Presentation Script for Oral Exam 2015

P1

Good afternoon, dear professors. My name is Jiaqi Yan. I am a second year PhD student. Professor Dong Jin is my academic advisor. Today I am going to introduce you my work since my first semester: a virtual time system for container-based emulation of software defined network.

P2

First of all, even if this project is not one hundred percent accomplished, it did get some achievements. 2 research papers are published this summer, one in SOSR and one in PADS. It also appeared as a poster on the PhD colloquium holding by this year's PADS committee.

P3

After both paper are accepted, I also published implementation and experiment code on Github, as you can see from this snapshot.

P4

Now back to my topic. Today I plan to cover these sections. To summary our work in one sentence, we use virtual time to improve temporal fidelity of container based emulation and integrated it with Mininet. To start with, let's see some background first.

P5

Computer network is a large, complex system. It is very important to fully test and evaluate it before deployment. Generally speaking, we have three ways to evaluate a networking system: through physical testbed, run simulations or via emulation

P6

To preserve fidelity, the most straightforward way is to test a prototype on a physical testbed.

Even though now we have several open testbed, it is still technically challenging and economically infeasible to create a testbed for general usage.

Besides, sometimes it is difficult for users to deploy their system on these available platforms.

P7

In contrast, simulation is very user friendly. An simulator try to deal with scalability issue by just modeling the real world system mathematically. However, the fidelity of modeled system is always in question due to abstraction and simplification.

P8

Network emulation comes in as a promising alternative. On one hand, it run real code or unmodified software; on the other hand, it takes the advantage of virtualization to run as much virtual machines as possible.

Moreover, it can also be integrated with simulators in many scenarios to further improve scalability.

In this project, I am particularly interested in container-based emulator for software defined networks. So what is container-based emulator and what is SDN?

P9

Let's look at SDN first. In essence, software defined networking decouples the logic system that make decisions about **where traffic is send** from the hardware that actually forward the traffic to its destination.

The former part is called Control Plane; the later is called Data Plane.

P10

So why SDN? Or what can we benefit after SDN decouples Control plane and Data plane?

First, network operators from various domains, like data center, service providers, enterprises and campus, can manage network with more ease.

For example, SDN has already played an important role in Google and Yahoo in terms of network function virtualization.

Secondly, SDN makes innovation faster. On one hand, many third party vendors can provides flexible SDN solutions for different requirements from user;

on the other hand, combined with network slicing and virtualization, SDN let researchers experiment their ideas on an isolated slice of network without impacting the network's original functions itself.

P11

Now let's go back to container-based emulator. Mininet is a very good example.

- First, it uses linux container to support up to hundreds of nodes in emulation on a single PC or laptop.
  Linux Container provide OS level virtualization through a virtual environment that has its own process namespace and network namespace. So it is more lightweight than other virtualization like Xen.
- Unlike simulator, in Mininet network nodes runs real applications, so the test results are reliable.
- This also makes another good thing possible: all the software and scripts you tested in Mininet can be directly migrated into real world.
- By the way, it is extremely popular in SDN community; this can be proved by these statistics at Github.

So in SDN emulation, typically researchers use Mininet as network environment, build network with Open vSwitch and write controller with POX or NOX, who communicate with open switches using OpenFlow protocol.

P12

However, it is difficult to accurately emulate large scale computer network system under high traffic load in Mininet.

The first reason is that physical resource is limited: target experiment's aggregate resource must fit within the physical machine, pointed out by the very author of Mininet itself.

On the other hand, sharing resources on single platform bring another problem: unlike physical testbed, execution of multiple virtual machines is serialized, or multiplexed in time.

P13

Let’s see the some example that demonstrate the limitation of Mininet. We setup this chain topology in Mininet with forty switches and a controller who routes traffic similar to L2-learning switches. You can see the comparison of TCP throughput between Mininet and a physical testbed. Obviously, Mininet cannot emulate link with more than eight gigabit

P14

Taking the same topology but we varies the number of switches. When all links are configured 4 gigabit per second, we can see Mininet cannot provide reasonable emulation result when switch number is greater than sixty.

P15

We think virtual time can be the answer to these problems.

Like the previous examples, when physical resource cannot guarantee accurate emulation, we slow down emulator’s clock so that time is traded for fidelity.

We borrow the definition of time dilation factor from this paper published in two thousand eleven. By their definition, TDF is the ratio between the rate at which wall-clock time has passed to the emulated host's perception of time passing rate.

For example, setting TDF equals to 10 in emulation, a one-hundred megabit link becomes a one gigabit link from the view point of emulated network hosts.

P16

Virtual time is not a new concept in the world of emulation and simulation. We can see many great works has been done by many researchers.

P17

This table briefly compared related works with respect to the virtualization platform and how virtual time is implemented.

P18

So what is this project's contribution?

In short, we want a virtual time system that is more scalable, more lightweight and more specific to SDN.

Our system should requires zero modification for network applications.

It should also provide accurate emulation results.

P19

Now this figure describe various components of general container based emulator.

In emulation, **containers** virtualize network hosts, where one or more applications are running in it.

According to network configuration, these hosts are connected through **virtual interfaces, virtual links and virtual switches**, thus build a **virtual network**.

P20

Virtual time system can be easily plugged into this architecture.

The **time dilation manager** is responsible for computing and maintaining virtual time. In an separated emulation, all the containers usually share a global TDF. So the time dilation management module provides a **virtual clock** to all of them.

Emulation monitor and **time dilation adapter** together control the speed of virtual clock so that we can run emulation as fast as possible but still guarantee fidelity.

Emulation monitor is responsible for collecting various statistics that reflects the emulation resource in real time. **Time dilation adapter** is a heuristic algorithm that computes TDF for next epoch.

P21

Particularly, we can apply this general architecture to Mininet, and this picture shows the software architecture. Here we see Mininet adopts virtual ethernet to emulate network interfaces under different namespace; adopts **tc** traffic control suite to emulate traffic- and delay-controlled network links; adopts Open vSwitches to build network connectivities. In fact virtual time modules are just very minor modification comparing to the entire Mininet software.

P22

Following this architecture, we implemented a virtual time system that cooperates well with Mininet. Generally, the implementation can be divided into 2 layers.

- First we modified Linux kernel so that operating system can support virtual time.
- Then we let Mininet use this new feature and expose API to its users.

P23

I have published source code at Github and also issued a pull request to Mininet's main stream. Let's see more details.

P24

We add several fields into processes and use them to calculate virtual time in Linux's timekeeping subsystem.

The calculation part is very similar to TimeKeeper's implementation.

We need some variables to be added per process so that we can keep track of how long passed in physical world between two successive *gettimeofay* system call;

then we divide this interval by *dilation* and return how long passed in virtual clock to caller if needed.

P25

Then we need to create containers with time dilation factor. Moreover, we need to change container's TDF so that emulation can be slow down or speed up. The first task can be easily done by modifying system call unshare, which we call *virtual_time_unshare*.

The second task in fact ask us to find all processes inside the container and change their TDF together, which is the responsibility of an additional system call *set_time_dilation_factor*.

P26

To allow applications to run in virtual time with zero modification, we disabled virtual Dynamic Shared Object so that *gettimeofday* can be appropriately dilat-ed.

Even though it is fairly easy to say now, I did put many effort on finding and fixing this issue during development.

P27

So did this similar trick to dilate queue discipline's rate. Mininet use *tc* command suite to emulate high fidelity links with specific bandwidth and latency. No matter what type of queue discipline is used, this trick can ensure virtual links can be correctly emulated.

P28

These OS kernel modification is sufficient to support virtual time for Mininet. It is straightforward to add necessary new API in Mininet and delegate low level works to modified kernel.

P29

As shown in this call graph, virtual time related commands are passed into kernel through backend program written in C.

P30

As recalled from previous architecture, we need resource monitor and TDF adaptor to adaptively adjust the speed of emulation.

P31

For now we choose to monitor emulation's and only emulation's CPU usage with respect to the whole machine.

Using that, we set different thresholds to control when to enlarge TDF and when to reduce TDF.

This part of work actually need more study in the future. And I would discuss more in the end.

P32

To evaluate the virtual time system, we conducted many experiments. This slide summaries our machine's hardware and software configurations. We have patched linux kernel 3.16.3 with virtual time; in the following experiment, we usually use iperf to measure TCP throughput.

P33

The purpose of this set of experiment is to evaluate how virtual time can improve Mininet's performance and scalability.

We create a SDN network with chain topology.

The controller is a simple layer 2 learning switch. When a packet goes in a switch, the switch will see the source address of the packet; so controller will instruct switch to add an entry in its forwarding table.

An entry looks like "if a packet's destination is the source address it just saw, forward it to the past packet's incoming port."

In all experiments, delay usually is not a problem for TCP throughput because we set a very low value here.

P34

So this is the simple network with chain topology.

- To evaluate performance, we varies link's bandwidth.
- To evaluate scalability, we increasing the number of switches in this network.

P35

From the precondition and the simple structure of network setup, we can predict what the emulation result should be.

In this cases with fixed number of switches, TCP throughput should approximate link's bandwidth.

We also create a physical test bed to further validate our expectation of emulation result.

P36

First we can see that Mininet is unable to emulate link with more than eight gigabit bandwidth.

With the help of virtual time, now it can even support a link with ten gigabit bandwidth.

P37

Similarly, on the condition that round trip time is not large enough to have impact on TCP throughput, we want to see if virtual time can further enlarge the scale of network Mininet can support.

P38

When the chain length increases to sixty switches, Mininet cannot ensure a line rate of four gigabit. With virtual time, in contrast, Mininet can support a chain of length 100.

Notice that we cannot further increase the number of switches and expect a line rate TCP throughput *because* too much switches will results in long round trip delay that cuts down the TCP throughput.

P39

In the second experiment, we create a linear topology and test TCP throughput between multiple pair of hosts simultaneously.

Notice that even though we have multiple TCP flows in emulation, they did not share link resources.

So the expected result should be the same among the 3 different phases.

We want to use this experiment to show how virtual time improve temporal fidelity and how adaptive virtual time schedule the speed of emulation.

P40

Again, Mininet failed to give us expected results. The contention is very obvious during phase 2, when we have multiple parallel TCP flows.

P41

On the other hand, if we slow down the emulation eleven times, the disturbance in phase two disappears.

P42

At last, we let the adaptive TDF scheduler to control the speed of emulation. At first we can see some chaos in the beginning of phase 2. However, all TCP flows converge to a stable state quickly.

P43

Comparing to a fixed TDF equals to eleven, the adaptive TDF scheduler runs about only 5.5 times slower than normal situation.

In other words, it reduced forty six percent of running time with little fidelity loss.

P44

In this last experiment, we want to use Mininet to reproduce the limitation of ECMP, which is carefully studied in this paper called Hedera.

Many data centre adopt fat tree topology, as shown here. Due to its multiple root structure, there are many possible path between a pair of hosts.

ECMP based protocol balances traffic load by performing a hashing over packets' header.

P45

However, communication bottleneck will occurs if several large and long-lived traffic flows conflict during hashing.

For example in this figure, we can see two kinds of collisions:

local collision on the link going up from aggregate layer to core layer; down stream collision on the link going down from core layer to aggregate layer.

P46

Now let’s use Mininet to reproduce this phenomenon.

To do this, first we create fat tree data centre network topology with degree four.

Then connect all of the switches to a Ripl-Pox based controller, who routes traffic according to hash based ECMP.

P47

To generate conflicting traffic, we create TCP flows with stride pattern:

when stride step is one, host send traffic to its direct neighbor, not too many collisions will occur in this case.

P48

when stride step is four, many flow may go through core switches, then conflict may occur more frequently during routing.

P49

First we just set links in this data centre to be one hundred megabit per second, which is very unrealistic in any data centre network. Clearly, with stride step equals four, we see a drop from the one when stride is one. This proves our expectation that ECMP works well when no collision but poorly when there are many many collisions.

P50

Then we configure links to be ten gigabit per second. We emulate both with and without virtual time.

With TDF equals to four, the result is very similar to that in the last slide: TCP throughput drops from nine gigabit to less than two gigabit.

In contrast, if without virtual time, there is still drop but the drop is not from a reasonably high throughput.

P51

At last, we take some measurements on the overhead of virtual time system. Generally speaking, it costs very little both when calculating virtual time and pausing emulation during TDF adjustment.

P52

In conclusion, I have shown you how useful virtual time can be to SDN emulation.

It provide virtual resources for emulation and mitigate the serialization of virtual machines.

P53

On the other hand, it is not perfect yet. We still have a lot of works if we want to dynamically schedule the emulation optimally.

One very important question is to have a comprehensive understanding of what resources in the OS and the machine that are shared by Mininet;

moreover, how they are shared and exactly when they are shared. After knowing this we can directly or indirectly set them as fidelity's indicator.

In fact currently I am exploring the detailed process of

- packet transmit and delivery
- the mechanism of software interrupt

- kernel's scheduling algorithm in network stack as well as task
- possible effect that virtual time may have on them.

I believe a better understanding of these processes will help discovery the shared resource

P54

Then, as the result of discussion with Linux timekeeping's maintainer and the author of mininet, I decide to provide a better implementation so that it can be merged into their mainstream.

Typically, the most important thing to do is to eliminate any additional system call and just modify existing system calls to provide virtual time.

Proc virtual file system is a possible way to export virtual time to user space.

Then there are several other interesting features regarding timekeeping, such as freeze and leap.

Moving things into linux module can make a developer's life more easier. So I am also considering doing that some time.

Finally I am very grateful that Jeremy shared their code base of TimeKeeper.

P55

I would like to end up with a quote: time is an illusion.