Računarske mreže i mrežne tehnologije

5. termin

1. Osnovni protokoli sloja veze podataka (data link layer)

- Da bi se objasnila funkcionalnost protokola koji slede, uvode se određene pretpostavke koje leže u osnovi komunikacionog modela
- Prva pretpostavka je da se **za svaki sloj odigrava poseban proces**, dakle fizički sloj, sloj veze podataka i mrežni sloj su posebni procesi u sistemu. U praksi, fizički sloj i sloj veze se izvršavaju u NIC (*Network Interface Controller*) kontroleru.
- Druga pretpostavka je da računar A želi da pošalje računaru B dugačak niz bitova **koristeći pouzdanu, direktno uspostavljenu vezu**.
- Treća pretpostavka modela je da **računari rade besprekorno**, da greške isključivo nastaju u njihovoj komunikaciji.
- Četvrta pretpostavka je da **mrežni sloj uvek ima paket za slanje,** ali se ova pretpostavka kasnije odbacuje.
- Čekanje sloja veze na neki događaj rešeno je procedurom *wait_for_event(&event)*.
- Paketi se među slojevima prenose putem interfejsa, npr. funkcijom *from_network_layer()* i *to_network_layer()*.
- Osnovna jedinica prenosa je okvir (*frame*) koji se sastoji iz zaglavlja i podataka,u programu označen sa *struct frame*.
- Podaci se prenose od računara A do računara B procedurom *to_physical_layer()*, a na računaru B se preuzimaju putem procedure *from_physical_layer()*.
- Kada okvir pristigne primaocu, hardverski se izračuna CRC. Ukoliko postoji greška, o tome se obaveštava sloj veze podataka (*event=cksum_error*). Ako je okvir stigao neoštećen, odgovarajući događaj je *event=frame_arrival*.
- MAX_PKT određuje veličinu paketa mrežnog sloja u bajtovima.
- Deklaracije struktura i funkcija simulatora date su na slici.

```
#define MAX_PKT 1024                        /* determines packet size in bytes */

typedef enum {false, true} boolean;         /* boolean type */
typedef unsigned int seq_nr;                /* sequence or ack numbers */
typedef struct {unsigned char data[MAX_PKT];} packet;/*    packet definition */
typedef enum {data, ack, nak} frame_kind;   /* frame_kind definition */

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

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

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

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

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

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

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

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

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

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

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

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

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

- Redni brojevi okvira se nalaze u intervalu 0..MAX_SEQ.
- Problem probijanja granice MAX_SEQ rešava se makroom *inc(k)*.

2. Protokol za neograničen jednosmeran prenos podataka

- Prvi primer je najjednostavniji protokol iprema tome potpuno nerealan.
- Obezbeđuje prenos u samo jednom smeru (od pošiljaoca do primaoca)
- Pretpostavlja da se na komunikacionom kanalu ne mogu pojaviti greške
- Pretpostavlja da prijemnik može beskonačno brzo da obradi podatke koji mu pristižu (ili da poseduje beskonačan bafer)

```
/* Protocol 1 (utopia) provides for data transmission in one direction only, from
   sender to receiver.  The communication channel is assumed to be error free
   and the receiver is assumed to be able to process all the input infinitely quickly.
   Consequently, the sender just sits in a loop pumping data out onto the line as
   fast as it can. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender1(void)
{
  frame s;                          /* buffer for an outbound frame */
  packet buffer;                    /* buffer for an outbound packet */

  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,
                                       Creeps in this petty pace from day to day
                                       To the last syllable of recorded time.
                                           - Macbeth, V, v */
}


void receiver1(void)
{
  frame r;
  event_type event;                 /* filled in by wait, but not used here */

  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 */
  }
}
```

3. Simplex protokol “stani i čekaj” (*stop & wait*)

- Drugi primer odbacuje najmanje realističnu pretpostavku prvog protokola, a to je da mrežni sloj primaoca može beskonačno brzo da obradi podatke koji stižu.
- I dalje se pretpostavlja da nema grešaka u komunikaciji, i da saobraćaj ide u samo jednom smeru
- Ako je primaocu potrebno konačno vreme *t* da izvrši procedure *from_physical_layer()* i to*_network_layer()*, pošiljalac mora da šalje okvire brzinom koja je manja od jednog okvira u vremenu *t*.
- Kako *t* može da varira, vremenska zadrška nije dobro rešenje.
- Pravo rešenje je da primalac šalje neku povratnu informaciju da je okvir primio i obradio. Uprkos tome, kanal je i dalje simplex.
- Ova vrsta protokola ne šalje novi okvir dok ne dobije potvrdu (*acknowledgement frame*) od prijemu prethodnog, zbog čega i nosi naziv “stop & wait”:

```
/* Protocol 2 (stop-and-wait) also provides for a one-directional flow of data from
   sender to receiver. The communication channel is once again assumed to be error
   free, as in protocol 1. However, this time, the receiver has only a finite buffer
   capacity and a finite processing speed, so the protocol must explicitly prevent
   the sender from flooding the receiver with data faster than it can be handled. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender2(void)
{
  frame s;                          /* buffer for an outbound frame */
  packet buffer;                    /* buffer for an outbound packet */
  event_type event;                 /* frame_arrival is the only possibility */

  while (true) {
      from_network_layer(&buffer);  /* go get something to send */
      s.info = buffer;              /* copy it into s for transmission */
      to_physical_layer(&s);        /* bye-bye little frame */
      wait_for_event(&event);       /* do not proceed until given the go ahead */
  }
}

void receiver2(void)
{
  frame r, s;                       /* buffers for frames */
  event_type event;                 /* frame_arrival is the only possibility */
  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 */
      to_physical_layer(&s);        /* send a dummy frame to awaken sender */
  }
}
```

3. Simplex protokol za slanje podataka bučnim kanalom

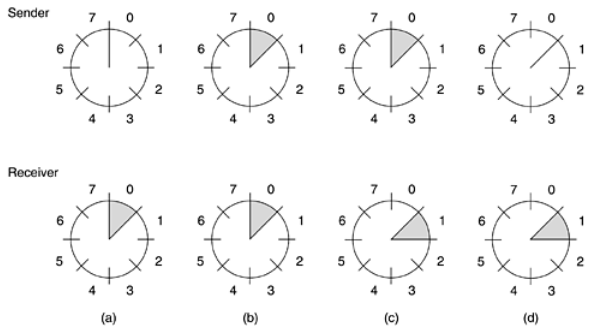
- Treći primer odbacuje pretpostavku da je komunikacioni kanal bešuman
- Pretpostavlja se da **hardver ume da izračuna kontrolni zbir** (npr. CRC) i da na taj način razdvoji ispravne od neispravnih okvira.
- Međutim, **okvir može i potpuno da se izgubi u prenosu**, a to se odnosi i na okvire koji nose podatke, a i na kontrolne okvire.
- Da pošiljalac ne bi čekao beskonačno dugo na potvrdu o prijemu, uvodi se tajmer. Ako potvrda ne stigne za neko razumno vreme *t*, ponovo se šalje isti okvir.
- **Međutim, šta ako je okvir sa podacima ispravno stigao prijemniku, a potvrda se izgubila?**
- U toj situaciji, pošiljalac ponovo šalje okvir, a **mrežni sloj dva puta prima isti paket**, što je nedopustivo.
- Trivijalno rešenje je da **pošiljalac stavi redni broj okvira u zaglavlje okvira**.
- Postavlja se pitanje kolika treba da bude veličina broja sekvence u bitovima. Kada se radi o jednostavnom protokolu koji ima bafer od samo jednog okvira, **dovoljan je 1 bit**, jer pošiljalac nema dozvolu da pošalje okvir rednog broja *m*+1 dok okvir *m* nije ispravno primljen.
- Ako pošiljaocu stigne oštećena potvrda o prijemu ili istekne tajmer, on ne dira bafer u kome se već spreman nalazi frejm spreman za ponovno slanje.
- Ova vrsta protokola poznta je kao PAR (*Positive Acknowledgement with Retransmission*) ili ARQ (*Automatic Repeat reQuest*).

```
/* Protocol 3 (par) allows unidirectional data flow over an unreliable channel. */
#define MAX_SEQ 1                       /* must be 1 for protocol 3 */
typedef enum  {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
  seq_nr next_frame_to_send;            /* seq number of next outgoing frame */
  frame s;                              /* scratch variable */
  packet buffer;                        /* buffer for an outbound packet */
  event_type event;

  next_frame_to_send = 0;               /* initialize outbound sequence numbers */
  from_network_layer(&buffer);          /* fetch first packet */
  while (true) {
      s.info = buffer;                  /* construct a frame for transmission */
      s.seq = next_frame_to_send;       /* insert sequence number in frame */
      to_physical_layer(&s);            /* send it on its way */
      start_timer(s.seq);               /* if answer takes too long, time out */
      wait_for_event(&event);           /* frame_arrival, cksum_err, timeout */
      if (event == frame_arrival) {
          from_physical_layer(&s);      /* get the acknowledgement */
          if (s.ack == next_frame_to_send) {
              stop_timer(s.ack);        /* turn the timer off */
              from_network_layer(&buffer); /* get the next one to send */
              inc(next_frame_to_send);  /* invert next_frame_to_send */
          }
      }
  }
}

void receiver3(void)
{
  seq_nr frame_expected;
  frame r, s;
  event_type event;

  frame_expected = 0;
  while (true) {
      wait_for_event(&event);           /* possibilities: frame_arrival, cksum_err */
      if (event == frame_arrival) {     /* a valid frame has arrived. */
          from_physical_layer(&r);      /* go get the newly arrived frame */
          if (r.seq == frame_expected) {/* this is what we have been waiting for. */
              to_network_layer(&r.info);/* pass the data to the network layer */
              inc(frame_expected);      /* next time expect the other sequence nr */
          }
          s.ack = 1 – frame_expected;   /* tell which frame is being acked */
          to_physical_layer(&s);        /* send acknowledgement */
      }
  }
}
```

4. Protokoli kliznih prozora

- U prethodnim protokolima podaci su se prenosili samo u jednom smeru. Međutim, u većini primena se zahteva **dvosmerna komunikacija**.
- Jedan od načina da se to reši je postavljanje dva paralelna komunikaciona kanala, ali takvo rešenje je potpuno neefikasno jer korisnik plaća kapacitet dva kanala a koristi samo jedan u datom momentu.
- Bolje je da sav saobraćaj teče preko jednog komunikacionog kanala. Po tom modelu, okviri sa podacima i sa potvrdama se smenjuju na istom kanalu (polje *kind* u *struct frame*).
- Procedura se dodatno može ubrzati i pojednostaviti korišćenjem tzv. **“šlepovanja”** (*piggybacking*). Umesto da računar B pošalje zaseban okvir potvrde o prijemu okvira sa računara A, on sačeka da njegov mrežni sloj (računara B) prosledi sledeći pakt podataka i uz njega priključi i potvrdu.
- Polje potvrde (*ack*) je veoma malo i ne predstavlja nikakvo naročito opterećenje.
- Međutim, **šta ako mrežni sloj računara B nema šta da pošalje**? Tada se šalje potvrda o prijemu kao poseban prazan okvir, kao u prethodnom protokolu.
- **Protokoli klase kliznih prozora** odlikuju se postojanjem posebnih “prozora“ za slanje i prijem određene, u generalnom slučaju različite veličine. Prozor je skup rednih brojeva okvira koje protokol sme u nekom trenutku da pošalje ili primi.
- **Redni brojevi pošiljaočevog prozora** odgovaraju okvirima koji su poslati ili se mogu poslati, a čiji prijem još nije potvrđen. Kad god novi paket stigne iz mrežnog sloja, dodeljuje mu se prvi najviši redni broj, a granica prozora se povećava za 1. Kada stigne potvrda, donja granica se povećava za 1.
- **Redni brojevi prozora primaoca** su oni koje sme da primi. Svaki okvir izvan ovog spiska se odbacuje. Kada stigne okvir koji odgovara donjoj granici prozora, prozor se rotira za 1

Figure 3-13. 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.



- Polje *ack* sadrži redni broj poslednjeg okvira primljenog bez greške.

```
/* 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 */
 }
}
```

- **Samo jedan od dva sloja veze (na računarima A i B) počinje komunikaciju**, tj. šalje prvi okvir. Znači, samo jedan od njih treba da ima uključen *to_physical_layer* i *start_timer* izvan *while* petlje.
- **Sloj veze primaoca** proverava da li je to duplikat nekog drugog okvira, pa ako je okvir koji se očekuje, prosleđuje se mrežnom sloju, a prozor primaoca se pomera za (*frame_expected*).
- **Sloj veze pošiljaoca**: ako se *ack* slaže s rednim brojem okvira koji pošiljalac pokušava da pošalje (*next_frame_to_send*), on zna da je završio posao i može da uzme sledeći paket od svog mrežnog sloja. Ako ne, šalje isti okvir ponovo.
- Nijedna kombinacija nepravilno podešenog tajmera ili grešaka u prenosu **ne može da natera protokol da primi neispravan okvir kao ispravan**, na slici:

Figure 3-15. Two scenarios for protocol 4. (a) Normal case. (b) Abnormal case. The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet.