

天津大学

《计算机网络》课程设计 周进度报告



题目：TCP 可靠数据传输与流量控制

学 号：	3019244160 3020202184
姓 名：	程子姝 刘锦帆
学 院：	智能与计算学部
专 业：	计算机科学与技术
年 级：	2020 级
任课教师：	石高涛

2022 年 10 月 4 日

目 录

第一章	任务要求	1
1.1	可靠数据传输	1
1.2	流量控制	1
1.3	环境和语言	1
第二章	总体设计	2
2.1	可靠数据传输相关设计	2
2.2	模块功能设计	2
第三章	连接建立的设计	4
3.1	状态机设计	4
3.1.1	Client 端状态机设计	4
3.1.2	Server 端状态机设计	4
3.2	超时重传机制	5
3.3	RTO 计算法则	5
3.4	流量控制	6
第四章	可靠数据传输的实现	7
4.1	Client 端	7
4.1.1	Tju_Send 实现	7
4.2	Server 端	7
4.2.1	Tju_Recv 实现	7
4.3	通用	7
4.3.1	Socket 数据结构	7

4.3.2 超时重传机制	7
第五章 连接建立的功能测试与结果分析	9
5.1 测试环境及方法	9
5.2 RTO 动态调整	9
5.3 可靠数据传输	9
5.4 AutoLab 测试	10
5.5 人员分工	10

一 任务要求

1.1 可靠数据传输

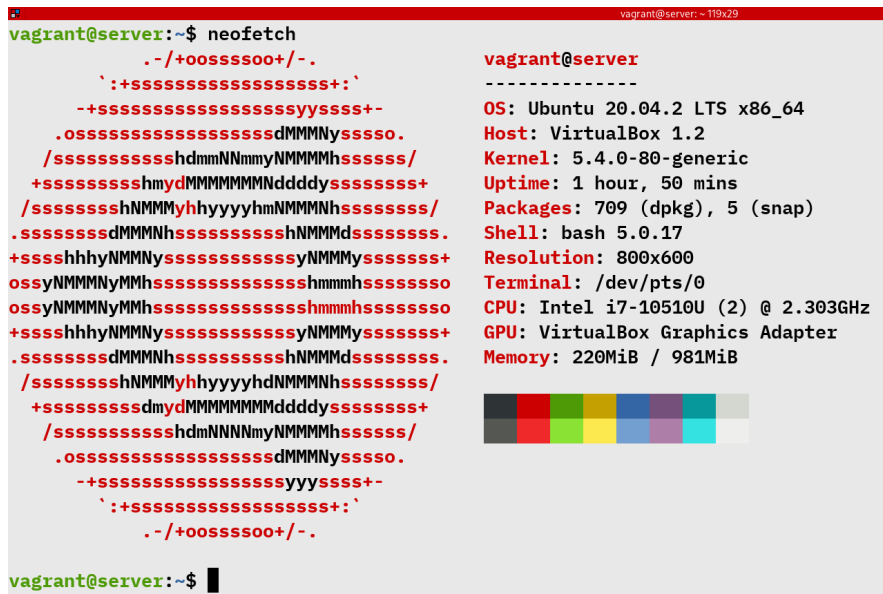
1. 需要在两端设置、管理发送和接收缓冲区，实现滑动窗口的机制（我们采用了 AVL，自平衡二叉搜索树的数据结构提升效率）
2. 需要依照 RFC793 Sec 3.7 的描述实现并行的可靠数据传输（即流水线技术）
3. 需要依照 RFC793 Sec 3.7 的描述估计 RTT，计算 RTO，并根据 RTO 进行数据包的超时重传
4. 能够在有不可靠的链路上实现至少 100 MB 数据的传输

1.2 流量控制

1. 发送方根据 “advertised window” 字段的信息调整发送窗口
2. 依据 RFC 793 Sec 3.7 的标准实现发送方应对接收方 0 窗口情况并进行规避

1.3 环境和语言

使用指定的 Linux 操作系统，C 语言实现 TJU_TCP 协议。系统配置如图1.3所示。



```
vagrant@server:~$ neofetch
      .-/+oosssso+/-.
      `:+ssssssssssss++:`
      -+ssssssssssssssyyssss+-
      .osssssssssssssssdMMMMyssso.
      /ssssssssshdmmNNmmyNMMMhsssss/
      +ssssssshmydMMMMMMNdddyssssss+
      /ssssssshNMMMyhhyyyhmNMMNhsssss/
      .ssssssdMMMNhssssssshNMMdssssss.
      +sssshhhyNMMNysssssssssyNMMMyssssss+
      ossyNMMMNyMMhssssssssshmmhssssssso
      ossyNMMMNyMMhssssssssshmmhssssssso
      +sssshhhyNMMNysssssssssyNMMMyssssss+
      .ssssssdMMMNhssssssshNMMdssssss.
      /ssssssshNMMMyhhyyyhdNMMNhsssss/
      +sssssssdmydMMMMMMNdddyssssss+
      /ssssssshdmmNNNmyNMMMhsssss/
      .osssssssssssssdMMMMyssso.
      -+ssssssssssssssyyssss+-
      `:+ssssssssssss++:`
      .-/+oosssso+/-.

vagrant@server:~$ cat /etc/os-release
OS: Ubuntu 20.04.2 LTS x86_64
Host: VirtualBox 1.2
Kernel: 5.4.0-80-generic
Uptime: 1 hour, 50 mins
Packages: 709 (dpkg), 5 (snap)
Shell: bash 5.0.17
Resolution: 800x600
Terminal: /dev/pts/0
CPU: Intel i7-10510U (2) @ 2.303GHz
GPU: VirtualBox Graphics Adapter
Memory: 220MiB / 981MiB
```

图 1-1 测试服务器配置信息

二 总体设计

根据 TCP 协议具有的功能和实践内容结合, 本次实验的协议设计分为以下三个部分, 分别是连接管理和流量控制、可靠数据传输和拥塞控制。

2.1 可靠数据传输相关设计

为了向应用层提供可靠的端到端数据传输服务。我们需要有连接建立、数据传输、连接关闭三个方面的功能。在本次实验中, 我们首先在第一周完成了连接建立的过程, 即通过三次握手完成 TCP 连接的建立。本周, 我们将重点完成第二个功能, 即可靠的数据传输服务。

相关的 API 如下:

1. Client

(a) `int tju_send(...)`: 将应用层的信息放入发送缓冲区, 并进行发送。

2. Server

(a) `int tju_recv(...)`: 接收由 `handle_packet` 等函数放入接收缓冲区的信息

3. 通用

(a) `tju_tcp_t * tju_socket()`: 创建 socket 并初始化数据结构 (包括发送、接收缓冲区等), 根据本周的要求, 还增加了关于时钟初始化、trace 文件初始化等操作, 最终返回 socket 的指针

(b) `tju_handle_packet()`: 处理接收到的包裹。在 Established 状态下, 根据接收缓冲区的大小, 选择性的将不按顺序到来的包裹放入接收树 (自定义数据结构) 或者丢弃。并在接收到正确包裹时, 将树中正确的包裹按顺序放入缓冲区中等待应用层获取。

2.2 模块功能设计

从图中可以看到, 我们通过一个 TIMER 线程专门检查所有 registered TIMER 并在其 TIMEOUT 时执行 RE-TRAN, 即重传。同时, 我们在 HANDLE_PACKET 函数, 在收到 ACK 时, 关闭响应的 TIMER (TIMER 的实现采用 queue 的数据结构, 确保快速增删)。

对于 Server 端, 我们在 HANDLE_PACKET 中设计了根据 SEQ number 的检验功能, 当 SEQ number 正确时, 我们直接将内容放入缓冲区, 否则将其放入一颗 AVL 树中等待正确 SEQ 达到后, 再进行查找并放入缓冲区, 确保缓冲区中的内容都是连贯一致的。

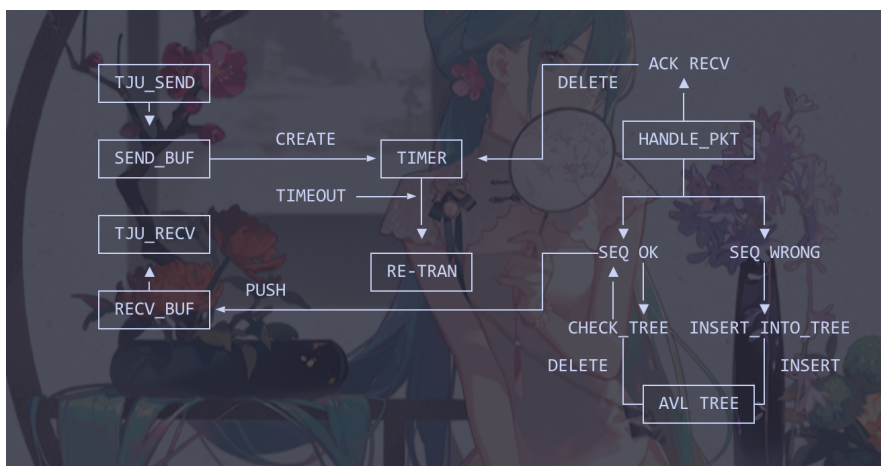


图 2-1 可靠数据传输相关功能设计总览

三 连接建立的设计

该部分涉及到的 TCP 的状态转变如下图??所示：

3.1 状态机设计

TCP 可靠数据传输过程如下所示，对于出现丢包的情况，在 Client 端采用重传机制，不在 Server 端关于 ACK 丢掉的处理机制。同时，在 Server 端采用了 AVL Tree 的数据结构存储乱序到达的数据包，确保最终呈现在接收缓冲区的内容是有序的。

3.1.1 Client 端状态机设计

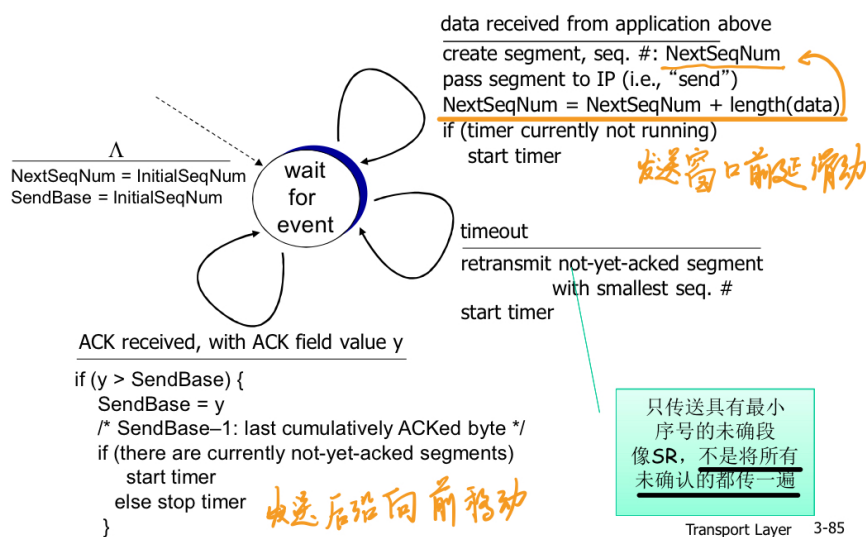


图 3-1 Client FSM

Client 端状态机如图??所示。Client 端需要响应三种事件：

1. 来自上层的调用，将数据打包并发送给下一层实体（toLayer3），然后新建 TIMER 并注册之
2. 来自 TIMEOUT 通过 TIMER 的信息进行超时重传（需要调整一下 rtt 等信息）
3. 收到 ACK，确认 ACK 的状态，根据 ACK 信息选择性地关闭 TIMER

3.1.2 Server 端状态机设计

在 Server 端，我们目前暂不考虑数据出错的情况，即达到的包，都是正确的。在此情况下，我们需要处理一下情况：

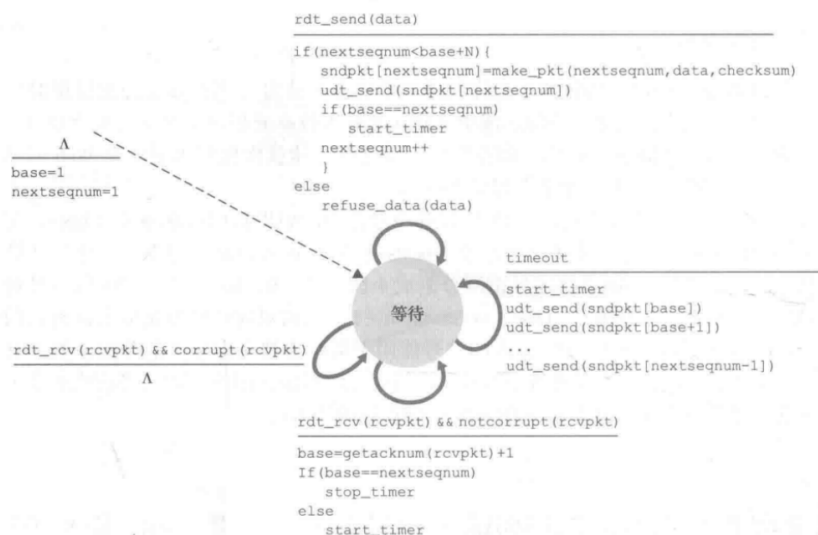


图 3-2 Server FSM

1. 到达 SEQ 正确：判断缓冲区大小，选择性的将其放入缓冲区，发送 ACK。
同时在 AVL 中递归地查找连续的正确包裹
2. 到达 SEQ 大了：将其放入 AVL Tree 并发送 ACK
3. 到达 SEQ 小了：收到过，丢弃

3.2 超时重传机制

连接建立时 丢失有三种：第一次握手丢失，服务端返回第二次握手丢失，以及第三次确认丢失。第一、二个由 Client 端设置定时器保护，第三个由 Server 端设置定时器保护，即当收到第三次确认时，Server 端再将信息定时器进行关闭。

发送数据时 丢失有两种：发送数据丢失 (Seq)，发送 ACK 丢失。二者对于 Client 端的表现均是长时间 (RTO) 没收到 ACK 信息，于是我们将定时器设置在 Client 端，由 Client 端进行超时重传。而 Server 只对于前者，即发送数据丢失的情况作出反馈，即设置乱序到达，顺序提交的机制，确保因超时而呈现乱序到达的数据得以顺序提交给上层用户。

3.3 RTO 计算法则

RTO 是超时重传机制的基础。由于网络变化，RTO 应当是动态调整的。如果 TCP 过早重传，会导致注入多数不必要的报文，阻塞网络。如果过晚重传，则会影响网络使用效率。我们根据 RFC793 的标准，让 Socket 维护了一个 SampleRTT 均值 (EstimatedRTT) 通过如下公式进行更新：

$$EstimatedRTT := (1 - \alpha) \times EstimatedRTT + \alpha \times SampleRTT \quad (3-1)$$

除了 RTT, RFC 6298 还定义了 RTT 的偏差 $DevRTT$ 的计算, 用于 $usuan$ $SampleRTT$ 和 $EstimatedRTT$ 的偏离程度

$$DevRTT := (1 - \beta) \times DevRTT + \beta \times |SampleRTT - EstimatedRTT| \quad (3-2)$$

于是, 最终的 TRO 计算公式如下:

$$RTO := EstimatedRTT + 4 \times DevRTT \quad (3-3)$$

3.4 流量控制

流量控制也是 TCP 标准功能之一, 是构成可靠传输的关键步骤, 能够与其他 TCP 实体一起, 维护整个网络的传输可靠性。如果流量过大, 则会导致接收方不断丢弃传输的结果, 发送方不断进行重发。故而, Server 端需要将 $rwnd$ 的信息通过 ACK 信息携带给 Client 端, 让 Client 能够正确地调整发送速率。

四 可靠数据传输的实现

4.1 Client 端

4.1.1 Tju_Send 实现

在 Tju_Send 中，首先需要判断上层内容的大小，若过大，则需要进行切片，以免在下层被切割。然后在进入阻塞状态，等待发送窗口出现空闲。然后交给发送窗口，同时调用发送函数，在 rwnd 的情况下将发送缓冲区的内容进行发送（此处在后期需要调整为线程）

4.2 Server 端

4.2.1 Tju_Recv 实现

在 Tju_Recv 中，首先进行阻塞等待，直到其他线程给接收缓冲区放入内容。待有内容后，再将内容通过 Memcpy 提交给上层调用的实体。

4.3 通用

4.3.1 Socket 数据结构

- 给 window.wnd_send 增加了 rwnd 和 cwnd 等控制发送的信息
- 给 window.wnd_recv 增加了 AVL Tree 等数据结构，用于存放乱序到达的数据包
- 给 window.wnd_send 增加了 rto 等字段，动态控制超时事件
- 给 window.wnd_recv 增加了 expe_seq 等字段，判断接收到的数据顺序正确与否

4.3.2 超时重传机制

超时重传定时器

实现了 Timer_Helper 子系统，通过设置 set_timer() 函数来新建一个 Timer，Delete_timer() 来终止一个 Timer。使用了一个线程不断检查是否有超时的 Timer，并进行重传和重置。使用 Mutex Lock 进行存储区域的一致性，使得在增加、删除和检查时只能有一个线程存在。

特别的，我们使用了链表的数据结构进行 Timer 的增加、删除和检查，链表能够高效地进行增加删除。同时，设置了 event 数据类型来处理当前 Timer 的超时操作（可以通过传入不同的函数和变量达到不同类型 Timer 的统一调用）。

RTO 调整

由于我们每个需要重传的报文都对应一个 `TIMER`，所以我们能通过发送时的 `Created_At` 和删除时的系统事件算出当前 `Timer` 提供的 `sampleRTT` 并在每次进行 `Timer` 删除时，通过公式 3.3 进行 `RTO` 的更新。

值得注意的是，这里依然需要大量使用 `Mutex Locker` 来控制：在更新 `RTO` 时，不能创建新的 `Timer`，直到 `RTO` 更新完成。

流量控制

接收方发送 `ACK` 时，将自己缓冲区大小放入 `advertised_wind` 字段。发送方在接收到 `ACK` 时，需要提取 `rwnd` 值。在设置发送窗口的大小时，我们需要考虑 `rwnd`（发送窗口大小），`cwnd`（拥塞控制），以及自己本身的大小。

当得到 `rwnd == 0` 的情况时，发送方需要发送 1 比特的试探报文进行发送窗口的试探，同时需要设置超时机制进行不断试探。

日志记录

实现了日志 `trace` 模块，在 `Server Client` 端调用 `tju_sock` 的时候进行日志的初始化（即定义日志名，创建日志文件等）。然后通过提供的日志格式和事件，在相应事件发生时，调用响应函数进行日志的录入。

五 连接建立的功能测试与结果分析

5.1 测试环境及方法

通过 VirtualBox 虚拟机进行测试，脚本默认丢包率 6%，延迟 6ms。

测试方法是使用脚本和修改后的测试脚本进行运行，分析输出的 log 进行 debug

5.2 RTO 动态调整

如图是 RTO 根据获得的 EstimatedRTT 进行的调整

```

50 [1664877134018] [RTT] [SampleRTT:0.657533 EstablishedRTT:0.626699 DeviationRtt:0.030946 TimeoutInterval:0.750483]
49 [1664877134018] [RWND] [size:44000]
48 [1664877134019] [RECV] [seq:0 ack:1146417 flag:ACK]
47 [1664877134020] [SEND] [seq:2139961 ack:0 flag:]
46 [1664877134021] [SEND] [seq:2139961 ack:0 flag:]
45 [1664877134021] [SEND] [seq:2141337 ack:0 flag:]
44 [1664877134022] [SEND] [seq:2141337 ack:0 flag:]
43 [1664877134022] [SWND] [size:44000]
42 [1664877134023] [RTT] [SampleRTT:0.660575 EstablishedRTT:0.630934 DeviationRtt:0.033143 TimeoutInterval:0.763508]
41 [1664877134023] [RWND] [size:44000]
40 [1664877134023] [RECV] [seq:0 ack:1147793 flag:ACK]
39 [1664877134025] [SEND] [seq:2141713 ack:0 flag:]
38 [1664877134027] [SEND] [seq:2141713 ack:0 flag:]
37 [1664877134028] [RTT] [SampleRTT:0.665238 EstablishedRTT:0.635222 DeviationRtt:0.034014 TimeoutInterval:0.771279]
36 [1664877134028] [RWND] [size:44000]
35 [1664877134029] [RECV] [seq:0 ack:1149169 flag:ACK]
34 [1664877134032] [RTT] [SampleRTT:0.664263 EstablishedRTT:0.638852 DeviationRtt:0.030284 TimeoutInterval:0.759989]
33 [1664877134033] [RWND] [size:44000]
32 [1664877134034] [RECV] [seq:0 ack:1150545 flag:ACK]
31 [1664877134040] [RTT] [SampleRTT:0.671368 EstablishedRTT:0.642917 DeviationRtt:0.031958 TimeoutInterval:0.770748]
30 [1664877134040] [RWND] [size:44000]
29 [1664877134041] [RECV] [seq:0 ack:1150921 flag:ACK]
28 [1664877134041] [SEND] [seq:2143089 ack:0 flag:]
27 [1664877134042] [SEND] [seq:2143089 ack:0 flag:]
26 [1664877134042] [SEND] [seq:2144465 ack:0 flag:]
25 [1664877134042] [SEND] [seq:2144465 ack:0 flag:]
24 [1664877134044] [SEND] [seq:2145841 ack:0 flag:]
23 [1664877134045] [SEND] [seq:2145841 ack:0 flag:]
22 [1664877134047] [SEND] [seq:2147217 ack:0 flag:]
21 [1664877134048] [SEND] [seq:2147217 ack:0 flag:]
20 [1664877134050] [RTT] [SampleRTT:0.681771 EstablishedRTT:0.647773 DeviationRtt:0.037130 TimeoutInterval:0.796295]
19 [1664877134051] [RWND] [size:44000]
18 [1664877134051] [RECV] [seq:0 ack:1232361 flag:ACK]
17 [1664877134051] [SEND] [seq:2148593 ack:0 flag:]
16 [1664877134052] [SEND] [seq:2148593 ack:0 flag:]
15 [1664877134053] [RTT] [SampleRTT:0.683951 EstablishedRTT:0.652296 DeviationRtt:0.036416 TimeoutInterval:0.797959]
14 [1664877134053] [RWND] [size:44000]
13 [1664877134054] [RECV] [seq:0 ack:1152297 flag:ACK]
12 [1664877134054] [SEND] [seq:2149969 ack:0 flag:]
11 [1664877134055] [SEND] [seq:2149969 ack:0 flag:]
10 [1664877134055] [SEND] [seq:2151345 ack:0 flag:]
9 [1664877134055] [SEND] [seq:2151345 ack:0 flag:]
8 [1664877134056] [SWND] [size:44000]
7 [1664877134057] [SEND] [seq:2151721 ack:0 flag:]
6 [1664877134060] [SEND] [seq:2151721 ack:0 flag:]
5 [1664877134060] [RTT] [SampleRTT:0.688660 EstablishedRTT:0.656841 DeviationRtt:0.036377 TimeoutInterval:0.802350]
4 [1664877134061] [RWND] [size:44000]
3 [1664877134061] [RECV] [seq:0 ack:1153673 flag:ACK]
2 [1664877134064] [SEND] [seq:2153097 ack:0 flag:]
1 [1664877134064] [SEND] [seq:2153097 ack:0 flag:]
668 [1664877134068] [RTT] [SampleRTT:0.696781 EstablishedRTT:0.661834 DeviationRtt:0.039049 TimeoutInterval:0.818029]
1 [1664877134069] [RWND] [size:44000]

```

图 5-1 网站测试结果

5.3 可靠数据传输

我们通过 Client 端的 Trace 可以看到接受到连续 ACK 表明接收到了正确的信息。

```

23 [1664877130334] [RECV] [seq:0 ack:1377 flag:ACK]
22 [1664877130334] [SEND] [seq:404449 ack:0 flag:]
21 [1664877130334] [SEND] [seq:404449 ack:0 flag:]
20 [1664877130334] [RTTS] [SampleRTT:0.041259 EstablishedRTT:3.837762 DeviationRtt:-4.261212 TimeoutInterval:0.050000]
19 [1664877130334] [RWND] [size:42625]
18 [1664877130334] [RECV] [seq:0 ack:2753 flag:ACK]
17 [1664877130334] [SEND] [seq:405825 ack:0 flag:]
16 [1664877130334] [SEND] [seq:405825 ack:0 flag:]
15 [1664877130334] [RTTS] [SampleRTT:0.041736 EstablishedRTT:3.363258 DeviationRtt:-3.974926 TimeoutInterval:0.050000]
14 [1664877130334] [RWND] [size:41250]
13 [1664877130334] [RECV] [seq:0 ack:4129 flag:ACK]
12 [1664877130339] [SEND] [seq:427217 ack:0 flag:]
11 [1664877130340] [RTTS] [SampleRTT:0.046709 EstablishedRTT:2.585482 DeviationRtt:-3.133166 TimeoutInterval:0.050000]
10 [1664877130340] [RWND] [size:38500]
9 [1664877130340] [RECV] [seq:0 ack:6881 flag:ACK]
8 [1664877130340] [RTTS] [SampleRTT:0.047279 EstablishedRTT:2.268206 DeviationRtt:-2.757862 TimeoutInterval:0.050000]
7 [1664877130340] [RWND] [size:37125]
6 [1664877130340] [RECV] [seq:0 ack:8257 flag:ACK]
5 [1664877130340] [SEND] [seq:428593 ack:0 flag:]
4 [1664877130340] [RTTS] [SampleRTT:0.047465 EstablishedRTT:1.990614 DeviationRtt:-2.426219 TimeoutInterval:0.050000]
3 [1664877130340] [RWND] [size:35750]
2 [1664877130340] [RECV] [seq:0 ack:9633 flag:ACK]
1 [1664877130341] [SEND] [seq:428593 ack:0 flag:]
s26 [1664877130341] [RTTS] [SampleRTT:0.048127 EstablishedRTT:1.747803 DeviationRtt:-2.135610 TimeoutInterval:0.050000]
1 [1664877130341] [RWND] [size:34375]
2 [1664877130341] [RECV] [seq:0 ack:10009 flag:ACK]
3 [1664877130341] [RTTS] [SampleRTT:0.048323 EstablishedRTT:1.535368 DeviationRtt:-1.880997 TimeoutInterval:0.050000]
4 [1664877130341] [RWND] [size:34000]

```

图 5-2 网站测试结果

5.4 流量控制

该部分由于系统原因，暂时无法运行提供脚本正确生成图像，将在下周的进度报告中补充说明。

5.5 AutoLab 测试

13	479404627@qq.com_13_handin.zip	2022-10-02 16:22:04 +0800	100.0	Submitted 0 days late 100.0
12	479404627@qq.com_12_handin.zip	2022-10-02 16:12:13 +0800	33.3	Submitted 0 days late 33.3
11	479404627@qq.com_11_handin.zip	2022-10-02 15:58:22 +0800	0.6	Submitted 0 days late 0.6
10	479404627@qq.com_10_handin.zip	2022-10-02 15:56:13 +0800	0.7	Submitted 0 days late 0.7

图 5-3 网站测试结果

5.6 人员分工

人员分工如表5-1所示。

人员	项目分工
程子姝	完成大部分代码工作，以及协议实现部分
刘锦帆	完成 client 和 server 端的处理和 debug.h 以及写作

表 5-1 人员分工表