Computer Networks

Chapter 3:Data Link Layer

(Version March 22, 2023)

Xiao Mingzhong

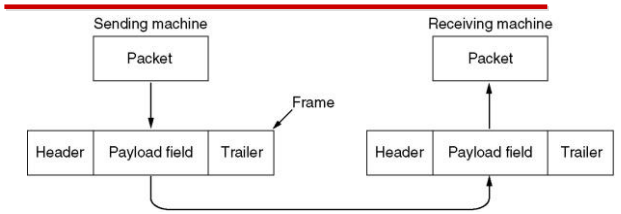
CIST, Beijing Normal University

Data Link Layer

- Physical: 二进制位的传输问题，涉及机械和电气的问题。
- Data Link: 探讨如何使用shared信道进行通信，以及数据帧的reliably 传输问题。
- Network: 主要涉及通信子网的路由问题。
- Transport: 通常提供面向连接和无连接的服务给应用，具有不同的E2E传输可靠性。
- Application: contains the stuff that user see: e-mail, remote logins, the web's exchange protocol, etc.

- Note: we'll just concentrate on transmission issues. Channel access is discussed in next chapter.

设计问题



Relationship between packets and frames.

- 提供给网络层良好的 service interface
- 差错控制：处理 point-to-point(P2P) 传输 errors
- 流量控制：get sender and receiver in the same pace

Basic services

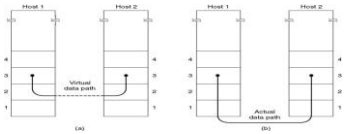
- Network layer passed a number of bits (packet) to the data link layer.
- Data link layer is responsible for transmitting the packet to the destination machine.
- Receiving layer passed received packet to its network layer.

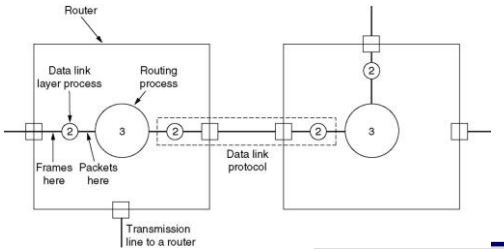
Basic services commonly provided:

- Unacknowledged无连接的服务 (LANs)
- Acknowledged无连接的服务(Wireless Systems)
- Acknowledged面向连接的服务 (WANs)

Transmission and routing

- (a) Virtual communication.
- (b) Actual communication.



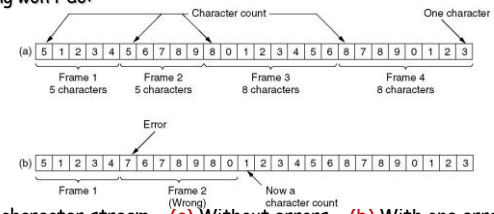


Placement of the data link protocol in Router.

成帧

- The physical layer doesn't do much: it just pumps bits from one end to the other. But things may go wrong → the data link layer needs a means to do retransmissions. The unit of retransmission is a frame (which is just a fixed number of bits).
- Problem: How can we break up a bit stream into frames?

Counting won't do:

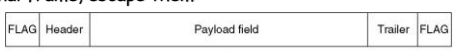


A character stream. (a) Without errors. (b) With one error.

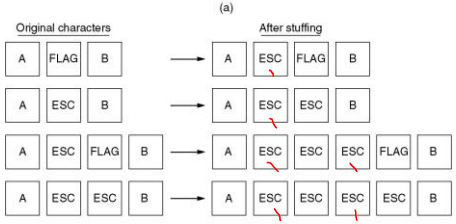
成帧：填充(1/2)

- Byte stuffing: Mark the beginning and end of a byte frame with two special flag bytes – a special bit sequence (e.g. 01111110). If such bytes appear in the original frame, escape them:

(a) A frame delimited by flag bytes.



(b) Four examples of byte sequences before and after stuffing.



评:flag bytes考虑了错误之后的重新同步问题:两个连续的flag

成帧：填充(2/2)

- Byte stuffing is closely tied to the use of 8-bit character
- Bit stuffing: Escape the flag byte (e.g., 01111110) through an additional bit:

(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

(b) 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0

Stuffed bits

(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Bit stuffing

(a) The original data.

(b) The data as they appear on the line.

(c) The data as they are stored in receiver's memory after destuffing.

评:支持任意长度的字符,而不只是8位长.

纠错和检错

- Problem: 假定帧传输过程中出了错误，接收方如何知道出了错误，并纠正呢？ (m+r → m'+r')
- Definition: 海明距离指两个帧同样位置上，取值不同的位置数目。Example: 10001001 and 10110001海明距离是3.
  - 为检测至多k位错，充要条件是the hamming distance between any two frames is k+1 or more.
  - 为纠正至多k位错，充要条件是the hamming distance between any two frames is 2k+1 or more.

检错码: Parity

- Essence: add a bit to a bit string such that the total number of 1-bits is even (or odd) → the distance between all frames is at least 2.
  - e.g., (even parity) 1011010 → 10110100
- Conclusion: we can detect a single error

纠错码: Hamming(1/3)

- Essence: every bit at position 2^k,k>=0 is used as a parity bit for those positions to which it contributes (mark positions left to right, start with position 1):

	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11
1	X		X		X		X		X		X
2		X	X			X	X			X	X
4				X	X	X	X				
8								X	X	X	X

- Conclusion: check bit at position 2, is used to even out the bits for positions 2,3,6,7,10, and 11, so that 1100001 is encoded as (check bits in boldface/blue):

	b1	**b2**	b3	**b4**	b5	b6	b7	**b8**	b9	b10	b11
1	**X**		X		X		X		X		X
2		**X**	X			X	X			X	X
4				**X**	X	X	X				
8								**X**	X	X	X
	1	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**

纠错码: hamming (2/3)

- Observation: if a check bit at position p is wrong upon receipt, the receiver increments a counter v with p; the value of v will, in the end, give the position of the wrong bit, which should then be swapped.

	b1	**b2**	b3	**b4**	b5	b6	b7	**b8**	b9	b10	b11
1	**X**		X		X		X		X		X
2		**X**	X			X	X			X	X
4				**X**	X	X	X				
8								**X**	X	X	X
3→ C:	**0**	**0**	1	**1**	1	0	0	**0**	**1**	0	1
1→ S:	**1**	**0**	1	**1**	1	0	0	**1**	0	0	1
2→ R:	**1**	**0**	1	**1**	1	0	0	**1**	1?	0	1
4→ F:	**1**	**0**	1	**1**	1	0	0	**1**	0	0	1

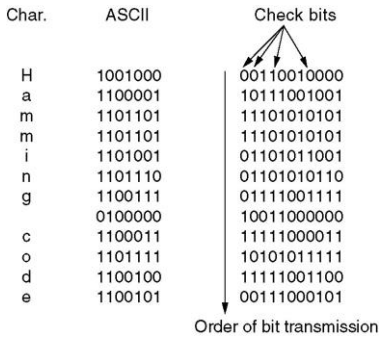
S: string sent R: string received F: final result after correction

C: string corrected on the check bits: #1 and #8 corrected → bit #9 is wrong

海明码只能纠正单个错误

纠错码： hamming (3/3)

- Use of a Hamming code to correct burst errors.



如果突发性错误长度为12，则每个字符至多只有一位受影响

检错码: CRC

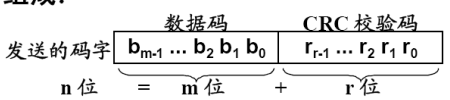
- Problem: Error correcting codes are simply too expensive → only use error detection combined with retransmissions.
  - m 个报文位, r个校验位, 能纠单个错误,n=m+r
  - 2^m个合法报文,每个都对应有n个非法的码字,他们与该报文的距离为1. 每个合法报文都要求n+1个位模式,专门供它使用
  - (n+1)*2^m 应该小于等于 2^n
  - 即: (m+r+1)<=2^r
  - 意义：给定m的情况下，纠单个错误需要的校验位数目下界。
- Example: cyclic redundancy check (CRC)

检错码：循环冗余码－CRC码

特点：实现容易，检错能力强，广泛使用。

常结合反馈重传法来保证信息的可靠传输。

码字组成：



编码与解码的计算采用二进制比特序列多项式。

二进制比特序列多项式：

$$M(x)=b_{m-1}x^{m-1}+\dots+b_i x^i+\dots+b_1 x+b_0$$

其中，$b_i=0$或1，$m-1 \geqslant i \geqslant 0$，共$m$位

例：若数据码 = 110011，

可表示为：$M(x)=1\cdot x^5+1\cdot x^4+0\cdot x^3+0\cdot x^2+1\cdot x+1$

即：$M(x)=x^5+x^4+x+1$。

生成多项式$G(x)$：

$$G(x)=g_r x^r+\dots+g_i x^i+\dots+g_1 x+g_0$$

其中，$g_i=0$或1，$r \geqslant i \geqslant 0$，$g_r \neq 0$，$g_0 \neq 0$，共r+1位

$G(x)$被通信双方事先共同选定使用：

发送端：通过$G(x)$生成校验码；

接收端：通过$G(x)$校验接收的码字。

对于多项式的运算：采用模2计算（加法不进位，减法不借位）；加减法是一样的。

模2计算即是异或运算。

在发送端:

1. 生成校验码$R(x)$：

把要发送的数据码$M(x)\cdot x^r$去除$G(x)$，所得的余数值$R(x)$就是循环冗余码（简称CRC校验码）。

$$\frac{M(x)\cdot x^r}{G(x)}=Q(x)+\frac{R(x)}{G(x)}$$

$Q(x)$为商

$R(x)$为余数(CRC校验码)

$$\frac{M(x)\cdot x^r-R(x)}{G(x)}=Q(x)$$

$\because G(x)$共r+1个bit位，

$\therefore R(x)$共r个bit位

2. 发送$M(x)\cdot x^r+R(x)$:

即：$M(x)\cdot x^r$ 末r位是$R(x)$

实际上，把CRC校验码$R(x)$附加到数据码$M(x)$的后面，就构成编码多项式$M(x)\cdot x^r+R(x)$，然后发送传输。

在接收端:

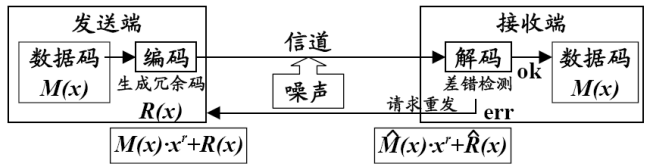
接收到 $\hat{M}(x)\cdot x^r+\hat{R}(x)$ 后，按如下操作进行校验:

$$\frac{\hat{M}(x)\cdot x^r+\hat{R}(x)}{G(x)}=Q(x)+\frac{E(x)}{G(x)}$$

$Q(x)$为商

$E(x)$为余数

若$E(x)$ $=0$ 无错；$\neq 0$ 有错



Example:

Frame : 1101011011

Generator: 10011

Message after 4 zero bits are appended: 11010110110000

```
            1100001010
      ----------------
10011 | 11010110110000
        10011
        -----
        10011
        10011
        -----
        00001
        00000
        -----
        00010
        00000
        -----
        00101
        00000
        -----
        01011
        00000
        -----
        10110
        10011
        -----
        01010
        00000
        -----
        10100
        10011
        -----
        01110
        00000
        -----
        01110   <- Remainder
```

Transmitted frame: 11010110111110

流行的生成多项式有：

CRC-8: $G(x)=x^8+x^2+x+1$

CRC-12: $G(x)=x^{12}+x^{11}+x^3+x^2+x+1$

CRC-16: $G(x)=x^{16}+x^{15}+x^2+1$

CRC-CCITT: $G(x)=x^{16}+x^{12}+x^5+1$

检测能力：

所有单个错、奇数个错和离散的二位错。

所有长度≤r位的突发差错。

CRC的实现：

①硬件：采用多段移位寄存器及异或门组成CRC校验电路来实现。

②软件：通过快速CRC校验软件来实现。

数据链路层协议

- Concentrated on design aspects and error control
  - Framing (stuffing), error correction & detection

Now: basic protocols and real-world examples

- Some basic assumptions:
  - we have a machine A that wants to send data to machine B
  - There is always enough data for A to send
  - There is a well-defined interface to the network layer, and to the physical layer
    - Void from_network_layer(packet *)
    - Void to_network_layer(packet *)
    - Void from_physical_layer(frame *)
    - Void to_physical_layer(frame *)
  - The receiver generally waits for an event to happen by calling wait-for-event
    - Void wait_for_event(event_type *event)

无约束的单工协议

```
01 typedef enum {false, true} boolean;
02 typedef unsigned int seq_nr;
03 typedef struct {unsigned char data[MAX_PKT];} packet;
04 typedef enum {data, ack, nak} frame_kind;
05
06 typedef struct {
07   frame_kind kind;  /* what kind of a frame is it? */
08   seq_nr seq;       /* sequence number */
09   seq_nr ack;       /* acknowledgement number */
10   packet info;      /* the network layer packet */
11 } frame;
12
13 typedef enum {frame_arrival} event_type;
14
15 void sender1(void){                          /* buffer for an outbound frame */
16   frame s; packet buffer;                    /* buffer for an outbound packet */
17   while (true) {
18         from_network_layer(&buffer);         /* go get something to send */
19         s.info = buffer;                     /* copy it into s for transmission */
20         to_physical_layer(&s);               /* send it on its way */
21   }                                          * Tomorrow, and tomorrow, and tomorrow,
22 }                                              Creeps in this petty pace from day to day
23                                                To the last syllable of recorded time
                                                     - Macbeth. V. v */
24 void receiver1(void){
25   frame r; event_type event;       /* filled in by wait, but not used here */
26   while (true) {
27         wait_for_event(&event);    /* only possibility is frame_arrival */
28         from_physical_layer(&r);   /* go get the inbound frame */
29         to_network_layer(&r.info); /* pass the data to the network layer */
30   }
31 }
```

Question: What are some of the underlying assumptions here? How does the flow control manifest itself?

单工停-等协议 (*Simplex Stop-and-Wait*)

```
01 typedef enum {frame_arrival} event_type;
02 #include "protocol.h"
03
04 void sender2(void){                          /* buffer for an outbound frame */
05   frame s; packet buffer;                    /* buffer for an outbound packet */
06   event_type event;                          /* frame_arrival is the only possibility */
07   while (true) {
08         from_network_layer(&buffer);         /* go get something to send */
09         s.info = buffer;                     /* copy it into s for transmission */
10         to_physical_layer(&s);               /* bye bye little frame */
11         wait_for_event(&event);              /* do not proceed until given the go ahead */
12   }
13 }
14
15 void receiver2(void){                        /* buffers for frames */
16   frame r, s; event_type event;              /* frame_arrival is the only possibility */
17   while (true) {
18         wait_for_event(&event);              /* only possibility is frame_arrival */
19         from_physical_layer(&r);             /* go get the inbound frame */
20         to_network_layer(&r.info);           /* pass the data to the network layer */
21         to_physical_layer(&s);               /* send a dummy frame to awaken sender */
22   }
23 }
```

Question: What are the assumptions in this case?

有噪音信道的单工协议

帧可能受损或丢失→ **Problems:**

- 发送者怎么知道一个帧是否正确到达接收方？

  Solution: let the receiver acknowledge undamaged frames.
- Acknowledgements may get lost.

  Solution: let the sender use a timer by which it simply retransmits unacknowledged frames after some time.
- The receiver cannot distinguish duplicate transmissions. Solution: use sequence numbers.
  - 序列号不必无休止下去，事实上we can (and need) only use a few of them. In our example: we need only two (0 & 1).

单工协议#3 (1/2)

```
01 #define MAX_SEQ 1
02 typedef enum {frame_arrival, cksum_err, timeout} event_type;
03 #include "protocol.h"
04
05 void sender3(void) {
06   seq_nr      next_frame_to_send;          /* seq number of next outgoing frame */
07   frame       s;                           /* scratch variable */
08   packet      buffer;                      /* buffer for an outbound packet */
09   event_type event;
10
11   next_frame_to_send = 0;                  /* initialize outbound sequence numbers */
12   from_network_layer(&buffer);             /* fetch first packet */
13   while (true) {
14
15     s.info = buffer;                       /* construct a frame for transmission */
16     s.seq = next_frame_to_send;            /* insert sequence number in frame */
17     to_physical_layer(&s);                 /* send it on its way */
18     start_timer(s.seq);                    /* if answer takes too long, time out */
19     wait_for_event(&event);                /* frame_arrival, cksum_err, timeout */
20     if (event == frame_arrival) {
21
22       from_physical_layer(&s);             /* get the acknowledgement */
23       if (s.ack == next_frame_to_send) {
24
25         stop_timer(s.ack);                 /* turn the timer off */
26         from_network_layer(&buffer);       /* get the next one to send */
27         inc(next_frame_to_send);           /* invert next frame to send */
28       }
29     }
30   }
31 }
```


单工协议#3(2/2)

```
01 void receiver3(void) {
02
03   seq_nr frame_expected; frame r, s; event_type event;
04
05   frame_expected = 0;
06   while (true) {
07
08     wait_for_event(&event);          /* possibilities: frame_arrival, cksum_err */
09     if(event == frame_arrival) {     /* a valid frame has arrived. */
10
11       from_physical_layer(&r);       /* go get the newly arrived frame */
12       if (r.seq == frame_expected) {/* this is what we have been waiting for. */
13
14         to_network_layer(&r.info);   /* pass the data to the network layer */
15         inc(frame_expected);         /* next time expect the other sequence nr */
16       }
17       s.ack = 1 - frame_expected;
18       to_physical_layer(&s);         /* tell which frame is being acked */
19     }                                /* send acknowledgement */
20   }
21 }
```


协议建模

- 为什么要建模?
  - 实际的协议及其实现是非常复杂的
  - 描述和验证协议正确性的需要
- 如何建模 ?
  - 抓住协议核心，忽略细节
  - 形式化的描述+严谨的数学分析技术
  - 两种建模工具：有限状态机和Petri网

有限状态机（1/5）

- 协议机及其建模技术
  - 协议机：指发送方和接收方（进程），在**任何一个时刻**总是处于一种特定的状态。其状态可通过所有变量的值来反映，包括程序计数器。
  - 协议机建模技术
    - 关注稳态，忽略瞬时态
      - 通常选择协议机在**等待**下一事件发生的那些时刻的状态
    - 抓住主要变量，舍弃不影响反映协议实质的变量
      - 状态数量2^n，n是表达这些变量所需要的位数
  - 例：协议3协议机的状态
    - 收：等待接收0/1号帧；发：发了0/1号帧

有限状态机（2/5）

- 通信信道及其建模技术
  - 信道：指**“对等”**通信双方收发信息的通路，在**任何一个时刻**总是处于一种特定的状态。
  - 信道建模技术
    - 信道状态可通过正在传输的内容来反映。
  - 例：协议3信道的状态
    - 0/1号帧从发送方往接收方移动
    - 确认帧沿着相反的方向移动
    - 信道为空
- 整个通信系统（协议）的状态
  - 定义为两个协议机和信道的所有状态的组合
  - 例: SRC=(0,0,0)---发了0，待收0，0在信道上

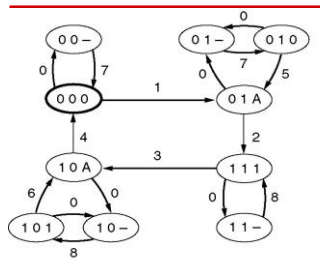
有限状态机（3/5）

- 转换及其建模技术
  - 转换（变迁）：实现系统状态的转变。对于每一种系统状态，有0个或者多个可能的转换，从而到达其他的状态。
  - 建模技术
    - 事件（发生）驱动
    - 事件举例：发出数据、数据到达、定时器到期、发生中断、数据丢失，…

有限状态机（4/5）

- 协议的有限状态机模型(S,M,I,T)
  - S:指协议机和信道可能的状态集合
  - M:能在信道上进行传输的帧（数据内容）的集合
  - T:转换的集合
  - I:系统的初始状态。即系统开始工作时的状态。
  - 有向图表示：状态为节点，转换为弧.称为状态转换图或可达性图
- 可达性分析确定一个协议的正确性
  - 图论技术考察哪些状态可达或不可达（如计算某图的传递闭包）
  - 某帧出现在某一系统状态时，有限状态机并没有指出如何动作，称协议规范是不完整的。
  - 有限状态机中存在某个状态子集，从这些状态出发既无出口，也不能再正确收发数据，称协议规范存在死锁现象。
  - 某个状态指名如何处理某个事件，而事件为不可能发生，称协议规范存在多余变迁。
  - ……

有限状态机（5/5）



Transition	Who runs?	Frame accepted	Frame emitted	To network layer
0	–	(frame lost)		–
1	R	0	A	Yes
2	S	A	1	–
3	R	1	A	Yes
4	S	A	0	–
5	R	0	A	No
6	R	1	A	No
7	S	(timeout)	0	–
8	S	(timeout)	1	–

(b)

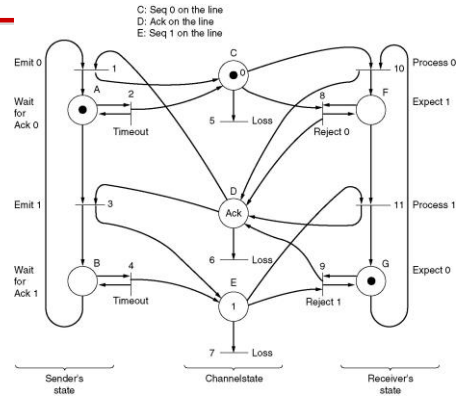
- 省略了不可达状态及发生校验和错误时的处理（因为这种帧不会改变协议3的系统状态,简化模型）
- 正确性分析： //结合提供的服务来做分析
  - 接收方永远不会递交两个连续奇（偶）数序列号的分组
  - 收发双方状态依次变化，不存在某个变了两次而另一个没变
  - 协议3无死锁

Petri网的图形表示

- 库所定义为收、发、信道状态，用圆圈表示
- Token，标记当前系统状态及资源个数，在圆圈内用多个点表示
- 转换=变迁，用短线表示。连接库所，表示其间的转换（I/O）关系
  - 若每个输入库所至少有一个Token，称该转换**随时**都可被激发
  - 激发后，输入库所资源数减1，输出库所资源数加1
  - 多个处于激活状态的变迁，**随时**都可被激发
    - Petri网适宜描述并发异步系统行为
- For example.

协议3的Petri网模型

- 没有复合状态，收、发、信道状态单独描述
- 检测协议的错误，类似有限状态机模型
  - 激发系列中，两次10转换间,无11转换发生.协议错误
  - …



Summary

- The data link layer is responsible for transmitting frames from sender to receiver.
- It can use only the physical layer, which supports only transmission of a bit (some bits) at a time.
- The DLL has to take into account that transmission errors may occur → error control (ACKs,NACKs, checksums, etc.)
- The DLL has to take into account that sender and receiver may operate at different speeds → flow control (windows, frame numbers, etc.)
- Protocol Verification
  - Finite State Machine Models & Petri Net Models

From Simplex to Duplex

- Problem: we want to allow symmetric frame transmission between two communicating parties, rather than transmission in one direction. Don't waste channels, so use the same channel for duplex communication.
- Solution: just transmit frames, but distinguish between data, acknowledgements (acks), and possibly negative acks (nacks) in the frame's type field.
- Idea: if the other party is going to send data as well, it might as well send acknowledgments along with its data frame →piggybacking
- Question: what's good and bad about piggybacking?

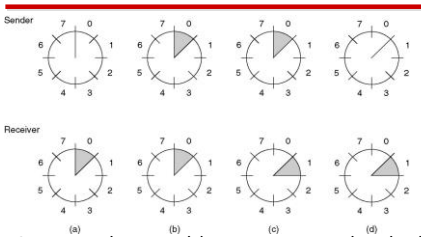
捎带确认

- 好处主要反映在提高通信效率
  - 单独帧的开销大,捎带只需占据帧头中的几位
  - 发送的帧越少,"帧到达" 中断也越少
- 问题主要反映在捎带时机
  - 即为了捎带一个确认,DLL应该等待要传分组多长时间?
    - 捎带者问题
    - 超过了发送方的超时周期,则该帧将会被重传
  - 策略
    - 等待固定毫秒数,没有待传分组就发送一个单独的确认帧

滑动窗口协议

- Principle: 不是单帧发送，而是可以一次发送若干帧。某时刻，允许发送的帧集合，称为发送窗口。
  - A frame从发送窗口中删除掉当且仅当it has been acknowledged.
  - 接收方的接收窗口表示允许接收的帧集合.
  - 用(循环使用的)frame sequence number($0\sim2^{n-1}$)标识帧
  - 若双方窗口大小为1，它就是stop-and-wait 协议
  - A damaged frames一直在接收窗口中，除非已正确收到。此外， frame #N is kept in the window until frame #N-1 has been received.

例： *Window size* =1， *sequence number=3-bit*



a: initially.
b: after sending frame #1
c: after receiving frame #1
d: after receiving ack for frame #1

Question: how would you interpret the shaded areas?

- A sliding window of size 1, with a 3-bit sequence number.
- 发送方:当有新的分组从网络层到来,被赋予最高的序列号且窗口的上边界加1; //阴影代表未被确认的帧
- 接收方:帧序列号等于窗口"最小号"时,交给网络层并回送确认,然后窗口上下边界加1; //阴影代表期待接收的帧

1-Bit sliding window(1/2)

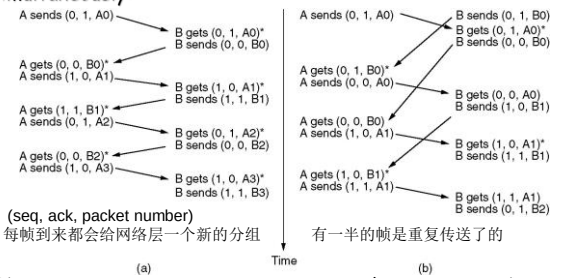
```
01 void protocol4 (void) {
02   seq_nr next_frame_to_send, frame_expected;
03   frame r, s;
04   packet buffer;
05   event_type event;
06
07   next_frame_to_send = 0; frame_expected = 0;
08   from_network_layer(&buffer);
09   s.info = buffer;
10   s.seq  = next_frame_to_send;
11   s.ack  = 1 - frame_expected;   //捎带占1位
12   to_physical_layer(&s); start_timer(s.seq);
13
```

```
14   while (true) {
15     wait_for_event(&event);
16     if (event == frame_arrival) {
17       from_physical_layer(&r);
18       if (r.seq == frame_expected){
19          to_network_layer(&r.info);
20          inc(frame _expected);
21       }
22       if (r.ack == next_frame_to_send){
23          from_network_layer(&buffer);
24          inc(next_frame_to_send);
25       }
26     }
27     s.info = buffer;
28     s.seq = next_frame_to_send;
29     s.ack = 1 - frame_expected;
30     to_physical_layer(&s); start_timer(s.seq);
31   }
32 }
```

1位滑动窗口停-等协议：窗口尺寸为1，发送方发送下一帧之前需等待前一帧的确认.
程序有问题吗？会发生帧未超时或错误，而再次发送的情况.

1-Bit Sliding Window(2/2)

- Observation: all things go well（如果B在发送自己的帧之前先等待A的第一帧）, but behavior is a bit strange when A and B transmit simultaneously:



- Observation: we are transmitting more than once, just because the two senders are more or less out of sync.

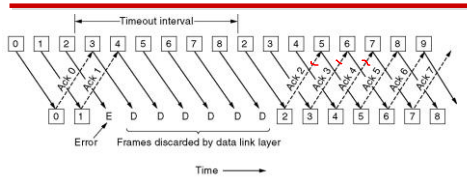
管道化技术

- 假设信道的容量是B位/秒，帧长为L位，往返传播时间为R秒（RTT），则：
  - 传输一帧时间是L/B秒；
  - 在停等协议中，发送方有L/B秒是忙的，而R秒是空歇的；
  - 所以，线路利用率=(L/B)/((L/B)+R)=L/(L+BR)
- 管道化技术
  - 允许发送方在阻塞之前发送多帧。
  - 适宜场景：RTT*B较大
  - 在**很长的帧流中，有一损坏或丢失，咋办？**

Error control（1/3）

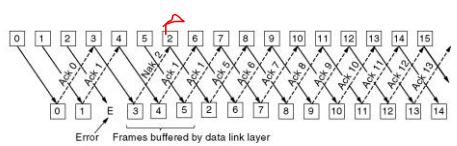
- Problem: what should the receiver do if a frame is damaged?
  - Simply request retransmission of all frames starting from #N. if any other frames had been received in the meantime, they'll just be ignored →go back n.
  - Request just retransmission of the damaged frame, and wait until it comes in before delivering any frames after that （顺序递交的需要）→selective repeat.

Error control（2/3）



为何不NAK?

(a) Go-back-n：就是对应接收方的窗口尺寸为1的情形。



为何不Time out?

(b) Selective repeat：对应于接收方的窗口尺寸大于1的情形

Error control（3/3）

- Go-back-N is really simple: the sender keeps a frame in its window until it is acknowledged.
  - If the window is full, the network layer is not allowed to submit new packets.
  - The receiver hardly needs to keep an account on what happens: if a frame is damaged, its successors in the receive window are ignored.
- Selective repeat seems to do better because frames aren't discarded, but the administration is much harder.
- 前者需要消耗带宽，后者需要高缓存空间支持。

Go-back-n（1/4）

```
/* Protocol 5 (pipelining) allows multiple outstanding frames. The sender may transmit up
   to MAX_SEQ frames without waiting for an ack. In addition, unlike the previous protocols,
   the network layer is not assumed to have a new packet all the time. Instead, the
   network layer causes a network_layer_ready event when there is a packet to send. */

#define MAX_SEQ 7                       /* should be 2^n – 1 */    n:帧头中占的位数
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
#include "protocol.h"                   //网络层希望发送一个分组时引发此事件

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Return true if a <=b < c circularly; false otherwise. */
  if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
      return(true);
    else
      return(false);
}

static void send_data(seq_nr frame_nr, seq_nr frame_expected, packet buffer[ ])
{
/* Construct and send a data frame. */
  frame s;                              /* scratch variable */

  s.info = buffer[frame_nr];            /* insert packet into frame */
  s.seq = frame_nr;                     /* insert sequence number into frame */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);/* piggyback ack */
  to_physical_layer(&s);                /* transmit the frame */
  start_timer(frame_nr);                /* start the timer running */
}
```

Go-back-n（2/4）

```
void protocol5(void)
{
  seq_nr next_frame_to_send;            /* MAX_SEQ > 1; used for outbound stream */
  seq_nr ack_expected;                  /* oldest frame as yet unacknowledged */
  seq_nr frame_expected;                /* next frame expected on inbound stream */
  frame r;                              /* scratch variable */
  packet buffer[MAX_SEQ + 1];           /* buffers for the outbound stream */
  seq_nr nbuffered;                     /* # output buffers currently in use */
  seq_nr i;                             /* used to index into the buffer array */
  event_type event;

  enable_network_layer();               /* allow network_layer_ready events */
  ack_expected = 0;                     /* next ack expected inbound */
  next_frame_to_send = 0;               /* next frame going out */
  frame_expected = 0;                   /* number of frame expected inbound */
  nbuffered = 0;                        /* initially no packets are buffered */
```

Go-back-n（3/4）

```
while (true) {
  wait_for_event(&event);               /* four possibilities: see event_type above */

  switch(event) {
    case network_layer_ready:           /* the network layer has a packet to send */
         /* Accept, save, and transmit a new frame. */
         from_network_layer(&buffer[next_frame_to_send]); /* fetch new packet */
         nbuffered = nbuffered + 1;     /* expand the sender's window */
         send_data(next_frame_to_send, frame_expected, buffer);/* transmit the frame */
         inc(next_frame_to_send);       /* advance sender's upper window edge */
         break;

    case frame_arrival:                 /* a data or control frame has arrived */
         from_physical_layer(&r);       /* get incoming frame from physical layer */

         if (r.seq == frame_expected) {
              /* Frames are accepted only in order. */
              to_network_layer(&r.info); /* pass packet to network layer */
              inc(frame_expected);      /* advance lower edge of receiver's window */
         }
```

Go-back-n（4/4）

```
        /* Ack n implies n – 1, n – 2, etc.  Check for this. */
        while (between(ack_expected, r.ack, next_frame_to_send)) {
            /* Handle piggybacked ack. */
            nbuffered = nbuffered – 1; /* one frame fewer buffered */
            stop_timer(ack_expected); /* frame arrived intact; stop timer */
            inc(ack_expected);      /* contract sender's window */
        }
        break;

    case cksum_err: break;          /* just ignore bad frames */

    case timeout:                   /* trouble; retransmit all outstanding frames */
        next_frame_to_send = ack_expected;    /* start retransmitting here */
        for (i = 1; i <= nbuffered; i++) {
            send_data(next_frame_to_send, frame_expected, buffer);/* resend 1 frame */
            inc(next_frame_to_send);  /* prepare to send the next one */
        }

  }

  if (nbuffered < MAX_SEQ)          //任何时候未确认的帧个数都不应该超过MAX_SEQ,这里控制
      enable_network_layer();       //了网络层的行为。
  else
      disable_network_layer();
 }
}
```

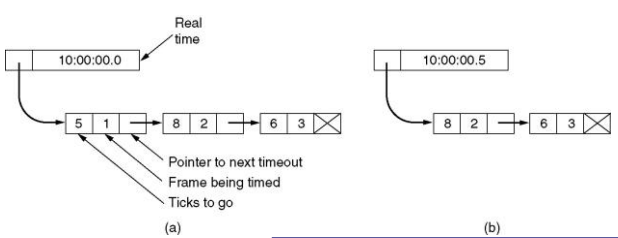
P1：为什么未确认帧的最大数量是MAX_SEQ,而不是MAX_SEQ+1?

- 即发送方最大窗口尺寸等于序列号个数-1？
  - 反证法：
    - The sender sends frames 0 through 7.
    - A piggybacked acknowledgement for frame 7 eventually comes back to the sender.
    - The sender sends another eight frames, again with sequence numbers 0 through 7.
    - Now another piggybacked acknowledgement for frame 7 comes in.

    第二批的8帧全部成功达到 还是 全部丢失？两种情况下，接收方都会发送第7帧的确认（一个是成功接收，另一个是确认重发），发送方无法辨认。
  - *有问题吗?*

P2：用软件模拟多个定时器

1)每一个未被确认的帧都需要一个定时器
2)链表中节点记载离过期还有多少个时钟嘀嗒（相对于前者）
3)假设每100ms时钟嘀嗒一次，(a)给出了3个超时事件*.00.5*.01.3*.01.9
4)Start_timer 和 Stop_timer需要查表。

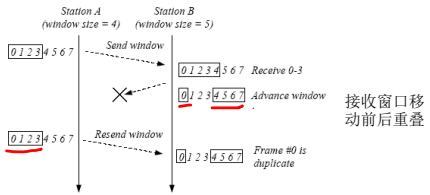


P3：单向流量问题

- 协议5中假设链路上总是有反向的流量可以稍带确认。若没有这样的流量，确认报文不会被送出且发送允许上限帧后协议停止。
  - 协议4没有这样的要求，每次收到一帧就送回一帧，即便是刚刚已经送出了这一帧(但会无用重发，浪费带宽)。
- 下一协议
  - 当一个按正常次序发送的数据帧到达之后，接收方通过start-ack-timer启动一个辅助的定时器。若在定期器到期之前没有反向流量，则发送一个单独的确认帧。

Selective repeat – the problems

- Frames need not be received in order, i.e. we may have an undamaged frame #N, while still waiting for an undamaged version of #N-1.
- If the receiver delivers all frames in its window just after sending an ack for the entire window, we may have a serious problem:



接收窗口移动前后重叠

- Solution: we must avoid overlapping send and receive windows → the highest sequence number must be at least twice the window size.

Example: Selective Repeat (1/3)

```
00  static boolean between(seq_nr a, seq_nr b, seq_nr c) {...}
01
02  static void send_frame(frame_kind fk, seq_nr frame_nr,
03        seq_nr frame_expected, packet buffer[]){
04    frame s;  s.kind = fk;
05    if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
06    s.seq = frame_nr;
07    s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
08    if (fk == nak) no_nak = false;
09    to_physical_layer(&s);
10    if (fk == data) start_timer(frame_nr % NR_BUFS);
11    stop_ack_timer();
12  }
13
14  void protocol6(void){
15    seq_nr ack_expected, next_frame_to_send, frame_expected;
16    seq_nr nbuffered, too_far; event_type event;
17    int i; frame r;
18    packet out_buf[NR_BUFS], in_buf[NR_BUFS];
19    boolean arrived[NR_BUFS];
20
21    enable_network_layer();
22    ack_expected = 0; next_frame_to_send = 0; frame_expected = 0;
23    too_far = NR_BUFS; nbuffered = 0;
24    for (i = 0; i < NR_BUFS; i++) arrived[i] = false;
25    while (true) {
26      wait_for_event(&event);
27      switch(event) {
28      case network_layer_ready:
29        nbuffered = nbuffered + 1;
30        from_network_layer(&out_buf[next_frame_to_send % NR_BUFS]);
31        send_frame(data,next_frame_to_send,frame_expected,out_buf);
32        inc(next_frame_to_send);
33        break;
```

Example: Selective Repeat (2/3)

```
35      case frame_arrival:
36        from_physical_layer(&r);
37        if (r.kind == data) {
38          if ((r.seq != frame_expected) && no_nak)
39            send_frame(nak, 0, frame_expected, out_buf);
40          else start_ack_timer();
41          if (between(frame_expected, r.seq, too_far) &&
42              (arrived[r.seq % NR_BUFS] == false)) {
43            arrived[r.seq % NR_BUFS] = true;
44            in_buf[r.seq % NR_BUFS] = r.info;
45            while (arrived[frame_expected % NR_BUFS]) {
46              to_network_layer(&in_buf[frame_expected % NR_BUFS]);
47              no_nak = true;
48              arrived[frame_expected % NR_BUFS] = false;
49              inc(frame_expected);
50              inc(too_far);
51              start_ack_timer();
52            }
53          }
54        }
55        if((r.kind == nak) &&
56           between(ack_expected, (r.ack+1) % (MAX_SEQ+1),
57                   next_frame_to_send))
58          send_frame(data, (r.ack+1) % (MAX_SEQ + 1),
59                     frame_expected,out_buf);
60
61        while (between(ack_expected, r.ack, next_frame_to_send)) {
62          nbuffered = nbuffered - 1;
63          stop_timer(ack_expected % NR_BUFS);
64          inc(ack_expected);
65        }
66        break;
```

Example: Selective Repeat (3/3)

```
68      case cksum_err:
69        if (no_nak) send_frame(nak, 0, frame_expected, out_buf);
70        break;
71
72      case timeout:
73        send_frame(data, oldest_frame, frame_expected, out_buf);
74        break;
75
76      case ack_timeout:            //辅助定时器
77        send_frame(ack,0,frame_expected, out_buf);
78      }
79      if (nbuffered < NR_BUFS) enable_network_layer();
80      else disable_network_layer();
81    }
82  }
```

分析：

- 所需要的缓冲区数量等于窗口尺寸
- 所需要的定时器数量等于缓冲区数量
- 只需一个辅助定时器，若运行时，strat-ack-timer再次被调用的话，它将被重置为一个完整的确认超时间隔。
  - 该间隔必须小于数据帧重发超时间隔。
- 需要复杂的定时器管理，比如：确定哪一帧引发了超时。

Data link layer protocols

Now let's take a look how point-to-point connections are supported in, for example, the Internet.

Recall:

- The data link layer is responsible for transmitting frames from sender to receiver.
- It can use only the physical layer, which supports only transmission of a bit (some bits) at a time.
- The DLL has to take into account that transmission errors may occur → error control (ACKs, NACKs, checksums, etc.)
- The DLL has to take into account that sender and receiver may operate at different speeds → flow control (windows, frame numbers, etc.)

High-Level Data Link Control（Protocol)

HDLC: a pretty old, but widely used protocol for point-to-point connections. Is bit-oriented.//位填充保证数据的透明性

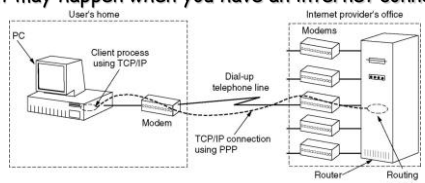
Bits	8	8	8	≥0	16	8
	01111110	Address	Control	Data	Checksum	01111110

Question: what do we need the address field for?
P2P线路用来区分应答和命令;广播线路区分终端.

- HDLC uses a sliding window protocol with 3-bit sequencing
  - Go-back n and Selective repeat 都能支持, control里面有类型域
- The control field is used to distinguish different kinds of frames:
  - the control field contains sequence numbers, acks, nacks, etc.

Internet Point-to-Point connections

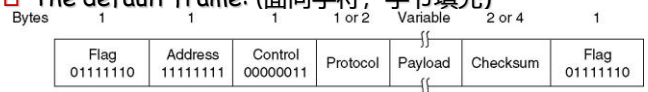
- This is what may happen when you have an internet connection to a provider:



- Problem: One way or the other we'd like to use the internet protocol stack at our home. The bottom line is that we'll have to transfer IP (network) packets across our telephone line. Note: we may also be talking ADSL here! //与物理层技术无关性
- Issue: How can we (1) embed IP Packets into frames, that can be (2) unpacked at the other end, to be handed to the network layer?

PPP: Point-to-Point Protocol

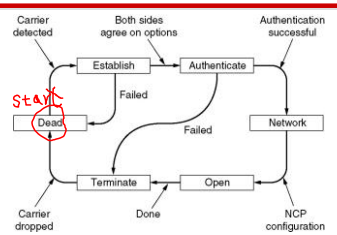
- PPP is the data link protocol for Point-to-point connections with respect to the Internet (home-ISP and router-to-router):
  - Proper framing, i.e. the start and end of a frame can be unambiguously detected. 以及差错控制和其他DLL功能等。
  - Has a separate protocol for controlling the line (setup, testing, negotiating options, and tear-down) ($Link Control Protocol$), LCP
  - Supports many different network layer protocols, not just IP.
    - Network Control Protocol, NCP
    - There's no need for fixed network addresses.
- The default frame: (面向字符，字节填充)



PPP-Example（1/3）

- Suppose you want to set up a true Internet connection to your provider.
  - Set up a physical connection through your modem.
    - Pc通过Modem呼叫provider且收到回答，建立物理连接
  - Your Pc starts sending a number of Link Control Packets (LCP) to negotiate the kind of PPP connection you want. Note that these packets are embedded in PPP frames:
    - The maximum payload size in data frames
    - Do authentication (e.g. ask for a password)
    - Monitor the quallity of the link (e.g. how many frames didn't come through).
    - Compress headers (useful for slow links between fast computers)
  - Then, we negotiate network layer stuff, like getting an IP address that the provider's router can use to forward packets to you. // NCP, network control protocol,破坏了层间关系。

PPP-Example（2/3）线路状态转换图



A simplified phase diagram for bring a line up and down

- If an IP address is dynamically assigned ,who does the assignment? The provider.
- If an IP address is dynamically assigned, can someone else ever send you data (they don't know your address, do they? We need to contact them first (our address is included in the request).

PPP-Example（3/3）

Name	Direction	Description
Configure-request	I → R	List of proposed options and values
Configure-ack	I ← R	All options are accepted
Configure-nak	I ← R	Some options are not accepted
Configure-reject	I ← R	Some options are not negotiable
Terminate-request	I → R	Request to shut the line down
Terminate-ack	I ← R	OK, line shut down
Code-reject	I ← R	Unknown request received
Protocol-reject	I ← R	Unknown protocol requested
Echo-request	I → R	Please send this frame back
Echo-reply	I ← R	Here is the frame back
Discard-request	I → R	Just discard this frame (for testing)

The LCP frame types, I:建议方，R:应答方

Summary

- 小结
  - 成帧的方法：字符计数、字节填充和位填充；
  - DLL提供差错控制功能，以便重传丢失或损坏帧；
  - 滑动窗口机制是一种广泛应用的流控技术；
    - 回退n帧协议　和　选择性重发
  - 常用两种协议建模技术,分析协议正确性
    - Finite State Machine Models & Petri Net Models
  - DLL协议例：HDLC 和 PPP
    - Internet使用PPP作为Point-to-Point线路上的基本数据链路协议
    - 前者使用滑动窗口机制实现流控.