POLITECNICO DI TORINO

Master of Science in Computer Engineering

Master Thesis

A Policy-Based Architecture for Container Migration in Software Defined Infrastructures

Supervisors
Prof. Guido Marchetto Prof. Flavio Esposito

Candidate
Xu Tao

March 2018

To my family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The recent surge in popularity of Cloud Computing, Internet of Things(IoT) has resulted in a number of wide deployment of IoT network. As new technologies showing up, today's network is much harder and more complex to manage and monitor. Thus, new network solutions come up. For instance, Software Defined Network(SDN) is the latest network theory to solve the complexity of networking. Providing the benefits by detaching networking controller layer and data layer, providing the possibility to use powerful central commands to meet the requirements of underlying demand data planes. Network Functions Virtualization(NFV) is a new method to design, deploy, and manage networking services. Virtual Machine is widely used to implement NFV. Despite VM, Lightwave VM(LVM), such as Docker, is a more interesting solution. Docker allows the true independence between application and infrastructure and developers and IT Ops. It enables creating a model for better collaboration and innovation.

Based on these new network solutions, migration is a new solution widely used in cloud network structure and data center. Migration is the movement of a virtual machine from one physical host to another, it happens without the awareness of end users. It can achieve networking maintain, load balancing, network failure repair for providing an always available system. Apart from these, it also can be used as a security moving target defense strategy.

Nowadays migration solutions mostly focus on VMs [1], and Virtual Routers(VR) [2]. Besides, they are usually in ad-hoc, concerning a specific policy of the migration mechanisms. For instance, loading balancing[3] , optimize energy [4]. There is less concern on container migration. The container is known as the lightweight virtual machine. It not only virtualizes the hardware, but also the operating system. Comparing with the virtual machine, it is much lighter. If there is a high requirement with respecting to the speed for migration, container migration could be a good solution.

Virtualization enables network programmability, software defined network is a good example. Above control plane, it is flexible to develop applications such as

routing, access control, etc. But it is only good for forwarding mechanism. In addition, network protocols are usually designed in the ad-hoc fashion, Different versions of TCP or Routing exist, some of them are suitable for bandwidth sensitive applications, some are for delay sensitive applications, some aim to achieve security, some aim to provide better performance. There is no one-size-fits-all, a policy-based programmable migration mechanism is needed.

We design a policy-programmable container migration architecture based on Docker. The policy-based architecture allows us to change policies with a simple configuration file, so programming the migration mechanism is easy. Second, we test security and load balancing policies within our SDN-based prototype over MININET and GENI. Third, we design & evaluate novel Moving Target Defense solutions inspired by network coding.

The policy-based migration system can do software defined measurement based on the network traffic statistics obtained though SDN controller. We create our algorithms to make migration decision and apply two use cases, one is for load balancing, there are 3 policies: bandwidth-based, shortest path, random. Another is for moving target defense, by creating the novel moving target defense solutions inspired by network coding. There are three policies for moving target defense, Shamir, Digital Fountain, and Pseudo Random function.

This thesis is structured as follows:

- **Chapter 2:** explains the problem that this work intends to solve, along with the existing work related to network migration architecture, container migration solutions, and moving target defense strategies.
- **Chapter 3:** proposes an overview of the architectures on which this thesis is based and the technologies used in the system.
- **Chapter 4:** describes the architecture of the Policy-based Docker Migration System and each component in the architecture.
- **Chapter 5:** details the system implementation including tools, techniques, process, some algorithms designed.
- **Chapter 6:** applies Load Balancing and Moving Target Defense as two use cases, describes the policies used for each use cases.
- **Chapter 7:** provides an overview of the results obtained by measuring the performance between different policies provided by the implementation in SDN environment.
- **Chapter 8:** exposes the conclusions and provides future work that will follow this thesis work.

Chapter 2

State of the art

In this chapter we present the current architectures for network migration, moving target defense strategies, also the existing work on container migration.

2.1 Existing Work on Network Migration Technologies

As the development of virtualization, clouding computing technology, the network becomes more and more dynamic and complex to manage. Also, hardware resource is a considerable cost when deploying software and applications. Virtualization allows sharing the physical resource and runs the application more flexible. Under the virtualization umbrella, migration becomes an essential mechanism for network management. There are several network migration solutions, such as virtual machine migration.

2.1.1 Virtual Machine Migration

System migration refers to moving the operating system and applications on the source host to the destination host, and it can run normally on the destination host. Before when there is no virtualization technology, migration between physical machines relied on system backup and recovery technologies. The status of the operating system and applications are backed up in real time on the source host, then the storage medium is connected to the target host. Finally, the system is restored to the target host.

The Server migration can save the management cost, maintenance costs, and upgrade costs. Previous x86 servers were relatively bulky. Today's servers are much smaller than before, and migration technology allows users to use one server instead of many servers. This saves users a lot of physical space. In addition, virtual machines have a unified "virtual hardware resource". Unlike previous servers,

there are many different hardware resources (such as different motherboard chip sets, different network cards, hard disks, RAID cards, and different graphics cards). The migrated servers can be managed in a unified interface, but also through certain virtual machine software, such as high-availability tools provided by VMware. When these servers are shut down due to various faults, they can automatically switch to another virtual machine in the same network without shut down the service. In a word, the advantages of migration are simplifying system maintenance management, improving system load balancing, enhancing system fault tolerance, and optimizing system power management.

Virtual Machine Migration Classification:

- **Physical-to-Virtual migration**
  The Server migration can save the management cost, maintenance costs, and upgrade costs. Previous x86 servers were relatively bulky; today's servers are much smaller than before, and migration technology allows users to use one server instead of many servers. This saves users a lot of physical space. In addition, virtual machines have a unified "virtual hardware resource". Unlike previous servers, there are many different hardware resources (such as different motherboard chipsets, different network cards, hard disks, RAID cards, and different graphics cards). The migrated servers can not only be managed in a unified interface, but also through certain virtual machine software, such as high-availability tools provided by VMware. When these servers are shut down due to various faults, they can automatically switch to another virtual machine in the same network without shut down the service. In a word, the advantages of migration are to simplify system maintenance management, improve system load balancing, enhance system fault tolerance, and optimize system power management.

- **Virtual-to-Virtual migration**
  V2V migration is the movement of operating systems and data between virtual machines as shown in Figure 2.1. It does not require to take care of host-level differences and dealing with different virtual hardware. The virtual machine is migrated from the VMM on one physical machine to the VMM on another physical machine. The two VMMs can be of the same type or different types. For example, VMware migrates to KVM, KVM migrates to KVM. There are several ways to move a virtual machine from one VM Host system to another.

  There are some popular migration tools, such as VMware's VMotion and XEN's xenMotion, both of them require physical machines to use SANs (storage area network), NAS (network-attached storage) and other centralized shared external storage devices, in this case, during the migration process, it

only needs to consider the operating system memory state, in order to obtain better migration performance.

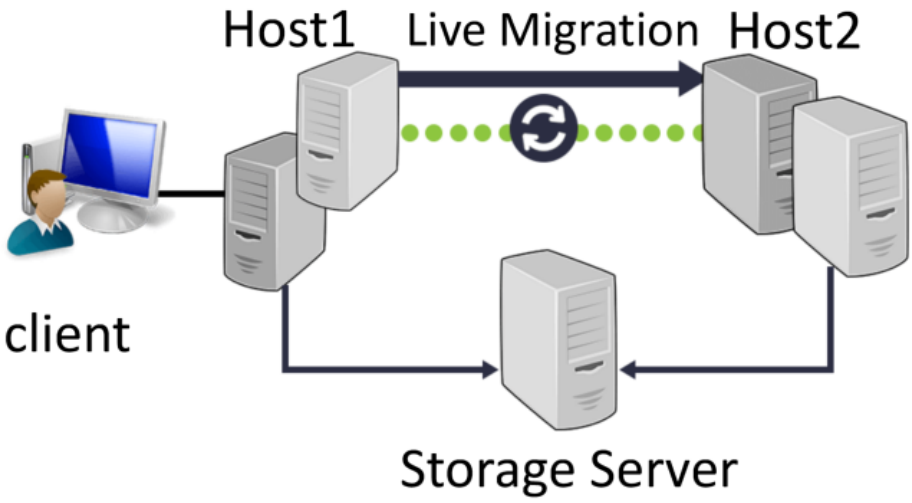

Figure 2.1. Virtual machine migraton

In addition, in some cases where shared storage is not used, live migration of V2V virtual machines can be implemented using storage blocks. Compared to live migration based on shared storage, the live migration of data blocks requires the simultaneous migration of virtual machine disk images, system memory status. And the migration performance is compromised. However, it is possible to use migration technology to transfer computer environments in a distributed local storage environment to ensure the availability of operating system services during the migration process. It also allows to expand the scope of application of virtual machine live migration. V2V live migration technology eliminates hardware and software dependencies, it is a powerful tool for software and hardware system upgrades and maintenance operations. KVM [6] is a tool which is able to do offline or live migration.

There are a lot of work has been done in Live Virtual Machine migration. Some people try to modify current migration solution and improve the exist methods for achieving better migration performance. [7] concentrates on writable working set, they build a high performance migration service with liveness constraints and try to find a way to balance the cost between migration downtime and resource

utilization time. The research is done on how to transfer dirty memory pages for lower the total migration time.

Some research are done for searching the solutions to improve the efficiency for migration. They concentrate on how to minimize the migration time by how to manage dirty memory transfer. But virtualization technique also promotes container(light weight virtual machine) showing up. In these thesis work, we concentrate on container migration. Because in container technique, it not only virtualize the hardware infrastructure, but only the operating system. So comparing with virtual machine, it is lighter. When talking about migration speed, migrate a container could be faster then migrating a virtual machine.

There are a set of papers in which they do comparison and analysis the possible factors that could affect the migration performance. In paper [8] [9], they analysis the major issues of virtual machine live migration with some metrics, e.g downtime, total migration time and transferred data. On the other hand, they classified the techniques and compared the different solutions. [10] analysis the cost of live migration. And the migration overhand in different virtual machine migration solutions.

All these work aims to compare the migration performance of different migration solutions. but no one of them build a architecture to enable programmable migration mechanism. Apart from that, they focus on reducing migration time, not concern about the network traffic situation. Our system performs migration adapting different application needs, it is policy-based programmable system. In another word, we can use different policy just changing the policy name in the configuration file, then programming different algorithms to decide when and where to do migration.

2.2 Moving Target Defense Strategies

Moving target Defense is a new security paradigm. Instead of defending unchanging infrastructure by detecting, preventing, monitoring , tracking and remediating threats, moving target defense makes the attack surface dynamic. The idea of moving target defense(MTD) is applying the same asymmetric disadvantage on attackers by making system dynamic. Therefore, it will make it harder for attackers to explore and predict. The final goal of MTD is to increase the attackers' workload so as to level the cybersecurity playing field for both defenders and attackers.

This strategy can be used in different filed to achieve security, there are plenty of research achievements related to MTD. [11]U-TRI adopts a randomly changing identifier to replace the original static data link layer address. By obfuscating the identifiers in network and transport layer to defend traffic analysis. In our migration system, we try to moving the container from one host to another one, so the hosted machine IP address is keep changing. We create a dynamic container

running surface to confuse the attacker.

WebMTD [12] randomizes certain attributes of Web elements to differentiate the application code from injected code and disallow its execution. This is done without requiring Web developer involvement and browser code modification. It employs the attribute random strategy. In our moving target defense method, we adopt the polynomial function concept to reconstruct the original key by any K hosts presents for decryption process.

Mutated Policies [13] is an attribute-based defense strategy aims for access control. It collects the attributes from the entities involved in the access request process, then carefully select the attributes that uniquely identify the entities, and randomly mutate the original access policies over time by adding additional policy rules constructed from the newly-identified attributes.

[14] proposes a secure mechanism that deploys a limited number of defensive mechanisms, honeypots and network-based detectors optimally and dynamically in a large network. It aims to shorten the lifetime of stealthy botnets by maximizing the number of bots identified and taken down through a sequential decision-making process.

It can be seen that moving target defense technique already has been widely used in different areas. Most mechanisms adopt mutating the attributes or IP address to confuse attacker with the mutated information. However, there is no work related to moving target defense in migration field. Firstly, our moving target defense mechanism is based on randomly moving the container. Secondly, to protect the container destination host IP address, we provide a policy-based mechanism. This model employs multiple methods to perform moving target defense strategy, so it is flexible to choose the speed and security level. Third, the main difference is that we create the novel algorithms inspired by network coding, integrating polynomial concept and digital fountain mechanism in our moving target defense policies.

2.3 Existing Work on Container Migration

Container migration is still a new topic nowadays. There are some study on container migration. Main trends for container migration is migration tools. There is not a complete system for container migration according to network traffic situation or some other policies.

The concept for migration is that moving a virtual machine or application from one host to another one. It requires that storage, dependences, networking and others which are needed to run the new virtual machine or application keeping the same. There are a lot works have been done on virtual machine migration, but the techniques on container migration are not as many as on virtual machine. Container technique virtualizes both hardware and operating system. So migrating a container is similar with migrating a process.

CRIU is a tool for process and container migration, it provide two useful functionalities:

- **Checkpoint** store the current process status as files in userspace.
- **Restore** resume the process using the stored files in userspace.

The way to use criu as follwing:

1. criu dump|pre-dump –t $PID –images-dir=xx . . . . . .
2. criu restore –images-dir=xx . . . . . .
3. cuiu page-server

The detailed commands and usage can be found in the website [15].
when using CRIU for container migration, there are some disadvantages:

- The user process storage
- various status stored in the kernel by process
  - virtual memory mappings
  - open files
  - credentials
  - timers
  - process ID
  - ...

Also, there are some functionalities CRIU does not support:

- properties used as following:
  - Tasks with debugger attached
  - Tasks running in compat mode
  - UNIX sockets with relative path
  - Sockets other than TCP, UCP, UNIX, packet and netlink
  - Cork-ed UDP sockets
  - SysVIPC memory segment without IPC namespace
  - ...
- graphic application

Docker container can be migrated by using two command:

- docker checkpoint $CONTAINER
- docker restore $CONTAINER

But in this way, it can only realize that local container restore and resume, migration can not be done in different hosts.

Another technique for container migration is P.Haul [16]. P.Haul is an extension of CRIU, it enables migration between different hosts. Basically, it is the library of a pair of Go classes, one launched on the source code, and another one on the destination. The working model is shown in Figure 2.2.

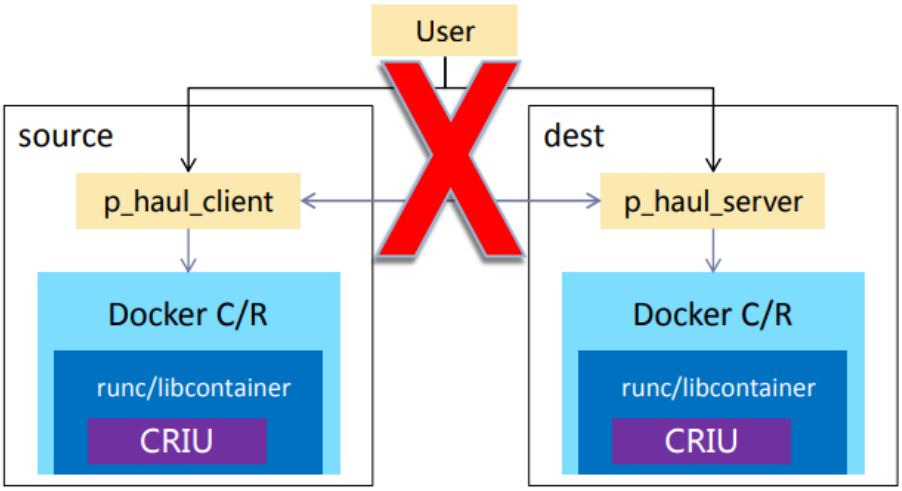

Figure 2.2. P.Haul container migration model

But the disadvantage is that there is no CLI provided. So the user is not able to control the migration process.

Based on this model, HuaWei proposed another model for container migration. In this model, it involves the docker api/cli as user/orchestration as show in Figure 2.3. This model allows that user can take part in the migration activity, and control the migration process. On the other side, the cli can be extended by user. So it is more flexible to monitor the migration process.

In the second model, the migration process has 4 steps:

1. pre-checking
   Validate checkpoint availability, store the basic status files.

2. pre-restore
   Validate restore availability, create container according to configuration file.

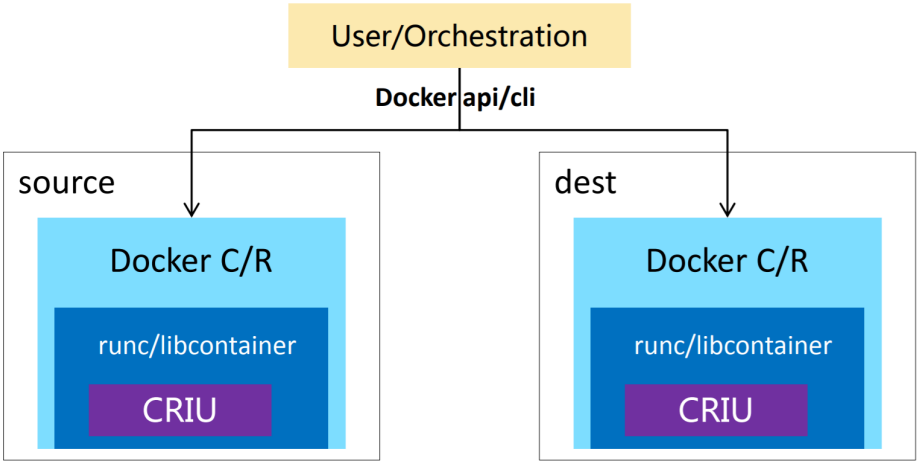

Figure 2.3. Hua Wei container migration model

3. checkpoint
   Store container process running status, back up root file system.

4. restore
   Restore container root file system, resume networking environment using Libnetwork, recover data using volume Driver, restore container process using criu restore.

In this model, it allows that container migration between two different host, but it is still manually migration.

As we can see that the current works on container migration restrict to manually migration by the tools. There is not a migration system that is policy-based, SDN-based to perform the migration according to some criterion. Criu is a tool for container migration. Hua Wei container migration solution is a mechanism that extends the functionalities of Criu, migration can be done between two different hosts. However, this model is not programmable, we can not use any policy to decide where to migrate. In our migration system, we designed the policy-based programmable migration architecture, users can change the policy through a configuration file. In this way, we can have multiple criteria to do container migration. In another word, migration can be done adapting different application needs.

Chapter 3

Background

In this chapter we present the current network solutions, Software Defined Network, OpenFlow, Container solutions, Docker container and software modules which have been integrated and configured to implement the migration system.

3.1 Software Defined Network and OpenFlow

Our programmable policy-based migration architecture is build in software defined infrastructures. Software defined network allows us easily acquiring the network information through controller northbound REST API, getting the under layer device provision through OpenFlow protocol. OpenFlow is the standard protocol, and it can integrated with the software switches which speak OpenFlow protocol with the software defined network controller.

3.1.1 Software Defined Network

In the traditional network architecture diagram, the most important point is the separation of the control layer and the data layer. Each layer has different tasks. Different layers work together to provide the entire data forwarding and routing functions. Here, the control layer is responsible for device configuration and data flow programmability. When we manage a switch, we are actually dealing with the switch's control layer. Routing tables, spanning tree protocols are all calculated by the control layer. These tables are constructed from the transmission of messages packages. Based on these messages, the switch determines the available forwarding paths. Once the forwarding path of these packets is determined, the path information will be sent down to the data layer, usually stored on hardware. This model is very efficient, the decision process is very fast, the overall delay is controllable and the control plane can handle the heavy configuration requirements.

Software Defined Network is the latest solution to solve complexity problem of

Computer Networking. The general architecture is shown in Figure 3.1. The Internet has led to the creation of a digital society, where everything is connected and is accessible from anywhere. It is difficult to configure the network according to predefined policies, and to reconfigure it to respond to faults, load and changes. Software Defined Networking(SDN) is an emerging paradigm that promises to change this state of affairs by breaking vertical integration, separating the network's control logic from underlying routers and switches, promoting centralization of network control,and introducing the ability to program the network. Current networks are also vertically integrated, the control plane(that decides how to handle network traffic) and the data plane(that forwards traffic according to the decisions made by control plane).
Define a SDN with 4 factors:

1. The control and data planes are decoupled. Control functionality is removed from network devices that will become simple(packet forwarding) elements.

2. Forwarding decisions are flow-based, instead of destination-based. Flow programming enables unprecedented flexibility, limited only to the capabilities of the implemented flow tables.

3. Control logic is moved on an external entity, the so-called SDN controller or Networking Operating System(NOS).

4. The network is programmable through software applications running on the top of the NOS that interacts with the underlying data plane device. This is the fundamental characteristic of SDN, considered as its main value proposition.

Firstly, SDN breaks the vertical integration by separating the network's control logic(the control plane) from the underlying routers and switches that forward the traffic(data plane). Secondly,with the separation of the control and data planes, network switches become simple forwarding devices, the control logic is implemented in a logically centralized controller(or network operating system), it simplifies the policy enforcement and network (re)configuration and evolution.

Basically, computer network can be divided in three planes of functionality: the data, control and management plane. The manage plane includes the software service, such as SNMP-based tools, used to remotely monitor the configuration and control functionality. Network policy is defined in the management plane, the control plane enforces the policy, and the date plane executes it by forwarding data accordingly.

3.1.2 OpenFlow

OpenFlow was first proposed as a prototype of SDN, it was mainly composed of OpenFlow switches and controllers.

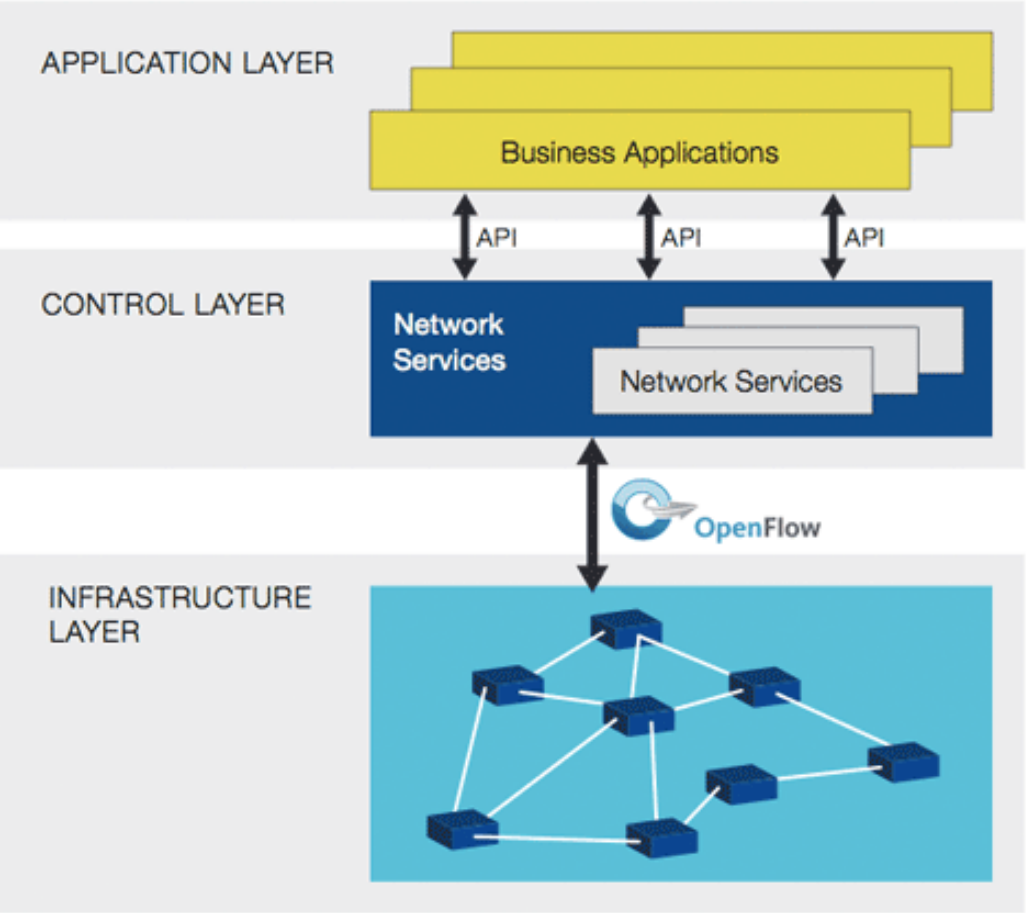

Figure 3.1. General architecture of SDN

OpenFlow switches forward the data packet according to the flow table, it represents the data forwarding plane. The controller implements the control function over the entire network view, and the control logic represents the control plane.

The architecture is explained in detail as flowing:

1. **OpenFlow Switch**
   The OpenFlow switch is responsible for the data forwarding. The main technique consists of 3 parts: flow table, secure channel and OpenFlow protocol, as shown in Figure 3.2:

   The processing unit of each OpenFlow switch consists of a flow table. Each flow table consists of a number of entries. The flow entry represents forwarding rules.

   The switch obtains the corresponding operation for each data packet by querying the flow table. To improve the query efficiency of the flow, the current

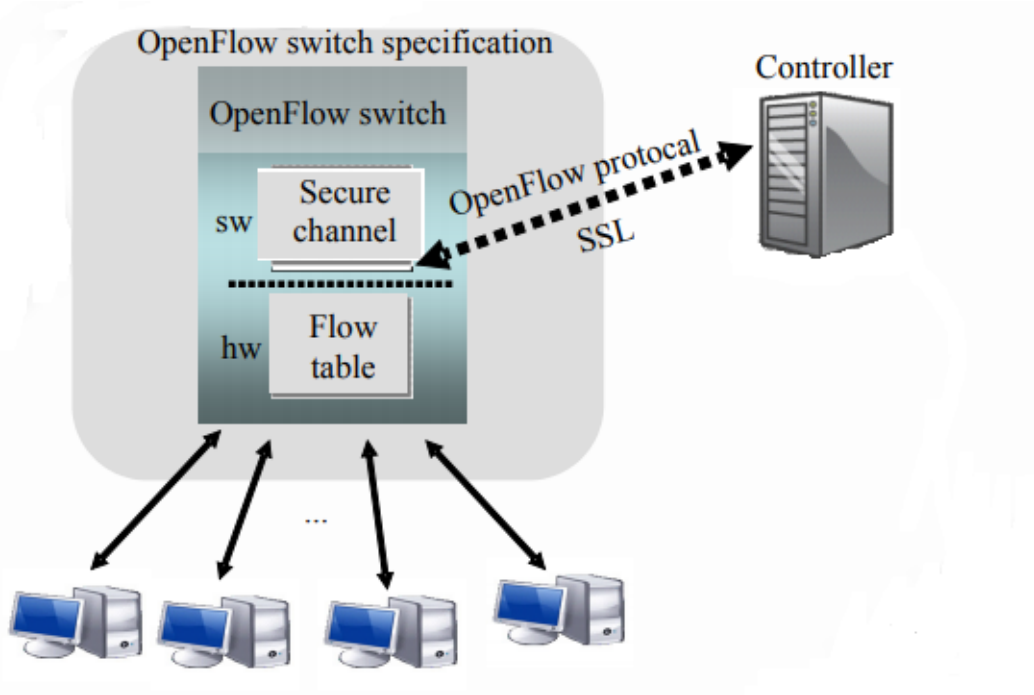

Figure 3.2. OpenFlow switch structure

flow table query passes the multi-level flow table and using pipeline mode to get the corresponding operation. The flow table entry consists of three parts: matching field, counter, and instruction. The structure of the match field contains many match fields, covering most of the link, network, and transport layer identities.

As the OpenFlow specification continues updating, VLANs, MPLS, and IPv6 protocols have also been gradually extended to the OpenFlow standard. Because of the flow filed match mode, in the OpenFlow network, there is no longer difference between routers and switches, but both are known as OpenFlow switches. In addition, the counter is used for flow counting, and the operation indicates the next operation that the data packet matching the flow table entry.

The secure channel is the interface between the OpenFlow switch and the controller. The controller configures and manages OpenFlow switches though this interface and follow the format defined by the OpenFlow protocol. Currently, there are two main versions of OpenFlow software-based switches deployed on Linux systems. User-space-based software OpenFlow switches are easy to operate, modify, but poor performance; core-space-based software OpenFlow Switches are faster and provides virtualization capabilities that allow each

virtual machine to transmit traffic through multiple virtual network cards, but the actual modification and manipulation process is complicated.

2. **Controller**
   In the controller, the network operating system (NOS) implements the control logic function. This was first introduced by NOX [17]. The concept is the central execution unit for programmable control of the network in an OpenFlow network. In fact, the NOS here refers to the control in the SDN concept.

   The system can achieve different logic control functions by running different applications on the NOS.

   In NOX-based OpenFlow networks, NOX is the control core and OpenFlow switches are operational entities. NOX maintains a network view to maintain basic information about the entire network, such as topology, network elements, and services provided. The application running on top of NOX manages and controls the entire network by calling the global data in the network view to operate the OpenFlow switch.

   From the perspective of the functions performed by the controller, NOX implements the basic network control functions and provides a basic control platform for the OpenFlow network. However, there is not much advantage in performance. It fails to provide sufficient reliability and flexibility to meet the scalable requirements. However, NOX is in the design of controllers. The earliest implementation has been as the foundation and template for the OpenFlow network controller platform.

3.2 Floodlight controller

Floodlight controller is a open SDN controller. It is build in Java. First, it can speak OpenFlow which enables working with both virtual and physical switches that using OpenFlow protocol. Second, it is Apache-licensed. So, users can use it to do almost whatever they want for the network. Third, It is easy to set up with minimal dependencies.

The general architecture is showing in Figure 3.3.

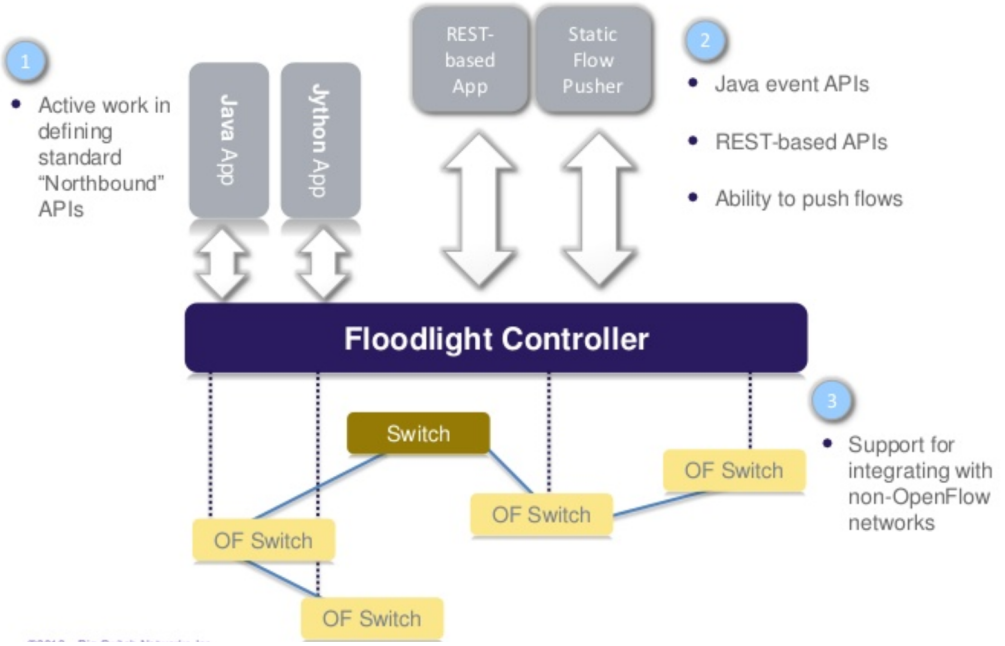

Figure 3.3. Floodlight Controller Structure

In floodlight controller, there are a number of module blocks, they are independent from each other. Modules are exporting "service". Basically, a service is an interface while a module is an implementation. The same service can have several different implementations. All modules are in java. The main module is FloodlightProvider, it manages I/O to switches, translates OF messages to Floodlight event. Besides, it contains multi-threaded via Netty library. Moreover, it provides rich and extensible REST API.

The module description is shown in Figure 3.4.

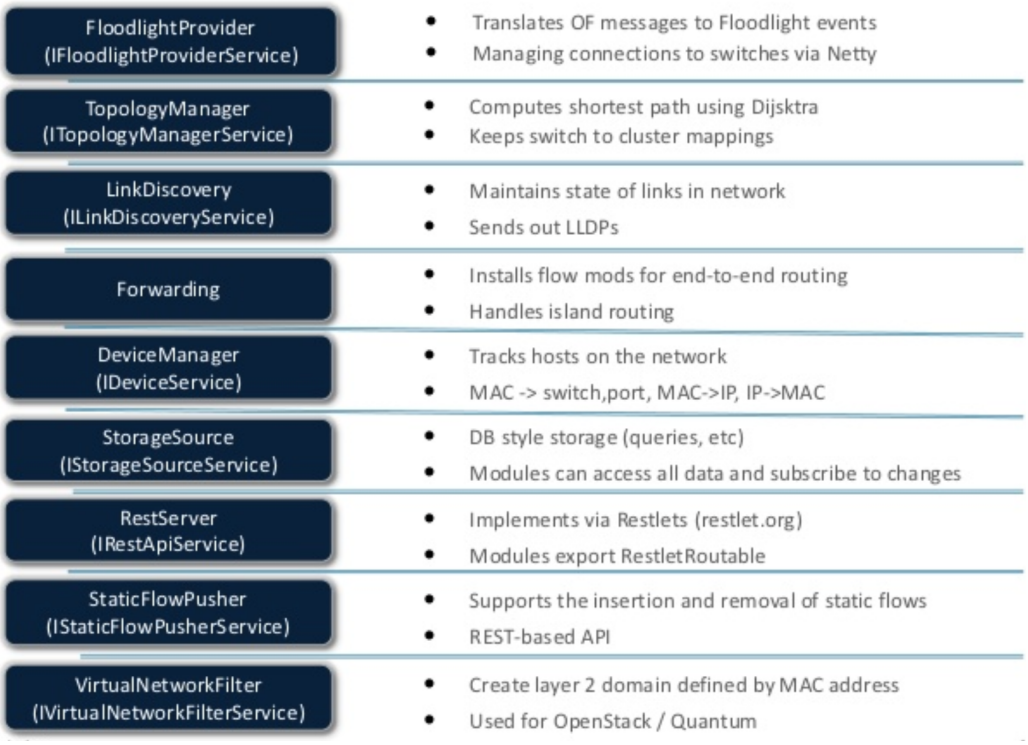

Figure 3.4. floodlight Controller Module Description

In our system, we use the Floodlight Controller northbound API, to retrieve the network traffic statistics in the database. The REST API used in our system as following:

```
http://127.0.0.1:8081/wm/core/switch/all/aggregate/json
{
"00:00:00:00:00:00:00:02":{
"aggregate":{
"version":"OF_13",
"flow_count":"1",
"packet_count":"3740",
"byte_count":"636136",
"flags":[

]
}
},
"00:00:00:00:00:00:00:03":{
"aggregate":{
"version":"OF_13",
"flow_count":"1",
```

```
"packet_count":"3360",
"byte_count":"610786",
"flags":[

]
}
},
"00:00:00:00:00:00:00:01":{
"aggregate":{
"version":"OF_13",
"flow_count":"1",
"packet_count":"3361",
"byte_count":"607897",
"flags":[

]
}
}
}
```


```
http://127.0.0.1:8081/wm/statistics/bandwidth
/00:00:00:00:00:00:00:01/1/json
```


```
[
{
"dpid":"00:00:00:00:00:00:00:01",
"port":"1",
"updated":"Mon Sep 11 21:39:33 CEST 2017",
"link-speed-bits-per-second":"10000000",
"bits-per-second-rx":"0",
"bits-per-second-tx":"235"
}
]
```


```
http://127.0.0.1:8081/wm/routing/paths
/00:00:00:00:00:00:00:01/00:00:00:00:00:00:00:03/2/json
```


```
{
"results":[
{
"src_dpid":"00:00:00:00:00:00:00:01",
"dst_dpid":"00:00:00:00:00:00:00:03",
"hop_count":"2",
"latency":"24",
```

```
"path_index":"0",
"path":[
{
"dpid":"00:00:00:00:00:00:00:01",
"port":"2"
},
{
"dpid":"00:00:00:00:00:00:00:02",
"port":"2"
},
{
"dpid":"00:00:00:00:00:00:00:02",
"port":"3"
},
{
"dpid":"00:00:00:00:00:00:00:03",
"port":"2"
}
]
}
]
}
```


3.3 Container Technologies and Docker

The container is a lightweight, portable, self-contained software. OS-level virtualization allows packaging technology that allows applications to run in the same way almost anywhere. Developers create and test containers on their own laptops and can run on production systems and virtual machines, physical servers, or public cloud hosts without any modification.

Many years ago, applications were monolithic, they usually contain a large block of a binary library, they were usually built on a single stack such as .NET or java. It is long lived, usually takes a long time to develop new patches and functionalities. they are usually developed in a single server, integrated all the functionalities in the same place, and data are stored in the backend. Today's applications are constantly developed, new versions are being deployed often, and entire applications are built from loosely coupled components, deployed to a multitude of servers, a piece of the application talk to each other.

Applications are built with a lot of small parts which are called micro services.Container makes your application portable, it looks the same everywhere, no matter where you run it. Another benefit of the container is that it doesn't need

you to install all the application dependencies on your host. All the necessary components that are required to run an application are packaged as a single image and can be re-used. While an image is executed, it runs in an isolated environment and does not share memory, CPU, or the disk of the host OS. This guarantees that processes inside the container cannot watch any processes outside the container.

Difference between virtual machine and docker

Our traditional virtual machine needs to emulate the entire machine including the hardware. Each virtual machine needs its own operating system. Once the virtual machine is started, the pre-allocated resources will all be occupied. Each virtual machine includes applications, necessary binary and libraries, and a complete user operating system.

The container contains the application and all its dependencies but shares the kernel with other containers. The container runs on the host operating system and in the user space as a separate process.

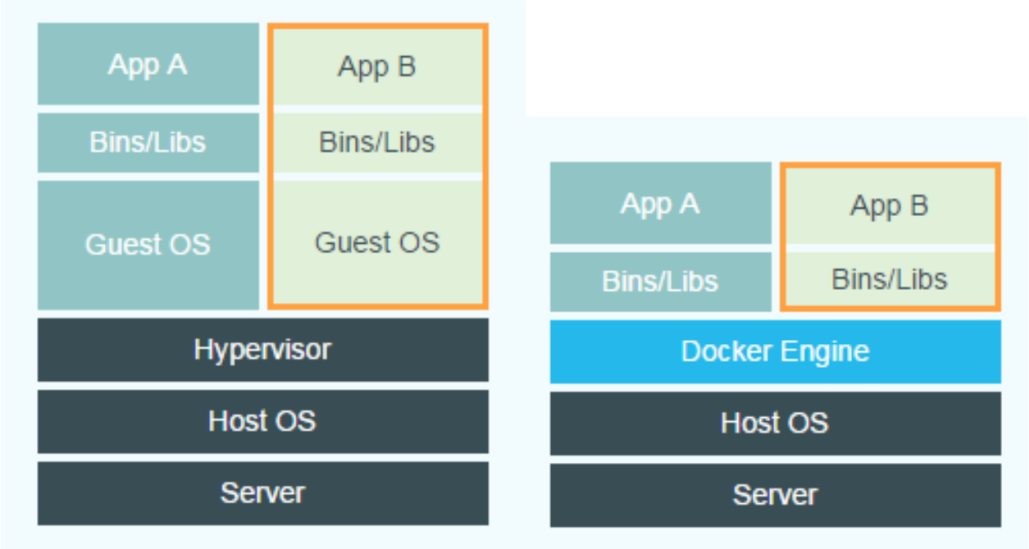

Figure 3.5. container and virtual machine structure comparison

As shown in Figure 3.5, both virtual machines and containers are on the top of hardware and operating system. The virtual machine has a hypervisor layer, and the hypervisor is the core of the entire virtual machine. it provides a virtual operating platform for virtual machines and manages the operating system of virtual machines. Each virtual machine has its own system and system libraries and applications.

Containers does not have a hypervisor layer, and each container shares hardware resources and operating systems with the host. The cost of hypervisor does not exist on the linux container.

However, virtual machine technology also has its advantages. It can provide a more isolated environment for applications without causing any threat to the host due to application vulnerability. It also supports virtualization across operating systems. For example, you can run windows virtual machines as a host system over the Linux operating system.

From the perspective of virtualization, the traditional virtualization technology is the virtualization of hardware resources, and the container technology is the virtualization of processes, which can provide more lightweight virtualization and isolation of processes and resources.

From the architectural point of view, Docker is two layers less than virtualization. It eliminates the hypervisor layer and GuestOS layer. It uses Docker Engine for scheduling and isolation. All applications share the host operating system. Therefore, Docker is lighter than virtual machines in terms of weight. It provides better virtualization performance. From the application point of view, Docker and virtualization have their own areas of expertise and have their own advantages and disadvantages in software development, test environment, and production and maintenance.

Because of the light weight, when the migration has the high requirement of speed. Migrating a container will be faster than migrating a virtual machine.

3.4 Moving Target Defense

As a dynamic and active defense technology, moving target defense can frustrate the attacker by constantly changing the attack surface. Moving target defense has a significant meaning for the network security.

With the wide development of computer network information systems in various industries, network information resources have been shared and fully utilized, network security became a serious problem to be solved.

Although traditional network defense technologies, such as authentication, access control, information encryption, intrusion detection, vulnerability scanning, and virus prevention technologies, can provide a certain level of security. they have evolved along with the automation of attacks, the acceleration of speed, and the diversification of methods of attack. Traditional network defense methods are ineffective. At the same time, the continuous increase in the complexity of the network environment makes the network administrator's work more and more onerous, and one-time negligence may leave serious security risks.

Moving Target Defense is completely different from previous network security research idea. It doesn't eatablish a complete system to defense attacks. Instead, the idea of moving target defenses is that the mechanisms, strategies, evaluating, and deploying are dynamic and constantly changing. This changing can increase the attacker's attack difficulty and cost, effectively limit the exposure of vulnerability

and the opportunity to be attacked, and improve the flexibility of the system. Due to the dynamic change characteristics of the moving target defense technology, it can overcome the shortcomings of traditional network defense technologies and make the defense party from passive to active.

Moving target defense technology focuses on the following aspects:

The attack surface transfer method. The attack surface of the system refers to the system resources that can be exploited and attacked by the attacker. Attack surface transfer is the essential concept of the moving target defense technology, and it is also a hot issue of moving target defense research. Many scholars proposed a method to transfer the attack surface. According to the method used for the transfer, the moving target defense technology can be divided into three categories:

1. **Disturbance**

   Disturbance is adjusting the configuration of the system dynamically to disturb the attacker's attack behavior, such as IP address and port transition.

2. **Diversification**

   Diversification is a kind of moving target defense technology that provides an input, an interpreter, and a software stack component that have the same functions and different implementations. In this way, it can confuse and defense the attacker.

3. **Redundancy**

   Redundancy is a moving target defense technology that protects the system by providing multiple copies of data, services, or nodes to determine whether the system has been compromised.

In our system, migration can be used as a kind of moving target defense method. Because here we try to move the container which is running a server or inside a data center. By moving the container from one host to another one, the host IP address will be changed. In this way, the network is more dynamic, and server always changing host machine. It can make the attack confused about where is the container service running. By applying some algorithms to the moving target defense policies, we could add more difficulties to the attacker.

Chapter 4

System Architecture Design

In this chapter, we will focus on the system architecture design inspiration, as well as the usage of each component. Furthermore, database design and some essential data structures will be presented and explained. The architecture aims to achieve a programmable policy-based migration system, provides flexibility for adapting different application needs by just changing one parameter in the system configuration file. Besides, the system is designed that can collect network traffic statistics leveraging software defined network controller, then software defined measurement will be applied based on the statistics. Thus, a more accuracy migration decision will be made by the policy used based on the current network situation.In addition, migration process could be performed from the migration source host which running the container to destination host within cloud edge.

The key value of this architecture is to enable the container migration between different hosts along with programmable policy-based mechanism.

A well-designed migration system should be able to answer three questions:

1. **Which container(LVM) should use migration?**
2. **When should we migrate?**
3. **Where to migrate?**

Following those questions, we designed our system architecture. Each component in our architecture is responsible to answer one or more these questions.

4.1 System Architecture Overview

Figure 4.1 shows the general architecture and key components of the system. The green blocks are my contribution. There are 4 main blocks designed for migration system, Database & VIB, Software defined measurement, migration manager, migration daemon. The database is designed to store the network traffic statistics,

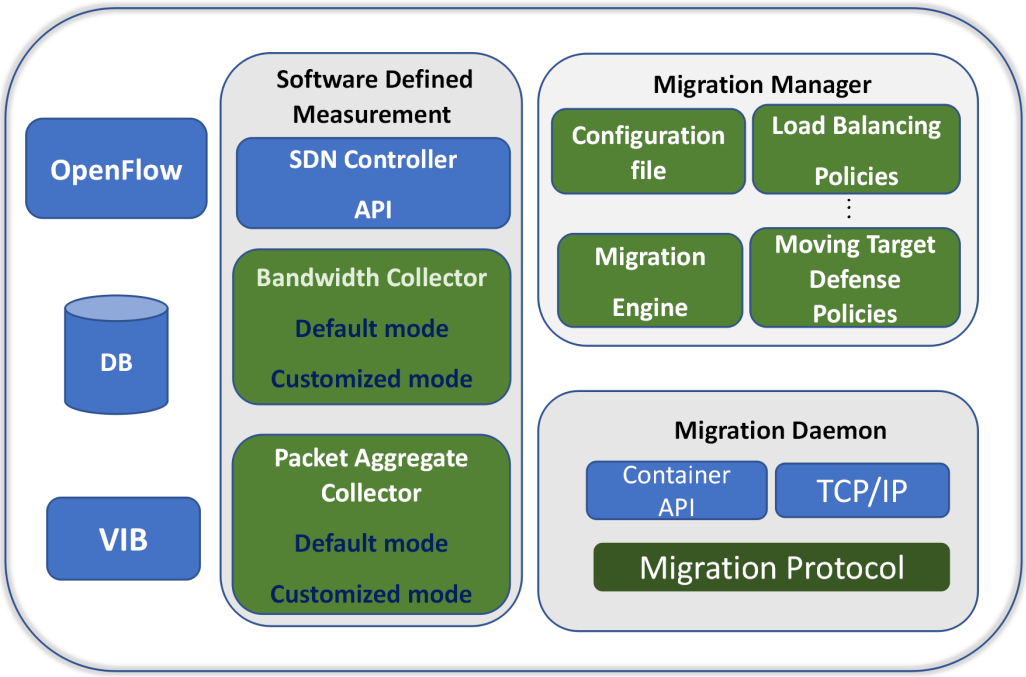

Figure 4.1. System Architecture and Components

migration manager is for maintaining policy sets, executing the specified policy in the configuration file, in order to make migration decision. Migration daemon basically is the migration process that running on each host which running the container service. In the following sections, we will explain the details of each component.

4.1.1 Software Defined Measurement

Software defined network(SDN)enables network traffic measurement more flexible and efficient. Software Defined Traffic Measurement with OpenSketch [18] is a good example by leveraging SDN. It provides a programmable software network traffic measurement tool by separating the data measurement plane from data plane.

Inspired by OpenSketch, we create our software defined measurement system in our SDN-based prototype environment. SDN controller provides the API for acquiring the network traffic statistics. We concentrate on the switch aggregate packets number and bandwidth consumption in each switch port. We collect network traffic statistics through SDN controller northbound API and stores the data in the database. There are two statistics collectors, one is bandwidth collector, another is aggregate collector:

- **Bandwidth collector** it is used for getting the bandwidth consumption per

switch per port at each query time from SDN controller.

- **Aggregate collector** it is used for getting the aggregate packet number per switch at each query time from SDN controller.

Both of them have two working modes. One is default mode which is used by collector with a standard query frequency. When the aggregate packet number exceeds the threshold, the switch will be detected and put in target migration switch set. Then it will enter the customized mode by changing a smaller monitor frequency among the switch set.

Software defined measurement system enables the simplicity and flexibility for collecting the network traffic statistics. By using two working modes, the accuracy of selecting the migration source host will be improved.

Software measurement block answers the essential migration question 1, and 2.

4.1.2 Migration Manager

Migration Manager is the core component to realize policy-based architecture. It is responsible for maintaining the system configuration file, policy sets and executing the required policy to make migration decision. In the configuration file, it specifies a set of threshold parameters, system running environment parameter and the required policy name.(see table 4.1)

Each policy set can be used for a use case. For example, here we implement two policy sets as two use cases, one is for load balancing, and another is for moving target defense. However, the user can implement and integrate the policy set to the system according to their perspective of the application needs.

Migration Engine executes the corresponding policy which specified in the configuration file to make migration destination decision.

Parameter	**Value**
port	SDN controller REST API port
image	Migration container image name
container	Container name
hostInfoDir	Path where store the host information
lookupTableDir	Path where store the look up table for moving target defense
serverdeployRatio	Ratio of hosts in user infrastructure that will run the container at system initialize stage
policy	Name of the policy will be used
securityMode	Name of the policy for security(moving target defense)

keyNumber	Number of the keys needed for moving target defense(default 4)
aggregateThreshold	Threshold of the switch aggregate packet number to trigger migration used by aggregate collector in default mode
aggregateDifferenceThreshold	Threshold of the switch aggregate packet number difference used by aggregate collector in customized mode
aggregateDefaultFrequency	Default monitoring frequency used by aggregate collector in default mode
aggregateCustomizeFrequency	Customized monitoring frequency used by aggregate collector in customized mode
bandwidthDefaultFrequency	Default monitoring frequency used by bandwidth collector in default mode
bandwidtCustomizeFrequency	Customized monitoring frequency used by aggregate collector in customized mode
customizedMornitorTime	Customized monitoring frequency used by migration software defined measurement process for detecting the migration source
checkingFrequency	Migration checking frequency used by migration engine

Table 4.1: **migration manager configuration file**

4.1.3 Migration Daemon

Migration Daemon is the process running on each host which is running the docker container to perform migration process, and enables the communication between migration source and destination host.

Container API is used to create, start, stop a container, as well as take the snapshot of the current container status. when the migration process starts, migration source host will negotiate with the migration destination host, and send the container image files which are needed to resume the container in migration destination host.

TCP/IP is used for communication between hosts by defined migration protocol.

Migration Protocol

1. Migration Manager has the knowledge of the migration source and destination

IP address after making the decision, opens a TCP connection to the source host, send command:
MIGRATE “destination host IP”

2. Each host listens on TCP port 8088, waiting for commands from Migration Manager or another host.
pseudo code:
switch(command):
case “MIGRATE”:
do docker checkpoint command;
Open a TCP connection to the destination host;
Send the files which save the concurrent status of the container;
case “Restart”:
Receive files;
Execute the docker restart command to resume the container;

4.2 Migration Process

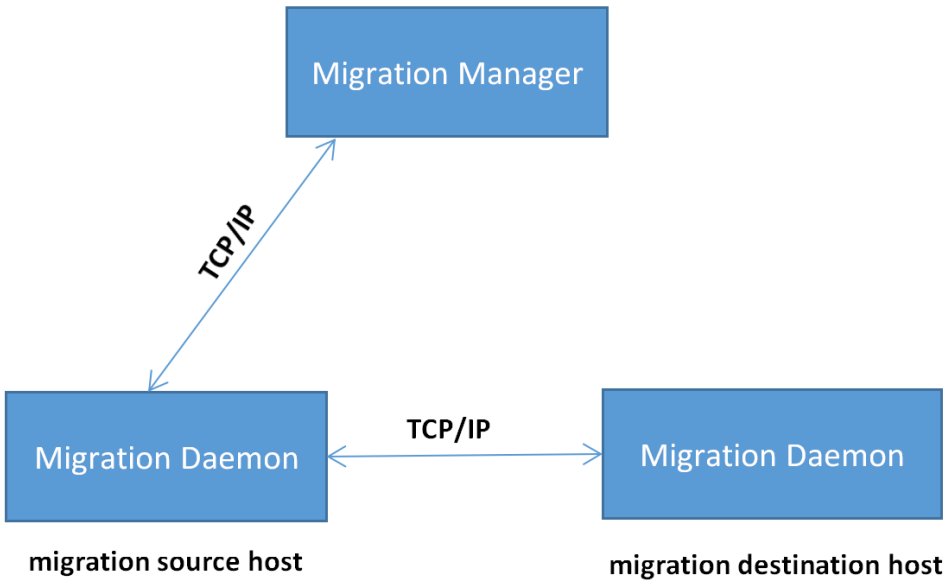

Figure 4.2. migration model: negotiation between migration manager and migration souce host, migration source host and migration destination host

The negotiation model is as 4.2, after migration manager making the migration

decision. It means that the migration source host and migration destination host is known. Then it will first negotiate with migration source host, send the migration decision. After the migration source receives the migration command and migration decision IP address, it will negotiate with migration destination to perform the migration process.

Each block in Figure 4.2 acting the role and perform the task as following:

- Migration Manager makes the migration decision, negotiates with migration source host through TCP/IP for sending migration command.
- Each host is running migration daemon as a migration server.
- Each host can receive the two kinds of connections:
    1. Connection is from migration manager for receiving migration command and migration decision IP address.
    2. Connection is from another host for performing container migration process and receiving the container image files.

Migration Process:

1. Migration Manager opens TCP connection to migration source host after making migration decision.
2. Migration source host checks the command:
    - If the command is "MIGRATE", it does docker checkpoint, then open a TCP connection to migration destination host for transferring container image files.
    - if the command is "RESTART", it prepares to receive the container image files, restarts the container.

The message exchange sequence is shown in Figure 4.3

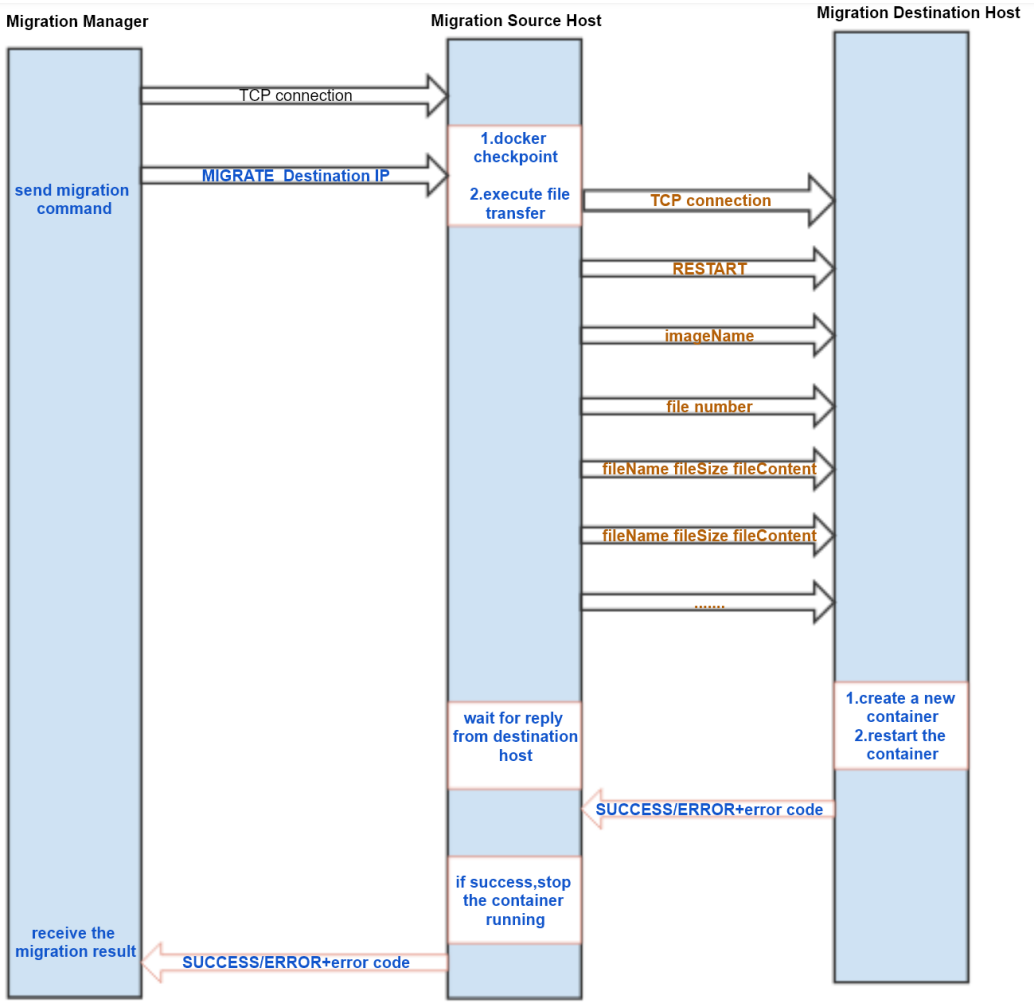

Figure 4.3. migration: message exchange between migraton manager and migration source host, migration source host and migration destination host

Error code:
-1: Migration source hosts execute docker checkpoint Error.
-2: Migration hosts execute file transfer Error.
-3: Connection between migration source host and migration destination host Error.
-4: Migration destination hosts receives files Error.

4.3 Database Design for Software Defined Measurement

In this section, we will show all the relational tables used in the database and the usage for each one.
"Connection"

source	dstType	destiantion
S1-eth1	host	h1-eth0
S1-eth1	server	s2-eth2

Explain:
Store the connection information between switch interface and the host interface in order to know the attached host in each switch port.
The information of the connection table can be got from MININET, GENI or from the network administrator.

"Mapping"

name	value
00:00:00:00:00:00:00:01_1	S1-eth1
h1	10.0.0.1

Explain:
The switch identifier used in floodlight is SwitchDPID number,such as "00:00:00:00:00:00:00:01_1" is the SwitchDPID number "00:00:00:00:00:00:00:01" with the switch interface number "1". However, in network topology, the switch is identified by the switch name. This mapping table is for matching the switchDPID with switch interface name.
Another kind of mapping is for matching the host name with the the IP address. The IP address is used for opening the TCP connection.

"StatisticAggregate"

switchDPID	Time	Flow_count	Packet_count	textbfPCDifference
00:00:00:00:00:00:00:01	2017-09-25 15:41:47	3	5750	734

Explain:
This table can be used as the first input data set for the migration decision algorithms in order to measure the switch aggregate packets number difference in each query time. If the packets aggregate difference changes dramatically, it could have the high possibility that the corresponding switch port experiencing a heavy traffic. This table is used by software defined measurement in standard mode.

"StatisticsBandwidth"

switchDPID	port	Time	bitPerSecondRx	bitsPerSecondTx
00:00:00:00:00:00:00:01	1	2017-09-25 15:41:47	0	126

Explain:
This table is the second input data set for the migration decision algorithm. After it detects the heavy switch, it will check the bandwidth consumption on each interface. The host will be selected as the migration source when the attaching switch interface is detected with the minimum available bandwidth.

"temp_bandwidth_xxxxxxxxxxxxxxxxPx"

id	Time	bitPerSecondRx	bitsPerSecondTx
1	2017-09-25 15:41:47	0	126

Explian:
This table is used for software defined measurement in customized mode. After the heavy switch set is known, it will use a smaller monitoring frequency to check the switch aggregate packets more precisely. The table is created for each switch in the heavy switch set for store the customized mode monitoring. In the end, only one switch will be selected as the migration target switch.

How to construct these tables?

- From MININET/GENI, we can get the connections between host and switch . ("Connection" table)

- From SDN controller, we can get host information and switch information, then do the mapping. Since SDN controller uses the interface number and DPID as the identifier of the switch, MININET/GENI use the switch name and interface name, like s1, eth1, to identify the switch and hosts, we need collect the information on both side to creating the mapping. ("mapping" table)
- From SDN controller northbound REST API, we can acquire the network traffic statistics: bandwidth and aggregate packet number.("StatisticsBandwidth", "StatisticsAggregate", "temp_bandwidth_xxxxxxxxxxxxxxxxPx" table)

Chapter 5

System Implementation

The system is build in Java. In this chapter, we will show the tools and technologies we use to provide some necessary functionalities for the system. Nevertheless, we will discuss the import threads in the system to perform important task.

5.1 Prototype Components

In this section, we will describe the tools and technologies we employed and how they are integrated with the system.

5.1.1 SDN Floodlight controller

Floodlight controller is an open Java-Based SDN Controller. We can get benefits from floodlight controller thanks to the following features:

1. It provides a module loading system, so the needed module can be flexibly loaded.

2. It is Apache-licensed, we can use Floodlight for almost any purpose.

3. It offers the Northbound API, we can get the topology information and network traffic statistics very easily by just sending the REST API request. It will return JSON object which can be interpreted easily.

4. It offers the Southbound API, which can work with physical and virtual switches which speak the OpenFlow protocol. It simplifies the procedure and hinds the complex part of the communication between network and SDN floodlight controller.

We build our system on the top of Floodlight controller, it is the bridge that links our migration system with the physical or virtual network device for network topology prevision, and network traffic statistics message exchange. It is mainly integrated with Software defined measurement bock which shows in 4.1. The network statistics are acquired leveraging the following REST API:

1. Retrive the port list information per switch.
   http://<controller-ip>:8080/wm/core/switch/all/port-desc/json

2. Retrieve aggregate stats across all switches.
   http://<controller-ip>:8080/wm/core/switch/all/aggregate/json

3. Retrieve bandwidth consumption per switch, per port.
   http://<controllerip>:8080/wm/statistics/bandwidth/<switchId>/<portId>/json

4. Get an ordered list of paths from the shortest to the longest path.
   http://<controllerip>:8080/wm/routing/paths/<src-dpid>/<dst-dpid>/<num-paths>/json

Some information will be stored in the database as described in 4.3, some will be used in the algorithms for executing policies.

5.1.2 Open VSwitch

Open vSwitch is a multilayer virtual switch licensed under the open source Apache 2.0 license. It can be used as a software switch. It provides the programmability for massive network automation. Besides, it is compatible with the standard management interface and protocols. Figure 5.1 shows that what can be done with Open VSwitch.

In our system, we deploy the network topology using Open VSwitch. Each switch can be configured with the remote SDN controller. It enables the visible possibility to SDN floodlight controller. It speaks OpenFlow protocol, so it can be integrated with SDN controller though controller southbound API.

Hence, whenever the Open VSwitch is attached to the Floodlight controller. It can be noticed by the controller, and all the topology information and statistic information can be got through the controller REST API.

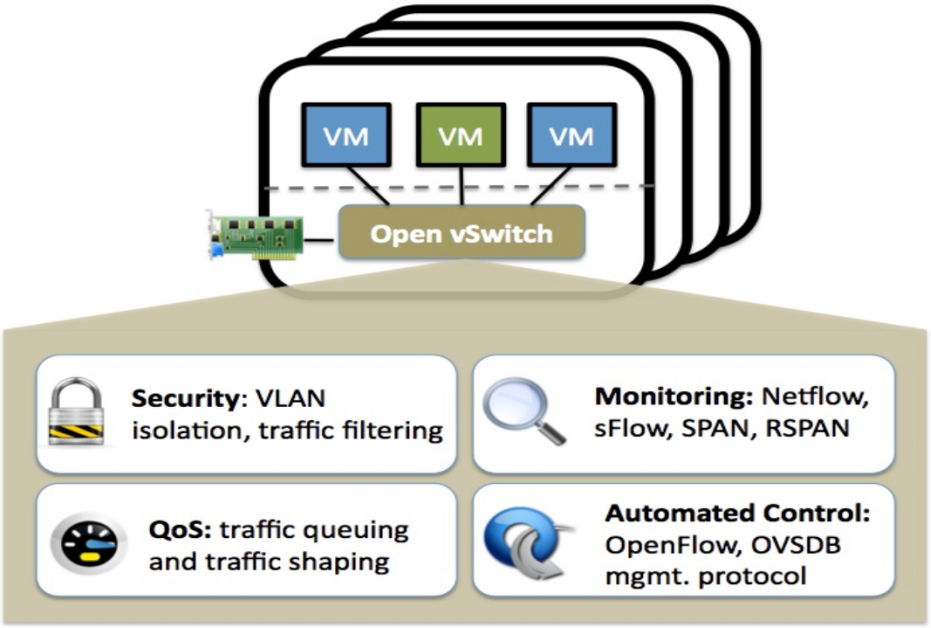

Figure 5.1. Open VSwitch: management interface and protocols that are compatible with Open VSwitch

5.1.3 Criu

It is a software tool for the Linux operating system. A running application can be frozen and checkpointed. The application current status can be stored as a set of files. It providess the possibility to restore the application or container and run it by using the files without losing the current status.
In our system, Criu is integrated with Docker Container API. Thus, we can checkpoint a running Docker container, leave the container running, transfer the stored container image files to another host in order to resume the container. Thus, we can achieve the container live migration.

Docker API Commands employed in our system:

1. Create a new container.
   $ **docker create [OPTIONS] IMAGE [COMMAND] [ARG...]**
   eg. docker create -t -i fedora bash

2. Start a new containe.
   $ **docker start [OPTIONS] CONTAINER [CONTAINER...]**
   eg. docker start myContainer.

3. Create a checkpoint from a running container.
   $ **Docker checkpoint create [OPTIONS] CONTAINER CHECKPOINT**
   eg. docker checkpoint create looper checkpoint1

4. Restart a container using checkpoint files. $ **docker start –checkpoint-dir ContainerRestoreDir –checkpoint=checkpoint NewContainerName**
   eg. docker start –checkpoint chpointDir –checkpoint=checkpoint ubuntu

5.1.4 Database

Here, we create the database using MySQL. It is easy to instill in Linux System. It provide the high level flexibility, scalability, reliability. We can create the relational tables. By using library, It enables to interact with the Database.

5.2 Software Defined Measurement

The goal of software defined measurement is creating a system which can be used to analyze the data from the database in which there are network traffic statistics, such as bandwidth, aggregate packets number. The key point of the software defined measurement is that we can control the monitoring frequency. According to the needs, it is flexible to switch the working modes. One mode is using the standard query frequency, while another mode is using the customized frequency which could be smaller.

Standard frequency enables us to select the preliminary migration target switches with heavy traffic, while customized frequency allows to further monitoring the preliminary migration target switches in a small range with a small frequency. It helps to improve the accuracy to select the migration source host.

This software defined measurement model provide us the flexibility to create our algorithm to make migration source and destination decision.

The process is shown in Figure 5.2.

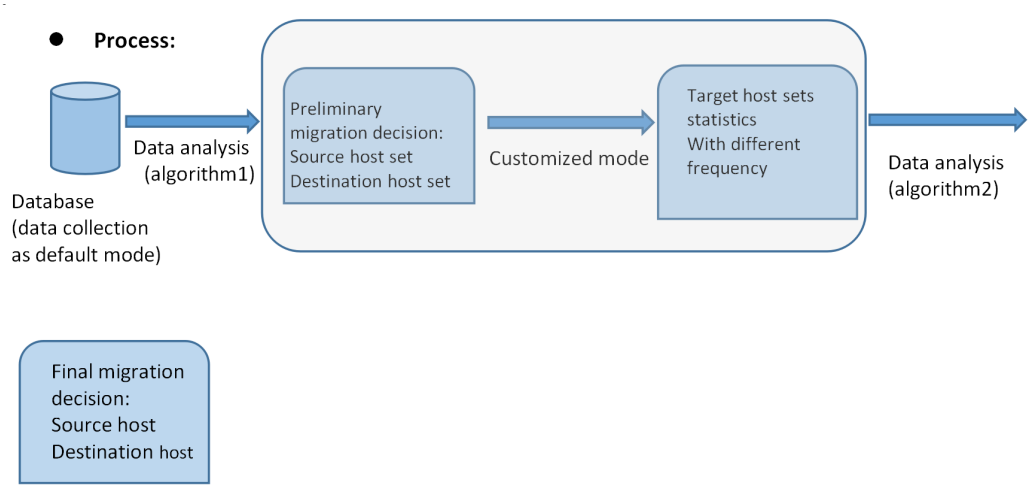

Figure 5.2. System Implementation: the process of software defined measurement system

5.3 Key Threads and Classes

In this section, we will show some essential threads and classes that used in the system.

Aggregate Thread

A thread that collects the packets aggregates statistics and store in table "StatisticAggregate". There are two modes, one is the default mode, it collects all the switches aggregate package number and calculate the package difference according to the frequency specified in the configuration file. In the customized mode, the statistics data is stored in a a temporary tables for each specified switch, and a different frequency is used in order to monitor one switch precisely.

Bandwidth Thread

A thread that collects the bandwidth consumption per switch per port for both send and receive speed. These statistics are stored in table "StatisticsBandwidth". There are two modes, one is default mode as aggregate thread, it collects all the switch ports bandwidth using the default frequency specified in the configuration file. Another mode is customize mode, it receives the port lists, creates a table for each port, monitors it using a smaller frequency in order to analyze the bandwidth consumption more precisely.

Checking Thread

A class that checks the statistics according to different policies specified in the configuration file. For random policy, it will directly run the policy executor. However, for the bandwidth and shortest path policies, it will execute the migration source host decision made algorithm. When it gets the migration source host, it will run the policy executor by passing migration source host as a parameter.

Statistics Client
A class that creates all the statistics threads and also exchanges the messages between checking threads and bandwidth thread, checking thread and aggregate thread.

Policy Executor
A class that executes different algorithms according to different policy used.

HostInformationCollector
A class that collect the host information and link bandwidth from the network topology(Mininet/GENI) or files provide by network administrator, then store them in the database to construct the "connection" and "mapping" table.

systemInitialize
A class that loads the configuration file to the system.

tcpClient
It opens TCP connections to host to send start container and migration command.

ClientWorker
A class that each time migration daemon in the host receive a TCP connection, it will create a new client worker thread to execute the command form migration manager or another host. If it receives “START” command, it will deploy the a new docker container in the host.
If it receives ”MIGRATE” command, it will receive the migration destination IP Address.
If it receives “RESTART” command, this command comes from another host and it will receives the image files to restart the container.
The detailed sequence of software defined measurement with two different working modes, and how it switch from one mode to another one is shown in Figure 5.3.

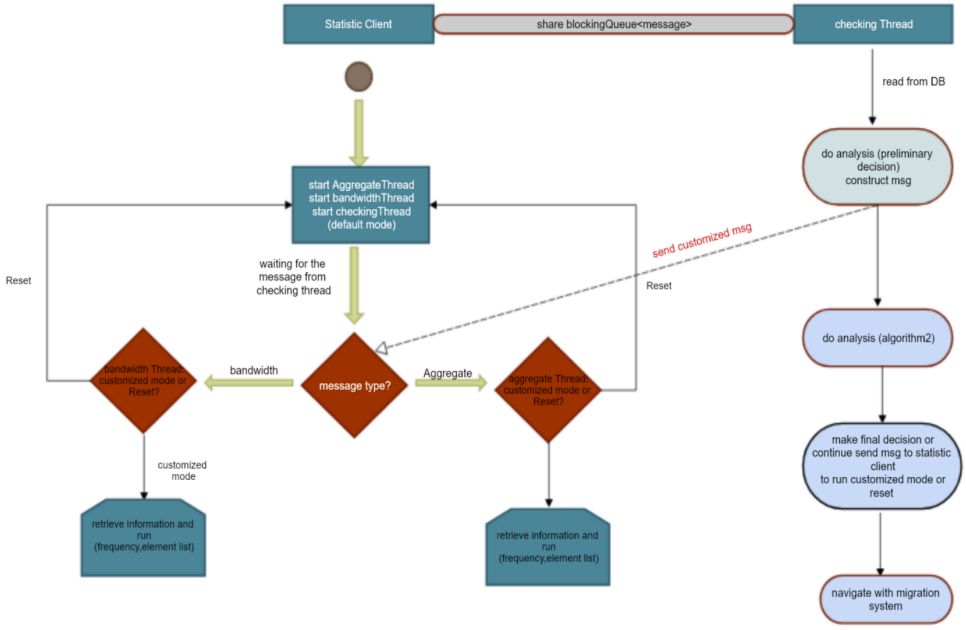

Figure 5.3. System Implementation: the detailed sequence of software defined measurement system

5.4 Decision Made Process

When Statistic Client starts running, first it retrieves the policy specified in the configuration file, if it is random policy, it will start checking thread. If it is bandwidth or shortest path, it will start the aggregate thread, bandwidth thread, checking thread. In the case of random policy, checking thread will directly run policy executor. In case of bandwidth and shortest path policy, checking thread will first make migration source host decision, then run policy executor.

In order to make migration source decision, it retrieves all the switches aggregate package number, and select a set of switches which both the package aggregate number and the package difference between current time and last time exceed the threshold specified the configuration file. After it selects the set of target migration source switches with container running, for example S1, S2. it will enter the second checking phase. In this phase, the aggregate thread enters customized mode, which checking thread create a table for each switch within the moving target switch set and monitor each one in a smaller frequency, it selects the one with the package difference keeping greater than 0, and continuing increase. After aggregate thread phase 1 default mode, and phase 2 customized mode, the checking thread has the target migration switch, then it checks which port is consuming the highest

bandwidth with the minimum available bandwidth. From this port, we can get the migration source host.

Migration Source host Decision:

Algorithm1:
Select the target switch set.
Factor1: aggregate packets number and packets difference.
If the switch is detected by algorithm 1, it means that the switch, at this moment, experiences heavy traffic. And it has the potential possibility to be a heavy switch also in the soon future.

Alogithm2:
Select the target host
Factor2: Attached switch aggregate package number difference great than 0 constantly and the host consumes the highest bandwidth.

After the migration source host decision is made, the policy executor can execute the migration policy.

If it is Random policy, the policy executor randomly select a migration source host from the free hosts, a migration destination from the busy hosts.

If it is bandwidth policy,the policy executor will request path to each free host from floodlight. Then for each path, it calculate the available bandwidth using MaxMin(bandwidth) function, and it select the one with the maximum available bandwidth.

If it is shortest path policy, the policy executor will request the fast path to each free host from the floodlight, then it sorts the latency of all the path. The one with smallest latency will be selected.

5.5 System Implementation prototype

Floodlight controller provides Northbound REST API to migration engine, gets the under layer network topology prevision thought OpenFlow protocol.
Migration engine maintains database, runs software defined measurement and migration manager. It can send statistics request though REST API to floodlight controller, and store the results in database. After migration Manager making the decision, it will send this migration decision to migration source host which is running migration Daemon though migration protocol. After migration source host receives the migration decision, it will start the migration process with migration destination host, send the migration container files.

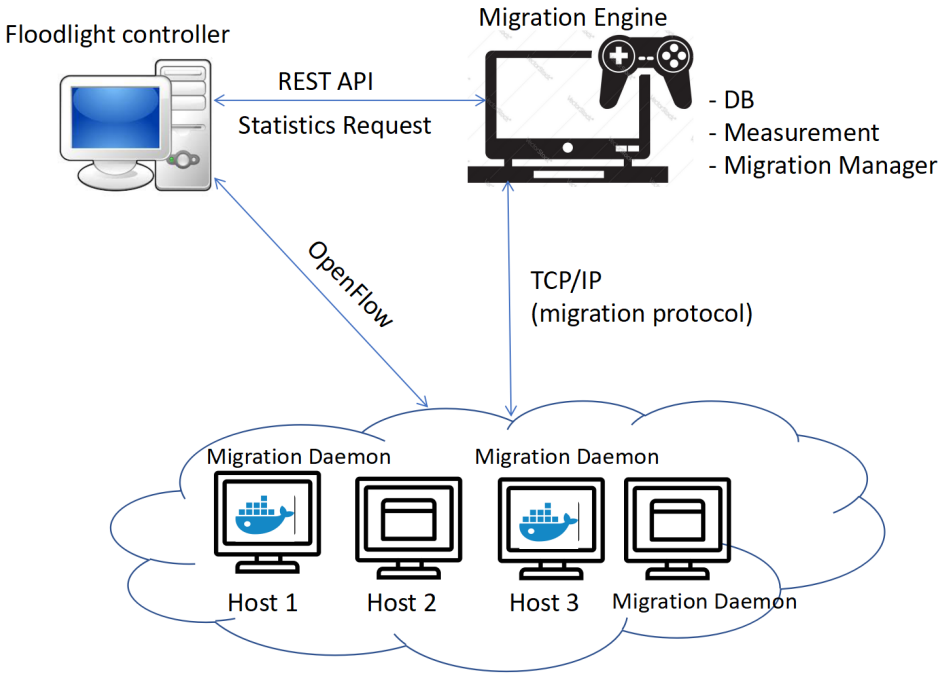

Figure 5.4. System Implementation: System Integration

Chapter 6

Migration Policy Tradeoff and Use Cases

In this chapter, we give two use cases Load Balancing and Moving Target Defense for testing our migration system. The policies used in each use case will be listed and compared.

6.1 Use Case 1: Load Balancing

This application allows migration happen by monitoring the network traffic. Through monitoring the network traffic, migration can be done regarding to the current network traffic status. The migration destination host is selected according to different network measurement criterion.

6.1.1 Policies for Load Balancing

Random: Migration source host and destination host are selected by the system randomly, regardless traffic situation.
Bandwidth: Migration decision is made according to the network traffic situation. In particular, according to the path bandwidth, system select the migration destination host which has the maximum available bandwidth.
Shortest Path: Migration decision is made according to the network traffic situation. In particular, according to the shortest path, the system selects the migration destination host which is the closest one with respect to some matrix, here we use latency.

Policy Implementation
The process is shown as in Figure 6.1

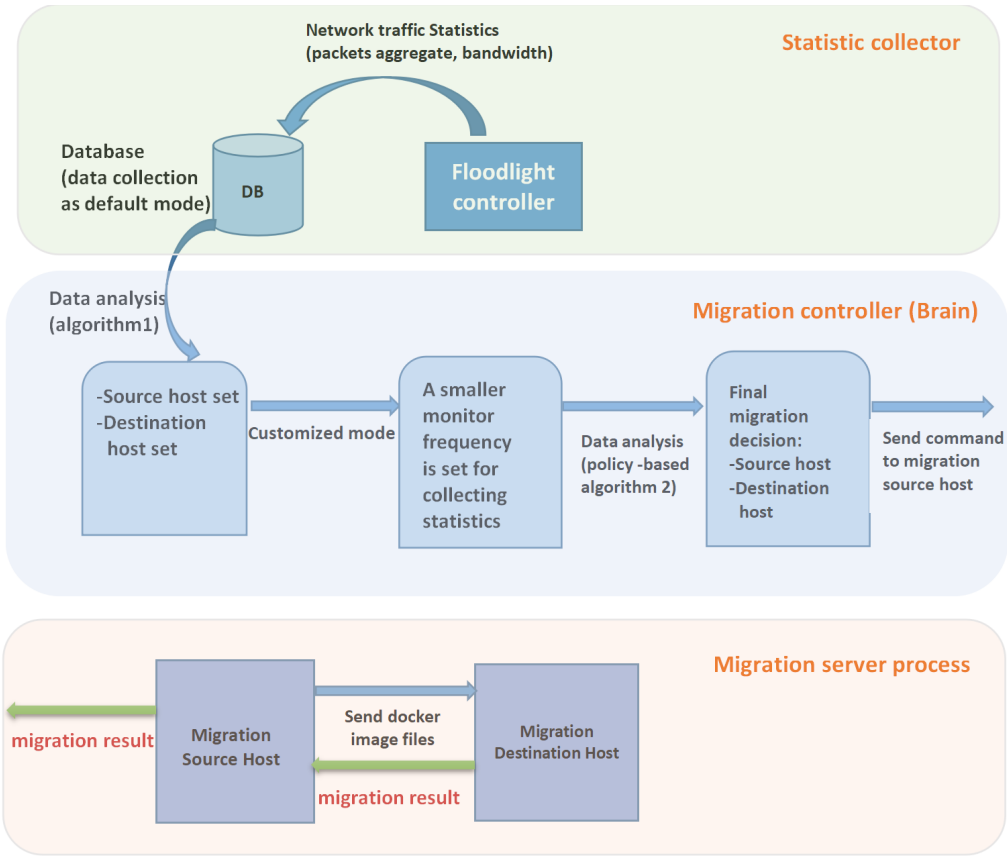

Figure 6.1. Load Balancing policy process: a detailed process for load balancing

Migration Source Host selection(Algorithm 1):

1. check AggregateStatistics table from the database, search the switches which both aggregate packet number and aggregate packet difference number are greater than the threshold. These switches are in the heavy switch set, then it uses customized mode for selecting one switch from the heavy switch set which will be potentially heavy in the future. ——-> heavy switch selected

2. check BandwidthStatistics table from the database, search the host which has the maximum Bandwidth consumption ——> migration source host selected

Migration Destination selection is policy based(Algorithm 2):

1. **Bandwidth-based:** Send REST API requests to floodlight controller to get the paths to each free switch, then use the function Maxmix(Bandwidth) to select the switch which has the maximum available

Path bandwidth. The host which has the maximum available bandwidth is selected as migration destination host.

2. **Shortest Path:** Send REST API requests to floodlight to get the fast path to each free switch, then select one has the fastest path with the smallest latency as the target destination switch. Select the first free host from the target destination switch.

Random Policy
One pseudo random function is used, source host and destination host are selected at random.

6.1.2 Compared 3 policies for Load Balancing

Random

Pros: no computation, less CPU usage, can be used to defence server attack.
no interact with SDN controller.

Cons: high possibility of making bad decision.
-select light host as migration source.
-select heavy host as migration host.
Migration time is not predictable.

Bandwidth-Based

Pros: more compatible with dynamic network traffic situation.
guarantee server can provide better performance after migration.

Cons: more CPU usage(executing algorithm to make migration decision).
migration time is not predictable.

Shortest Path

Pros: more compatible with dynamic network traffic situation
migration travels less network device

Cons: more CPU usage(executing algorithm to make migration decision).
Server performance are not guaranteed.

6.2 Use Case 2: Moving Target Defense

Instead of defending unchanging infrastructure by detecting, preventing, monitoring, tracking, or remediating threats, moving target defense makes the attack surface dynamic. It tries to make the system dynamic and therefore harder to explore and predict. The ultimate goal of moving target defense is to increase the attacker's workload so as to level the cybersecurity playing field for both defenders and attackers.

In the original system, we can use the random policy as moving attacker defense strategy. The system every time generates two random number using pseudo random function for selecting the migration source and destination host. These two numbers are easy to predict if someone studies the random generation function, and monitor the network. Hence, we start to think about updating the system with a more secure way to protect the container service against attack.

Initial System

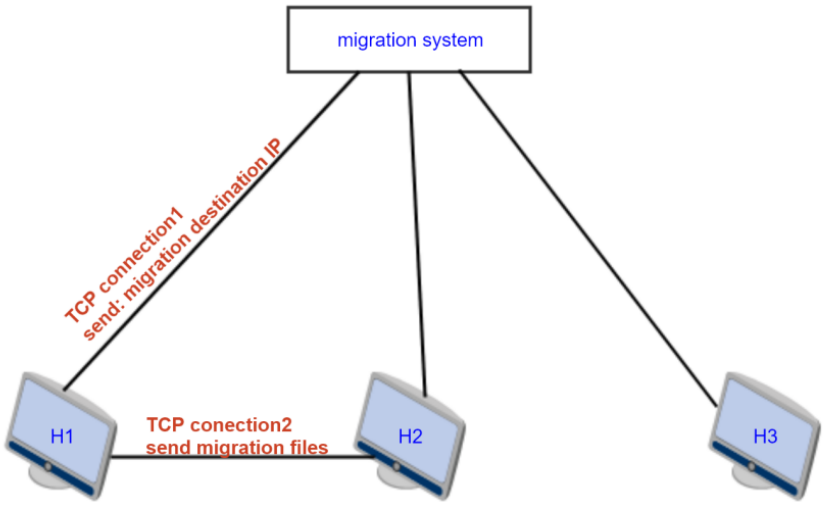

Figure 6.2. Random Policy: one moving target defense strategy

As shown in Figure 6.2, the migration decision is made by the system. After that, it sends the migration destination host IP to migration source host through TCP connection. Here, we propose that part task of making migration decision is delegated to host. Migration system will not send the clear migration destination IP anymore.

6.2.1 Possible Attack

1. Migration destination host is selected by a pseudo random function, which is easily predicted.

2. The attacker could be inside the network, monitor the traffic and listen to the network.
   TCP connection1
   The migration destination IP is sent explicitly.
   TCP connection2
   For small traffic generation network, the attacker could do network traffic analysis. It can determine the destination host if the migration container images files are huge, the throughput from source to destination will increase dramatically.

6.2.2 How to share a secret?

1. Shamir threshold schema a(k,n)
   we adopted the concept of Shamir threshold schema a(k,n) [19]. That is dividing a secret into n parts, distributing each participate an unique part. Any k of the parts are needed to reconstruct the original secrete.
   Here, we choose at random k-1 positive integers as the polynomial multinomial coefficient. we use the polynomial interpolation:

   $$Q(x) = a_0 + a_1 x + ... + a^{k-1} x^{k-1}$$

   Secret is a_0.
   Every participant is given with a point (x,y). Given any subset of k different points (x,y), we can construct the secret a_0.

2. Digital Fountain
   Here each participate still has one unique key pair(x,y), and in order to construct the original secrete, it need k key pairs. But these k key pairs can be non-disjoint. The host will be asked for the key pair according a probability P. P is calculated by the following formula, the key factor is the latency.

   $$P(i,k) = \frac{\frac{1}{latency(i,k)}}{\sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{1}{latency(i,j)}} \tag{6.1}$$

   Suppose there are n hosts in total. Host i is the migration source host, the probability of host k selected depends on the latency from host i to host k, if the latency is smaller, the numerator will be bigger. So the probability will be higher.

6.2.3 New Items and System Initialization for Moving Target Defense

After improve the system security, new model we use is shown in Figure 6.3

New Items

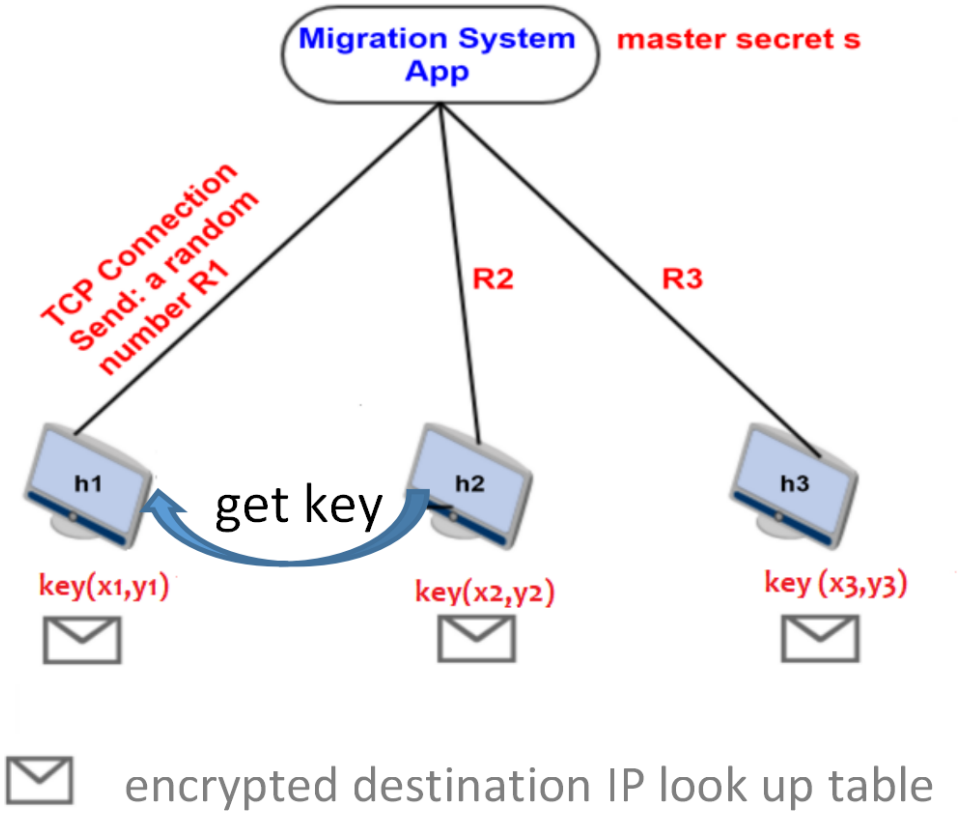

Figure 6.3. Moving target defense model

- **Threshold Schema:** a(k,n)
- **Mater Secrete S:** a random polynomial function only known by migration system.

$$Q(x) = a_0 + a_1 x + ... + a^{k-1} x^{k-1}$$

$$S = a_0$$

- **Random Number R:** a number generated by migration system and send to host to make the migration decision.

- **Key (x,y):** a point on the polynomial function generated and distributed by the migration system, a part of information to construct the Master Secrete.
- **Look Up Table:** A table contains two columns, first column is the index, second column is the host IP Address. It is encrypted using the master secrete S by the migration system.
  The table is shown as following:

Index	IP
1	10.0.0.1
2	10.0.0.2

System Initialization for Moving Target Defense

1. Configure Polynomial Function, the parameter of K which stands for the number of keys will participate in the decryption process, and N can be the total host number.
2. Hash function hash(x)=Hash(R1+X1*Y1)%(N+1) (N is the total hosts number).
3. Encrypt look up table using master key A

6.2.4 Compared 3 policies for Moving Target Defense

Random
Migration Source and Destination host are selected at random.

Pros: very fast

Cons: attacker still can predict

Shamir
To obtain migration destination host IP address, migration source host needs to ask k disjoint hosts for k different key pairs.

Pros: highest safety level

Cons: performance will be slow
more operation for system configuration
more network traffic overhead will be created

Digital Fountain
-To obtain destination need to ask k (non-disjoint) hosts.
-k hosts are selected according to a probability P.

Pros: Safety level is between Shamir and Random.

Cons: more operation for system configuration
more network traffic overhead will be created

6.2.5 Moving Target Defense Process

The process sequence diagram is shown in Figure 6.4.

1. Migration System distributes different encrypted look up table with the master secret to each host.
2. Migration system generates a set of key pairs (x,y) and distributes to each host. Hence each host has a part of the information to decrypt the look up table.
3. Migration system chooses an arbitrary host, opens TCP connection(TCP connection1 in the figure), send a random number to the host(Host M).
4. Host M applies hash(x), the result is the migration destination host index i.
5. According to the configuration file, Host M uses corresponding policy to decrypt the table:

   Digital Fountain: Host M opens random k TCP connections to other k hosts for asking the key pair in order to decrypt the look up table. The hosts will be asked may be not disjoint. The one which has the shorter path will have the higher probability to be chosen. It also could be chosen for multiple times.

   Shamir: Host M randomly opens k TCP connections to other k disjoint hosts for asking the key pair in order to decrypt the look up table. Each host has the equal probability to be chosen.
6. After getting the k key pairs, the migration source host applies the algorithm(Digital Fountain or Shamir) to get the master secrete S.
7. Migration source host H1 decrypt the look up table using S, get the migration destination host IP.
8. Migration source host H1 start migration process with migration destination host H2.

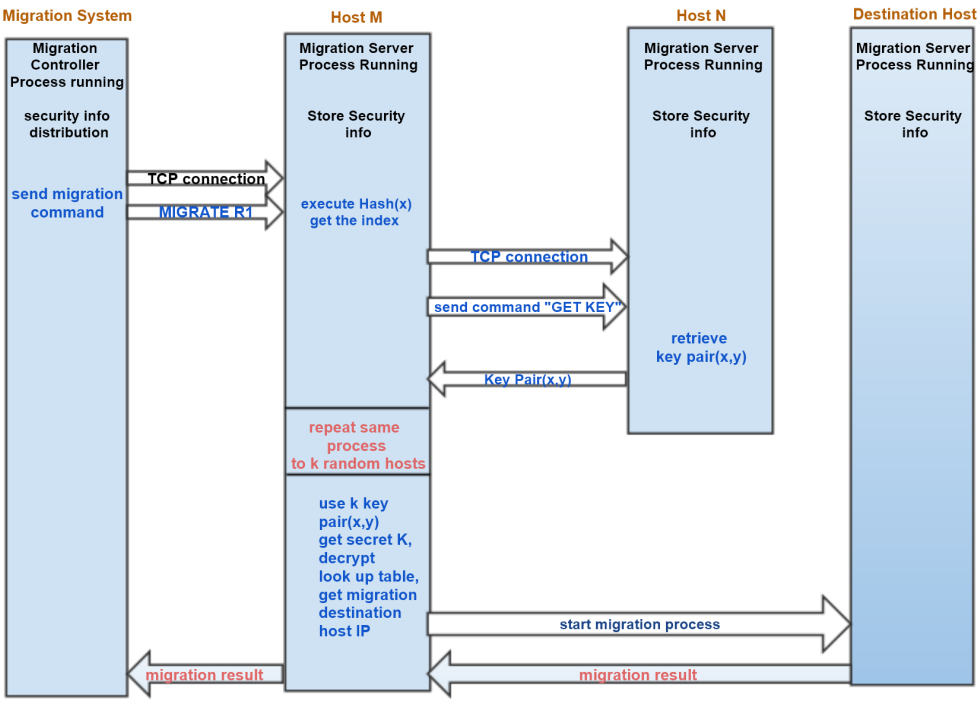

Figure 6.4. Sequence Diagram for Moving Target Defense

Why look up table should be encrypted?

1. The migration system distributes the table while the host is setting up, the attacker could do man-in-the-middle attack to get the table of each host, and ask for the key pairs from each host to get the secrete.

2. Even the docker itself is sage, but it can not be sure that other docker container in the same host is not a attacker to steal this table.

Why when migration source host asks key pair from other hosts, don't need to do authentication?

1. If the attack is after the loop up table distributed, it does not have the cypher look up table, so even it get the k key pairs, it still can not decrypt.

2. Here, each host has different look up table, even the attacker asks k key pairs from other hosts, it just can decrypt the one table, it does not know the migration destination other host will go.

Chapter 7

Experimental Validation

In this chapter, we test our system within software defined network. Compared for each use case different policies tradeoff and performance.

7.1 Test Environment

7.1.1 Hardware and OS

Processor: Intel i7-6500U CPU 2.50GHZ

RAM: 8.00 GB

System Type: 64-bits operation system, x64-based processor

OS: windows 10

System is running in Linux environment, host ubuntu operating system is:

Name: ubuntu

Operating System: Ubuntu(64-bit)

Base Memory: 6096 MB

7.1.2 Network topology deployment

Network topology is deployed within MININET. which is a network simulator. We can create realistic virtual network using Mininet, runing real kernel, switch, and application code on a single machine, such as VM, Cloud.

7.2 Use Case1: 3 policies evaluation for Load Balancing

7.2.1 Scenario 1: link capacity is heterogeneous

Network Topology

The topology is shown in Figure 7.1, H1 runs iperf client,H1 and H2 runs a con-

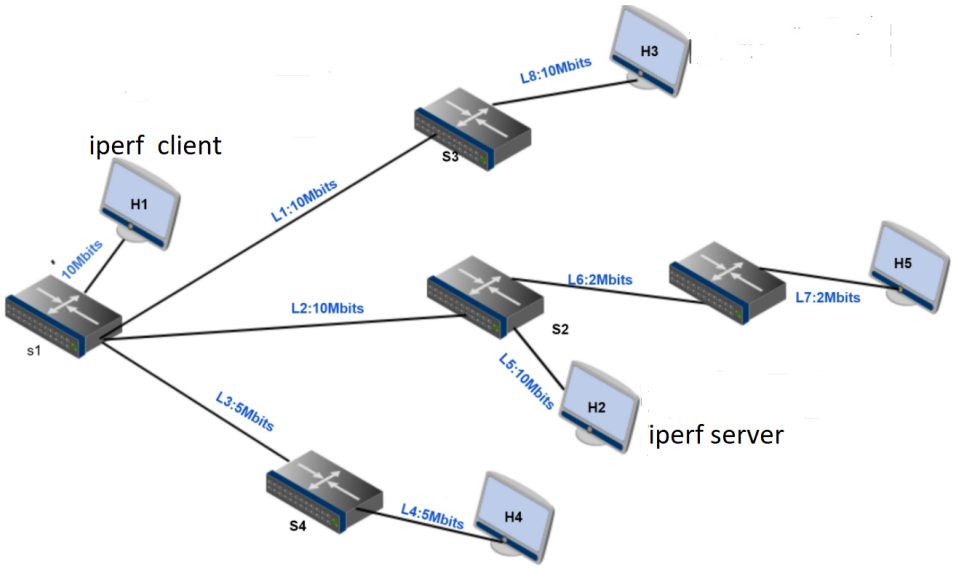

Figure 7.1. Network topology for evaluation link capacity heterogeneous

tainer hosting the iperf server. H1 sends request to H2, traffic is generated from H1 to H2. The link capacity varies a lot. In this case bandwidth-based policy and shortest path make different decision.

3 policies analysis

1. **Bandwidth-based policy**

   Migration Analysis: using Max(min(bandwidth)) select one has the highest available path bandwidth.

H2 is detected as heavy host(migration source host) H2 -> H3, path bandwidth=10
H2 -> H4, path bandwidth=5
H2 -> H5, path bandwidth=2
So H3 is selected as migration destination host.

Migration Decision: H2->H3

2. **Shortest path policy**

   Migration Analysis: using floodlight REST API query the fastest path,according to the path latency
   H2 is detected as heavy host(migration source host)
   H2 -> H3 , path H2->S2->S1->S3->H3
   H2 -> H4 , path h2->S2->S1->S4->H4
   H2 -> H5 , path h2->S2->S5->H5
   So H5 will be considered as the shortest path with least latency

   Migration Decision: H2->H5

3. **Random Policy**

   Migration Analysis: randomly select migration source host from busy host set, randomly select migration destination host from free host set.
   busy host set (H1, H2)
   Free host set (H3,H4,H5)

   Migration Decision: H2 -> H3 , path H2->S2->S1->S3->H3
   H2 -> H4 , path h2->S2->S1->S4->H4
   H2 -> H5 , path h2->S2->S5->H5

Migration source host bandwidth consumption change

How do we do the plot?

1. For bandwidth consumption plot, the value is the sum of send and receive bandwidth consumption, it starts before migration occurs. It experiences migration process, a period after migration done. The plot clearly showed the bandwidth consumption change of the migration source host before, during, and after migration.

2. For throughput accumulation plot, the value is the number of bytes transfer along with the time. It showed that the total traffic generated during the migration process measured during the migration source host.

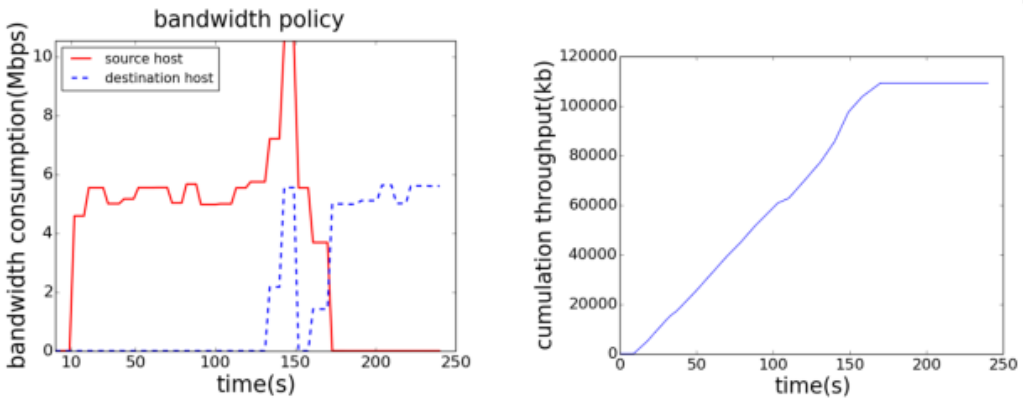

Figure 7.2. bandwidth-based policy left:Migration Source Host Bandwidth Consumption Change right:throughput accumulation

From Figure 7.2 left picture bandwidth consumption, it can prove that migration happens. we can characterize in 3 periods:

First period, the traffic data is collected before migration process from the system starts running, so for the red line(source host) initially it just running the docker container which acts as the iperf server, it receives and send data to the iperf client.

Second period, as the time going after the first period, the iperf traffic is increasing. Then the switch is detected as a heavy switch and the host is selected to be the migration source host. After, It starts the migration process. During this period, the source host not only generates traffic with the iperf client, but also has the traffic to the migration destination host for Docker container image files transfer. As for the blue line(destination host), it starts to receive the migration files, so bandwidth consumption is starting increase during this interval. Then after migration process is done, it enters the third period.

Third period, source host(red line), does not run the iperf server anymore, so there is no traffic (since here we only have iperf server, no other server). Instead of destination host(blue line), it starts to run the iperf server after migration, it will receive the connection from the iperf client. So the bandwidth consumption for destination host is increasing continuously.

Thoughput accumulation comparison
How do we do the plot?

The traffic is collected from migration source host before migration process starting. It stops when the migration is done. Here the host only runs iperf server, so when the server is migrated from the host, there is no traffic anymore, the cumulation throughput enters a stable stage.
It is shown in Figure 7.3.

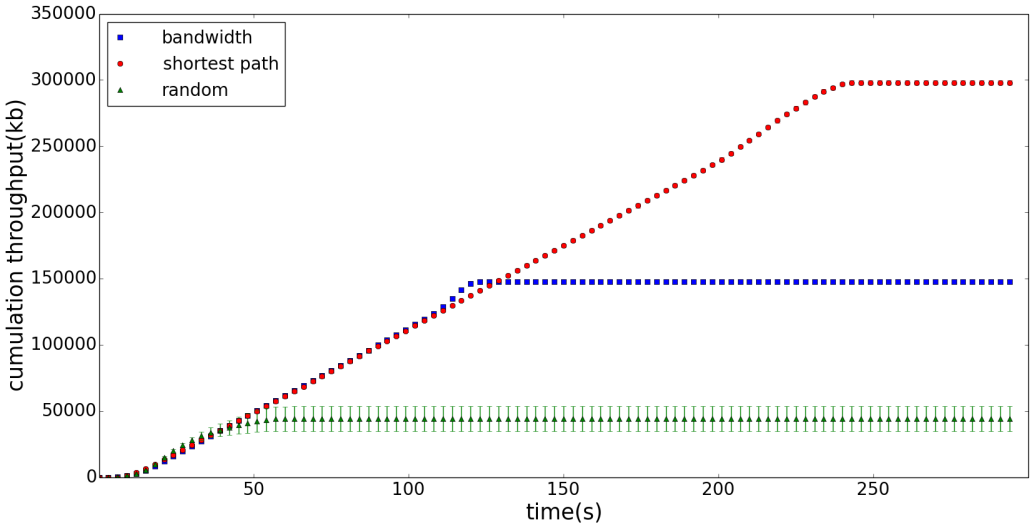

Figure 7.3. Traffic throughput accumulation in Migration Source host

Vertical comparison:
It shows that in the first 50s, the throughput accumulation of random policy is increasing faster than bandwidth and shortest path policy. It implies that system starts migration process earlier when using random policy than bandwidth and shortest path policy. When random migration occurs, the network situation is not congestion, and the migration source host doesn't suffer a high load of traffic. After 50s, the random policy already finished the migration process, the accumulation doesn't increase anymore. Instead of bandwidth and shortest path, it continues increasing, this means the network traffic is heavier and heavier until when it is detected, and migration is done.Both bandwidth and shortest path policy have a larger number accumulation throughput , it means that when migration occurs, the migration source host is suffering heavy traffic.
Horizon comparison:
Here we compare the saturation point, which means the migration process complete time. It can be seen clearly that random policy arrives much earlier than bandwidth and shortest path policy at about 50s. Then bandwidth policy finishes the migration process at about 120s, which is earlier than shortest path.
Random < bandwidth < shortest path
When the network has high requirement of latency, for example, migration must be done within 50s, it is better to choose random policy. If we want to choose one policy considering the tradeoff of migration finish time and load balancing. It is better to choose bandwidth policy in this case.
As we can see, when the link bandwidth is heterogeneous, the bandwidth policy performance is better than shortest path.
Accumulation confidence Interval:

The migration decision made by bandwidth and shortest path is determinate. For bandwidth policy is H2->H3, for shortest path policy is H2->H5. So each run the value is very similar. As can be seen in figure5, the error bar is very small nearly invisible. Instead of Random, the migration destination is not determinate, so each run, it may choose different migration decision, the accumulation value at each time point is different. Hence, the error bar is bigger.

Migration time comparison
We run 10 times, collect the migration time of 3 policies as shown in table 7.1.

	bandwidth	shortest path	random
1	9s	40s	9s
2	9s	40s	9s
3	9s	40s	9s
4	9s	40s	9s
5	9s	40s	16s
6	9s	40s	17s
7	9s	40s	40s
8	9s	40s	40s
9	9s	40s	40s
10	10s	41s	40s
average	(9.1 ±0.15)s	(40.1 ±0.15) s	(23.6 ±7.12) s

Table 7.1. migration time of 3 policies for Scenario 1

This table implies that if the link bandwidth change, it affects the migration time greatly for different policies. Here bandwidth policy takes the advantage of the bigger link bandwidth, so the migration time is much smaller than shortest path policy, random policy. For the confidence interval, random policy varies a lot. Because different migration destination path experiences different path available bandwidth.

Combine all the features, if the network topology link capacity is heterogeneous. From different requirement aspects, if the user wants the smaller migration finish time, the random policy may be considered. If user wants do network load balancing and balance the migration time, bandwidth policy is considered.

7.2.2 Scenario 2: Link capacity bottleneck presence

Network Topology
The topology is shown in Figure 7.4, it is the same as in scenario 1, but we change the value of link capacity. H1 runs iperf client, H2 runs a container hosting the iperf server. H1 sends request to H2, traffic is generated from H1 to H2.

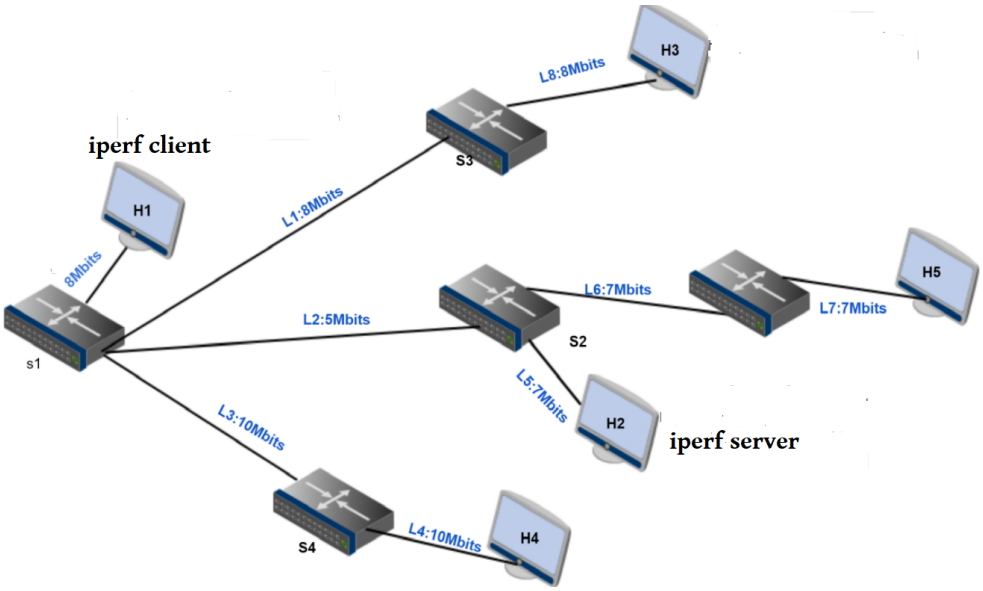

Figure 7.4. Network topology for evaluation link bandwidth bottleneck presence

L2 is the bottleneck link in the path bandwidth from H2 to H3, and from H2 to H4.

3 policies analysis

1. **Bandwidth-based policy**

   Migration Analysis: using Max(min(bandwidth)) select one has the highest available path bandwidth.
   H2 is detected as heavy host(migration source host) H2 -> H3, path bandwidth=5
   H2 -> H4, path bandwidth=5
   H2 -> H5, path bandwidth=7
   So H5 is selected as migration destination host.

   Migration Decision: H2->H5

2. **Shortest path policy**

   Migration Analysis: same as in scenario 1

   Migration Decision: H2->H5

3. **Random Policy**

   Migration Analysis: same as in scenario 1

Migration Decision: H2 -> H3 , path H2->S2->S1->S3->H3
H2 -> H4 , path h2->S2->S1->S4->H4
H2 -> H5 , path h2->S2->S5->H5

Thoughput accumulation comparison

How do we do the plot?

The traffic is collected from migration source host before migration process starting. It stops when the migration is done. Here the host only runs iperf server, so when the server is migrated from the host, there is more traffic anymore, the cumulation throughput enters a stable stage.
It is shown in Figure 7.5.

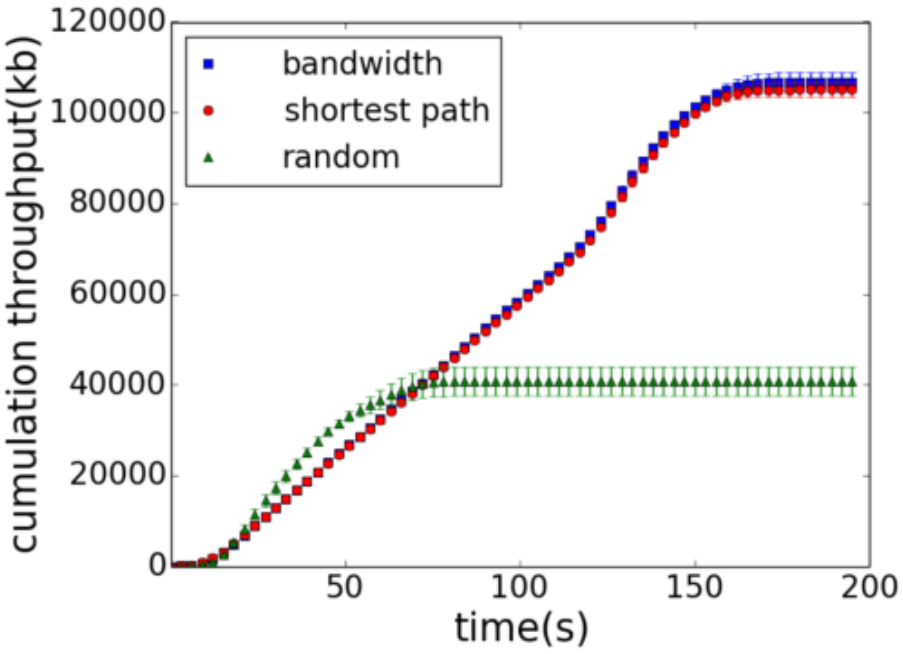

Figure 7.5. Traffic throughput accumulation in Migration Source host

Vertical comparison:
The same conclusion as in scenario 1.
Horizon comparison:
Here we compare the saturation point, which means the migration process complete time. It can be seen clearly that random policy arrives much earlier than bandwidth and shortest path policy. When the network has high requirement of latency, for

example, migration must be done in 50s, it is better to choose random policy. As we can see, when the migration decision made by shortest and bandwidth policy is the same, their performance are compatible with each other.

Accumulation confidence Interval:

The migration decision made by bandwidth and shortest path is determinate. For bandwidth policy is H2->H3, for shortest path is H2->H3. So each run in the same network condition the throughput accumulation value varies lightly. As can be seen in Figure 7.5 , the error bar is very small. Instead of random policy, the migration destination is not determinate, so each run, it may choose different migration decision, the accumulation value at each time point is different. Hence, the error bar is bigger comparing with bandwidth and shortest path policy.

Migration time comparison

We run 20 times, collect the migration time of 3 policies as shown in table 7.2.

	bandwidth	shortest path	random
1	12s	12s	30s
2	12s	37s	29s
3	12s	13s	12s
4	14s	17s	33s
5	15s	17s	34s
6	15s	28s	12s
7	15s	17s	33s
8	17s	28s	37s
9	37s	19s	12s
10	18s	21s	35s
11	33s	19s	35s
12	18s	20s	12s
13	31s	19s	31s
14	18s	20s	31s
15	29s	19s	12s
16	19s	20s	38s
17	27s	20s	38s
18	24s	20s	12s
19	25s	20s	35s
20	25s	20s	12s
average	(20.8 ±2.7)s	(20.3 ±1.9) s	(26.15 ±3.91) s

Table 7.2: migration time of 3 policies for Scenario 2

For the random policy, since the decision is non-determinate. We collect 20 times for each decision, compared the migration time.

	H2-H3	H2-H4	H2-H5
1	30s	29s	12s
2	33s	34s	12s
3	37s	37s	12s
4	31s	29s	12s
5	35s	35s	12s
6	38s	38s	12s
7	31s	32s	12s
8	35s	35s	12s
9	38s	38s	12s
10	31s	32s	12s
11	35s	36s	12s
12	43s	39s	12s
13	32s	33s	12s
14	35s	36s	12s
15	32s	34s	13s
16	36s	36s	13s
17	32s	34s	13s
18	37s	36s	13s
19	33s	34s	13s
20	37s	37s	13s
average	(34.6 ±1.2) s	(34.7 ±1.0) s	(12.3 ±0.2) s

Table 7.3: random policy migration time

From table 7.2 , it can be seen that the migration time between bandwidth, shortest path, random, from the same source host H2 to the same destination host H5, concerning the average migration time, random has the smallest migration time. Shortest path, and bandwidth they are almost the same.
The story behind: For bandwidth and shortest path policy, the main goal is load balancing, the system first detects the heavy switch, then apply the policy algorithm, so in this case, the available bandwidth for migration will be much smaller, this results in larger migration time. Instead of random, it doesn't concern the load of each switch, just simply choose the source and destination. Most of the case,

before the switch getting heavy, it starts the migration process, so the available bandwidth is larger, this results in smaller migration time.

From table 7.3 , it can be seen that the migration time various under policy random while migrating to a different host. It proves that L2 is the bottleneck link with small bandwidth. When migration to h3 or h4, it always needs larger migration time. It implies that random policy has uncertainty, this feature can be used to do simple attack defense. But in terms of the migration performance, it can not guarantee the best migration destination.

Combine 7.2 and 7.3, we can conclude that shortest path and bandwidth can do load balancing. After migration, it can guarantee the better service performance. But under switch overloading, migration process takes longer time. The random policy is good for attack defense, but it concerns less the traffic situation. It does migration even when the source switch is not overloaded. Or it may migrate to a host with the attaching switch is heavy.

7.2.3 Scenario 3: Link capacity is heterogeneous

Network Topology

The topology is shown in Figure 7.6, it is the same as in scenario 1,scenario 2, but

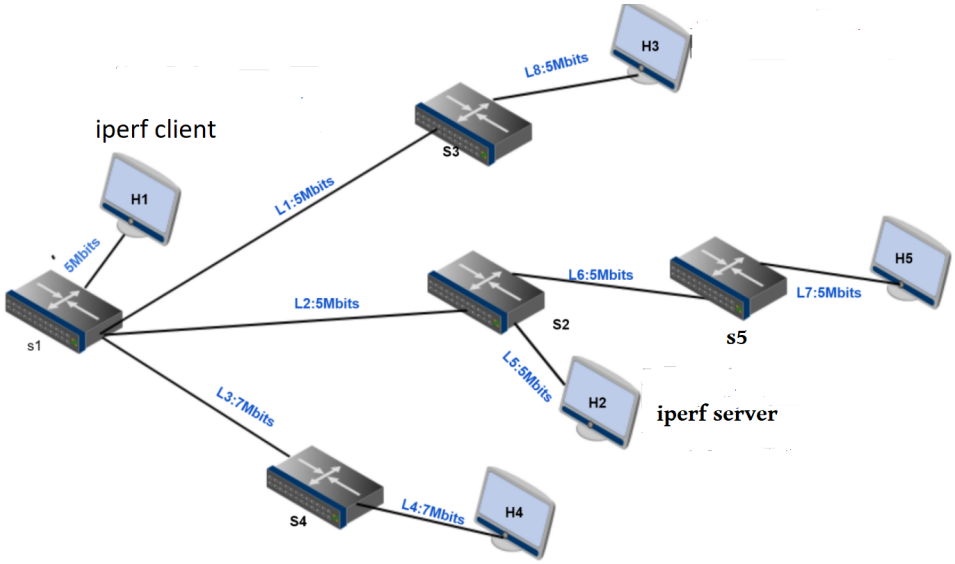

Figure 7.6. Network topology for evaluation link capacity is homogeneous

we change the value of link capacity.The link capacity is heterogeneous. H1 runs iperf client, H2 runs a container hosting the iperf server service. H1 sends request to H2, traffic is generated from H1 to H2. In this case, bandwidth-based decision

is non-determinate, it is the same with random policy.

3 policies analysis

1. **Bandwidth-based policy**

   Migration Analysis: using Max(min(bandwidth)) select one has the highest available path bandwidth.
   H2 is detected as heavy host(migration source host) H2 -> H3, path bandwidth=5
   H2 -> H4, path bandwidth=5
   H2 -> H5, path bandwidth=5
   So H3,H4,H5 may be selected as migration destination host.

   Migration Decision: H2->H5, or H2->h4, or H2->H5.

2. **Shortest path policy**

   Migration Analysis: same as in scenario 1,scenario 2

   Migration Decision: H2->H5

3. **Random Policy**

   Migration Analysis: same as in scenario 1,scenario 2

   Migration Decision: H2 -> H3 , path H2->S2->S1->S3->H3
   H2 -> H4 , path h2->S2->S1->S4->H4
   H2 -> H5 , path h2->S2->S5->H5

Thoughput accumulation comparison

Vertical comparison:
The same conclusion as in scenario 1.
Horizon comparison:
Here we compare the saturation point, which means the migration finishing time.
Random < Shortest Path < bandwidth
When the network has high requirement of latency, for example, migration must be done within 50s, it is better to choose random policy. If we want to choose one policy considering the tradeoff of migration finish time and network load balancing. It is better to choose bandwidth policy or shortest path policy.
As we can see, when the link bandwidth is evenly distributed, probably there are multiple migration destinations when using bandwidth policy. If shortest path policy chooses the destination has the equivalent bandwidth with bandwidth based policy, shortest path policy is better than bandwidth-based policy.
Accumulation confidence Interval:

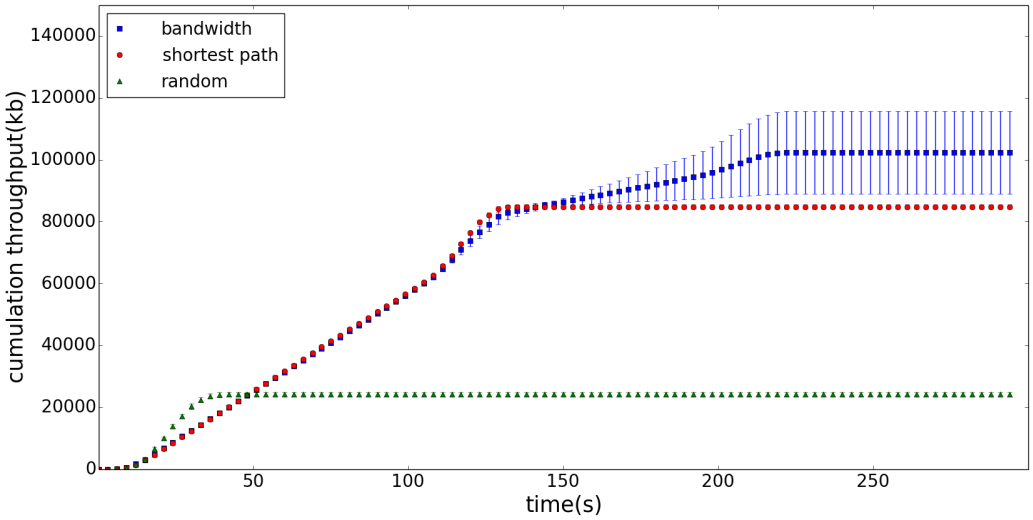

Figure 7.7. Traffic throughput accumulation in Migration Source host for scenario 3

The migration decision made by bandwidth is not determinate. For bandwidth,the decision could be H2->H3, H2->H4, H2->H5. So when the migration decision will be different, during the migration process, the network traffic condition will also be different, in the figure, we can see that the error bar for bandwidth is bigger than shortest path and random.

Migration time comparison

We run 10 times, collect the migration time of 3 policies as shown in table 7.4.

	bandwidth	**shortest path**	**random**
1	17s	17s	17s
2	17s	17s	17s
3	17s	17s	17s
4	20s	17s	22s
5	21s	17s	17s
6	18s	17s	18s
7	18s	17s	19s
8	19s	17s	18s
9	17s	17s	21s
10	22s	18s	20s
average	(18.6 ±0.91)s	(17.1 ±0.16) s	(18.6±0.91) s

Table 7.4: migration time of 3 policies for Scenario 3

Table 7.4 implies that when the link bandwidth is homogeneous in the network with a smaller value. It causes bandwidth policy has multiple choices. So the migration destination is not determinate, and it causes the migration time is also different when concerning the load balancing. And in this case, shortest path, and random, they use compatible migration time.

Combine all the features, if the network topology link bandwidth is homogeneous, shortest path performs better than bandwidth. From different requirement aspects, if the user wants the sooner migration finish time, random policy is considered. If user wants do network load balancing and minimize the migration time, shortest path policy is considered.

7.3 Use Case2: 3 policies evaluation for Moving Target Defense

We evaluated how much should we pay in order to do system security. We evaluate how much should we pay in order to do system security. Here we plot accumulation throughput to evaluate how much network overhead will it be with reference to random policy. Besides, we also collect and compare the migration time.

	Digital Fountain	**Shamir**	**Random**
1	40s	45s	12s
2	44s	45s	12s
3	44s	46s	12s
4	44s	54s	12s
5	45s	55s	13s
6	51s	58s	13s
7	52s	71s	29s
8	38s	44s	29s
9	43s	44s	30s
10	44s	45s	31s
11	48s	51s	31s
12	50s	55s	32s
13	53s	56s	33s
14	41s	60s	34s
15	15s	15s	35s
16	16s	15s	35s
17	16s	15s	37s
18	18s	16s	37s
19	15s	17s	38s

20	15s	14s	38s
average	(36.6 ±5.2) s	(40.1 ±6.6) s s	(27.2 +3.7) s

Table 7.5: migration time of 3 policies for Moving Target Defense

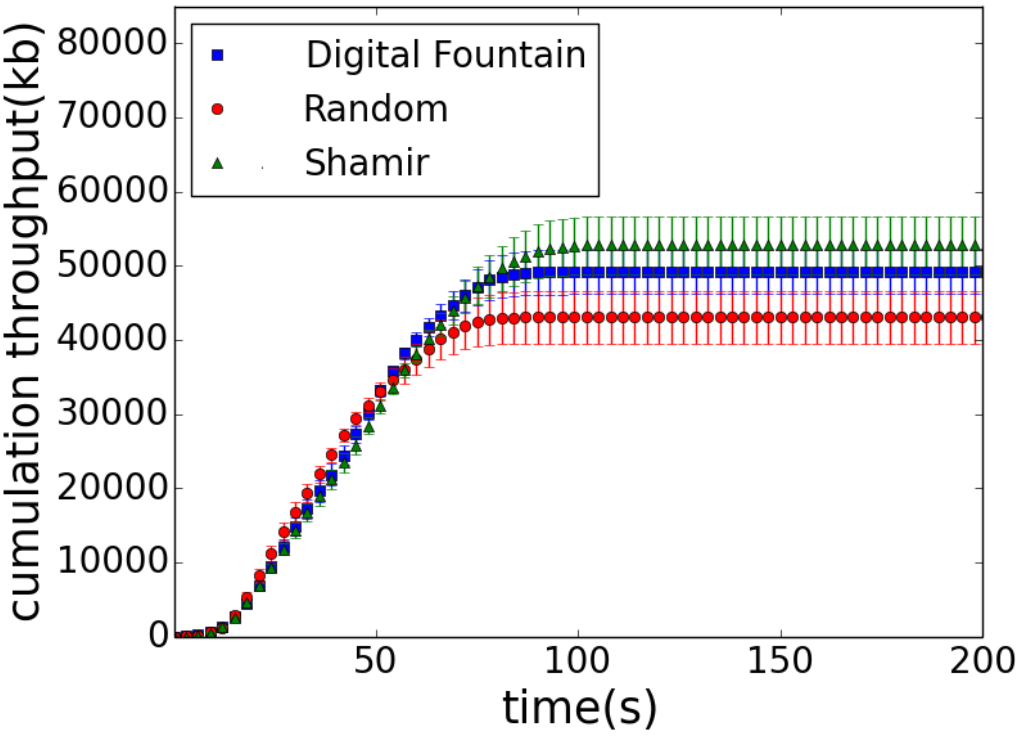

Figure 7.8. accumulation throughput comparison for 3 policies

From Figure 7.8 and Table 7.5 we can get the messages are that since Shmir policy asks for k disjoint hosts for k different keys, so it may choose some very far and high latency host, In this case, the network overhead will be higher, and migration time will be bigger. Random policy applies a pseudo function by the migration system to choose one migration destination host in the free host list. It is the fast one and has less network traffic overhead. Digital Fountain policy asks for k non-disjoint hosts for k keys. It probably asks the host with small latency more times. In this case, it will create less network traffic, and also the migration time is smaller.
In conclusion, Digital fountain has the better speed-security tradeoff.

7.3.1 Experiment in GENI

Apart from using Mininet, we also deploy our topology in GENI to evaluate whether the behavior and policies performance are the same or not.

Environment difference between GENI and MININET:
In MININET, when we specify the link bandwidth, we can use approximately the same link capacity. However, in GENI, since the physical resource is shared by us and other users, when we reserve the link bandwidth, for example, we set a link bandwidth with 10Mbps when we deploying the network, it can not guarantee that in any moment we can use 10Mbps. But the maximum capacity is 10 Mbps, sometimes it is 10Mpbps, 8Mbps, sometimes it is 6 Mbps or 7 Mbps. We tested the link bandwidth using Iperf. So, we can see from the experiment result that the average migration time is bigger.

We evaluated the same topology as we used in Mininet to compare that whether the policy performance in GENI is compatible with it in Mininet or not. I found that the policy performance conclusion is almost the same, but in GENI, the migration time, and throughput varies a lot because of the available link bandwidth varies a lot.

For each topology with different link capacity, I run 10 times, computed the average migration time.

Load Balancing Policies:link capacity is heterogeneous

The link capacity is shown as in Figure 7.1, Compared 3 policies performance: Bandwidth-based policy provides better performance(it has less network overhead) Here accumulation throughput represents the network overhead before,during and after migration.We can notice that random has a bigger error bar because every time the choice could be different, also link capacity varies a lot. Shortest Path is pretty slow, even worse than random policy.

bandwidth	shortest path	random
(39.0± 8.21)s	(259.7±49.41)s	(139.5 ±60.79) s

Table 7.6. migration time of 3 policies for Scenario 1:link capacity is heterogeneous(GENI)

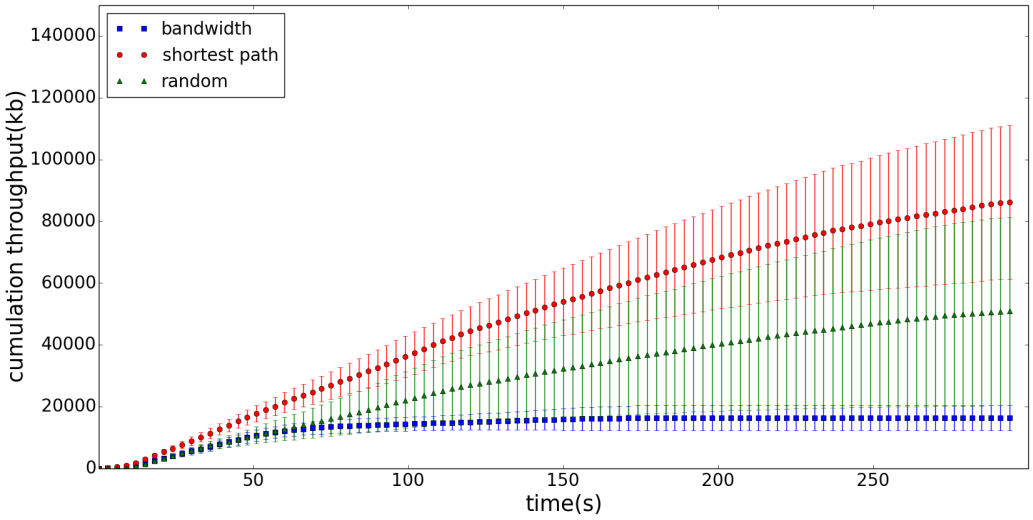

Figure 7.9. Bandwidth accumulation throughput for Scenario 1: link capacity heterogeneous(GENI)

Load Balancing Policies:link capacity is homogeneous

The link capacity is shown as in Figure 7.6, Compared 3 policies performance: Shortest path policy provides better performance(it has less network overhead) Bandwidth-based and random have compatible performance. Here accumulation throughput represents the network overhead before,during and after migration.

bandwidth	shortest path	random
(97.7± 41.27)s	(63.4±17.08)s	(96.7 ±29.90)s

Table 7.7. migration time of 3 policies for Scenario 3:link capacity homogeneous(GENI)

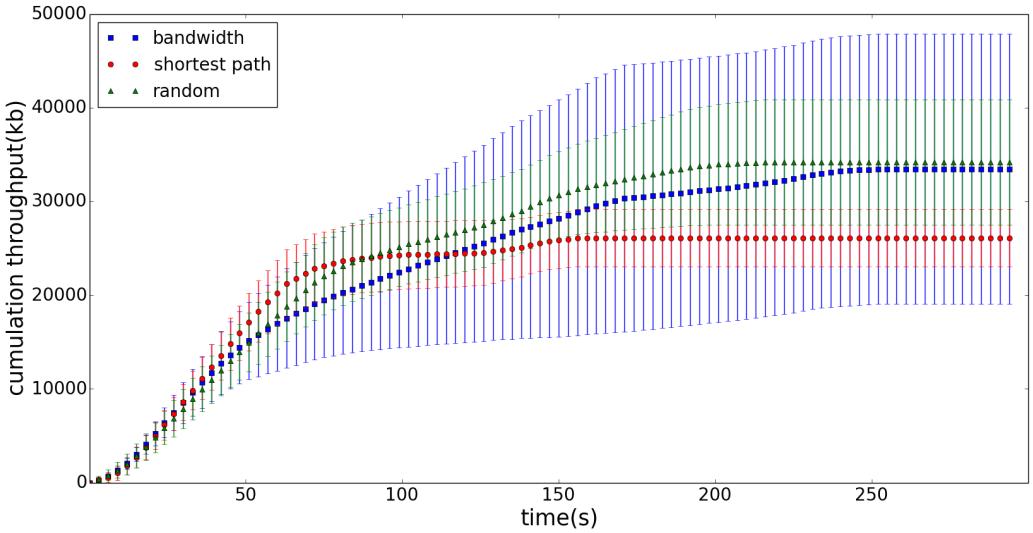

Figure 7.10. Bandwidth accumulation throughput for Scenario 3:link capacity homogeneous(GENI)

Moving Target Defense:Digital Fountain has better speed-security tradeoff

For the security, 3 Moving Target Defense policies, Digital Fountain policy still has the better speed, network overhead tradeoff. Shamir asks disjoint hosts for the keys, we can notice that the confidence interval is bigger comparing with the random and digital fountain policies.

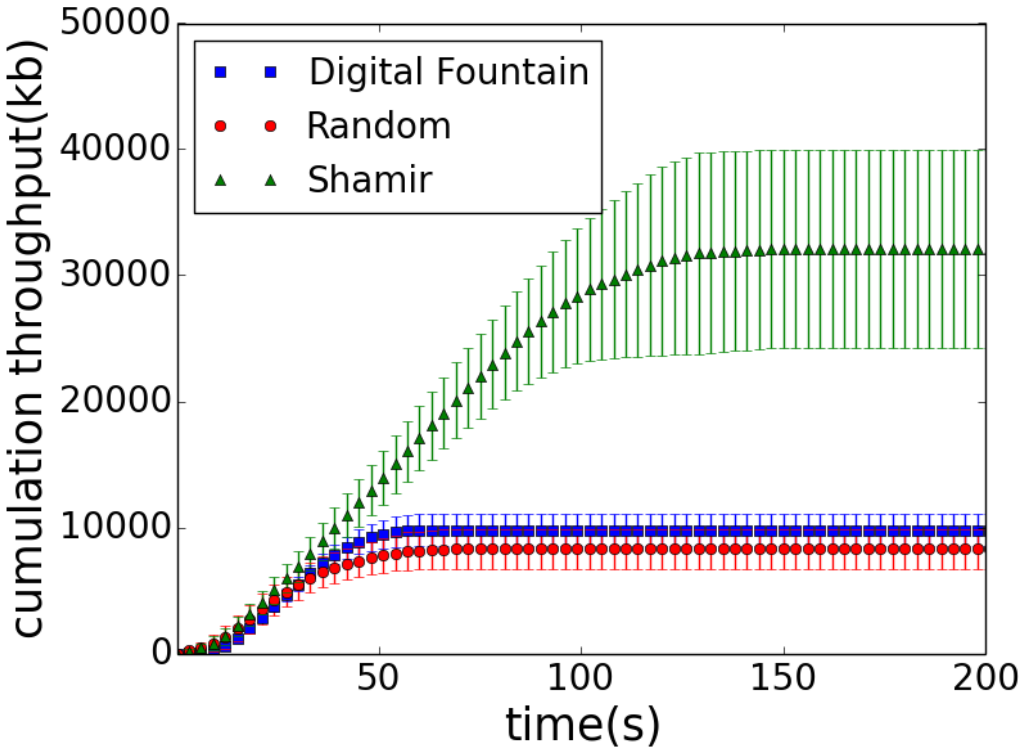

Figure 7.11. Bandwidth accumulation throughput for Moving Target Defense(GENI)

Shamir	Digital Fountain	Random
((74.7 ± 15.76)s)s	(24.8± 4.85)s	(18.4 ± 6.96) s

Table 7.8. migration time of 3 policies for Moving Target Defense(GENI)

In GENI environment, we can compare the policies performance in different network link capacity type. But the bandwidth used for migration varies each time because of the link bandwidth we specified in GENI topology is the maximum bandwidth we can use, not the exact bandwidth. So it happens that the available bandwidth we can use is in the range [0, bandwidth]. So, from the plot and also the migration time, we can notice that there is very big confidence interval value.

Chapter 8

Conclusion and Future Plan

In this thesis, we designed a policy-programmable container migration architecture based on Docker within SDN prototype. By policy, it means the architecture allows you to change policies with a simple configuration file, so programming the migration mechanism is easy. Software defined network makes network management simpler, it provides the REST API which can easily get the network topology and traffic statistics from the system.The architecture contains 3 main components. software defined measurement is a system that monitors the network traffic statistic, such as bandwidth, packets aggregate. It helps to make migration source host decision by detecting switches suffering the heavy traffic. migration manager is a system that maintains the database, policy sets for different application needs. Here we implement Loading Balancing policies, Moving Target Defense policies. Moving Target Defense policies aims to improve the system security, protect the migration destination host IP address. migration daemon is a process running in the host running Docker container. it enables the migration process between migration source host and migration destination host.

In order to test and evaluate the security and load balancing policies, we deploy the network within our SDN-based prototype over MININET. We test the migration happens from one host to another host, and compare the policies performance in different network situations, in particular, with different link capacity(heterogeneous and homogeneous). It proves that different policy in different network type, the performance is different, so this policy-based programmable architecture enables the flexibility to change the policy for adapting different application needs. Furthermore, we evaluate the payment in order to achieve network security by moving target defense policies. In particular, we compare the network traffic overhead and migration time using random policy as a reference.We get that Digital Fountain has a better better speed-security tradeoff.

As a plan for the future, we want to improve the system in several aspects. For the software defined measurement, we could integrate with big data and machine learning. In this case, the migration destination host can be predicted. By doing

this we can improve the network management service. For the testing part, we could deploy the system in GENI. In addition, we could test the policies tradeoff using different topology type, such as tree, linear, star, fully connected, evaluate different policies performance.

Chapter 9

Appendix: Migration System Install Guide

These installation instructions have been tested on Ubuntu 14.04.5.

According to the Architecture Design in Figure 4.1, we suggest that one machine can host Floodlight Controller, Database, and SDN Statistics Controller project which implements software Defined Measurement block and Migration Manager block. This host we call it Statistics Controller. Then, a set of hosts running docker container and need to perform migration. Each of the host running Migration Daemon block, we implement this block in a project named Migration Server. In the following steps, we will explain how to install software and tools needed as well as how to run this project.

9.1 Statistics Controller Side installation

In Statistics Server Side, we need to install SDN floodlight controller, mysql database, Maven, and Statistics Server project.

```
; Install floodlight controller
git clone http://github.com/floodlight/floodlight
sudo apt-get update
sudo apt-get -y install software-properties-common
   python-software-properties
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get -y install oracle-java8-installer
java -version
sudo apt-get -y install ant
sudo apt-get -y install curl
cd floodlight
```

```
ant

; Install maven
sudo apt-get install maven

; Install mysql database
sudo apt-get update
sudo apt-get install mysql-server
mysql_secure_installation
systemctl status mysql.service

; clone Statistics Controller maven project from Git hub
git clone
   https://github.com/TaoXu00/sdnStatisticController.git
```

After install all the tools, in sdnStatisticController project, in the resource folder, there is the database file. You can follow these step to import the database.

```
; import the database file to the mysql
mysql -u your username -p SDN < path/SDN.sql
```

9.2 Deploy network topology

We deploy and test our system in both Mininet testbed and Geni Testbed, you can build your own topology. Also can find the example file for mininet and Geni, from the Migration Statistics project resource folder. There are two script examples.

If you will deploy the system in Mininet, follow this link to install and get familiar with Mininet.

http://mininet.org/download/

If you will deploy the system in Geni, follow this link to register and learn how to use floodlight controller in Geni.

http://groups.geni.net/geni/wiki/GENIExperimenter/Tutorials/OpenFlowOVS-Floodlight

After deploying your topology, there are some configurations should be done in each open vswitch. First, you need to find all the interfaces that connecting with the hosts and other switches, for example, in switch1, you have port eth1, eth2, eth3, eth4, and eth5. Do not touch eth0. The commands are:

```
;eraser the IP address for the interface, set up the bridge
sudo ifconfig eth1 0
sudo ifconfig eth2 0
sudo ifconfig eth3 0
sudo ifconfig eth4 0
```

```
sudo ifconfig eth5 0
sudo ovs-vsctl add-br br0
sudo ovs-vsctl add-port br0 eth1
sudo ovs-vsctl add-port br0 eth2
sudo ovs-vsctl add-port br0 eth3
sudo ovs-vsctl add-port br0 eth4
sudo ovs-vsctl add-port br0 eth5
sudo ifconfig
sudo ovs-vsctl list-ports br0
;configure the switch with a remote floodlight controller,
   the controller ip should be the controller host eth0 IP
   address
ovs-vsctl set-controller tcp:<FloodLight_controller_ip>:6633
```

9.3 Migration Server Side Installation

In each host which will perform migration, we need to install Java, Maven, Docker, Criu. Criu is a tool use together with Docker API to perform container checkpoint.

After all the installation, we need to download Migration Server project from Git.

```
; Install Java
sudo apt-get update
sudo apt-get -y install software-properties-common
   python-software-properties
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get -y install oracle-java8-installer
java -version
```

If you already have java and maven installed, you can skip this step.

```
; Install maven
sudo apt-get install maven
mvn --version
```

Each host could run Docker container, and Criu is a tool integrated with docker to perform container checkpoint, store container image files.

```
; Install Docker
sudo apt-get update
curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
   sudo apt-key add -
```

```
sudo add-apt-repository "deb [arch=amd64]
   https://download.docker.com/linux/ubuntu $(lsb_release
   -cs) stable"
sudo apt-get update
sudo apt-cache policy docker-ce
sudo apt-get install docker-ce=17.06.0~ce-0~ubuntu
sudo docker version
```

In order to integrate Criu with Docker API, we need to set Docker Server experimental to true.

```
;set Docker Server experimental mode to true
cd /lib/systemd/system
sudo vim docker.service
ExecStart=/usr/bin/dockerd -H fd:// --experimental=true
sudo systemctl daemon-reload
sudo systemctl restart docker
```

```
;install Criu
sudo apt-get update \&\& sudo apt-get install -y
   protobuf-c-compiler libprotobuf-c0-dev protobuf-compiler
   libprotobuf-dev:amd64 gcc build-essential bsdmainutils
   python git-core asciidoc make htop git curl supervisor
   cgroup-lite libapparmor-dev libseccomp-dev
   libprotobuf-dev libprotobuf-c0-dev protobuf-c-compiler
   protobuf-compiler python-protobuf libnl-3-dev libcap-dev
   libaio-dev apparmor
git clone  https://github.com/xemul/criu.git criu
sudo apt-get -y install libnet1-dev
cd criu
make clean
make
sudo make install
cd ..

; clone Migration Server maven project from Github
git clone https://github.com/TaoXu00/migrationServer_geni.git
```

9.4 Configurations

In the Statistic Controller side,the configurations are :

- **floodlight controller**
  In order to provide the possibility to allow the Statistic Controller collect the

statistics from floodlight controller, we should enable the statistic module in floodlight configuration file.

Floodlight stores its startup configuration src/main/resources/floodlightdefault.properties. Add or set the following to modify the startup behavior of the statistics module.

```
; open file
   src/main/resources/floodlightdefault.properties
; change
   net.floodlightcontroller.statistics.enable=TRUE
```

- **Statistics Controller system files**
  host information
  After you develop your topology, you should construct a file containers the information of each hosts. The file name is hostsInfor.txt. The format is as following:

  connections
  ;if the connection is between switch and host, add switch port MAC address
  s1-eth2#02:5a:31:eb:98:57<->h1-eth1
  ;if connection is between switch and switch, only specify the interface name
  s1-eth4<->s3-eth1
  ;the connections switch and switch, host and host.

  IP
  ;host name and IP address h1 10.0.0.1
  h2 10.0.0.2
  h3 10.0.0.3
  h4 10.0.0.4
  h5 10.0.0.5
  link bandwidth
  ;link bandwidth followed by the bandwidth value(Mbps) s1<->h1:10
  s2<->h2:10
  s3<->h3:10
  s4<->h4:5
  s5<->h5:2
  s1<->s3:10
  s1<->s4:5
  s1<->s2:10
  s2<->s5:2
  configuration file
  In the Statistics Controller project, there is the configuration file, in order to test the policy, you can have several different choices. Change the policy name in the file named configuration.

policy for load balancing:
policy=random,secureMode=no
policy=bandwidth,secureMode=no
policy=shortest path,secureMode=no

policy for security(moving target defense):
policy=random,secureMode=Shamir
policy=random,secureMode=Digital_Fountain

9.5 Instructions to run system

First, launch Floodlight controller.

```
;In the floodlight controller root folder
 java -jar target/floodlight.jar
```

Second, launch Migration Server in each host.

```
cd migrationServer_geni
sudo chmod +x startup.sh
./startup.sh
```

Third, launch Statistics Controller project.

```
cd sdnStatisticController
sudo chmod +x startup.sh
./startup.sh
```

Bibliography

[1] Divya Kapil, Emmanuel S Pilli, and Ramesh C Joshi. «Live virtual machine migration techniques: Survey and research challenges». In: *Advance Computing Conference (IACC), 2013 IEEE 3rd International*. IEEE. 2013, pp. 963–969.

[2] Yi Wang, Jacobus E van der Merwe, and Jennifer Rexford. «VROOM: Virtual ROuters On the Move.» In: *HotNets*. 2007.

[3] Peng Lu et al. «Adaptive live migration to improve load balancing in virtual machine environment». In: *European Conference on Parallel Processing*. Springer. 2013, pp. 116–125.

[4] Inderjit Singh Dhanoa and Sawtantar Singh Khurmi. «Energy-efficient virtual machine live migration in cloud data centers». In: *International Journal of Computer Science and Technology (IJCST)* 5.1 (2014), pp. 43–47.

[5] Blain Barton. *Step-By-Step: Migrating Physical Servers to Virtual Machines*. 2014. URL: https://blogs.technet.microsoft.com/blainbar/2014/09/09/step-by-step-migrating-physical-servers-to-virtual-machines-blain-barton/ (visited on 09/2014).

[6] *KVM for virtual machine migration*. 2014. URL: https://www.linux-kvm.org/page/Migration.

[7] Christopher Clark et al. «Live migration of virtual machines». In: *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association. 2005, pp. 273–286.

[8] Divya Kapil, Emmanuel S Pilli, and Ramesh C Joshi. «Live virtual machine migration techniques: Survey and research challenges». In: *Advance Computing Conference (IACC), 2013 IEEE 3rd International*. IEEE. 2013, pp. 963–969.

[9] Gulshan Soni and Mala Kalra. «Comparative study of live virtual machine migration techniques in cloud». In: *International Journal of Computer Applications* 84.14 (2013).

[10] William Voorsluys et al. «Cost of virtual machine live migration in clouds: A performance evaluation». In: *IEEE International Conference on Cloud Computing*. Springer. 2009, pp. 254–265.

[11] Yulong Wang et al. «U-TRI: Unlinkability Through Random Identifier for SDN Network». In: *Proceedings of the 2017 Workshop on Moving Target Defense*. ACM. 2017, pp. 3–15.

[12] Amirreza Niakanlahiji and Jafar Haadi Jafarian. «WebMTD: Defeating Web Code Injection Attacks using Web Element Attribute Mutation». In: *Proceedings of the 2017 Workshop on Moving Target Defense*. ACM. 2017, pp. 17–26.

[13] Carlos E Rubio-Medrano et al. «Mutated Policies: Towards Proactive A ribute-based Defenses for Access Control». In: (2017).

[14] .

[15] *Criu for process and container migration*. URL: https://criu.org/Main_Page.

[16] *Criu for process and container migration*. URL: https://criu.org/P.Haul.

[17] Natasha Gude et al. «NOX: Towards an Operating System for Networks». In: *SIGCOMM Comput. Commun. Rev.* 38.3 (July 2008), pp. 105–110. ISSN: 0146-4833. DOI: 10.1145/1384609.1384625. URL: http://doi.acm.org/10.1145/1384609.1384625.

[18] Minlan Yu, Lavanya Jose, and Rui Miao. «Software Defined Traffic Measurement with OpenSketch.» In: *NSDI*. Vol. 13. 2013, pp. 29–42.

[19] Adi Shamir. «How to Share a Secret». In: *Commun. ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176. URL: http://doi.acm.org/10.1145/359168.359176.