拥塞控制改进报告

一、 拥塞控制原理

在网络通信中，拥塞控制是一项至关重要的技术，用于防止网络因过多的数据包涌入而导致的性能下降。本实验的目标是优化基于滑动窗口机制的传输协议，在此基础上添加入拥塞控制算法。报告中展现所实现的具体拥塞控制算法，相应的代码实现，以及效果展示。

拥塞控制的主要任务是在保证高效数据传输的同时，避免过多的数据包涌入网络，造成网络拥塞。TCP协议中的经典拥塞控制算法包括慢启动（Slow Start）、拥塞避免（Congestion Avoidance）和快速恢复（Fast Recovery）。这些机制通常与滑动窗口配合使用，控制每次可发送的数据量。

1. 慢启动（Slow Start）

在慢启动阶段，拥塞窗口（ `cwnd` ）从1开始，并随着每次收到确认（ACK）而快速增长。这一过程持续到 `cwnd` 达到一个阈值（ `ssthresh` ），此时算法转向拥塞避免阶段。

2. 拥塞避免（Congestion Avoidance）

当 cwnd 达到 ssthresh 后，拥塞控制进入拥塞避免阶段。在这个阶段， cwnd 会线性增长，通常每收到一个ACK确认，这里我 cwnd 每次增长 MSS*(MSS/cwnd) 。

3. 快速恢复（Fast Recovery）

当发生丢包时，TCP会通过快速恢复机制来提高恢复速度。通常是通过检测到重复的ACK来触发。当收到三个重复的ACK时， cwnd 会缩小并开始快速恢复，避免过多的重传导致不必要的延迟。

二、 代码实现

1. 窗口变量

```
int cwnd = 1;
int ssthresh = 10;
```

- cwnd：拥塞窗口，将其初始化为1开始递增
- ssthresh：慢启动阈值，设置为之前实验中的窗口大小

2. 慢启动阶段

```
if (cwnd < ssthresh) {
    cout << "Slow Start阶段: cwnd=" << cwnd << endl;
    cwnd++; // 慢启动阶段窗口大小每次增加1
}
```

- 在 cwnd 小于 ssthresh 时，在循环中检查cwnd，并在慢启动阶段指数级增长，每次成功接收ACK时，每次增加1。

3. 拥塞避免阶段

```
else {  // 拥塞避免阶段
    cout << "Congestion Avoidance阶段: cwnd=" << cwnd << endl;
    count++;
    // 每收到一个完整窗口的数据的ACK，cwnd增加才增加1
    if (count == cwnd) {
        cwnd++;
        count = 0;
    }
}
```

- 当 cwnd 达到 ssthresh 后，进入拥塞避免阶段， cwnd 的增长速度减慢。只有每收到一个完整窗口的数据的 ACK，窗口数增加1，即 count == cwnd 。

4. 快速恢复阶段

```
if (duplicateACKs == 3) {
    cout << "Fast Recovery阶段: 三次重复ACK，快速恢复!" << endl;
    ssthresh = cwnd / 2; // 将ssthresh设置为当前cwnd的一半
    cwnd = ssthresh + 3 * MSS;  // cwnd设置为ssthresh + 3个MSS
    // 重传丢失的报文段
    for (int i = 0; i < window.size(); i++) {
        sendPacket(sockfd, serveraddr, serveraddr_len, window[i]);
        cout << "[重新发送] Send packet " << window[i].seq << ", size: " << window[i].len << " b
    }
    duplicateACKs = 0;  // 重置重复ACK计数
    inFastRecovery = true;
    delay = chrono::duration<double>::zero();
}
```

- 快速恢复发生在接收到三次重复ACK 时。当三次重复 ACK 被检测到时，不等待超时，而是立即重传丢失的报文段并更新 ssthresh 和 cwnd 。将 ssthresh 设为当前 cwnd 的一半，并将 cwnd 设为 ssthresh 加3倍的 MSS 。
- 并重置计数 duplicateACKs

5. 接收端错误处理

同时还需要改进接收端的处理逻辑，在之前实验中如果没有收到与期望序列号相同的数据包则不做任何回应，等待发送端超时重传；而现在则要再次发送ack，也就是告诉发送端所期待的序列号

```
if (recvMsg.seq == expectedSeq) {
    cout <<"=======================================================" <<endl;
    cout << "Received packet " << recvMsg.seq << ", size: " <<recvMsg.len << " bytes" << endl;

    // // 模拟固定延迟
    // std::this_thread::sleep_for(std::chrono::milliseconds(500));

    // 发送 ACK
    sendMsg.type = ACK;
    sendMsg.ack = recvMsg.seq + 1;
    sendto(sockfd, &sendMsg, sizeof(sendMsg), 0, (const structsockaddr *)&cliaddr, cliaddr_len)
    expectedSeq++;
    cout << "校验和："<< calculateChecksum(recvMsg) << " 校验和正确"<< ", Send ack="<< sendMsg.ac

    if (recvMsg.len > 0) {
        output_file.write(recvMsg.data, recvMsg.len);
    }
    continue;
}
else {
    // 发送 ACK 以重传
    sendMsg.type = ACK;
    sendMsg.ack = expectedSeq;
    sendto(sockfd, &sendMsg, sizeof(sendMsg), 0, (const structsockaddr *)&cliaddr, cliaddr_len)
    cout << "[重新发送] Send ack="<< sendMsg.ack << endl;
}
```

- 将 `expectedSeq` 再次作为ack发送给发送端，发送端会有快速回复阶段的处理

6. 超时处理

```
if (delay > timeout) {
    cout << "Timeout发生，进行重传..." << endl;
    ssthresh = cwnd / 2;  // 将ssthresh设置为cwnd的一半
    handleTimeout(sockfd, serveraddr, serveraddr_len, window); // 重传
    cwnd = 1;  // 重置cwnd为1
}
```

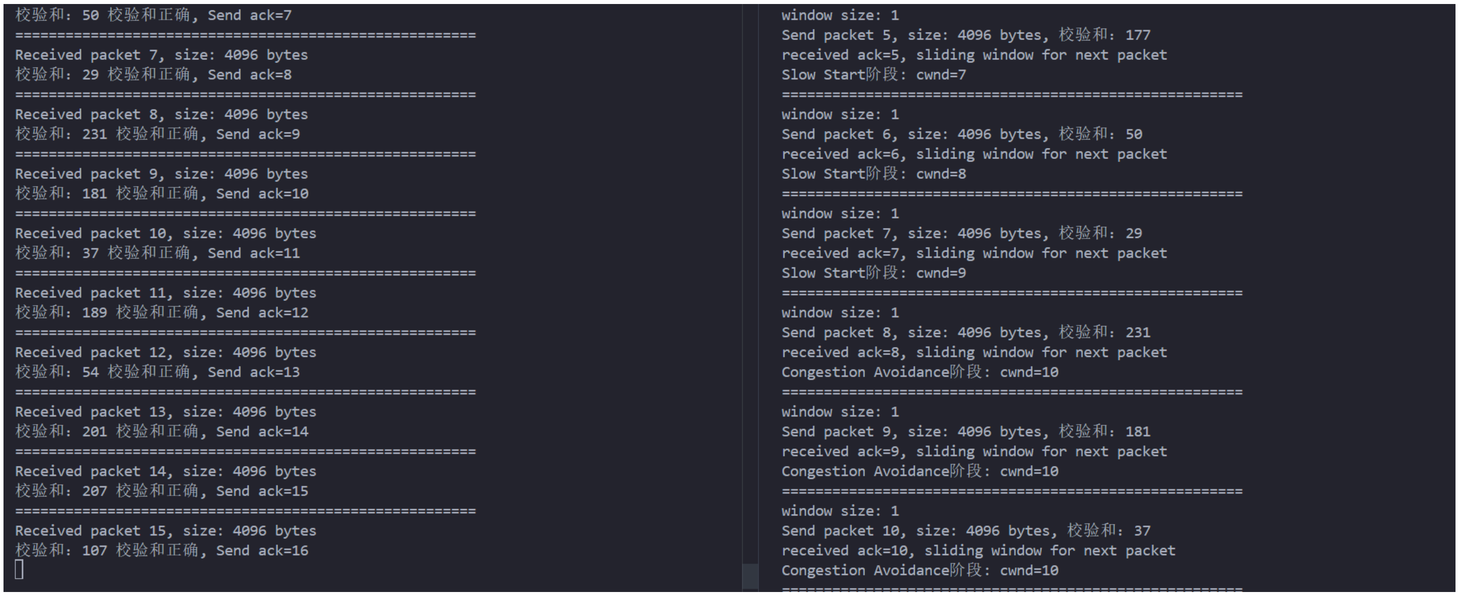
- 当传输数据包的确认时间过长时，触发超时重传，并将 `ssthresh` 和 `cwnd` 调整为cwnd的一半，以便快速恢复。
- 重置cwnd窗口大小为1

三、 效果演示

为了体现拥塞控制算法的实现，设定：

- ssthresh = 10
- PACKET_LOSS_RATE 0.5 // 丢包率

1. 慢启动到拥塞避免

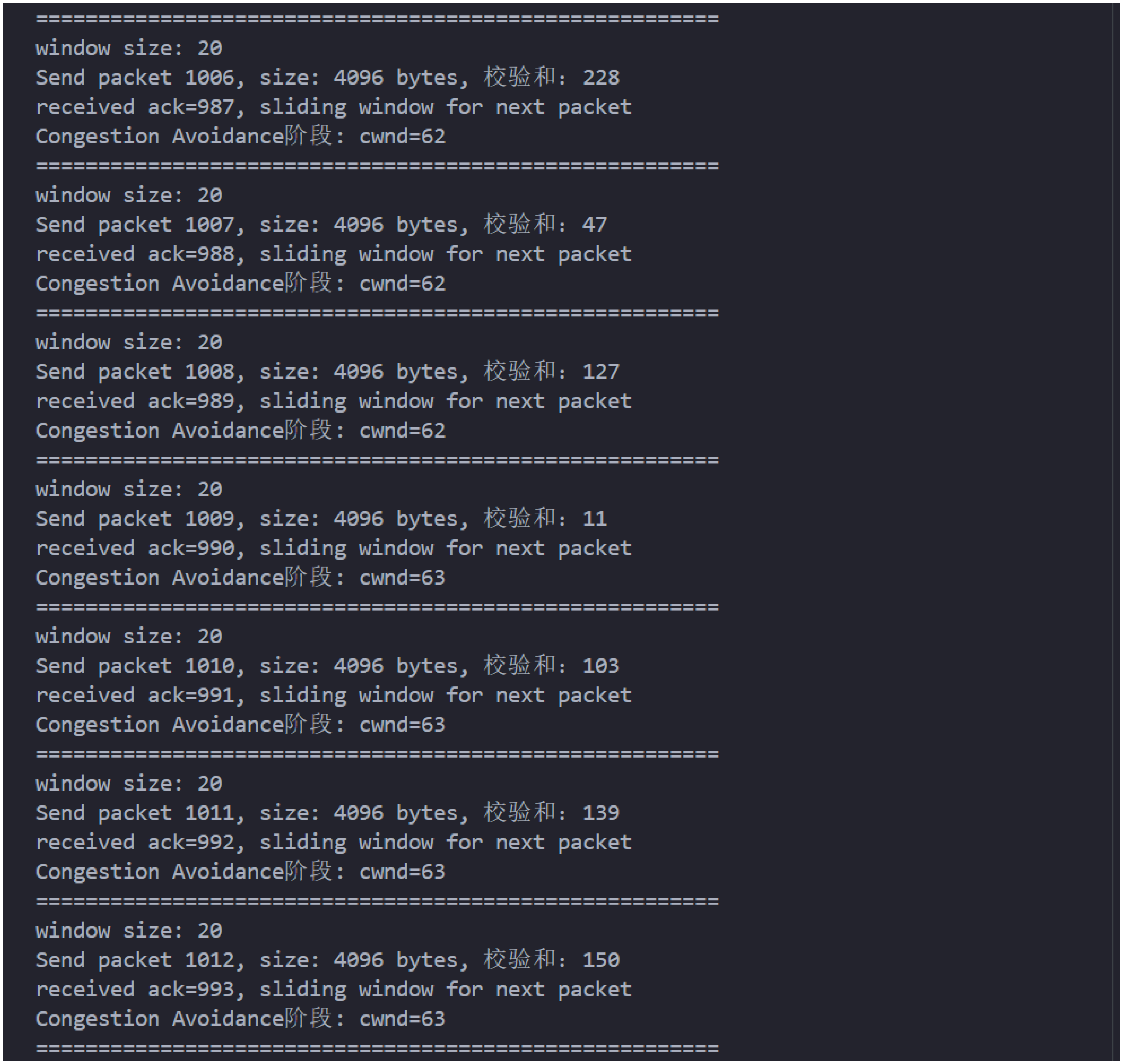


```
校验和：50 校验和正确, Send ack=7
=======================================================
Received packet 7, size: 4096 bytes
校验和：29 校验和正确, Send ack=8
=======================================================
Received packet 8, size: 4096 bytes
校验和：231 校验和正确, Send ack=9
=======================================================
Received packet 9, size: 4096 bytes
校验和：181 校验和正确, Send ack=10
=======================================================
Received packet 10, size: 4096 bytes
校验和：37 校验和正确, Send ack=11
=======================================================
Received packet 11, size: 4096 bytes
校验和：189 校验和正确, Send ack=12
=======================================================
Received packet 12, size: 4096 bytes
校验和：54 校验和正确, Send ack=13
=======================================================
Received packet 13, size: 4096 bytes
校验和：201 校验和正确, Send ack=14
=======================================================
Received packet 14, size: 4096 bytes
校验和：207 校验和正确, Send ack=15
=======================================================
Received packet 15, size: 4096 bytes
校验和：107 校验和正确, Send ack=16
```

```
window size: 1
Send packet 5, size: 4096 bytes, 校验和：177
received ack=5, sliding window for next packet
Slow Start阶段：cwnd=7
=======================================================
window size: 1
Send packet 6, size: 4096 bytes, 校验和：50
received ack=6, sliding window for next packet
Slow Start阶段：cwnd=8
=======================================================
window size: 1
Send packet 7, size: 4096 bytes, 校验和：29
received ack=7, sliding window for next packet
Slow Start阶段：cwnd=9
=======================================================
window size: 1
Send packet 8, size: 4096 bytes, 校验和：231
received ack=8, sliding window for next packet
Congestion Avoidance阶段：cwnd=10
=======================================================
window size: 1
Send packet 9, size: 4096 bytes, 校验和：181
received ack=9, sliding window for next packet
Congestion Avoidance阶段：cwnd=10
=======================================================
window size: 1
Send packet 10, size: 4096 bytes, 校验和：37
received ack=10, sliding window for next packet
Congestion Avoidance阶段：cwnd=10
=======================================================
```

- cwnd每次接受到新的ack增加1，在cwnd到达ssthresh = 10时变为拥塞避免状态

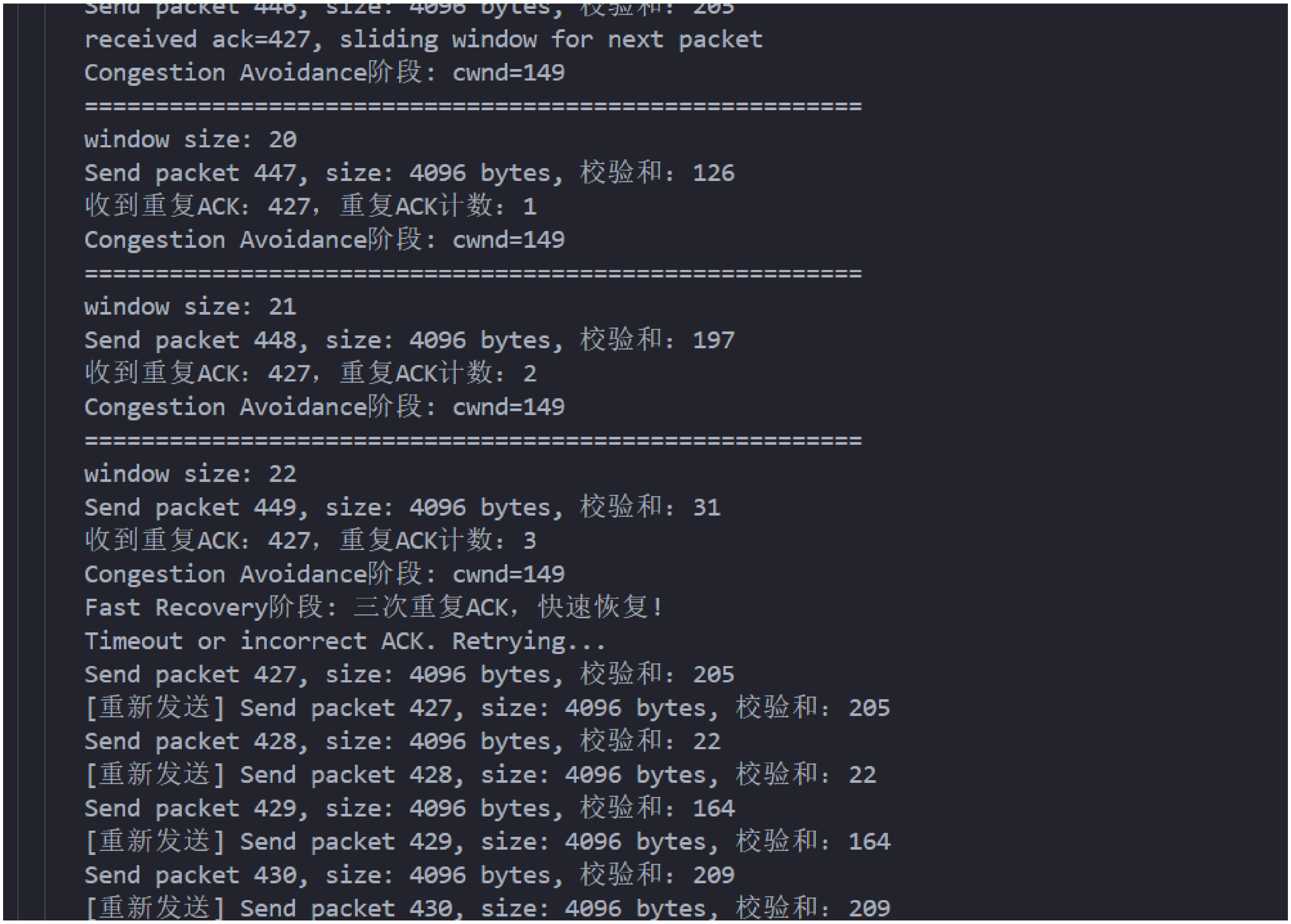
3. 拥塞避免阶段



```
=======================================================
window size: 20
Send packet 1006, size: 4096 bytes, 校验和：228
received ack=987, sliding window for next packet
Congestion Avoidance阶段：cwnd=62
=======================================================
window size: 20
Send packet 1007, size: 4096 bytes, 校验和：47
received ack=988, sliding window for next packet
Congestion Avoidance阶段：cwnd=62
=======================================================
window size: 20
Send packet 1008, size: 4096 bytes, 校验和：127
received ack=989, sliding window for next packet
Congestion Avoidance阶段：cwnd=62
=======================================================
window size: 20
Send packet 1009, size: 4096 bytes, 校验和：11
received ack=990, sliding window for next packet
Congestion Avoidance阶段：cwnd=63
=======================================================
window size: 20
Send packet 1010, size: 4096 bytes, 校验和：103
received ack=991, sliding window for next packet
Congestion Avoidance阶段：cwnd=63
=======================================================
window size: 20
Send packet 1011, size: 4096 bytes, 校验和：139
received ack=992, sliding window for next packet
Congestion Avoidance阶段：cwnd=63
=======================================================
window size: 20
Send packet 1012, size: 4096 bytes, 校验和：150
received ack=993, sliding window for next packet
Congestion Avoidance阶段：cwnd=63
=======================================================
```

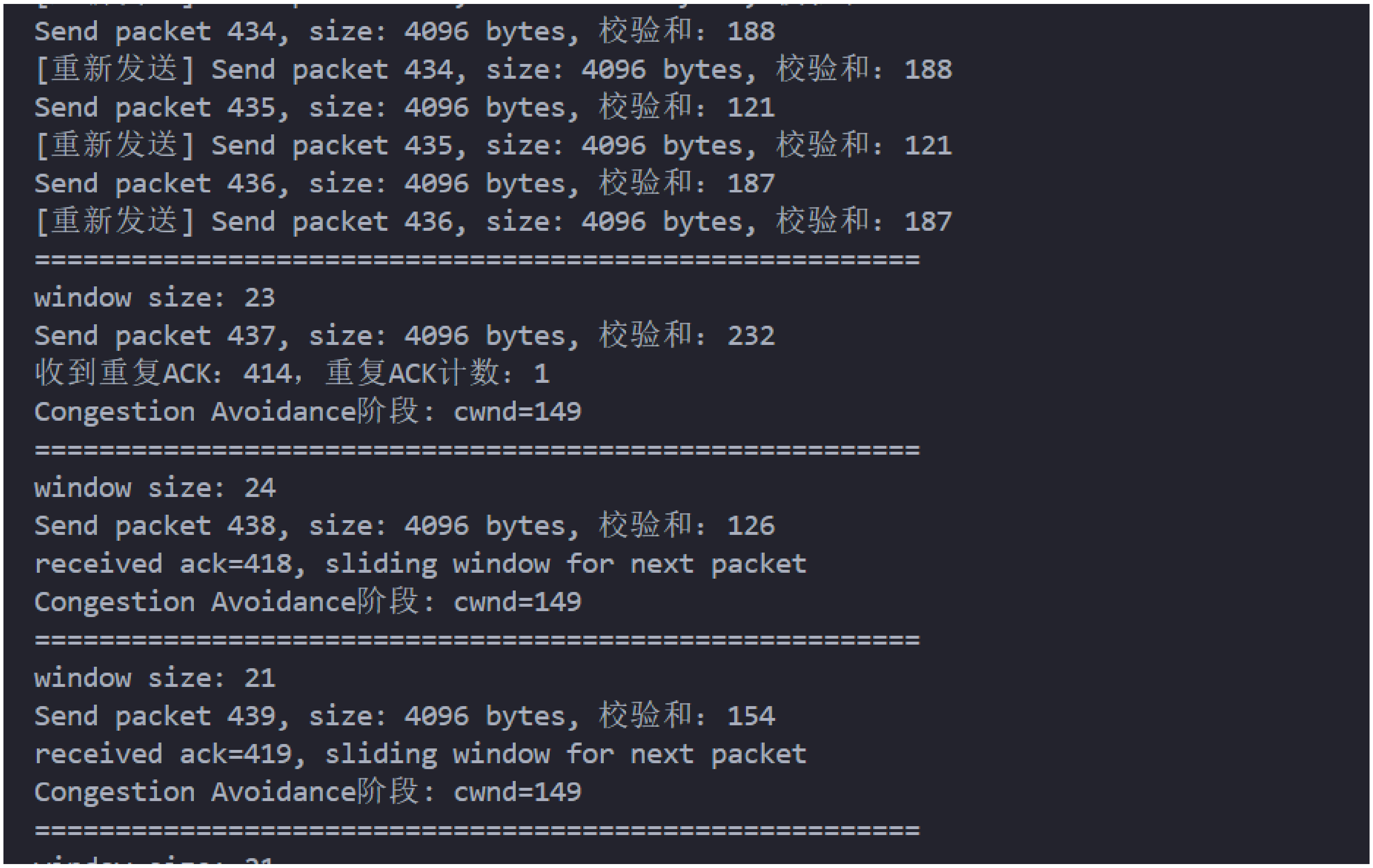
- 使用count计数，每收到一个完整窗口（cwnd）的数量的ACK，cwnd增加1

3. 快速回复阶段



```
Send packet 446, size: 4096 bytes, 校验和: 205
received ack=427, sliding window for next packet
Congestion Avoidance阶段: cwnd=149
=======================================================
window size: 20
Send packet 447, size: 4096 bytes, 校验和: 126
收到重复ACK: 427, 重复ACK计数: 1
Congestion Avoidance阶段: cwnd=149
=======================================================
window size: 21
Send packet 448, size: 4096 bytes, 校验和: 197
收到重复ACK: 427, 重复ACK计数: 2
Congestion Avoidance阶段: cwnd=149
=======================================================
window size: 22
Send packet 449, size: 4096 bytes, 校验和: 31
收到重复ACK: 427, 重复ACK计数: 3
Congestion Avoidance阶段: cwnd=149
Fast Recovery阶段: 三次重复ACK, 快速恢复!
Timeout or incorrect ACK. Retrying...
Send packet 427, size: 4096 bytes, 校验和: 205
[重新发送] Send packet 427, size: 4096 bytes, 校验和: 205
Send packet 428, size: 4096 bytes, 校验和: 22
[重新发送] Send packet 428, size: 4096 bytes, 校验和: 22
Send packet 429, size: 4096 bytes, 校验和: 164
[重新发送] Send packet 429, size: 4096 bytes, 校验和: 164
Send packet 430, size: 4096 bytes, 校验和: 209
[重新发送] Send packet 430, size: 4096 bytes, 校验和: 209
```

- 由于丢包造成接收端重复发送ack，接收端接受与上一次相同ack达到3次，进入快速回复阶段，并立刻重发当前窗口内的所有序列包
- 这是提前避免超时重传，所以还需将计时清零，防止又因为超时进行重传

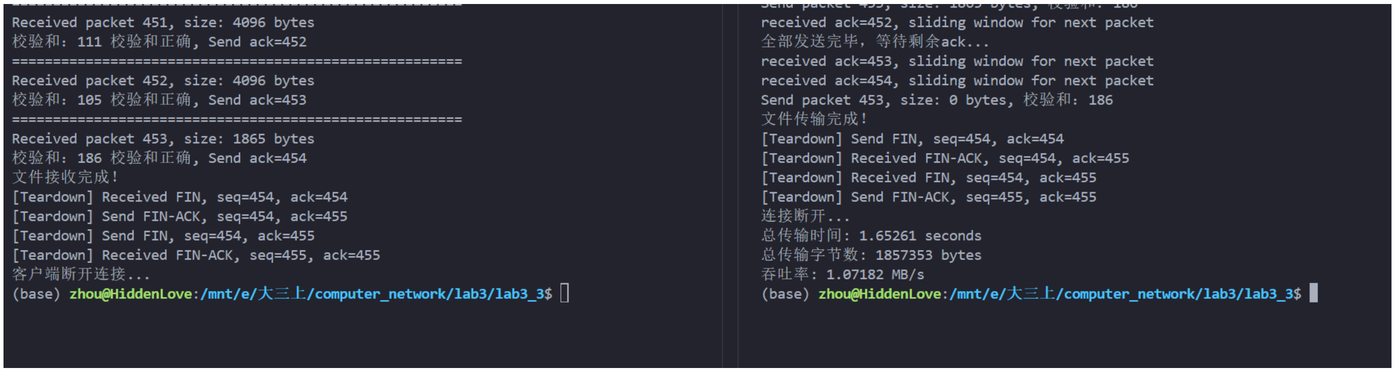


```
Send packet 434, size: 4096 bytes, 校验和: 188
[重新发送] Send packet 434, size: 4096 bytes, 校验和: 188
Send packet 435, size: 4096 bytes, 校验和: 121
[重新发送] Send packet 435, size: 4096 bytes, 校验和: 121
Send packet 436, size: 4096 bytes, 校验和: 187
[重新发送] Send packet 436, size: 4096 bytes, 校验和: 187
=======================================================
window size: 23
Send packet 437, size: 4096 bytes, 校验和: 232
收到重复ACK: 414, 重复ACK计数: 1
Congestion Avoidance阶段: cwnd=149
=======================================================
window size: 24
Send packet 438, size: 4096 bytes, 校验和: 126
received ack=418, sliding window for next packet
Congestion Avoidance阶段: cwnd=149
=======================================================
window size: 21
Send packet 439, size: 4096 bytes, 校验和: 154
received ack=419, sliding window for next packet
Congestion Avoidance阶段: cwnd=149
=======================================================
window size: 21
```

- 将ssthresh设为当前cwnd的一半，并将cwnd设为ssthresh加3倍的MSS
- 为了体现过程将将丢包率提高

4. 传输结果

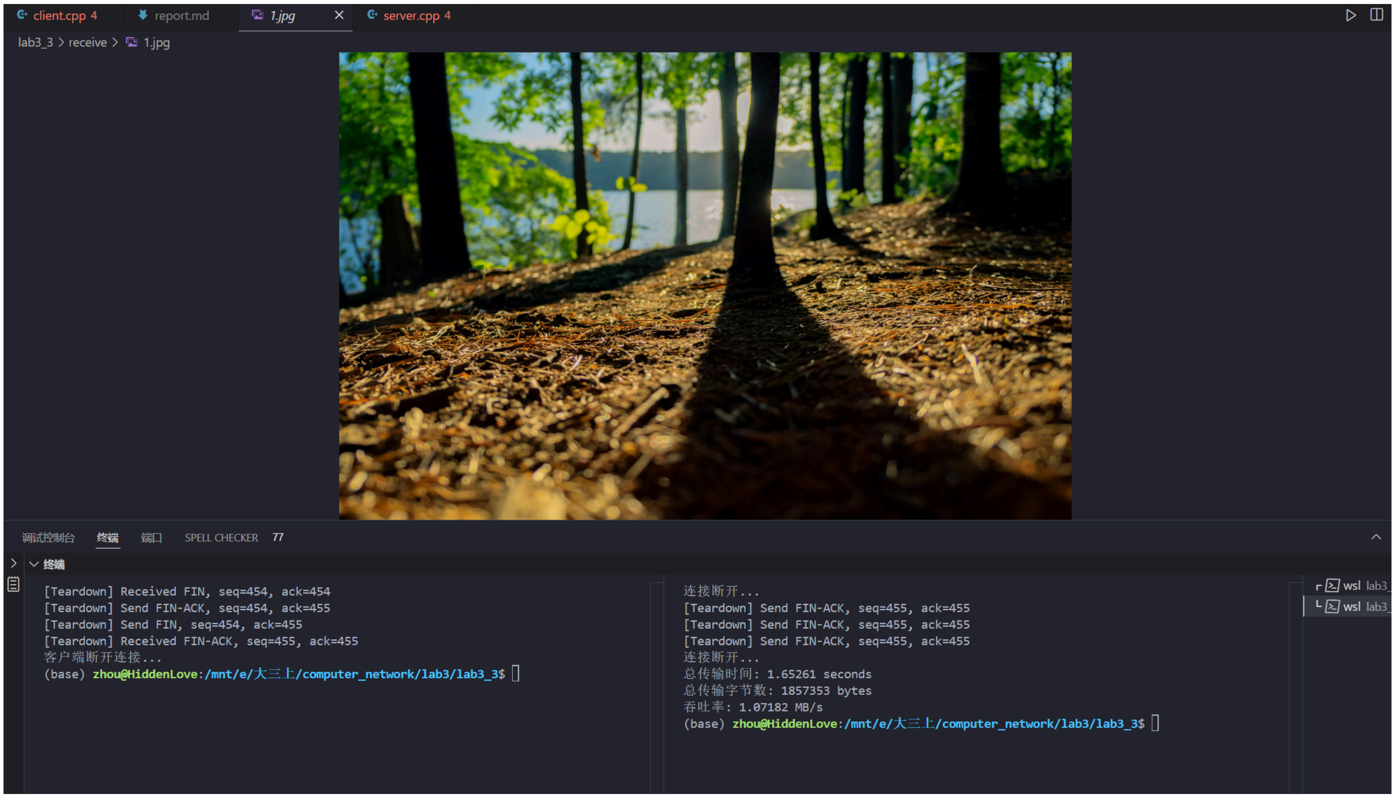
- 在丢包率为0，延迟为50ms时



```
=======================================================
Received packet 451, size: 4096 bytes
校验和: 111 校验和正确, Send ack=452
=======================================================
Received packet 452, size: 4096 bytes
校验和: 105 校验和正确, Send ack=453
=======================================================
Received packet 453, size: 1865 bytes
校验和: 186 校验和正确, Send ack=454
文件接收完成!
[Teardown] Received FIN, seq=454, ack=454
[Teardown] Send FIN-ACK, seq=454, ack=455
[Teardown] Send FIN, seq=454, ack=455
[Teardown] Received FIN-ACK, seq=455, ack=455
客户端断开连接...
(base) zhou@HiddenLove:/mnt/e/大三上/computer_network/lab3/lab3_3$
```

```
Send packet 453, size: 1865 bytes, 校验和: 186
received ack=452, sliding window for next packet
全部发送完毕, 等待剩余ack...
received ack=453, sliding window for next packet
received ack=454, sliding window for next packet
Send packet 453, size: 0 bytes, 校验和: 186
文件传输完成!
[Teardown] Send FIN, seq=454, ack=454
[Teardown] Received FIN-ACK, seq=454, ack=455
[Teardown] Received FIN, seq=454, ack=455
[Teardown] Send FIN-ACK, seq=455, ack=455
连接断开...
总传输时间: 1.65261 seconds
总传输字节数: 1857353 bytes
吞吐率: 1.07182 MB/s
(base) zhou@HiddenLove:/mnt/e/大三上/computer_network/lab3/lab3_3$
```

- 吞吐率: 1.6526 MB/s



```
[Teardown] Received FIN, seq=454, ack=454
[Teardown] Send FIN-ACK, seq=454, ack=455
[Teardown] Send FIN, seq=454, ack=455
[Teardown] Received FIN-ACK, seq=455, ack=455
客户端断开连接...
(base) zhou@HiddenLove:/mnt/e/大三上/computer_network/lab3/lab3_3$
```

```
连接断开...
[Teardown] Send FIN-ACK, seq=455, ack=455
[Teardown] Send FIN-ACK, seq=455, ack=455
[Teardown] Send FIN-ACK, seq=455, ack=455
连接断开...
总传输时间: 1.65261 seconds
总传输字节数: 1857353 bytes
吞吐率: 1.07182 MB/s
(base) zhou@HiddenLove:/mnt/e/大三上/computer_network/lab3/lab3_3$
```

- 图片正常传输