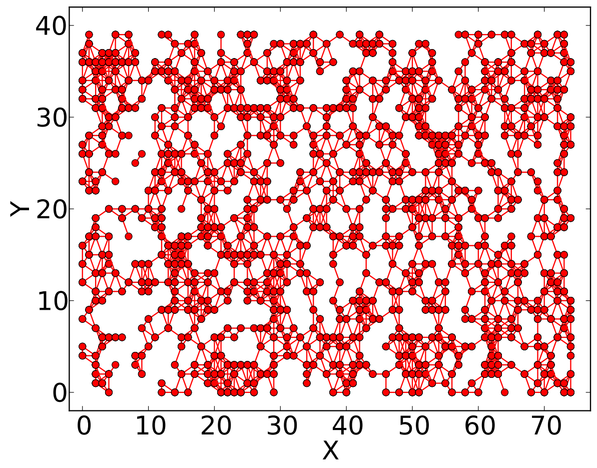


Figure 3.6: A random topology with 1000 nodes to test the scalability of CloudNet.

3.4 Scalability

To test the scalability of CloudNet, we pick a 1000 node topology shown in Figure 3.6. The minimum, average, and maximum node degree in this topology are 1, 6.4, and 13 respectively. We setup this topology using the technique described in Section 2. Every instance contained 200 containers. So we used 5 cluster instances. We split the graph into sections and then randomly picked 100 pair of random nodes from the different sections and sent data between each pair using iperf. The maximum bandwidth for the iperf experiment was limited to 100kbps. The maximum path length between a pair of node was 22 and the shortest path length was 1. We then started the UDP tests between all the pair of nodes simultaneously. The results from the tests are shown in the Figure 3.7. The net throughput achieved per flow is close to the expected value of 100kbps.

Next, we try to understand if the latencies stay close to the ground truth even

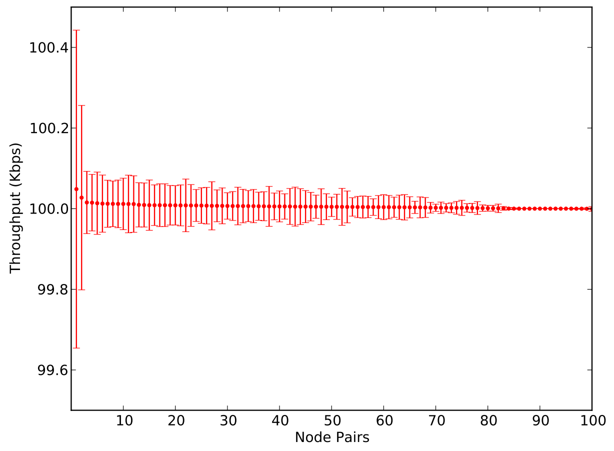


Figure 3.7: Iperf throughput results with maximum bandwidth limited to 100kbps on the 1000 node network. The node pairs have been sorted by the mean of the throughput values.

Hops	RTT	
	Mean (ms)	σ
1	2000.98	2.28
2	4002.16	5.89
3	6005.70	9.20
4	8006.28	10.64

Table 3.4: RTT for paths of different hops in a 1000 node network. Each link was configured with a delay of 1s.

when the user topologies add delays. We emulate a 1000 node network in CloudNet and specify a link latency of 1 second on each link. Table 3.4 shows the mean RTT and standard deviation in delay between random pairs of nodes at hops varying from 1 to 4. In an ideal setup we should have no base latency between a pair of nodes. However as we know that in real life situations there would always be some base latency between a pair of nodes. This is a system overhead of CloudNet. We observe that the overhead increases as expected with the number of hops but stays small.

Thus, we learn that CloudNet offers specified througput and delay in the topology, even at a large scale.

To compare Cloudnet's scalability with Mininet, we perform a 1000 node experiment on Mininet on a 8 core server with 48 GB of RAM. This experiment stressed Mininet in two ways. First, the experiment setup took more than an hour and the machine became unresponsive. Second, we were not able to use ping to send data between the nodes: the system dropped the packets because this experiment was overloading the server. This result should not come as a surprise: Mininet uses a single machine architecture and is designed to work with a few hundred nodes.

Through these experiments, we understand that CloudNet can scale to 1000 node topologies with predictable and expected performance even at this scale.

3.5 Case Study

As a sample case study of a realistic use of CloudNet, we perform large scale experiments with DTN2. By emulating a DTN network, we show that we are able to run real network protocols on CloudNet. We randomly chose 100 pair of nodes from the 1000 node topology Figure 3.6. We use dtnperf and dtping to send data between the node pairs in the topology. As expected, the goodput achieved is higher when the path length is smaller Figure 3.8. The one hop goodput obtained is similar to what has been reported in prior work [17] when the authors did experiments with physical machines. Figure 3.9 shows the time it takes for dtnping to send packets between each node pair in the same topology. The high variation in latency is due

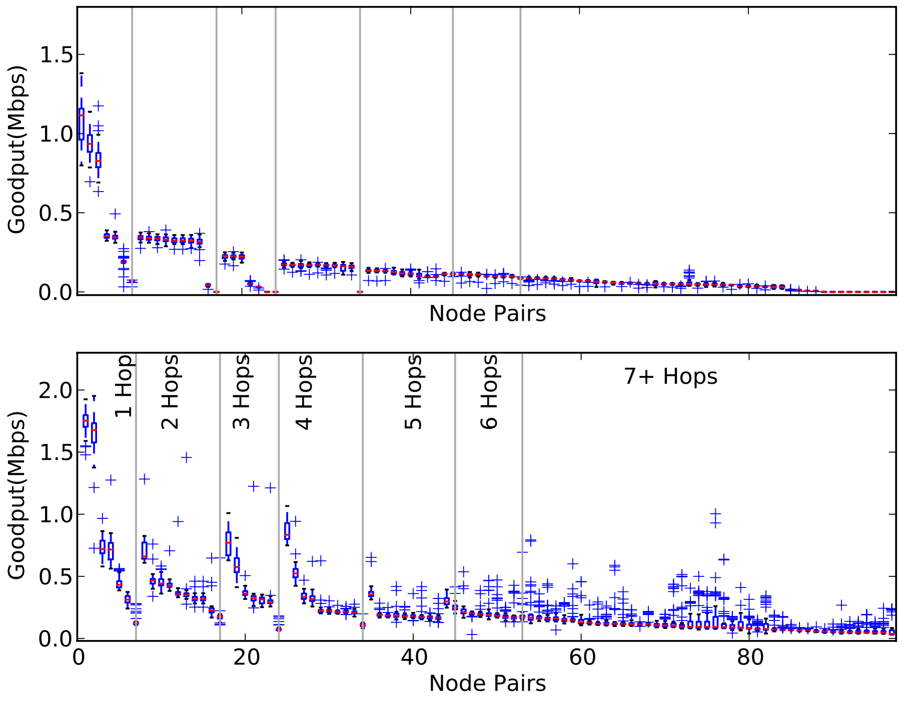


Figure 3.8: Throughput achieved between node pairs using DTN when each link has 1 s latency (top) and without any network impairment (bottom). Payload size is 100KB.

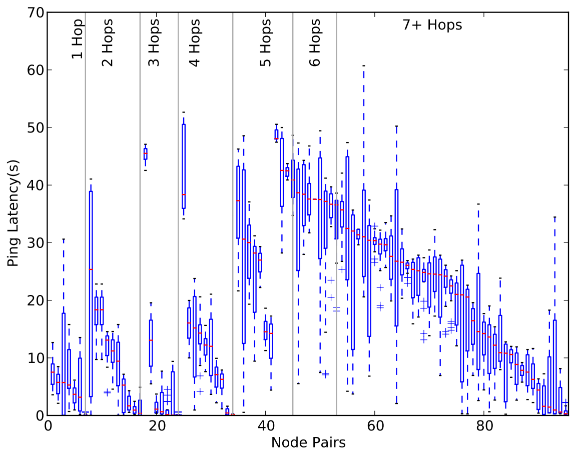


Figure 3.9: End to End bundle latency for each pair of node using dtnping.

	CloudNet	Medium	Small	EC2 Region
Unit Price	$2.4	$0.12	$0.06	U.S. East(N. Virginia)
Number of Instances	5	1000	1000	U.S. East(N. Virginia)
Total Price	$12	$120	$60	U.S. East(N. Virginia)

Table 3.5: Cost analysis for emulating a 1000 node network for an hour.

to bundle queuing in the intermediate hops as different flows cross each other. Thus, the throughput and latencies we obtained in our experiment match the expected output and also what has been previously reported.

Table 3.5 compares the cost of performing a similar experiment using small or medium instances separately for each node. The cost analysis affirms that CloudNet provides a cheaper alternative to performing emulation directly on top of EC2 instances, while providing correct results and scaling beyond state of the art solutions such as Mininet.

Chapter 4

Related Work

Networking community has built a rich set of tools to evaluate network protocols. In this section, we focus on topics related to network emulators to understand the context in which we designed CloudNet.

4.1 Link Emulation

Single link emulation can be done on hardware (using channel emulators) or on software (using tools such as netem and Tc). Prior work compared the results achieved using a hardware emulator and a software emulator using netem and concludes that when correctly configured, netem provides a realistic estimation of impaired network conditions and is sufficient for most networking experiments [10]. One disadvantage of software emulators is that they don't perform well in situations where we need high timer resolution. Netem module comes as a part of the Linux kernel and Tc comes

	Netem	**Dummynet**	**NistNet**
Operating System	FreeBsd	Linux 2.4 - 2.6.14	Linux >2.6
Time resolution	System clock (up to 10 KHz)	Real time clock	System clock (up to 1 KHz) or High resolution timers
Interception point	Input and Output	Input	Originally Output. Newer versions allow Input
Latency	Constant delay	Constant and variable delay.	Constant and variable delay.
Bandwidth Limitation	Yes	Yes	Yes
Packet drop	Yes but without Correlation	Yes and optionally with correlation	Yes and optionally with correlation
Packet reordering	No	Yes and optionally with correlation	Yes and optionally with correlation
Packet duplication	No	Yes and optionally with correlation	Yes and optionally with correlation
Packet corruption	No	Yes and optionally with correlation	Yes and optionally with correlation

Table 4.1: Comparison of software emulators.

as a part of iproute suite of tools. It borrows its idea from Nistnet and dummynet. Though the design goals of the systems are same they have been implemented differently. Nistnet was the original network software emulator for Linux and is no longer maintained. It was a standalone emulation software as compared to netem and tc. Dummynet is available for freebsd. Table 4.1 shows a comparison between different kind of software emulators in a study done by this work [9].

4.2 Network Emulation

Mininet [11][6] provides a system for emulating a large network using limited physical resources. Instead of using system virtualization techniques like KVM or VMware which use paravirtualization, Mininet uses lightweight process virtualization techniques by isolating certain OS resources, thus allowing emulation of large networks in a single machine. Linux kernel has the ability to create network namespaces. Each network namespace will have its own interfaces and routing tables. Mininet exploits this feature of Linux kernel to launch multiple network nodes each of which have their own interfaces and routing tables. Since all the nodes share the same OS kernel as the host operating system, we can run only Linux nodes. Mininet is a very nice way to perform networking experiments when size of the networking experiments is small. It allows to run real unmodified code just like Cloudnet. However, scalability becomes an issue when we want to emulate larger networks since we run into different kind of resource bottlenecks which are related to memory or cpu.

Emulab [7] is another emulator which took OS level lightweight virtualization technique, FreeBSD jails, and modified it to allow multiple virtual interfaces per process group, similar to Mininet and CloudNet. CloudNet provides better resource isolation and flexibility across the emulated nodes than Emulab. CloudNet presents a concrete design that can be deployed on the commodity clouds.

There is some prior work in data centers to optimize VM placement and routing [8]. CloudNet uses the concept of cluster placement groups in Amazon EC2 [20]. A cluster placement group is a logical entity within Amazon EC2. It helps us to create a cluster of instances by launching instances as part of a group. The cluster of instances then provides low latency, full bisection 10 Gigabit Ethernet bandwidth connectivity between instances in the group. In this way we are able to efficiently use the resources in Amazon EC2.

4.3 Network Emulation Timing

Time Warp [5] explores the possibility of using time dilation in network emulation experiments. Its uses the concept of virtual time. The idea is to give the operating system a different illusion to time change compared to physical time. Suppose the maximum bandwidth that a system can support is 100Mbps. We may wish to convince a system that, for every 10 seconds of wall clock time, only one second of time passes in the operating system's dilated time frame. In that way the maximum bandwidth relative to the system would be 1000Mbps.

Thus, for a given CPU or network resource, we can emulate a much faster or slower network by manipulating the rate at which time elapses. Future version of CloudNet may use this technique to offer added consistency in performance for emulations that requires very high bandwidth.

Slicetime is another effort to provide scalable and accurate network emulation [21]. Slicetime makes the simulations independent of real time constraint thus allowing simulation of complex and high performance networks when we have limited physical resources.

4.4 Network Performance in Amazon EC2

In a prior study [19], the researchers tried to characterize the impact of virtualization on the networking performance of the Amazon Elastic Cloud Computing (EC2) data center. They measured various attributes like processor sharing, packet delay, TCP (Transmission Control Protocol) / UDP (User Datagram Protocol) throughput and packet loss among Amazon EC2 virtual machines. Their results showed that even if the data center network is lightly utilized, virtualization can still cause significant throughput instability and abnormal delay variations. However in that study only small and medium size instances were considered and performance in case of the medium instances was significantly better than the smaller instances. The study found that the smaller instances always get around 40% to 50% of the physical CPU sharing. Their suspicion was that Amazon EC2 uses strict virtual machine scheduling policy to control the computation capacity of instances. They also found that medium instances get around 100% CPU sharing for most of the cases. As far as network TCP and UDP throughput performance is concerned it was unstable in case of the smaller instances but in case of the medium instances it was found to be relatively stable.

Chapter 5

Limitations

The experiments in this thesis showed emulation of homogeneous network. CloudNet design accommodates emulation of heterogeneous networks. The containers within an instance share the host operating system. However, emulated nodes of different platforms can be placed on different instances running different OS. Heterogeneity in CPU and memory resources can be implemented directly with the container mechanism.

A disadvantage of our setup is we cannot assure complete fidelity while using Amazon EC2 since we have no control over the infrastructure. However the containers that we launch within an Amazon EC2 instance are completely controllable. We can control the amount of memory, CPU, and bandwidth available to them. Thus we turn an uncontrolled system into a system that provides predictable performance.

Since Cloudnet implements a distributed architecture, we are limited by the maximum bandwidth we can have between the instances. Hence, we cannot emulate large scale networks with high bandwidth links. Cloudnet at present is suitable for performing large scale experiments with low bandwidth requirements. However, this can be resolved if we implement the concept of virtual time in Cloudnet.

Chapter 6

Conclusion

In this thesis, we presented the design of CloudNet, which is an emulator that runs on commodity cloud services such as Amazon EC2. CloudNet's design uses techniques to achieve consistent network performance despite having little control over the EC2 infrastructure over which it runs. Through extensive experiments, we showed that the emulator was able to replicate the results from experiments that used physical machines. We presented the results from emulation of DTN on a 1000-node network. Thus, we showed that CloudNet meets the design requirements of scalability and economic viability in network emulation design. As future work, we will implement CloudNet on other cloud service providers. We will also test other types of network protocols to better understand the limits of Cloud-based network emulation.

Bibliography

[1] Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[2] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695(5), 2006.

[3] Kevin Fall. A delay-tolerant network architecture for challenged internets. In *SIGCOMM*, SIGCOMM '03, pages 27–34, New York, NY, USA, 2003. ACM.

[4] Stephane Graber. Linux containers in ubuntu. `https://www.stgraber.org/2012/05/04/lxc-in-ubuntu-12-04-lts`.

[5] Diwaker Gupta, Kenneth Yocum, Marvin McNett, Alex C Snoeren, Amin Vahdat, and Geoffrey M Voelker. To infinity and beyond: time warped network emulation. In *SOSP*, pages 1–2. ACM, 2005.

[6] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. Reproducible network experiments using container-based emulation. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, pages 253–264. ACM, 2012.

[7] Mike Hibler, Robert Ricci, Leigh Stoller, Jonathon Duerig, Shashi Guruprasad, Tim Stack, Kirk Webb, and Jay Lepreau. Large-scale virtualization in the emulab network testbed. In *NSDI*, pages 113–128. USENIX Association, 2008.

[8] Joe Wenjie Jiang, Tian Lan, Sangtae Ha, Minghua Chen, and Mung Chiang. Joint vm placement and routing for data center traffic engineering. In *INFOCOM*, pages 2876–2880. IEEE, 2012.

[9] Audrius Jurgelionis, J-P Laulajainen, Matti Hirvonen, and Alf Inge Wang. An empirical study of netem network emulation functionalities. In *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*, pages 1–6. IEEE, 2011.

[10] Ezra Kissel and Martin Swany. Validating linux network emulation. `http://damsl.cs.indiana.edu/projects/phoebus/netem_validate.pdf`.

[11] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *HotNets*, page 19. ACM, 2010.

[12] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. In *ACM SenSys*, pages 126–137. ACM, 2003.

[13] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM CCR*, 38(2):69–74, 2008.

[14] Linux Organization. Linux containers. `http://lxc.sourceforge.net`.

[15] Ben Pfaff, Justin Pettit, Teemu Koponen, Keith Amidon, Martin Casado, and Scott Shenker. Extending networking into the virtualization layer. *Proc. HotNets (October 2009)*, 2009.

[16] György Pongor. Omnet: Objective modular network testbed. In *Proceedings of the International Workshop on Modeling, Analysis, and Simulation on Computer and Telecommunication Systems*, pages 323–326. Society for Computer Simulation International, 1993.

[17] Wolf-Bastian Pottner. An empirical performance comparision of dtn bundle protocol implementations. `http://www.ibr.cs.tu-bs.de/papers/poettner-tr201108.pdf`.

[18] Network Simulator. ns-2, 1989.

[19] Guohui Wang and TS Eugene Ng. The impact of virtualization on network performance of amazon ec2 data center. In *INFOCOM*, pages 1–9. IEEE, 2010.

[20] Amazon Webservices. Cluster placement groups in amazon ec2. `http://aws.amazon.com/ec2/faqs/#What_is_a_cluster_placement_group`.

[21] Elias Weingärtner, Florian Schmidt, Hendrik Vom Lehn, Tobias Heer, and Klaus Wehrle. Slicetime: A platform for scalable and accurate network emulation. In *NSDI*, pages 19–19. USENIX Association, 2011.

[22] Don Capps William Norcott. Iozone. `http://www.iozone.org/`.