高级计算机网络实验报告

学号：SA18011081　姓名：苑福利

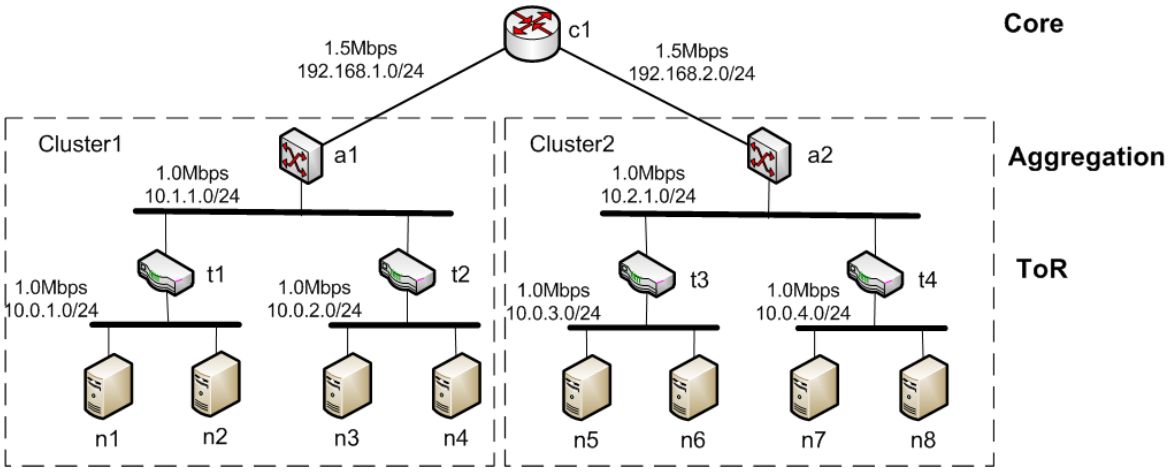
Project 1: Datacenter network simulation using ns-3

1、实验目标

- 使用 ns3 模拟数据中心网络。
- 实验模拟仿真两种流量模式。
- 仿真测试网络。
- 发现网络瓶颈，并改善网络。

2、实验要求

a) 数据中心网络拓扑如下：



根据提供的网络拓扑图完成实验。

b) 拓扑图描述：

- 8 servers: n1~8; 4 ToR switches: t1~4; 2 aggregation switches: a1~2; 1 core switch: c1；
- 网络被划分为两个簇；
- 连接到 C1 的链路是 PPP，其他网络是 Ethernets，网络的容量显示在拓扑图上。
- 网络上所有的端到端延迟都是 500ns。

- IP 地址分配显示在拓扑图上。
- 所有交换机都像 OSPF 路由器一样。

c) **两种通信模式（traffic pattern）**

- 模式一：簇间通信（inter-cluster traffic）
  每个服务器使用 TCP 与来自不同簇的另一个服务器进行通信，比如 1—5,6—2,3—7,8—4；
- 模式二：多对一通信（many-to-one traffic）
  选择一个服务器作为接收器，所有其他服务器与之通信；
- 分别模拟两种模式，获得网络可以达到的吞吐量，找出网络的瓶颈，并改善网络。

3、实验内容

3.1、模式一仿真

（1） 实现与代码注释

a) 自定义命令参数：

定义一个是否启用 Echo Client 和 Echo Server 应用中内置的两个日志组件的命令参数：verbose，默认为 TRUE，即启用。也可以自己定义一些其他辅助命令参数。

```
bool verbose = true;

CommandLine cmd;
//添加自己的变量 （属性名称，属性说明，变量）
cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);

cmd.Parse (argc,argv);

if (verbose){
    LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
    LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
}
```

b) 创建节点：

根据上面的网络拓扑图，可知需要创建 15 个节点，包括 8 个 servers，4 个 ToR switches，2 个 aggregation switches，1 个 core switch；从整个网络拓扑结构中从上往下可以看出网络分为三层：

- 顶层：由 aggregation 和 core 组成的 PPP 网络，具体由代码中的 aggregation1 和 aggregation2 体现；

- 中间层：由 aggregation 和 ToR 组成的 Ethernets 网络，具体由代码中的 toR1 和 toR2 体现；
- 底层：由服务器和 ToR 组成的 Ethernets 网络，具体由代码中的 camaNode1、camaNode2、camaNode3 和 camaNode4 体现；

```
/************网络拓扑部分*************/
//顶层网络：创建2个aggregation:a1,a2 core-->a1 core-->a2 ptpNo
//a1 = aggregation1.Get(1)   a2 = aggregation2.Get(1)  core = 
NodeContainer aggregation1;
aggregation1.Create(2);   //c1,a1

NodeContainer aggregation2;
aggregation2.Add(aggregation1.Get(0));//C1
aggregation2.Create(1);  //a2

//中间网络部分 ： a1-->t1,t2  a2-->t3,t4
//创建ToR
NodeContainer toR1,toR2;
toR1.Add(aggregation1.Get(1));//a1
toR1.Create(2);  //t1,t2
toR2.Add(aggregation2.Get(1));//a2
toR2.Create(2);  //t3,t4

//底层网络部分：t1-->n1,n2  t2-->n3,n4  t3-->n5,n6  t4-->n7,n8
//创建节点Nodes   csmaNodes
NodeContainer csmaNode1,csmaNode2,csmaNode3,csmaNode4;
csmaNode1.Add(toR1.Get(1));//t1
csmaNode1.Create(2);//n1,n2
csmaNode2.Add(toR1.Get(2));//t2
csmaNode2.Create(2);//n3,n4
csmaNode3.Add(toR2.Get(1));//t3
csmaNode3.Create(2);//n5,n6
csmaNode4.Add(toR2.Get(2));//t4
csmaNode4.Create(2);//n7,n8
```

c) 设置 P2P 传输速率及信道延迟：

顶层 PPP 网络的数据传输速率为 1.5Mbps，延迟都为 500ns：

```
//设置传送速率和信道延迟
PointToPointHelper pointToPoint1,pointToPoint2;
pointToPoint1.SetDeviceAttribute ("DataRate", StringValue ("1.5Mbps"));
pointToPoint1.SetChannelAttribute ("Delay"   , TimeValue(NanoSeconds(500)) );
pointToPoint2.SetDeviceAttribute ("DataRate", StringValue ("1.5Mbps"));
pointToPoint2.SetChannelAttribute ("Delay"   , TimeValue(NanoSeconds(500)) );

//安装P2P网卡设备到P2P网络节点
NetDeviceContainer p2pDevices1,p2pDevices2;
p2pDevices1 = pointToPoint1.Install(aggregation1);
p2pDevices2 = pointToPoint2.Install(aggregation2);
```

d) 创建和连接 csma 设备及信道：

中间层和底层是 Ethernets 网络，并且数据传输速率为 1.0Mbps，延迟都为 500ns：

```
//创建和连接CSMA设备及信道
CsmaHelper csma;
csma.SetChannelAttribute("DataRate", StringValue("1Mbps"));
csma.SetChannelAttribute("Delay"    , TimeValue(NanoSeconds(500)) );

//中间层csma
NetDeviceContainer toR1Devices1,toR2Devices2;
toR1Devices1 = csma.Install(toR1);
toR2Devices2 = csma.Install(toR2);
//底层 csma
NetDeviceContainer csmaNode1Devices1,csmaNode1Devices2,
                   csmaNode1Devices3,csmaNode1Devices4;
csmaNode1Devices1 = csma.Install(csmaNode1);
csmaNode1Devices2 = csma.Install(csmaNode2);
csmaNode1Devices3 = csma.Install(csmaNode3);
csmaNode1Devices4 = csma.Install(csmaNode4);
```

e) 安装网络协议：

```
//安装网络协议
InternetStackHelper stack;
stack.Install (aggregation1);
stack.Install (aggregation2.Get(1));
stack.Install (toR1.Get(1));
stack.Install (toR1.Get(2));
stack.Install (toR2.Get(1));
stack.Install (toR2.Get(2));
stack.Install (csmaNode1.Get(1));
stack.Install (csmaNode1.Get(2));
stack.Install (csmaNode2.Get(1));
stack.Install (csmaNode2.Get(2));
stack.Install (csmaNode3.Get(1));
stack.Install (csmaNode3.Get(2));
stack.Install (csmaNode4.Get(1));
stack.Install (csmaNode4.Get(2));
```

f) 为网络分配 IP 地址：

```
//两个网段的IP地址类对象
Ipv4AddressHelper address;
//安排P2P网段的地址
address.SetBase("192.168.1.0","255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces1 = address.Assign(p2pDevices1);
address.SetBase("192.168.2.0","255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces2 = address.Assign(p2pDevices2);

//安排CSMA网段地址
address.SetBase("10.1.1.0","255.255.255.0");
Ipv4InterfaceContainer toRInterfaces1 = address.Assign(toR1Devices1);
address.SetBase("10.1.2.0","255.255.255.0");
Ipv4InterfaceContainer toRInterfaces2 = address.Assign(toR2Devices2);
address.SetBase("10.0.1.0","255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces1 = address.Assign(csmaNode1Devices1);
address.SetBase("10.0.2.0","255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces2 = address.Assign(csmaNode1Devices2);
address.SetBase("10.0.3.0","255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces3 = address.Assign(csmaNode1Devices3);
address.SetBase("10.0.4.0","255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces4 = address.Assign(csmaNode1Devices4);
/************网络拓扑部分结束************/
```

g) 创建 PacketSink 和 OnOffclient：

以 n1—>n5 为例，另外的三个通信：n6—>n2, n3—>n7 ,n8—>n4 代码是完全类似的。其中使用 PacketSinkHelper 来创建一个 TCP 接收器，其中端口 sinkPort 为 8080，同样地，创建为服务器端 OnOff 应用程序发送的 TCP 数据。

```
//建立 traffic : 1-5
PacketSinkHelper packetSinkHelper1("ns3::TcpSocketFactory",
            InetSocketAddress(csmaInterfaces3.GetAddress(1),sinkPort));//n5
ApplicationContainer sinkApp1 =
                        packetSinkHelper1.Install(csmaNode3.Get(1));
sinkApp1.Start(Seconds(0.0));
sinkApp1.Stop(Seconds(20.0));

OnOffHelper client1("ns3::TcpSocketFactory",
            InetSocketAddress(csmaInterfaces3.GetAddress(1),sinkPort));
client1.SetAttribute ("OnTime",
            StringValue("ns3::ConstantRandomVariable[Constant=50]"));
client1.SetAttribute ("OffTime",
            StringValue("ns3::ConstantRandomVariable[Constant=0]"));
client1.SetAttribute ("DataRate", DataRateValue (DataRate ("1.0Mbps")));
client1.SetAttribute ("PacketSize", UintegerValue (2000));
ApplicationContainer clientApp1 = client1.Install (csmaNode1.Get (1));
clientApp1.Start(Seconds (1.0 ));
clientApp1.Stop (Seconds (21.0));
//使用 csma 网段第二个节点进行 sniff，True 开启混杂模式
csma.EnablePcap ("Pattern1_n1_to_n5",csmaNode1Devices3.Get(1),true);
```

h) 创建路由表：

这里我们直接调用 ns3 中自带的路由实现：

```
/****调用全局路由,帮助建立网络路由****/
//全局路由管理器根据节点产生的链路通告为每个节点建立路由表
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

i) 生成 trace files：

这里开启 pcap 跟踪，生成.pcap 文件，便于使用 wireshark 分析跟踪：

```
/***********开启pcap跟踪**************/
//生成.pcap文件，便于使用Wireshark分析跟踪
pointToPoint1.EnablePcapAll ("Pattern1_p2p1");
//csma2.EnablePcapAll ("csma2");
csma3.EnablePcapAll ("cama3");

Simulator::Run();
Simulator::Destroy();
return 0;
```

（2） 实验结果分析

a) Pattern1 实验结果：

执行命令：./waf --run scratch/dcn 对编写的仿真代码编译运行

```
fuli@fuli-virtual-machine:~/fox/ns-3/ns-allinone-3.27/ns-3.27$ ./waf --run scratch/dcn
Waf: Entering directory `/home/fuli/fox/ns-3/ns-allinone-3.27/ns-3.27/build'
[2325/2715] Compiling scratch/dcn.cc
[2658/2715] Linking build/scratch/dcn
Waf: Leaving directory `/home/fuli/fox/ns-3/ns-allinone-3.27/ns-3.27/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (6.060s)
fuli@fuli-virtual-machine:~/fox/ns-3/ns-allinone-3.27/ns-3.27$ ll
```

同时会产生如下的.Pcap 文件，我们可以使用 wireshark 软件对抓取的包进行分析。

```
fuli@fuli-virtual-machine:~/fox/ns-3/ns-allinone-3.27/ns-3.27$ ll *.pcap
-rw-rw-r-- 1 fuli fuli  731938 4月    8 21:50 cama3-10-0.pcap
-rw-rw-r-- 1 fuli fuli  691940 4月    8 21:50 cama3-11-0.pcap
-rw-rw-r-- 1 fuli fuli 2738560 4月    8 21:50 cama3-1-1.pcap
-rw-rw-r-- 1 fuli fuli  714326 4月    8 21:50 cama3-12-0.pcap
-rw-rw-r-- 1 fuli fuli  601708 4月    8 21:50 cama3-13-0.pcap
-rw-rw-r-- 1 fuli fuli  731938 4月    8 21:50 cama3-14-0.pcap
-rw-rw-r-- 1 fuli fuli 2738560 4月    8 21:50 cama3-2-1.pcap
-rw-rw-r-- 1 fuli fuli 1405762 4月    8 21:50 cama3-3-0.pcap
-rw-rw-r-- 1 fuli fuli 1405922 4月    8 21:50 cama3-3-1.pcap
-rw-rw-r-- 1 fuli fuli 1333142 4月    8 21:50 cama3-4-0.pcap
-rw-rw-r-- 1 fuli fuli 1333302 4月    8 21:50 cama3-4-1.pcap
-rw-rw-r-- 1 fuli fuli 1405762 4月    8 21:50 cama3-5-0.pcap
-rw-rw-r-- 1 fuli fuli 1405922 4月    8 21:50 cama3-5-1.pcap
-rw-rw-r-- 1 fuli fuli 1333142 4月    8 21:50 cama3-6-0.pcap
-rw-rw-r-- 1 fuli fuli 1333302 4月    8 21:50 cama3-6-1.pcap
-rw-rw-r-- 1 fuli fuli  691940 4月    8 21:50 cama3-7-0.pcap
-rw-rw-r-- 1 fuli fuli  714326 4月    8 21:50 cama3-8-0.pcap
-rw-rw-r-- 1 fuli fuli  601708 4月    8 21:50 cama3-9-0.pcap
-rw-rw-r-- 1 fuli fuli 2636736 4月    8 21:50 Pattern1_p2p1-0-0.pcap
-rw-rw-r-- 1 fuli fuli 2636736 4月    8 21:50 Pattern1_p2p1-0-1.pcap
-rw-rw-r-- 1 fuli fuli 2636736 4月    8 21:50 Pattern1_p2p1-1-0.pcap
-rw-rw-r-- 1 fuli fuli 2636736 4月    8 21:50 Pattern1_p2p1-2-0.pcap
```

我们对上述抓到的包进行分析，主要按照网络拓扑的三层也可以分为三类：其中以 Pattern1 开头的是最上层由 aggregation 和 core 组成的 PPP 网络抓取的包；其中以 csma3-1 和 csma3-2 开头代表 a1 和 a2 节点；以 csma3-3，csma3-4，csma3-5 和 csma3-6 分别是 t1~t4 由 aggregation 和 ToR 组成的 Ethernets 网络抓取的包；最后以 csma3-7~csma3-14 开头的是从服务器和 ToR 组成的 Ethernets 网络抓取的包。

对实验数据进行分析时，主要分别从三层网络抓取的包中选取一个，进行分析，主要的分析性能指标是：**吞吐量**；还有往返时间和窗口尺寸不做重点分析。

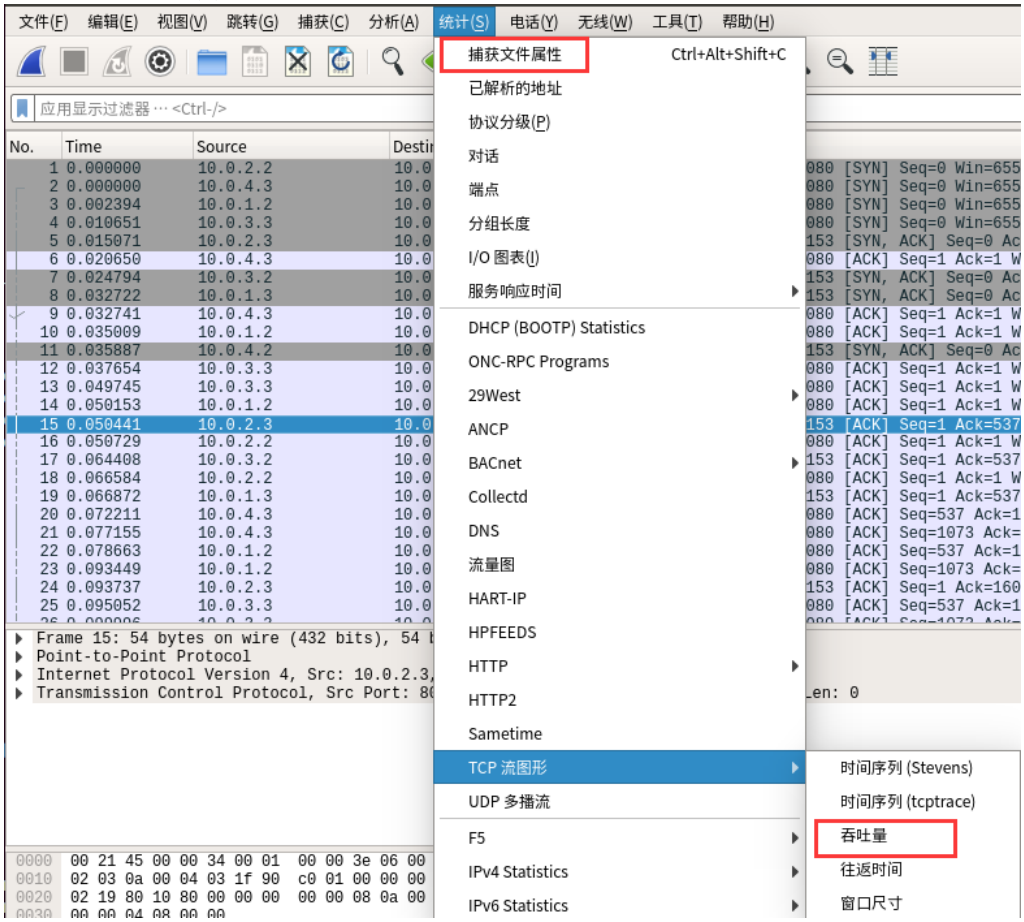
编写的代码是对于服务器 1—5,6—2,3—7,8—4 之间通信，因此：底层通过 n1→n5，即 csma3-7-0.pcap 来估计分析；

中间层通过 ToR t1 即 csma3-3-0.pcap 来估计分析；

顶层通过 core C1 即 pattern1-p2p1-0-0.pcap 来进行估计分析；

具体分析方法：

- 打开 wireshark 软件，分别打开保存的捕获文件；
- 点击统计(s)→TCP 流图形→吞吐量（往返时间/窗口尺寸）；
- 点击统计(s)→捕获文件属性，会显示总结报告；



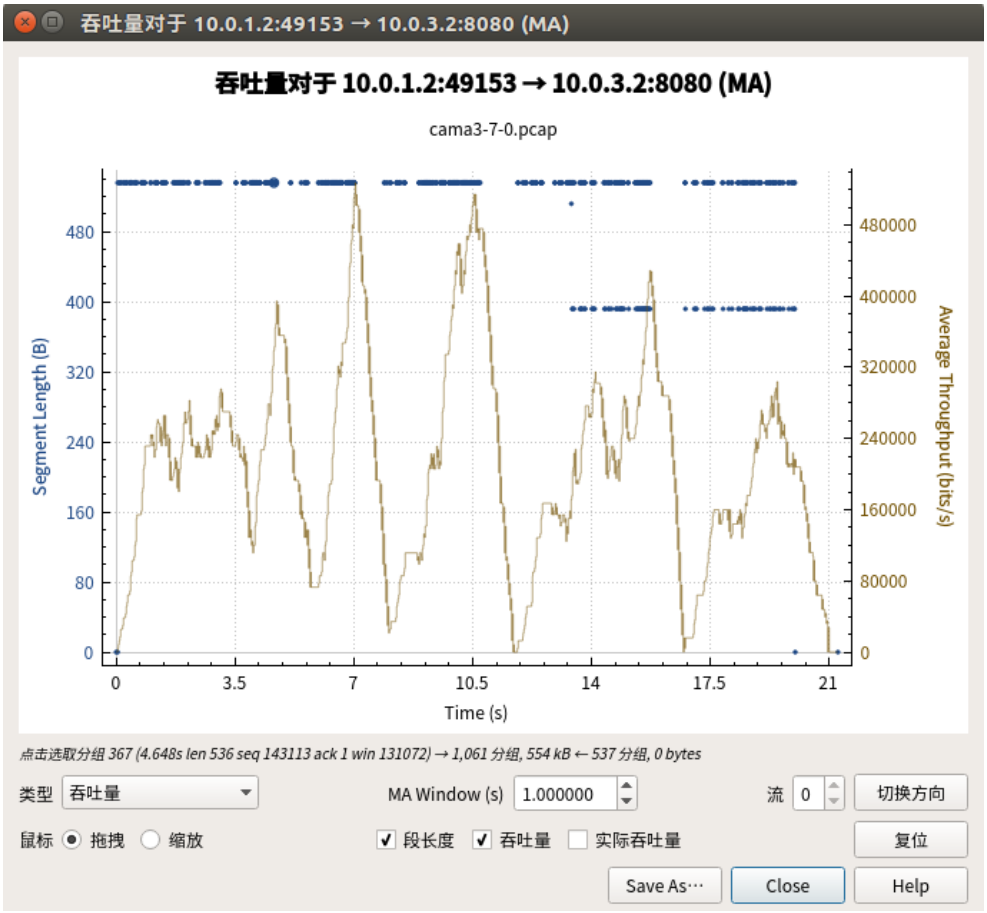
对于 server n1 测量结果如下：

接口

接口	丢弃分组	捕获过滤器	链路类型	分组大小限制
未知	未知	未知	Ethernet	65535 字节

统计

测量	已捕获	已显示	标记
分组	1604	1604 (100.0%)	—
时间跨度,s	21.781	21.781	—
平均 pps	73.6	73.6	—
平均分组大小,B	415	415	—
字节	666252	666252 (100.0%)	0
平均 字节/秒	30 k	30 k	—
平均 比特/秒	244 k	244 k	—



从上面两张图的仿真结果看出：是由服务器和 ToR 组成的 Ethernets 网络，网络的平均吞吐量是 244Kbit/s=0.244Mbps；

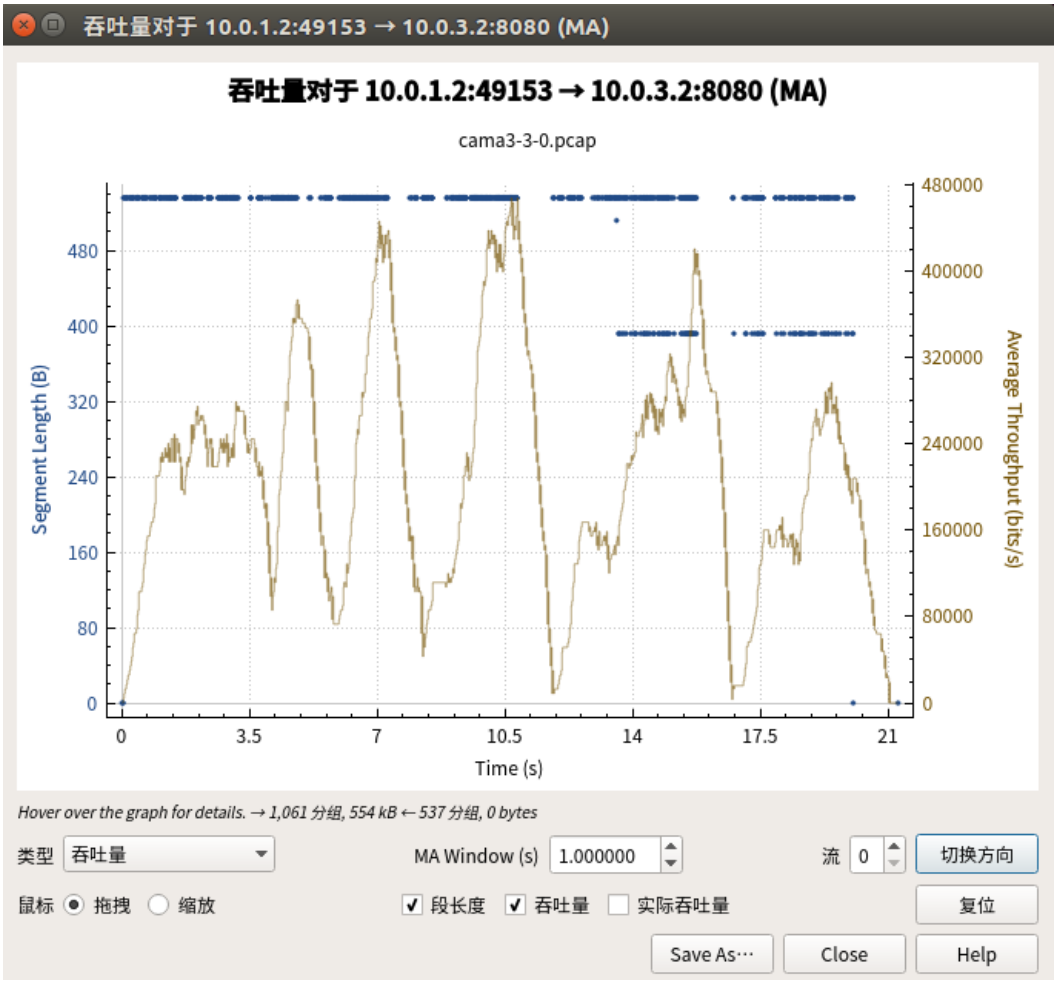
对于中间层通过 ToR t1 测量结果如下：

接口

接口	丢弃分组	捕获过滤器	链路类型	分组大小限制
未知	未知	未知	Ethernet	65535 字节

统计

测量	已捕获	已显示	标记
分组	3253	3253 (100.0%)	—
时间跨度,s	21.770	21.770	—
平均 pps	149.4	149.4	—
平均分组大小,B	416	416	—
字节	1353690	1353690 (100.0%)	0
平均 字节/秒	62 k	62 k	—
平均 比特/秒	497 k	497 k	—



从上面两张图的仿真结果看出：是由 aggregation 和 ToR 组成的 Ethernets 网络，网络的平均吞吐量是 497Kbit/s=0.497Mbps；

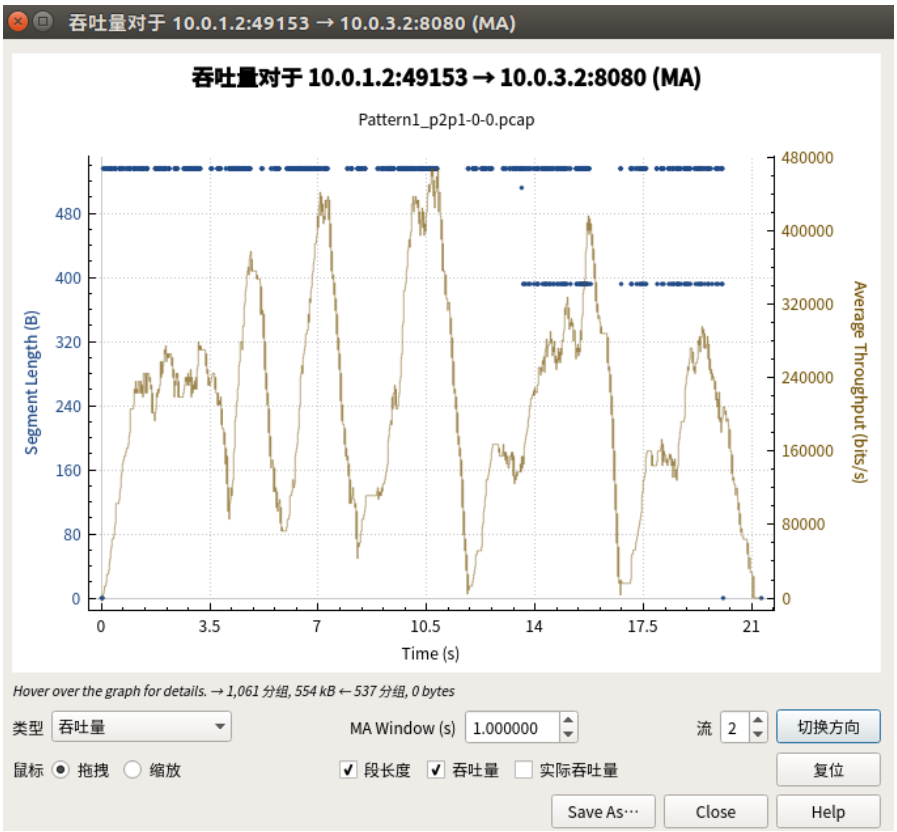
对于顶层通过 core C1 测量结果如下：

接口

接口	丢弃分组	捕获过滤器	链路类型	分组大小限制
未知	未知	未知	PPP	65535 字节

统计

测量	已捕获	已显示	标记
分组	6324	6324 (100.0%)	—
时间跨度,s	21.680	21.680	—
平均 pps	291.7	291.7	—
平均分组大小,B	401	401	—
字节	2535528	2535528 (100.0%)	0
平均 字节/秒	116 k	116 k	—
平均 比特/秒	935 k	935 k	—



从上面两张图的仿真结果看出：是由 aggregation 和 core 组成的 PPP 网络，网络的平均吞吐量是 935Kbit/s=0.935Mbps；

b) Pattern1 实验结果分析：

将上述得到的结果用图表汇总显示：

模式一仿真汇总

网络	结点	带宽	网络平均吞吐量
底层	Server n1	1.0 Mbps	0.244 Mbps
中间	ToR t1	1.0 Mbps	0.497 Mbps
顶层	Aggregation a1	1.5 Mbps	0.935 Mbps

从上面测得结果可以看出：

底层网络平均吞吐量为 0.244Mbps，带宽利用率为：24.4%；

中间网络平均吞吐量为 0.497Mbps，带宽利用率为：49.7%；

顶层网络平均吞吐量为 0.935Mbps，带宽利用率为：62.3%；

c) Pattern1 瓶颈分析：

从实验结果可以看出，以网络的平均吞吐量来看：顶层>中间层>底层，宽带利用率上也是如此，，所以作为 core switch 的 c1 连接两个子网络，但带宽太小导致顶层网络成为整个网络的瓶颈，因此可以增大顶层带宽或增加 core switch，具体的改进方法在最后进行阐述和仿真。

3.2、模式二仿真

（1） 实现与代码注释

前面的网络拓扑结构和模式一是完全一样的，所以就不再重复叙述，主要的不同之处是在于通信模式，在这里我们选择 server n1 作为 sinkApp，其余的其他 server n2~n8 都与 n1 进行通信，这里只列举了 n2→n1 通信代码，其他的类似。

```
//select n1 as sinkApp
PacketSinkHelper packetSinkHelper1("ns3::TcpSocketFactory"
                         ,InetSocketAddress(csmaInterfaces1.GetAddress(1),sinkPort));
ApplicationContainer sinkApp1 = packetSinkHelper1.Install(csmaNode1.Get(1));
sinkApp1.Start(Seconds(0.0));
sinkApp1.Stop(Seconds(60.0));

//n2 --> n1
OnOffHelper client1("ns3::TcpSocketFactory",
                    InetSocketAddress(csmaInterfaces1.GetAddress(1),sinkPort));
client1.SetAttribute ("OnTime",StringValue("ns3::ConstantRandomVariable[Constant=50]"));
client1.SetAttribute ("OffTime",StringValue("ns3::ConstantRandomVariable[Constant=0]"));
client1.SetAttribute ("DataRate",DataRateValue (DataRate ("1.0Mbps")));
client1.SetAttribute ("PacketSize", UintegerValue (2000));
ApplicationContainer clientApp1 = client1.Install (csmaNode1.Get (2));
clientApp1.Start(Seconds (1.0 ));
clientApp1.Stop (Seconds (21.0));
```

（2） 实验结果分析

a) Pattern2 实验结果：

执行命令：./waf --run scratch/dcn2 对编写的仿真代码编译运行

```
fuli@fuli-virtual-machine:~/fox/ns-3/ns-allinone-3.27/ns-3.27$ ./waf --run scratch/dcn2
Waf: Entering directory `/home/fuli/fox/ns-3/ns-allinone-3.27/ns-3.27/build'
[ 955/2717] Compiling scratch/dcn2.cc
[2699/2717] Linking build/scratch/dcn2
Waf: Leaving directory `/home/fuli/fox/ns-3/ns-allinone-3.27/ns-3.27/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (4.140s)
fuli@fuli-virtual-machine:~/fox/ns-3/ns-allinone-3.27/ns-3.27$ ll
```

同时会产生如下的.Pcap 文件，我们可以使用 wireshark 软件对抓取的包进行分析。

```
-rw-rw-r--  1 fuli fuli  636674 4月    9 11:23 Pattern2_cama3-10-0.pcap
-rw-rw-r--  1 fuli fuli  167454 4月    9 11:23 Pattern2_cama3-11-0.pcap
-rw-rw-r--  1 fuli fuli  917058 4月    9 11:23 Pattern2_cama3-1-1.pcap
-rw-rw-r--  1 fuli fuli  167454 4月    9 11:23 Pattern2_cama3-12-0.pcap
-rw-rw-r--  1 fuli fuli  416298 4月    9 11:23 Pattern2_cama3-13-0.pcap
-rw-rw-r--  1 fuli fuli  167454 4月    9 11:23 Pattern2_cama3-14-0.pcap
-rw-rw-r--  1 fuli fuli  917308 4月    9 11:23 Pattern2_cama3-2-1.pcap
-rw-rw-r--  1 fuli fuli 2131432 4月    9 11:23 Pattern2_cama3-3-0.pcap
-rw-rw-r--  1 fuli fuli 2131092 4月    9 11:23 Pattern2_cama3-3-1.pcap
-rw-rw-r--  1 fuli fuli 1214718 4月    9 11:23 Pattern2_cama3-4-0.pcap
-rw-rw-r--  1 fuli fuli 1214878 4月    9 11:23 Pattern2_cama3-4-1.pcap
-rw-rw-r--  1 fuli fuli  334404 4月    9 11:23 Pattern2_cama3-5-0.pcap
-rw-rw-r--  1 fuli fuli  334564 4月    9 11:23 Pattern2_cama3-5-1.pcap
-rw-rw-r--  1 fuli fuli  583248 4月    9 11:23 Pattern2_cama3-6-0.pcap
-rw-rw-r--  1 fuli fuli  583408 4月    9 11:23 Pattern2_cama3-6-1.pcap
-rw-rw-r--  1 fuli fuli 3433818 4月    9 11:23 Pattern2_cama3-7-0.pcap
-rw-rw-r--  1 fuli fuli 1303070 4月    9 11:23 Pattern2_cama3-8-0.pcap
-rw-rw-r--  1 fuli fuli  578548 4月    9 11:23 Pattern2_cama3-9-0.pcap
-rw-rw-r--  1 fuli fuli  883068 4月    9 11:23 Pattern2_p2p-0-0.pcap
-rw-rw-r--  1 fuli fuli  883068 4月    9 11:23 Pattern2_p2p-0-1.pcap
-rw-rw-r--  1 fuli fuli  883068 4月    9 11:23 Pattern2_p2p-1-0.pcap
-rw-rw-r--  1 fuli fuli  883068 4月    9 11:23 Pattern2_p2p-2-0.pcap
```

对实验数据进行分析时，主要分别从三层网络抓取的包中选取一个，进行分析，主要的分析性能指标是：吞吐量；还有往返时间和窗口尺寸不做重点分析。

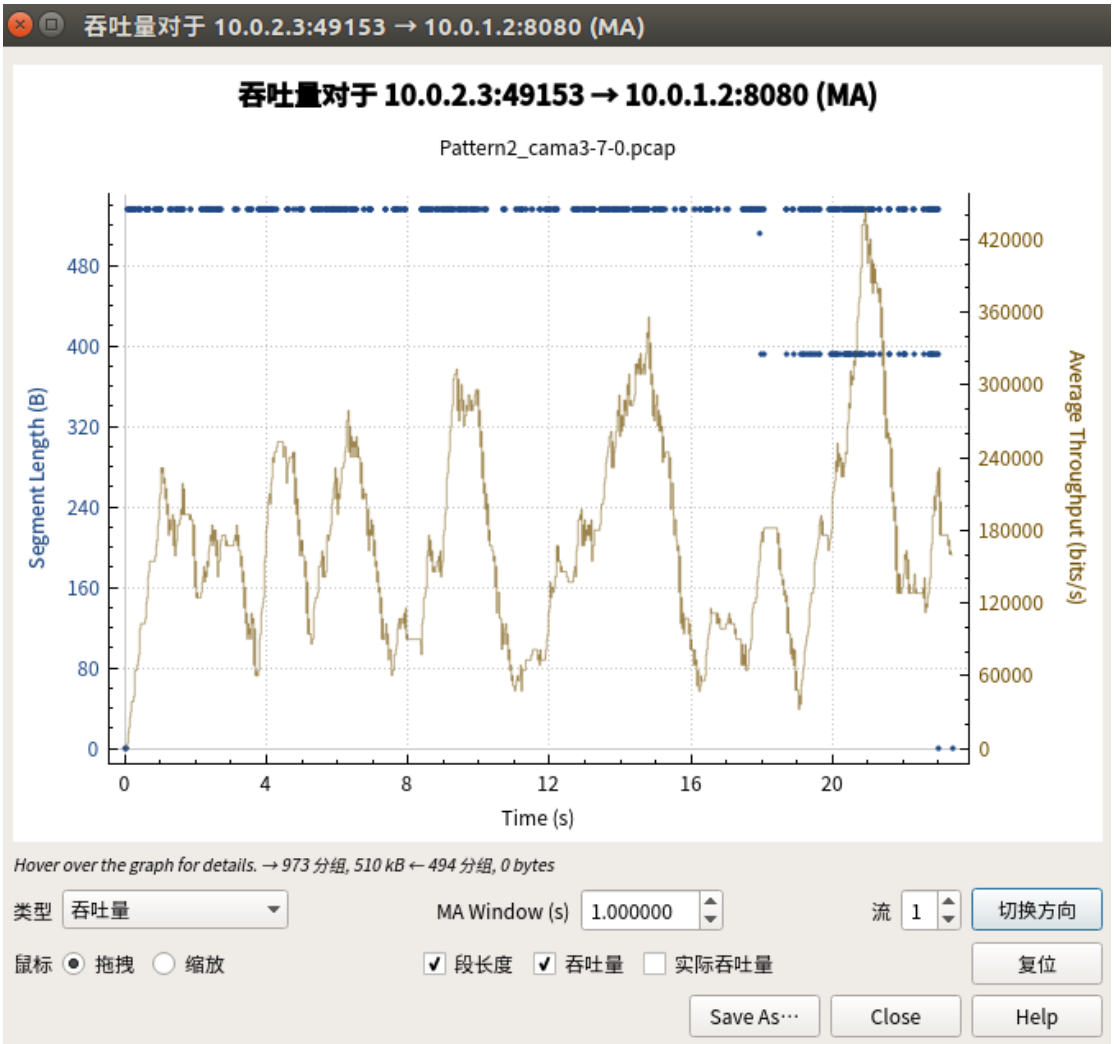
编写的代码是所有服务器 n2~n8 都向 n1 间通信，因此：

底层通过 server n1， n4， n7 即 pattern2_csma3-7-0.pcap，pattern2_csma3-10-0.pcap，pattern2_csma3-13-0.pcap 来估计分析；

中间层通过 ToR t1、t3 即 pattern2_csma3-3-0.pcap 和 pattern2_csma3-5-0.pcap 来估计分析；

顶层通过 core a1 即 pattern2-p2p-1-0.pcap 来进行估计分析；

对于 server n1 测量结果如下：



接口

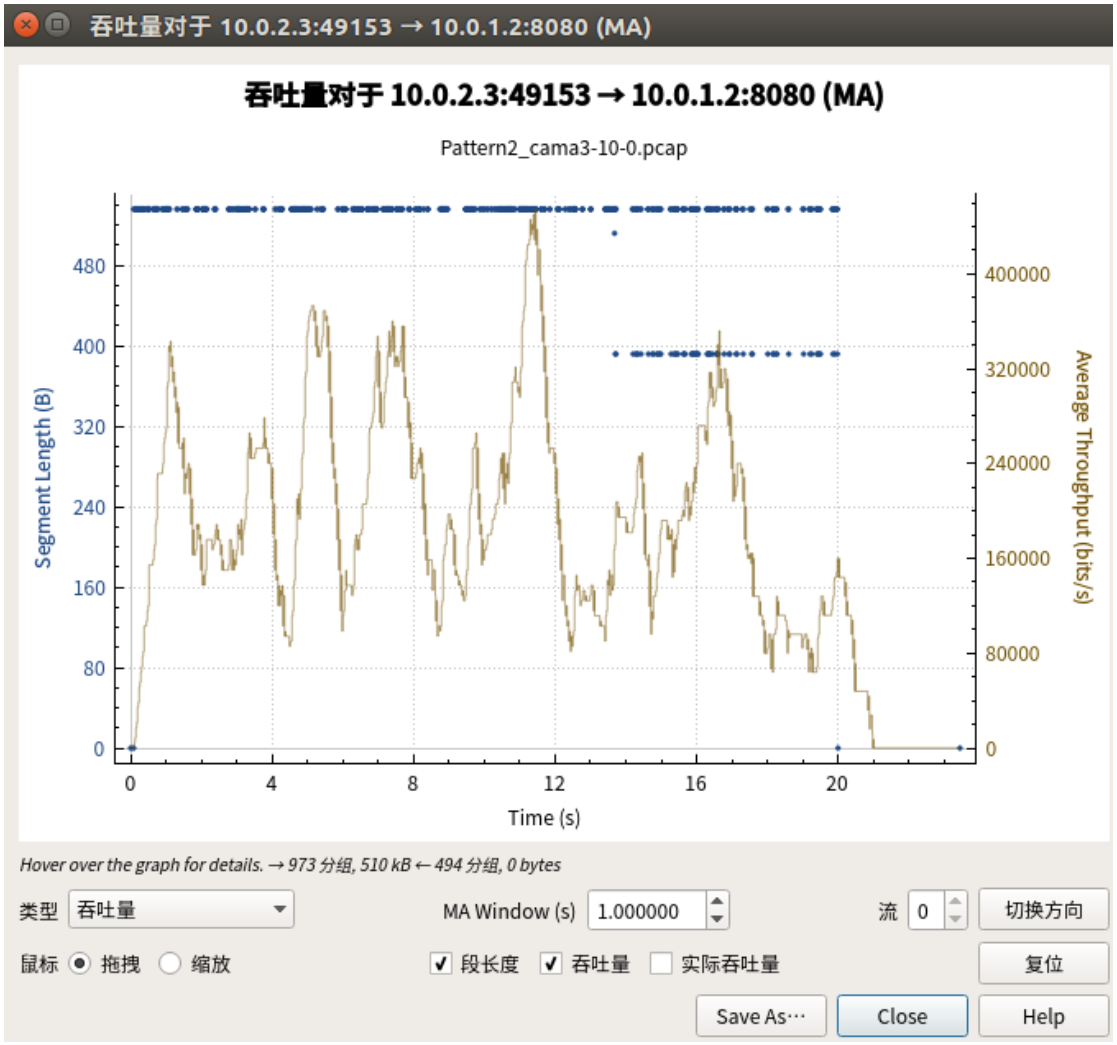
接口	丢弃分组	捕获过滤器	链路类型	分组大小限制
未知	未知	未知	Ethernet	65535 字节

统计

测量	已捕获	已显示	标记
分组	7951	7951 (100.0%)	—
时间跨度,s	27.171	27.171	—
平均 pps	292.6	292.6	—
平均分组大小,B	416	416	—
字节	3306578	3306578 (100.0%)	0
平均 字节/秒	121 k	121 k	—
平均 比特/秒	973 k	973 k	—

从上面两张图的仿真结果看出：是由服务器间组成的 Ethernets 网络，网络中节点 n1 的平均吞吐量是 973Kbit/s=0.973Mbps；

对于 server n4 测量结果如下：



接口

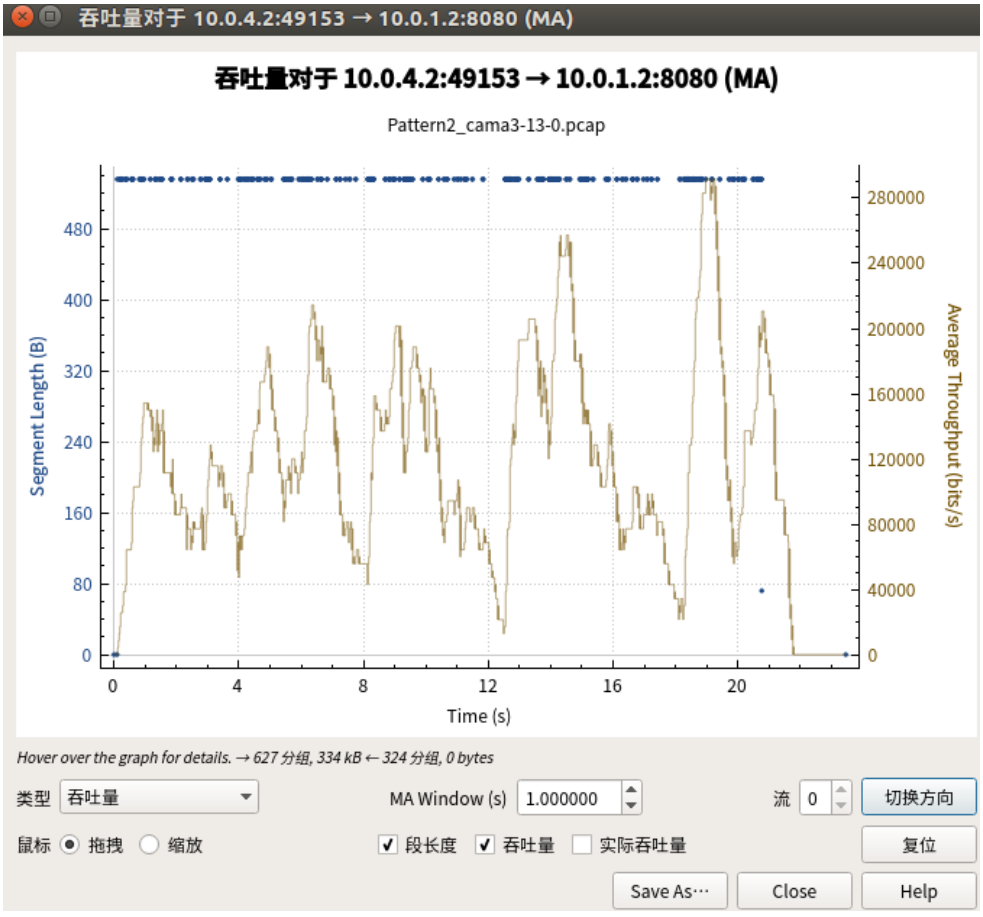
接口	丢弃分组	捕获过滤器	链路类型	分组大小
未知	未知	未知	Ethernet	65535 字

统计

测量	已捕获	已显示	标记
分组	1473	1473 (100.0%)	—
时间跨度,s	23.447	23.447	—
平均 pps	62.8	62.8	—
平均分组大小,B	416	416	—
字节	613082	613082 (100.0%)	0
平均 字节/秒	26 k	26 k	—
平均 比特/秒	209 k	209 k	—

从上面两张图的仿真结果看出：是由服务器间组成的 Ethernets 网络，网络中节点 n4 的平均吞吐量是 209Kbit/s=0.209Mbps；且该节点吞吐量波动，在仿真 21s 后，吞吐量为 0 停止通信，与代码符合。

对于 server n7 测量结果如下：



接口

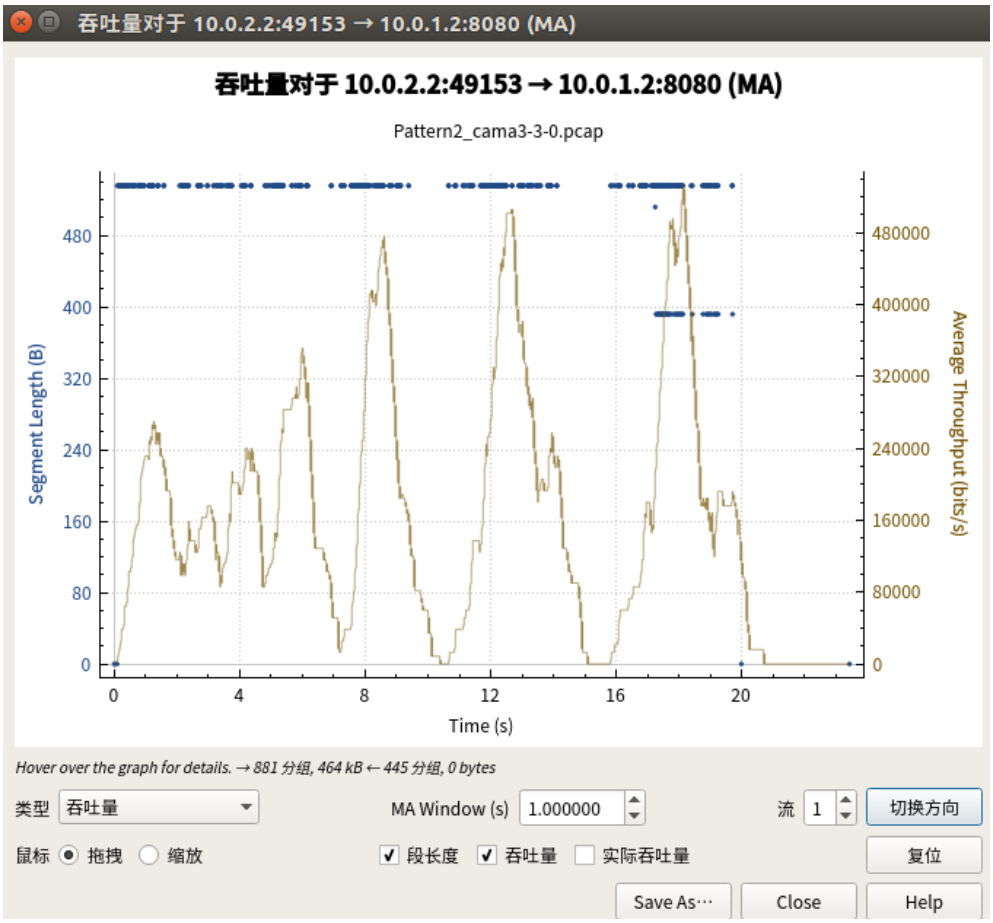
接口	丢弃分组	捕获过滤器	链路类型	分组大小限制
未知	未知	未知	Ethernet	65535 字节

统计

测量	已捕获	已显示	标记
分组	957	957 (100.0%)	—
时间跨度,s	23.460	23.460	—
平均 pps	40.8	40.8	—
平均分组大小,B	419	419	—
字节	400962	400962 (100.0%)	0
平均 字节/秒	17 k	17 k	—
平均 比特/秒	136 k	136 k	—

从上面两张图的仿真结果看出：是由服务器间组成的 Ethernets 网络，网络中节点 n7 的平均吞吐量是 136Kbit/s=0.136Mbps；且该节点吞吐量波动，在仿真 21s 后，吞吐量为 0 停止通信，与代码符合。

对于中间层通过 ToR t1 测量结果如下：



接口

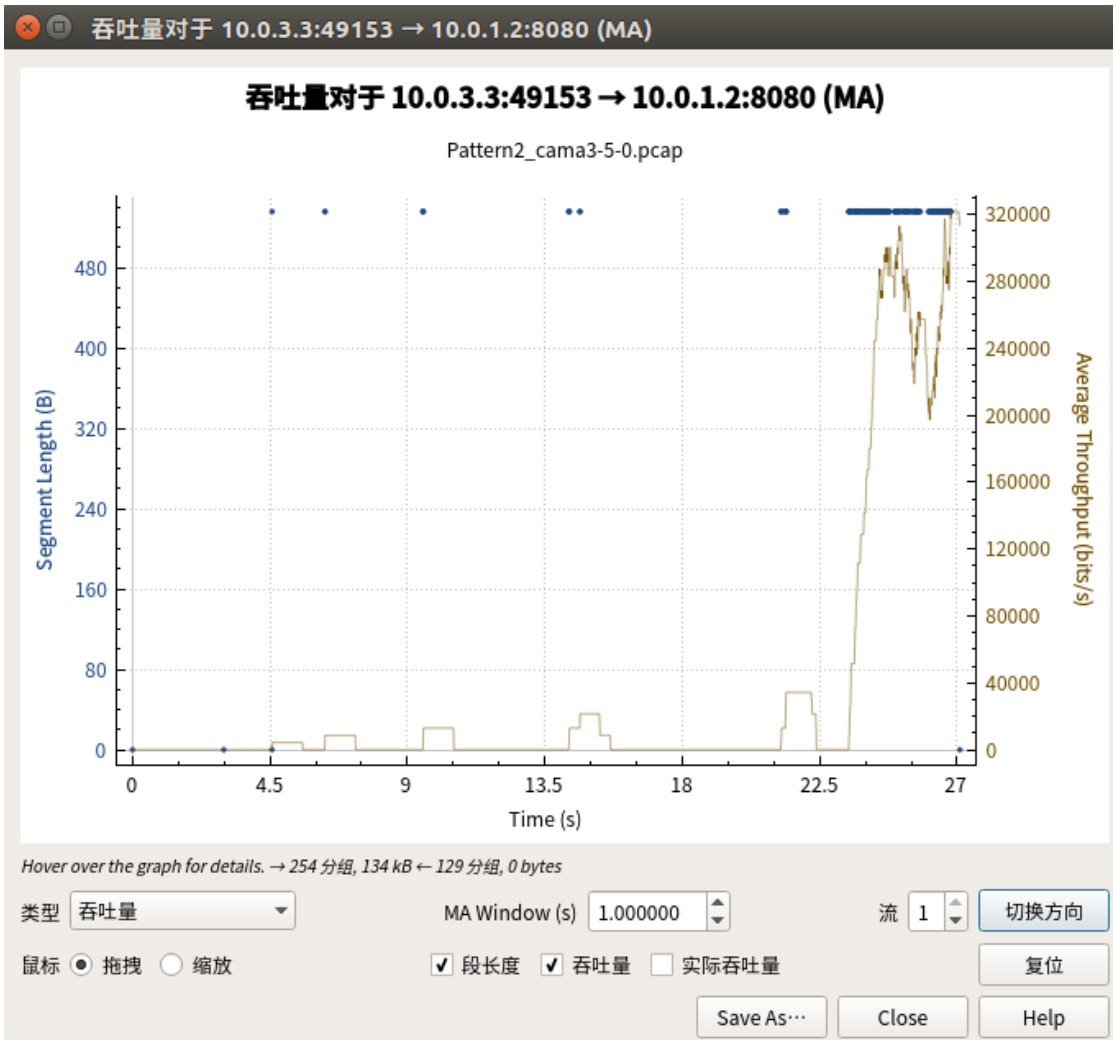
接口	丢弃分组	捕获过滤器	链路类型	分组大小限制
未知	未知	未知	Ethernet	65535 字节

统计

测量	已捕获	已显示	标记
分组	4900	4900 (100.0%)	—
时间跨度,s	27.172	27.172	—
平均 pps	180.3	180.3	—
平均分组大小,B	419	419	—
字节	2053008	2053008 (100.0%)	0
平均 字节/秒	75 k	75 k	—
平均 比特/秒	604 k	604 k	—

从上面两张图的仿真结果看出：是由 ToR 间组成的 Ethernets 网络，网络中节点 t1 的平均吞吐量是 604Kbit/s=0.604Mbps；且该节点吞吐量波动，在仿真 21s 后，吞吐量为 0 停止通信，与代码符合。

对于中间层通过 ToR t3 测量结果如下：



接口

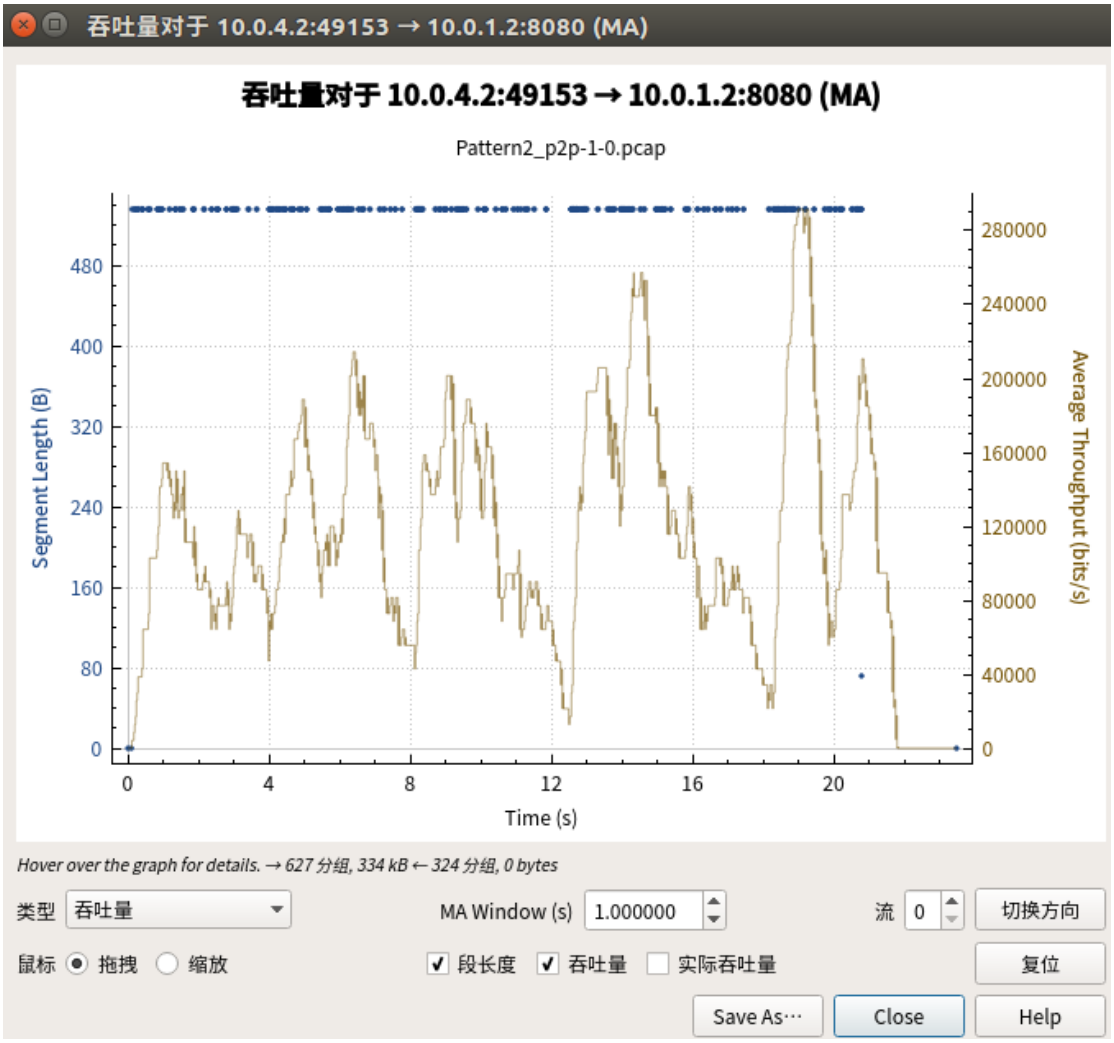
接口	丢弃分组	捕获过滤器	链路类型	分组大小限
未知	未知	未知	Ethernet	65535 字节

统计

测量	已捕获	已显示	标记
分组	772	772 (100.0%)	—
时间跨度,s	27.168	27.168	—
平均 pps	28.4	28.4	—
平均分组大小,B	417	417	—
字节	322028	322028 (100.0%)	0
平均 字节/秒	11 k	11 k	—
平均 比特/秒	94 k	94 k	—

从上面两张图的仿真结果看出：是由 ToR 间组成的 Ethernets 网络，网络中节点 t3 的平均吞吐量是 94Kbit/s=0.0944Mbps；

对于顶层通过节点 a1 测量结果如下：



接口

接口	丢弃分组	捕获过滤器	链路类型	分组大小限
未知	未知	未知	PPP	65535 字节

统计

测量	已捕获	已显示	标记
分组	2100	2100 (100.0%)	—
时间跨度,s	27.167	27.167	—
平均 pps	77.3	77.3	—
平均分组大小,B	404	404	—
字节	849444	849444 (100.0%)	0
平均 字节/秒	31 k	31 k	—
平均 比特/秒	250 k	250 k	—

从上面两张图的仿真结果看出：是由 aggregation 和 core 组成的 PPP 网络，网络的平均吞吐量是 250Kbit/s=0.250Mbps；

b) Pattern2 实验结果分析：

将上述得到的结果用图表汇总显示：

模式二仿真汇总

网络	结点	带宽	网络平均吞吐量
底层	Server n1	1.0 Mbps	0.973 Mbps
底层	Server n4	1.0 Mbps	0.209 Mbps
底层	Server n7	1.0 Mbps	0.136 Mbps
中间	ToR t1	1.0 Mbps	0.604 Mbps
中间	ToR t3	1.0 Mbps	0.0944 Mbps
顶层	Aggregation a1	1.5 Mbps	0.250 Mbps

从上面测得结果可以看出：

底层网络 n1 平均吞吐量为 0.973Mbps，带宽利用率为：97.3%；

底层网络 n4 平均吞吐量为 0.209Mbps，带宽利用率为：20.9%；

底层网络 n7 平均吞吐量为 0.136Mbps，带宽利用率为：13.6%；

中间网络 t1 平均吞吐量为 0.604Mbps，带宽利用率为：60.4%；

中间网络 t3 平均吞吐量为 0.0944Mbps，带宽利用率为：9.44%；

顶层网络 a1 平均吞吐量为 0.250Mbps，带宽利用率为：16.7%；

c) Pattern2 瓶颈分析：

从上面的实验仿真得到的数据分析可以知道，整个网络中吞吐量最大的是 server n1 所在网络，带宽利用率也比较高，而其余部分吞吐量很小且带宽利用率不高，因此在 pattern2 中，节点 n1 所在网络是整个网络的瓶颈，可以增大节点 n1 所在底层网络的带宽进行改进。

4、瓶颈改进

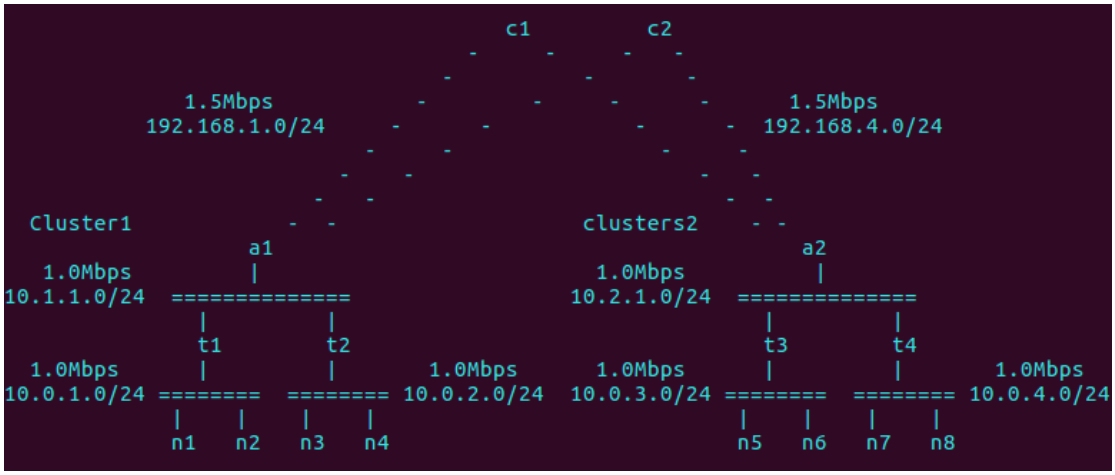
4.1、模式一瓶颈改进

从上面的瓶颈分析知道，作为 core switch 的 c1 连接两个子网络，但带宽太小导致顶层网络成为整个网络的瓶颈，因此可以增大顶层带宽或增加 core switch，重新编写仿真代码。

A. 增加 core

（1） 瓶颈改进代码注释

主要的改进地方是 core switch，其余的代码和之前模式一是一样的，因此这里只贴出改动部分。



主要修改代码中的网络拓扑结构，添加一个 core switch c2，并修改相应的安装网卡、设置传输速率为 1.5Mbps，延时为 500ns，安装网络协议和分配 IP 地址等等，和之前代码类似，不再讲解。

```
/*************网络拓扑部分*************/
//add 1 core switch
//创建2个aggregation:a1,a2 core1->a1 core1->a2 ptpNodes
NodeContainer aggregation1,aggregation2;
aggregation1.Create(2);    //c1,a1
aggregation2.Add(aggregation1.Get(0));//C1
aggregation2.Create(1);  //a2

//core2 --> a1  core2 --> a2
NodeContainer aggregation3,aggregation4;
aggregation3.Create(1);//c2=aggregation3.Get(0)
aggregation3.Add(aggregation1.Get(1));//a1
aggregation4.Add(aggregation3.Get(0));
aggregation4.Add(aggregation2.Get(1));//a2
```

按照 PPT 上要求添加更多核心交换机并启用 ECMP 路由以改善网络：

```
/****调用全局路由，帮助建立网络路由****/
//全局路由管理器根据节点产生的链路通告为每个节点建立路由表
Config::SetDefault("ns3::Ipv4GlobalRouting::RandomEcmpRouting", BooleanValue(true));
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

（2） 瓶颈改进结果分析

a) 瓶颈改进实验结果：

执行命令：./waf --run scratch/dcn1_add_1core 对编写的仿真代码编译运行:

```
fuli@fuli-virtual-machine:~/fox/ns-3/ns-allinone-3.27/ns-3.27$ ./waf --run scratch/dcn1_add_1core
Waf: Entering directory `/home/fuli/fox/ns-3/ns-allinone-3.27/ns-3.27/build'
Waf: Leaving directory `/home/fuli/fox/ns-3/ns-allinone-3.27/ns-3.27/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (17.937s)
fuli@fuli-virtual-machine:~/fox/ns-3/ns-allinone-3.27/ns-3.27$
```

同时会产生如下的.Pcap 文件，我们可以使用 wireshark 软件对抓取的包进行分析。

```
fuli@fuli-virtual-machine:~/fox/ns-3/ns-allinone-3.27/ns-3.27$ ll *.pcap
-rw-rw-r-- 1 fuli fuli  681704 4月  16 18:32 cama3_add_1core-10-0.pcap
-rw-rw-r-- 1 fuli fuli  664092 4月  16 18:32 cama3_add_1core-11-0.pcap
-rw-rw-r-- 1 fuli fuli  609256 4月  16 18:32 cama3_add_1core-12-0.pcap
-rw-rw-r-- 1 fuli fuli 2703422 4月  16 18:32 cama3_add_1core-1-2.pcap
-rw-rw-r-- 1 fuli fuli  749722 4月  16 18:32 cama3_add_1core-13-0.pcap
-rw-rw-r-- 1 fuli fuli  681704 4月  16 18:32 cama3_add_1core-14-0.pcap
-rw-rw-r-- 1 fuli fuli  664092 4月  16 18:32 cama3_add_1core-15-0.pcap
-rw-rw-r-- 1 fuli fuli 2703422 4月  16 18:32 cama3_add_1core-2-2.pcap
-rw-rw-r-- 1 fuli fuli 1358474 4月  16 18:32 cama3_add_1core-4-0.pcap
-rw-rw-r-- 1 fuli fuli 1358634 4月  16 18:32 cama3_add_1core-4-1.pcap
-rw-rw-r-- 1 fuli fuli 1345292 4月  16 18:32 cama3_add_1core-5-0.pcap
-rw-rw-r-- 1 fuli fuli 1345452 4月  16 18:32 cama3_add_1core-5-1.pcap
-rw-rw-r-- 1 fuli fuli 1358474 4月  16 18:32 cama3_add_1core-6-0.pcap
-rw-rw-r-- 1 fuli fuli 1358634 4月  16 18:32 cama3_add_1core-6-1.pcap
-rw-rw-r-- 1 fuli fuli 1345292 4月  16 18:32 cama3_add_1core-7-0.pcap
-rw-rw-r-- 1 fuli fuli 1345452 4月  16 18:32 cama3_add_1core-7-1.pcap
-rw-rw-r-- 1 fuli fuli  609256 4月  16 18:32 cama3_add_1core-8-0.pcap
-rw-rw-r-- 1 fuli fuli  749722 4月  16 18:32 cama3_add_1core-9-0.pcap
-rw-rw-r-- 1 fuli fuli 2602926 4月  16 18:32 Pattern1_add_1core-0-0.pcap
-rw-rw-r-- 1 fuli fuli 2602926 4月  16 18:32 Pattern1_add_1core-0-1.pcap
-rw-rw-r-- 1 fuli fuli 2602926 4月  16 18:32 Pattern1_add_1core-1-0.pcap
-rw-rw-r-- 1 fuli fuli      24 4月  16 18:32 Pattern1_add_1core-1-1.pcap
-rw-rw-r-- 1 fuli fuli 2602926 4月  16 18:32 Pattern1_add_1core-2-0.pcap
-rw-rw-r-- 1 fuli fuli      24 4月  16 18:32 Pattern1_add_1core-2-1.pcap
-rw-rw-r-- 1 fuli fuli      24 4月  16 18:32 Pattern1_add_1core-3-0.pcap
-rw-rw-r-- 1 fuli fuli      24 4月  16 18:32 Pattern1_add_1core-3-1.pcap
```

采用和上述模式一一样的节点进行分析，具体的步骤一样，就不贴出每个节点仿真结果图，直接统计出下面的汇总表。

b) 瓶颈改进结果分析：

底层通过 n1→n5，即 pattern1_add_1core_n1-12-0.pcap 来估计分析；

中间层通过 ToR t1 即 pattern1_add_1core_t1-4-0.pcap 来估计分析；

顶层通过 a1 即 pattern1_add_1core_c1 -0-0.pcap 来进行估计分析；

增加 1 个 core switch

网络	结点	带宽	网络平均吞吐量
底层	Server n1	1.0 Mbps	0.487 Mbps
中间	ToR t1	1.0 Mbps	0.969 Mbps
顶层	Aggregation a1	1.5 Mbps	0.936 Mbps

从上面测得结果可以看出：

底层网络 n1 平均吞吐量为 0.487Mbps，带宽利用率为：48.7%；

中间网络 t1 平均吞吐量为 0.969Mbps，带宽利用率为：96.9%；

顶层网络 a1 平均吞吐量为 0.936Mbps，带宽利用率为：62.4%；

对比改进前后数据分析可以知道，增加一个 Core Switch 后，确实提高了整个网络的吞吐量，尤其是对于底层 server 和中间层来说，带宽利用率显著增加。但是从现在仿真结果来看，中间层和顶层带宽利用率还是很高，成为整个网络的新瓶颈。因此可以将 Core Switch 增加 3 个再试试。

增加 3 个 core switch

网络	结点	带宽	网络平均吞吐量
底层	Server n1	1.0 Mbps	0.491 Mbps
中间	ToR t1	1.0 Mbps	0.969 Mbps
顶层	Aggregation a1	1.5 Mbps	0.935 Mbps

从上面测得结果可以看出：

底层网络 n1 平均吞吐量为 0.491Mbps，带宽利用率为：49.1%；

中间网络 t1 平均吞吐量为 0.969Mbps，带宽利用率为：96.9%；

顶层网络 a1 平均吞吐量为 0.935Mbps，带宽利用率为：62.3%；

我们发现增加 3 个 core switch 和 1 个 core switch 相比，整个网络性能改善并不明显，对于底层 server 来说，仅仅提高一点点，而中间层和顶层甚至没有变化，这主要原因是上面提到的，中间层带宽利用率很高，成为整个网络的新瓶颈，因此可以考虑将中间层带宽增加到 1.5Mbps 再试试。

增加 3 core switch & 中间层 1.5Mbps

网络	结点	带宽	网络平均吞吐量
底层	Server n1	1.0 Mbps	0.702 Mbps
中间	ToR t1	1.5 Mbps	1.449 Mbps
顶层	Aggregation a1	1.5 Mbps	1.395 Mbps

从上面测得结果可以看出：

底层网络 n1 平均吞吐量为 0.702Mbps，带宽利用率为：70.2%；

中间网络 t1 平均吞吐量为 1.449Mbps，带宽利用率为：96.6%；

顶层网络 a1 平均吞吐量为 1.395Mbps，带宽利用率为：93%；

我们发现增加 3 个 core switch 并且将中间层的带宽增加为 1.5Mbps 后，整个网络性能又改善了不少，对于底层 server 来说，从 3 个 core 时的 49.1%提升到 70.2%，整个网络的带宽利用率都很高，负载也比之前好很多，但是顶层和中间层带宽利用率较高，带宽过小会成为整个网络的新瓶颈，因此想要再改进的话，可以考虑将中间层和顶层带宽增加到 2Mbps 再试试。

c) 瓶颈改进结论：

瓶颈改进对比

网络	节点	改进前带宽利用率	改进1 (+1 core)	改进2 (+3 core)	改进3 (+3 core &1.5M)	改进1提升	改进2提升	改进3提升
底层	n1	24.4%	48.7%	49.1%	70.2%	+24.3%	+24.7%	+45.8%
中间	t1	49.7%	96.9%	96.9%	96.6%	+47.2%	+47.2%	+46.9%
顶层	a1	62.3%	62.4%	62.3%	93%	+0.1%	0	+30.7%

从上面得到仿真结果，可以看出，通过增加 core switch，可以解决网络中出现的瓶颈问题，其中增加一个 core switch 后会发现中间层网络会成为瓶颈，以至于再增加 3 个 core 时，改进效果也并不怎么明显。当进行改进 3 时，将中级带宽从 1Mbps 增加到 1.5Mbps 后，再增加 3 个 core switch，整个网络性能会改进许多，但是顶层和中间层带宽利用率较高，带宽过小会成为整个网络的新瓶颈，因此想要再改进的话，可以考虑将中间层和顶层带宽增加到 3Mbps 再试试。

B. 增加顶层带宽

（1） 瓶颈改进代码注释

主要的改进地方是将 dcn1.cc 文件中的底顶层网络带宽从 1.5Mbps 改为 2.0Mbps，其余的代码和之前模式一是一样的，因此这里只贴出改动部分。

```
//设置传送速率和信道延迟
PointToPointHelper pointToPoint1,pointToPoint2;
pointToPoint1.SetDeviceAttribute ("DataRate", StringValue ("2.0Mbps"));
pointToPoint1.SetChannelAttribute ("Delay"   , TimeValue(NanoSeconds(500)) );
pointToPoint2.SetDeviceAttribute ("DataRate", StringValue ("2.0Mbps"));
pointToPoint2.SetChannelAttribute ("Delay"   , TimeValue(NanoSeconds(500)) );
```

（2） 瓶颈改进结果分析

a) 瓶颈改进实验结果：

执行命令：./waf --run scratch/dcn1_2Mbps 对编写的仿真代码编译运行:

```
fuli@fuli-virtual-machine:~/fox/ns-3/ns-allinone-3.27/ns-3.27$ ./waf --run scratch/dcn1_2Mbps
Waf: Entering directory `/home/fuli/fox/ns-3/ns-allinone-3.27/ns-3.27/build'
[ 955/2729] Compiling scratch/dcn1_2Mbps.cc
[2668/2729] Linking build/scratch/dcn1_2Mbps
Waf: Leaving directory `/home/fuli/fox/ns-3/ns-allinone-3.27/ns-3.27/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (11.743s)
fuli@fuli-virtual-machine:~/fox/ns-3/ns-allinone-3.27/ns-3.27$ ll *.pcap
```

同时会产生如下的.Pcap 文件，我们可以使用 wireshark 软件对抓取的包进行分析。

```
-rw-rw-r-- 1 fuli fuli 2622216 4月  17 17:20 Pattern1_2Mbps_c1-0-0.pcap
-rw-rw-r-- 1 fuli fuli 1375902 4月  17 17:20 Pattern1_2Mbps_n1-11-0.pcap
-rw-rw-r-- 1 fuli fuli 1375902 4月  17 17:20 Pattern1_2Mbps_n6-8-0.pcap
-rw-rw-r-- 1 fuli fuli 2723464 4月  17 17:20 Pattern1_2Mbps_t1-3-0.pcap
-rw-rw-r-- 1 fuli fuli 2723464 4月  17 17:20 Pattern1_2Mbps_t3-5-0.pcap
```

采用和上述模式一一样的节点进行分析，具体的步骤一样，就不贴出每个节点仿真结果图，直接统计出下面的汇总表。

b) 瓶颈改进结果分析：

顶层带宽为 2Mbps

网络	结点	带宽	网络平均吞吐量
底层	Server n1	1.0 Mbps	0.489 Mbps
中间	ToR t1	1.0 Mbps	0.969 Mbps
顶层	Aggregation a1	2.0 Mbps	0.936 Mbps

从上面测得结果可以看出：

底层网络 n1 平均吞吐量为 0.489Mbps，带宽利用率为：48.9%；

中间网络 t1 平均吞吐量为 0.969Mbps，带宽利用率为：96.9%；

顶层网络 a1 平均吞吐量为 0.936Mbps，带宽利用率为：46.8%；

对比改进前后数据分析可以知道，将顶层带宽增加为 2Mbps 后，确实提高了整个网络的吞吐量，尤其是对于底层 server 和中间层来说，带宽利用率显著增加。但是从现在仿真结果来看，中间层和顶层带宽利用率还是很高，成为整个网络的新瓶颈。因此可以将顶层带宽增加为 3Mbps 再试试。

顶层带宽为 3Mbps

网络	结点	带宽	网络平均吞吐量
底层	Server n1	1.0 Mbps	0.487 Mbps
中间	ToR t1	1.0 Mbps	0.969 Mbps
顶层	Aggregation a1	3.0 Mbps	0.936 Mbps

从上面测得结果可以看出：

底层网络 n1 平均吞吐量为 0.487Mbps，带宽利用率为：48.7%；

中间网络 t1 平均吞吐量为 0.969Mbps，带宽利用率为：96.9%；

顶层网络 a1 平均吞吐量为 0.936Mbps，带宽利用率为：31.2%；

我们发现将顶层带宽增加到 3Mbps 和之前的 2Mbps 相比，整个网络性能并没有改善，而中间层和顶层甚至没有变化，这主要原因是上面提到的，中间层带宽利用率很高，成为整个网络的新瓶颈，因此可以考虑将中间层带宽增加到 2.0Mbps 再试试。

顶层带宽为 3Mbps &中间层 2Mbps

网络	结点	带宽	网络平均吞吐量
底层	Server n1	1.0 Mbps	0.959 Mbps
中间	ToR t1	2.0 Mbps	1.923 Mbps
顶层	Aggregation a1	3.0 Mbps	1.848 Mbps

从上面测得结果可以看出：

底层网络 n1 平均吞吐量为 0.959Mbps，带宽利用率为：95.9%；

中间网络 t1 平均吞吐量为 1.923Mbps，带宽利用率为：96.1%；

顶层网络 a1 平均吞吐量为 1.848Mbps，带宽利用率为：61.6%；

我们发现将顶层带宽增加到 3Mbps，并且将中间层的带宽增加为 2.0Mbps 后，整个网络性能又改善了不少，对于底层 server 来说，从之前的 48.7%提升到 95.9%，整个网络的带宽利用率都很高，负载也比之前好很多，但是顶层和中间层带宽利用率较高。

c) 瓶颈改进结论：

网络	节点	改进前带宽利用率	改进1(顶层带宽2Mbps)	改进2(顶层带宽3Mbps)	改进3(顶层带宽3Mbps+中间2Mbps)	改进1提升	改进2提升	改进3提升
底层	n1	24.4%	48.9%	48.7%	95.9%	+24.5%	+24.3%	+71.5%
中间	t1	49.7%	96.9%	96.9%	96.1%	+47.2%	+47.2%	+46.4%
顶层	a1	62.3%	46.8%	31.2%	61.6%	-15.5%	-31.1%	-0.7%

从上面得到仿真结果，可以看出，通过增加底层网络的带宽，可以解决网络中出现的瓶颈问题，其中顶层带宽增加到 2Mbps 和 3Mbps 都会对瓶颈有所改进，但是当增加顶层带宽后会使中间层网络成为瓶颈，因此可以在增加顶层网络带宽时同时增加中间层带宽，得到的效果会最

好。仿真中出现顶层带宽利用率降低，是因为增加的带宽过大，要想改进的话可以换成顶层带宽 2Mbps 和中间层带宽 2Mbps，再仿真效果会更好。对于一个大型网络，需要多次分配网络带宽使网络达到负载平衡，达到利用率最高。

4.2、模式二瓶颈改进

从上面的实验仿真得到的数据分析可以知道，整个网络中吞吐量最大的是 server n1 所在网络，带宽利用率也比较高，而其余部分吞吐量很小且带宽利用率不高，因此在 pattern2 中，节点 n1 所在网络是整个网络的瓶颈，可以增大节点 n1 所在底层网络的带宽进行改进。

（1） 瓶颈改进代码注释

前面的网络拓扑结构和模式一是完全一样的，所以就不再重复叙述，主要的改进地方是将底层网络的带宽分别增加到 3Mbps 或 2Mbps，，其余的代码和之前模式二是一样的，因此这里只贴出改动部分。

```
//创建和连接CSMA设备及信道
CsmaHelper csma2,csma3;
csma2.SetChannelAttribute("DataRate", StringValue("1Mbps"));
csma2.SetChannelAttribute("Delay"   , TimeValue(NanoSeconds(500)) );
csma3.SetChannelAttribute("DataRate", StringValue("2Mbps"));
csma3.SetChannelAttribute("Delay"   , TimeValue(NanoSeconds(500)) );
```

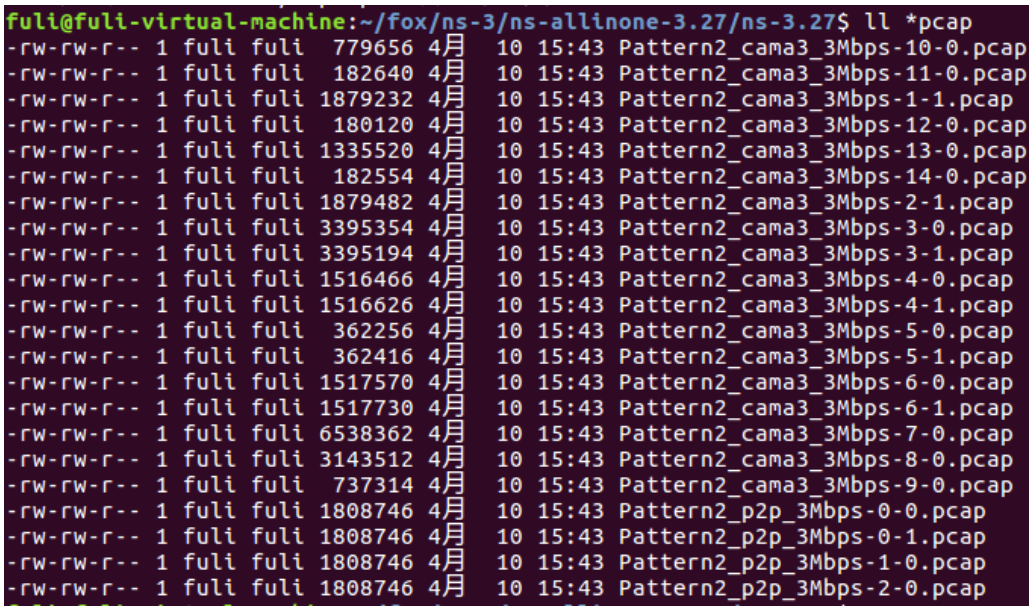
（2） 瓶颈改进结果分析

a) 瓶颈改进实验结果：

执行命令：./waf --run scratch/dcn2_3Mbps 对编写的仿真代码编译运行:

```
fuli@fuli-virtual-machine:~/fox/ns-3/ns-allinone-3.27/ns-3.27$ ./waf --run scratch/dcn2_3Mbps
Waf: Entering directory `/home/fuli/fox/ns-3/ns-allinone-3.27/ns-3.27/build'
[ 956/2723] Compiling scratch/dcn2_3Mbps.cc
[2711/2723] Linking build/scratch/dcn2_3Mbps
Waf: Leaving directory `/home/fuli/fox/ns-3/ns-allinone-3.27/ns-3.27/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (4.611s)
fuli@fuli-virtual-machine:~/fox/ns-3/ns-allinone-3.27/ns-3.27$ ll
```

同时会产生如下的.Pcap 文件，我们可以使用 wireshark 软件对抓取的包进行分析。



采用和上述模式二一样的节点进行分析，具体的步骤一样，就不贴出。

b) 瓶颈改进结果分析：

将上述得到的结果用图表汇总显示：

底层网络带宽加到 3Mbps

网络	结点	带宽	网络平均吞吐量
底层	Server n1	3.0 Mbps	1.877 Mbps
底层	Server n4	3.0 Mbps	0.245 Mbps
底层	Server n7	3.0 Mbps	0.438 Mbps
中间	ToR t1	1.0 Mbps	0.975 Mbps
中间	ToR t3	1.0 Mbps	0.104 Mbps
顶层	Aggregation a1	1.5 Mbps	0.518 Mbps

从上面测得结果可以看出：

底层网络 n1 平均吞吐量为 1.877Mbps，带宽利用率为：62.6%；

底层网络 n4 平均吞吐量为 0.245Mbps，带宽利用率为：8.17%；

底层网络 n7 平均吞吐量为 0.438Mbps，带宽利用率为：14.6%；

中间网络 t1 平均吞吐量为 0.975Mbps，带宽利用率为：97.5%；

中间网络 t3 平均吞吐量为 0.104Mbps，带宽利用率为：10.4%；

顶层网络 a1 平均吞吐量为 0.250Mbps，带宽利用率为：34.5%；

对比改进前后数据分析可以知道，底层网络带宽增加到 3Mbps 后，server n1 所在网络带宽利用率会下降，而其余部分吞吐量和带宽利用率都会有所提升，尤其是顶点网络 a1 和 t1 提升最大，但是由于将底层增加到 3Mbps，会使底层其余带宽利用率不高，而且使 t1 成为瓶颈，网络

负载不是很平衡，因此可以将底层带宽增加到 2Mbps 再试试。

底层网络带宽加到 2Mbps

网络	结点	带宽	网络平均吞吐量
底层	Server n1	2.0 Mbps	1.769 Mbps
底层	Server n4	2.0 Mbps	0.256 Mbps
底层	Server n7	2.0 Mbps	0.415 Mbps
中间	ToR t1	1.0 Mbps	0.974 Mbps
中间	ToR t3	1.0 Mbps	0.107 Mbps
顶层	Aggregation a1	1.5 Mbps	0.502 Mbps

从上面测得结果可以看出：

底层网络 n1 平均吞吐量为 1.769Mbps，带宽利用率为：88.5%；

底层网络 n4 平均吞吐量为 0.256Mbps，带宽利用率为：12.8%；

底层网络 n7 平均吞吐量为 0.415Mbps，带宽利用率为：20.8%；

中间网络 t1 平均吞吐量为 0.974Mbps，带宽利用率为：97.4%；

中间网络 t3 平均吞吐量为 0.107Mbps，带宽利用率为：10.7%；

顶层网络 a1 平均吞吐量为 0.502Mbps，带宽利用率为：33.5%；

c) 瓶颈改进结论：

瓶颈改进对比

网络	结点	改进前带宽利用率	改进 1（3Mbps）	改进 2（2Mbps）	改进 1 提升	改进 2 提升
底层	n1	97.3%	62.6%	88.5%	-34.7%	-8.8%
底层	n4	20.9%	8.17%	12.8%	-12.73%	-8.1%
底层	n7	13.6%	14.6%	20.8%	+1.0%	+7.2%
中间	t1	60.4%	97.5%	97.4%	+37.1%	+37%
中间	t3	9.44%	10.4%	10.7%	+0.96%	+1.26%
顶层	a1	16.7%	34.5%	33.5%	+17.8%	+16.8%

从上面得到仿真结果，可以看出，通过增加底层网络的带宽，可以解决网络中出现的瓶颈问题，其中底层带宽增加到 2Mbps 取得的效果要比增加到 3Mbps 要好一些，但是因为是 many-to-one traffic 模式，会使 n1，t1 和 a1 成为瓶颈，可以通过增加网络带宽来消除瓶颈。同时对那些 n4，n7 和 t3 等通信量不多的网络，可以减少带宽来增加带宽利用率，对于一个大型网络，需要多次分配网络带宽使网络达到负载平衡，达到利用率最高。

5、实验总结

通过本次实验使用 ns3 来模拟数据中心网络，基本熟悉了数据中心网络，对 servers、ToR switches、Aggregation switches 和 Core switches 构成的网络结构有了一个基础的认识，也熟悉了如何使用 ns3 来编写代码仿真网络，以及通过 wireshark 软件来分析抓取的包，并通过仿真分析网络中的吞吐量、往返时间、窗口尺寸等等，并发现系统可能的瓶颈，以及思考瓶颈产生的原因以及改善方案。

在此次实验中还遇到一些问题，如下，在模式一中添加 core 时，会有多条可能的路由路径，需要用到 ECMP 路由协议，但是在代码最后加上下图中的代码时，运行会出错，一开始一直找不到原因，最后在网上 Google 到，需要把这段代码放在 Top。但是按照解决方案，并不会报错了，但是抓取的包使用 wireshark 打开会是空的，最后请教助教，发现这是 ns3 里的一个 bug, 把报错那行的代码注释掉就可以。

```
fuli@fuli-virtual-machine:~/fox/ns-3/ns-allinone-3.27/ns-3.27$ ./waf --run scrat
ch/dcn1_add_1core
Waf: Entering directory `/home/fuli/fox/ns-3/ns-allinone-3.27/ns-3.27/build'
Waf: Leaving directory `/home/fuli/fox/ns-3/ns-allinone-3.27/ns-3.27/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (2.497s)
assert failed. cond="m_ecmpRootExits.size () <= 1", msg="Assumed there is at mos
t one exit from the root to this vertex", file=../src/internet/model/global-rout
e-manager-impl.cc, line=316
terminate called without an active exception
Command ['/home/fuli/fox/ns-3/ns-allinone-3.27/ns-3.27/build/scratch/dcn1_add_1c
ore'] terminated with signal SIGIOT. Run it under a debugger to get more informa
tion (./waf --run <program> --command-template="gdb --args %s <args>").
```

```
/****调用全局路由,帮助建立网络路由****/
//全局路由管理器根据节点产生的链路通告为每个节点建立路由表
Config::SetDefault("ns3::Ipv4GlobalRouting::RandomEcmpRouting",BooleanValue(true));
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```