可靠传输协议实验报告

停等协议及回退 N 协议

2021 年 6 月 3 日

学院: 智能与计算学部

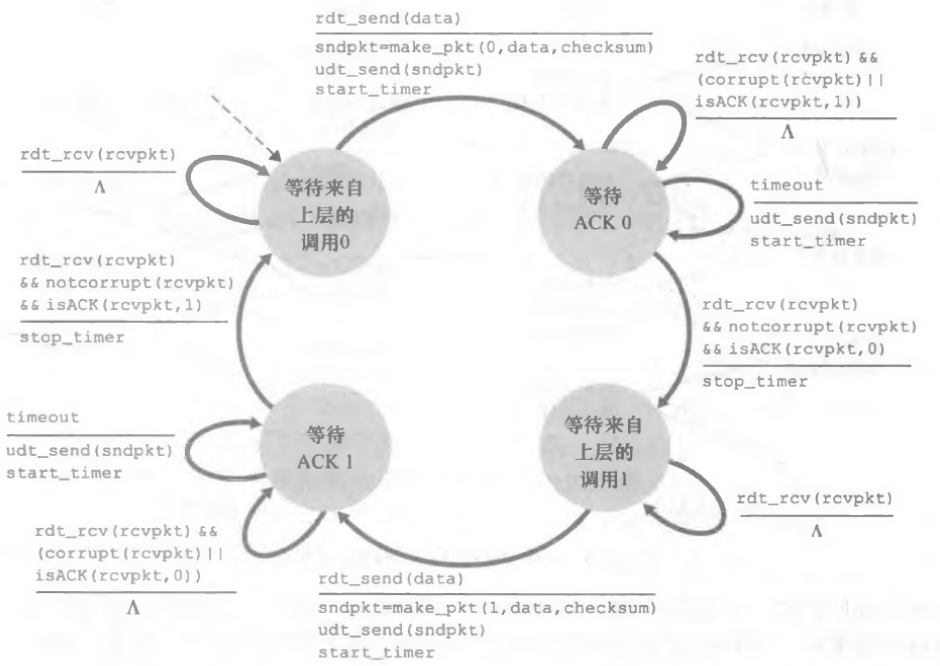
班级: 网络安全一班

姓名: 谢远峰

学号: 3019244283

1 比特交替协议

1 原理



停止等待协议是 tcp 保证传输可靠的重要途径,“停止等待”就是指发送完一个分组就停止发送, 等待对方的确认, 只有对方确认过, 才发送下一个分组。在停止-等待协议中，源站发送单个帧后必须等待确认，在目的站的回答到达源站之前，源站不能发送其他的数据帧。从滑动窗口机制的角度来看，停止-等待协议相当于发送窗口和接收窗口大小均为 1 的滑动窗口协议。

2 过程

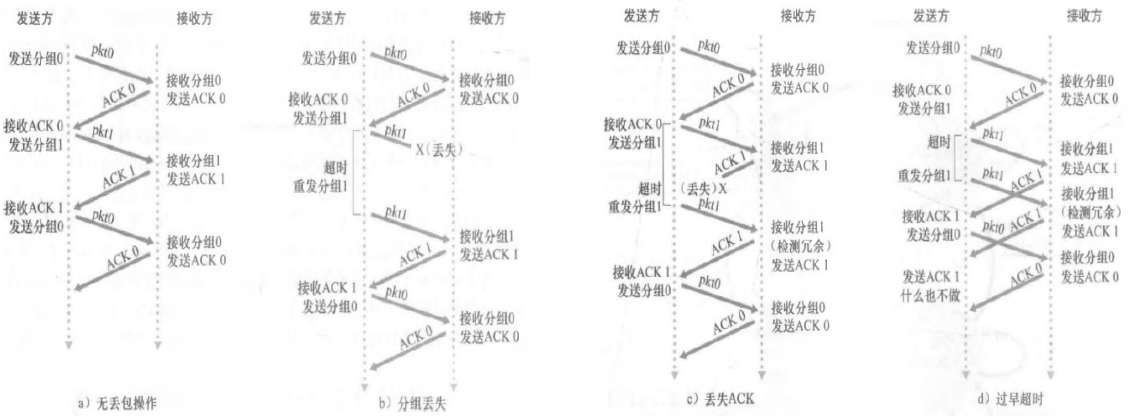


图 1.1: 情况列举

1. 无差错情况
   发送方发送分组, 接收方在规定时间内收到, 并且回复确认. 发送方再次发送
2. 分组丢失
   发送方发送分组, 接收方没有收到分组, 那么接收方不会发出确认, 只要发送方过一段时间没有收到确认, 就认为刚才的分组丢了, 那么发送方就会再次发送.

3. 确认丢失
   发送方发送成功, 接收方接收成功, 确认分组也被发送, 但是分组丢失, 那么到了等待时间, 发送方没有收到确认, 又会发送分组过去, 此时接收方前面已经收到了分组, 那么此时接收方要做的事就是: 丢弃分组, 重新发送确认.
4. 传送丢失
   发送方发送成功, 接收方接收成功, 确认分组也被发送, 没有丢失, 但是由于传输太慢, 等到了发送方设置的时间, 发送方又会重新发送分组, 此时接收方要做的事情: 丢弃分组, 重新发送确认. 发送方如果收到两个或者多个确认, 就停止发送, 丢弃其他确认.

3 算法实现规划

- 明确发送包的发送顺序: A 发送端接收应用层消息，进行打包转到传输层，B 接受端接收发送的包并进行校验，如果匹配则进行拆解等到消息上传至应用层，否则进行消息回传，通知 A 发送方进行包的重传
- A 发送端进行初始化: 确立发送方状态 (等待应用层消息/等待接受端的反馈)，发送包的序列号（0/1），计时器，打包的备份
- B 接收端进行初始化: 确定待接收的序列号
- 检验和的计算: 无论是发送方还是接收方，都需要对包进行校验和的计算：发送方计算序列号、消息的校验和（ACK 为默认为 0）；接收方需要对消息进行校验
- NAK/ACK 的发送: 若通过校验，赋值 ACK，否则，赋值最近一次成功的 ACK，重新计算校验和并发送
- 超时处理: 计时超限，发送备份包，并重新计时
- 异常处理: 发送方接收应用层消息前，需判断当前状态是否为等待消息；发送方接收 ACK 时需判断是否处于等待 ACK 状态；接收序列号不匹配时，需进行重传处理
- 双向传输: 在包中添加发送方状态信息，并在初始化阶段，分别建立接收端和发送端，在收发函数中，根据包内含有的发送方的信息进行判定。

4 实现具体细节

单向传输

- 初始化: A 发送端默认初始状态为等待消息输入，发送的序列号为 0，计时为 20；B 接收端默认待接收的序列号为 0
- A 发送方发送: 判定发送方状态，初始化消息报，装载序列号，拷贝消息进入包中，默认 ACK 为 0，计算校验和，标记发送方，备份消息包，设置期待接收的 ACK，发送包至传输端，开始计时。
- B 接收方接收: 判定包传输途中消息是否出错，判定包序列号是否正确，若未通过发送 NAK，否则发送 ACK，更新待接收的序列号
- A 发送方接收 ACK: 判定发送方状态，校验和判定，序列号正确性判定，停止计时，更新待发送的序列号，更新发送方的状态

5 问题复现

- 函数格式的更改: 由于底层函数的声明不同于大众化写法，故在文件开始进行正确声明
- 消息发送语句的次序问题: 先赋值后计算校验和，而后发送，计时，最后转换状态，顺序错乱可能导致消息错误率过高，超时频率增强
- 双向传输的书写: 在发送包中添加信息，需要在底层同样增加信息的声明初始化，同时 A，B 都应同时设置收发两种状态，并在函数中进行发送方的确认
- 函数传参的正确书写: 双向传输中，0，1 参数的错误需借助函数内部自定义的输出锁定错误位置

6 仿真结果

```
Enter the number of messages to simulate: 5
Enter  packet loss probability [enter 0.0 for no loss]:0.0
Enter packet corruption probability [0.0 for no corruption]:0.0
Enter average time between messages from sender's layer5 [ > 0.0]:10
Enter TRACE:2

EVENT time: 0.935697,  type: 1, fromlayer5  entity: 1
 B_output: send packet: aaaaaaaaaaaaaaaaaaa

EVENT time: 5.026246,  type: 2, fromlayer3  entity: 0
(B_sender)A_input: get the message in the packet aaaaaaaaaaaaaaaaaaa
(B_sender)A_input: A receiver send ACK.

EVENT time: 9.240669,  type: 2, fromlayer3  entity: 1
(B_sender)B_input ack is successfully received.

EVENT time: 13.373211,  type: 1, fromlayer5  entity: 0
 A_output: send packet: bbbbbbbbbbbbbbbbbbb

EVENT time: 14.264961,  type: 1, fromlayer5  entity: 0
 A_output: wrong state

EVENT time: 18.809900,  type: 2, fromlayer3  entity: 1
(A_sender)B_input: get the message in the packet bbbbbbbbbbbbbbbbbbb
(A_sender)B_input: B receiver send ACK.

EVENT time: 23.765648,  type: 2, fromlayer3  entity: 0
 (A_sender)A_input ack is successfully received.
```

图 1.2: 实现理想状态下的消息传输

以单向传输为例，A 为发送方，B 为接收方，设定发送消息为 3，丢包率和错包率为 0，消息的发送间隔为 1000 个时间单位，进行过程消息追踪，消息能够成功进行传输

7 算法功能/性能测试: 双向传输展示

```
Enter the number of messages to simulate: 10
Enter  packet loss probability [enter 0.0 for no loss]:0.0
Enter packet corruption probability [0.0 for no corruption]:0.0
Enter average time between messages from sender's layer5 [ > 0.0]:1000
Enter TRACE:2

EVENT time: 93.569748,  type: 1, fromlayer5  entity: 1
 B_output: send packet: aaaaaaaaaaaaaaaaaaa

EVENT time: 97.660294,  type: 2, fromlayer3  entity: 0
(B_sender)A_input: get the message in the packet aaaaaaaaaaaaaaaaaaa
(B_sender)A_input: A receiver send ACK.

EVENT time: 101.874718,  type: 2, fromlayer3  entity: 1
(B_sender)B_input ack is successfully received.

EVENT time: 1337.321045,  type: 1, fromlayer5  entity: 0
 A_output: send packet: bbbbbbbbbbbbbbbbbbb

EVENT time: 1342.757690,  type: 2, fromlayer3  entity: 1
(A_sender)B_input: get the message in the packet bbbbbbbbbbbbbbbbbbb
(A_sender)B_input: B receiver send ACK.

EVENT time: 1350.997314,  type: 2, fromlayer3  entity: 0
 (A_sender)A_input ack is successfully received.

EVENT time: 1426.496094,  type: 1, fromlayer5  entity: 0
 A_output: send packet: cccccccccccccccccc

EVENT time: 1435.072266,  type: 2, fromlayer3  entity: 1
(A_sender)B_input: get the message in the packet cccccccccccccccccc
(A_sender)B_input: B receiver send ACK.

EVENT time: 1439.648682,  type: 2, fromlayer3  entity: 0
 (A_sender)A_input ack is successfully received.

EVENT time: 2305.551270,  type: 1, fromlayer5  entity: 1
 B_output: send packet: ddddddddddddddddddd

EVENT time: 2309.649170,  type: 2, fromlayer3  entity: 0
(B_sender)A_input: get the message in the packet ddddddddddddddddddd
(B_sender)A_input: A receiver send ACK.

EVENT time: 2313.646973,  type: 2, fromlayer3  entity: 1
(B_sender)B_input ack is successfully received.
```

图 1.3: 无 loss，error

双向传输，A 作为发送方发送 b,c 两个串，B 作为发送方发送 a,d 两个串, 正常运行

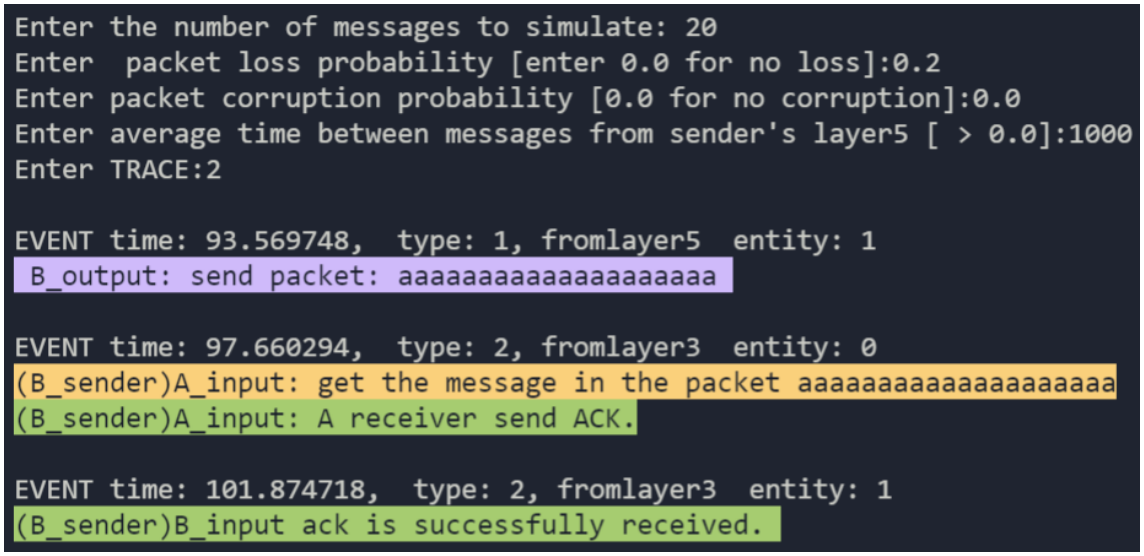


图 1.4: 数据包、ACK/NAK 包有 loss

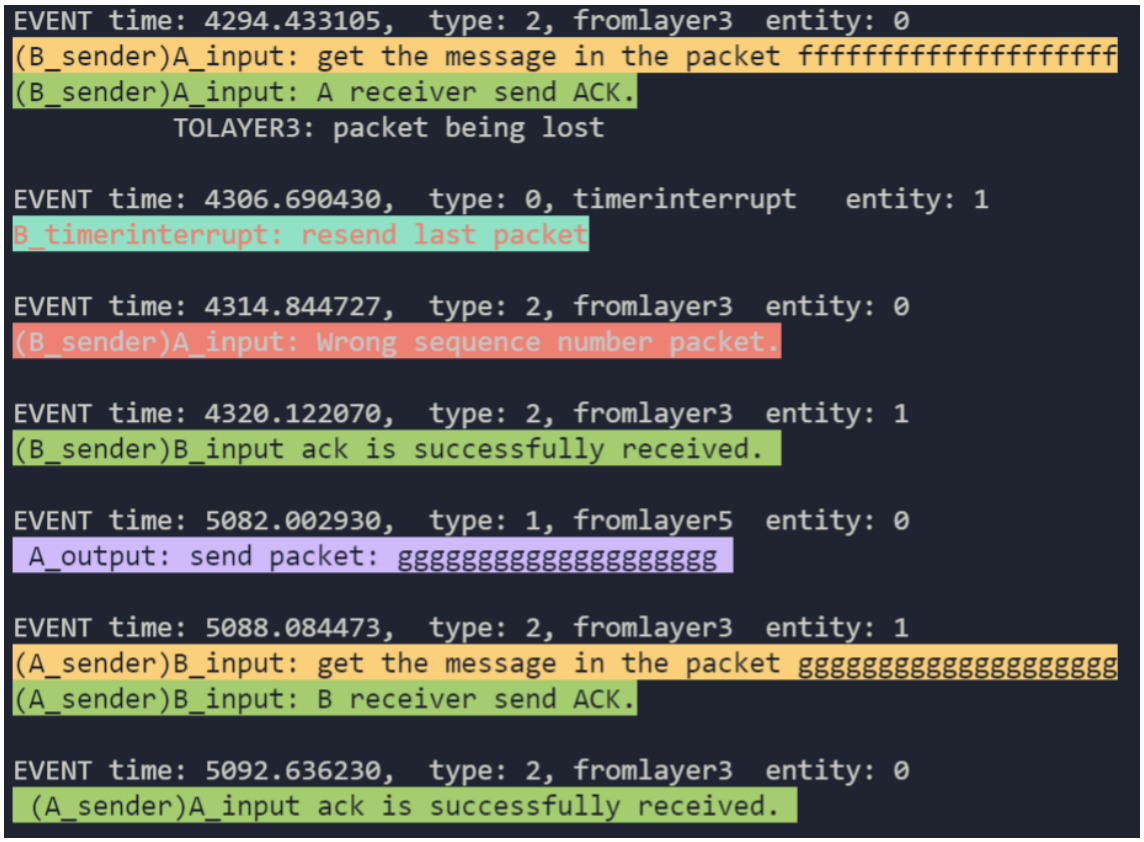


图 1.5: 数据包、ACK/NAK 包有 loss

数据包、ACK/NAK 包有丢失状态，测试采用 20 个样例，包的丢失率为 20%, 无包的错误率, 应用层消息的发送间隔为 1000 个时间单位, 进行信息的追踪。B 发送 F 串时, 返回的 ACK 丢失，造成 B 的计时器超时重传，A 接收方预期收到的序列号不匹配，发送最新匹配成功的 ACK 包，B 发送方成功进行接收。A 发送 g 串，B 成功接收并返回 ACK，A 成功接收。

```
Enter the number of messages to simulate: 20
Enter  packet loss probability [enter 0.0 for no loss]:0.0
Enter packet corruption probability [0.0 for no corruption]:0.3
Enter average time between messages from sender's layer5 [ > 0.0]:1000
Enter TRACE:2

EVENT time: 93.569748,  type: 1, fromlayer5  entity: 1
 B_output: send packet: aaaaaaaaaaaaaaaaaaa
          TOLAYER3: packet being corrupted

EVENT time: 97.660294,  type: 2, fromlayer3  entity: 0
(B_sender)A_input: Wrong checksum.
          TOLAYER3: packet being corrupted

EVENT time: 103.749313,  type: 2, fromlayer3  entity: 1
(B_sender)B_input not the expected ack.

EVENT time: 113.569748,  type: 0, timerinterrupt   entity: 1
B_timerinterrupt: resend last packet

EVENT time: 119.006439,  type: 2, fromlayer3  entity: 0
(B_sender)A_input: get the message in the packet aaaaaaaaaaaaaaaaaaa
(B_sender)A_input: A receiver send ACK.
          TOLAYER3: packet being corrupted

EVENT time: 127.246101,  type: 2, fromlayer3  entity: 1
(B_sender)B_input packet corrupted.

EVENT time: 133.569748,  type: 0, timerinterrupt   entity: 1
B_timerinterrupt: resend last packet

EVENT time: 136.454788,  type: 2, fromlayer3  entity: 0
(B_sender)A_input: Wrong sequence number packet.

EVENT time: 143.341171,  type: 2, fromlayer3  entity: 1
(B_sender)B_input ack is successfully received.
```

图 1.6: 数据包、ACK/NAK 包有 error

数据包、ACK/NAK 包有错误状态，测试采用 20 个样例，仅截取部分进行说明，包的错误率为 30%, 无包的丢失率, 应用层消息的发送间隔为 1000 个时间单位, 进行信息的追踪。B 发送 a 串时, 包发送比特错误，A 接收方预期收到的序列号不匹配，B 计时器超时重新发送，A 接收方成功接收并返回 ACK。

2 回退 N 协议

1 原理

图 2.1: 情况列举

GBN 是属于传输层的协议, 它负责接收应用层传来的数据, 将应用层的数据报发送到目标 IP 和端口

滑动窗口: 假设在序号空间内，划分一个长度为 N 的子区间，这个区间内包含了已经被发送但未收到确认的分组的序号以及可以被立即发送的分组的序号，这个区间的长度就被称为窗口长度。（随着发送方对 ACK 的接收，窗口不断的向前移动，并且窗口的大小是可变的）

接收方只需要按序接收分组，对于比当前分组序号还要大的分组则直接丢弃。假设接收方正在等待接收分组 n，而分组 n ＋ 1 却已经到达了，于是，分组 n ＋ 1 被直接丢弃，所以发送方并不会出现在连续发送分组 n，分组 n ＋ 1 之后，而分组 n ＋ 1 的 ACK 却比分组 n 的 ACK 更早到达发送方的情况。

2 过程

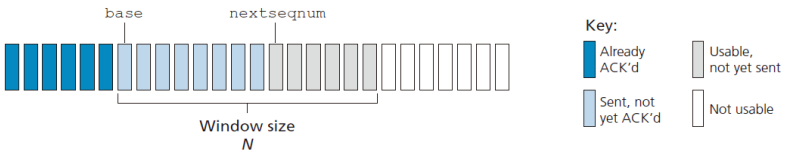


图 2.2: 消息包的分类

GBN 协议的事件判断:

1. $[0, send\_base - 1]$: 已发送并且收到确认了的序列号
2. $[send\_base, nextseqnum - 1]$: 已发送但未收到确认了的序列号
3. $[nextseqnum, send\_base + N - 1]$: 可被用于发送新分组的序列号
4. 大于等于 $[send\_base + N]$: 不能使用的序列号

协议中，已发送但未被确认的分组数目不能超过 N。因此 N 可以看做是从第一个已发送但未被确认的序号（就是基序号）开始的长为 N 的序号窗口，也称为发送窗口。随着分组被确认，该窗口逐渐滑动，$send\_base$ 增加，但是其大小不变，因此该协议也称为滑动窗口协议。假设序号空间总大小为 X，所有的序号计算都要对 X 取模。

对于超时的触发，GBN 协议会将当前所有已发送但未被确认的分组重传，即如果当前窗口内都是已发送但未被确认的分组，一旦定时器发现窗口内的第一个分组超时，则窗口内所有分组都要被重传。每次当发送方收到一个 ACK 的时候，定时器都会被重置。

3 算法实现规划

发送方需要处理事件

1. 上层调用进行发送：当发送数据时，首先判断发送窗口是否已满，只有不满时才会启动发送，如果满了则不能发送或者缓存或者通知调用者，这取决于实现。(判满)
2. 收到 ACK：如果收到了分组 n 的 ACK，并且分组 n 的序号在 $[send\_base, nextseqnum-1]$ 之间，则更新 send_base。GBN 协议采用累积确认，含义是如果发送方接收到了对分组 n 的确认，则表明分组 n 之前的所有分组都已经被接收。
3. 超时事件：GBN 协议名字来自该协议对分组丢失或超时事件的处理。如果分组丢失或者指定的时间内仍未收到对已发送但是未收到确认的分组的确认，则发送方重传 $[send\_base, nextseqnum-1]$ 之间的所有分组。实现中可以只为序号为 send_base 的分组启动一个定时器，如果 send_base 被更新就重启 send_base 相关的定时器，如果没有已经被发送但未被确认的分组，则关闭定时器。

接收方需要处理事件

接收方接收到分组 n 时，如果 n 是按序接收的，即分组 n 之前的所有分组都已经到达，则发送一个对分组 n 的确认，并将分组提交给上层。所有其它情况，接收方都丢弃分组，并重传对最后一个按序接收的分组的确认。这种设计简化了接收方接收缓存的设计，同时被丢弃的分组早晚都会被重传，因而可靠性也是有保证的。GBN 协议的接收方只维护了一个期望收到的下一个序号的信息，并保存在 expectedseqnum 中，在 expectedseqnum 之前的所有分组都已经被正确接收并提交给了上层。接收方具体的行为如下：

1. 如果收到的分组的序号与 expectedseqnum 相同，则将分组提交给上层，并更新它的值为下一个期望收到的序号。
2. 如果收到的分组的序号大于 expectedseqnum，就丢弃分组。
3. 如果收到的分组的序号小于 expectedseqnum，就重传对最后一个按序接收的分组的确认。由于 GBN 是累积确认的，因此该确认可以确保发送窗口可以向前移动。送但未被确认的分组，则关闭定时器。

4 实现具体细节

- 发送方信息确定: 先赋值后计算校验和，而后发送，计时，最后转换状态，顺序错乱可能导致消息错误率过高，超时频率增强
- 接收方信息确定: 在发送包中添加信息，需要在底层同样增加信息的声明初始化，同时 A，B 都应同时设置收发两种状态，并在函数中进行发送方的确认
- 宏定义: 定义窗口大小，序列号带线啊哦
- 发送方窗口发送: 进行缓冲区长度判定，初始化发送包，获取序列号，装载应用层消息，定义发送端信息，初始化 ACK 值，计算校验和，更新发送端缓冲区末端，发送窗口内部消息
- 接收方窗口接收: 进行校验和及序列号判定，解包获取消息上传至应用层，赋值 ACK，重新计算校验和，发送至网络层，更新下一次的预期序列号

5 遭遇问题

- ACK 及窗口基地址的更新: 将缓冲区内的标号模序列号进行运算, 其中包括接收方对于 ACK 的赋值和发送方对于窗口基地址的更新，新的窗口基地址等于原地址与获取到 ACK 到基地址的距离

6 仿真结果

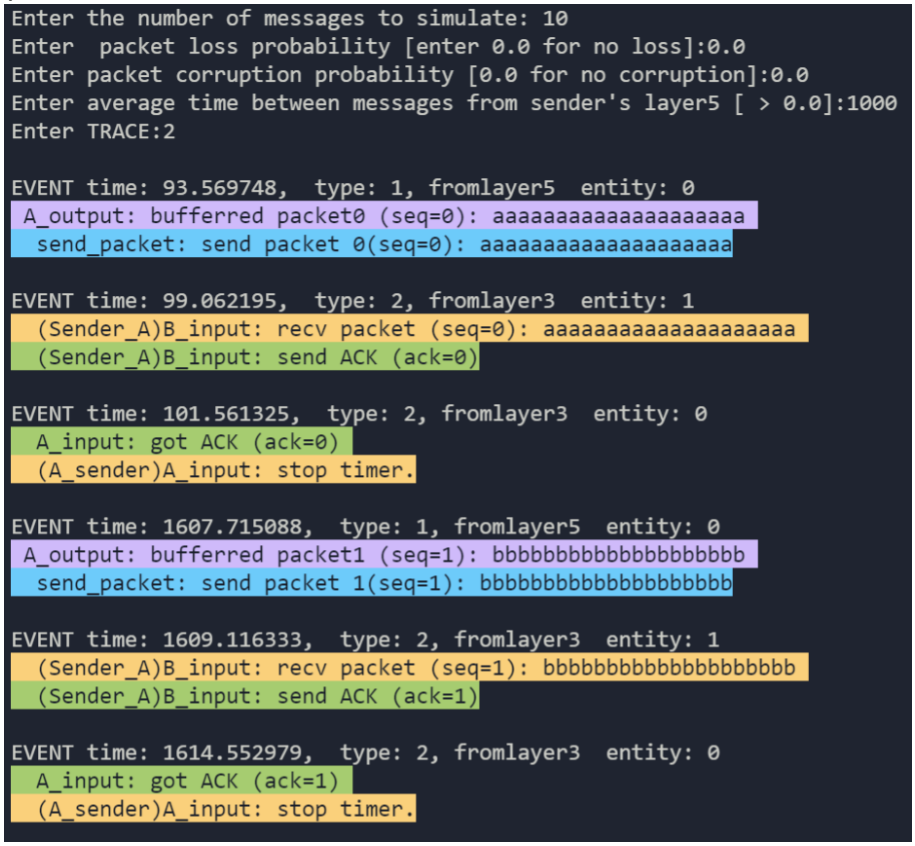


图 2.3: 消息包的分类

以单向传输为例，A 为发送方，B 为接收方，设定发送消息为 3，丢包率和错包率为 0，消息的发送间隔为 1000 个时间单位，进行过程消息追踪，消息能够成功进行传输

7 算法功能/性能测试

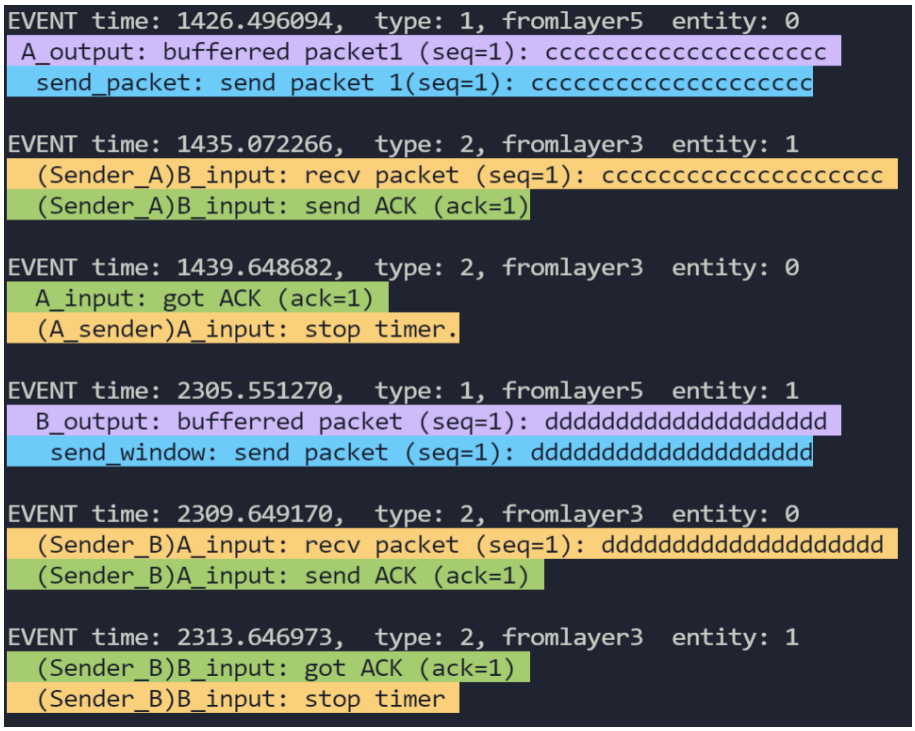


图 2.4: 无 loss,error

图 2.5: 有 loss, 无 error 测试 1

双向传输，b 作为发送方发送 a 串，首次传输的包发生丢失，B 收到 NAK 信息，B 重传，A 成功接收，返回 ACK 丢失，B 重传，A 返回 NAK，B 收到对应的 ACK 信息。

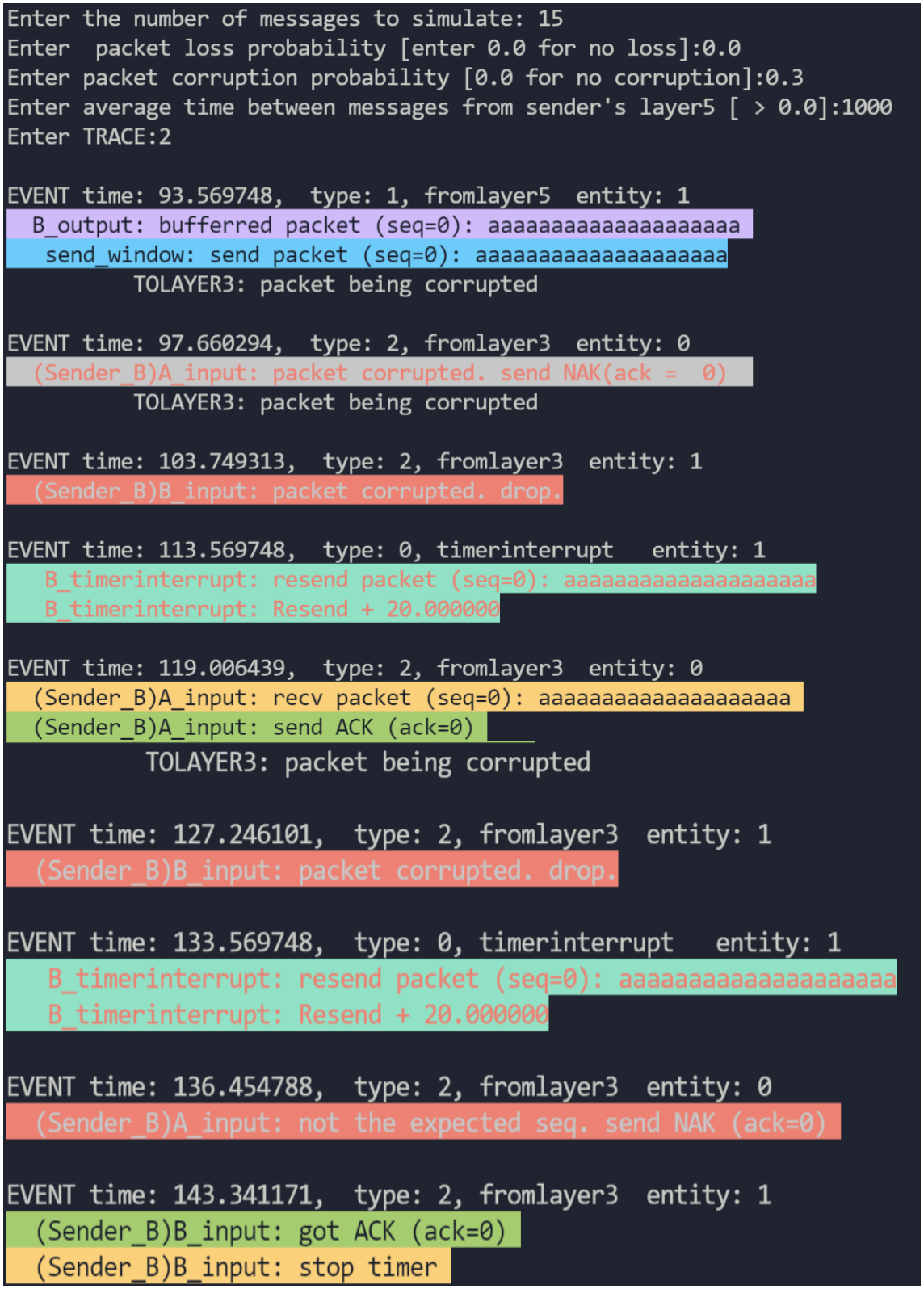


图 2.6: 无 loss，有 error

双向传输，b 作为发送方发送 a 串，首次传输的包发生错误，B 收到 NAK 信息，B 重传，A 成功接收，返回 ACK 错误，B 重传，A 返回 NAK，B 收到对应的 ACK 信息。

3 分析和总结

对于比特交换协议，由于其停等的特性，导致数据包在传输过程中发生丢失或错误时，消息的收发效率会大大降低。

对于回退 N 协议，因其具有流水线的机制，采用滑动窗口的方法确保在短时间内同时发送多个数据包，在一定程度上提高了传输效率。

对于输出消息的颜色设置能够将传输过程更加清晰地呈现。由单向传输转换为双向传输时，需要对底层传输代码进行适当的发送发信息添加。