Report of Project4

Visualizing Sliding Window and Flow Control for TCP Via Python

Qiong Yang

1 Principles of Reliable Data Transfer

rt3.0 is good, but it's performance is very bad. The key of bad performance is stop-and-wait protocol. To solve this problem, people think out of a simple solution: givie up the stop-and-wait protocol and admit sending the multiple packages at the same time without waiting ack. This technique is called pipelining. Pipelining has the following consequences for reliable data transfer protocols:

- The range of sequence numbers must be increased, since each in-transit packet (not counting retransmissions) must have a unique sequence number and there may be multiple, in-transit, unacknowledged packets.

- The sender and receiver sides of the protocols may have to buffer more than one packet. Minimally, the sender will have to buffer packets that have been transmitted but not yet acknowledged. Buffering of correctly received packets may also be needed at the receiver, as discussed below.

- The range of sequence numbers needed and the buffering requirements will depend on the manner in which a data transfer protocol responds to lost, corrupted, and overly delayed packets. Two basic approaches toward pipelined error recovery can be identified: Go-Back-N and selective repeat.

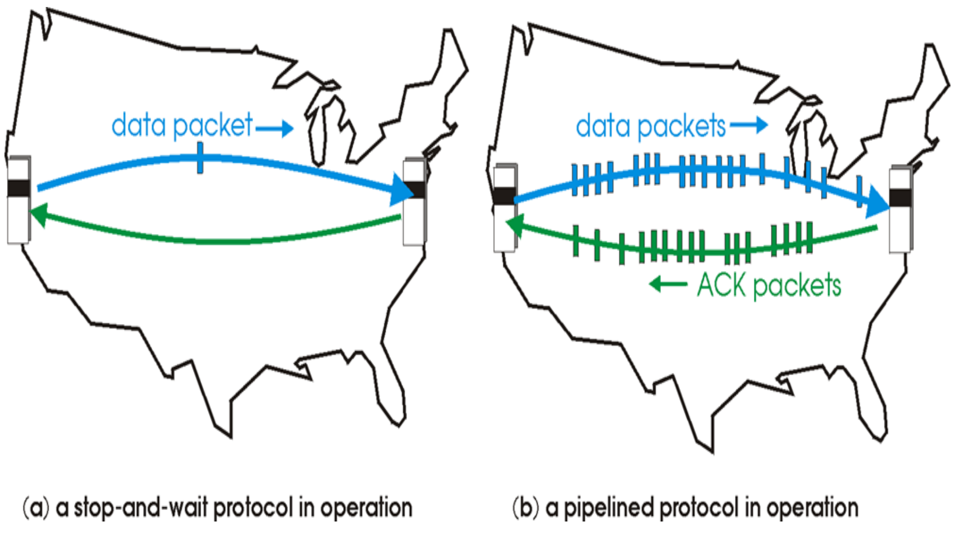
data packet
(a) a stop-and-wait protocol in operation
data packets
ACK packets
(b) a pipelined protocol in operation

Figure 1: Stop-and-wait and pipelined protocol

Next, we will analyse the principles of Go-Back-N and Selective repeat.

1.1 Go-Back-N

In a Go-Back-N (GBN) protocol, the sender is allowed to transmit multiple packets (when available) without waiting for an acknowledgment, but is constrained to have no more than some maximum allowable number, N, of unacknowledged packets in the pipeline.
In GBN, we define textbfbase to be the sequence number of the oldest unacknowledged packet and textbfnextseqnum to be the smallest unused sequence number (that is, the sequence number of the next packet to be sent) We can devide the range of sequence numbers into four intervals:[0,base-1],[base,nextseqnum-1],[nextsequem,base+N-1] and [base+N,default],where

- interval [0,base-1]:correspond to packets that have already been transmitted and acknowledged
- interval [base,nextseqnum-1]:corresponds to packets that have been sent but not yet acknowledged
- interval [nextsequem,base+N-1]:be used for packets that can be sent immediately
- interval [base+N,default]:cannot be used until an unacknowledged packet currently in the pipeline (specifically, the packet with sequence number base) has been acknowledged.

Figure 2 shows the senders view of the range of sequence numbers in a GBN protocol.

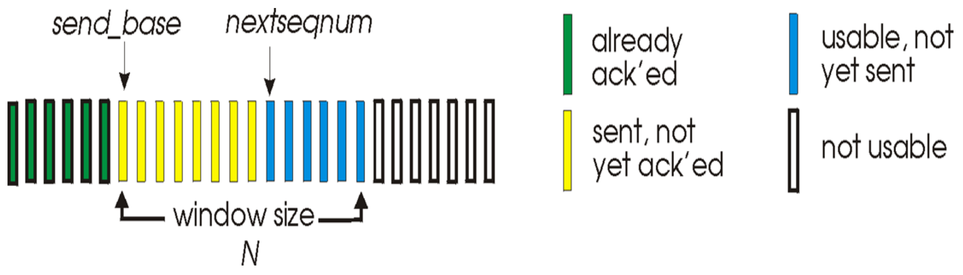

Figure 2: Sender's view of sequence numbers in Go-Back-N

The algorithm can be described as follows:

Algorithm 1 Go-Back-N Receiver Algorithm

```
re_num=0
while true do
  WaitForEvent()
  if Event(ArrivalNotification) then
    Receive(Frame)
    if corrupted(Frame) then
      Sleep()
    end if
    if seqNo ==re_num then
      DeliverData()
      re_num = re_num +1
      SendACK(re_num)
    end if
  end if
end while
```

1.2 Selective repeat

The GBN protocol allows the sender to potentially fill the pipeline with packets, thus avoiding the channel utilization problems. However, In particular, when the window size and bandwidth-delay product are both large, many packets can be in the pipeline. A single packet error can thus cause GBN to retransmit a large number of packets, many unnecessarily.As the probability of channel errors increases, the pipeline can become filled with these unnecessary retransmissions.
Selective-repeat protocols avoid unnecessary retransmissions by having the sender retransmit only those packets that it suspects were received in error

Algorithm 2 Go-Back-N Sender Algorithm

```
se_win=$2^m$-1
se_fl=0
se_num=0
while true do
    WaitForEvent()
    if Event(RequestToSend) then
        if se_num-se_fl $\geq$ se_win then
            Sleep()
        end if
        GetData()
        MakeFrame(se_num)
        StoreFrame(se_num)
        SendFrame(se_num)
        se_num = se_num+1
        if time not running then
            StartTimer()
        end if
    end if
    if Event(ArrivalNotfication) then
        Receive(ACK)
        if corrupted(ACK) then
            Sleep()
        end if
        while se_num $\geq$ackNo do
            purgeFrame(se_fl)
            se_fl=se_fl+1
        end while
        StopTimer()
    end if
    if Event(TimeOut) then
        StartTimer()
        Temp = se_fl
        while Temp ¡ se_num do
            SendFrame(se_fl)
            se_fl=se_fl+1
        end while
    end if
end while
```

(that is, were lost or corrupted) at the receiver. This individual, asneeded, retransmission will require that the receiver individually acknowledge correctly received packets. A window size of N will again be used to limit the number of outstanding, unacknowledged packets in the pipeline. However, unlike GBN, the sender will have already received ACKs for some of the packets in the window. Figure 3.23 shows the SR senders view of the sequence number space. Figure 3.24 details the various actions taken by the SR sender.

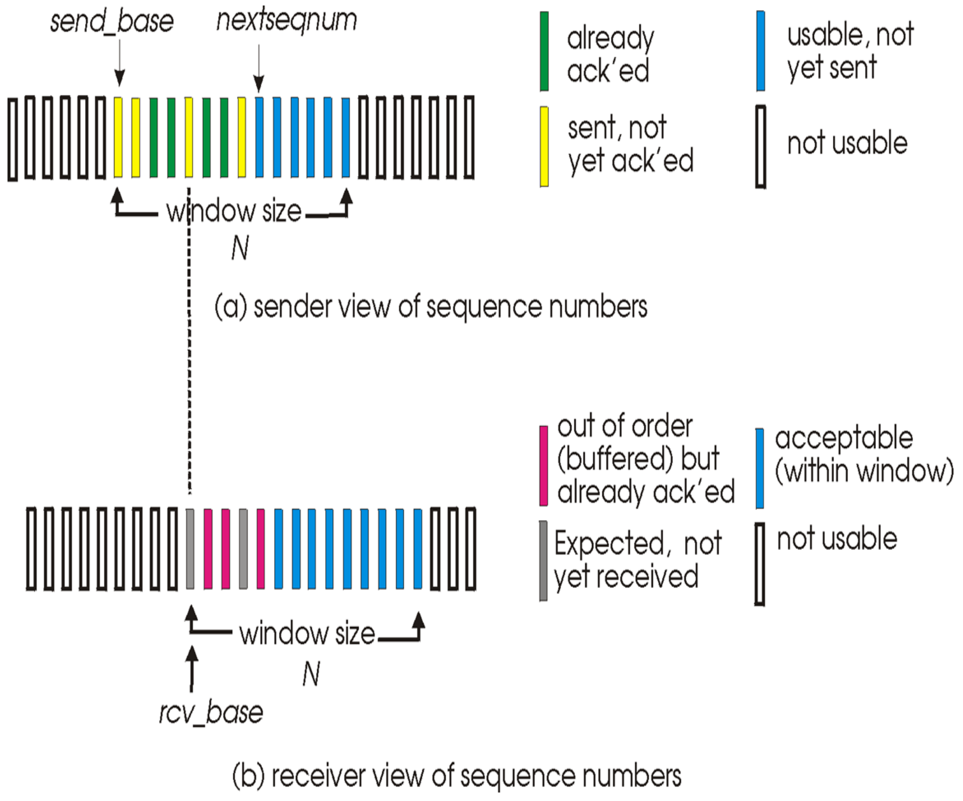
send_base
nextseqnum
already ack'ed
usable, not yet sent
sent, not yet ack'ed
not usable
window size N
(a) sender view of sequence numbers
out of order (buffered) but already ack'ed
acceptable (within window)
Expected, not yet received
not usable
window size N
rcv_base
(b) receiver view of sequence numbers

Figure 3: Selective-repeat (SR) sender and receiver views of sequence-number space

The algorithm can be described as follows:

Algorithm 3 Slective Repeat Receiver Algorithm

```
re_num=0
NakSent = false
AckNeeded = false
Repeat(for all slots)
Marked(slot)=false
while true do
   WaitForEvent()
   if Event(ArrivalNotification) then
      Receive(Frame)
      if corrupted(Frame) and (NOT NakSent) then
         SendNAK(re_num)
         NakSent = true
         Sleep()
      end if
      if seqNo ¡¿re_num and (NOT NakSent) then
         SendNAK(re_num)
         NakSent = true
         if (seqNo in window) and (!Marked(seqNo)) then
            StoreFrame(seqNo)
            Marked(seqNo)=true
            while Marked(re_num) do
               DeliverData(re_num)
               purge(re_num)
               re_num = re_num +1
               AckNeeded = true
            end while
            if AckNeeded then
               SendACK(re_num)
               AckNeeded = false
               NakSent = false
            end if
         end if
      end if
   end if
end while
```

Algorithm 4 Slective Repeat Sender Algorithm

```
se_win=$2^m$-1
se_fl=0
se_num=0
while true do
  WaitForEvent()
  if Event(RequestToSend) then
    if se_num-se_fl $\geq$ se_win then
      Sleep()
    end if
    GetData()
    MakeFrame(se_num)
    StoreFrame(se_num)
    SendFrame(se_num)
    se_num = se_num+1
    StartTimer(se_num)
  end if
  if Event(ArrivalNotfication) then
    Receive(frame)
    if corrupted(frame) then
      Sleep()
    end if
    if FrameType == NAK then
      if nakNo between se_fl and se_num  then
        Resend(nakNo)
        StartTimer(nakNo)
      end if
    end if
    if FrameType == ACK then
      if ackNo between se_fl and se_num  then
        while se_fl ¡ ackNo do
          purge(se_fl)
          StopTimer(StopTimer())
          se_fl=se_fl+1
        end while
      end if
    end if
    if Event(TimeOut(t)) then
      StartTimer(t)
      SendFrame(t)
    end if
  end if
end while
```