计算机网络实践说明书

2023年8月

# 简介

本次实践中，同学们需要在应用层使用UDP实现TCP的基本功能。依据TCP的RFC文档，实现标准TCP的listen、connect、accept等接口，实现连接管理、可靠数据传输、流量控制和拥塞控制等基本功能。实现基本的TCP，下面称为TJU\_TCP。

本次实践采用Linux操作系统。用c语言模拟基本的Linux内核操作，并实现TJU\_TCP功能。设置两个虚拟机分别模拟服务端和客户端，进行双方的通信。按照计算机网络的TCP/IP协议标准逐层进行数据包封装、传输和解封装。同学们可以使用Wireshark等抓包工具观察到自己实现的协议是如何在真实的网络环境中工作的。

为了增进公平性，避免人工验收标准不一的问题，本次实践大量采用自动验收方式进行程序测试和打分。提供测试的源码，同学们可以先在本地进行多次测试，然后再提交线上验收。

# TCP任务要求

根据RFC文档，一个完整的TCP需要应对各种情况，非常复杂。为降低任务量，从RFC中选取了TJU\_TCP必须实现的功能，其他功能选做，不参与评分。**注意：本节是评分的依据。在进度报告和实践报告中，需要详细说明本节所要求的每一条功能的设计和实现。**

## TJU\_TCP的包头定义

TJU\_TCP的包头是标准TCP包头的简化，共20个字节，如图1所示。



图1 TJU\_TCP的包头

包头的各项字段及其注解如表1所示。

表1 TJU\_TCP的包头字段说明

|  |  |
| --- | --- |
| **字段名称** | **说明** |
| Source Port Number | 源端口 |
| Destination Port Number | 目的端口 |
| Sequence Number | Sequence Number |
| Acknowledgement Number | Acknowledgement Number |
| Header Length | 包头长度，在TJUTCP中全部为20 |
| Packet Length | 整个包长度，包括包头和包携带的数据 |
| Flags | 标志位 比如 SYN FIN ACK 等 |
| Advertised Window | 接收方发送给发送方的建议窗口大小 用于流量控制 |
| Extension | 额外的数据 为了将header的大小凑整到20bytes 没有实际意义 |

## TJU\_TCP连接管理

**2.2.1连接建立**

1. 需要实现基本的三次握手建立连接，即实现RFC793[1]中定义的“Basic 3-Way Handshake” （参见Sec 3.4）。
2. 能够在有丢包和延迟的情况下建立连接。需要处理三次握手中每一种数据包丢失的情况。

**注：**

1. 不需要考虑同时建立连接的情况。
2. 不需要考虑“Old Duplicate SYN”的情况。
3. 不需要考虑建立连接过程中双方的崩溃（Crash）。
4. 不需要实现“Half-Open Connection Discovery”。
5. 不需要实现标准TCP的RST机制。

**2.2.2 连接关闭**

1. 需要按RFC793要求实现基本的四次挥手关闭连接。
2. 能够在双方先后断开时正确关闭连接。
3. 能够在双方同时断开时正确关闭连接。
4. 能够在有丢包和延迟的情况下断开连接。
5. 关闭连接前能发送完缓冲区的数据。连接的关闭不能影响数据传输，即发起关闭连接的一方仍然要将发送缓冲区中剩余的数据传输到接收方后才能关闭连接。

**注：**

1. 不需要考虑关闭连接的过程中双方崩溃。
2. 不需要考虑一方发送FIN或收到自己发送的FIN的ACK后直接关闭程序，不等待和回应对方的FIN的情况。
3. 不需要考虑so\_linger机制。
4. 不需要实现标准TCP的RST机制(比如在接收缓冲区存在数据时关闭socket不需要发送RST，已经关闭的socket接收到数据不需要发送RST)。

## TJU\_TCP可靠数据传输

需要依据RFC793 Sec 3.7的描述实现数据的可靠传输。具体要求如下：

1. 能够正确管理发送端缓冲区，实现滑动窗口机制。
2. 能够正确管理接收端缓冲区，实现滑动窗口机制。
3. 能够实现计算checksum，并能检测checksum出错。
4. 能够实现累积ACK （cumulative ACK）。
5. 能够实现快速重传。
6. 能够检测出丢包。
7. 能够依据RFC793 Sec 3.7的描述估计RTT，计算RTO。根据RTO进行数据包的超时重传。
8. 能够在有延迟和丢包的链路上实现至少**100MB**数据的可靠传输。

**注：**

1. 不需要考虑TCP urgent机制。
2. 不需要考虑传输过程中双方的崩溃。
3. 不需要实现延时应答。
4. 不需要实现捎带应答。

## TJU\_TCP流量控制

1. 接收方能够计算当前可用缓冲区的大小，并设置Advertised Window。
2. 发送方能够根据接收方的“Advertised Window”字段调整发送窗口。
3. 发送方能正确处理接收方的“Advertised Window”大小为0的情况，进行正确的0窗口探测 （参见RFC793 Sec 3.7）。

**注：**

1. 不需要考虑“silly window syndrome”，不需要实现Nagle等算法。

## TJU\_TCP拥塞控制

1. 需要按TCP Reno 的RFC5681[2]要求实现慢启动、拥塞避免和快速恢复机制。
2. 不需要实现SACK和D-SACK机制。

# TCP接口规范与全局约定

为统一标准，方便测试，提供了项目的基础代码，并规定了TJU\_TCP需要实现的各个接口。同学们需要按照接口的定义和功能描述实现具体的代码。测试时将编写代码调用同学们封装的接口，模拟服务端和客户端的行为，进行自动测试和验收。

## 不能修改的函数接口

具体来说，同学们需要实现以下8个接口。

1. tju\_tcp\_t \* tju\_socket ()

创建TJU\_TCP

1. int tju\_bind (tju\_tcp\_t \*sock, tju\_sock\_addr bind\_addr)

绑定监听的地址

1. int tju\_listen (tju\_tcp\_t \*sock)

被动打开

1. tju\_tcp\_t \* tju\_accept (tju\_tcp\_t \*sock)

接受连接

1. int tju\_connect (tju\_tcp\_t \*sock, tju\_sock\_addr target\_addr)

连接到服务端

1. int tju\_send (tju\_tcp\_t \*sock, const void \*buffer, int len)

发送数据

1. int tju\_recv (tju\_tcp\_t \*sock, void \*buffer, int len)

接收数据

1. int tju\_close (tju\_tcp\_t \*sock)

关闭连接

**有关接口的功能定义，输入输出和实现提示等更加详细的描述，请参考documentation目录下tju\_\_tcp\_8h.html**

**要求：**

1. **不能修改以上接口的函数签名**（函数名、参数位置、参数数量）。
2. **不能另起炉灶编写自己的TCP功能接口**（比如不实现规定的tju\_send接口，而是自己定义另一个用于发送数据的函数）。
3. **可以自由添加辅助函数**，**自由在接口的实现中调用其他函数**，只要保证接口按需求运行即可。

## 限制修改的文件

同学们可以修改基础代码中除这8个接口之外的所有内容，但为了保证公平，杜绝作弊和取巧，方便自动测试，同学们对于以下内容的修改需要联系助教进行申请和确认。

1. kernel.h和kernel.c中除了cal\_hash外各个函数和变量的定义和实现。
2. tju\_packet.h和tju\_packet.c中除TJU\_TCP报文结构定义(tju\_packet\_t)外的结构定义和各个函数。
3. global.h中对于TCP socket 状态定义、最大包长的定义、各处都可能用到的简化FLAG定义和数据长度定义。

## 参数设置

为了统一难度和标准，本次实践对同学们实现TJU\_TCP的性能，以及TJU\_TCP运行的环境进行如下要求和说明。

1. TJU\_TCP的最大包长MSS（包括包头和数据）设置为**1400字节**，防止IP层分片。
2. 发送缓冲区和接收缓冲区的大小至少为**5000个MSS**。
3. 拥塞窗口的阈值ssthresh初始化为接收窗口的最大值。
4. 用于测试TCP功能和性能的文件长度应不小于**100MB**。

# 实践环境说明

本章节介绍实践环境，说明其搭建和测试方法。

如图2所示，本次实践采用Linux操作系统，设置两个虚拟机模拟服务端和客户端，进行双方的通信。

图示

描述已自动生成

图2 实践环境

1. 两台虚拟机均为Ubuntu 20.04服务器版本, 没有GUI。
2. 宿主机的tju\_tcp目录同时挂载在了两台虚拟机中的/vagrant/tju\_tcp目录下，这意味着在宿主机中对tju\_tcp目录的任何改动都会实时同步到两台虚拟机中。
3. 建议同学们在宿主机中使用IDE进行编码，在虚拟机中进行编译和测试
4. 两台虚拟机都开启了ssh密码验证, 并且登录用户名和密码都为vagrant，server虚拟机ssh的端口为2200，client的端口为2222。
5. 两台虚拟机在同一个子网中，是互通的。server的IP地址为10.0.0.1，client的IP地址为10.0.0.2。
6. 两台虚拟机之间网络通讯的默认环境设置为：通信延迟20ms, 最大带宽100Mbps，没有丢包。网络环境可以在实验过程中修改。
7. 两台虚拟机之间使用UDP进行数据收发，同学们需要在UDP之上封装形成TJU\_TCP。

有关实践环境的搭建和测试方法，请参照：



# 实践源码验收及评分细则

本次实践的很多内容采用自动验收方式。自动验收程序分别开启服务端和客户端，测试同学们的客户端和服务端。自动验收程序调用同学们封装的函数接口，模拟服务端和客户端的行为，进行测试和验收。

目前，编写了三个部分的自动验收程序：连接的建立、可靠数据传输和连接的断开。自动测试程序根据测试结果进行打分。

流量控制和拥塞控制部分目前是人工验收。提供了自动抓包和绘图脚本，同学们需要根据一段时间内的抓包结果绘制sequence number随时间变化的图像，展示流量控制的工作过程和拥塞控制的各个部分。同学们需要提交绘制的图片并进行说明。

使用Tcconfig[3]可以配置网络环境参数，比如丢包率、延迟、带宽等，具体使用方法请参见[4]。

下面详细说明自动验收的方法和评分细则。

## 建立连接的验收

建立连接的验收满分为100分，验收详情如下：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **验收环境** | | | | | | | | |
| 丢包率 | <2% | 延迟 | <50ms | 带宽 | 100Mbps | 限时 | 5s | |
| **评分细则** | | | | | | | | |
| 能够正确发送SYN | | | | | | | | 20分 |
| 能够正确响应SYN发送SYNACK | | | | | | | | 40分 |
| 能够正确响应SYNACK，发送ACK | | | | | | | | 40分 |
| **具体流程** | | | | | | | | |
| 1、测试学生的客户端   1. 开启自动测试程序的服务端和客户端。 2. 客户端调用待测试的函数tju\_connect。 3. 服务端检验能否收到SYN以及SYN的各项字段，若检测通过，则发送正确的SYNACK至客户端，等待客户端的ACK。 4. 服务端检测能否收到ACK以及ACK的各项字段。 5. 关闭服务端和客户端。   2、测试学生的服务端   1. 开启服务端和客户端。 2. 服务端调用tju\_bind、tju\_listen和tju\_accept。 3. 客户端向正确的ip和端口发送正确的SYN，等待SYNACK。 4. 客户端检验能否收到SYNACK以及SYNACK的各项字段。 5. 关闭服务端和客户端。 | | | | | | | | |

## 断开连接的验收

断开连接的验收满分为100分，断开连接将测试双方先后断开和双方同时断开两种情况，验收详情如下：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **验收环境** | | | | | | | | | |
| 丢包率 | 0% | 延迟 | <50ms | 带宽 | 100Mbps | 限时 | 27s | | |
| **评分细则-双方先后断开连接-测试客户端功能** | | | | | | | | | |
| 能够发送正确的FINACK（第一次挥手） | | | | | | | | | 15分 |
| 能响应正确的FINACK，发送正确的ACK（第四次挥手） | | | | | | | | | 15分 |
| **评分细则-双方同时断开连接-测试客户端功能** | | | | | | | | | |
| 能够在已发送FINACK还未收到ACK时正确响应对方的FINACK | | | | | | | | 10分 | |
| 能够响应FINACK并发送正确ACK | | | | | | | | 10分 | |
| **评分细则-双方先后断开连接-测试服务端功能** | | | | | | | | | |
| 能够响应FINACK并发出正确ACK（第二次挥手） | | | | | | | | 15分 | |
| 能发送正确FINACK（第三次挥手） | | | | | | | | 15分 | |
| **评分细则-双方同时断开连接-测试服务端功能** | | | | | | | | | |
| 能够在已发送FINACK还未收到ACK时正确响应对方的FINACK | | | | | | | | 10分 | |
| 能够响应FINACK并发送正确ACK | | | | | | | | 10分 | |
| **具体流程-双方先后断开连接-测试客户端功能** | | | | | | | | | |
| 1. 开启自动测试程序第服务端和客户端，双方首先建立连接 2. 模拟端调用待测试的tju\_close。 3. 服务端检查FINACK，响应发送ACK。 4. 服务端发送FINACK，等待最后的ACK。 5. 服务端检查最后的ACK。 6. 重新运行服务端和客户端，双方建立连接。 7. 这次服务端发送FINACK，等待客户端的ACK。 8. 服务端检查ACK。 | | | | | | | | | |
| **具体流程-双方同时断开连接-测试客户端功能** | | | | | | | | | |
| 1. 开启自动测试程序第服务端和客户端，双方首先建立连接。 2. 客户端调用待测试的tju\_close。 3. 服务端检查FINACK，首先发送自己的FINACK。 4. 服务端发送FINACK后再响应客户端，发送ACK。 5. 服务端等待并检查最后的ACK。 | | | | | | | | | |
| **具体流程-双方先后断开连接-测试服务端功能** | | | | | | | | | |
| 1. 开启自动测试程序的服务端和客户端，双方首先建立连接 2. 模拟端调用待测试的tju\_close。 3. 客户端发送FIN，等待ACK。 4. 客户端检查ACK，等待FIN或FINACK。 5. 客户端检查FIN/FINACK，发送ACK。 | | | | | | | | | |
| **具体流程-双方同时断开连接-测试服务端功能** | | | | | | | | | |
| 1. 开启自动测试程序第服务端和客户端，双方首先建立连接。 2. 服务端调用待测试的tju\_close。 3. 客户端发送FIN。 4. 客户端检查FIN或FINACK后再响应客户端，发送ACK。 5. 服务端等待并检查最后的ACK。 | | | | | | | | | |

为方便同学们测试，降低自动验收网站的负载，我们提供了自动测试的所有源码，以供同学们在本地进行测试。

**建议同学们在本地测试得到符合要求的结果后再提交线上验收。**

**注意，本地的测试打分结果不作为评分依据，最终以线上评分结果为准。**

有关本地自动测试的具体步骤，请参考：



有关如何提交线上验收网站，查看自己的验收成绩和排行榜，请参考：



## 可靠数据传输的验收

可靠数据传输的验收满分为100分，验收详情如下：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **验收环境** | | | | | | | | |
| 丢包率 | <10% | 延迟 | <300ms | 带宽 | 100Mbps | 限时 | 60s | |
| **评分细则** | | | | | | | | |
| 服务端连续正确接收的来自客户端的数据 | | | | | | | |  |
| **具体流程** | | | | | | | | |
| 1. 开启自动测试程序的服务端和客户端。 2. 服务端调用待测试的tju\_bind、tju\_listen和tju\_accept。 3. 客户端调用待测试的tju\_connect。 4. 建立连接后，客户端读取总大小为100MB的测试文件（test.txt），发送完毕调用tju\_close。 5. 服务端将收到的数据写入接收文件(recv.txt)。 | | | | | | | | |

**注意：**

1. **可靠数据传输要首先建立连接，因此建立连接的过程也存在丢包，需要做相应处理**。
2. **客户端需要调用tju\_close，在TCP连接关闭时如果发送缓冲区还存有数据，需要待数据发送完毕后再关闭连接。**

## 流量控制和拥塞控制部分的验收

流量控制和拥塞控制的部分目前不能进行自动测试，为此提供了trace文件格式和绘图脚本。

同学们需要将绘图结果体现在报告中并加以解释。运行程序后生成trace文件，通过提供的绘图脚本绘制序列号、发送窗口大小等信息随时间变化的图像，进而展示流量控制和拥塞控制的过程。

注意，**不建议使用全自动的测试脚本进行抓包和绘图**，同学们最好采用手动方式调整网络环境，进行抓包和绘图，以保持最大的灵活性，获得符合要求的图像。

流量控制和拥塞控制的脚本和测试方法，请参考：



# 提示与补充说明

为方便同学开发，整理了往年实践中同学们频繁遇到的问题，并针对一些难点进行提示，形成了如下文档：



# 参考文献

1. RFC793, TRANSMISSION CONTROL PROTOCOL, <https://datatracker.ietf.org/doc/html/rfc793>
2. RFC5681,TCP congestion control, <https://datatracker.ietf.org/doc/html/rfc5681>
3. TCConfig download: <https://github.com/thombashi/tcconfig>
4. TCConfig document <https://tcconfig.readthedocs.io/en/latest/pages/introduction/index.html>