单位代码 **10006**  

课程名称 **高等计算机网络**

分 类 号 **TP311.1**

****

中期报告

基于SDN的DDoS攻击防御实现

|  |  |
| --- | --- |
| 学院名称 | 计算机学院 |
| 团队成员 | 温雅楠SY2006345 |
| 团队成员 | 武仕沛ZY2006357 |
| 团队成员 | 王宇翔ZY2006160 |

2020年 11 月24日

目录

[1. 研究内容 3](#_Toc57127047)

[2. 已完成工作 3](#_Toc57127048)

[2.1环境搭建 3](#_Toc57127049)

[2.2模块设计与实现 10](#_Toc57127050)

[2.3实验设计与结果分析 10](#_Toc57127051)

[3. 关键技术与难点 10](#_Toc57127052)

[3.1 FloodLight核心组件与工作流程 10](#_Toc57127053)

[3.2伪造攻击源占用链路带宽 10](#_Toc57127054)

[3.3链路带宽占用解决方案 10](#_Toc57127055)

[3.4服务器计算资源占用 10](#_Toc57127056)

[3.5链路节点请求过滤算法 10](#_Toc57127057)

[4. 作业分工与进度安排 10](#_Toc57127058)

## 研究内容

分布式拒绝服务攻击（DDoS）通过大量合法分布式节点对服务器发送请求，从而淹没服务器临近链路带宽或耗尽服务器本身计算资源，最终使正常用户无法获得服务器的响应。

由于现行网络本身具有分布式的特点，因此在面对DDoS攻击时，往往难以及时协调一致的进行响应。通常分布式的网络设备是不具备状态的，也很难进行集中管理以记录状态，因此在收到转发包时，直接采取查表的方式进行转发，而不对数据流进行分析，这就给DDoS攻击流留有可乘之机。因此，当网络检测到DDoS攻击时，被攻击服务器临近链路已经被消耗了大量带宽，且服务器也已经收到了大量非法的请求。可见传统的网络在面临该攻击行为时，其反制措施往往有一定后滞性。

传统的采用分布式协议的网络如不能如期对DDoS攻击进行有效的防御，那么便考虑集中控制式的网络模型。因此，本实验采用SDN网络模型，结合mininet搭建起虚拟网络环境，控制器采用FloodLight，控制器和交换机之间采用OpenFlow协议传输报文和下发流控制规则，网络流量监视器采用sFlow进行监控。

本实验的工作可以分为两大部分，一是通过在mininet中搭建虚拟网络环境，并结合FloodLight开发，通过源码级的调试，来测试诸如控制器模块、报文交换模块、链路发现模块、拓扑管理模块和转发模块，进而对控制器有一个全面而详细的认识，并结合网络流量监测工具sFlow对网络流量进行实时监控，通过设置转发规则并查看前后流量变化，从而对SDN网络架构及其工作流程有一个宏观的认识。

另一部分的工作内容在于DDoS的攻防对抗上，正如前文所述，DDoS攻击可分为两大类，一是对服务器临近链路带宽的占用，二是对服务器本身计算资源的耗费。因此，这部分工作重点就在于如何设计攻击流量以达到占用链路带宽和耗尽服务器计算资源的效果，以及设计相应的算法或者逻辑规则，并部署到控制器上，针对这二者的攻击，起到良好的防御效果。

## 已完成工作

### 2.1环境搭建

本实验采用Windows+Linux双系统联调的方式进行。Windows上部署FloodLight，并在IDEA集成开发环境下进行开发。Linux上部署mininet实现虚拟网络组网，部署sFlow服务，实现网络流量监控。下面是具体的环境清单：

|  |  |  |
| --- | --- | --- |
| Windows10 64位  家庭版 1909 | Java环境 | Jdk 1.8.0\_271 |
| 集成开发环境 | IntelliJ IDEA 2020.2.2 |
| 虚拟机 | Vmware Pro 15.5.6 |
| 网络控制器 | Floodlight v0.9 src |
| Ubuntu 18.04  LTS 64位 | Java环境 | Jdk 1.8.0\_272 |
| Python环境 | Python 2.7.17 |
| 组网软件 | mininet 2.2-2 |
| 流量监控 | sFlow 3.0-1529 |

由于采用主机与虚拟机联调开发，因此需要在虚拟机上配置网关，使得两边网络联通。具体做法是在Vmware上配置NAT地址映射，虚拟机的网关设置为192.168.137.2，子网掩码为255.255.255.0，Linux下设置本机ip为192.168.137.181，然后测试双方连通性，双方连通性测试如图2-1和2-2所示。

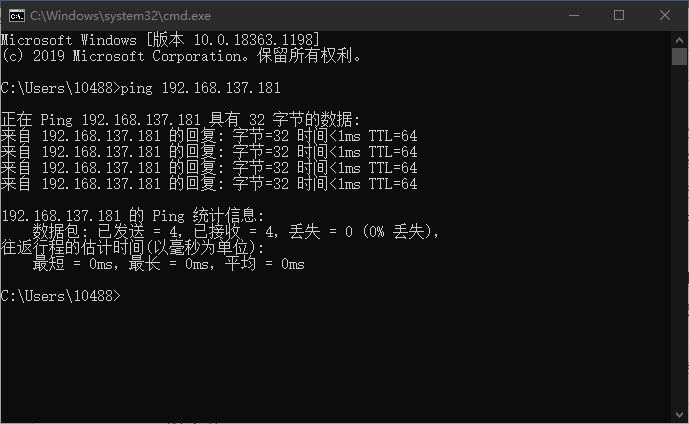
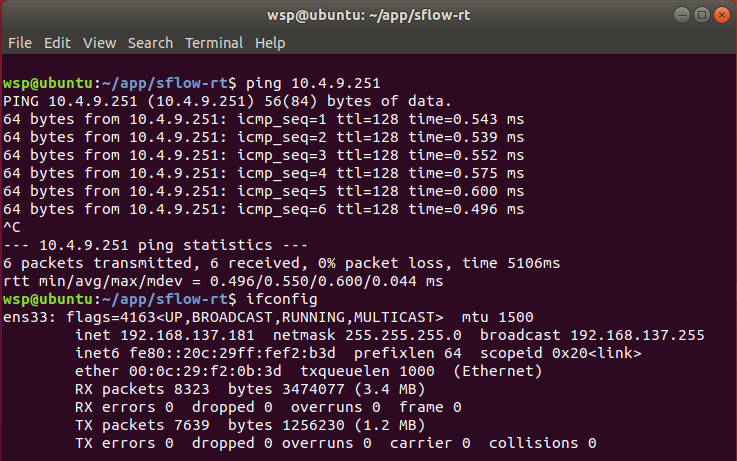


图2-1

图2-2

FloodLight控制器开发，本次实验不仅需要会使用FloodLight对网络进行管理，而且还要能够开发相应模块，以针对DDoS攻击实施转发规则的下发。因此本次实验FloodLight采用源码开发，配合maven对项目进行管理、构建。FloodLight源码可通过git clone git://github.com/floodlight/floodlight.git 命令下载，之后在IDEA中导入该工程，即可进行开发。初始版本可直接进行编译运行，FloodLight自带前端模块，可通过可视化的形式对所管理网络进行展示，初始启动环境配置参数在resources/floodlightdefault.properties文件中，这里列举一二：

|  |  |
| --- | --- |
| net.floodlightcontroller.restserver.RestApiServer.port | 8080 |
| net.floodlightcontroller.core.FloodlightProvider.openflowport | 6633 |
| net.floodlightcontroller.core.FloodlightProvider |  |
| ...... |  |

第一行参数为北向访问端口号，除了浏览器界面通过该端口对网络进行查看意外，流表的查看/下发也是通过该端口传输的。第二行参数为南向访问端口号，ovs交换机与控制器进行绑定就是通过该端口实现。第三行之后是控制器默认支持的模块，包括常见的链路发现模块、拓扑管理模块、转发模块等，这一部分将在3.1节中详细介绍。FloodLigtht开发界面和启动界面分别如图2-3、2-4所示。

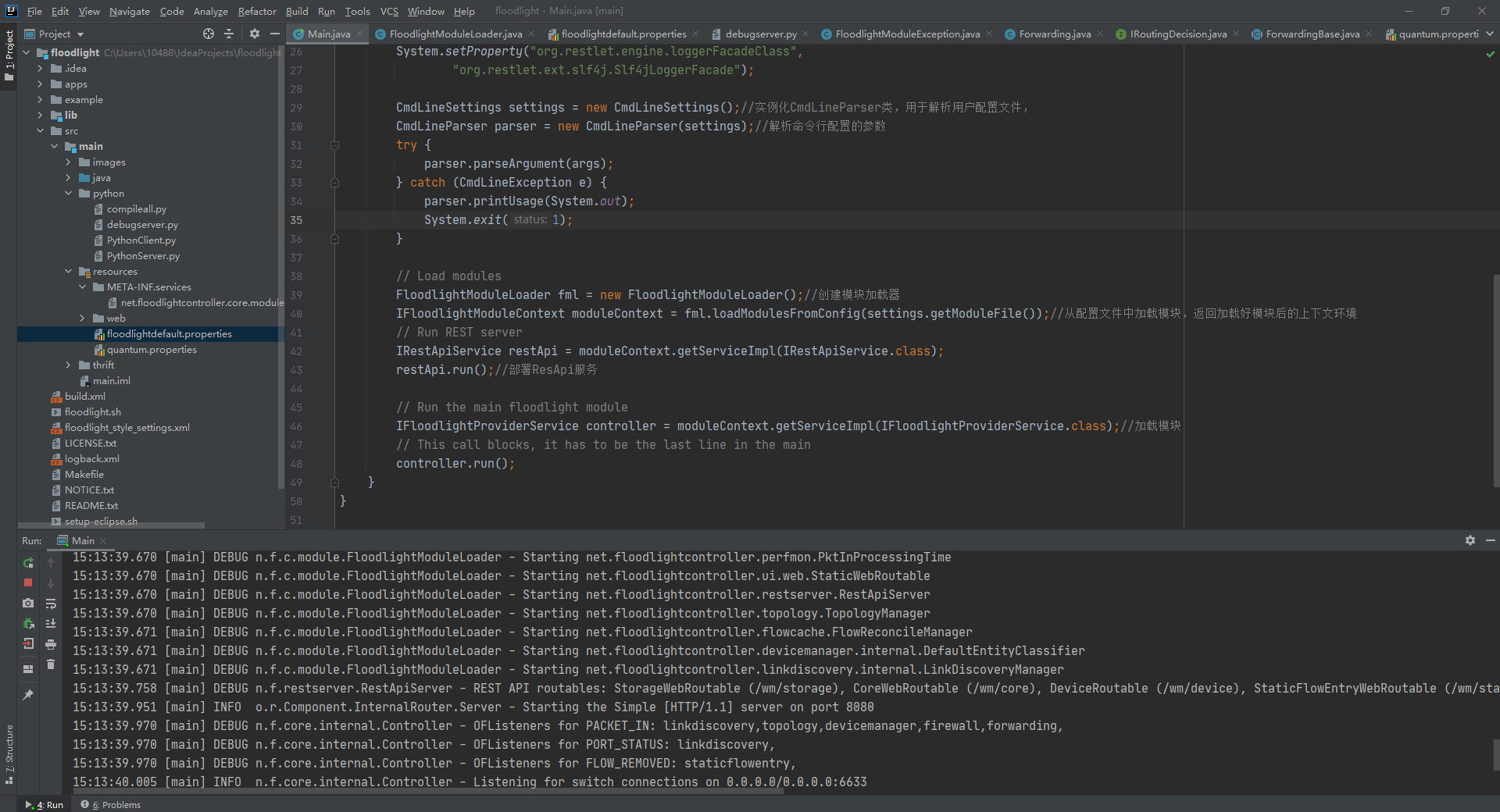


图2-3

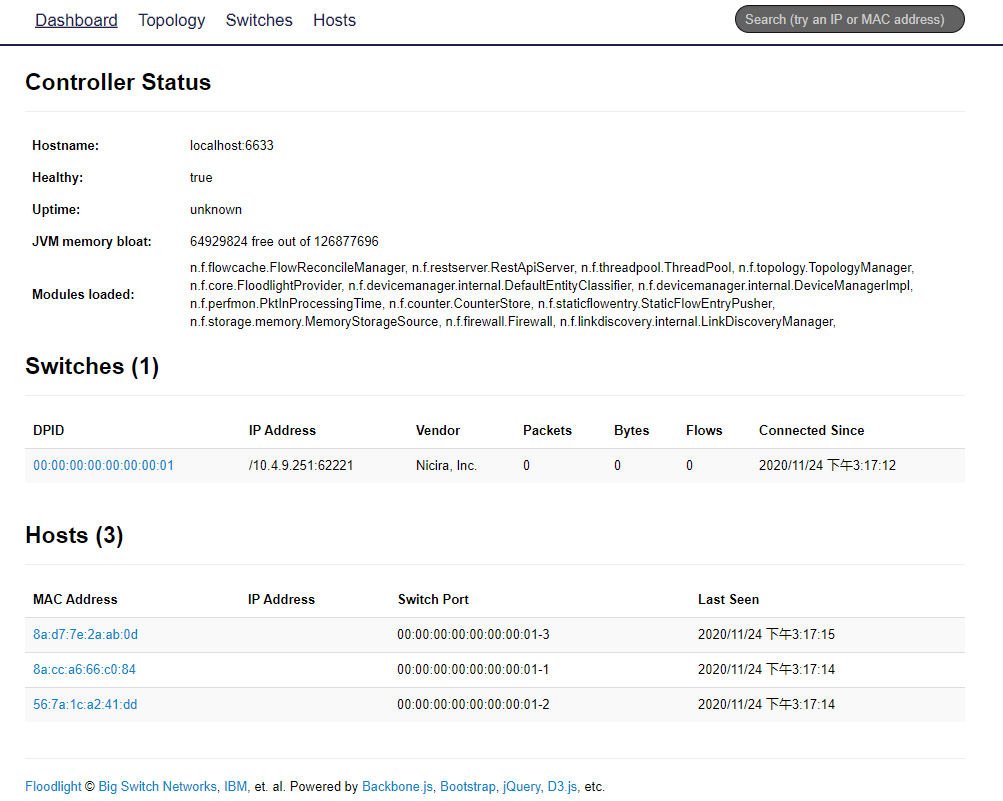


图2-4

Linux环境下，通过apt-get方式安装mininet组网软件，命令为sudo apt-get install mn，Java和Python环境以及sFlow监控软件通过源码包方式安装。

mininet组网软件使用方式比较简单，可通过命令式/交互式的方式生成期望的拓扑结构，这里以一个中心节点出度为8，深度为2的树形网络拓扑结构创建为例，使用的命令为：sudo mn --switch ovsk --topo tree,depth=2,fanout=8 --controller=remote,ip=10.4.9.251,port=6633，各参数的意义如下：

sudo：由于虚拟组网环境需要调用到内核网卡接口，因此需要管理员权限运行。

--switch：指定网络中交换机的类型，由于该实验交换机需要与控制器进行交互，因此需选用能够支持OpenFlow协议的交换机，ovsk中的ovs即代表Open vSwitch。

--topo：网络拓扑结构类型，常见的有single（单点）、linear（总线）、tree（树形）结构，根据选择的类型不同，需要指定具体的细分参数，如树形结构中需指明树深度，节点度数。

--controller：指定控制器来源，需指明ip地址和端口号。

创建过程如图2-5所示，另外，创建的网络拓扑结构可视化如图2-6所示。

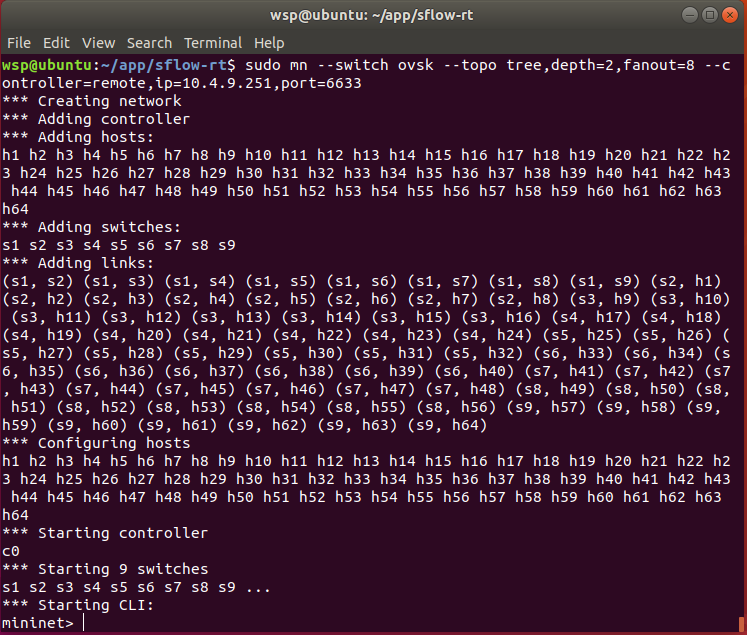


图2-5

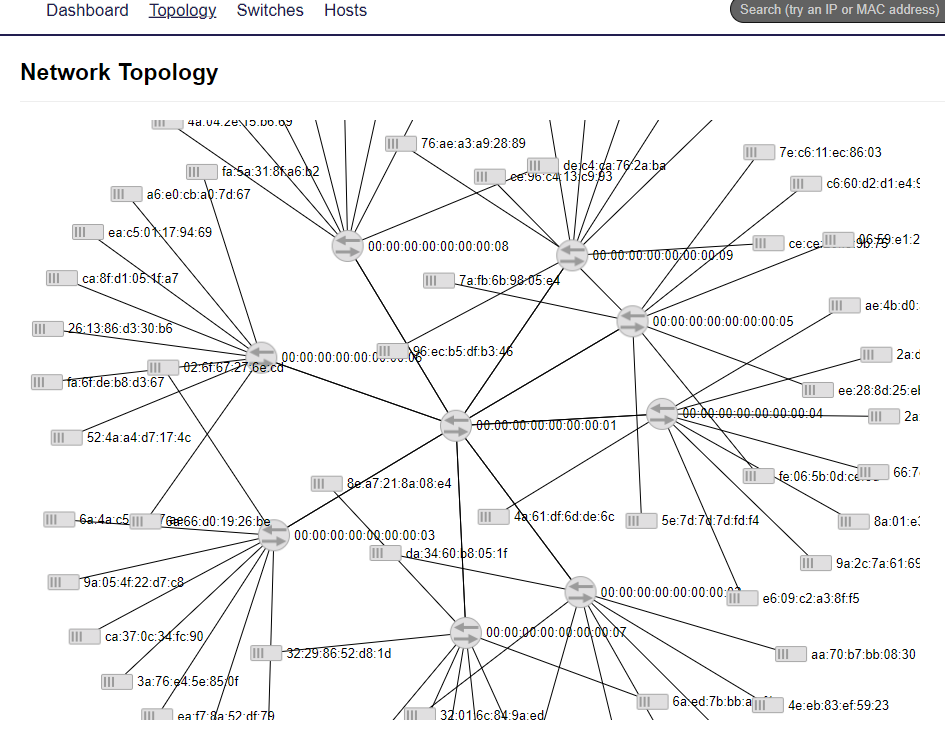


图2-6

mininet除了可以进行虚拟网络搭建以外，还可以针对交换机、主机进行单独管理，例如ping报文测试，这也为后面的攻击行为提供了环境基础。使用方式实在mininet交互式命令行中，通过xterm -name单独对打开的节点进行管理，（-name代表交换机或主机节点），这里以xterm h1为例，打开h1主机的管理窗口，并进行ping报文的测试，被ping主机为h32（10.0.0.32），ping命令结果如图2-7所示。

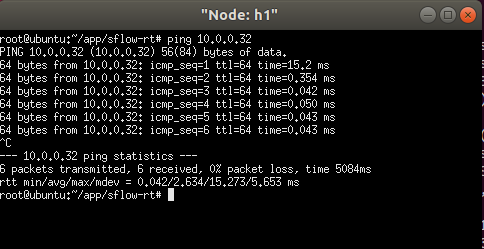


图2-7

流量监控软件sFlow可以实现对网络中单个或多个被管交换机进行流量监控，这取决于配置的sFlow-agent的数量。首先启动sFlow服务端，命令为：./sflow-rt/start.sh。之后需要针对被管交换机设置sFlow-agent，设置了agent的交换机会将自身链路接口处的流量信息上传给sFlow-agent，agent配置命令为：sudo ovs-vsctl -- --id=@sflow create sflow agent=eth0 target=\"127.0.0.1:6343\" sampling=10 polling=20 -- -- set bridge s1 sflow=@sflow。命令参数意义如下：

agent：指定采样的网卡，由于虚拟组网环境中，所有的流量都会经过虚拟机的主网卡，因此网卡为eth0。

target：sFlow服务端地址，默认端口号为6343。

sampling：采样率，即每隔N个Packet采样一次。

polling：轮询时间，即每隔N秒轮询一次。

bridge：被监控的网络设备，这里以h1所在的交换机s1为例，待会儿以h1主机进行ping报文测试，查看流经s1的流量。需说明的是，如果要针对全网环境测试的话，该命令需要多次配置，以覆盖所有交换机/路由器。

服务端，客户端都配置完成以后，便可打开服务端界面，默认端口号为8008，总览界面如图2-8所示。

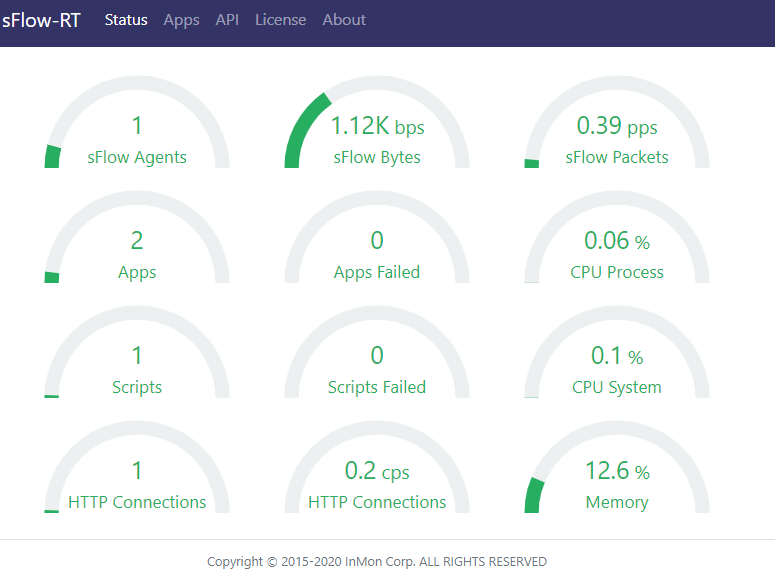


图2-8

这时在mininet中执行h1 ping h32，查看sFlow流量情况，通过设置规则为“ipsource，ipdestination”查看该指标下的流量情况，流量结果如图2-9所示。

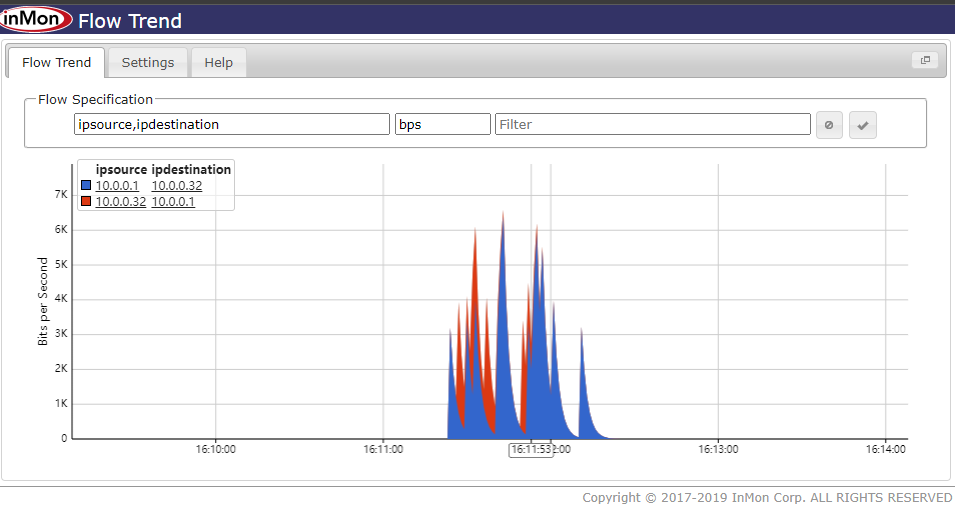


图2-9

### 2.2模块设计与实现

#### 2.2.1合法请求流量生成模块

合法请求流量生成模块用于检验网络的连通性，在配置了DDoS防御机制以后，通过对合法请求流量的监控，判断该防御机制是否影响到了正常的请求。另外还可以随非法请求流量生成模块一起启动，通过检测非法攻击流量对合法请求流量的影响程度，来判断攻击模块是否有效。

合法请求流量生成模块通过模拟请求报文的产生，将源ip地址和目标ip地址在10.0.0.1-10.0.0.64范围内随机产生，并设置请求报文的发送周期，以最大限度的模拟全网范围内真实且合理的网络请求。

以下是合法请求流量生成模块的代码，由Python编写。

|  |
| --- |
| #coding=utf-8  import sys  import getopt  import time  from os import popen  import logging  logging.getLogger("scapy.runtime").setLevel(logging.ERROR)  from scapy.all import sendp, IP, UDP, Ether, TCP  from random import randrange  #源ip地址产生函数，生成范围为10.0.0.start-10.0.0.end  def generateSourceIP(start, end):  first = 10  second = 0  third = 0  ip = ".".join([str(first), str(second), str(third), str(randrange(start,end))])  return ip  #目标ip地址产生函数，生成范围为10.0.0.start-10.0.0.end  def generateDestinationIP(start, end):  ...逻辑与generateSourceIP函数完全相同  def main(argv):  #从启动命令中获取start，end，num参数，start-end代表ip地址随机范围为10.0.0.start-10.0.0.end，num代表发送报文的次数  try:  opts, args = getopt.getopt(sys.argv[1:], 's:e:l:', ['start=','end=','num='])  except getopt.GetoptError:  sys.exit(-1)  for opt, arg in opts:  if opt =='-s':  start = int(arg)  elif opt =='-e':  end = int(arg)  elif opt =='-l':  num = int(arg)  if start == '':  sys.exit(-1)  if end == '':  sys.exit(-1)  if num == '':  sys.exit(-1)  interface = popen('ifconfig | awk \'/eth0/ {print $1}\'').read()  #获取网路接口，之后通过该接口发送报文  for i in xrange(num):  packets = Ether() / IP(dst = generateDestinationIP (start, end), src = generateSourceIP (start, end)) / UDP(dport = 80, sport = 2)  #生成源和目的IP地址在10.0.0.start-10.0.0.end范围内的ping报文。  print("packets",i,repr(packets))  #打印当前ping报文信息，指明当前已经发出去且接受到响应的报文数量  sendp(packets, iface = interface.rstrip(), inter = 0.2)  #以0.2s为周期发送报文，sendp方法需经历一个完整的发送与接受过程，如果没有收到返回来的响应报文，则被阻塞，导致后续报文无法发送，通过这一特性，可以检验合法请求流量是否得到及时处理  if \_\_name\_\_ == '\_\_main\_\_':  main(sys.argv) |

合法请求流量最重要的部分在于sendp(packet =, iface =, inter = )由于该方法除了发送报文以外，在

#### 2.2.2非法请求流量生成模块

#### 2.2.3 packet-in协调转发模块

#### 2.2.4基于时间序列的非法请求报文过滤模块

2.2.5小型服务器检测模块

### 2.3实验设计与结果分析

## 关键技术与难点

### 3.1 FloodLight核心组件与工作流程

### 3.2伪造攻击源占用链路带宽

### 3.3链路带宽占用解决方案

### 3.4服务器计算资源占用

### 3.5链路节点请求过滤算法

## 作业分工与进度安排