天津大学

RDT 实验报告

学生姓名 赵万旭

学院名称 智能与计算学部

班　　级 软件工程 2 班

学　　号 3019244207

一、实验目的

1. 深入体会可靠数据传输的思想和理念。
2. 加深对 Stop-and-Wait 和 Go-Back-N 两种协议的理解。
3. 掌握 Stop-and-Wait 和 Go-Back-N 两种协议的具体实现方式。

二、实验内容

编写传输层代码，实现单向传输情景下的 Stop-and-Wait 和 Go-Back-N 两种协议。

为专注于协议的开发，本次实验已经提供了诸如网络仿真过程的模拟代码、数据报具体收发的代码、节点除传输层以外的各层功能代码等大量基础代码。这些基础代码已经构建起了一套完善的网络仿真环境。

只有每个节点的传输层代码部分留空，需要实验人员自行填补，具体包括节点传输层的初始化操作、节点接收到应用层消息的处理过程、节点接收到网络层数据包的处理过程、节点计时器到时的响应过程等内容。

为方便试验人员进行实验，本实验还提供了以下实现好的过程供试验人员调用：

starttimer()和 stoptimer ()：试验人员可以通过调用此函数启动和停止节点的计时器。

tolayer3(calling_entity, packet)：试验人员可以通过调用此函数时，传入需要发送的 packet，将数据包传递至网络层。这之后的传输工作将由框架自动完成

tolayer5(calling_entity, message)：试验人员可以通过调用此函数时，传入 message，将消息传递至应用层。这之后仿真框架将检查数据是否完好、是否按序到达。

实验人员完成所有节点的算法后，运行网络仿真，记录仿真过程和结果，并对结果进行分析和总结。

三、实验方式

实验环境：

操作系统：Windows 10

编程语言：c 语言

编程环境：vscode

四、停止等待协议

1. 实验原理

如图为停止等待协议的原理图，参考 rdt3.0

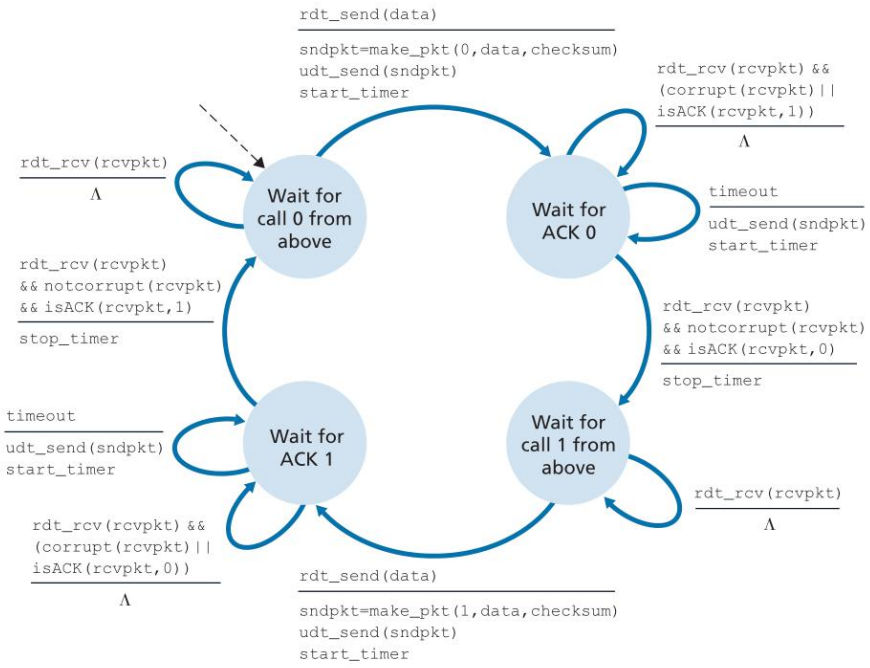


Figure 3.15 ♦ rdt3.0 sender

rdt 协议经历了 rdt1.0，rdt2.0，rdt2.1，rdt2.2，rdt3.0.一步

步完善，使得网络得到很好的安全性稳定性。

rdt1.0 是基于理想情况下的协议，假设所有信道都是可靠的，没有比特位的翻转，没有数据包的丢失与超时，所以 rdt1.0 的传输功能就是 发送方发送数据，接收方接受数据。

rdt2.0 在 rdt1.0 的基础上解决了比特位翻转的问题，这里的比特位防撞发生在运输层下面的不可信信道中数据包中的 1 可能会变 0，0 可能会变成 1。rdt2.0 增加了 3 种新机制：1.错误检验 2.接收者反馈接受信息（ACK,NAK）3.重传机制。在运输层对应用层的数据进行打包处理时，新增 checksum（校验和），从而接收端可以对其数据包进行检验，如果正确，返回 ACK，发送者继续发送下一个数据包；如果不正确，返回 NAK，发送者重传数据。

rdt2.1 在 rdt2.0 的基础之上，发送方在打包数据包时添加了 0 或者 1 编号，同样 ACK,NAK 字段上也添加了 0，1 字段，表示 0.1 号字段的确认或者否定。发送方就有了 2 种状态发送 0 号数据包，1 号数据包，接收方也有了 2 种状态等待 0 号数据包和等待 1 号数据包。现在假设情景发送方向接收方发送 0 号数据包，如果接收方接收到 0 号数据包，返回 ACK，但是 ACK 出现翻转，接收方处于等待 1 号数据状态，发送方重复发送 0 号数据，接收方会拒绝 0 号数据，避免重复。如果接收方接收到 0 号数据包出现错误，返回 NAK，但是 NAK 出现翻转，接收方处于等待 0 号数据状态，发送方继续发送 1 号数据，接收方会拒绝 1 号数据，避免错序。

rdt2.2 之前的版本都重在处理数据包的比特位翻转情况，却没有考虑到数据包在传输过程中出现的数据包丢失问题，这样数据包丢失会使得网络处于拥塞状态。

rdt3.0 在 rdt2.2 的基础之上处理了数据包丢失的情况，增加了计时器的机制，如果在 RTT 时间段内，发送方没有接收到反馈信息，

2. 算法实现

那么发送方默认数据包已经丢失了，会自动重传。

下面描述一下算法实现

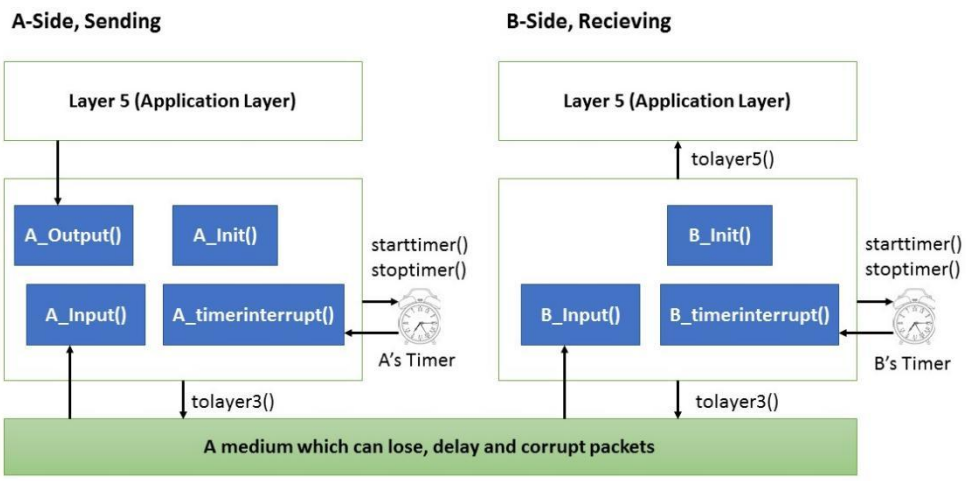
```
struct pkt {
    int seqnum;
    int acknum;
    int checksum;
    char payload[20];
     };
```

结构体 pkt 为一个包，其中的参数分别为序号，ACK，校验码和传递数据

如图为发送方和接受方

```
struct sender
{
    int state;  //描述发送方的状态
    int seq;  //发送包的序号
    struct pkt saved_packet;  // 将发送的包内容进行缓存，发生丢包情况进行重发
} A;

struct receiver
{
    int seq;   //序号
} B;
```

接下来实现如图的几个函数



应用层向运输层发送信息，运输层通过 A_Output()函数接收，将其放入到一个包 pkt 中，通过 A_Input()函数发送给下层进行处理，并最终发送到接收方，其中，为了处理丢包、误码等错误，根据 rdt3.0 协议设计相关代码，pkt 中的 checksum 用来判断是否发生误码，A_timerinterrupt()函数用来判断是否超时。

信息通过下层传递到接收方运输层，运输层需要对接收的信息进行甄别，一旦发现误码或序号不同等现象，向发送方传回包，传回 NAK 包，这时发送方需要重传；如果成功接收，传回 ACK 包,表明接收成功，进而向上层提供服务

3. 网络仿真

```
PS D:\Program\network\rdt> .\Stop-and-Wait.exe > a.txt
```

无 loos，error 情况

```
nsimmax=100;
lossprob=0.0;
corruptprob=0.0;
lambda=10.0;
TRACE=1;
```

```
:Enter TRACE: Simulator terminated at time 972.063416
 after sending 100 msgs from layer5
```

100*10=1000，与 972 十分接近

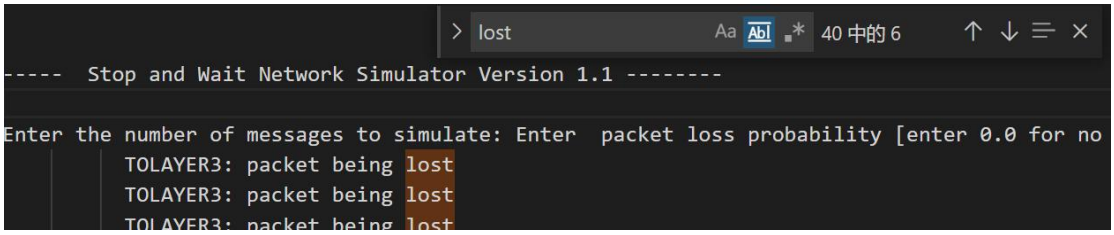
将 trace 改为 2，运行结果如下

```
EVENT time: 44.434948,  type: 1, fromlayer5  entity: 0

EVENT time: 49.011383,  type: 2, fromlayer3  entity: 1
        TOLAYER5: data received: ddddddddddddddddddd

EVENT time: 57.040649,  type: 2, fromlayer3  entity: 0

EVENT time: 61.685841,  type: 1, fromlayer5  entity: 0

EVENT time: 62.411144,  type: 2, fromlayer3  entity: 1

EVENT time: 68.347420,  type: 1, fromlayer5  entity: 0

EVENT time: 68.495773,  type: 2, fromlayer3  entity: 0
```

有 loos，无 error 情况

参数如下

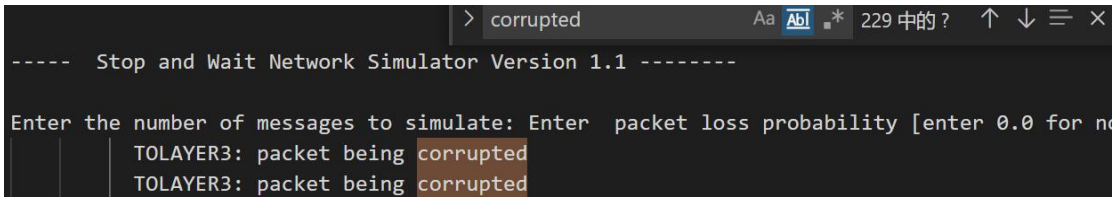
```
nsimmax=200;
lossprob=0.2;
corruptprob=0.0;
lambda=10.0;
TRACE=1;
```



200*0.2=40

无 loos，有 error 情况

```
nsimmax=200;
lossprob=0.0;
corruptprob=0.4;
lambda=10.0;
TRACE=1;
```



200*2*（0.4+0.4*0.4）=224

说明实验结果符合预期。

五、Go-Back-N 协议

1. 实验原理

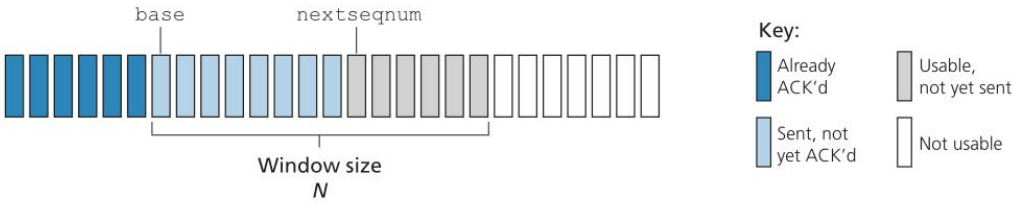


Figure 3.19 ♦ Sender's view of sequence numbers in Go-Back-N

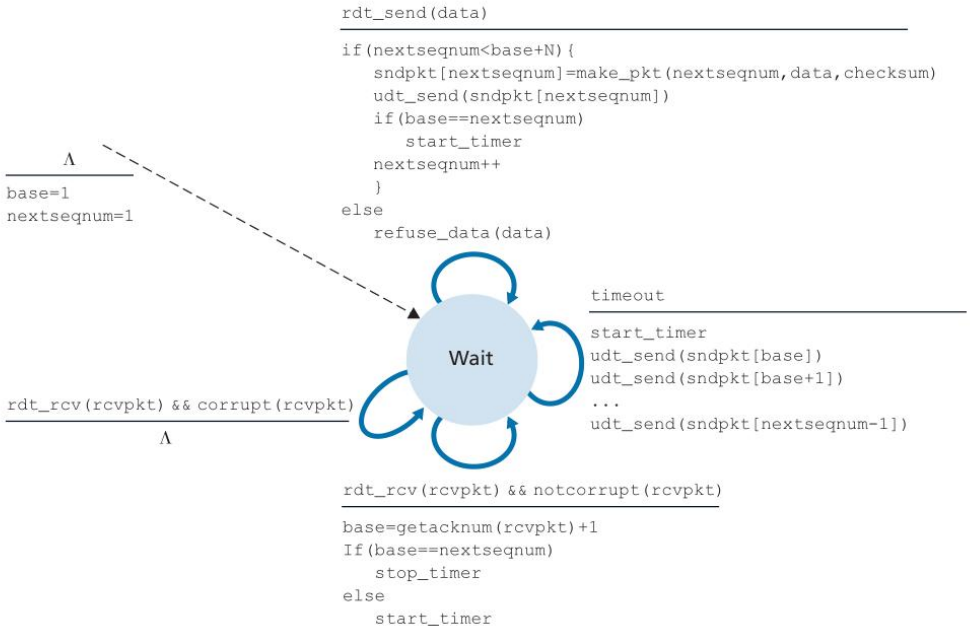


Figure 3.20 ♦ Extended FSM description of the GBN sender

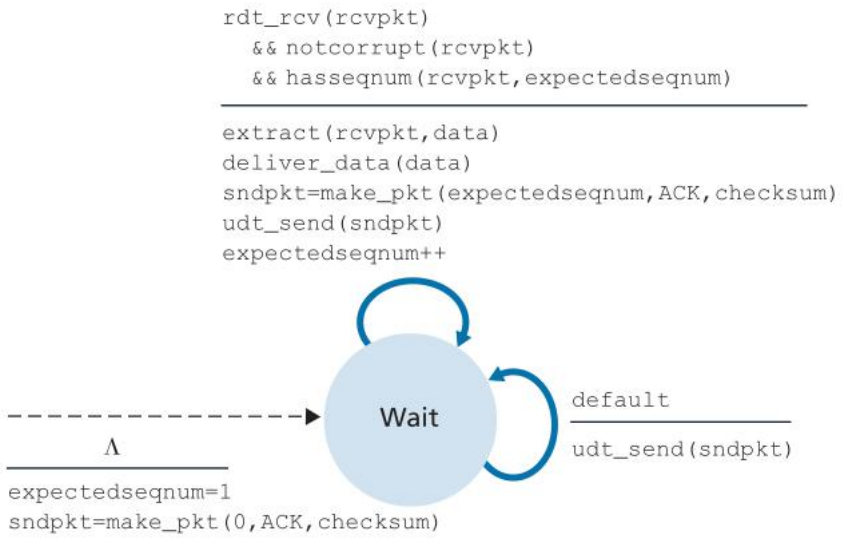


Figure 3.21 ♦ Extended FSM description of the GBN receiver

2. 算法实现

1）. 无差错情况流程

发送方将序号落在发送窗口内的 0~4 号数据分组，依次连续发送出去

他们经过互联网传输正确到达接收方，就是没有乱序和误码，接收方按序接收它们，每接收一个，接收窗口就向前滑动一个位置，并给发送方发送针对所接收分组的确认分组，在通过互联网的传输正确到达了发送方

发送方每接收一个、发送窗口就向前滑动一个位置，这样就有新的序号落入发送窗口，发送方可以将收到确认的数据分组从缓存中删除了，而接收方可以择机将已接收的数据分组交付上层处理

回退 n 帧协议采用累计确认

优点:

• 即使确认分组丢失，发送方也可能不必重传

• 减小接收方的开销

• 减小对网络资源的占用

缺点：

• 不能向发送方及时反映出接收方已经正确接收的数据分组信息

2）.有差错情况

例如

在传输数据分组时，5 号数据分组出现误码，接收方通过数据分组中的检错码发现了错误

于是丢弃该分组，而后续到达的这剩下四个分组与接收窗口的序号不匹配

接收同样也不能接收它们，讲它们丢弃，并对之前按序接收的最后一个数据分组进行确认，发送 ACK4，每丢弃一个数据分组，就发送一个 ACK4

当收到重复的 ACK4 时，就知道之前所发送的数据分组出现了差错，于是可以不等超时计时器超时就立刻开始重传，具体收到几个重复确认就立刻重传，根据具体实现决定

3. 网络仿真

```
PS D:\Program\network\rdt> .\Go-Back-N.exe > a.txt
```

无 loos，error 情况

```
nsimmax=100;
lossprob=0.0;
corruptprob=0.0;
lambda=10.0;
TRACE=1;
```

```
Simulator terminated at time 973.391968
after sending 100 msgs from layer5
```

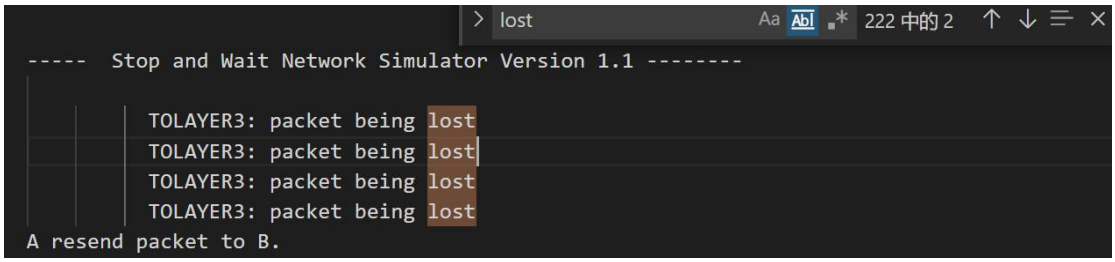
100*10=1000 ≈ 973

将 trace 改为 2，运行结果如下(部分截图)

```
EVENT time: 206.690613,  type: 2, fromlayer3  entity: 1

EVENT time: 209.221054,  type: 2, fromlayer3  entity: 1

EVENT time: 209.659256,  type: 0, timerinterrupt   entity: 0
```

有 loos，无 error 情况

```
nsimmax=200;
lossprob=0.2;
corruptprob=0.0;
lambda=10.0;
TRACE=1;
```

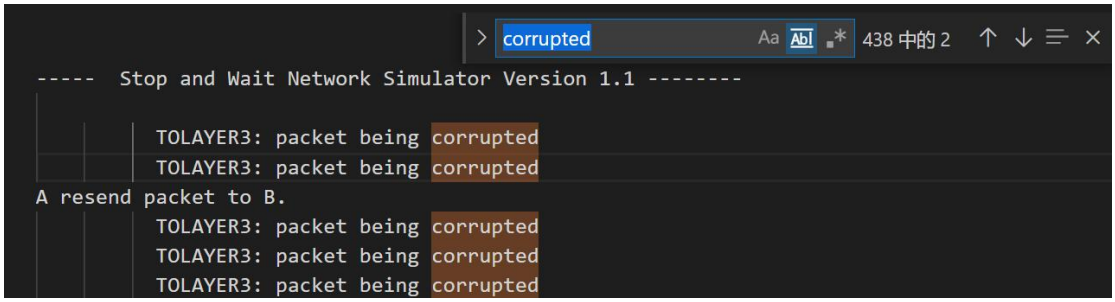


Lost 由 40 个变成 222 个

这是反映了停等协议的缺点：不能正确反映已经接收分组的信息。在 GBN 协议中，接收方丢弃所有失序分组，这样做的优点是接收缓存简单，即接收方不需要缓存任何失序分组。丢弃正确分组的缺点就是随后对该分组的重传也许会丢失或出错，因此甚至会需要更多的重传

无 loos，有 error 情况

```
nsimmax=200;
lossprob=0.0;
corruptprob=0.4;
lambda=10.0;
TRACE=1;
```



由于上述所提到的 Go-Back-N 协议的缺点，corrupted 大大增加。

六、实验心得体会

通过本次实验我熟悉并掌握了不同版本的 rdt 协议的特点，对于停止等待协议和回退 n 帧协议了解更为透彻，同时，本次实验也让我从实践意义上理解了网络协议，不再如同书本那样抽象。

当然，理解了 rdt 的思想之后，将其转化为代码又是一个巨大的难点。在编写代码的过程中特别难受的就是调试 bug，实验仿真结果与预期不符时不断调试代码，最终攻克难关。受益匪浅。