计算机网络自选实验报告

ns-3 在多种网络环境下模拟 vStomp 协议

罗浩 1752547，李一珉 1752882
江宵汉 1752916，张子健 1752894

目录

第一章　实验简介

1.1　实验目的

本实验通过 ns-3 来模拟在实验室中难以搭建的网络基本结构（蜂窝 LTE 网络），并在应用层实现了 vStomp[1] 协议以支持类似于消息队列的聊天系统，并进行基本的可视化演示。通过本次实验，获得使用软件进行网络模拟的基本技能，并获得使用 Socket 实现一个简单应用层协议的体验。

1.2　实验设备与参考资料

- ns-3.30.1 软件包
- Stomp 协议规范与参考实现 http://stomp.github.io/
- 《开源网络模拟器 ns-3 - 架构与实践》
- 《ns-3 网络模拟器基础及应用》

[1] very Simple Text Oriented Messaging Protocol，将在后文进行详细描述

第二章　实验拓扑结构

2.1　网络结构简介

实验中，我们选择最具代表性的蜂窝网络 LTE 结构来作为虚拟网络环境，一个蜂窝网络的典型结构如图2.1所示。其中 UE 即用户设备，我们将从中随机选择一定数量的用户设备作为我们 Stomp 协议的客户端，eNB 为无线基站，PGW 为 PNG 网关，SGW 为服务网关，MME 为移动性管理组件。在网络的另一端，我们使用一个点对点网络连接 PGW 和模拟的远程主机，即我们 Stomp 协议的服务端。

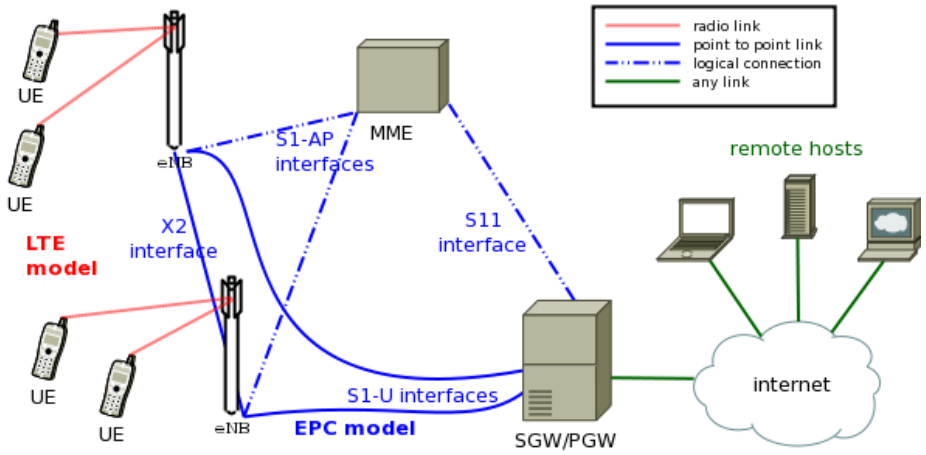

图 2.1: 网络拓扑图

2.2　网络参数设置

这里我们允许通过命令行参数对基站数量、单个基站对应的用户设备数量、参与 *Stomp* 的客户端主机数量以及是否进行数据报追踪进行设置。

我们在基站（eNB 结点）上应用静止的移动模型，空间布局为网格状排列，每行三个基站，基站之间水平/竖直距离均为 60 米。我们在用户设备（UE 结点）上应用随机移动模型，设置用户模型在一个以其对应的无线服务基站为中心的，边长为 40 米的正方形范围内进行随机移动，移动速度设为默认。

对于蜂窝网络，所有的用户设备都默认被分配在 7.0.0.0/8 这一网段中；对于远程主机和 PGW 构成的点对点网络，我们设置 100Gb/s 的数据传输速率、1500bytes 的最大传输单元以及 10ms 传输延迟，同时，将其分配到 1.0.0.0/8 这一网段中。

```
1    PointToPointHelper p2ph;
```

```
2     p2ph.SetDeviceAttribute ("DataRate", DataRateValue (
          DataRate ("100Gb/s")));
3     p2ph.SetDeviceAttribute ("Mtu", UintegerValue (1500));
4     p2ph.SetChannelAttribute ("Delay", TimeValue (
          MilliSeconds (10)));
5     NetDeviceContainer internetDevices = p2ph.Install (pgw,
          remoteHost);
6     Ipv4AddressHelper ipv4h;
7     ipv4h.SetBase ("1.0.0.0", "255.0.0.0");
8     Ipv4InterfaceContainer internetIpIfaces = ipv4h.Assign (
          internetDevices);
9     Ipv4Address remoteHostAddr = internetIpIfaces.GetAddress
          (1);
```

此外,在追踪数据报的过程中,我们观测到了 14.0.0.0/8、13.0.0.0/8 以及 10.0.0.0/8 网段，我们推测这些网段是本身在 LTE 蜂窝网络的内部实现中就已经被封装好的，用以给诸如 SGW、MME 等模拟网络结点的设备进行使用。在本实验，它们作为远程主机服务端和用户设备客户端交换数据报的必不可少的中介，我们并不需要具体考虑它们的底层实现细节，远程主机、PGW 和用户设备之间的交互才是我们关注的重点。

基于上述的网络结构，我们为远程主机配置了 7.0.0.0/8 网段的静态路由，为用户设备配其置默认路由为默认网关（即其对应的基站地址）。如下所示

```
1         Ipv4StaticRoutingHelper ipv4RoutingHelper;
2     Ptr<Ipv4StaticRouting> remoteHostStaticRouting =
          ipv4RoutingHelper.GetStaticRouting (remoteHost->
          GetObject<Ipv4> ());
3     remoteHostStaticRouting->AddNetworkRouteTo (Ipv4Address
          ("7.0.0.0"), Ipv4Mask ("255.0.0.0"), 1);
4
5     Ipv4InterfaceContainer ueIpIface;
6     ueIpIface = epcHelper->AssignUeIpv4Address (
          NetDeviceContainer (ueLteDevs));
7     for (uint16_t u = 0; u < ueNodes.GetN (); ++u)
8       {
9         Ptr<Node> ueNode = ueNodes.Get (u);
10        Ptr<Ipv4StaticRouting> ueStaticRouting =
              ipv4RoutingHelper.GetStaticRouting (ueNode->
              GetObject<Ipv4> ());
11        ueStaticRouting->SetDefaultRoute (epcHelper->
              GetUeDefaultGatewayAddress (), 1);
12      }
```

第三章　vStomp 协议

3.1　Stomp 协议简介

Stomp 协议是一种简单而易于实现的协议，经常被用于 **Websocket** 协议上从而实现 Web 端的实时通信功能。在我们本学期的其他项目中，为了实现 Web 端的在线聊天功能，我们采用的就是 Stomp 协议，因此对于本协议所受到的广泛支持[1]印象深刻，也正是这次接触使得我们决定在计算机网络实验的自选项目中尝试对本协议进行模拟。

Stomp 协议的全称是 **Simple Text Oriented Messaging Protocol**，正如这个名字中所暗示的那样，它的设计哲学就是简明与可交互性，用户可以自主进行一系列约定从而达成自己的目标。因此每个部分的具体信息都相当灵活，使得 Stomp 可以被用于各种各样的系统之中。

Stomp 的帧格式如下：

```
1 COMMAND
2 header1:value1
3 header2:value2
4 header3:value3
5
6 bodyEOF
```

其中，COMMAND 代表了这条消息的主要语义，而 header 中的键值对则补充了具体的语义，body 是要传递的信息，EOF 标志这帧的结束，例如 SEND、SUBSCRIBE、UNSUBSCRIBE、CONNECT 以及 DISCONNECT。具体的列表可以参看 Stomp 的官网，这里就不再赘述。我们可以想象成用户在连接之后就可以收听不同的频道，也可以向各个频道发送信息，而收听该频道的所有用户都将会接受到相应的信息。在这里我们使用我们另一个项目[2]的例子在展示 Stomp 协议。在用户登录之后，我们将会自动为他建立 WebSocket 连接，并完成与后端 RabbitMQ 的握手。具体的消息信息请查看截图 3.1。

[1]前端可以直接使用 JavaScript 中相应的库而成为 Stomp Client，后端 Spring Boot 框架中自带必要的支持，而类似于 RabbitMQ、Kafka 以及 ActiveMQ 这样的消息队列都提供了官方的插件成为高性能的 Stomp Server。

[2]该项目是关于设计并实现一个社交网站，其中的一个重要功能就是在线聊天。

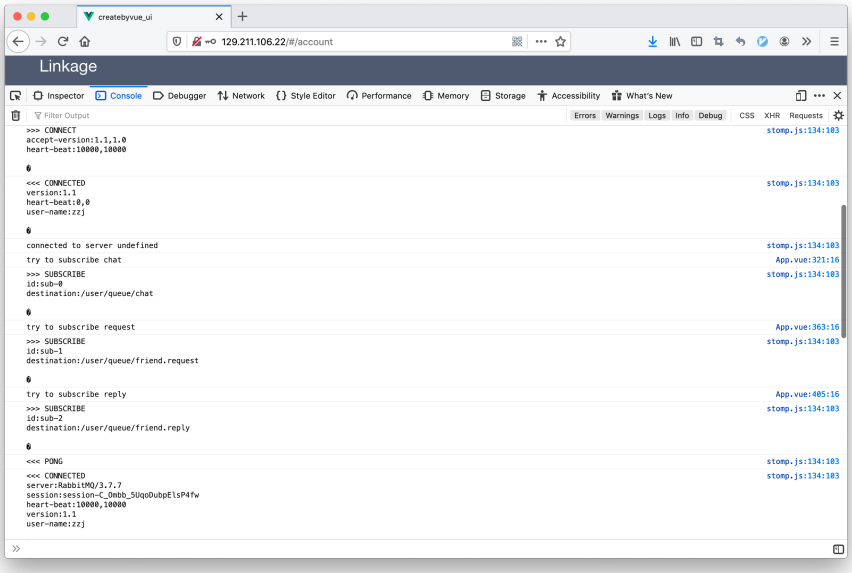

图 3.1: Stomp 协议浏览器截图

3.2　vStomp 协议详述

在我们的自选实验中，我们对 Stomp 协议中一些不常用的功能进行了删除，对一些较为复杂的约定进行了简化之后，得到了我们自己的协议，我们生动的为之命名 vStomp，very Simple Text Oriented Messaging Protocol。下面我们对各个命令做出说明

1. **CONNECT 命令**。在用户初次进入系统时，需要向 Server 注册，发送以下格式的帧

```
1 CONNECT
2 name:x
```

该命令对应的语义为：

(a) 名为 x 的用户向服务器注册。

(b) 服务器为该用户创建专门的频道，名字和用户名相同。

2. **CONNECTED 命令**。在服务器接受用户连接之后，向用户回复以下格式的帧

```
1 CONNECTED
```

该命令对应的语义为：

(a) 服务器中已经保存了用户的信息。

(b) 用户可以进一步发送消息。

3. **DISCONNECT 命令**。用户在完成会话之后，向服务器告知切断连接，发送以下格式的帧

```
DISCONNCT
name:x
```

该命令对应的语义为:

(a) 名为 x 的用户将断开连接。

(b) 服务器将该用户从所有收听了的频道中移除。

4. **SUBSCRIBE 命令**。用户在完成连接之后，可以收听一些频道，发送以下格式的帧

```
SUBSCRIBE
channel:net
name:x
```

该命令对应的语义为:

(a) 名为 x 的用户收听名为 net 的频道。

5. **UNSUBSCRIBE 命令**。用户可以选择取消收听一些频道，发送以下格式的帧

```
UNSUBSCRIBE
channel:net
name:x
```

该命令对应的语义为:

(a) 名为 x 的用户取消收听名为 net 的频道。

6. **SEND 命令**。用户可以向频道发送一些信息，发送以下格式的帧

```
SEND
target:net

I love computer network. Challenging and interesting.
```

该命令的语义为:

(a) 向 net 频道发送信息: "I love computer network. Challenging and interesting."

(b) 所有收订本频道的用户都将收到本条信息。

可以看到，我们的 vStomp 协议可以支持基础的订阅者-发布者模式，我们的例子比较偏向于聊天，单人聊天在 CONNECT 命令之后就自动具备，而群聊可以通过共同收听某个频道来达成。虽然在我们的实现中信息都暂时是 UTF-8 的文本信息，但是拓展到二进制协议也是非常方便的。

第四章　vStomp 实现

下面叙述协议的整体工作方式。在物理层连接搭建完毕之后，我们会在其中一台机器上安装 Server，在其他的各个机器上安装 Client。使用 Simulator::Schedul() 函数来预约在模拟开始之后分别开启 Server 和 Client 服务。Server 开启之后将会有一个空的哈希表，键是频道的名字，值是收听该频道的用户的集合，具体表达为类图 4.1 中 vStompServer 具有的类型为 std::map<std::string,std::set<std::shared_ptr<ClientStub>>>> 的变量。而后随着计划的时间到来，Client 向 Server 预先设置好的信息之后，Server 将依照前面我们所描述的协议内容进行处理，并始终保持上述变量的正确性，从而可以向应当接受某条信息的用户发送信息。

我们使用 ns-3 提供的类 socket 接口对本协议进行了实现，虽然我们在实验的过程中可以感受到 ns-3 可能并不是为了应用层的协议而创造的[1]，但是我们仍旧克服困难，对 vStomp 协议进行了实现。基本的类图如下所示

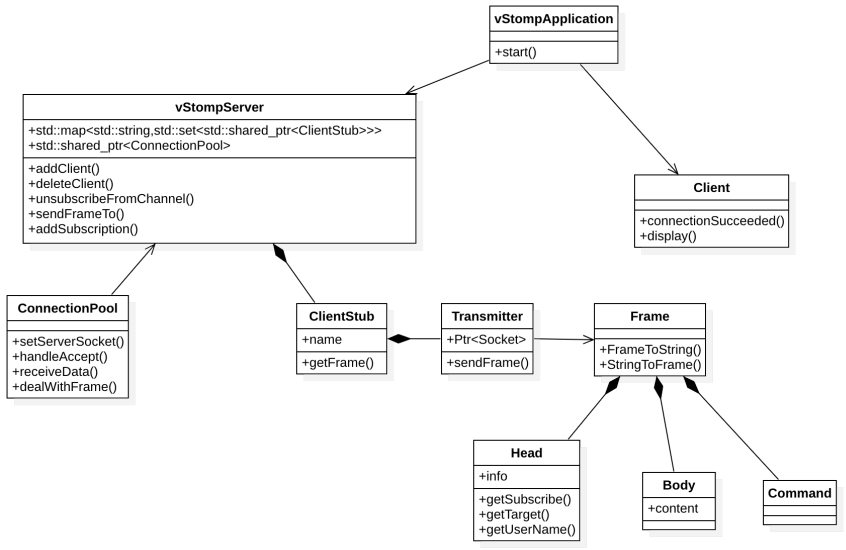

图 4.1: vStomp 基本类型

在我们的系统开始运行之后，将会进行类似于下图的输出：

[1]这一点在 Packet 类很难支持包含真正内容而可以很轻松的指定大小可以看出，ns-3 可能更适合进行对网络下面几层的研究

图 4.2: 系统输出

其中可以清晰看到我们发送的帧的格式与内容。这些内容都是真实发送的，使用的 NS_LOG_INFO 来进行的输出，具体处理可以到代码中查看。限于篇幅，不对每个类进行单独介绍，但是对于较为重要的设计我们将进行必要的描述。

4.1 Frame

Frame 类就是我们的 vStomp 帧在内存中的具体表现，我们书写了它的序列化与反序列化函数，分别是 FrameToString() 与 StringToFrame()，这两个函数的具体实现可以查看代码 frame.h。与之相关的类型定义如下所示：

```
enum Command{
 CONNECT,
 CONNECTED,
 SUBSCRIBE,
 SEND,
 DISCONNECT,
 UNSUBSCRIBE,
};

class Head{
 public:
   std::string getSubscribe();
   std::string getTarget();
   std::string getUserName();
 private:
    std::map<std::string, std::string> info;
};

```

```
19 class Body{
20  public:
21    std::string content;
22 };
23 class Frame{
24  public:
25    Command command;
26    Head head;
27    Body body;
28 };
```

4.2 ConnectionPool

为了适应 ns-3 中大量的回调机制，我们将服务器做了必要的切分来控制复杂性。有关用户的所有信息都是放置在 vStompServer 中的（具体表现为它内部存储的那个哈希表，记录这 channel 名称与该 channel 下的用户集合，而用户在我们的 Server 中表达被为 ClientStub），而 ConnectionPool 则会负责具体应对用户的连接，它将会取出用户发送的信息，并根据用户的命令来调用 vStompServer 中的函数，从而完成相应的目标。我们可以在 ConnectionPool 最主要的函数中清晰看到上设计方式

```
1  void dealWithFrame(Frame frame,Ptr<Socket> socket){
2   NS_LOG_INFO(std::to_string(Simulator::Now().GetSeconds()) +
        "s ["+ CLASSNAME +"]:     " + "Parse the frame");
3   switch ((frame.command))
4   {
5    case CONNECT:{
6           NS_LOG_INFO(std::to_string(Simulator::Now().
                GetSeconds()) +"s ["+ CLASSNAME +"]:     " + "
                This is a connect message");
7           Transmitter transmiter(socket);
8           std::shared_ptr<ClientStub> client(new ClientStub(
                transmiter,frame.head.getUserName()));
9           server->addClient(client);
10          Frame frame;
11          frame.command = CONNECTED;
12          client->getFrame(frame);
13   }break;
14   case SUBSCRIBE:{
15          NS_LOG_INFO(std::to_string(Simulator::Now().
                GetSeconds()) +"s ["+ CLASSNAME +"]:     " + "
                This is a subscribe message");
16          server->addSubscription(frame.head.getUserName(),
                frame.head.getSubscribe());
```

```
17  }break;
18  case SEND:{
19          NS_LOG_INFO(std::to_string(Simulator::Now().
                GetSeconds()) +"s [" + CLASSNAME +"]:     " + "
                This is a send message");
20          server->sendFrameTo(frame.head.getTarget(), frame);
21  }break;
22  case DISCONNECT:{
23          NS_LOG_INFO(std::to_string(Simulator::Now().
                GetSeconds()) +"s [" + CLASSNAME +"]:     " + "
                This is a disconnect message");
24          server->deleteClient(frame.head.getUserName());
25  }break;
26  case UNSUBSCRIBE:{
27          NS_LOG_INFO(std::to_string(Simulator::Now().
                GetSeconds()) +"s [" + CLASSNAME +"]:     " + "
                This is a unsubscribe message");
28          server->unsubscribeFromChannel(frame.head.
                getUserName(), frame.head.getSubscribe());
29  }break;
30  default:
31  break;
32  }
33 }
```

下面再通过当服务器收到用户的 SUBSCRIBE 命令时的表现以及 DISCONNECT 命令时所做出的操作来进一步的展示我们的系统

```
1  void addSubscription(std::string userName, std::string
       channelName){
2  // the client automatically subscribe to a channel with the
       name of himself
3    std::set<std::shared_ptr<ClientStub>> temptClient =
         clientPool[userName];
4    std::shared_ptr<ClientStub> client = *(temptClient.begin()
         );
5    std::set<std::shared_ptr<ClientStub>> channelSubsribers =
         clientPool[channelName];
6    channelSubsribers.insert(client);
7    clientPool[channelName] = channelSubsribers;
8    NS_LOG_INFO(std::to_string(Simulator::Now().GetSeconds())
         +"s [" + CLASSNAME +"]:     " + userName +" is
         add to group " + channelName);
9  }
```

```
10 void deleteClient(std::string clientName){
11   for(auto iter = clientPool.begin();iter!=clientPool.end()
       ;){
12    if(iter->first == clientName){
13     clientPool.erase(iter ++ );
14    }else{
15     std::set<std::shared_ptr<ClientStub>> subscribers = iter
         ->second;
16     for(auto subsIter = subscribers.begin();subsIter!=
         subscribers.end();){
17      if((*subsIter)->getName()==clientName){
18       NS_LOG_INFO(std::to_string(Simulator::Now().GetSeconds
           ()) +"s ["+ CLASSNAME +"]: " +
           clientName + " unsubscribe " + iter->first);
19       subscribers.erase(subsIter++);
20       iter->second = subscribers;
21       break;
22      }else{
23       subsIter ++;
24      }
25    }
26   iter ++;
27   }
28  }
29  NS_LOG_INFO(std::to_string(Simulator::Now().GetSeconds()) +
      "s ["+ CLASSNAME +"]: " + clientName + " is
      leaving.");
30 }
```

第五章　实验心得

首先感谢金老师在这 2019-2020 学年秋季学期计算机网络与计算机网络实验课程中的指导和帮助。

这次自选实验标志着我们的计算机网络实验课程的结束，与前面所做的 6 次物理实验不同，最后这次实验我们使用的是虚拟的 ns-3 软件包来进行的。我们克服了一些困难，也获得了不少的收获。最主要的问题在于

- 语言的障碍。虽然我们在大一下学期有专门学习过一学期的 C++ 语言，但是在课程结束之后就几乎没有再使用过，而 C++ 本身又是如此复杂而令人困惑的。在完成 ns-3 自选实验的过程中，我们使用了不少特别的语言机制，其中最重要的就是回调函数的函数指针，我们确实感到陌生。在经过本次实验之后，我们有重新了解了有关的知识，增进了对 C++ 的理解和掌握。

- 对如何实现一个协议的困惑。虽然现在想起来较为直观，无非就是对帧做序列化和反序列化，然后根据语义要求来进行必要的操作，但是在刚刚开始的时候我们确实对此感到迷茫。最终我们在 Stomp 的官网上找到了有关 Stomp 协议的开源实现，里面有一个非常简单的叫做 Gozirra（http://www.germane-software.com/software/Java/Gozirra/），是一个 Java 的实现，我们对它进行了研究，并最终做出了我们的 vStomp 实现。