Chapter3 Data Link Layer(2)

王昊翔：hxwang@scut.edu.cn

School of Computer Science & Engineering ,SCUT

Communication & Computer Network key-Lab of GD

Contents of this lecture

- Learn 6 elementary DLL protocol
- Learn & master sliding-window
- Learn & master ARQ（自动重复请求）/PAR
- Learn & master piggybacking(捎带确认)
- Other: pipeline, NAK

Elementary DLL protocol

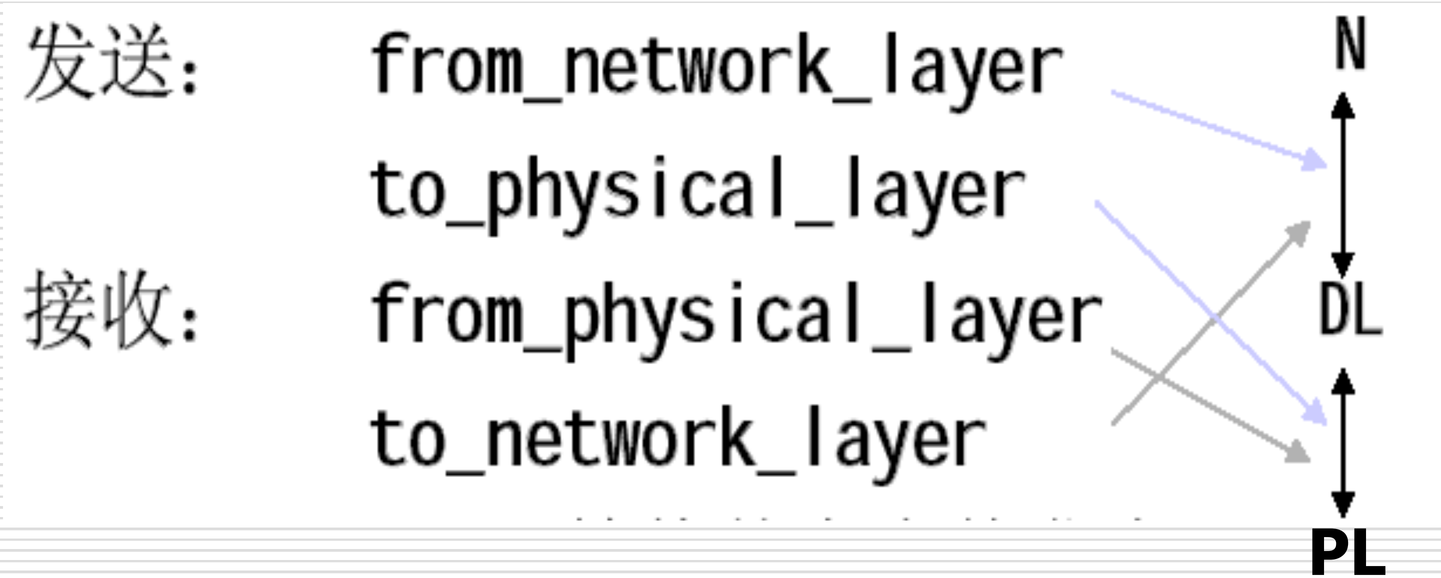
- Assumptions:
  - The physical layer, data link layer, and network layer are **independent processes**.
  - Machine A wants to send a long stream of data to machine B, using a **reliable, connection-oriented** service.
  - A's data link layer asks for data, the network layer is **always able** to comply immediately.
  - These protocols deal with communication **errors, but not the problems caused by computers** crashing and rebooting.
  - As far as the data link layer is concerned, the packet passed across the interface to it from the network layer is **pure data**.

Elementary DLL protocol（cont'd）

- 3 simplex(单工) protocol:
  - An unrestricted simplex protocol
    - 无限制的单工协议
  - A simplex stop-and-wait protocol
    - 单工的停—等协议
  - A simplex protocol for a noisy channel
    - 有噪声信道的单工协议

Protocol Declaration （1/2）

- All common data-type、functions are defined as （protocol.h）
- Data transmission between NL、PL



Protocol Declaration （2/2）

- **Wait_for_event (&event):** wait for something to happen
  - **fram_arrive,**
  - **cksum_err,**
  - **timeout**
- **Timer**
  - **start_timer, stop_timer**
  - **start_ack_timer，stop_ack_timer**

Frame structure

- typedef struct{

```
frame_kind kind;
seq_nr seq;
seq_nr ack;
packet info;
}frame;
```

Frame header

Unrestricted Simplex Protocol (protocol 1)

- Protocol 1 (utopia, 乌托邦)
- Data are transmitted in one direction only
- Both the transmitting and receiving network layers are always ready （随时待命）
- Processing time can be ignored （瞬间完成）
- Infinite buffer space is available （无限空间）
- The communication channel between the data link layer never damages or loses of frames（完美通道）

Unrestricted Simplex Protocol

- **Both the sender and the receiver are in an infinite while loop(无限的while循环)**
- **The sender:**

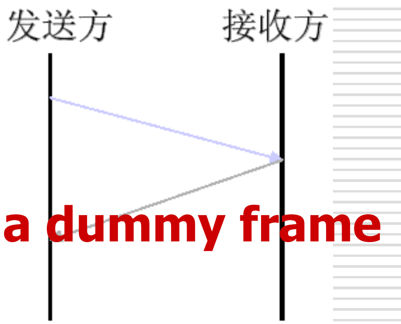
```
while (true) {
    from_network_layer(&buffer); /* go get something to send */
    s.info = buffer;             /* copy it into s for transmission */
    to_physical_layer(&s);       /* send it on its way */
}                                /* Tomorrow, and tomorrow, and tomorrow,
```

- **The receiver**

```
while (true) {
    wait_for_event(&event);      /* only possibility is frame_arrival */
    from_physical_layer(&r);     /* go get the inbound frame */
    to_network_layer(&r.info);   /* pass the data to the network layer */
}
```

Simplex Stop-and-Wait Protocol

- Protocol2 ： Drop the unrealistic restriction
  - Process abilities in receiver is limited: the receiver has only a finite buffer capacity and a finite processing speed.
- How to prevent the sender from flooding the receiver with data?
  - simply insert a delay into protocol 1
  - the receiver provide feedback to the sender



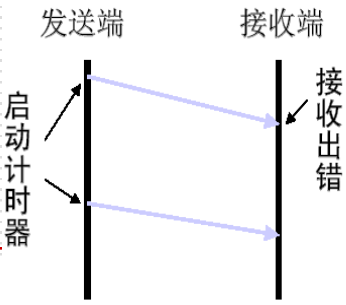
- Protocols in which the sender sends one frame and then waits for an acknowledgement before proceeding are called stop-and-wait

Pay attentions

- Data traffic is simplex（单工）, but frames do travel in both directions.
- The communication channel between the two data link layers needs to be capable of bidirectional information transfer.
  - A half- duplex（半双工） physical channel is enough
- Data flow
  - first the sender sends a frame, then the receiver sends a frame, then the sender sends another frame, then the receiver sends another one, and so on.

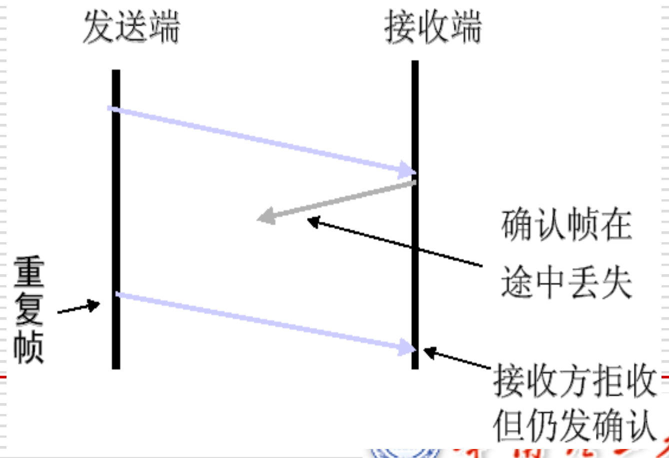
Simplex Protocol for a Noisy Channel

- Let us consider the normal situation: a communication channel may make errors.
- A scheme using timer and acknowledgement:
  - The sender starts a timer after sending a frame.
  - The receiver sends back an acknowledgement only when the incoming frame were correctly received.
  - The sender will retransmit the frame if the timer expired before
- Positive Acknowledgement with Retrasmission
- Automatic Repeat reQuest



PAR(主动确认重传)

- What is the fatal flaw in the above scheme ?
  - Loss of an acknowledgement
  - duplicated frame
- How to distinguish a frame being seen for the first time from a retransmission ?
  - Use a sequence number in the header of each frame.



Sliding window protocol

- How to improve efficiency of transmission?
  - full-duplex （全双工）
  - piggybacking （捎带确认）
  - sliding window （滑动窗口）（flow control）
- Sliding window protocol
  - Protocol 4：n=1——basic idea of sliding window
  - Protocol 5： Go Back n （回退n帧）
  - Protocol 6： Select Repeat （选择性重传）

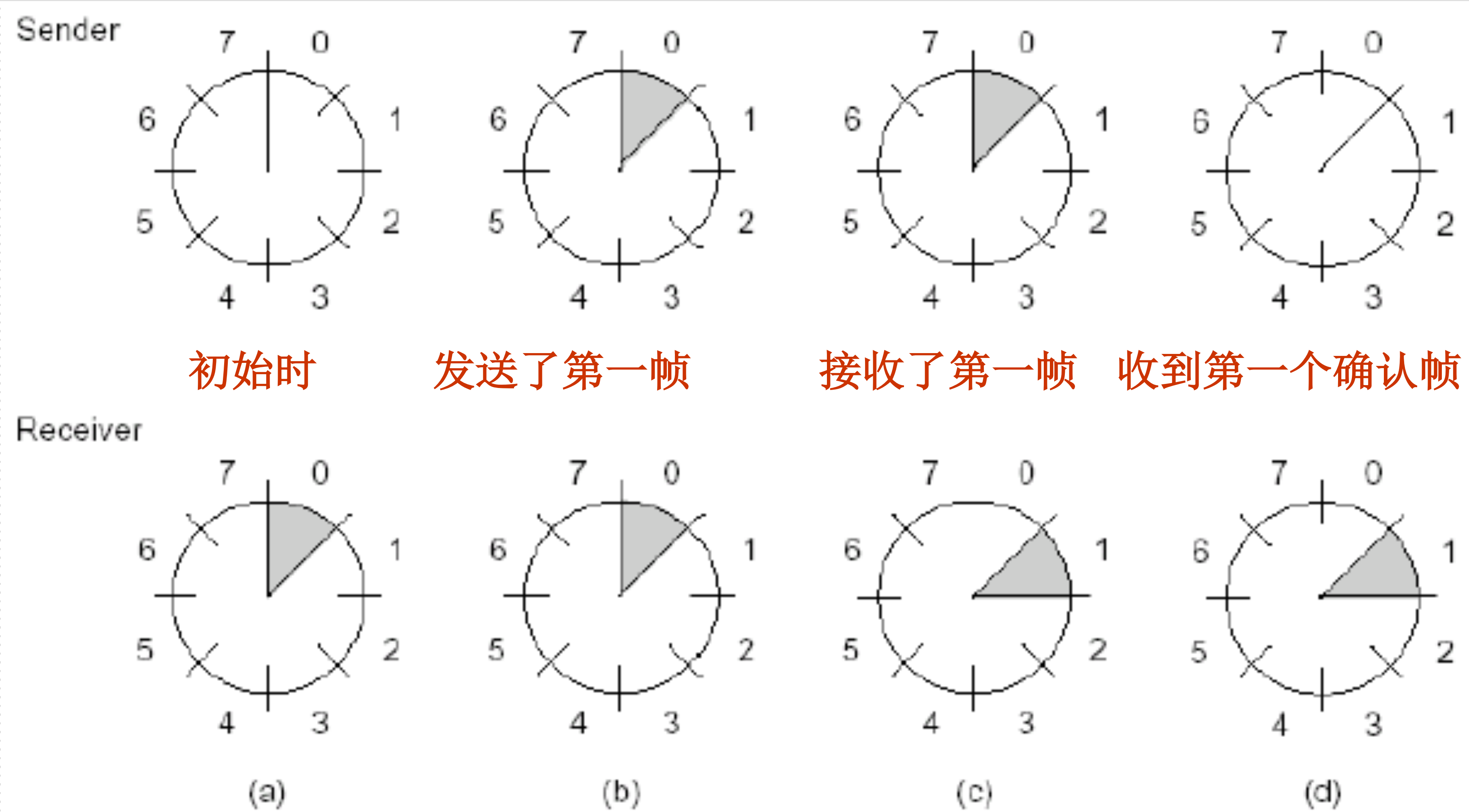
Sliding window protocol

- How to achieve full-duplex data transmission ?
  - Two separate simplex data channels
  - One circuit for data in both directions
- Piggybacking(捎带确认)
  - temporarily delay outgoing acknowledgements so that they can get a free ride on the next outgoing data frame
- How long should the receiver wait for piggybacking?
  - An ad hoc scheme: waiting a fixed number of milliseconds. （ACK-TIMER）

Sliding window protocol

- A sequence number is associated with each transmitted frame. Sequence numbers range from 0 up to some maximum ($2^n - 1$) circularly.
- A window is a list of sequence numbers.
- The sender maintains a sending window with sequence numbers corresponding to frames that have been sent but are as yet not acknowledged, or can be sent.
  - Whenever a new packet arrives from the network layer, it is given the next highest sequence number, and the upper edge is advanced by one.
  - When an acknowledgement comes in, the lower edge is advanced by one.
- The receiver maintains a receiving window with sequence numbers corresponding to frames it is permitted to accept.
  - When a frame whose sequence number is equal to the lower edge of the window is received, it is passed to the network layer, an acknowledgement is generated, and the window is rotated by one.

A sliding window of size 1, with a 3-bit sequence number



Explain

- Receiver: after receive frame， check if the sequence number is frame_expected(预期帧号) or not， if it is， receive it and frame_expected+1， that's sliding receiving-window（接收窗口）。
- Sender： When receive acknowledge frame， check if ack_number is next_frame_to_send or not， if it is, get packet from network_layer, and next_frame_to_send+1， that's sliding sending_window

w=1 sliding window

A的发送窗口

	A7
	A6
1	A5
0	A4
1	A3
0	A2
1	A1
0	A0
Seq	Data

窗口外的发送帧

待确认的发送帧

被确认的发送帧

B的接收窗口

Seq	Data
0	A0
1	A1
0	A2
1	A3
0	A4
1	A5
	A6

提交网络层的正确接收帧

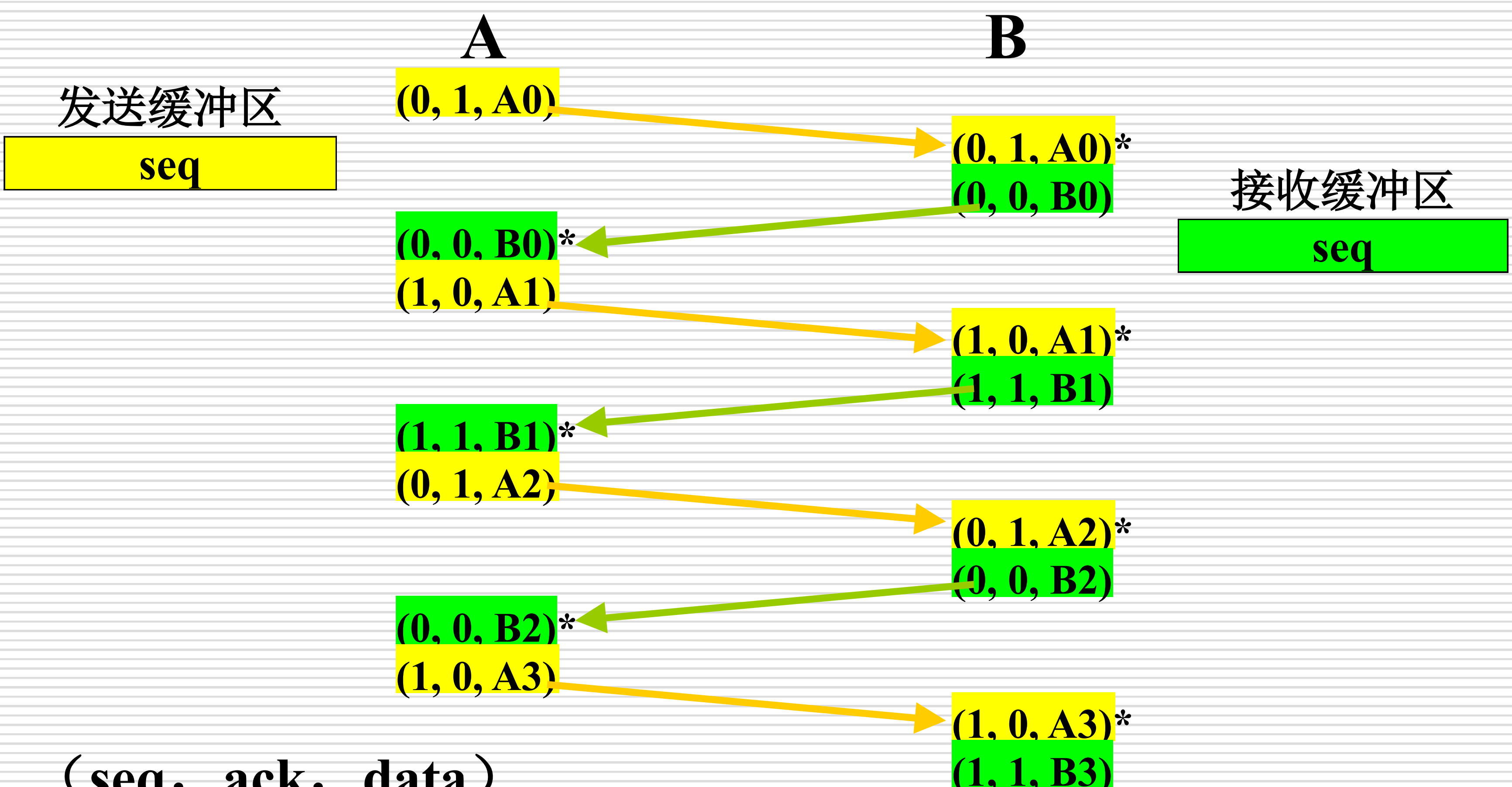
待提交的接收帧

窗口外的接收帧

Principle of protocol 4

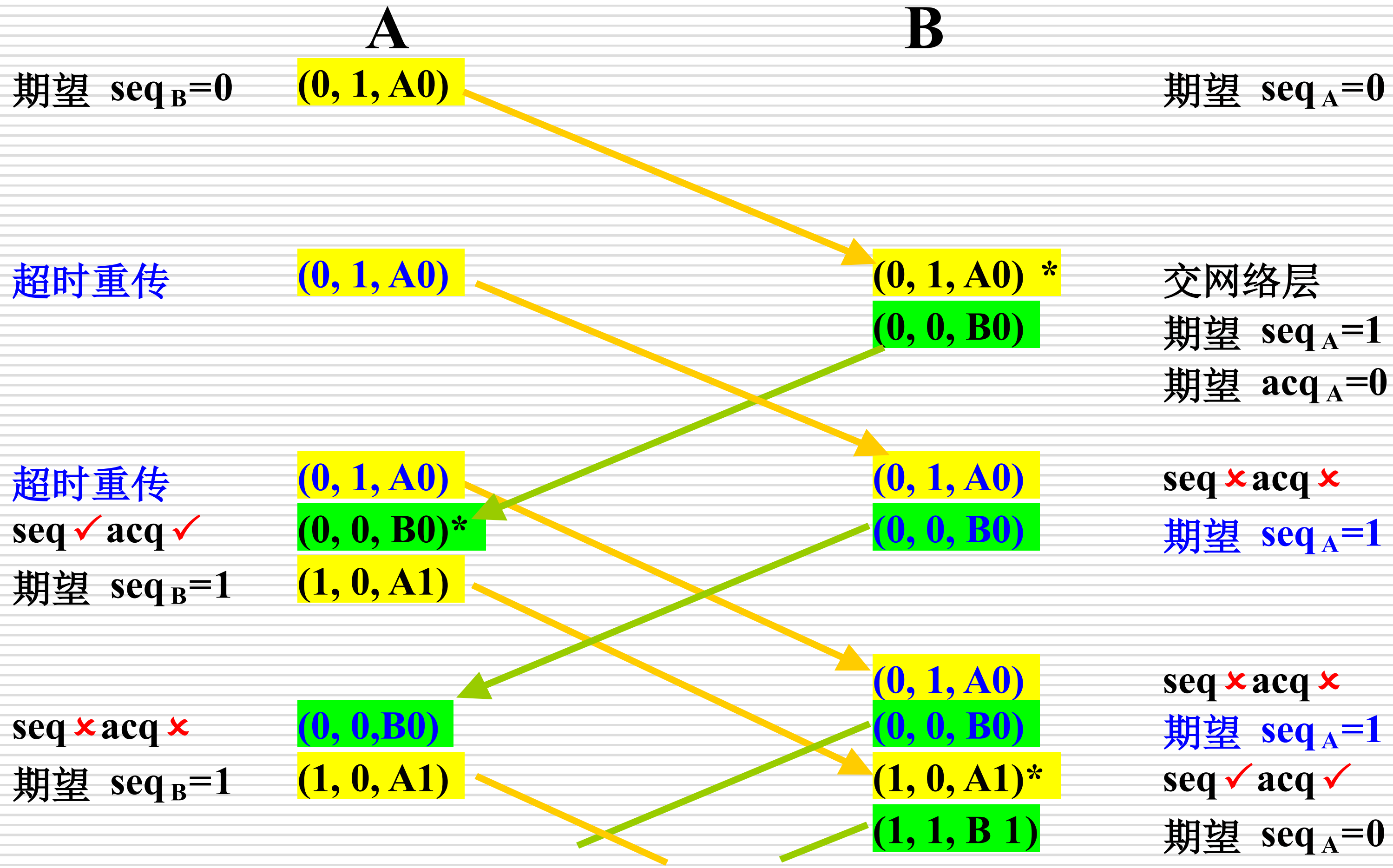
- 1-bit sliding window protocol （w=1）
- Window setting
  - Sliding window maximum： MAX_SEQ = 1
  - Initial value： seq =0, ack=1（期待接收seq=0）
- Window sliding scheme
  - A send frame firstly（seq=0, ack=1, A0）
  - B receive A0， send back piggyback-acknowledge （seq=0, ack=0, B0）
  - A receive acknowledge （to A0）， sliding window， send next frame（seq=1, ack=0, A1）
- characteristic
  - 序列号seq和确认值ack“0”“1”交替
  - 滑动窗口长度W=1，收到确认才移动窗口
  - 保证按顺序将接收到的正确帧只一次上交网络层

Normal case of protocol 4



（seq， ack， data）

Abnormal case 1: duplicated



Abnormal case 1: error control

	A	B	
期望(0,0)	(0, 1, A0)	(0, 1, B0)	期望(0,0)
		(0, 1, A0)*	seq✓ack✗
		(0, 0, B0)	期望(1,0)
seq✓ack✗	(0, 1, B0)*		
期望(1,0)	(0, 0, A0)		
丢弃	(0, 0, B0)	(0, 0, A0)	丢弃
期望(1,1)	(1, 0, A1)	(1, 0, B1)	期望(1,1)
		(1, 0, A1)*	seq✓ack✗
		(1, 1, B1)	期望(0,1)
seq✓ack✗	(1, 0, B1)*		
期望(0,1)	(1, 1, A1)		
丢弃	(1, 1, B1)	(1, 1, A1)	丢弃
期望(0,0)	(0, 1, A2)	(0, 1, B2)	期望(0,0)
		(0, 1, A2)	seq✓ack✗
		(0, 0, B2)	期望(1,0)

Line utilization rate of protocol4（信道利用率）

Line utilization rate of protocol 4 （信道利用率）

- **An assumption in protocol 4: the time is negligible.**
  - **The transmission time forward or back**
  - **The processing time for the receiver to handle the incoming frame**
- **In fact, the round-trip time can be very large in low rate channel, and the sender is blocked during the period.**
- **Line utilization rate:**
  - **channel capacity is b bps**
  - **frame size k bits**
  - **round-trip propagation time R sec**

$$\boxed{\text{Line Utilization Rate} = k/(k+bR)}$$

Example

- Assumption:
  - channel capacity b = 50 kbps
  - round-trip propagation R = 500 ms
  - frame size k = 1000 bit
  - Receiver immediately send back acknowledge frame when it receives a frame
- Question: how much is line utilization rate?

Reference key

- Time for sending frame

  $T_f = k/b = 20$ ms

- Round Trip Time

  R = 500 ms

- Time from very-beginning to acknowledgement-back

  $(T_f + R) = 20 + 500 = 520$ ms

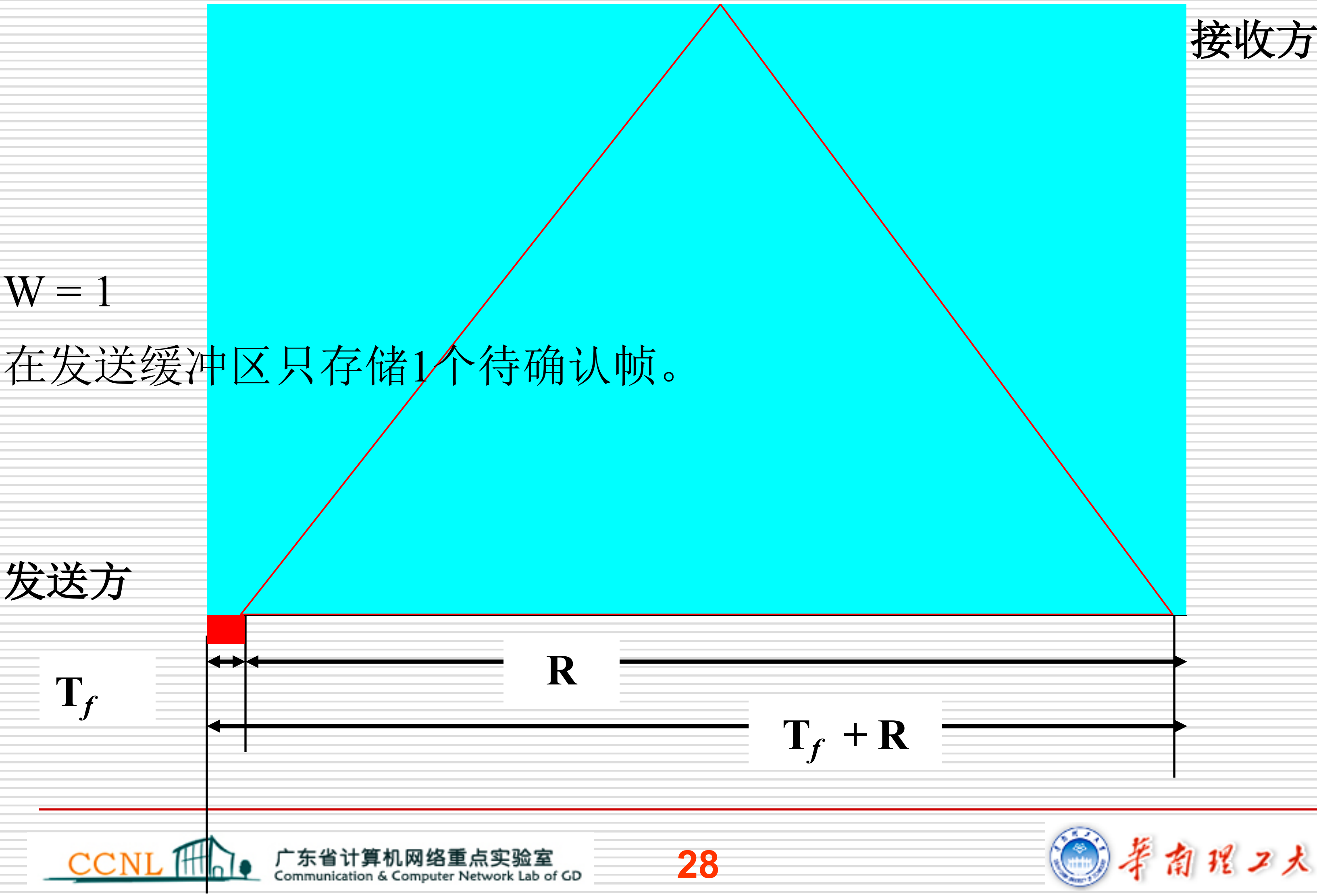
- Line utilization rate

  $T_f/(T_f + R) = 20/520 = 3.85\ \%$

Less than 4%

W = 1

在发送缓冲区只存储1个待确认帧。



How to improve line utilization?

- Increase maximum sliding-window size W:

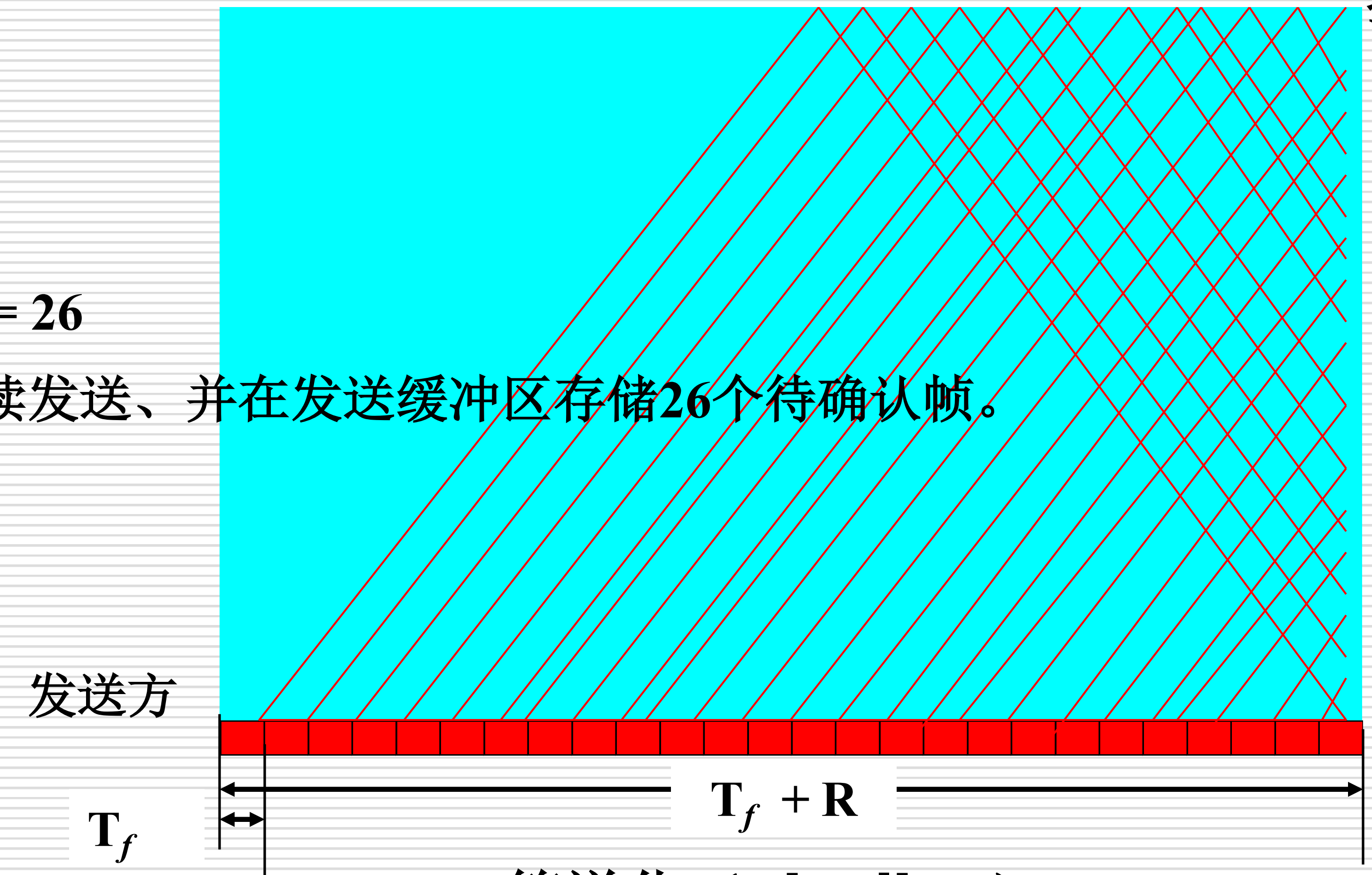
  Line utilization $= W*T_f / (T_f + R)$

  $= W*k / (k + bR)$

- Utopian case: line utilization rate is up to 100%，then maximum sliding-window size is:

  $W = (T_f + R) / T_f = 520 / 20 = 26$

Line utilization rate is 100%



W = 26

连续发送、并在发送缓冲区存储26个待确认帧。

管道化（**pipeling**）

Pipeline

- Allowing the sender to transmit up to **w frames** before blocking.
- Whenever the product of **bandwidth and round-trip-delay is large**, a large window on the sending side is needed.
- What to do if a frame in the middle of a long stream is damaged or lost?
  - Go back n
  - Selective repeat

Protocol 5：go back n

receiver：

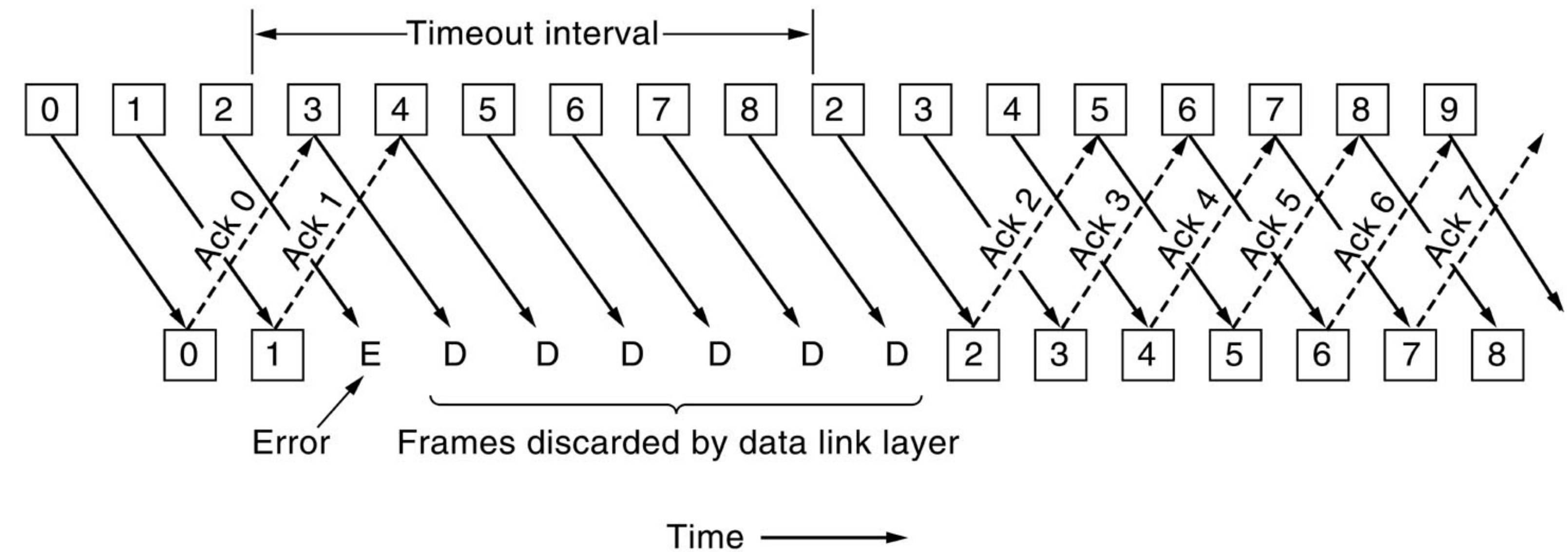
Just discard wrong frame and latter frames(后续帧) （receiving-window is 1）

Sender：

Resend wrong frame and latter frames

Basic idea of protocol 5

- The receiver's window size is 1, that is to say, the receiver only accepts frames in order.
- The receiver discards all frames following an error, sending no acknowledgements.
- After time out, the sender retransmits all unacknowledged frames in order, starting with the damaged or lost one.



Construct a data frame

s.kind=data

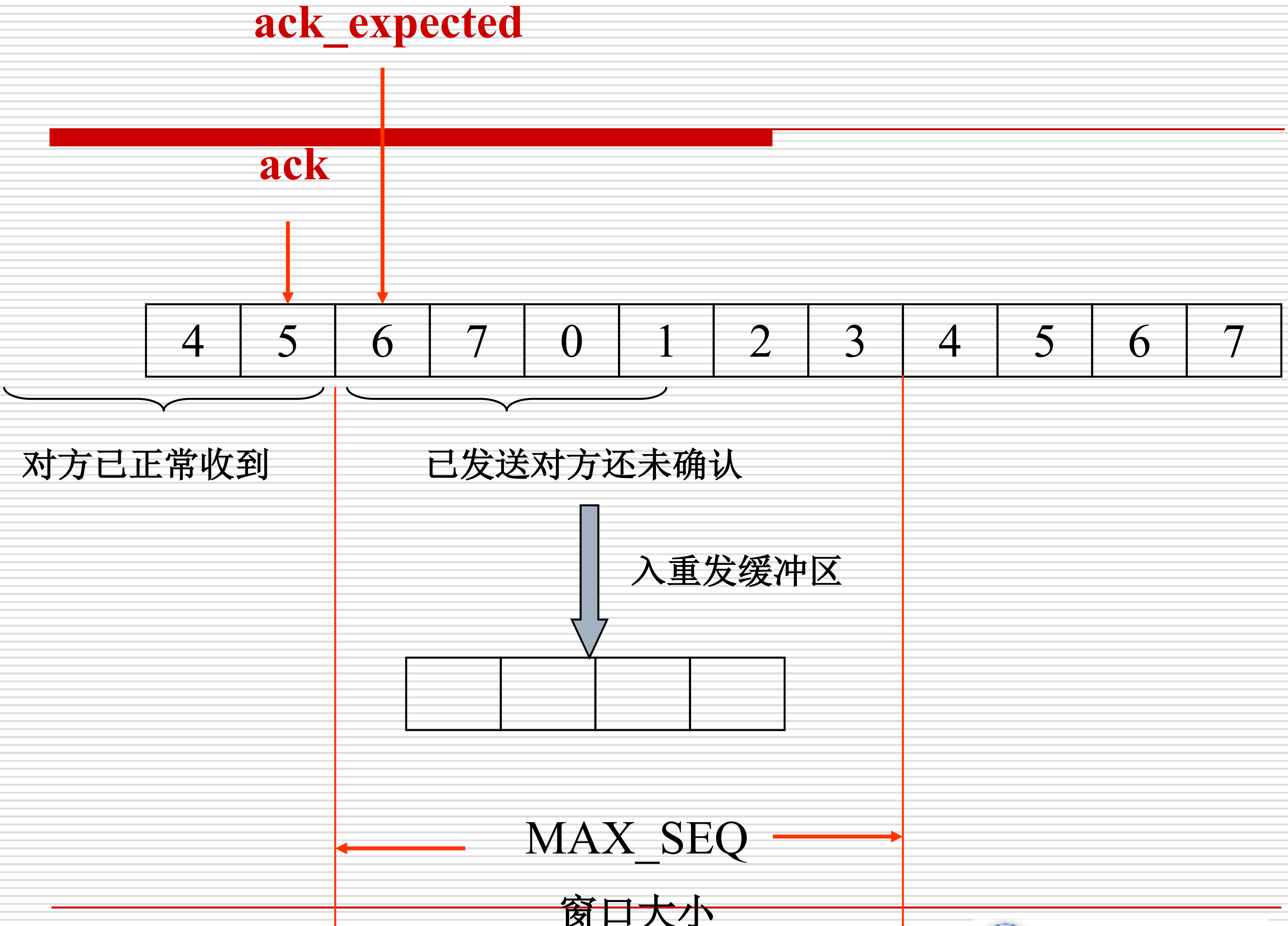
s.info=buffer[frame_nr]

s.seq=frame_nr

s.ack=(frame_expected+Max_seq)%(Max_seq+1)

F	A	C	N(S)	P/F	N(R)	Info.	FCS	F

```
While (between(ack_expected ,r.ack, next_frame_to_send ))
    {
    nbuffered= nbuffered-1        （发送窗口上边界）
    stop_timer(ack_expected )
    inc(ack_expected )            （发送窗口下边界）
    }
```


ack_expected
ack
4 5 6 7 0 1 2 3 4 5 6 7
对方已正常收到
已发送对方还未确认
入重发缓冲区
MAX_SEQ
窗口大小

回退n帧协议的工作原理分析

发送方

- 正常发送
  - 对帧编号，待确认帧缓存
- 收到确认
  - 释放确认帧所占缓冲区，滑动发送窗口
- 差错帧超时时间到
  - 回退到超时帧（出错帧），顺序重传最后被确认帧以后的缓冲区中缓存的帧

接收方

- 收到每一个期望的正确帧
  - 上交网络层、回送确认
- 收到出错帧或非期望帧
  - 丢弃，回送对接收的最后正确帧的确认
- 收到重传帧（即为一个期望的正确帧）

Protocol 6：selective repeat

Receiver：

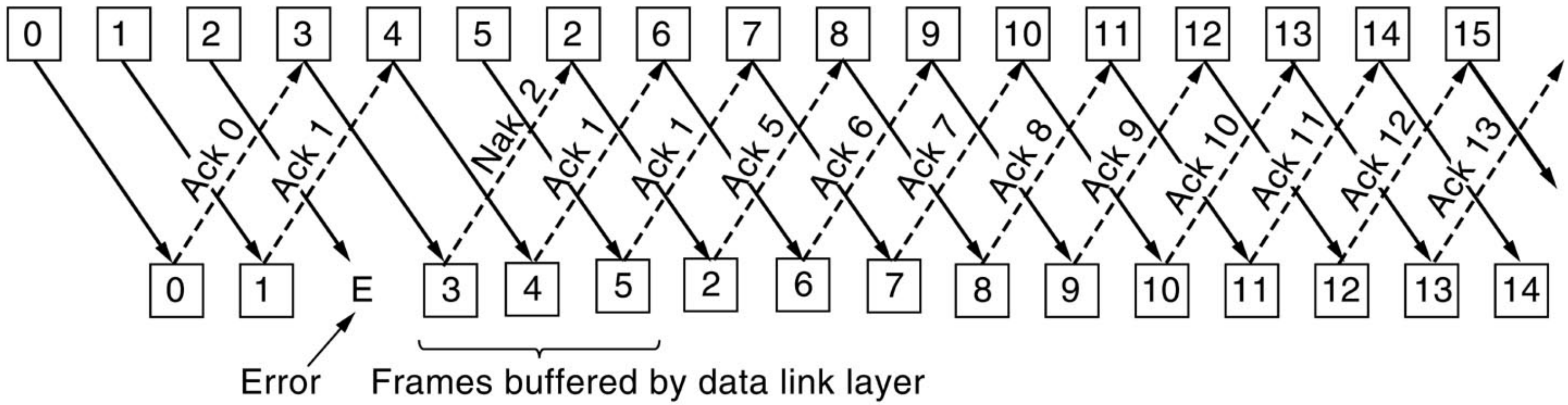
just discard wrong frame，and store all latter correct frames

Sender：

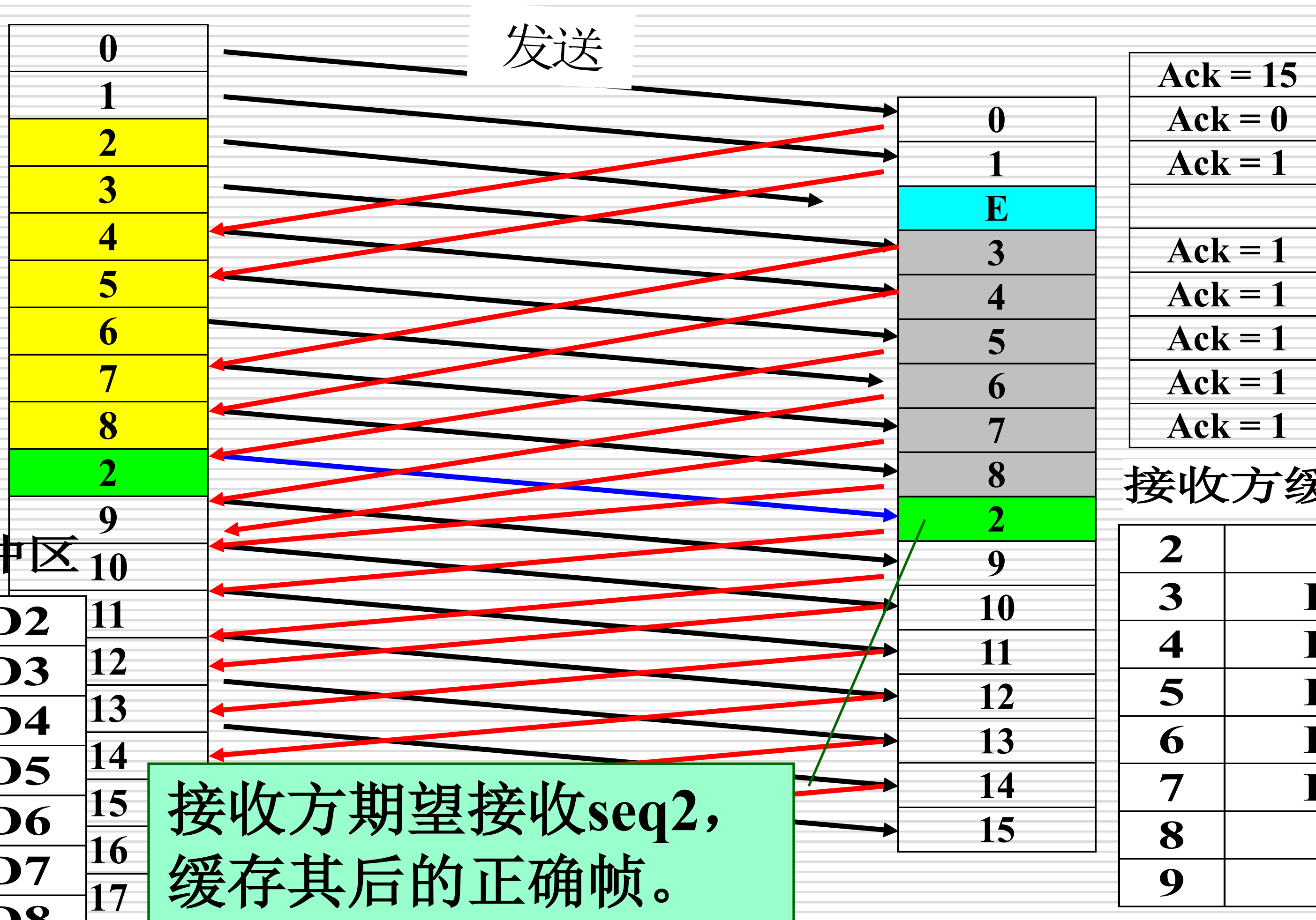
Just resend wrong one

Basic idea of protocol 6

- Protocol 6 accepts frames out of order but passed packets to the network layer in order.
  - Receives all frames whose sequence number fall within the receiving window
  - Doesn't pass the frame to the network layer until all the lower-numbered frames have already been delivered to the network layer in the correct order.



An example:
接收方收到非期望的正确帧—缓存



发送方缓冲区

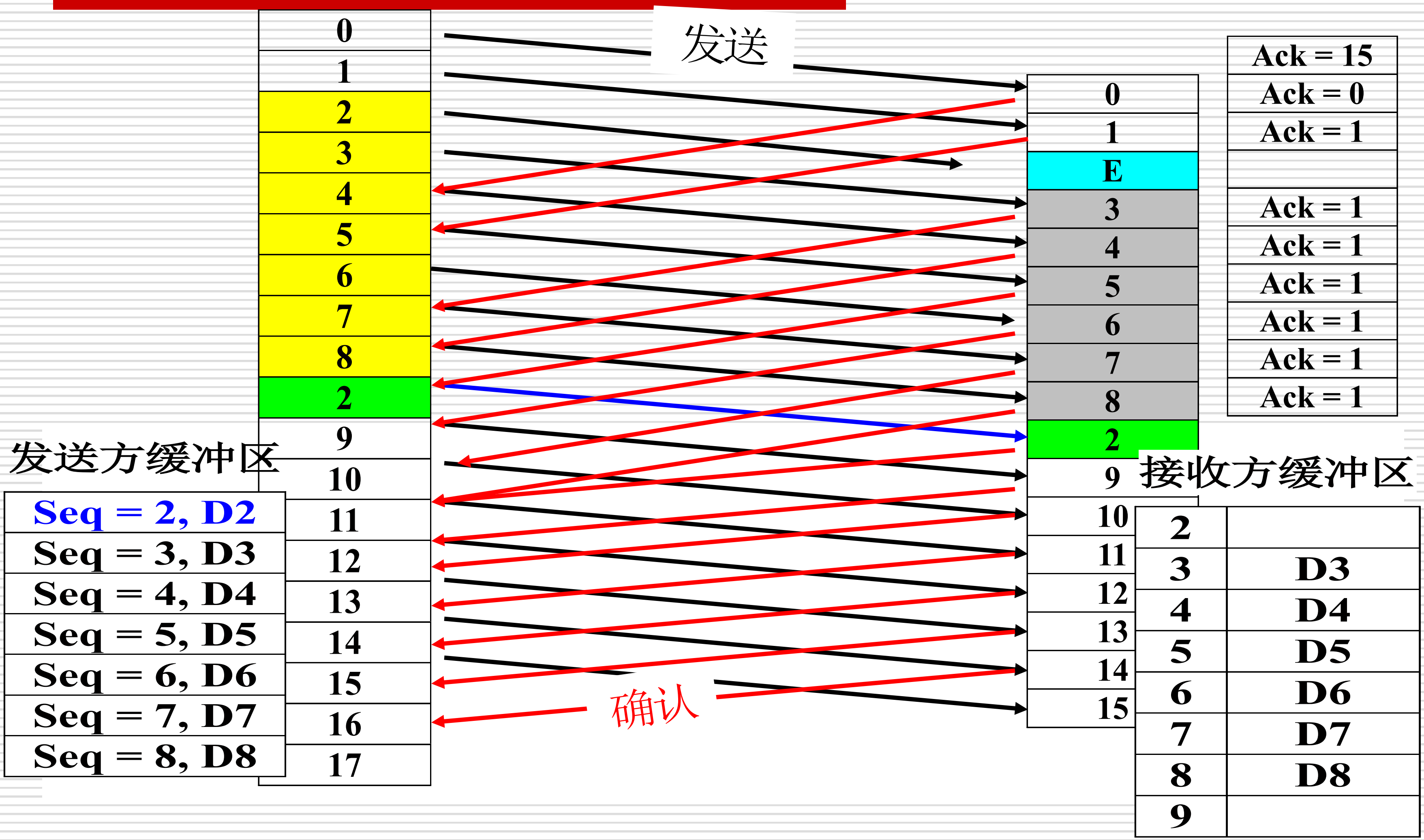
Seq = 2, D2
Seq = 3, D3
Seq = 4, D4
Seq = 5, D5
Seq = 6, D6
Seq = 7, D7
Seq = 8, D8

接收方缓冲区

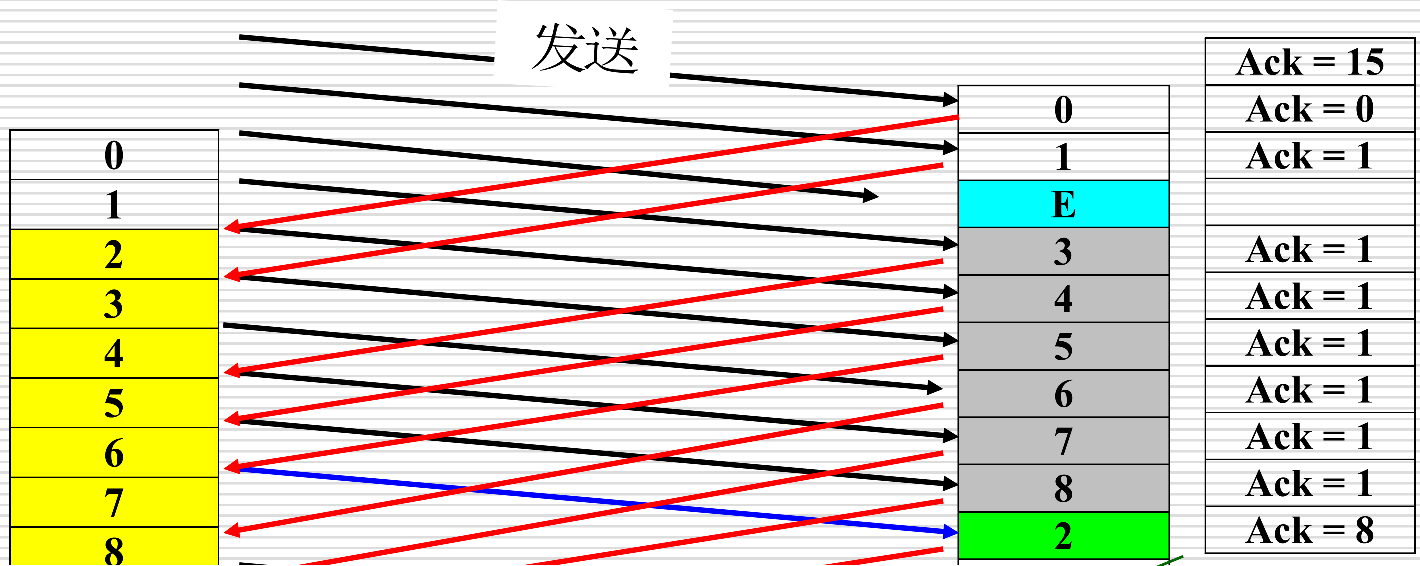
2	
3	D3
4	D4
5	D5
6	D6
7	D7
8	
9	

接收方期望接收seq2，缓存其后的正确帧。

An example：
发送方选择帧seq2重传



An example:
接收方收到重传帧seq2—排序上交



将帧seq2和接收方缓冲区中的帧正确排序，提交网络层，回送ack=8，滑动接收窗口。

接收方缓冲区

9	
10	
11	
12	
13	
15	
0	
1	

Function of NAK

- Negative acknowledgement (否定确认)
- More efficient than protocol 5
  - Send back NAK for wrong frame to sender, force sender resend the frame immediately instead of timeout

Comparison of error control policy and decision of sliding-window size

Comparison of protocol 5&6

- **Go back n (protocol 5)**
  - **Sender needs large buffer**
  - **Repeat rate is large (waste bandwidth)，suit for low-error rate environment**
- **Selective repeat (protocol 6)**
  - **Receiver needs large buffer**
  - **Repeat rate is small，suit for bad-quality channel**

Decision of sliding-window size

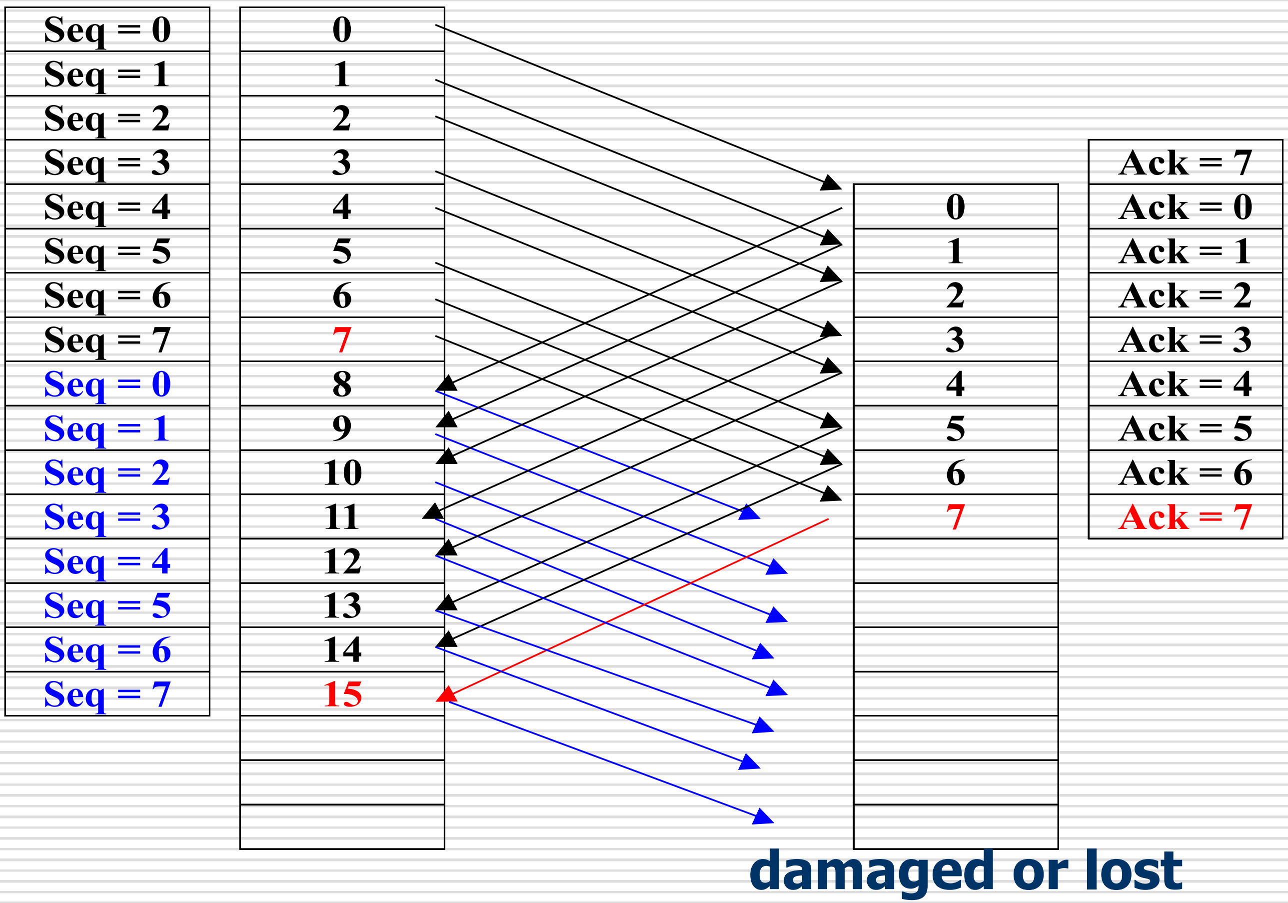
- Protocol 5（回退n帧）
  - MAX_SEQ = 7（ Seq=0～ MAX_SEQ ）
  - W = 7

W = MAX_SEQ (sending-window)

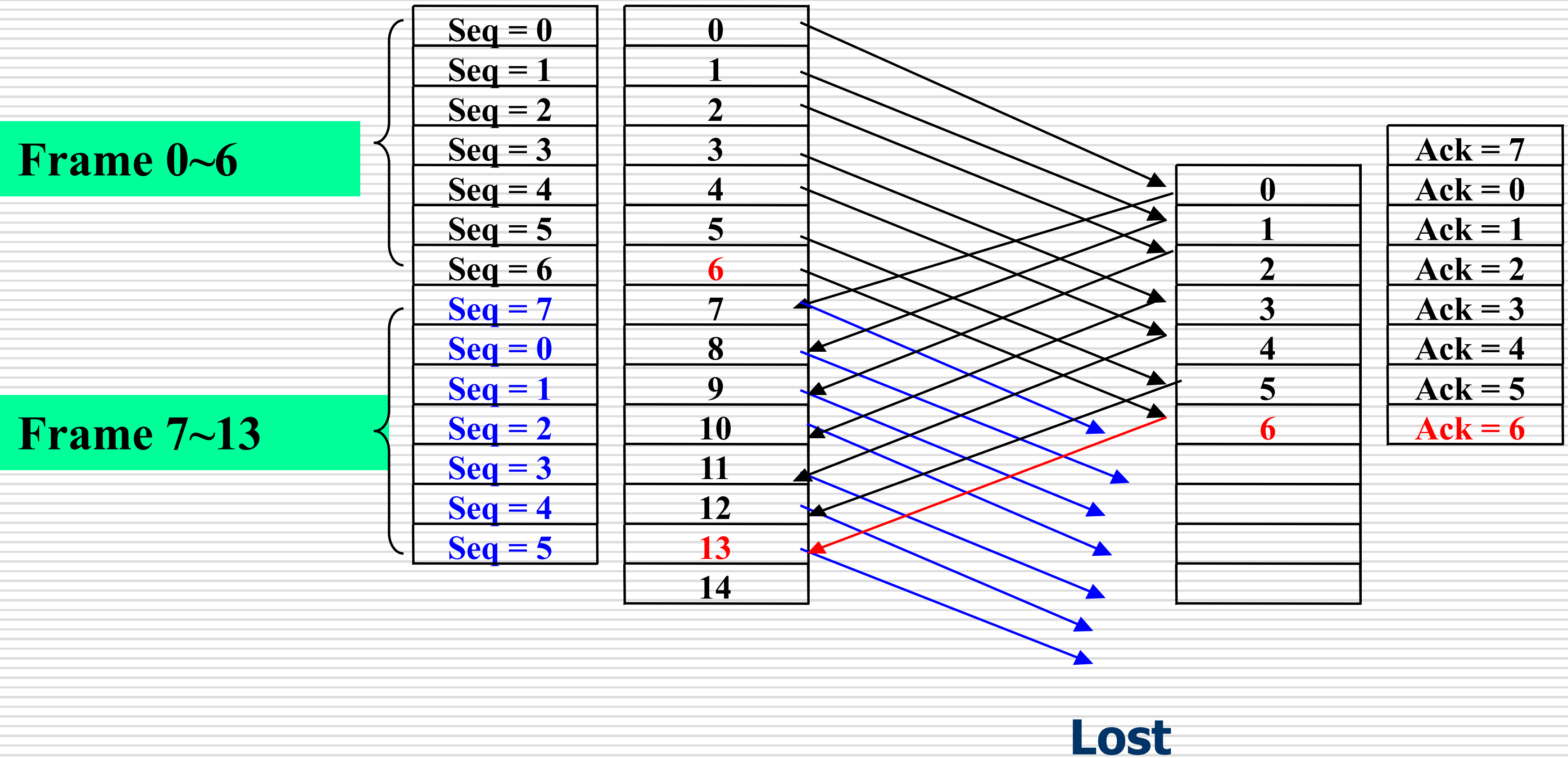
- Protocol 6（选择重传）
  - MAX_SEQ =7（ Seq=0～ MAX_SEQ ）
  - W = 4

W= (MAX_SEQ + 1) / 2 (receiving-window)

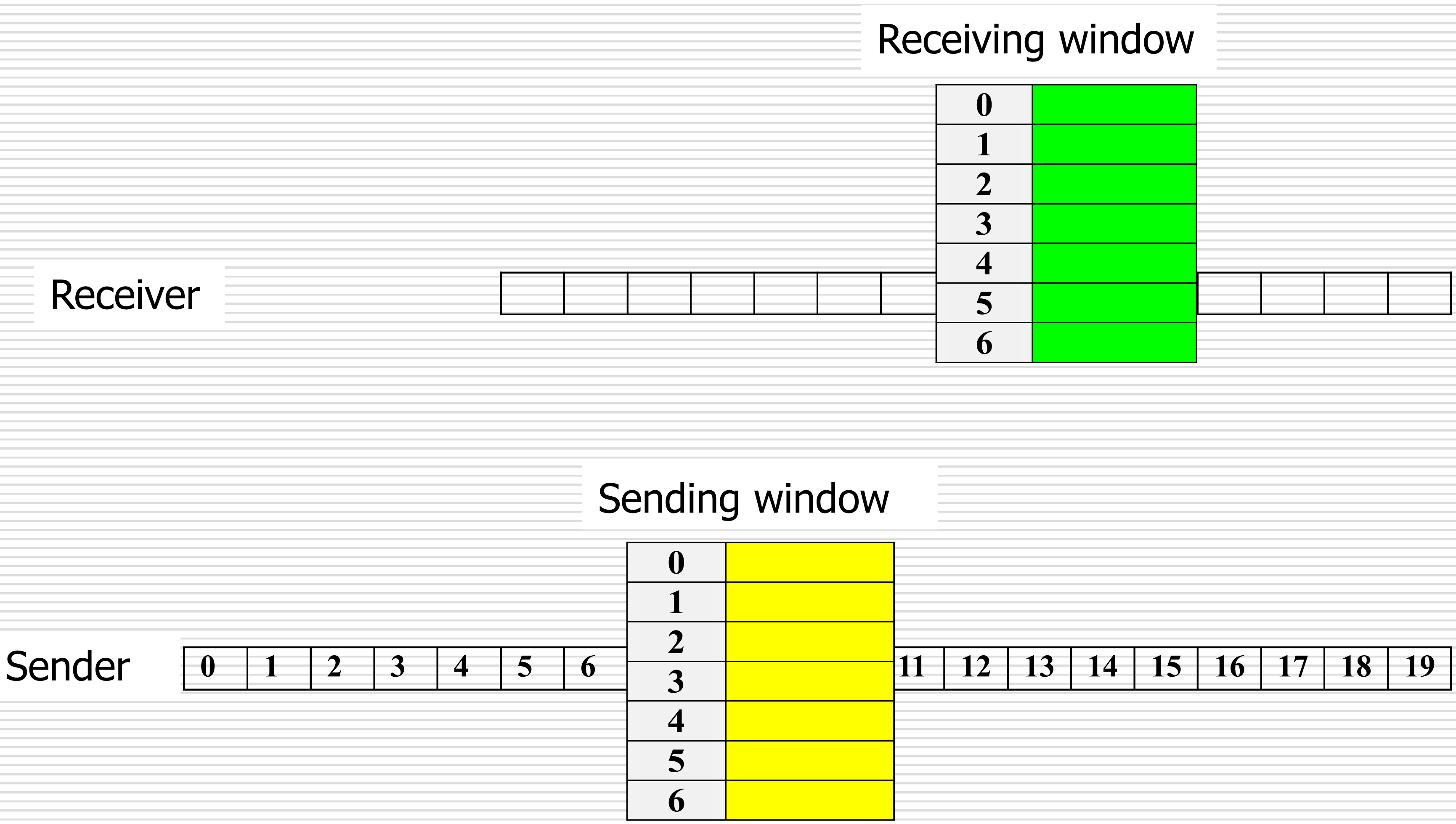
Protocol 5: W = 8, abnormal



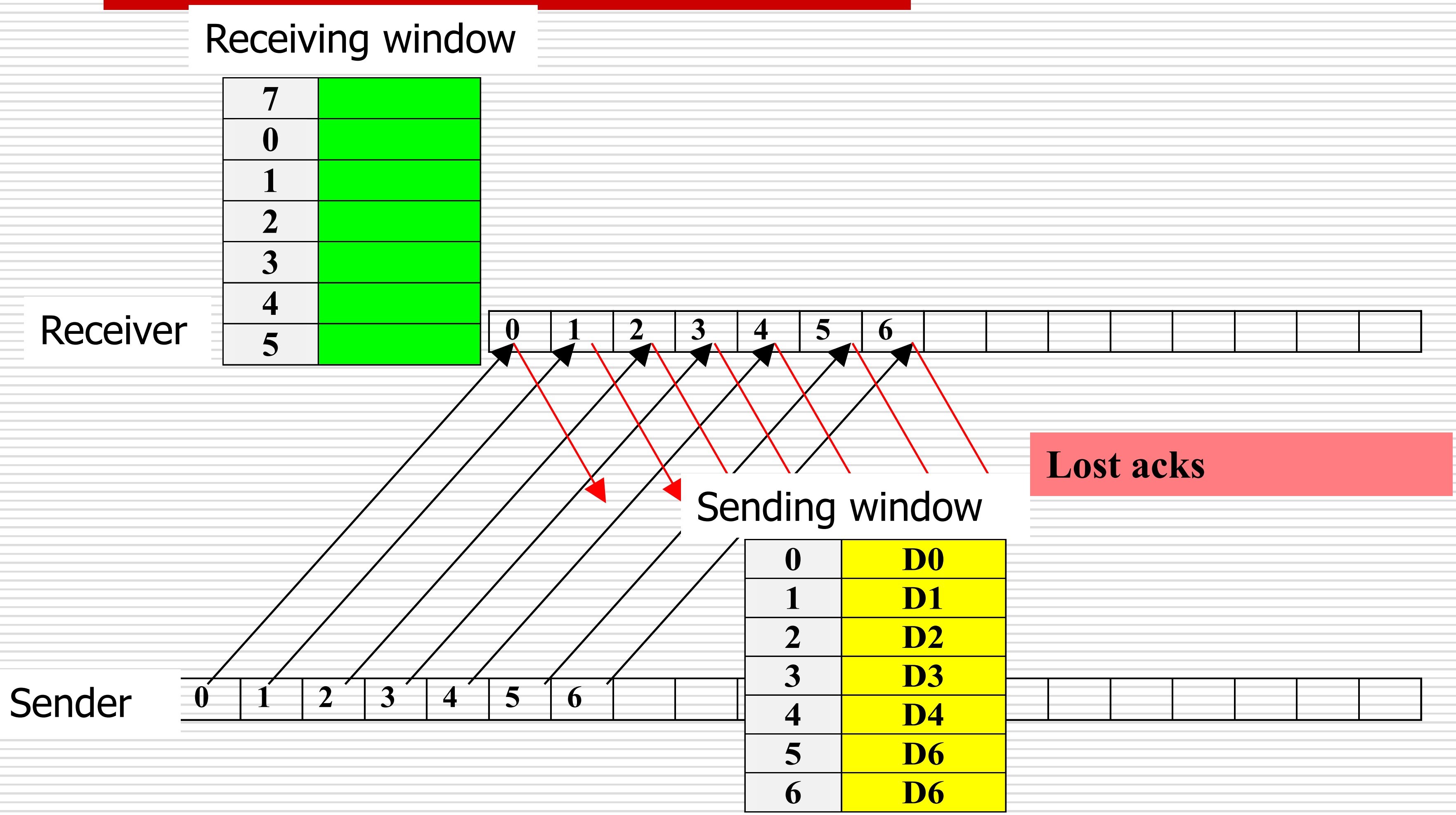
Protocol 5: W = 7， abnormal



Protocol 6: W=7



Protocol 6: Lost acknowledgement



Protocol 6: Lost acknowledgement

Receiving window

7	
0	D0
1	
2	
3	
4	
5	

Receiver

0	1	2	3	4	5	6	0							

Sending window

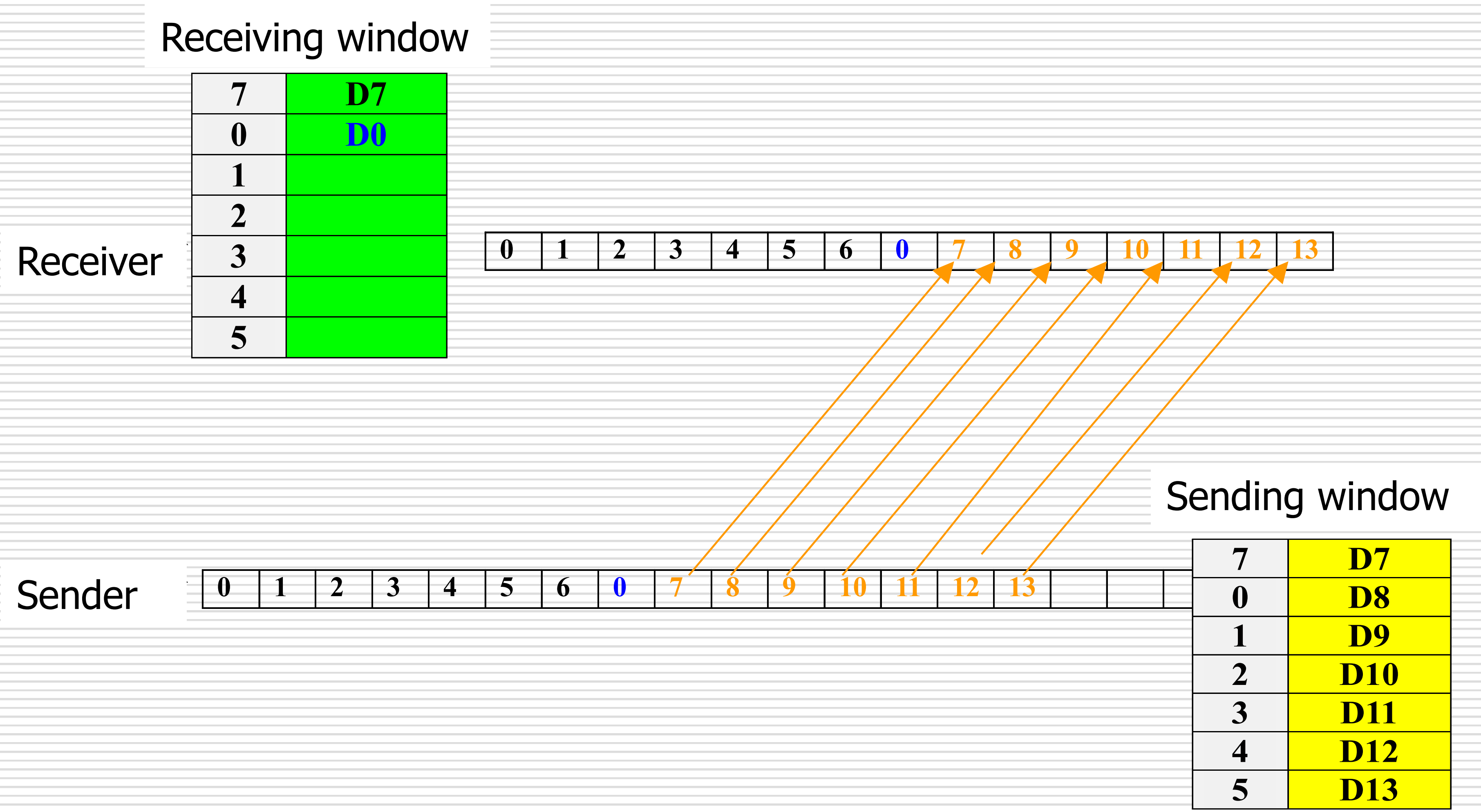
0	D0
1	D1
2	D2
3	D3
4	D4
5	D6
6	D6

Sender

0	1	2	3	4	5	6	0												

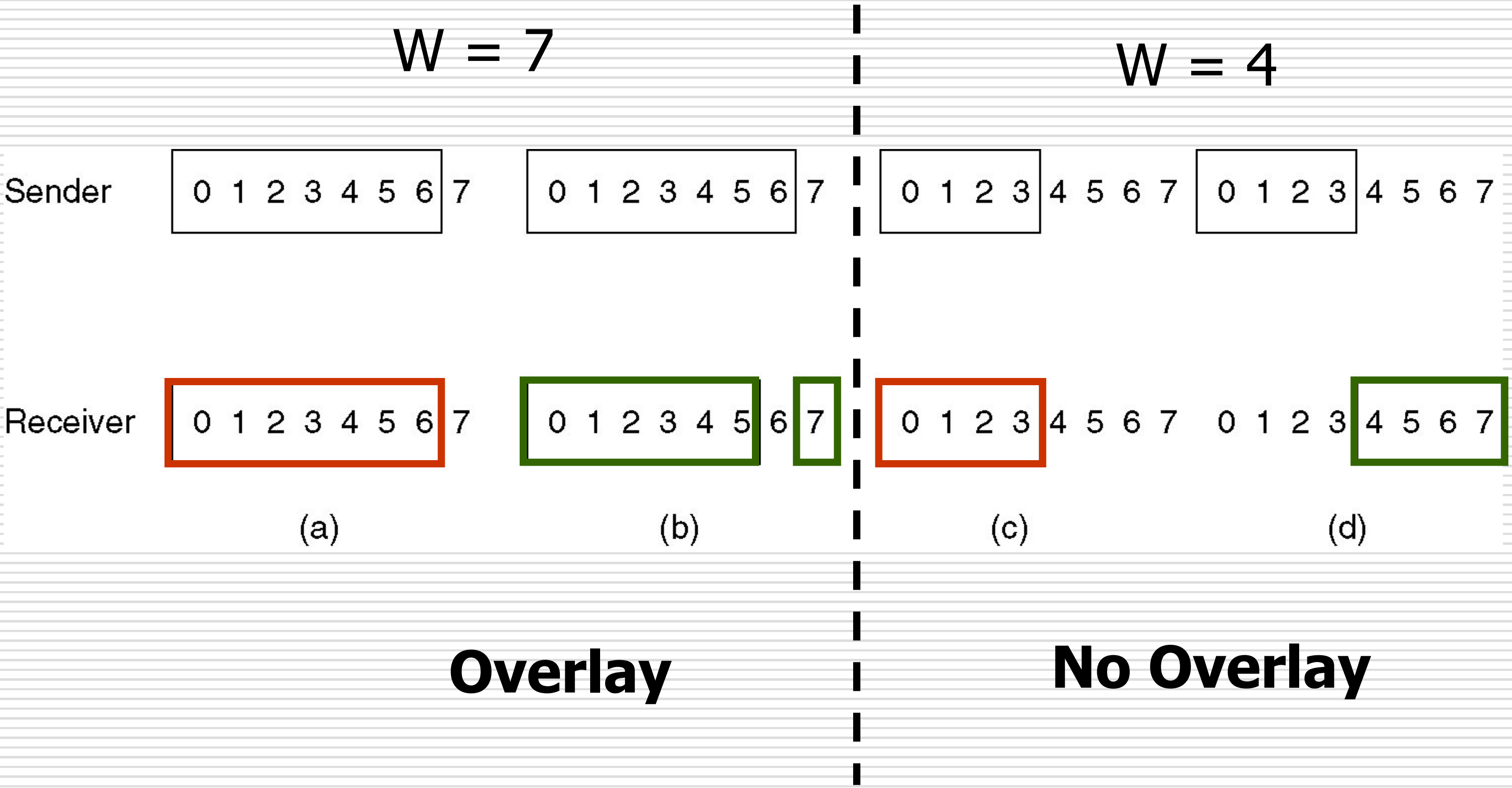
Retransmit frame 0

Protocol 6: Lost acknowledgement

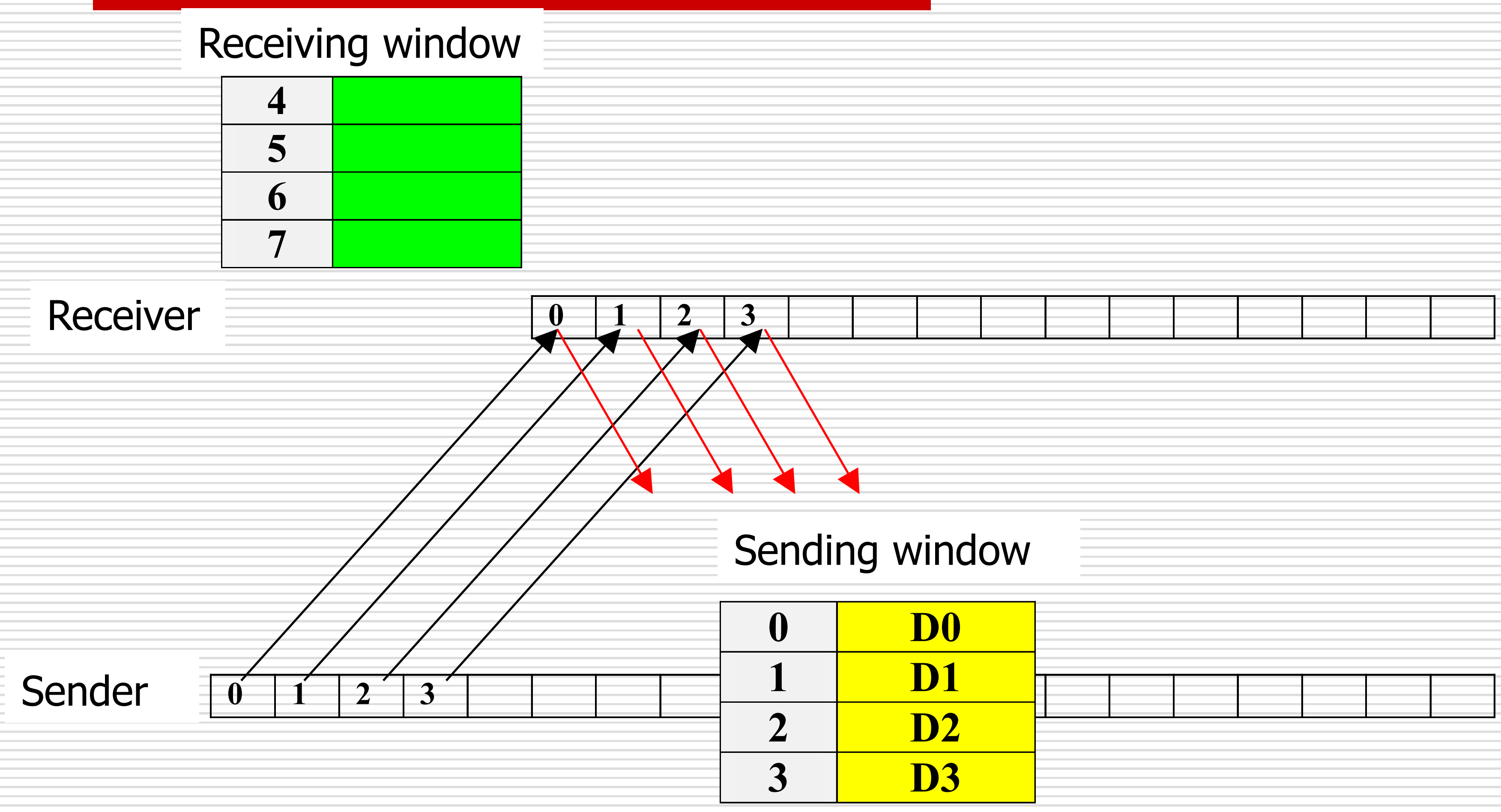


Solution: no overlay of windows

MAX_SEQ=7



Protocol 6: W=(MAX_SEQ+1)/2



W-Size difference of 3 protocol

- One-Bit sliding window (protocol 4):
  - 0 <= size of Sending window<=1
  - size of receiving window=1
- Go-back-N (protocol 5):
  - 0 <=size of Sending window<=MAX_SEQ
  - size of receiving window=1
- Selective Repeat (protocol 6):
  - 0 <= size of Sending window<= (MAX_SEQ+1)/2
  - size of receiving window= (MAX_SEQ+1)/2

Summary of this lecture

- Learn 6 elementary DLL protocol
- Learn & master sliding-window
- Learn & master ARQ（自动重复请求）/PAR
- Learn & master piggybacking(捎带确认)
- Learn pipeline
- Go back n & selective repeat