THE DATA LINK LAYER(後半)

第4回　輪講-COMPUTER NETWORKS-

岡部研究室　M2　松本和馬

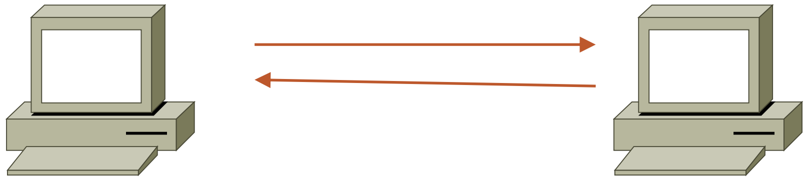
2015年6月11日(木)

Contents

SLIDING WINDOW PROTOCOLS

前に説明したプロトコル(A Utopian Simplex Protocol, A Simplex Stop & Wait Protocol)では単一方向通信を用いていたが、実際に使われるようなプロトコルでは双方向通信が必要

Separate link でデータ送信用チャンネルと ack 用チャンネルを作る方法（プロトコル２，３）
→ack 用チャンネルでのフレームの容量ほぼすべてが無駄になる

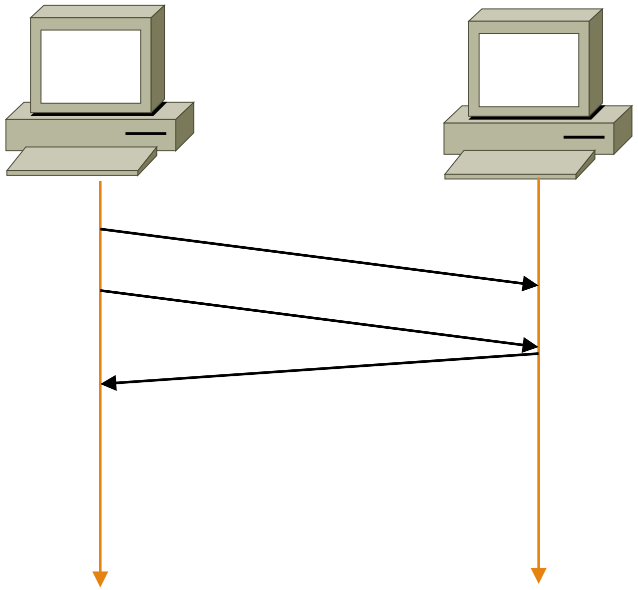


Same link での双方向通信；フレームのヘッダーフィールドを使って実現

Piggybacking

今まで:送信者からのデータフレームが受信者に届いた瞬間に ack を即返信

Piggybacking:受信者はデータフレームが届いても ack の返信を待機、次のフレームが届いた後の ack で二つ一緒に返信



利点
別々のフレームで全ての ack を行うよりもチャンネル帯域を効率的に利用可能
Piggybacking のコストは大抵数ビット以下

問題点
Separate Frame よりも複雑
待機時間をどう設定すべきか
もし待機時間まで次のフレームが来なかったら今までのシステム以下の効率

Sliding Window

送信フレームに $2^{n}-1$ (nはビットフィールド)が最大値となる sequence number を付ける

送信側：Sending Window で許可されている number まで送信

受信側：Receiving Window 受信できるフレームの number を決定

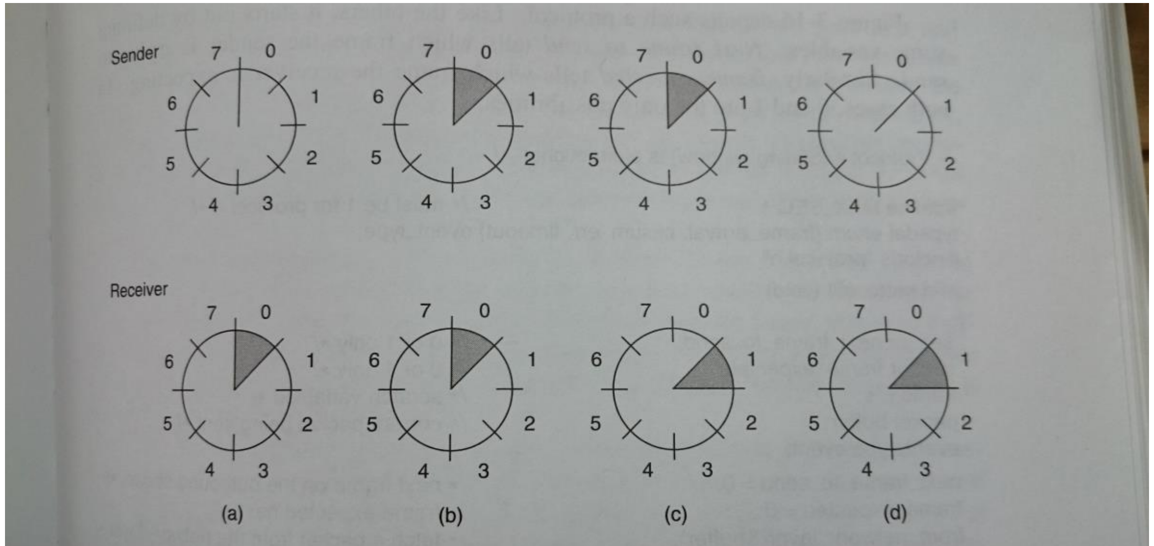


Figure 3-15. A sliding window of size 1, with a 3-bit sequence number. (a) Initially. (b) After the first frame has been sent. (c) After the first frame has been received. (d) After the first acknowledgement has been received.

A One-Bit Sliding Window Protocol

最初にサイズが1のものを考える

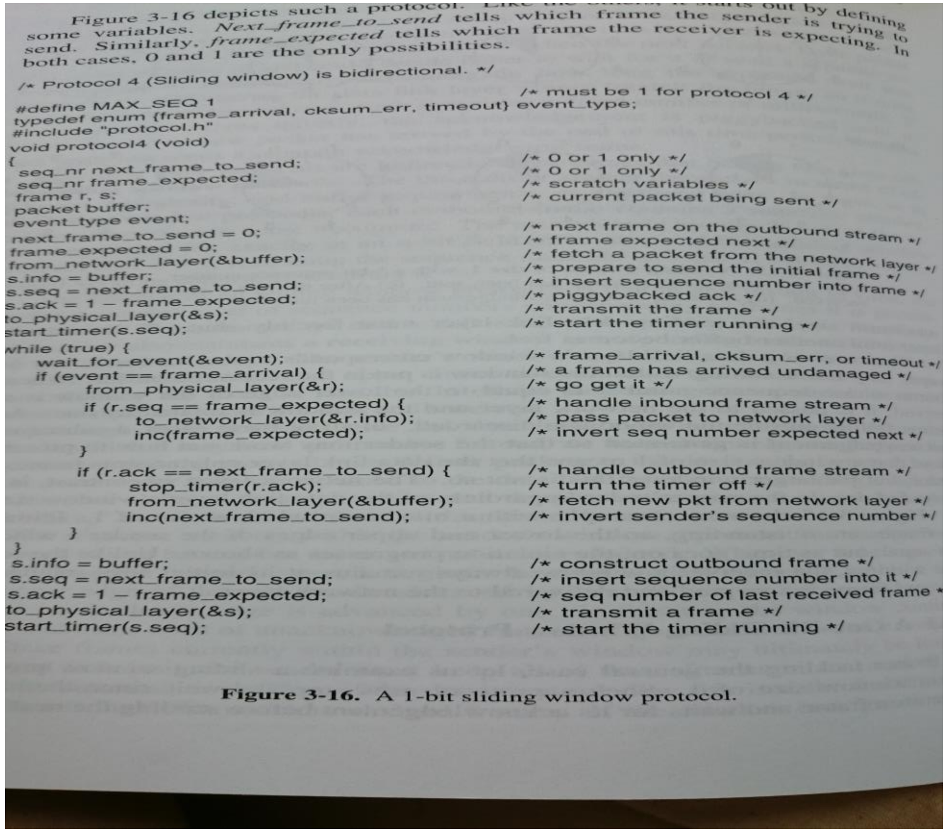


Figure 3-16 depicts such a protocol. Like the others, it starts out by defining some variables. *Next_frame_to_send* tells which frame the sender is trying to send. Similarly, *frame_expected* tells which frame the receiver is expecting. In both cases, 0 and 1 are the only possibilities.

```
/* Protocol 4 (Sliding window) is bidirectional. */

#define MAX_SEQ 1                                   /* must be 1 for protocol 4 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"
void protocol4 (void)
{
 seq_nr next_frame_to_send;                         /* 0 or 1 only */
 seq_nr frame_expected;                             /* 0 or 1 only */
 frame r, s;                                        /* scratch variables */
 packet buffer;                                     /* current packet being sent */
 event_type event;

 next_frame_to_send = 0;                            /* next frame on the outbound stream */
 frame_expected = 0;                                /* frame expected next */
 from_network_layer(&buffer);                       /* fetch a packet from the network layer */
 s.info = buffer;                                   /* prepare to send the initial frame */
 s.seq = next_frame_to_send;                        /* insert sequence number into frame */
 s.ack = 1 – frame_expected;                        /* piggybacked ack */
 to_physical_layer(&s);                             /* transmit the frame */
 start_timer(s.seq);                                /* start the timer running */

 while (true) {
     wait_for_event(&event);                        /* frame_arrival, cksum_err, or timeout */
     if (event == frame_arrival) {                  /* a frame has arrived undamaged */
          from_physical_layer(&r);                  /* go get it */
          if (r.seq == frame_expected) {            /* handle inbound frame stream */
               to_network_layer(&r.info);           /* pass packet to network layer */
               inc(frame_expected);                 /* invert seq number expected next */
          }
          if (r.ack == next_frame_to_send) {        /* handle outbound frame stream */
               stop_timer(r.ack);                   /* turn the timer off */
               from_network_layer(&buffer);         /* fetch new pkt from network layer */
               inc(next_frame_to_send);             /* invert sender's sequence number */
          }
     }
     s.info = buffer;                               /* construct outbound frame */
     s.seq = next_frame_to_send;                    /* insert sequence number into it */
     s.ack = 1 – frame_expected;                    /* seq number of last received frame */
     to_physical_layer(&s);                         /* transmit a frame */
     start_timer(s.seq);                            /* start the timer running */
 }
}
```

Figure 3-16. A 1-bit sliding window protocol.

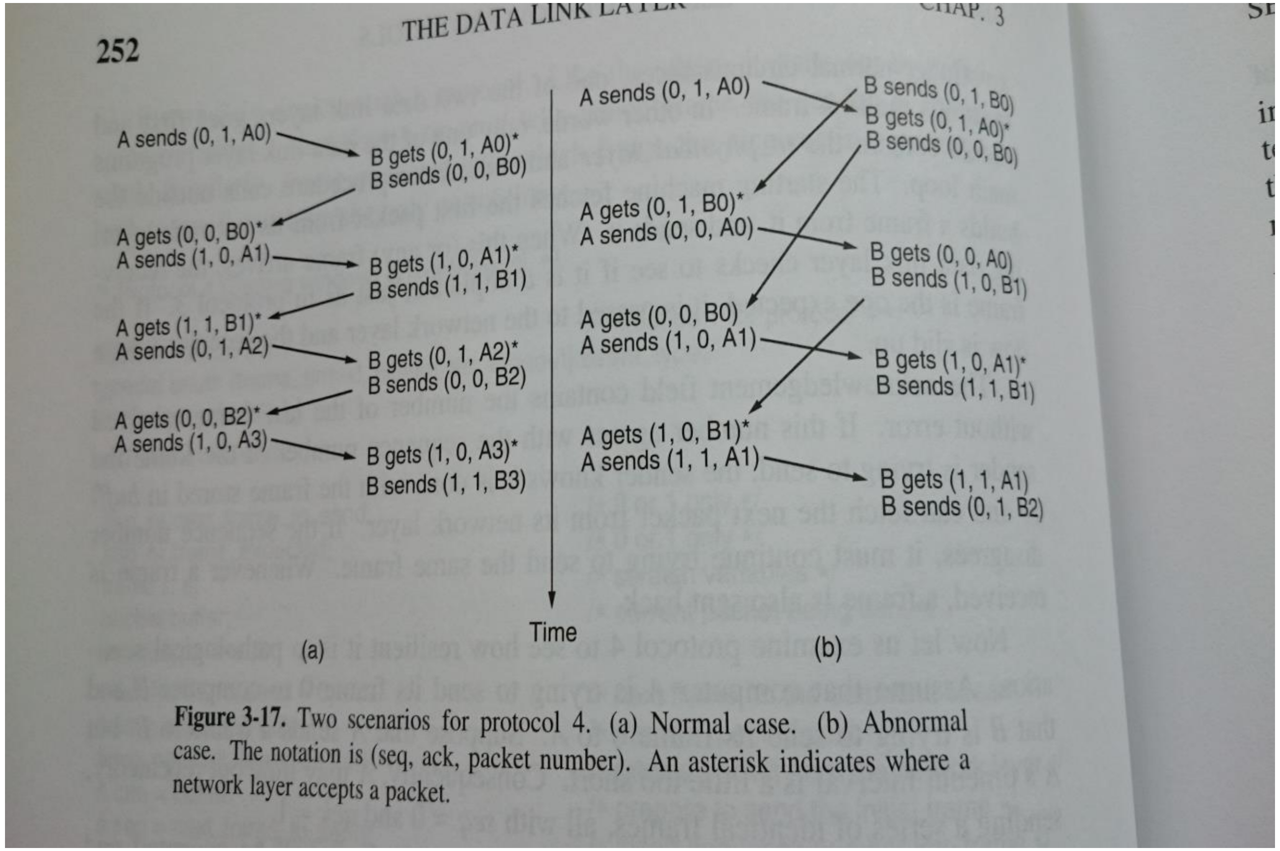
Sequece number 0 or 1

S.Seq = next_frame_send

S.ack = 1 – frame_expected

```
While(true){
        if(frame_arrival){
                if(r.seq==frame_expected){}
                if(r.ack==next_frame_to_send){}
        }
}
```

Two Scenarios Protocol 4



送信、受信のタイミングが想定と
ずれてもエラーが起こることなく通信
は行われる

A Protocol Using Go-Back-N

一つずつフレームの送信と返信を繰り返していると、ものすごい帯域幅の無駄使い
ウィンドウサイズを大きくして、帯域幅の使用率の効率化
Go-Back-NのNはウィンドウサイズの意味
Bandwidth-delay product(BDP) = $\frac{bits}{sec} * t(one\ way\ transmit\ time)$

Sending Window と同じ数までフレームを送信
Receiving window と同じ数までフレームを受信、しかしフレーム順序が乱れたら無視

```
/* Protocol 5 (Go-back-n) allows multiple outstanding frames. The sender may transmit up
   to MAX_SEQ frames without waiting for an ack. In addition, unlike in 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
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
#include "protocol.h"

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 */
}

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;                       /* number of 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 */

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

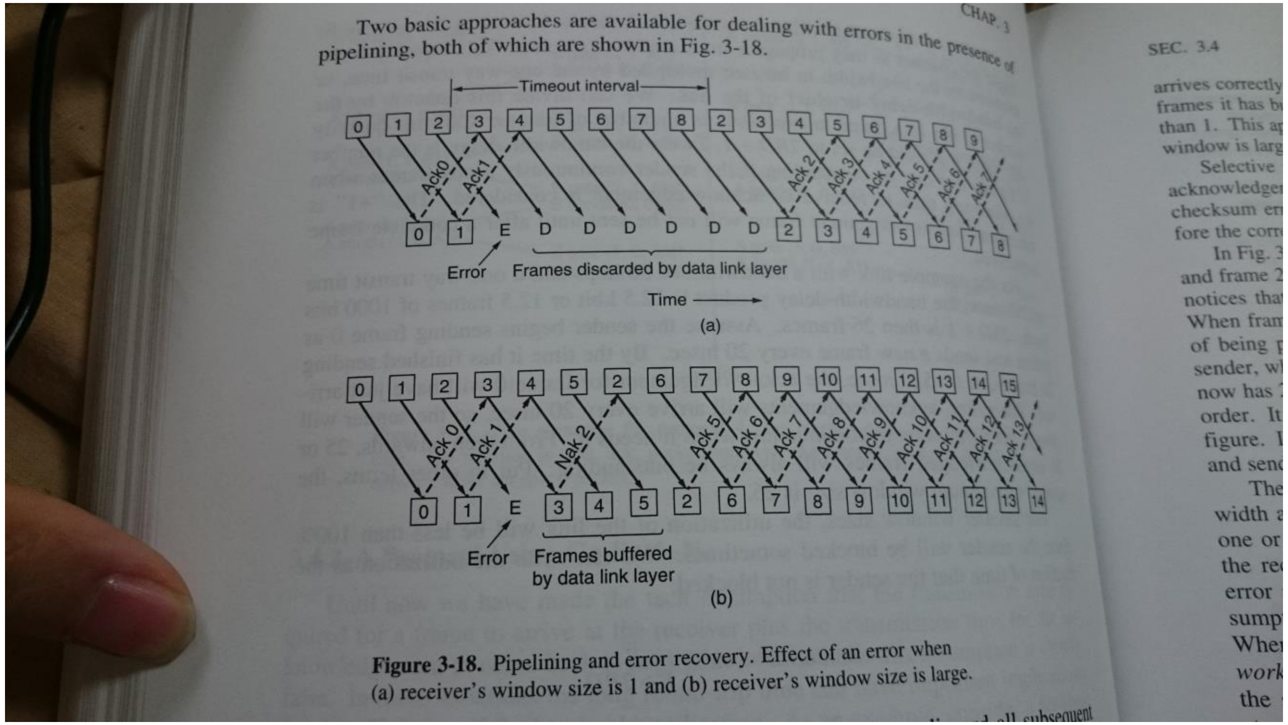
```
switch(event) {
  case network_layer_ready:                          /* the network layer has a packet to send */
       /* Accept, save, and transmit a new frame. */
       from_network_layer(&buffer[next_frame_to_send]); /* fetch new packet */
       nbuffered = nbuffered + 1;                    /* 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
       }

       /* 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 fran
       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 fram
             inc(next_frame_to_send);                /* prepare to send the next one */
       }
 }

 if (nbuffered < MAX_SEQ)
       enable_network_layer();
 else
       disable_network_layer();
 }
}
```

Figure 3-19. A sliding window protocol using go-back-n.

The question is this: did all eight frames belonging to the second batch ar cessfully, or did all eight get lost (counting discards following an error In both cases the receiver would be sending frame 7 as the acknowle

Pipelining

送信のデータがエラーで破壊されたときどのようにフォローが行われるか



(a)receving window size = 1
正常に送られたはずの4,5,6,7,8も捨てられている

(b)Receiving window size = large
2のみが再送される結果となった

バッファが存在することによってエラー処理の効率が良くなった

A Protocol Using Selective Repeat

Go-back-n との違いは受信で一部のデータでエラーがあっても受信をやめないこと

その後、ack でエラーの在ったseq number を送信側に伝えて再送要求を行う

Sending window と　receiving window のサイズを合わせる必要がある

また、ウィンドウサイズ＜＝(Max_SEQ+1)/2

Receiving window の状態がack によって伝えられsending window　は変化する

EXAMPLE DATA LINK PROTOCOLS

Packet over SONET

ADSL
(Asymmetric Digital Subscriber Loop)

SUMMARY