Osnovni protokoli sloja veze podataka

Računarske mreže i mrežne tehnologije
Mart 2018. god.

Sadržaj predavanja

1) Osnovni protokoli sloja veze podataka
2) Protokol za neograničen jednosmeran prenos podataka
3) Simplex protokol “stani i čekaj” (stop & wait)
4) Simplex protokol za slanje podataka bučnim kanalom
5) Protokoli kliznih prozora
6) Protokol tipa “vrati se N” (*go-back-N*)
7) Protokol sa selektivnim ponavljanjem (*selective repeat*)
8) Sloj veze podataka na Internetu (PPP – *Point-to-Point Protocol*)

Pretpostavke modela

Da bi se objasnila funkcionalnost protokola koji slede, uvode se određene pretpostavke koje leže u osnovi komunikacionog modela

I. **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.

II. Računar **A** želi da pošalje računaru **B** dugačak niz bitova **koristeći pouzdanu, direktno uspostavljenu vezu**.

III. **Računari rade besprekorno**, da greške isključivo nastaju u njihovoj komunikaciji.

IV. **Mrežni sloj uvek ima paket za slanje,** ali se ova pretpostavka kasnije odbacuje.

Pretpostavke modela

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

Osnovne strukture podataka modela

```
#define MAX_PKT 4   /* 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);
```

Osnovne strukture podataka modela (2)

```
/* 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
```

Protokol za neograničen jednosmeran prenos podataka

- Prvi primer je najjednostavniji protokol i prema 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)

Protokol za neograničen jednosmeran prenos podataka

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

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

- Ovaj protokol 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*

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

```
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 */
        init_frame(&s);
        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 */
  init_frame(&s);
  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 */
  }
}
```

Simplex protokol za slanje podataka bučnim kanalom

- Treći protokol 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 nalazi frejm spreman za ponovno slanje.
- Ova vrsta protokola poznta je kao PAR (*Positive Acknowledgement with Retransmission*) ili ARQ (*Automatic Repeat reQuest*).

Simplex protokol za slanje podataka bučnim kanalom (2)

```
/* 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) {
        init_frame(&s);
        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) {
                        from_network_layer(&buffer); /* get the next one to send */
                        inc(next_frame_to_send);     /* invert next_frame_to_send */
                }
        }
  }
}
```

Simplex protokol za slanje podataka bučnim kanalom (3)

```
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 */
                }
                init_frame(&s);
                s.ack = 1 - frame_expected; /* tell which frame is being acked */
                to_physical_layer(&s);/* only the ack field is use */
        }
  }
}
```

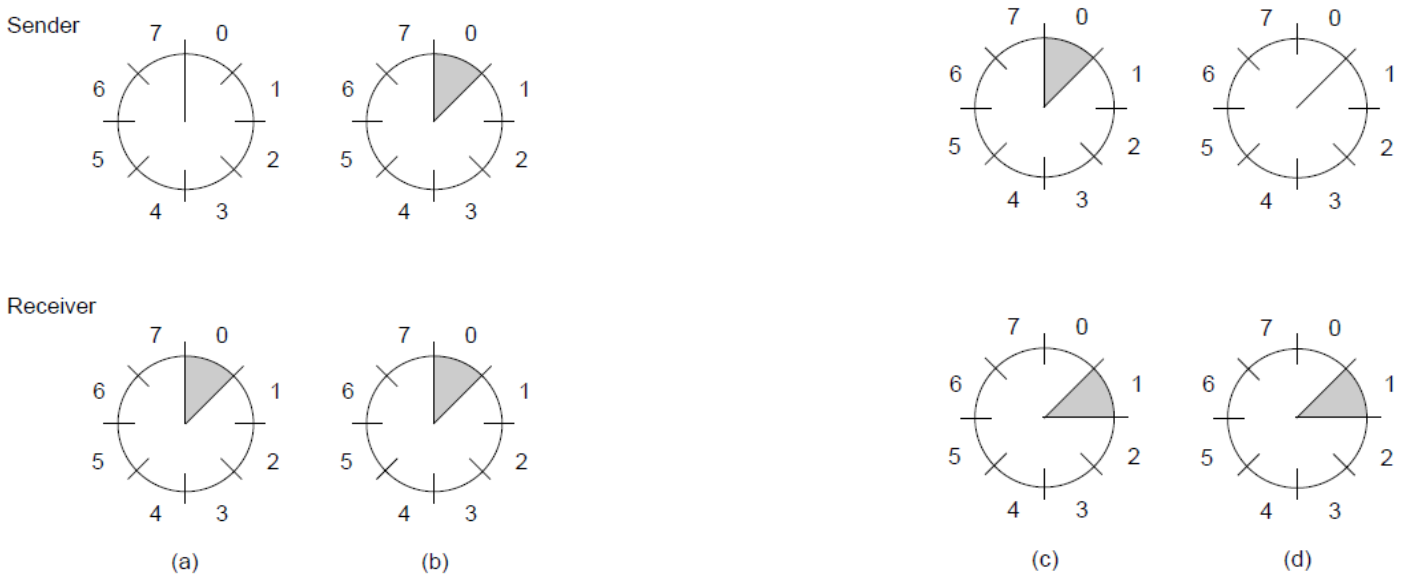
Protokoli kliznih prozora (*sliding window*)

- 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.

Protokoli kliznih prozora (2)

- 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

Protokoli kliznih prozora (3)



(a) Inicijalno stanje, (b) Nakon slanja prvog okvira, (c) Nakon prijema prvog okvira, (d) Nakon prijema potvrde

1-bitni protokol kliznih prozora

```
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; /* number of frame arriving frame expected */
  from_network_layer(&buffer);   /* fetch a packet from the network layer */
  init_frame(&s);
  s.kind = data;
  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);  /* could be: frame_arrival, cksum_err, timeout */
        if ((event != frame_arrival) && (event != cksum_err) &&
            (event != timeout)) printf("\n SOMETHING WEIRD\n");
        if (event == frame_arrival) { /* a frame has arrived undamaged. */
              from_physical_layer(&r);    /* go get it */
```

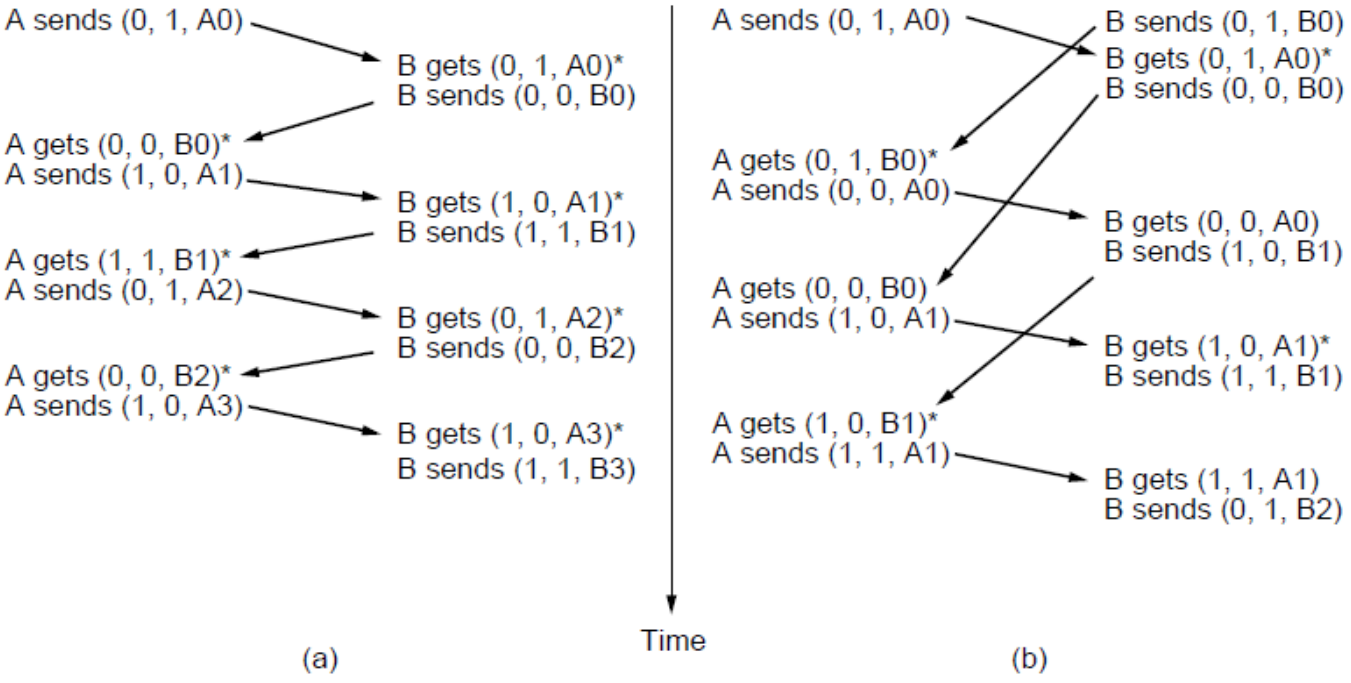
1-bitni protokol kliznog prozora

```
                if (r.seq == frame_expected) {
                        /* Handle inbound frame stream. */
                        to_network_layer(&r.info);     /* pass packet to network layer */
                        inc(frame_expected); /* invert sequence number expected next */
                }
                if (r.ack == next_frame_to_send) { /* handle outbound frame stream. */
                        from_network_layer(&buffer);   /* fetch new packet from network layer */
                        inc(next_frame_to_send);  /* invert sender's sequence number */
                }
        }
        init_frame(&s);
        s.kind = data;
        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 */
  }
}
```

1-bitni protokol kliznih prozora

- **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.**

1-bitni protokol kliznih prozora



Dva scenarija za protokol 4. (a) Normalni slučaj. (b) Neuobičajen slučaj.
Notacija je (*seq, ack, packet number*). "*" ukazuje na prijem paketa od strane mrežnog sloja

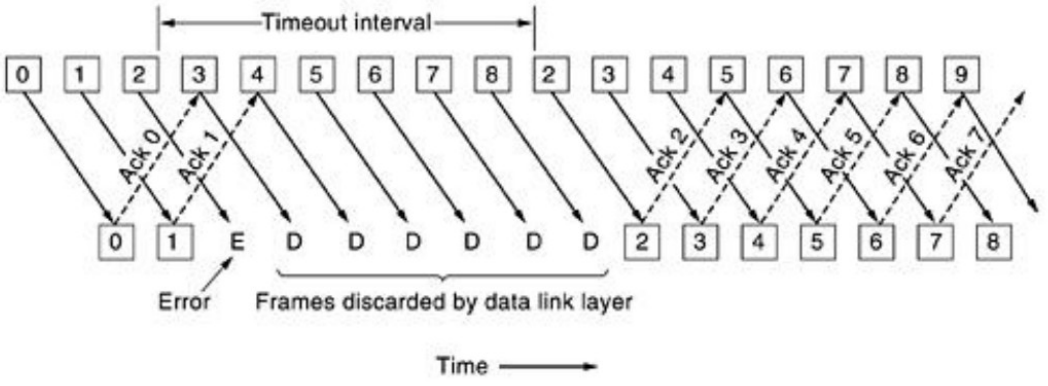
Protokol tipa “vrati se N” (go-back-N)

- **Pitanje:** Šta se dešava kada je vreme prolaska okvira sa kraja na kraj kanala nezanemarljivo?
- **Primer:** Satelitski kanal brzine 50kb/s čije je vreme prolaska vezom (do satelita i natrag) 500 ms. Protokolom 4 se šalje okvir od 1000 bita, nakon 20 ms se završava slanje, nakon 270 ms stiže do primaoca, a potvrda stiže tek nakon 520 ms. **Pošiljalac je blokiran 500 od 520 ms, tj 96% vremena!**
- Iskorišćenje linije je ***l/(l+bR)***, gde je *b* brzina kanala, *l* veličina okvira u bitovima, a *R* ukupno vreme prolaska vezom (napred i nazad).
- **Šta je uzrok problema?** Pravilo da se može poslati samo jedan okvir u isto vreme!
- Dozvoliti da se pošalje *w* okvira bez čekanja potvrde! U našem primeru, *w* bi trebalo da bude bar 26. Zašto?
- Kritično je i kada imamo veliki propusni opseg, a umereno kašnjenje.

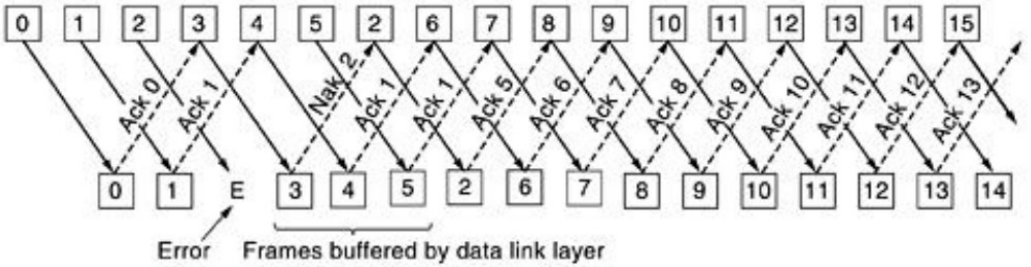
Protokol tipa “vrati se N” (2)

- Otvara se pitanje: Šta raditi kada dodje do greške u prijemu okvira negde u sred rafala? Moguća su dva pristupa:
  (1) Vratiti se do okvira mesta gde je nastala greška i ponoviti slanje (**vrati se *N***)
  (2) Zadržati sve ispravne okvire i ponoviti samo najstariji nepotvrđeni okvir. (**selektivno ponavljanje**)
- Ako se koristi i NAK (*Negative Acknowledgement*), dobije se na brzini. Zašto?
- Protokolu “**vrati se N**” odgovara prozor primaoca dužine 1, dok je kod protokola “**selektivnog ponavljanja**” prozor primaoca veći.
- **Protokol 5** koristi tehniku “vrati se N”, uz napuštanje ranije pretpostavke da je mrežni sloj uvek spreman (uvodi se događaj **network_layer_ready**). Takođe, mrežni sloj se aktivira/deaktivira koristeći **enable_network_layer** i **disable_network_layer**. Kada je to potrebno?
- http://www.ccs-labs.org/teaching/rn/animations/gbn_sr/

Protokol tipa “vrati se N” (3)



(a)



(b)

Protokol tipa “vrati se N” (4)

```
/* 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);
}
```

Protokol tipa “vrati se N” (5)

```
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 */
}
```

Protokol tipa “vrati se N” (6)

```
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 */
```

Protokol tipa “vrati se N” (7)

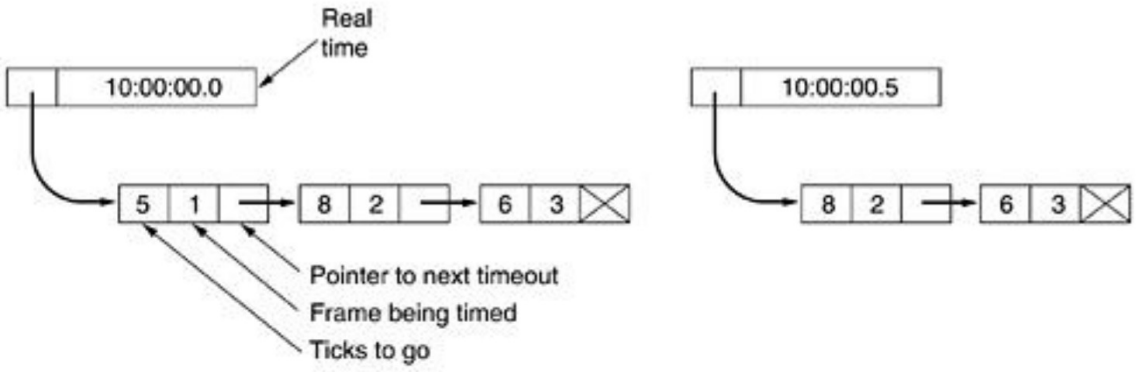
```
switch(event) {
  case network_layer_ready:                /* the network layer has a packet to send */
       /* Accept, save, and transmit a new frame. */
       from_network_layer(&buffer[next_frame_to_send]); /* fetch new packet */
       nbuffered = nbuffered + 1;          /* expand the sender's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

Protokol tipa “vrati se N” (8)

```
            /* 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 frame */
                inc(next_frame_to_send);            /* prepare to send the next one */
            }

    }

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

Protokol tipa “vrati se N” (9)

- Na kanalu sme biti najviše **MAX_SEQ** okvira, a ne **MAX_SEQ+1**. Zašto?
- Kada stigne potvrda za okvir *n*, automatski se potvrđuju i *n*-1, *n*-2, itd. To je bitno jer neka od prethodnih potvrda može da se izgubi. U ovom protokolu se takođe pretpostavlja da uvek postoji povratni saobraćaj kojim mogu da se šlepuju okviri.
- Svaki okvir ima svoj tajmer. Tajmeri se organizuju kao pointerska lista.

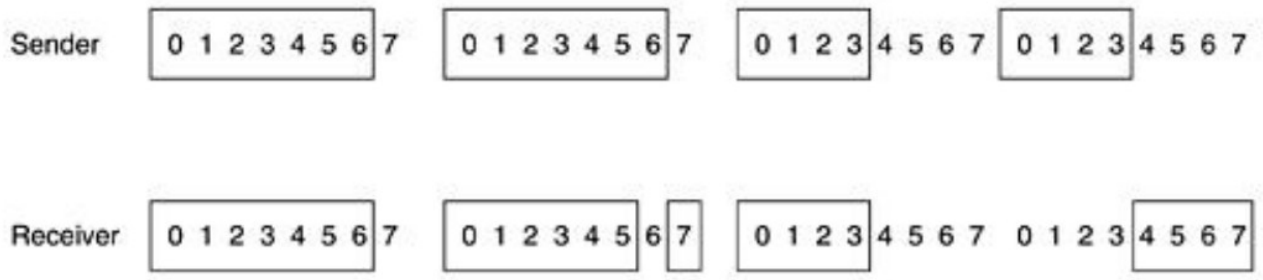


Protokol sa selektivnim ponavljanjem

- Protokol 5 je dobar, ali samo kada greške nisu česte. Ako su greške učestale, onda se troši previše propusnog opsega.
- Okviri mrežnom sloju moraju da se isporuče **ispravnim redosledom**!
- Prozor primaoca uvek ima konstantnu dužinu MAX_SEQ.
- http://www.ccs-labs.org/teaching/rn/animations/gbn_sr/
- Kada god stigne okvir, proverava ga funkcija **between()**, pri čemu utvrđuje da li se nalazi u granicama prijemnog prozora.
- Kada rok tajmera istekne, primaocu se šalje samo taj okvir, a ne svi koji se nalaze u prozoru pošiljaoca.

Protokol sa selektivnim ponavljanjem (2)

- Zašto broj bafera ne sme da pređe polovinu **MAX_SEQ**?
- Rešenje je da se redni brojevi okvira dva uzastopna bafera ne smeju preklapati!
- U protokolu 6 takođe se ne pretpostavlja da postoji povratni saobraćaj na koji mogu da se kače potvrde. Ako paket iz mrežnog sloja ne dođe neko vreme, šalje se posebna potvrda. Za to se koristi pomoćni tajmer koji se startuje sa **start_ack_timer**, a događaj koji ga aktivira je **ack_timeout**.
- NAK-okvir sa naznakom za ponovno slanje datog okvira. *no_nak* promenljiva služi da se izbegne višestruko slanje NAK-a za isti okvir (**frame_expected**).
- Ako se NAK izgubi-nema štete. Zašto?



Protokol sa selektivnim ponavljanjem (3)

```
/* Protocol 6 (Selective repeat) accepts frames out of order but passes packets to the
   network layer in order. Associated with each outstanding frame is a timer. When the timer
   expires, only that frame is retransmitted, not all the outstanding frames, as in protocol 5. */

#define MAX_SEQ 7                                   /* should be 2^n – 1 */
#define NR_BUFS ((MAX_SEQ + 1)/2)
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready, ack_timeout} event_type;
#include "protocol.h"
boolean no_nak = true;                              /* no nak has been sent yet */
seq_nr oldest_frame = MAX_SEQ + 1;                  /* initial value is only for the simulator */

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Same as between in protocol 5, but shorter and more obscure. */
  return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
}
```

Protokol sa selektivnim ponavljanjem (4)

```
static void send_frame(frame_kind fk, seq_nr frame_nr, seq_nr frame_expected, packet buffer[])
{
/* Construct and send a data, ack, or nak frame. */
  frame s;                                        /* scratch variable */

  s.kind = fk;                                    /* kind == data, ack, or nak */
  if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
  s.seq = frame_nr;                               /* only meaningful for data frames */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
  if (fk == nak) no_nak = false;                  /* one nak per frame, please */
  to_physical_layer(&s);                          /* transmit the frame */
  if (fk == data) start_timer(frame_nr % NR_BUFS);
  stop_ack_timer();                               /* no need for separate ack frame */
}
```

Protokol sa selektivnim ponavljanjem (5)

```
enable_network_layer();                          /* initialize */
ack_expected = 0;                                /* next ack expected on the inbound stream */
next_frame_to_send = 0;                          /* number of next outgoing frame */
frame_expected = 0;
too_far = NR_BUFS;
nbuffered = 0;                                   /* initially no packets are buffered */
for (i = 0; i < NR_BUFS; i++) arrived[i] = false;
```

Protokol sa selektivnim ponavljanjem (6)

```
while (true) {
  wait_for_event(&event);                          /* five possibilities: see event_type above */
  switch(event) {
    case network_layer_ready:                      /* accept, save, and transmit a new frame */
         nbuffered = nbuffered + 1;                /* expand the window */
         from_network_layer(&out_buf[next_frame_to_send % NR_BUFS]); /* fetch new packet */
         send_frame(data, next_frame_to_send, frame_expected, out_buf);/* transmit the frame */
         inc(next_frame_to_send);                  /* advance upper window edge */
         break;
```

Protokol sa selektivnim ponavljanjem (7)

```
case frame_arrival:                          /* a data or control frame has arrived */
    from_physical_layer(&r);                 /* fetch incoming frame from physical layer */
    if (r.kind == data) {
        /* An undamaged frame has arrived. */
        if ((r.seq != frame_expected) && no_nak)
            send_frame(nak, 0, frame_expected, out_buf); else start_ack_timer();
        if (between(frame_expected,r.seq,too_far) && (arrived[r.seq%NR_BUFS]==false)) {
            /* Frames may be accepted in any order. */
            arrived[r.seq % NR_BUFS] = true;      /* mark buffer as full */
            in_buf[r.seq % NR_BUFS] = r.info;     /* insert data into buffer */
```

Protokol sa selektivnim ponavljanjem (8)

```
            while (arrived[frame_expected % NR_BUFS]) {
                /* Pass frames and advance window. */
                to_network_layer(&in_buf[frame_expected % NR_BUFS]);
                no_nak = true;
                arrived[frame_expected % NR_BUFS] = false;
                inc(frame_expected);     /* advance lower edge of receiver's window */
                inc(too_far);            /* advance upper edge of receiver's window */
                start_ack_timer();       /* to see if a separate ack is needed */
            }
        }
}
```

Protokol sa selektivnim ponavljanjem (9)

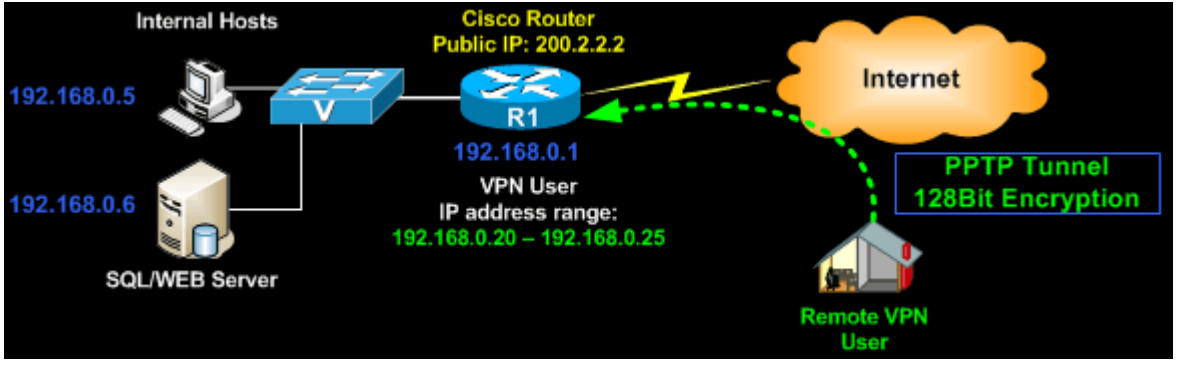
```
        if((r.kind==nak) && between(ack_expected,(r.ack+1)%(MAX_SEQ+1),next_frame_to_send))
                send_frame(data, (r.ack+1) % (MAX_SEQ + 1), frame_expected, out_buf);

        while (between(ack_expected, r.ack, next_frame_to_send)) {
                nbuffered = nbuffered – 1;              /* handle piggybacked ack */
                stop_timer(ack_expected % NR_BUFS);     /* frame arrived intact */
                inc(ack_expected);                      /* advance lower edge of sender's window */
        }
        break;
case cksum_err:
        if (no_nak) send_frame(nak, 0, frame_expected, out_buf); /* damaged frame */
        break;
```

Protokol sa selektivnim ponavljanjem (10)

```
        case timeout:
                send_frame(data, oldest_frame, frame_expected, out_buf); /* we timed out */
                break;
        case ack_timeout:
                send_frame(ack,0,frame_expected, out_buf);      /* ack timer expired; send ack */
    }
    if (nbuffered < NR_BUFS) enable_network_layer(); else disable_network_layer();
  }
}
```

Sloj veze podataka na Internetu (PPP – *Point-to-Point Protocol*)

- Kako se na Internet povezuje kućni ruter putem ADSL ili kablovske linije?
- Nakon povezivanja PPP protokolom kućni ruter dobija privremenu javnu IP adresu čime postaje punopravni član Interneta.



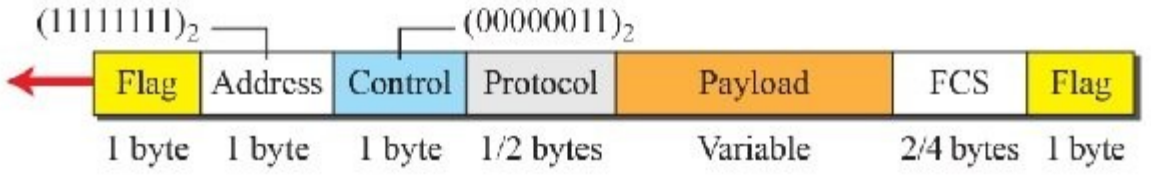
Funkcije PPP-a

Protokol PPP obezbeđuje sledeće funkcije:

1. **Metodu uokviravanja**
2. **Protokol za upravljanje vezom** (*Link Control Protocol, LCP*). Povezivanje, dogovaranje, proveravanje komunikacionih opcija
3. **Dogovaranje opcija mrežnog sloja** nezavisno od toga koji se protokol u tom sloju koristi (*Network Control Protocol, NCP*). Najvažniji je dogovor oko dinamičke IP adrese.

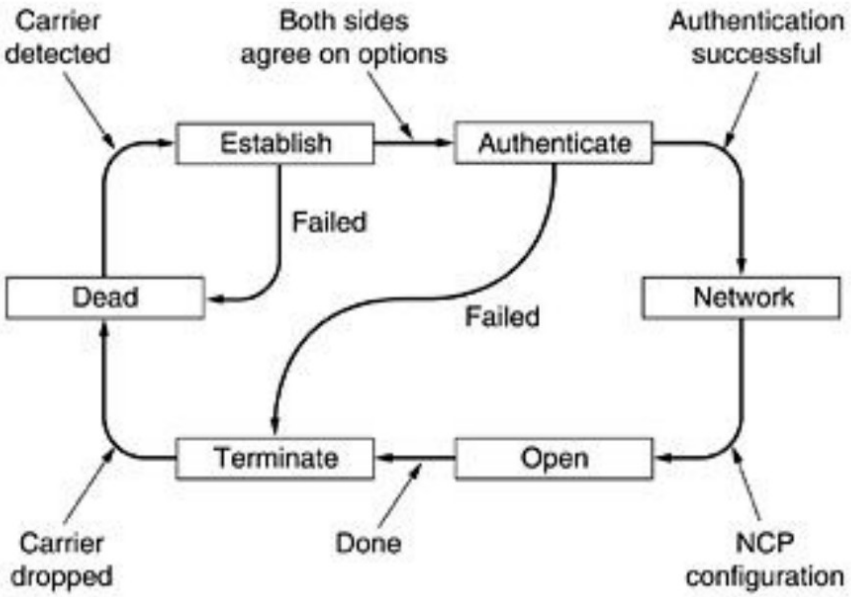
Struktura PPP okvira

- Polje *Address* je obično 11111111, što znači da sve stanice treba da prihvate okvir.
- Vrednost *Control* polja 00000011 znači da su okviri NENUMERISANI, tj. da PPP podrazumevano ne garantuje ispravan redosled prijema.
- Vrednost u polju *Protocol* označava vrstu paketa koji se prenosi u polju *Payload*. To mogu biti LCP, NCP, IP, IPX, ...
- *Payload* može biti različite dužine, sve do dogovorenog maksimuma.



PPP-uspostavljanje veze

Dijagram stanja protokola PPP



Vrste LCP okvira

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)