La couche de liaison

Le maintien du contact à deux partenaires

(sous-couche LLC)

Luc Lavoie
Département d'informatique
Faculté des sciences

luc.lavoie@USherbrooke.ca

Fonctions offertes

- Fournir une interface à la couche réseau
- Traiter les erreurs de transmission
- Réguler le flux

Services offerts

- sans connexion
  - sans accusé de réception
  - avec accusé de réception
- avec connexion
  - (donc) avec accusé de réception

Trame

- En-tête (header) et en-queue (trailer)
- Redondance
- Adressage local

Délimitation des trames

- Plusieurs techniques, dont
  - Compter les bits ou les octets
  - Utiliser des marqueurs
    - (donc) remplissage de bits
    - (donc) remplissage d'octets
  - Enfreindre le protocole physique
  - Synchroniser les partenaires (horloge commune)

Traitement des erreurs

- Corruption
- Synchronisation
- Temporisation (délai d'attente)

Régulation du flux

Situations :

- congestion
- débordement

Méthodes :

- rétroaction
- variation de débit

Présentation évolutive d'un protocole de liaison PàP (sans prise en compte des adresses physiques)

- P0 – Définitions communes
- P1 – Protocole unidirectionnel simple
- P2 – Protocole unidirectionnel avec confirmation
- P3 – Protocole unidirectionnel avec confirmation et reprise

P0a

```
1.  #define MAX_PKT 1024                                  /* determines packet size in bytes */
2.  typedef enum {false, true} boolean;                   /* boolean type */
3.  typedef unsigned int seq_nr;                          /* sequence or ack numbers */
4.  typedef struct {unsigned char data[MAX_PKT];} packet;     /*    packet definition */
5.  typedef enum {data, ack, nak} frame_kind;             /* frame_kind definition */

6.  typedef struct {    /* frames are transported in this layer */
7.    frame_kind kind;              /* what kind of a frame is it? */
8.    seq_nr seq;                   /* sequence number */
9.    seq_nr ack;                   /* acknowledgement number */
10.   packet info;                  /* the network layer packet */
11. } frame;

12. typedef enum {
13.         frame_arrival,
14.         cksum_err,
15.         timeout,
16.         network_layer_ready,
17.         ack_timeout}
18.         event_type;
```


P0b

```
1.  /* Wait for an event to happen; return its type in
    event. */
2.  void wait_for_event(event_type *event);

3.  /* Fetch a packet from the network layer for
    transmission on the channel. */
4.  void from_network_layer(packet *p);

5.  /* Deliver information from an inbound frame to the
    network layer. */
6.  void to_network_layer(packet *p);

7.  /* Go get an inbound frame from the physical layer
    and copy it to r. */
8.  void from_physical_layer(frame *r);

9.  /* Pass the frame to the physical layer for
    transmission. */
10. void to_physical_layer(frame *s);

11. /* Start the clock running and enable the timeout
    event. */
12. void start_timer(seq_nr k);

13. /* Stop the clock and disable the timeout event. */
14. void stop_timer(seq_nr k);
```

```
1.  /* Start an auxiliary timer and enable the
    ack_timeout event. */
2.  void start_ack_timer(void);

3.  /* Stop the auxiliary timer and disable the
    ack_timeout event. */
4.  void stop_ack_timer(void);

5.  /* Allow the network layer to cause a
    network_layer_ready event. */
6.  void enable_network_layer(void);

7.  /* Forbid the network layer from causing a
    network_layer_ready event. */
8.  void disable_network_layer(void);

9.  /* Macro inc is expanded in-line: Increment k
    circularly. */
10. #define inc(k) if (k<MAX_SEQ) k = k+1; else k = 0
```

Protocole unidirectionnel simple

Protocole 1, fig. 3.10

- Ce protocole n'assure que le transfert de l'émetteur au récepteur.
- Ne prend pas en compte :
  - Capacité de réception (traitement, stockage).
  - Possibilité d'erreur de transmission (perte, corruption).

Critique

- Solution non réaliste sauf dans un milieu sans perturbation aucune.

P1

```
1.  #include "protocol.h"
2.  void sender1(void)
3.  {
4.    frame s;                  /* buffer for an outbound frame */
5.    packet buffer;            /* buffer for an outbound packet */
6.    while (true) {
7.        from_network_layer(&buffer);     /* go get something to send */
8.        s.info = buffer;                 /* copy it into s for transmission */
9.        to_physical_layer(&s);           /* send it on its way */
10.   }
11. }

12. void receiver1(void)
13. {
14.   frame r;
15.   event_type event;         /* filled in by wait, but not used here */
16.   while (true) {
17.       wait_for_event(&event);          /* only possibility is frame_arrival */
18.       from_physical_layer(&r);         /* go get the inbound frame */
19.       to_network_layer(&r.info);       /* pass the data to the network layer */
20.   }
21. }
```

Protocole unidirectionnel avec confirmation

Protocole 2, fig. 3.11

- Ce protocole n'assure que le transfert de l'émetteur au récepteur.
- Il prend en compte :
  - Capacité de réception (en exigeant une confirmation explicite avant d'envoyer une trame).
- Ne prend pas à compte :
  - Possibilité d'erreur de transmission (perte, corruption).

Critique

- Très sensible à la perte (corruption) de l'accusé de réception.

P2

```
1.  void sender2(void)
2.  {
3.    frame s;                    /* buffer for an outbound frame */
4.    packet buffer;              /* buffer for an outbound packet */
5.    event_type event;           /* frame_arrival is the only possibility */
6.    while (true) {
7.        from_network_layer(&buffer); /* go get something to send */
8.        s.info = buffer;        /* copy it into s for transmission */
9.        to_physical_layer(&s);  /* bye-bye little frame */
10.       wait_for_event(&event); /* do not proceed until given the go ahead */
11.   }
12. }

13. void receiver2(void)
14. {
15.   frame r, s;                 /* buffers for frames */
16.   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. }
```

Protocole unidirectionnel avec confirmation et reprise

Protocole 3, fig. 3.12

- Ce protocole n'assure que le transfert de l'émetteur au récepteur.
- Il prend en compte :
  - Capacité de réception (en exigeant une confirmation explicite avant d'envoyer une trame).
  - Possibilité d'erreur de transmission (en redemandant le renvoi de la dernière trame reçue si elle est corrompue).

Critique

- Faible utilisation de la bande passante
- Très sensible à la perte de synchronisation

P3a

```
1.  #define MAX_SEQ 1                       /* must be 1 for protocol 3 */
2.  void sender3(void)
3.  {
4.    seq_nr next_frame_to_send;            /* seq number of next outgoing frame */
5.    frame s;                              /* scratch variable */
6.    packet buffer;                        /* buffer for an outbound packet */
7.    event_type event;
8.    next_frame_to_send = 0;               /* initialize outbound sequence numbers */
9.    from_network_layer(&buffer);          /* fetch first packet */
10.   while (true) {
11.       s.info = buffer;                  /* construct a frame for transmission */
12.       s.seq = next_frame_to_send;       /* insert sequence number in frame */
13.       to_physical_layer(&s);            /* send it on its way */
14.       start_timer(s.seq);               /* if answer takes too long, time out */
15.       wait_for_event(&event);           /* frame_arrival, cksum_err, timeout */
16.       if (event == frame_arrival) {
17.           from_physical_layer(&s);      /* get the acknowledgement */
18.           if (s.ack == next_frame_to_send) {
19.               stop_timer(s.ack);        /* turn the timer off */
20.               from_network_layer(&buffer); /* get the next one to send */
21.               inc(next_frame_to_send);  /* invert next_frame_to_send */
22.           }
23.       }
24.   }
25. }
```

P3b

```
1.   void receiver3(void)
2.   {
3.     seq_nr frame_expected;
4.     frame r, s;
5.     event_type event;
6.     frame_expected = 0;
7.     while (true) {
8.         wait_for_event(&event);                 /* possibilities: frame_arrival, cksum_err */
9.         if (event == frame_arrival) {           /* a valid frame has arrived. */
10.            from_physical_layer(&r);            /* go get the newly arrived frame */
11.            if (r.seq == frame_expected) {      /* this is what we have been waiting for. */
12.                to_network_layer(&r.info);      /* pass the data to the network layer */
13.                inc(frame_expected);            /* next time expect the other sequence nr */
14.            }
15.            s.ack = 1 - frame_expected;         /* tell which frame is being acked */
16.            to_physical_layer(&s);              /* send acknowledgement */
17.        }
18.    }
19.  }
```


Vers une amélioration

- Rendre la communication bidirectionnelle afin de profiter des « ack » pour transmettre des informations (« piggybacking »).
- Anticiper le bon déroulement du transfert mais en conservant la capacité de retransmettre, par la conservation, de part et d'autre, de tampons circulaires (fenêtres coulissantes) de façon à anticiper l'arrivée des « ack » (anticipation optimiste).

Protocoles à fenêtres

- Protocole bidirectionnel à tampon unique (protocole 4, fig. 3.14)
- Protocole bidirectionnel à tampons multiples et reprise unique (protocole 5, fig. 3.17)
- Protocole bidirectionnel à tampons multiples et reprises multiples et sélectives (protocole 6, fig. 3.19)

P5a

```
1.  #define MAX_SEQ 7                    /* should be 2^n - 1 */

2.  static boolean between(seq_nr a, seq_nr b, seq_nr c)
3.  {
4.  /* Return true if a <=b < c circularly; false otherwise. */
5.    if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
6.          return(true);
7.      else
8.          return(false);
9.  }

10. static void send_data(seq_nr frame_nr, seq_nr frame_expected, packet buffer[])
11. {
12. /* Construct and send a data frame. */
13.   frame s;                    /* scratch variable */
14.   s.info = buffer[frame_nr];  /* insert packet into frame */
15.   s.seq = frame_nr;           /* insert sequence number into frame */
16.   s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);     /* piggyback ack */
17.   to_physical_layer(&s);      /* transmit the frame */
18.   start_timer(frame_nr);      /* start the timer running */
19. }
```

P5b

```
1.  void protocol5(void)
2.  {
3.    seq_nr next_frame_to_send;    /* MAX_SEQ > 1; used for outbound stream */
4.    seq_nr ack_expected;          /* oldest frame as yet unacknowledged */
5.    seq_nr frame_expected;        /* next frame expected on inbound stream */
6.    frame r;                      /* scratch variable */
7.    packet buffer[MAX_SEQ + 1];   /* buffers for the outbound stream */
8.    seq_nr nbuffered;             /* # output buffers currently in use */
9.    seq_nr i;                     /* used to index into the buffer array */
10.   event_type event;
11.   enable_network_layer();       /* allow network_layer_ready events */
12.   ack_expected = 0;             /* next ack expected inbound */
13.   next_frame_to_send = 0;       /* next frame going out */
14.   frame_expected = 0;           /* number of frame expected inbound */
15.   nbuffered = 0;                /* initially no packets are buffered */
16.   while (true) {
17.     wait_for_event(&event);     /* four possibilities: see event_type above */
18.     /* voir 5c */
19.     if (nbuffered < MAX_SEQ)
20.           enable_network_layer();
21.     else
22.           disable_network_layer();
23.   }
24. }
```


P5c

```
1.  switch(event) {
2.      case network_layer_ready:                       /* the network layer has a packet to send */
3.          /* Accept, save, and transmit a new frame. */
4.          from_network_layer(&buffer[next_frame_to_send]); /* fetch new packet */
5.          nbuffered = nbuffered + 1;                  /* expand the sender's window */
6.          send_data(next_frame_to_send, frame_expected, buffer);  /* transmit the frame */
7.          inc(next_frame_to_send);                    /* advance sender's upper window edge */
8.          break;
9.
10.     case frame_arrival:         /* a data or control frame has arrived */
11.         /* voir 5d */
12.         break;
13.     case cksum_err:             /* just ignore bad frames */
14.         break;
15.
16.     case timeout:               /* trouble; retransmit all outstanding frames */
17.         next_frame_to_send = ack_expected;          /* start retransmitting here */
18.         for (i = 1; i <= nbuffered; i++) {
19.             send_data(next_frame_to_send, frame_expected, buffer);  /* resend 1 frame */
20.             inc(next_frame_to_send);                                /* prepare to send the next one */
21.         }
22.   }
```

P5d

```
1.  case frame_arrival:
2.      from_physical_layer(&r);      /* get incoming frame from physical layer */
3.
4.      if (r.seq == frame_expected) {
5.          /* Frames are accepted only in order. */
6.          to_network_layer(&r.info); /* pass packet to network layer */
7.          inc(frame_expected); /* advance lower edge of receiver's window */
8.        }
9.
10.     /* Ack n implies n - 1, n - 2, etc.  Check for this. */
11.     while (between(ack_expected, r.ack, next_frame_to_send)) {
12.         /* Handle piggybacked ack. */
13.         nbuffered = nbuffered - 1;          /* one frame fewer buffered */
14.         stop_timer(ack_expected);           /* frame arrived intact; stop timer */
15.         inc(ack_expected);                  /* contract sender's window */
16.     }
```


P6a

```
1.  static boolean between(seq_nr a, seq_nr b, seq_nr c)
2.  {
3.  /* Same as between in protocol5, but shorter and more obscure. */
4.    return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
5.  }
6.  static void send_frame(frame_kind fk, seq_nr frame_nr, seq_nr frame_expected,
    packet buffer[])
7.  {
8.  /* Construct and send a data, ack, or nak frame. */
9.    frame s;                              /* scratch variable */
10.   s.kind = fk;                          /* kind == data, ack, or nak */
11.   if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
12.   s.seq = frame_nr;                     /* only meaningful for data frames */
13.   s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
14.   if (fk == nak) no_nak = false;        /* one nak per frame, please */
15.   to_physical_layer(&s);                /* transmit the frame */
16.   if (fk == data) start_timer(frame_nr % NR_BUFS);
17.   stop_ack_timer();                     /* no need for separate ack frame */
18. }
```

P6b

```
1.  /* Protocol 6 (selective repeat) accepts frames out of order but passes packets to the
2.     network layer in order. Associated with each outstanding frame is a timer. When the timer
3.     expires, only that frame is retransmitted, not all the outstanding frames, as in protocol 5. */
4.  #define MAX_SEQ 7                       /* should be 2^n - 1 */
5.  #define NR_BUFS ((MAX_SEQ + 1)/2)
6.  #include "protocol.h"
7.  boolean no_nak = true;                  /* no nak has been sent yet */
8.  void protocol6(void)
9.  {
10.   seq_nr ack_expected;          /* lower edge of sender's window */
11.   seq_nr next_frame_to_send;    /* upper edge of sender's window + 1 */
12.   seq_nr frame_expected;        /* lower edge of receiver's window */
13.   seq_nr too_far;               /* upper edge of receiver's window + 1 */
14.   int i;                        /* index into buffer pool */
15.   frame r;                      /* scratch variable */
16.   packet out_buf[NR_BUFS];      /* buffers for the outbound stream */
17.   packet in_buf[NR_BUFS];       /* buffers for the inbound stream */
18.   boolean arrived[NR_BUFS];     /* inbound bit map */
19.   seq_nr nbuffered;             /* how many output buffers currently used */
20.   event_type event;
21.   ack_expected = 0;             /* next ack expected on the inbound stream */
22.   next_frame_to_send = 0;       /* number of next outgoing frame */
23.   frame_expected = 0;
24.   too_far = NR_BUFS;
25.   nbuffered = 0;                /* initially no packets are buffered */
26.   for (i = 0; i < NR_BUFS; i++) arrived[i] = false;
```


P6c

```
1.  enable_network_layer();         /* initialize */
2.   while (true) {
3.     wait_for_event(&event);      /* five possibilities: see event_type above */
4.     switch(event) {
5.       case network_layer_ready: /* accept, save, and transmit a new frame */
6.            /* voir développement [1] ci-après */
7.            break;
8.       case frame_arrival:                  /* a data or control frame has arrived */
9.            from_physical_layer(&r);/* fetch incoming frame from physical layer */
10.           /* voir développement [2] ci-après */
11.           break;
12.      case cksum_err:
13.           if (no_nak) send_frame(nak, 0, frame_expected, out_buf); /* damaged frame */
14.           break;
15.      case timeout:
16.           send_frame(data, oldest_frame, frame_expected, out_buf);  /* we timed out */
17.           break;
18.      case ack_timeout:
19.           send_frame(ack, 0, frame_expected, out_buf);  /* ack timer expired; send ack */
20.    }
21.    if (nbuffered < NR_BUFS)
22.      enable_network_layer();
23.    else
24.      disable_network_layer();
25.  }
26. }
```

P6c1

```
1.  /* développement [1] */
2.  case network_layer_ready:                /* accept, save, and transmit a new frame */
3.          nbuffered = nbuffered + 1;       /* expand the window */
4.          from_network_layer(&out_buf[next_frame_to_send % NR_BUFS]); /* fetch new packet */
5.          send_frame(data, next_frame_to_send, frame_expected, out_buf);/* transmit the frame */
6.          inc(next_frame_to_send); /* advance upper window edge */
7.          break;
```


P6c2

```
1.  /* développement [2] */
2.  case frame_arrival:                      /* a data or control frame has arrived */
3.          from_physical_layer(&r);/* fetch incoming frame from physical layer */
4.          if (r.kind == data) {
5.                  /* An undamaged frame has arrived. */
6.                  if ((r.seq != frame_expected) && no_nak)
7.                     send_frame(nak, 0, frame_expected, out_buf); else start_ack_timer();
8.                  if (between(frame_expected, r.seq, too_far) && (arrived[r.seq%NR_BUFS] == false)) {
9.                          /* Frames may be accepted in any order. */
10.                         arrived[r.seq % NR_BUFS] = true;          /* mark buffer as full */
11.                         in_buf[r.seq % NR_BUFS] = r.info;         /* insert data into buffer */
12.                         while (arrived[frame_expected % NR_BUFS]) {
13.                                 /* Pass frames and advance window. */
14.                                 to_network_layer(&in_buf[frame_expected % NR_BUFS]);
15.                                 no_nak = true;
16.                                 arrived[frame_expected % NR_BUFS] = false;
17.                                 inc(frame_expected);  /* advance lower edge of receiver's window */
18.                                 inc(too_far);         /* advance upper edge of receiver's window */
19.                                 start_ack_timer();    /* to see if a separate ack is needed */
20.                         }
21.                 }
22.         }
23.         if((r.kind==nak) && between(ack_expected,(r.ack+1)%(MAX_SEQ+1),next_frame_to_send))
24.                 send_frame(data, (r.ack+1) % (MAX_SEQ + 1), frame_expected, out_buf);
25.         while (between(ack_expected, r.ack, next_frame_to_send)) {
26.                 nbuffered = nbuffered - 1;       /* handle piggybacked ack */
27.                 stop_timer(ack_expected % NR_BUFS);          /* frame arrived intact */
28.                 inc(ack_expected);                /* advance lower edge of sender's window */
29.         }
30.         break;
```

Questions

- Quel est le lien entre « MAX_SEQ » et « NR_BUF »?
- Le nombre de tampons doit-il être le même de part et d'autre?
- Où utilise-t-on l'hypothèse que « MAX_SEQ = 2^n-1 »?
- Pourquoi la variable « no_nak » est-elle statique?
- Est-il nécessaire que les trames soient reçues dans l'ordre d'émission?
- Ces protocoles pourraient-ils être utilisés avec un support hertzien?