Интернет: управление потоком (Stop and Wait)

Введение в компьютерные сети

проф. Смелянский Р.Л.
Лаборатория Вычислительных комплексов
ф-т ВМК МГУ

Управление потоком

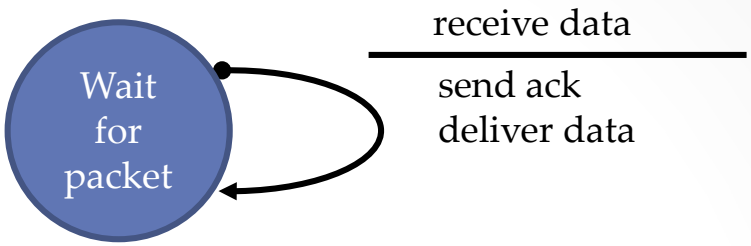
- *Не посылать пакетов больше, чем может принять получатель*
- *Есть обратная связь между отправителем и получателем*
- *Два основных подхода:*
  - *Stop and Wait*
  - *Скользящее окно (Sliding Window)*

Управление потоком (т.1 стр. 122-128)

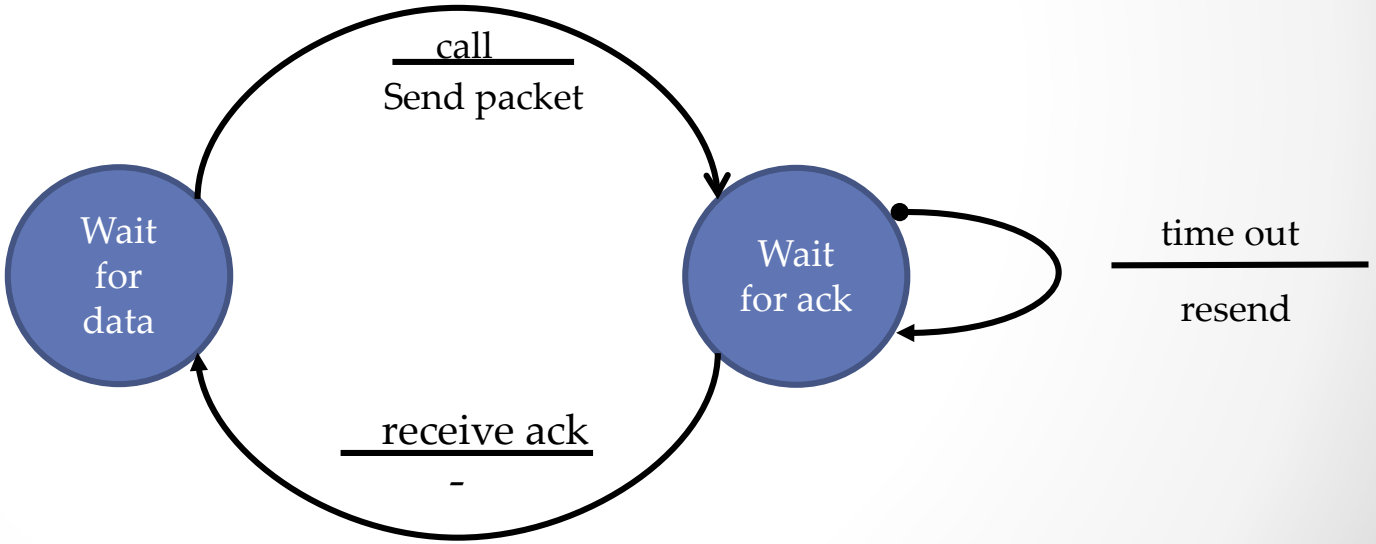
```
#define MAX_PKT 1024          /* determines packet size in bytes */

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
```

Управление потоком

Receiver FSM



Sender FSM



Управление потоком

```
/* 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 fast.
   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 */
  }
}
```

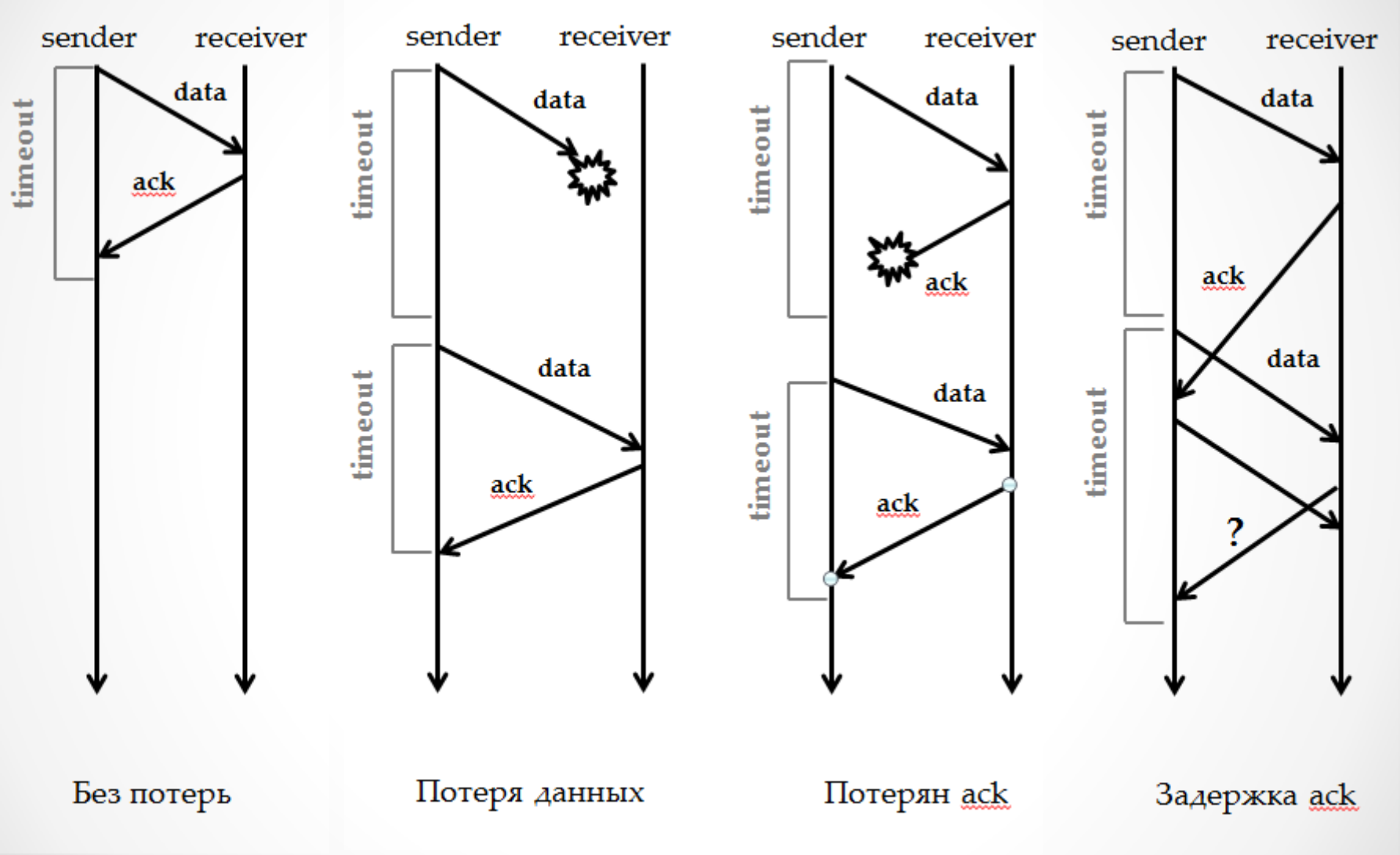
Управление потоком

```
/* 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); /*        wait event = ack V time_out          */
  }
}

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 packet with ack             */
  }
}
```

Управление потоком (stop and wait)

- В одно и то же время передают не более одного пакета
- Sender отправляет пакет
- Receiver посылает пакет с ack , когда получает пакет данных
- Получив ack, sender шлет новый пакет с данными
- По time_out, sender повторно посылает пакет с данными

Управление потоком



Дублирование

- *Счетчик на 1 бит в данных и уведомлении позволяет отличать новые данные от дубликатов*
- *Будем предполагать*
  - *Сама сеть не размножает пакеты*
  - *Запаздывание пакетов гарантированно не более одного time_out*