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 CS1706 姓名 胡澳 学号 U201714761

项目	Socket 编程（40%）	数据可靠传输协议设计（20%）	CPT 组网（20%）	平时成绩（20%）	总分
得分					

教学目标达成情况一览表

目标 / 实验	目标 1	目标 2	目标 3	目标 4	目标 5	目标 6	合计
Socket 编程	/10	/7	/6	/5	/4	/8	/40
数据可靠传输协议设计	/5	/5	/3	/3	/4		/20
CPT 组网	/5	/6		/3	/4	/2	/20
小计	/20	/18	/9	/11	/12	/10	/80

教师评语：

教师签名：

给分日期：

目　　录

实验一 Socket 编程实验

1.1 环境

1.1.1 开发平台

CPU: Intel(R) Core(TM) i5-5350U CPU @ 1.80GHz
内存: 8GB DDR3
操作系统: Arch Linux
操作系统内核: x86_64 Linux 5.4.3-arch1-1
编译器: g++ (GCC) 9.2.0
图形界面: Qt version 5.13.2
自动构建工具: QMake version 3.1
GNU Make 4.2.1
url 处理: curl 7.67.0 (x86_64-pc-linux-gnu)
libcurl/7.67.0

1.1.2 运行平台

CPU: Intel(R) Core(TM) i5-5350U CPU @ 1.80GHz
内存: 8GB DDR3
操作系统: Arch Linux
操作系统内核: x86_64 Linux 5.4.3-arch1-1
编译器: g++ (GCC) 9.2.0
图形界面: Qt version 5.13.2
自动构建工具: QMake version 3.1
GNU Make 4.2.1
url 处理: curl 7.67.0 (x86_64-pc-linux-gnu)
libcurl/7.67.0
压力测试: wrk 4.1.0-4-g0896020 [epoll] Copyright (C) 2012 Will Glozer

1.2 系统功能需求

题目：编写一个支持多线程处理的 Web 服务器软件，要求如下：

第一级：

- 可配置 Web 服务器的监听地址、监听端口和虚拟路径。
- 能够单线程处理一个请求。当一个客户(浏览器，输入 URL：http://127.0.0.1/index.html)连接时创建一个连接套接字；
- 从连接套接字接收 http 请求报文，并根据请求报文的确定用户请求的网页文件；
- 从服务器的文件系统获得请求的文件。创建一个由请求的文件组成的 http 响应报文。(报文包含状态行+实体体)；
- 经 TCP 连接向请求的浏览器发送响应，浏览器可以正确显示网页的内容；
- 服务可以启动和关闭。

第二级：

- 支持多线程，能够针对每一个新的请求创建新的线程，每个客户请求启动一个线程为该客户服务；
- 在服务器端的屏幕上输出每一个请求的来源(IP 地址、端口号和 HTTP 请求命令行)；
- 支持一定的异常情况处理能力。

第三级：

- 能够传输包含多媒体(如图片)的网页给客户端，并能在客户端正确显示；
- 对于无法成功定位文件的请求，根据错误原因，作相应错误提示；
- 在服务器端的屏幕上能够输出对每一个请求处理的结果；
- 具备完成所需功能的基本图形用户界面(GUI)，并具友好性。

1.3 系统设计

本系统主要划分为如下三个模块进行设计，服务器端 socket 监听、对客户端请求的响应以及图形界面。

1. 服务器端 socket 监听。服务启动后，程序根据用户输入的监听 IP 地址、监听端口、最大连接数量创建一个 socket 对连接进行监听，另外，保存用户指定的虚拟路径。接下来，程序开始监听在指定 IP 和端口上的连接请求，当收到一个连接请求时，创建对应的 socket，然后创建一个新的线程，该线程调用响应客户端请求的模块对连接进行相应的处理。线程创建完成后，监听线程继续监听连接请求。

2. 响应客户端请求。监听线程接收到连接请求后，将创建响应的 socket 并创建响应客户端请求线程对该请求进行处理。此线程从 socket 中读取请求报文，并对请求报文进行解析，若报文内容不合法，则关闭 socket 并退出线程；若报文内容合法，则调用 curl 库中的函数对报文中的请求信息进行处理，对字符串的编码方式进行转化。然后根据转化后得到的请求文件路径寻找该文件，若该文件存在，则创建相应的 HTTP 响应报文首部，使用一个缓冲区不断读取

文件内容并将其写入到 socket 中。若文件不存在，则创建响应的 404 响应报文并写入到 socket 中。完成上述操作，即实现了对一次文件请求的响应，接下来，线程继续从 socket 中读取数据，若有新的请求，则重复上述操作，对请求进行处理；若 socket 被客户端关闭，则调用相应的处理函数关闭服务器端的 socket 并退出线程；若没有新的请求且 socket 没有被关闭，则线程阻塞，直到客户端发送新的请求或关闭 socket。

3. 图形界面。图形界面作为主线程启动，用户可通过图形界面输入监听 IP 地址、监听端口、虚拟地址等参数，当用户完成参数的输入后点击启动服务按钮，程序将调用相应的槽函数启动监听 socket 线程，同时阻止用户修改输入的参数或再次点击启动服务按钮。服务启动后，用户可通过终止服务按钮来关闭服务器，当用户点击终止服务按钮时，程序将调用相应的函数终止监听 socket 线程并等待该线程停止运行，同时，关闭用于监听的 socket，并调整图形界面中参数的可修改性。

通过上述三个模块的分别设计，并在适当的位置添加输出信息，即可完成待实现的功能。

1.4 系统实现

根据 1.3 节系统设计中对系统的模块划分以及模块功能描述，可得到如图 1.1 所示的模块间关系，对三个模块进行分别实现并定义好模块间的接口，即可实现系统的具体实现。

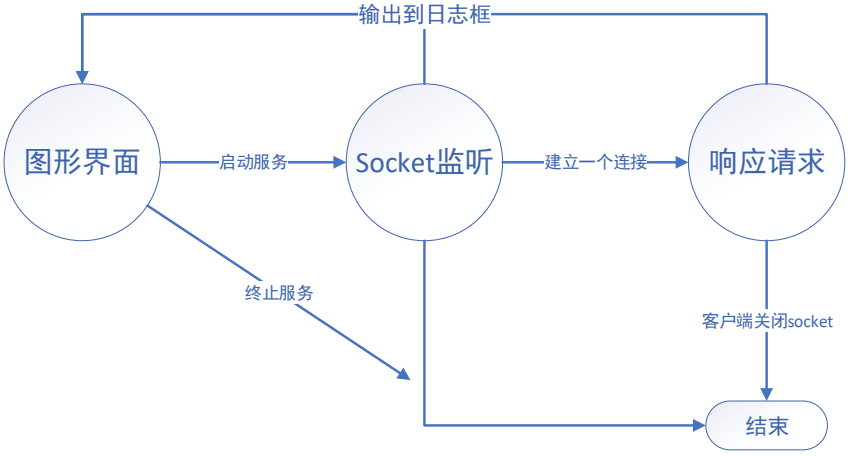

图 1.1 模块间关系

1. 图形界面的实现。本次实验中，我使用 QT 完成图形界面的实现。图形界面主要包括四个输入框，用于用户输入相应的参数，一个输出框，用于输出运行过程中的日志信息，以及三个按钮，分别用于启动服务、终止服务和清空日志框。图形界面的布局使用 Qt Designer 进行设计，并生成相应的 ui 文件。

接下来给输出文本框添加信号函数 sendMsg 和槽函数 showMsg 用于程序中将信息输出到图形界面的日志框中，当程序中调用信号函数时，将触发相应的槽函数，在槽函数中，我们将传入的字符串添加到输出框的最后，并判断输出框中的字符数量是否过长(例如大于 10000 个字符)，若输出框中字符过多，则将输出框中的内容清空。通过给输出日志框添加上述信号-槽

机制，即可实现将字符串输出到图形界面的功能。

最后，我们需要给三个按钮添加点击时触发的函数。对于启动服务按钮，当用户点击时，程序将在日志框中输出相应的信息，然后构造一个结构体，将用户向输入框中输入的参数转化为相应的格式并保存到该结构体中，为了使另外两个模块可以将数据输出到日志框中，还需要将图形界面的对象指针保存到该结构体中。然后创建一个线程，将上面构造的结构体作为参数传入到线程函数中，该线程将根据传入的参数进行 socket 监听工作。对于终止服务按钮，当用户点击时，程序向 socket 监听线程发送终止信号并等待该线程终止，当监听线程终止后，关闭监听使用的 socket 并在日志框中输出相应的信息。对于清空日志框按钮，当用户点击时，程序将对日志框调用相应的函数将其中的内容清空。

2. socket 监听的实现。当用户输入完参数并点击启动服务后，图形界面将把参数封装成一个结构体并将其作为参数创建 socket 监听线程。监听线程的流程图如图 1.2 所示。

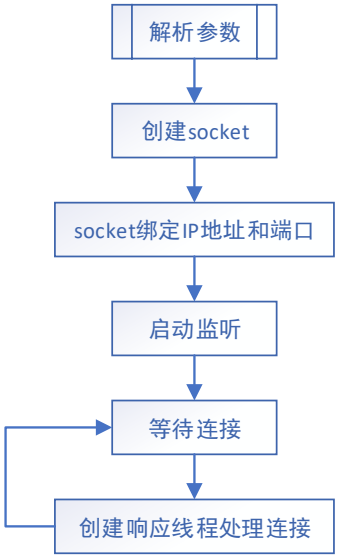

图 1.2 监听线程流程图

对于监听线程，首先，程序将从传入的参数结构体中提取出监听 IP 地址、端口以及虚拟地址等参数，其中，将虚拟地址作为全局变量保存，然后创建一个 ipv4 的流式套接字并将 IP 地址和端口号绑定到该套接字上，同时，线程将调用图形界面提供的信号函数 sendMsg，将操作的成功或失败的信息输出到日志框中。若任何一步操作出现错误，程序将输出相应的错误信息，并在关闭 socket 后退出进程。若监听使用的套接字创建成功，线程将在该 socket 上启动监听。接下来，线程等待来自客户端的连接，当获取到一个来自客户端的连接时，线程将创建相应的响应套接字并创建响应线程对该连接进行处理。处理线程创建完成后，监听线程立即回到上一步等待下一个连接的到来。通过上述过程，即可实现对客户端连接的正确响应，并且可以利用多线程并行处理多个连接。

由于监听线程可能会被来自图形界面的终止信号终止，因此调用 pthread 库中的相关函数，将该线程设置为可被其他线程终止，并将终止时机设置为异步终止方式。这样即可实现图形界面中通过点击终止服务按钮来终止监听线程。

3. 响应请求的实现。监听线程接收到一个请求后，将创建相应的响应套接字并创建响应线程对请求进行处理。响应线程中，首先将解析接收到的报文中的网络层信息，获取到其来源的 IP 地址及端口号，将线程 ID 作为线程标识，把上述报文来源信息输出到图形界面的日志框中。接下来，线程将从 socket 中读取数据，若该 socket 已经被客户端关闭，则相应线程关闭服务器端 socket 并退出线程。否则，使用 curl 库对从 socket 中读取到的请求报文进行编码转换，接下来，对请求报文进行解析，若该报文不是 HTTP 报文，则线程输出相应的错误信息、关闭 socket 并退出线程，否则，进一步解析报文，判断其是否时 HTTP GET 报文，若请求方法不是 GET，则输出相应的错误信息，并进行下一次报文读取，若接收到的请求报文是 HTTP GET 请求，请求文件路径进行处理，并结合启动服务时设置的虚拟路径，得到请求文件的实际路径。此时，将 HTTP 请求内容输出到日志框中。然后从文件名中分离出扩展名，并根据文件扩展名判断请求文件的类型，进而生成合适的响应报文首部，然后，尝试打开该文件，若文件无法打开，向客户端返回 404 信息，否则，使用一段缓冲区循环读取文件中的内容并将其写入到 socket 中，直到将文件全部写入到 socket 中后，关闭文件并输出相应的日志信息到图形界面，完成上述操作后，线程再次从 socket 中读取数据，若 socket 被关闭，则线程关闭服务器端的 socket 并退出，否则，线程阻塞并等待下一个请求报文或 socket 关闭信号。

1.5 系统测试及结果说明

系统测试的硬件环境如 1.1.2 节运行平台所示。首先使用 qmake 工具将 qt 中的 ui 文件转化为 C++文件，并生成 makefile，使用 make 完成项目的编译，得到可执行文件 socket。在命令行运行./socket 或者在图形界面双击可执行文件运行该文件。填写监听地址及端口、最大连接数以及虚拟路径，点击启动按钮，即可启动服务。测试过程中，将监听地址设置为本地地址 127.0.0.1 并将监听端口设置为 10000，将虚拟路径设置为“/home/huao/Desktop”，在该目录下创建 index.html 文件并在其中写入字符串“hello world”。使用浏览器访问 127.0.0.1:10000，对于未指定的文件请求，服务器将自动定向到该目录下的 index.html 文件，由此该操作服务器将把我们创建的 index.html 文件作为响应发送给客户端。浏览器解析响应报文后将把文件中的字符串显示出来。如图 1.3 为浏览器访问 127.0.0.1:10000 的截图。

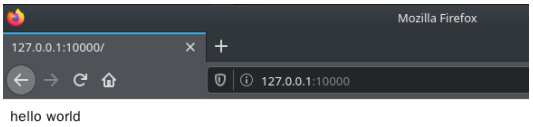

图 1.3 浏览器访问 127.0.0.1:10000 截图

在访问过程中，服务器日志框中将输出相应的日志信息，使用浏览器多次进行上述操作，得到如图 1.4 所示的服务器日志信息。

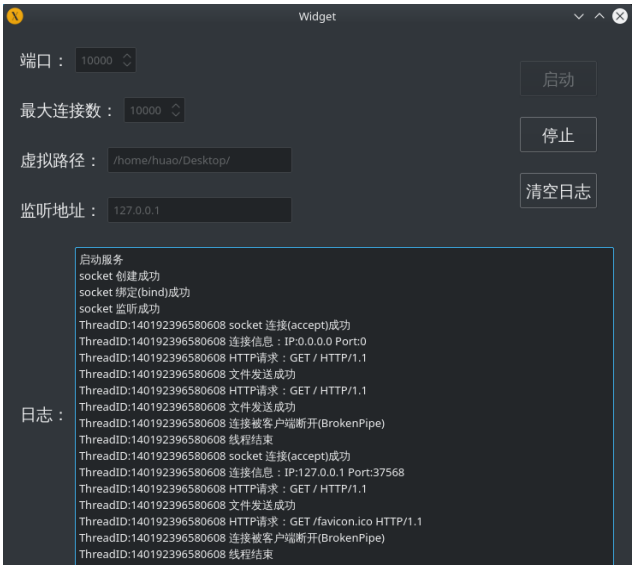

图 1.4 http 服务器运行截图

当待访问的文件不存在时，服务器将向请求方发送响应的 404 报文，使用服务器设置与上述测试中相同，使用浏览器访问 127.0.0.1:10000/aaa.html，由于在该文件夹不存在 aaa.html 文件，因此，服务器将响应 404 报文，浏览器解析该报文后得到如图 1.5 所示的运行结果。

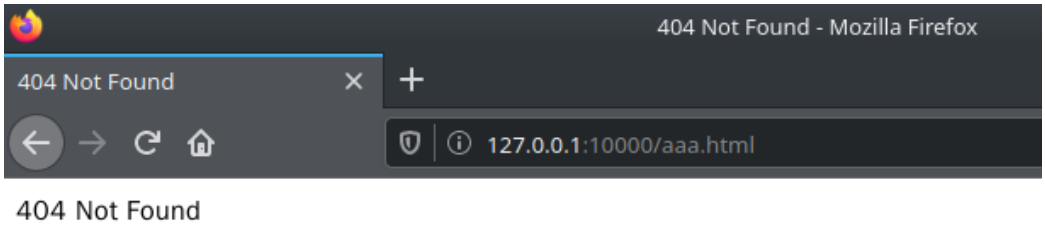

图 1.5 访问不存在的文件

在 1.html 中添加多个其他链接来测试服务器的多线程实现。使用浏览器访问链接 127.0.0.1:10000/1.html，服务器将该文件封装后发送给浏览器，浏览器解析文件内容后，根据其中的链接信息将同时向服务器请求创建多个 TCP 连接并行获取资源。使用 firefox 的 Web 工具中的网络视图，查看其请求文件的过程以及服务器的响应情况。如图 1.6 所示为上述访问过程。由时间图可知，firefox 获取到 1.html 并解析其内容后，创建了 6 个并行的 TCP 连接同时请求数据，对于前六个 1.mp4 文件，服务器同时发送相应的数据，此时后两个文件的请求被阻塞，等到 6 个 TCP 连接中的某一个完成上一个文件的传输后，浏览器将通过该 TCP 连接发送相应的请求，并得到响应。由此可知，服务器使用多线程成功实现了对多个请求的并行响应。

最后我们使用 wrk 对服务器进行压力测试。wrk 是一个开源的简易 HTTP 压力测试工具，它使用了 epoll 等方式，在使用较少的线程的情况下产生很大的并发量，用于模拟 HTTP 服务器在同时产生大量请求时的情况。从 Github 下载 wrk 的源码并编译得到可执行文件。

在测试过程中，我们使用 12 个线程，保持 10000 个 TCP 连接，请求 http://127.0.0.1:10000，另外将测试时间设置为 2 分钟并将一次请求的超时时间设置为 30 秒。压力测试的命令行为./wrk -t12 -c10000 -d2m -T30s http://127.0.0.1:10000/。启动服务器后，在终端执行上述测试命令，wrk

将向服务器建立大量的连接并发送请求。如图 1.7 所示，为访问 index.html 时压力测试的结果。

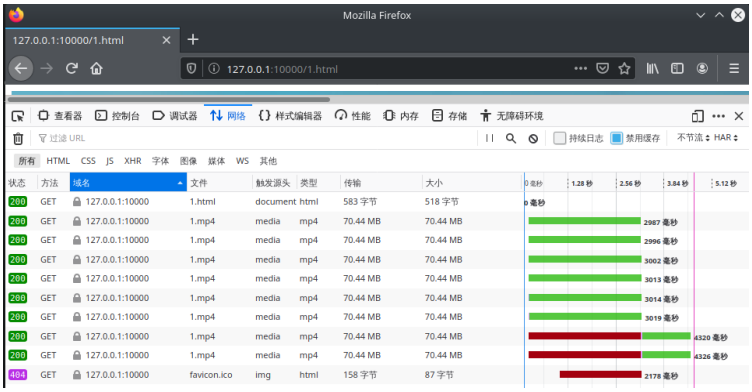

图 1.6 服务器多线程测试

```
~/aur/wrk (master ✔) ᐅ ./wrk -t12 -c10000 -d2m -T30s http://127.0.0.1:10000/
Running 2m test @ http://127.0.0.1:10000/
  12 threads and 10000 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   399.28ms  745.38ms  29.37s    88.47%
    Req/Sec   307.94    270.31     2.21k    77.45%
  282865 requests in 2.00m, 28.87MB read
  Socket errors: connect 0, read 0, write 0, timeout 74
Requests/sec:   2355.32
Transfer/sec:    246.14KB
```

图 1.7 压力测试(访问小文件 index.html)

由 wrk 输出的测试结果可以看到，在 2 分钟的测试中，一共完成了 282865 次请求，其中有 74 次请求超过了设置的超时时间，其他请求都得到了正确的完成。在该过程中，一共传输了 28.87MB 的数据。另外，类似的，我们将压力测试访问的文件修改为 1.mp4，该文件为一个大小为 70MB 左右的视频文件，来模拟客户端同时向服务器建立大量 TCP 连接并请求大文件时的状况。测试命令行为./wrk -t12 -c10000 -d2m -T30s http://127.0.0.1:10000/1.mp4。测试结果如图 1.8 所示。

```
~/aur/wrk (master ✔) ᐅ ./wrk -t12 -c10000 -d2m -T30s http://127.0.0.1:10000/1.mp4
Running 2m test @ http://127.0.0.1:10000/1.mp4
  12 threads and 10000 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    18.13s     4.50s   30.00s    68.99%
    Req/Sec     5.93      6.07    60.00     91.21%
  3381 requests in 2.00m, 259.07GB read
  Socket errors: connect 0, read 0, write 0, timeout 501
Requests/sec:     28.15
Transfer/sec:      2.16GB
```

图 1.8 压力测试(访问大文件 1.mp4)

由测试结果可知，在 2 分钟的测试过程中，一共完成了 3381 次请求，其中由 501 个请求出现超时，在这些请求中共传输了 259.07GB 数据。将此测试结果与访问小文件 index.html 的测试结果比较可知，在访问大文件时可以获得较大的数据吞吐量，但是能够支持的访问数量较少，而访问小文件可以支持大量的并发访问，但是不能够达到很大的数据吞吐量。该结果与实际情况下的预期是相符的。根据上面分别对小文件和大文件的压力测试结果可以看到，在实验中实现的 HTTP 服务器可以能够同时支持较大数量的 TCP 连接和 HTTP 请求。

通过上面对 HTTP 服务器基本功能的测试、异常情况的处理测试、多线程并发测试以及压力测试可以得知，我在实验中成功实现了能够满足基本功能的 HTTP 服务器，该服务器可以处理一定的异常请求情况，能够使用多线程来实现并行处理多个 HTTP 请求，同时，在同时建立大量的 TCP 连接并发送大量请求时，能够进行正常的工作。

1.6 其它需要说明的问题

最初，在使用 wrk 进行压力测试时，如果将 TCP 连接数设置得很大，那么在运行压力测试的过程中会出现大量的 socket 连接错误(socket connect error)，开始我猜测是由于操作系统对多线程的一些限制导致了这一结果。为了与该结果进行对比，我在云服务器上搭建了 nginx 作为 HTTP 服务器，并使用相同的方式进行压力测试，测试结果同样出现了大量的 socket 连接错误。因此，我认为应该是 wrk 在执行过程中收到了一些限制所以导致很多的 TCP 连接无法成功创建。在连接了 linux 对进程的一些限制以后，我发现 linux 在默认情况下限制一个进程只能够创建 1024 个文件描述符，由于 linux 将 socket 作为文件处理，因此 socket 连接的数量将会受到最大文件描述符数的限制，从而导致了上述问题。使用 ulimit 命令修改当前 shell 下进程文件描述符数量的上限，然后运行 wrk 程序对服务器进行压力测试，即可得到 1.5 节中压力测试的结果。

1.7 参考文献

[1] (美)詹姆斯 F 库罗斯(James F Kurose)，(美)基思 W 罗斯(Keith W Ross)著.计算机网络 自顶向下方法[M].北京：机械工业出版社,2018.06.

实验二 数据可靠传输协议设计实验

2.1 环境

CPU: Intel(R) Core(TM) i5-5350U CPU @ 1.80GHz
内存: 8GB DDR3
操作系统: Arch Linux
操作系统内核: x86_64 Linux 5.4.3-arch1-1
编译器: g++ (GCC) 9.2.0
自动构建工具: cmake version 3.16.1
GNU Make 4.2.1
文件比较工具: cmp(GNU diffutils) 3.7

2.2 实验要求

本实验包括三个级别的内容，具体包括：

- 实现基于 GBN 的可靠传输协议。
- 实现基于 SR 的可靠传输协议。
- 在实现 GBN 协议的基础上，根据 TCP 的可靠数据传输机制(包括超时后只重传最早发送且没被确认的报文、快速重传)实现一个简化版的 TCP 协议。报文段格式、报文段序号编码方式和 GBN 协议一样保持不变，不考虑流量控制、拥塞控制，不需要估算 RTT 动态调整定时器 Timeout 参数。

2.3 协议的设计、验证及结果分析

2.3.1 GBN 协议的设计、验证及结果分析

在 GBN 协议中，对于发送方，需要对如下 3 个事件做出相应的响应。其对应的有限状态机如图 2.1 所示。

1. 上层调用发送数据接口请求发送数据。首先判断窗口是否已满，若窗口已满，则返回 false 告知上层调用程序当前不能够发送数据，否则，根据上层传来的数据创建相应的报文并将其发出，同时，若该报文为窗口中的第一个报文，则启动计时器。

2. 计时器超时。若计时器超时，发送方将重启计时器并重新发送窗口中的所有包。

3. 收到接收方发来的 ACK 报文。首先，发送方计算 ACK 报文的检查和并将其与报文中的检查和进行比较，若检查和不同，则说明该包在传输过程中出现错误，此时，发送方忽略该包。若检查和相同，说明该包正确传输，此时，发送方更新窗口其实位置以及窗口中包的数量，确认该 ACK 包以及窗口中其对于序号前面的包被接收方正确确认。若此时发送方窗口为空，则关闭计时器，若发送方窗口中仍有未确认的包，则重启计时器。

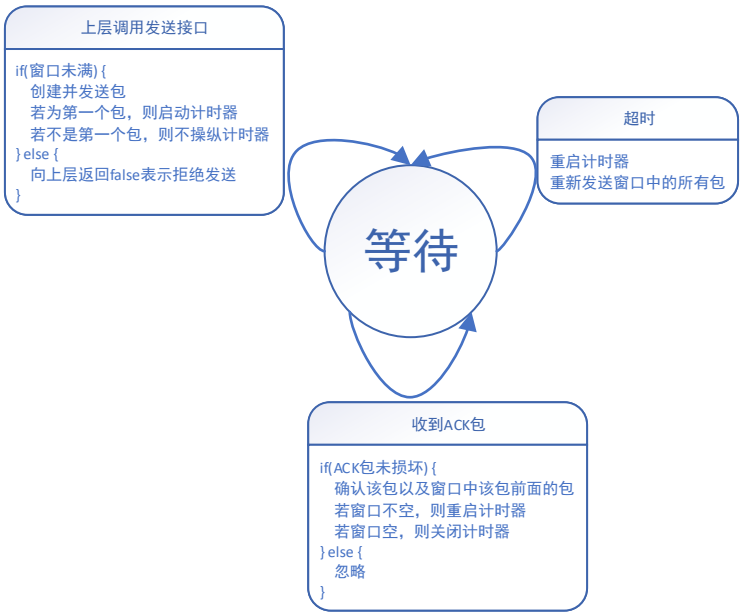

图 2.1 GBN 协议发送方有限状态机

接收方需要处理一个事件，即接受到一个包。当接收方接收到一个包时，首先计算报文检查和并判断报文是否正确传输，然后确定该报文是否为待接收的报文，若上述两个条件均满足，则取出报文中的数据部分并将其提交给上层应用，同时构建并发送相应的 ACK 报文。若报文检查和错误或者不是期待的报文，则接收方将该报文丢弃，同时重新发送上次的 ACK 报文。接收方的有限状态机如图 2.2 所示。

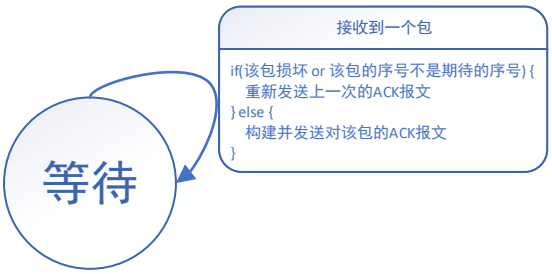

图 2.2 GBN 协议接收方有限状态机

根据上面对 GBN 发送方和接收方需要处理的事件的分析，即可完成 GBN 协议的实现。完成 GBN 协议的实现代码后，使用 cmake 生成 makefile，接下来使用 make 完成编译和链接的过程，生成可执行文件 GBN，然后运行测试脚本 check_linux.sh 对 GBN 实现的正确性进行测试。check_linux.sh 脚本将运行 10 次 GBN 程序，每次运行后使用 cmp 工具对输入文件 input.txt 和输出文件 output.txt 进行比较，若两文件存在不同，cmp 将输出两文件第一处不同的位置。

测试脚本运行结果如图 2.3 所示。由运行结果可知，GBN 程序实现了发送方与接收方之间的可靠数据传输。

图 2.3 GBN 协议正确性测试

在 GBN 程序运行时的输出信息中可以看到如图 2.4 所示的超时处理。由该输出可知，GBN 发送方在遇到超时事件时会将窗口中的包全部重发。

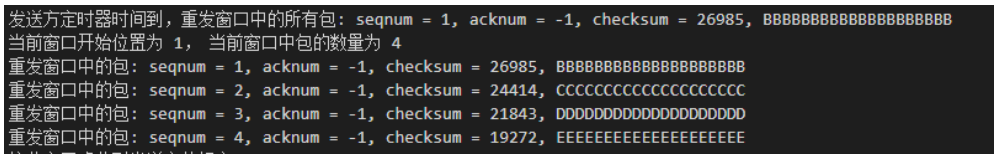

图 2.4 GBN 协议超时处理

2.3.2 SR 协议的设计、验证及结果分析

在 SR 协议中，发送方需要对如下 3 个事件做出相应的处理。其对应的有限状态机如图 2.5 所示。

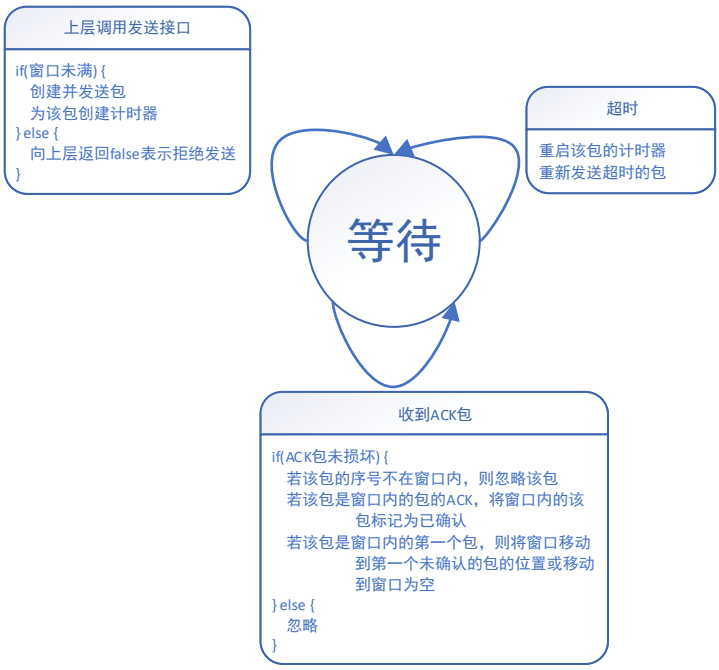

图 2.5 SR 协议发送方有限状态机

1. 上层调用发送数据接口发送数据。若窗口未满，则根据上层的数据创建包并将其发送给网络层，同时，为该包启动定时器。若此时窗口已满，则向上层返回 false 表明此时不能够

发送数据。

2. 计时器超时。当某个包的计时器超时时，SR 协议将重发该超时的包，并重启该包的计时器。

3. 收到来自接收方的 ACK 包。首先，发送方计算该包的检查和以判断该包是否损坏，若 ACK 包已损坏，则直接忽略该包。若该包正确传输，则判断该包的对应的发送包是否还在窗口中，若不在窗口中，则忽略该包，否则，将窗口中的该包对应的发送包标记为已确认。然后判断其对应的发送包是否为窗口中的第一个包，若不是，则不对窗口做出调整，若是第一个包，则将窗口向前移动，直到遇到一个仍未被确认的包或者窗口为空时，停止移动窗口。

SR 协议的接收方需要处理一个事件，即接收到一个来自发送方的包。当接收方接收到一个来自发送方的包时，实现计算检查和并判断该包是否正确传输，若包在传输过程中损坏，则直接忽略，若包正确传输，首先，接收方构建并发送对该包的 ACK，然后在接收方的窗口中缓存该包。接下来，判断该包是否为窗口中的第一个包，若为第一个包，则此时可以向上层提交数据，接收方将把窗口头向前移动并向应用层提交数据，直到窗口头指向的位置不是已经缓存的数据。接收方的有限状态机如图 2.6 所示。

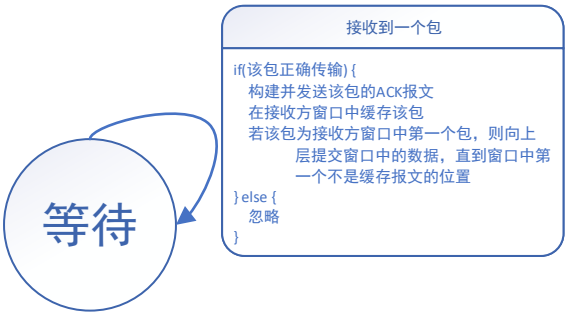

图 2.6 SR 协议接收方有限状态机

根据上面对 SR 协议发送方和接收方的分析，即可完成 SR 协议的实现代码。与 GBN 协议的测试类似，首先使用 cmake 和 make 工具完成代码的编译链接，得到可执行文件 SR，对测试脚本 check_linux.sh 进行相应的修改用于测试 SR 协议的正确性。运行测试脚本的结果如图 2.7 所示。由测试结果可知，程序中实现的 SR 协议正确实现了发送方和接收方之间的可靠数据传输。

图 2.7 SR 协议正确性测试

2.3.3 简单 TCP/IP 协议的设计、验证及结果分析

本次实验中，简单 TCP 协议是基于 2.3.1 节中设计的 GBN 协议实现的。在 GBN 协议的基础上，简单 TCP 协议仅做出两点修改。首先，当发送方检测到超时事件发生时，不像 GBN 中将窗口中的所有报文全部重发，而是仅仅重发窗口中的第一个报文；另外，当发送方接收到来自接收方的 3 次重复 ACK 报文时，将其视为一次超时事件，重发窗口中的第一个报文并重启计时器。

在 GBN 协议实现代码的基础上进行上述修改，即可实现简单 TCP 协议。使用 cmake 和 make 完成程序的编译和链接得到可执行文件 TCP，对测试脚本 check_linux.sh 进行相应的修改用于测试 TCP 协议实现的正确性，测试脚本运行结果如图 2.8 所示。由测试结果可知，TCP 正确实现了发送方与接收方之间的可靠数据传输。

图 2.8 TCP 协议正确性测试

在 TCP 程序运行过程中，通过其输出信息可以看到如图 2.9 所示的超时处理。当发送方定时器超时时，相比于图 2.4 所示的 GBN 超时重传窗口中的所有包，TCP 仅重传了窗口中的第一个包。如图 2.10 为 TCP 程序的快速重传操作。由输出结果可知，该时刻发送方接收到了来自接收方的 3 次重复 ACK，从而导致了发送方重传窗口中的第一个包。由此可知，TCP 在 GBN 的基础上正确实现了超时时仅重传窗口中的第一个包和快速重传两个特性。

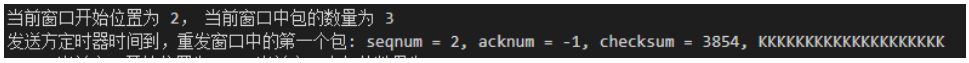

图 2.9 TCP 协议超时处理

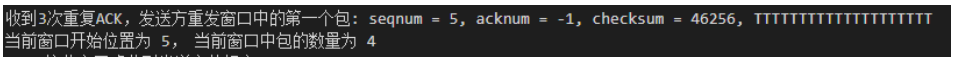

图 2.10 TCP 协议快速重传操作

2.4 其它需要说明的问题

本次实验我是在 linux 平台下完成的，在 linux 下换行默认使用“\n”，相比之下，windows 平台下，换行默认使用“\r\n”，这一区别导致了在运行过程中发送方的应用层对输入文件在不同平台下进行了不同的切分，进而导致在 linux 下运行程序时，每个包的内容并不是我们预期的一串相同的字母，虽然该区别对实验结果不会产生影响，但是在调试过程中会导致输出结果不便于阅读。在 linux 下，通过将换行由“\r\n”替换为“\n”，即可使得运行过程中每个包的数据部分为一串相同的数字，从而保证了输出信息的格式。

2.5 参考文献

[1] (美)詹姆斯 F 库罗斯(James F Kurose)，(美)基思 W 罗斯(Keith W Ross)著.计算机网络 自顶向下方法[M].北京：机械工业出版社,2018.06.

实验三 基于 CPT 的组网实验

3.1 环境

CPU: Intel(R) Core(TM) i5-5350U CPU @ 1.80GHz
内存: 8GB DDR3
操作系统: windows 10 专业版 版本 1903(OS 内部版本 18362.535)
Cisco Packet Tracer: 6.0.0.0045

3.2 实验要求

本次实验由基本部分和综合部分两部分组成。其中，基本部分包含了以下两项实验，每项实验包含了多个基本内容。

第一项实验——IP 地址规划与 VLan 分配实验：

使用仿真软件描述网络拓扑图 3.1。

基本内容 1

- 将 PC1、PC2 设置在同一个网段，子网地址是：192.168.0.0/24；
- 将 PC3~PC8 设置在同一个网段，子网地址是：192.168.1.0/24；
- 配置路由器，使得两个子网的各 PC 机之间可以自由通信。

基本内容 2

- 将 PC1、PC2 设置在同一个网段，子网地址是：192.168.0.0/24；
- 将 PC3、PC5、PC7 设置在同一个网段，子网地址是：192.168.1.0/24；
- 将 PC4、PC6、PC8 设置在同一个网段，子网地址是：192.168.2.0/24；
- 配置交换机 1、2、3、4，使得 PC1、PC2 属于 Vlan2，PC3、PC5、PC7 属于 Vlan3，PC4、PC6、PC8 属于 Vlan4；
- 测试各 PC 之间的连通性，并结合所学理论知识进行分析；
- 配置路由器，使得拓扑图上的各 PC 机之间可以自由通信，结合所学理论对你的路由器配置过程进行详细说明。

第二项实验——路由配置实验

使用仿真软件描述网络拓扑图 3.2

基本内容 1

- 将 PC1 设置在 192.168.1.0/24 网段；

- 将 PC2 设置在 192.168.2.0/24 网段；
- 将 PC3 设置在 192.168.3.0/24 网段；
- 将 PC4 设置在 192.168.4.0/24 网段
- 设置路由器端口的 IP 地址
- 在路由器上配置 RIP 协议，使各 PC 机能互相访问

基本内容 2

- 将 PC1 设置在 192.168.1.0/24 网段；
- 将 PC2 设置在 192.168.2.0/24 网段；
- 将 PC3 设置在 192.168.3.0/24 网段；
- 将 PC4 设置在 192.168.4.0/24 网段
- 设置路由器端口的 IP 地址
- 在路由器上配置 OSPF 协议，使各 PC 机能互相访问

基本内容 3

- 在基本内容 1 或者 2 的基础上，对路由器 1 进行访问控制配置，使得 PC1 无法访问其它 PC，也不能被其它 PC 机访问。
- 在基本内容 1 或者 2 的基础上，对路由器 1 进行访问控制配置，使得 PC1 不能访问 PC2，但能访问其它 PC 机

综合部分的内容及要求如下。

实验背景：

某学校申请了一个前缀为 211.69.4.0/22 的地址块，准备将整个学校连入网络。该学校有 4 个学院，1 个图书馆，3 个学生宿舍。每个学院有 20 台主机，图书馆有 100 台主机，每个学生宿舍拥有 200 台主机。

组网需求：

- 图书馆能够无线上网
- 学院之间可以相互访问
- 学生宿舍之间可以相互访问
- 学院和学生宿舍之间不能相互访问
- 学院和学生宿舍皆可访问图书馆。

实验任务要求：

- 完成网络拓扑结构的设计并在仿真软件上进行绘制(要求具有足够但最少的设备，不需要考虑设备冗余备份的问题)
- 根据理论课的内容，对全网的 IP 地址进行合理的分配
- 在绘制的网络拓扑结构图上对各类设备进行配置，并测试是否满足组网需求，如有无法满足之处，请结合理论给出解释和说明。

3.3 基本部分实验步骤说明及结果分析

3.3.1 IP 地址规划与 VLan 分配实验的步骤及结果分析

实验中完成的拓扑图如图 3.1 所示。

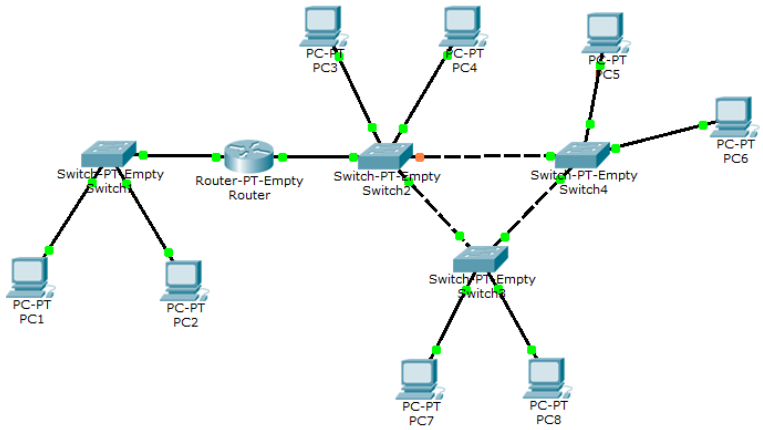

图 3.1 IP 地址规划与 VLan 分配实验拓扑图

1. 基本内容 1。IP 分配如表 3.1 所示。按表中内容将 IP 地址分配给 8 台 PC 和路由器的两个端口，由 IP 地址的分配可知，PC1 和 PC2 的网关地址为 192.168.0.254，PC3-PC8 的网关地址为 192.168.1.254。根据上述说明，为各 PC 和路由器配置 IP 地址，为 PC 配置网关地址，即可实现各 PC 之间的通信。

表 3.1 第一项实验基本内容 1 IP 地址划分

设备	IP 地址
PC1-PC2	192.168.0.1-192.168.0.2
PC3-PC8	192.168.1.1-192.168.1.6
路由器连接 PC1-PC2 的端口	192.168.0.254
路由器连接 PC3-PC8 的端口	192.168.1.254

2. 基本内容 2。首先，在每个交换机的 Vlan 数据库中添加 3 个 VLan，分别对应到实验内容中的三个虚拟虚拟局域网，令 PC1 和 PC2 处于 VLan2 中，PC3/PC5/PC7 处于 VLan3 中，PC4/PC6/PC8 处于 VLan4 中。IP 地址划分及各 PC 网关地址见表 3.2。为路由器连接 PC3-PC8 的端口创建两个子端口，并将两个子端口分别绑定到 VLan3 和 VLan4 上，为路由器连接 PC1-PC2 的端口创建一个子端口，将该子端口绑定到 VLan2 中。根据表 3.2 中的 IP 地址和网关地址，为各 PC 及路由器的各端口设置 IP 地址。设置完成后，即可实现各 PC 之间的相互通信。

表 3.2 第一项实验基本内容 2 IP 地址划分

设备	IP 地址	网关地址
PC1-PC2	192.168.0.1-192.168.0.2	192.168.0.254
PC3、PC5、PC7	192.168.1.1-192.168.1.3	192.168.1.254

PC4、PC6、PC8		192.168.2.1-192.168.2.3	192.168.2.254
路由器连接 PC1-PC2 的端口		192.168.0.254	
路由器连接 PC3-PC8 端口	与 PC3、PC5、PC7 同 VLan 的子端口	192.168.1.254	
	与 PC4、PC6、PC8 同 VLan 的子端口	192.168.2.254	

3.3.2 路由配置实验的步骤及结果分析

实验中的拓扑图如图 3.2 所示。本项实验中的三个内容对 IP 地址的分配相同，各 PC 和路由器的 IP 地址以及各 PC 的网关地址见表 3.3。根据表中对 IP 地址的划分方式，为拓扑图中各 PC 和路由器设置 IP 地址及网关地址。此时各 PC 之间不能够相互通信，需要在路由器上配置一定的路由算法才能够实现主机间的通信。

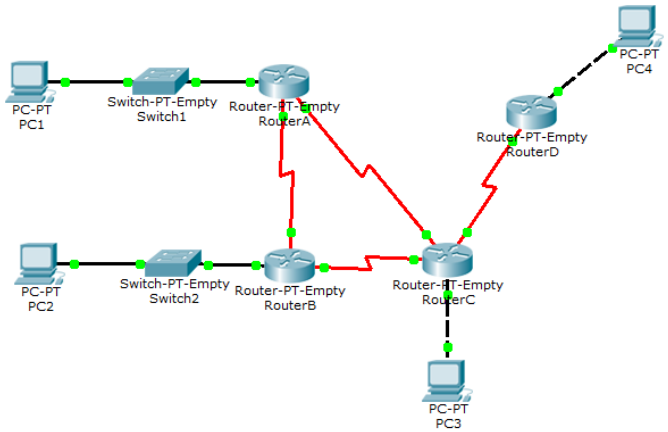

图 3.2 路由配置实验拓扑图

表 3.3 路由配置实验 IP 地址及网关地址分配

设备		IP 地址	网关地址
PC1		192.168.1.1	192.168.1.254
PC2		192.168.2.1	192.168.2.254
PC3		192.168.3.1	192.168.3.254
PC4		192.168.4.1	192.168.4.254
路由器 A	连接 PC1 的端口	192.168.1.254	
	连接路由器 B 的端口	192.168.5.1	
	连接路由器 C 的端口	192.168.6.1	
路由器 B	连接 PC2 的端口	192.168.2.254	
	连接路由器 A 的端口	192.168.5.2	
	连接路由器 C 的端口	192.168.7.1	
路由器 C	连接 PC3 的端口	192.168.3.254	
	连接路由器 A 的端口	192.168.6.2	
	连接路由器 B 的端口	192.168.7.2	
	连接路由器 D 的端口	192.168.8.1	
路由器 D	连接 PC4 的端口	192.168.4.254	
	连接路由器 C 的端口	192.168.8.2	

在路由配置实验中，将分别在路由器上配置 RIP 协议和 OSPF 协议，来实现拓扑图 3.2 中各主机之间的相互通信。另外，还需要在配置好 RIP 或 OSPF 的基础上，对路由器 A 进行访问控制的配置，从而实现对部分主机之间不能相互访问的控制。

1. 基本内容 1。为各路由器配置 RIP 协议实现主机间的相互通信。在 RIP 协议中，路由器需要直到其能够直接到达的网段的网络号，可以直接使用 CPT 的图形界面将与路由器直连的网段的网络号添加到 RIP 路由协议的网络地址中。根据表 3.3 中对 IP 地址的分配可知，四个路由器在 RIP 协议中需要添加的网络地址如表 3.4 所示。按表中的信息将网络地址添加到路由器的 RIP 协议中，即可完成 RIP 协议的配置。

表 3.4 路由配置实验 各路由器直连网段

路由器	路由器直连网段
路由器 A	192.168.1.0, 192.168.5.0, 192.168.6.0
路由器 B	192.168.2.0, 192.168.5.0, 192.168.7.0
路由器 C	192.168.3.0, 192.168.6.0, 192.168.7.0, 192.168.8.0
路由器 D	192.168.4.0, 192.168.8.0

配置完 RIP 协议后，打开路由器的命令行界面，首先输入 enable 进入特权模式，然后输入命令 show ip route 查看路由表，如图 3.3 为路由器 A 的路由表信息，路由表中，以 C 开头的行是与路由器直连的网段所产生的路由信息，以 R 开头的行是路由器通过 RIP 学习得到的路由信息，类似的，我们可以查看其他三个路由器的路由表。根据路由表的表项可知，运行 RIP 协议后的路由器可以正确的实现对发往路由器非直连网段的报文进行正确的转发。通过在各主机间发包测试可以看到各 PC 之间可以正常通信。

```
C    192.168.1.0/24 is directly connected, FastEthernet0/0
R    192.168.2.0/24 [120/1] via 192.168.5.2, 00:00:26, Serial1/0
R    192.168.3.0/24 [120/1] via 192.168.6.2, 00:00:07, Serial2/0
R    192.168.4.0/24 [120/2] via 192.168.6.2, 00:00:07, Serial2/0
C    192.168.5.0/24 is directly connected, Serial1/0
C    192.168.6.0/24 is directly connected, Serial2/0
R    192.168.7.0/24 [120/1] via 192.168.5.2, 00:00:26, Serial1/0
                    [120/1] via 192.168.6.2, 00:00:07, Serial2/0
R    192.168.8.0/24 [120/1] via 192.168.6.2, 00:00:07, Serial2/0
```

图 3.3 配置 RIP 协议后的路由器 A 的路由表

2. 基本内容 2。内容 2 需要给各路由器配置 OSPF 协议来实现各主机间的相互通信。在 CPT 中，OSPF 协议需要使用命令行进行配置。OSPF 协议的配置中同样需要使用与路由器直连网段的网络地址，不过在 CPT 中，使用 OSPF 还需要指定直连网段的子网掩码。本实验中，各路由器直连网段的网络地址如表 3.4 所示。对于路由器 A，使用如下命令为其配置 OSPF 路由协议。

```
Router>enable
Router#conf terminal
Router(config)#router ospf 1
Router(config-router)#network 192.168.1.0 0.0.0.255 area 0
Router(config-router)#network 192.168.5.0 0.0.0.255 area 0
```

Router(config-router)#network 192.168.6.0 0.0.0.255 area 0

Router(config-router)#end

Router#write

类似的，为其他三个路由器配置 OSPF 协议，配置完成后等待一段时间，打开路由器 A 的命令行界面查看其路由表信息如图 3.4 所示。由各路由表的表项以及各 PC 间发送报文的响应情况可知，正确完成了在路由器上配置 OSPF 协议并实现了各 PC 之间的相互通信。

```
C    192.168.1.0/24 is directly connected, FastEthernet0/0
O    192.168.2.0/24 [110/65] via 192.168.5.2, 00:14:09, Serial1/0
O    192.168.3.0/24 [110/65] via 192.168.6.2, 00:14:09, Serial2/0
O    192.168.4.0/24 [110/129] via 192.168.6.2, 00:14:09, Serial2/0
C    192.168.5.0/24 is directly connected, Serial1/0
C    192.168.6.0/24 is directly connected, Serial2/0
O    192.168.7.0/24 [110/128] via 192.168.5.2, 00:14:09, Serial1/0
                    [110/128] via 192.168.6.2, 00:14:09, Serial2/0
O    192.168.8.0/24 [110/128] via 192.168.6.2, 00:14:09, Serial2/0
```

图 3.4 配置 OSPF 协议后的路由器 A 的路由表

3. 基本内容 3。在配置好 RIP 协议的网络拓扑图上进行路由器的访问控制的配置。本实验内容包含两部分。首先，需要对路由器 A 进行配置，使得 PC1 不能够访问其他 PC，也不能被其他 PC 访问。为实现该控制，我们可以配置路由器 A 与 PC1 相连的端口阻止来自 PC1 的报文，这样，当 PC1 试图向其他主机发送报文时，其发出的报文将被路由器 A 丢弃，因此 PC1 无法访问其他 PC，当其他 PC 向 PC1 发送报文时，PC1 将收到该报文，但是 A 发送的响应报文将被路由器 A 丢弃，因此在其他 PC 看来，他们无法访问 PC1。故该阻止方法可以实现访问控制要求，对路由器 A 的访问控制设置的命令行如下。

Router>enable

Router#conf terminal

Router(config)#access-list 1 deny 192.168.1.1 0.0.0.0

Router(config)#access-list 1 permit any

Router(config)#interface fa0/0

Router(config-if)#ip access-group 1 in

Router(config-if)#end

Router#write

本内容的另一个访问控制要求为 PC1 不能访问 PC2，但是可以访问其他 PC。可使用如下方法实现该访问控制要求，让路由器 A 与 PC1 相连的端口阻止发往 192.168.2.1(PC2)的报文。这样，当 PC1 尝试访问 PC2 时，PC1 发出的报文将被路由器 A 丢弃，因此 PC1 无法访问 PC2，对于 PC1 向其他 PC 发送的报文，他们可以正常的被路由器 A 转发，因此 PC1 可以正常访问其他的 PC。给路由器 A 配置访问控制的命令行如下。

Router>enable

Router#conf terminal

Router(config)#access-list 1 deny 192.168.2.1 0.0.0.0

Router(config)#access-list 1 permit any

Router(config)#interface fa0/0

Router(config-if)#ip access-group 1 out

Router(config-if)#end

Router#write

对路由器 A 完成上述访问控制的配置后，多次从 PC1 向 PC2 发送报文，可以看到始终无法正常发送，使用模拟方式发送报文并查看报文从 PC1 发往 PC2 的过程可以看到，当报文发送到路由器 A 时，路由器 A 在检测访问控制表后将该报文丢弃。从 PC1 向其他 PC 发送报文，除最初由于使用 ARP 协议获取 mac 地址造成的第一次失败外，以后 PC1 可以与其他 PC 正常通信。

3.4 综合部分实验设计、实验步骤及结果分析

3.4.1 实验设计

首先，根据学校的层次结构以及各部分之间的关系，设计网络拓扑图。然后根据学校各部分拥有的主机的数量，将学校的 IP 地址块分配给学生宿舍、图书馆和学院等各部分。接下来，为拓扑图中的每个路由器配置路由选择算法，使得学校的每个主机之间可以相互访问，最后，根据组网要求，给部分路由器配置访问控制协议，使得部分主机间不能够相互访问。

3.4.2 实验步骤

首先，根据如图 3.5 所示的层次结构进行网络拓扑图的设计。为保证不同部分之间的区别，可以使用路由器将不同部分分隔开，形成真实的子网，也可以在全校范围内使用单个路由器，通过在交换机上配置 VLan 来实现各部分之间的分隔。

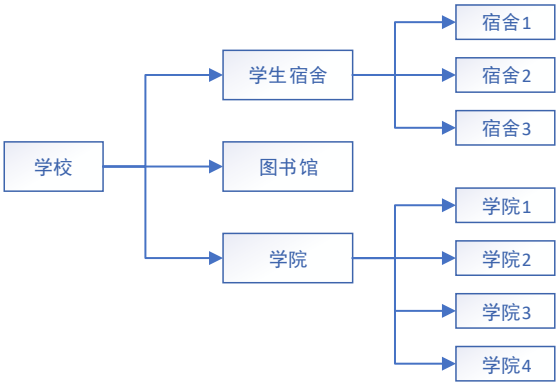

图 3.5 学校的层次结构

在本次实验中，我在不同的结构中使用一个路由器，从而实现不同机构的分割。综合考虑各部分拥有的主机数量以及单个交换机或路由器能够连接的主机数量，绘制如图 3.6 所示的拓扑图。图中，宿舍部分使用了 CPT 的集群功能将每个宿舍中的设备聚合在一起，每个宿舍内

部的结构如图 3.7 所示。由于实际包含的 PC 过多，拓扑图中使用了足够多的交换机和路由器，但是每个交换机上仅连接 1 或 2 台设备作为象征。

接下来，根据学校拥有的 IP 地址段以及学院、图书馆和学生宿舍对 IP 地址的需求，对 IP 地址进行分配，分配方式如表 3.5 所示。由于各部分实际上并未将分配到的 IP 全部用完，因此可以将其中未使用的部分进行细分，分配给路由器和路由器之间连线形成的子网。根据上述分析，给各 PC 及路由器的端口设置 IP 地址。

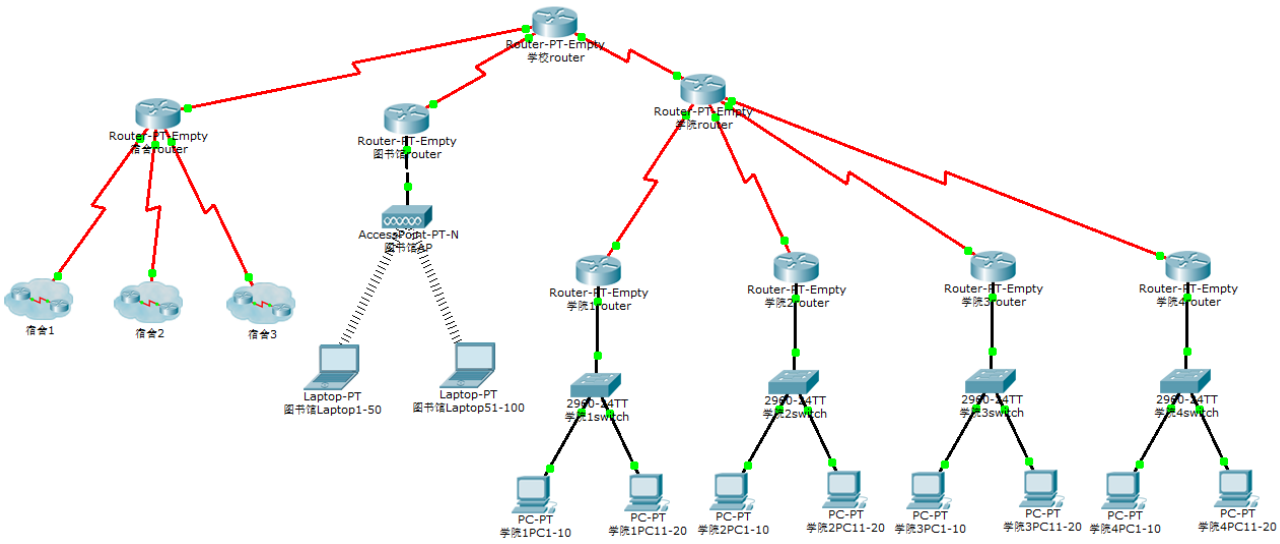

图 3.6 学校网络拓扑图

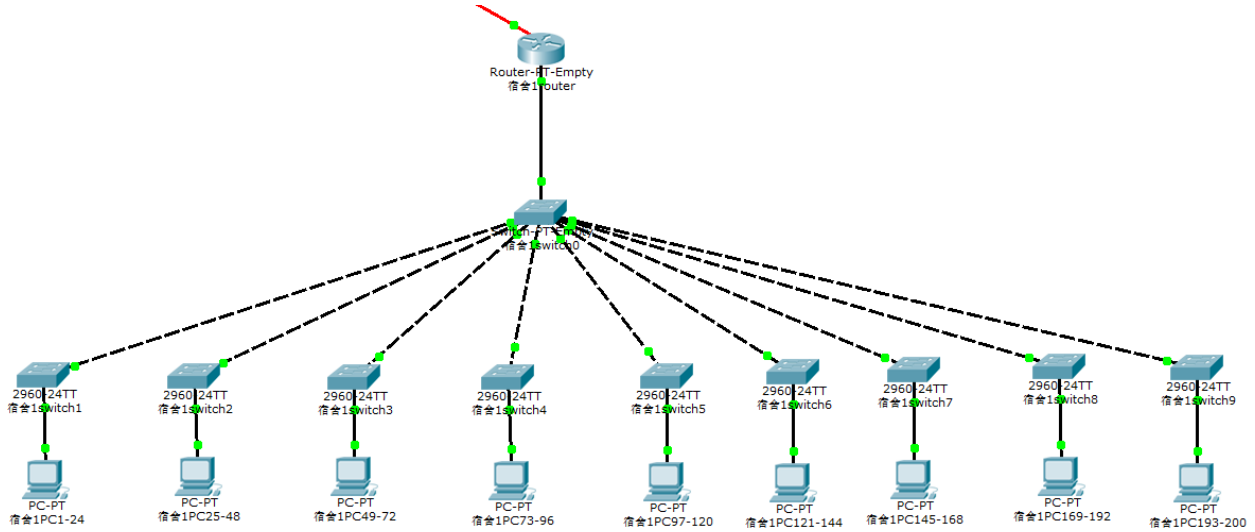

图 3.7 每个宿舍内部的网络拓扑结构

表 3.5 IP 地址分配

	IP 地址
学生宿舍 1	211.69.4.0/24
学生宿舍 2	211.69.5.0/24

学生宿舍 3	211.69.6.0/24
图书馆	211.69.7.0/25
学院 1	211.69.7.128/27
学院 2	211.69.7.160/27
学院 3	211.69.7.192/27
学院 4	211.69.7.224/27

根据组网需求，首先给各路由器配置路由协议，使得所有的主机之间都可以相互访问，然后对部分路由器进行访问控制的配置以阻止部分访问的进行，从而实现学院和学生宿舍之间不能相互访问的需求。对于路由协议，可以选择使用 OSPF 协议或 RIP 协议，在本次实验中，我选择使用 RIP 协议实现各主机之间的相互访问。对于每个路由器，将其直连网段的网络地址添加到 RIP 协议的网络地址中，即可完成 RIP 协议的配置。配置完 RIP 协议后，测试各主机间的连通性，确定 RIP 协议配置的正确性。

在组网要求中，不能够相互访问的只有学院和学生宿舍之间，因此只需要其进行限制。限制可以在学生宿舍路由器上进行，由表 3.5 中 IP 地址的分配可知，将四个学院视为一个整体，其分配到的 IP 地址可记为 211.69.7.128/25，在学生宿舍路由器与学校路由器连接的端口上，若检测到一个包的来源时 211.69.7.128/25 或者一个包将要被发往 211.69.7.128/25，则路由器将该报文丢弃。这样，当学生宿舍试图向学院发包时，该报文到达学生宿舍路由器后，路由器不会将其转发到学校路由器而是将其丢弃，因此学生宿舍不能够向学院发送报文，反过来，当学院向学生宿舍发包时，当报文被转发到学生宿舍路由器后，路由器检测到其来源是学院，将会把该报文丢弃而不是转发到学生宿舍内部。因此，访问控制的实现可以在学生宿舍路由器上实现，实现的命令行代码如下。

```
Router>enable
Router#conf terminal
Router(config)#access-list 1 deny 211.69.7.128 0.0.0.127
Router(config)#access-list 1 permit any
Router(config)#interface Serial0/0
Router(config-if)#ip access-group 1 in
Router(config-if)#ip access-group 1 out
Router(config-if)#end
Router#write
```

3.4.3 结果分析

完成上面的实验步骤以后，在学校不同部分之间相互发送简单的 ICMP 报文并观察各主机之间的可达性信息，可以看到宿舍与宿舍之间、学院和学院之间可以相互通信，宿舍和学院的主机都可以和图书馆的主机之间相互通信，但是宿舍和学院的主机之间不能够相互通信。由测试结果可知，实验中成功实现了组网需求中提出的要求。

3.5 其它需要说明的问题

本次组网实验中，要求图书馆可以使用无线上网，因此我设想使用 DHCP 来实现图书馆上网用户的 IP 地址动态分配，在 CPT 的无线设备中有一个名为 Linksys-WRT300N 的设备，该设备的配置中包含了 DHCP 服务器的配置，因此我认为该设备可以提供 DHCP 的功能，但是在实际的配置过程中，我遇到了很多问题，因此没能够成功使用该设备完成 DHCP 服务的配置。

3.6 参考文献

[1] (美)詹姆斯 F 库罗斯(James F Kurose)，(美)基思 W 罗斯(Keith W Ross)著.计算机网络 自顶向下方法[M].北京：机械工业出版社,2018.06.

心得体会与建议

4.1 心得体会

通过三次计算机网络实验，我对计算机网络实际的结构及其实现有了直观的认识。在第一次的 socket 编程实验中，我了解到了网络应用层程序的编写方法，对 linux 系统提供的 socket 编程接口有了一定的认识。另外，在这次实验中，我接触到了 qt 并使用它构建了简单的图形界面，同时，我对多线程编程有了一些初步的了解。第二次实验中，我们实现了 GBN 和 SR 协议以及简单的 TCP 协议，通过自己动手编写相应的代码，我对运输层提供的可靠数据传输有了更深的认识，对计算机网络理论课上讲述的可靠数据传输的原理有了直观的认识。第三次实验是基于 CPT 的组网，通过模拟组网，我学会了路由器和交换机的基本配置方法，在配置过程中，也加深了我对网络层和链路层的相关知识的了解，同时，在 CPT 中，我直观的看到了 OSPF、RIP 等路由协议的运行过程。

在这三次实验中，我直观的感受到了计算机网络的结构，对计算机网络理论课中讲到的知识进行了较多的补充。

4.2 建议

在计算机网络课程中，我们只学习到了与网络相关的理论知识，这导致在实验中我们需要自己动手实现一些功能时会感到不知从何入手，如果在实验课和理论课可以结合得更加紧密的话，可以让我们更好的学习这门课程。