华中科技大学计算机学院
《计算机通信与网络》实验报告

班级 CS2209 姓名 章宏宇 学号 U202210755

项目	TCP/UDP 及拥塞控制（15%）	多路仿真控制协议（20%）	路由协议（20%）	组网实验（25%）	平时成绩（20%）	总分
得分						

教学目标达成情况一览表

课程目标	1（20%）	2（15%）	3（10%）	4（5%）	5（15%）	6（20%）	7（15%）	总分
得分								

教师评语：

教师签名：

给分日期：

目　录

1 实验一 TCP/UDP 及拥塞控制

这一章节我就只谈谈问题反馈和建议吧

1.1 其它需要说明的问题

首先作为第一次实验, 我认为难度把控很合理, 只需要我们理解文件构成, 小小地修改 ini 文件即可运行仿真。对仿真结果的 log 搜索 `cwnd` 和 `ssthresh`，解释当前处于何种阶段，`cwnd` 和 `ssthresh` 变化原因为何即可。

任务看起来很简单，但在实际上手实践时就很难受了。下面列出几个实验中的困难点：

- 拥塞避免算法网上资料太少，甚至我第一次找到的 CSDN 博客是错的，对着错误的博客理解半天，浪费时间。最后只能阅读 inet 源码来逆向工程，纯折磨
- 实验一开始给出的 ini 文件中有多处 `warning`，意为不正确使用被编译器忽略；而恰好有行是关于缓存队列长度的 `warning`，如下：

```ini
**.ppp[*].queue.frameCapacity = 10  # in routers
```

代码 1-1 《计算机通信与网络实践》实验指导手册 中的 ini 代码

这句话被编译器忽略导致无法更改缓存队列长度，也就很难实现丢包。当然这个问题在 v2.0 版本的指导手册中修复了。但修复得有点慢，而且并没有更新日志啥的，也无从得知更新了啥。。。

- 指引不够完善，而且 omnetpp 没有补全和提示，网上教程也十分稀少，导致想略微修改服务器类型都很难。最后还是找助教帮忙才实现的。

1.2 优化建议

这里就前面的问题给出优化建议：

- 希望能给出些拥塞算法指定参考资料，或者直接看我的整理笔记吧（当然不保证全对，可以提 issue）
- 已经修复了，感觉是 inet 版本不一致的问题，下次强制规定版本号比较好
- 希望指导手册添加 inet 官方文档和使用示例

2 实验二 多路仿真控制协议

这一章节我就只谈谈问题反馈和建议吧

2.1 其它需要说明的问题

第二次实验我个人体验也挺好，难度不高，有了实验一的基础，前两级任务都很简单；第三级实现 CSMA 的任务也不算难，只需要修改一个函数即可。我认为是所有实验中设计最好的实验。

但是在实践中还是遇到了一些问题，如下：

- 实验检查表和实际助教检查内容不符。比如第一级需要我们“停等式 ARQ 协议仿真代码实现并添加注释”，但实际上检查时不需要添加注释；还有“数据成功传输时和数据传输失败时的信息截图”在实际检查也不需要实现。
- CSMA 中 p-持续并未给出明确的定义，网上资料也良莠不齐，模棱两可。导致我在实现代码时很纠结。
- CSMA 实验给出的非持续性方案代码并不能很好地实现交替传输。当一个 host 占据信道后需要很长时间才能交出信道所有权，实际模拟时很难切换。
- CSMA 实验给出的代码将 `busy` 变量设为全局，并不会在切换 configuration 和 reset 时重置，导致重复模拟时信道一直 busy，必须强制 finish 或 rebuild 才行。

2.2 优化建议

这里逐条给出建议：

- 希望能提前确定检查内容。
- 希望能给出明确定义，或给出官方文档 RFC（当然能给中文版资料最好
- 希望能优化一下等待时间等参数
- 希望能将修改代码将 `busy` 变量定为局部变量

3 路由协议

这一部分算是我耗费时间最长的实验，也是最无语的实验了。

3.1 环境

omnetpp v5.5.1

inet v4.2.5

3.2 实验要求

- 第一级
  - 补全 OSPF 代码，理解代码的实现流程，添加注释，结合代码阐述协议内容
  - 使用配置 OSPF 协议的路由器，设计网络结构进行仿真测试，使各 PC 机能互相访问
- 第二级
  - 补全 RIP 代码，理解代码的实现流程，添加注释，结合代码阐述协议内容
  - 使用配置 RIP 路由协议的路由器，设计网络结构进行仿真测试，使各 PC 机能互相访问
- 第三级
  - 补全 BGP 代码，理解代码的实现流程，添加注释，结合代码阐述协议内容
  - 使用配置 BGP 路由协议的路由器，设计网络结构进行仿真测试，使各 PC 机能互相访问

3.3 协议的设计、仿真及结果分析

这里详细说说实验中我的记录结果，思路是对着 RFC、资料和输出 log 一条条理解原理。

3.3.1 OSPF

这里就设计、仿真结果分析来说说 OSPF 协议

3.3.1.1 设计

OSPF，全称 `Open Shortest Path First`，依靠相邻路由器频繁交换链路状态信息，最终所有路由器都建立一个相同的链路状态数据库。具体设计放在下小节结合仿真结果解释

3.3.1.2 仿真结果分析

这里跑的是“/inet/examples/ospfv2/simpletest”部分，网络结构如图 3-1 所示，其中 R1 右侧接口 ip 地址为 `192.168.60.1`，R2 左侧接口 ip 地址为 `192.168.60.2`

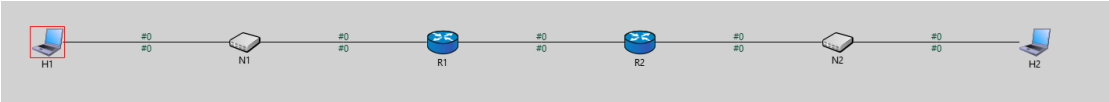

图 3-1　simpletest 仿真网络结构

恰好 OSPF 协议自带阶段，所以接下来将按照阶段分点描述

- **down 阶段**:

路由器通过 Interface 向外发送 Hellopacket 消息 发送给所有 OSPF 主机，其中组播地址为 `ip 224.0.0.5; mac 01:00:5E:00:00:05`

接收方收到 Hello 消息后，将发送方 ip 地址状态从 down 变为 init

- **init 阶段**:

在 init 阶段最重要的就是协商主从，参考下面的描述可知，`Router ID` 大的为 master，小的为 slave。这样做的目的是 master 统一控制时间，保证更新的有序。

```
The initialize(I), more (M) and master(MS) bits are set,  the contents
of the packet are empty, and the neighbor's Router ID is larger than the
router's own.  In this case the router is now Slave.  Set the master/slave
bit to slave, and set the neighbor data structure's DD sequence number to
                    that specified by the master.
```

代码 3-1　OSPF RFC 文档 中关于协商主从的描述

这里协商完主从，R1 的接口 ID 小，故 R1 为 slave，R2 为 Master。

init 阶段首先 master 发起信息交换

master 通过 Interface 向 slave 邻居发送 DDpacket: LSA (Link State Advertisement)。同时本机记录邻居状态从 init -> exchangestart

slave 收到后向 master 同样发送 DDpacket ,状态从 init -> exchangestart

init 阶段致敬 TCP 三次握手，完全一致。按照以下顺序和格式发送 LS 消息。

```
1  master -> slave: ddOptions=I M MS LSAHeader 空
2  slave -> master: ddOptions=I M MS LSAHeader 空
3  master -> slave: ddOptions=I M MS LSAHeader 空
```

- **exchangestart 阶段**:

该阶段也和三次握手类似。首先 slave -> master 发送 DDpacket, 更新 master 状态 exchange

```
1 ddOptions=_ _ __
2 LSAHeader (age: 0, type: RouterLsa, LSID: **192.168.60.1**, advertisingRouter: 192.168.60.1, seqNumber: -2147483646)
```

然后 master -> slave 发送 DDpacket, 更新 slave 状态 exchange

```
1 ddOptions=_ _ MS
2 LSAHeader (age: 0, type: RouterLsa, LSID: 192.168.60.2, advertisingRouter: 192.168.60.2, seqNumber: -2147483646)
```

slave 收到 exchangestart 阶段最后一条 packet,发送 DDpacket：

```
1 ddOptions=_ _ __
2 LSAHeader 空
```

- **exchange 阶段**：

在 exchangestart 中，master 发送 DDpacket 时，已经进入了 exchange 阶段，故会同时发送 exchange 阶段的数据 LSRpacket (Link State Request):

```
1 type=1, LSID=192.168.60.1, advertisingRouter=192.168.60.1
```

slave 收到 packet 后，发送 LSRpacket，更新状态为 loading：

```
1 type=1, LSID=192.168.60.2, advertisingRouter=192.168.60.2
```

master 收到 DDpacket 后,发送 DDpacket,更新状态 loading

```
1 ddOptions=_ _ __
2 LSAHeader 空
```

- **loading 阶段**：

slave 收到 DDpacket ,发现数据库不一致，故发送 LSUpacket：

```
1 LSAHeader (age: 1, type: RouterLsa, LSID: 192.168.60.1, advertisingRouter: 192.168.60.1, seqNumber: -2147483646)
2 bits=_ _ _
3 links:
4 ID=192.168.1.0, data=4294967040 (255.255.255.0), type= Stub , cost=1
```

master 收到 LSUpacket,发送 LSUpacket

```
1 LSAHeader (age: 1, type: RouterLsa, LSID: 192.168.60.2, advertisingRouter: 192.168.60.2, seqNumber: -2147483646)
2 bits=_ _ _
3 links:
4 ID=192.168.2.0, data=4294967040 (255.255.255.0), type= Stub , cost=1
```

slave 收到 LSUpacket,发送 LSUpacket

```
1 LSAHeader (age: 1, type: RouterLsa, LSID: 192.168.60.1, advertisingRouter: 192.168.60.1, seqNumber: -2147483645)
2 bits=_ _ _
3 links:
4 ID=192.168.1.0, data=4294967040 (255.255.255.0), type= Stub, cost=1
5 ID=192.168.60.2, data=3232250881 (192.168.60.1), type= PointToPoint, cost=2
6 ID=192.168.60.2, data=4294967295 (255.255.255.255), type= Stub, cost=2
```

同时 slave 会运行最短路算法，更新本机 routing table：

```
1 Results:
2 destination = 192.168.1.0, prefixLength = 24, nextHop = <unspec>, metric = 1, interface = eth0
3 destination = 192.168.60.2, prefixLength = 32, nextHop = 192.168.60.2, metric = 2, interface = eth1
```

master 收到 LSUpacket,发送 LSUpacket

```
1 LSAHeader (age: 1, type: RouterLsa, LSID: 192.168.60.2, advertisingRouter: 192.168.60.2, seqNumber: -2147483645)
2 bits=_ _ _
3 links:
4 ID=192.168.60.1, data=3232250882 (192.168.60.2), type=PointToPoint, cost=2
5 ID=192.168.60.1, data=4294967295 (255.255.255.255), type=Stub, cost=2
6 ID=192.168.2.0, data=4294967040 (255.255.255.0), type=Stub, cost=1
```

每次发送 LSUpacket 后,slave 和 master 都会更新自己的 routing table

当某一个路由器收到 LSUpacket 后，判断数据库无需更新，就不需要发送新的 LSUpacket，同时状态更新为 full

- **full 阶段**:

每隔 1s 对 LSA 进行 aging, 一旦超过 30s, 重新建立连接

3.3.2 RIP

这里也是就原理和仿真结果分析讲讲

3.3.2.1 原理

RIP 原理较为简单，主要有以下几点注意

- 每隔 30s 向相邻路由器发送路由表信息
- 通过 UDP 协议发送
- 上限 15 跳
- 存在路由环路问题,无穷计数问题
- metric 可自定义

3.3.2.2 仿真结果分析

这里我跑的是 “/inet/examples/rip/simpletest” 仿真。

因为 RIP 过于简单，故只简略描述仿真结果。其中 A ip 为 `192.168.60.2`，B ip 为 `192.168.60.1`

- A 发送 RIP request。因为是组播形式，所以不需要知道 arp。
- B 收到 RIP request,发送 RIP response
  - 初始化时 ip-mac 对应表为空,所以需要 ARP 请求,RIP response 进入 pending 状态

    ```
    1  ARP req: 192.168.60.2=? (s=192.168.60.1(0A-AA-00-00-00-03))
    ```

  - A 收到 ARP request 后,更新 arp cache

    ```
    1  Updating ARP cache entry: 192.168.60.1 <--> 0A-AA-00-00-00-03
    ```

    发送 ARP reply

    ```
    1  ARP reply: 192.168.60.2=0A-AA-00-00-00-04 (d=192.168.60.1(0A-AA-00-00-00-03))
    ```

  - B 收到 ARP reply 后,更新 arp cache,发送 RIP response
- A 收到 RIP response,更新 routing table

3.3.3 BGP

BGP 全称 `Border Gateway Protocol`

3.3.3.1 原理

每一个 AS（自治系统）要选择一个路由器作为该 AS 的“BGP 发言人”。两个 BGP 发言人通过一个共享网络连接在一起的。通过发言人之间的信息交换来实现寻路。

3.3.3.2 仿真结果分析

这里跑的是 “/inet/examples/bgpv4/BgpAndOspfSimple” 仿真

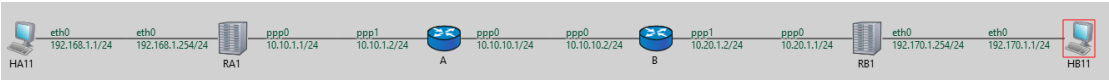

图 3-2　BGP 仿真网络结构

我们的目的是实现 `HA:192.168.1.1 -> HB:192.170.1.1` 的通信。

因为 BGP 是层次路由的一部分，所以首先得跑 IGP，实现域内寻路：

- 首先正常 OSPF 过程
- HA 发送 arp 请求：

```
1  inet::EthernetMacHeader, length = 14 B | ARP req: 192.170.1.1=?
   (s=192.168.1.1(0A-AA-00-00-00-03)) | inet::EthernetFcs, length =
   4 B
```

但是 RA1：arp cache miss，发送当前网关 mac 地址：

```
1  ARP reply: 192.170.1.1=0A-AA-00-00-00-01 (d=192.168.1.1(0A-
   AA-00-00-00-03))
```

- HA 收到 arp reply，发送 udp basic packet 向默认网关 A
- A 收到 udp basic packet，但是 routing 192.170.1.1 失败，故发送 icmp packet ICMP-error-#1-type3-code0。但往回发送时发现 routing 192.168.1.1 也失败，故 drop 该数据报
- 继续 ospf 过程，A 接收 RA1 的 LSU packet，更新 routing table
- 现在能发 icmp packet 了，HA 收到 icmp packet 了， 但是 app 层忽略 error：

```
1  Ignoring UDP error report ERROR
```

IGP 跑完后就该跑 **BGP** 了，接下来分点描述过程:

- B 启动监听 tcp 179 端口 (BGP 专用端口)

```
1 BgpNetwork.B.tcp.conn-11: Initializing module BgpNetwork.B.tcp.conn-11, stage 0
2 Tcp connection created for (inet::Request)PassiveOPEN
3 BgpNetwork.B.tcp.conn-11: Connection <none>:-1 to <none>:-1 on socketId=11 in INIT
4 BgpNetwork.B.tcp.conn-11: App command: OPEN_PASSIVE
5 BgpNetwork.B.tcp.conn-11: Starting to listen on: 10.10.10.2:179
6 BgpNetwork.B.tcp.conn-11: Transition: INIT --> LISTEN (event was: OPEN_PASSIVE)
```

发送 tcp packet，syn

```
1 inet::PppHeader, length = 5 B | inet::Ipv4Header, length = 20 B | inet::tcp::TcpHeader, 0->179 [Syn] Seq=4000000 Win=14336, length = 24 B | inet::PppTrailer, length = 2 B
```

- A 收到 syn，但还未启动监听，发送 rst + ack

```
1 inet::PppHeader, length = 5 B | inet::Ipv4Header, length = 20 B | inet::tcp::TcpHeader, 179->0 [Ack=4000001 Rst] Seq=0 Win=0, length = 20 B | inet::PppTrailer, length = 2 B
```

- B 收到 rst + ack，关闭连接
- A 启动监听 tcp 179 端口，同时启动监听

```
1 BgpNetwork.A.tcp.conn-9: Initializing module BgpNetwork.A.tcp.conn-9, stage 0
2 Tcp connection created for (inet::Request)PassiveOPEN
3 BgpNetwork.A.tcp.conn-9: Connection <none>:-1 to <none>:-1 on socketId=9 in INIT
4 BgpNetwork.A.tcp.conn-9: App command: OPEN_PASSIVE
5 BgpNetwork.A.tcp.conn-9: Starting to listen on: 10.10.10.1:179
6 BgpNetwork.A.tcp.conn-9: Transition: INIT --> LISTEN (event was: OPEN_PASSIVE)
```

发送 syn

- B 收到 syn，发送 syn + ack
- A 收到 syn + ack，发送 ack + bgp open Message。到此 TCP 连接建立完成

```
1 My AS: 65324
2 Hold time: 180s
3 BGP Id: 10.10.10.1
```

```
4  Optional parameters: empty
```

- B 收到 ack，发送 open Message

```
1  My AS: 65248
2  Hold time: 180s
3  BGP Id: 10.10.10.2
4  Optional parameters: empty
```

- B 收到 A 发的 open Message，发送 BGP Keep-alive message
- A 收到 B 发的 open Message，发送 BGP Keep-alive message
- B 收到 keep-alive message，发送 BGP Update Message。该包是 BGP 路由通告，包含前缀和属性：AS-PATH、NEXT-HOP

```
1   Withdrawn routes: empty
2   Path attribute 1: [len:4]
3     AS_PATH: 65248
4   Path attribute 2: [len:4]
5     NEXT_HOP: 10.10.10.2
6   Path attribute 3: [len:1]
7     ORIGIN: INCOMPLETE
8   Network Layer Reachability Information (NLRI):
9     NLRI length: 24
10    NLRI prefix: 10.20.1.0
```

```
1   Withdrawn routes: empty
2   Path attribute 1: [len:4]
3     AS_PATH: 65248
4   Path attribute 2: [len:4]
5     NEXT_HOP: 10.10.10.2
6   Path attribute 3: [len:1]
7     ORIGIN: INCOMPLETE
8   Network Layer Reachability Information (NLRI):
9     NLRI length: 16
10    NLRI prefix: 192.170.0.0
```

- A 收到 keep-alive message，发送 BGP Update Message

```
1   Withdrawn routes: empty
2   Path attribute 1: [len:4]
3     AS_PATH: 65324
4   Path attribute 2: [len:4]
5     NEXT_HOP: 10.10.10.1
```

```
  Path attribute 3: [len:1]
    ORIGIN: INCOMPLETE
  Network Layer Reachability Information (NLRI):
    NLRI length: 24
    NLRI prefix: 10.10.1.0
```

```
  Withdrawn routes: empty
  Path attribute 1: [len:4]
    AS_PATH: 65324
  Path attribute 2: [len:4]
    NEXT_HOP: 10.10.10.1
  Path attribute 3: [len:1]
    ORIGIN: INCOMPLETE
  Network Layer Reachability Information (NLRI):
    NLRI length: 16
    NLRI prefix: 192.168.0.0
```

- B 收到 update Message，更新 routing table

```
BgpNetwork.B.ipv4.routingTable:   add   route   B   10.10.1.0/24
gw:10.10.10.1  metric:1  if:ppp0 origin: INCOMPLETE ASlist: 65324
BgpNetwork.B.ipv4.routingTable:   add   route   B   192.168.0.0/16
gw:10.10.10.1  metric:1  if:ppp0 origin: INCOMPLETE ASlist: 65324
```

- A 收到 update Message，更新 routing table

```
BgpNetwork.A.ipv4.routingTable:   add   route   B   10.20.1.0/24
gw:10.10.10.2  metric:1  if:ppp0 origin: INCOMPLETE ASlist: 65248
BgpNetwork.A.ipv4.routingTable:   add   route   B   192.170.0.0/16
gw:10.10.10.2  metric:1  if:ppp0 origin: INCOMPLETE ASlist: 65248
```

到此，ipv4 routing table 同样实时更新。现在能完整 routing 了。

最后的最后，需要 `RB1 -> HA11` 的 arp 请求，得到 mac 地址，才能发送 udp packet

3.4 其它需要说明的问题

首先看到这个任务书，我真是无从下手，完全不知道怎么**补全**代码，对着 OSPF 官方文档 瞪眼瞧了一周也看不出啥。

后面快到检查时间了，迫不得已问同学，发现据小道消息（已经检查的同学说的），实际上完全不需要自己写代码，甚至检查时也只需要将各个协议所对的 `routing table` 调出来，讲讲寻路过程即可。

当然我那段时间刚好闲着，故花了一个周末的时间来对着输出 log 逆向。也算看出了点名堂，收获颇丰。

3.5 优化建议

这里我提几点建议：

- 指导手册里给出的代码中有几行注释：

```
1   if (signalID == interfaceCreatedSignal) {//接口创建
2         // configure interface for RIP
3         ie = check_and_cast<const InterfaceEntry *>(obj);
4         if (ie->isMulticast() && !ie->isLoopback()) {
5             // TODO
6         }
7     }
8     else if (signalID == interfaceDeletedSignal) {//接口删除
9         ie = check_and_cast<const InterfaceEntry *>(obj);
10        // TODO
11    }
```

  我一开始以为要我们补全的就是这些 `TODO`，结果后面发现 `inet` 源码里就有 `TODO`，我要是能补全这个那我直接给 `inet` 的 github 库 pull request 去了。。。 希望以后能提一嘴代码的来源。
- 同样的道理，网上关于 OSPF 和 BGP 介绍不多，而且很浅薄，希望有指定的中文参考资料。
- 希望以后能明确一下实验内容，不是用任务检查单的形式，而是将任务放在专门的任务书中，详细规定要实现的目标和实际检查内容。我觉得实际检查内容可以设置很多具体问题，比如：
  - OSPF 中的主从协商问题
  - OSPF 和 BGP 中有无三次握手
  - 各个专业名词的含义解释
  - BGP 和 RIP 都是用什么传输层协议，怎么体现在输出 log 中

  在检查时让学生ban几个没做的, 然后助教在剩下的问题中挑选几个问题来问, 我感觉会很有趣 :）

3.6 参考文献

- OSPF 技术连载
- 官方 OSPF RFC 文档
- 机翻 OSPF RFC 文档

4 组网实验

没做

4.1 其它需要说明的问题

4.2 优化建议

5 心得体会与建议

5.1 心得体会

首先，不论其他同学如何评价这个实验，我认为它瑕不掩瑜。OMNeT++能够仿真许多我们学习过但难以直观感受的协议，而通过这次实验，我对各类协议的运作机制有了更加深入且全面的理解。

实验的设计也十分合理，适度减少了代码编写和 OMNeT++ 操作的难度，将重点放在了微调 `ini` 文件、运行仿真以及分析结果上，这种设计思路我个人非常认可。

当然，作为首次改版的实验，也存在一些不足之处。细节上的欠缺往往对实验体验影响巨大，比如“实验任务”中的细微差异，就可能对整体效果产生较大影响。希望未来能够针对这些细节加以改进，进一步提升实验体验和效果。

5.2 建议

前面已经吐槽了不少，这里就不再多说了。放一张华小科的图，表达我的美好祝愿吧！希望计网实验能够不断完善，越改越好！: