Средства описания протоколов: Диаграммы переходов для конечных автоматов

Введение в компьютерные сети

чл.-корр. РАН, проф. Смелянский Р.Л.

Кафедра АСВК ф-т ВМК МГУ

Конечный автомат

<Ain; Aout; S; Q: aq -> bq’; s*>

Пример: HTTP запрос на TCP клиенте

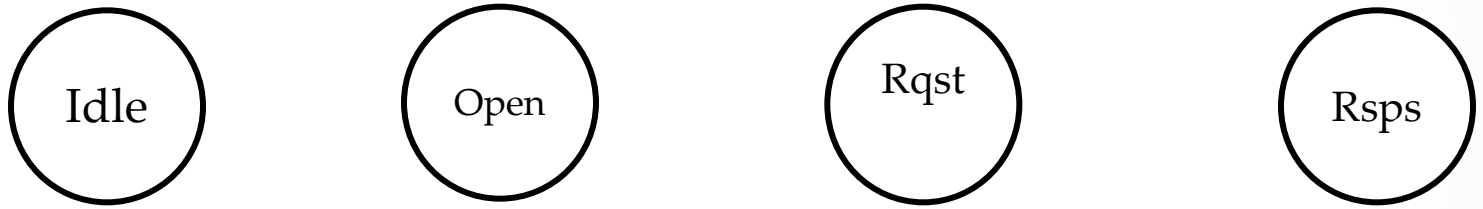


Диаграмма состояний TCP автомата

CLOSED

LISTEN
Сервер ожидает запросов установления соединения от клиента

SYN-RECEIVED
Сервер получил запрос на соединение, отправил ответный запрос и ожидает подтверждения

SYN-SENT
Клиент отправил запрос серверу на установление соединения и ожидает ответа

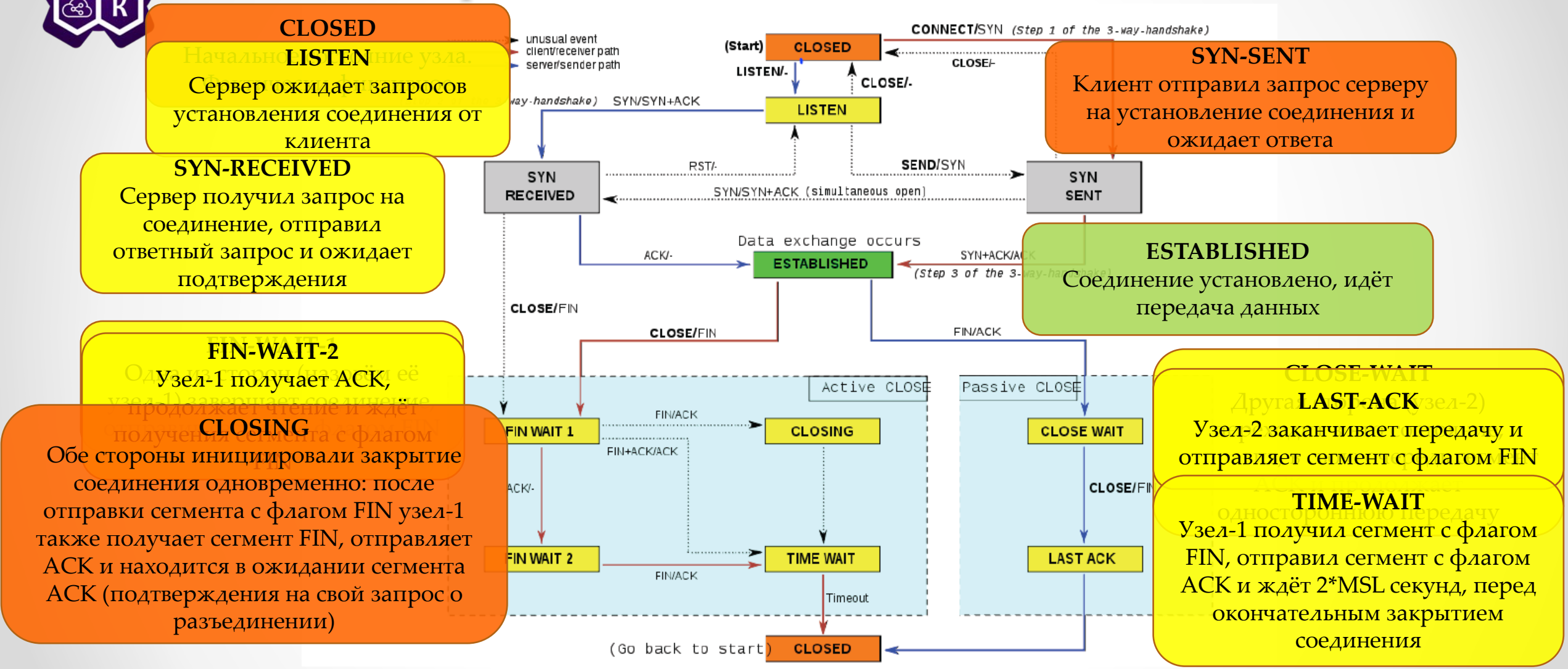
ESTABLISHED
Соединение установлено, идёт передача данных

FIN-WAIT-2
Узел-1 получает ACK,

CLOSING
Обе стороны инициировали закрытие соединения одновременно: после отправки сегмента с флагом FIN узел-1 также получает сегмент FIN, отправляет ACK и находится в ожидании сегмента ACK (подтверждения на свой запрос о разъединении)

LAST-ACK
Узел-2 заканчивает передачу и отправляет сегмент с флагом FIN

TIME-WAIT
Узел-1 получил сегмент с флагом FIN, отправил сегмент с флагом ACK и ждёт 2*MSL секунд, перед окончательным закрытием соединения



http://en.wikipedia.org/wiki/File:Tcp_state_diagram_fixed.svg

Интернет: управление потоком

Введение в компьютерные сети

чл.-корр. РАН Смелянский Р.Л.
Кафедра АСВК
ф-т ВМК МГУ

Управление потоком

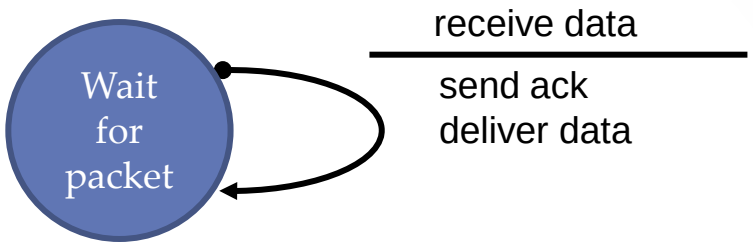
- *Не посылать пакетов больше, чем может принять получатель*
- *Есть обратная связь между отправителем и получателем*
- *Два основных подхода:*
  - *Stop and Wait*
  - *Скользящее окно (Sliding Window)*

Управление потоком (stop and wait)

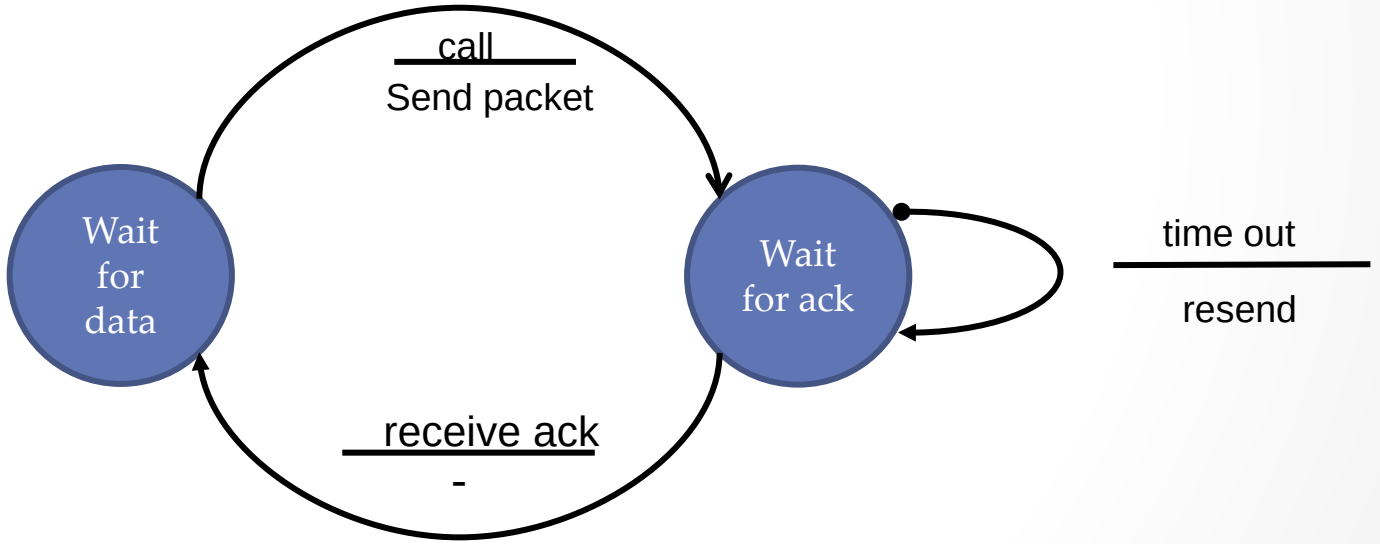
- В одно и то же время передают не более одного пакета
- Sender отправляет пакет
- Receiver посылает пакет с ack , когда получает пакет данных
- Получив ack, sender шлет новый пакет с данными
- По time_out, sender повторно посылает пакет с данными
- Счетчик на 1 бит позволяет выявлять дублирование

Управление потоком (Stop and Wait)

Receiver FSM



Sender FSM



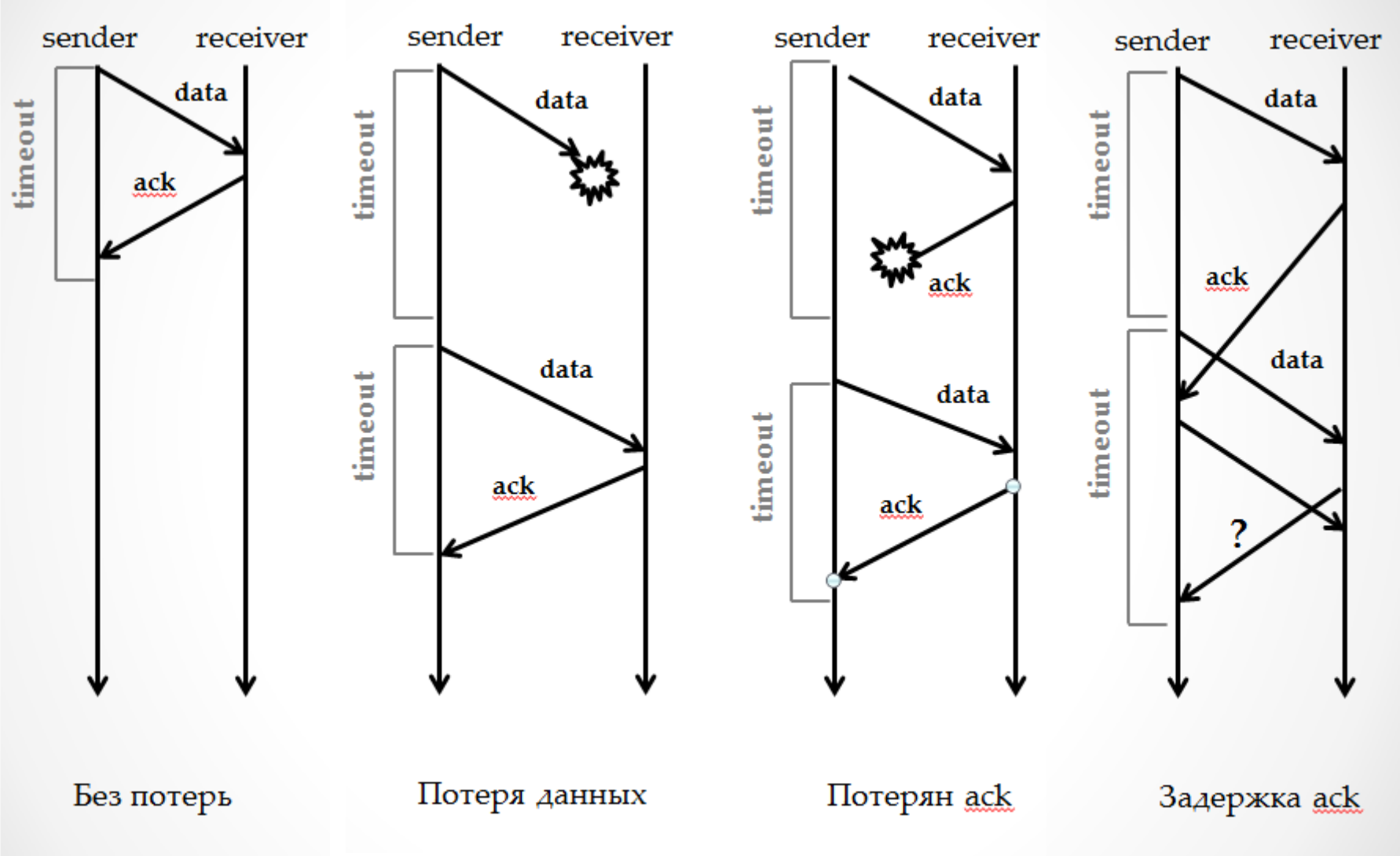
Управление потоком (т.1 стр. 122-128)

```
#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
```

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

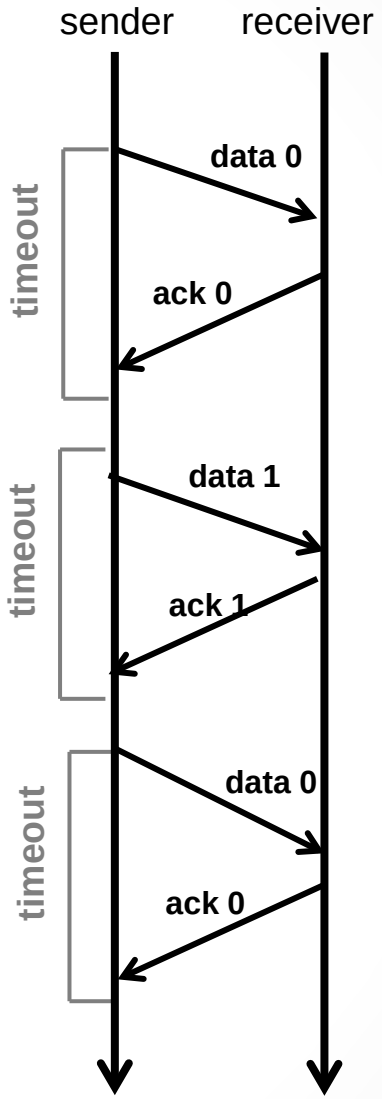
```
/* Protocol 2 (stop-and-wait) also provides for a one-directional flow of data from
   sender to receiver. The communication channel is once again assumed to be error
   free, as in protocol 1. However, this time, the receiver has only a finite buffer
   capacity and a finite processing speed, so 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      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);         send a packet with ack         rival */
      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 */
  }
}
```

Управление потоком



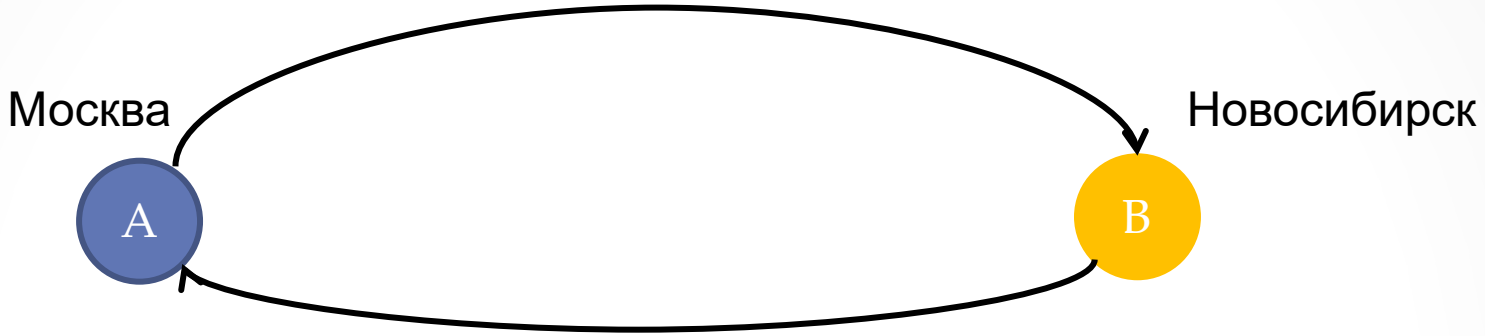
Дублирование

- *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*



Интернет: управление потоком (Sliding Window)

Проблема

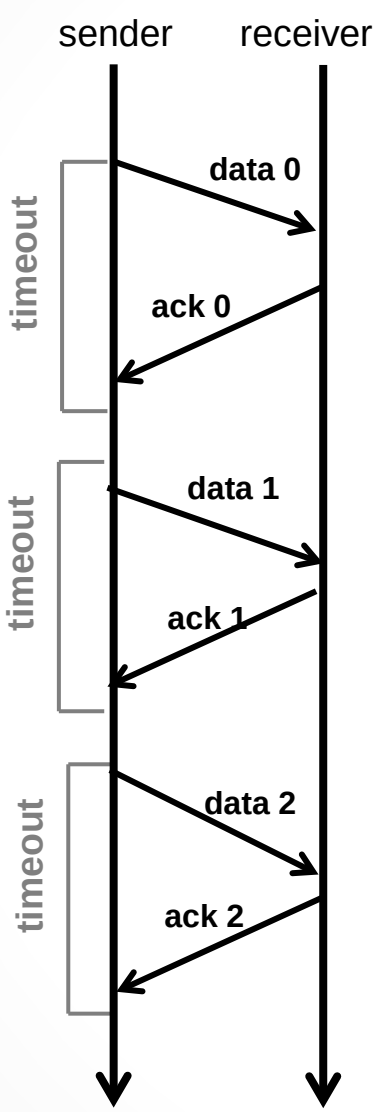


10 Мб/с
RTT = 50 mc

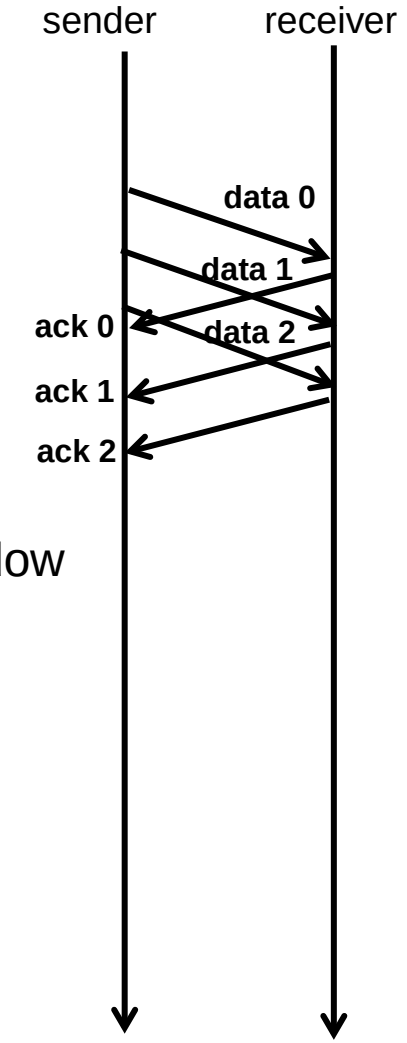
Макс. пропускная способность 10Мб/с
RTT – 50mc

Сделаем обобщение S&W протокола:
- *Разрешаем использовать сразу несколько неподтвержденных сегментов*
- *Максимальное число таких сегментов – окно*
- *Можем плотно «забить» канал*

Пример



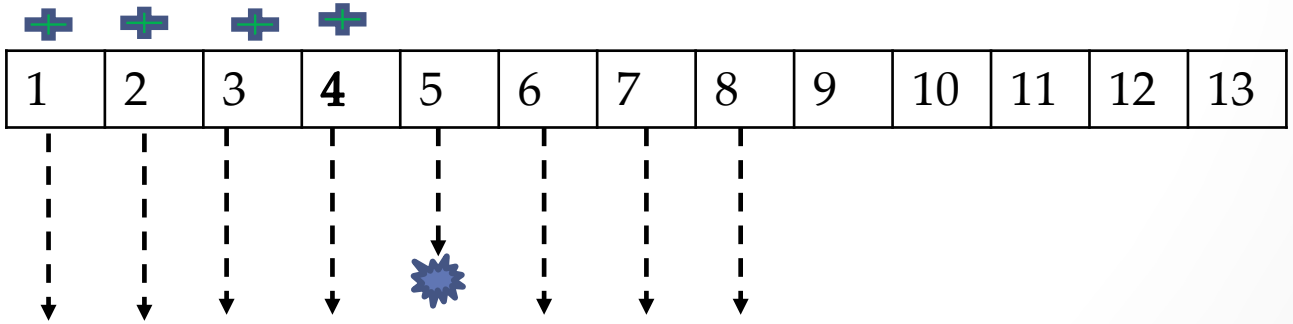
Sliding Window



SW Sender

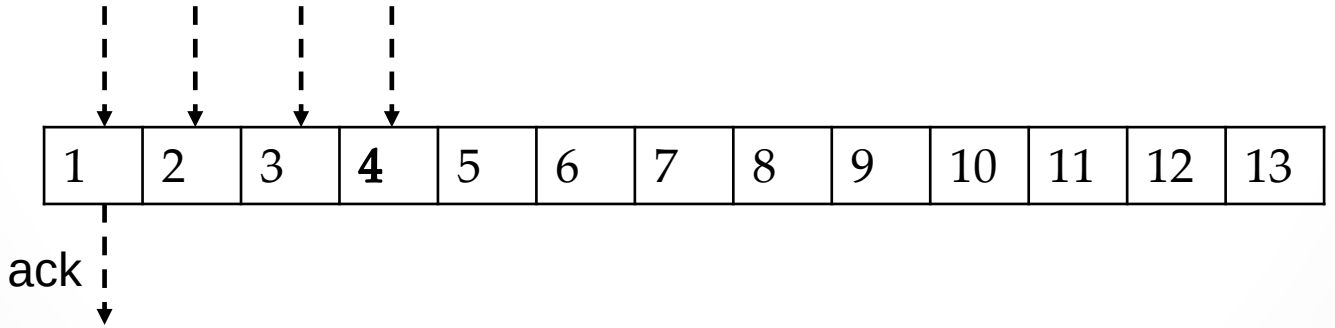
- *У каждого сегмента есть последовательный номер*
- *Поддерживаются 3 переменных*
  - *Размер окна отправки (SWS)*
  - *Последнее полученное от получателя подтверждение (LAR)*
  - *Последний отправленный сегмент (LSS)*
- *Всегда (LSS – LAR) ≤ SWS*
- *LAR возрастает при каждом новом подтверждении*
- *Буфер на SWS сегментов*

SWS = 4



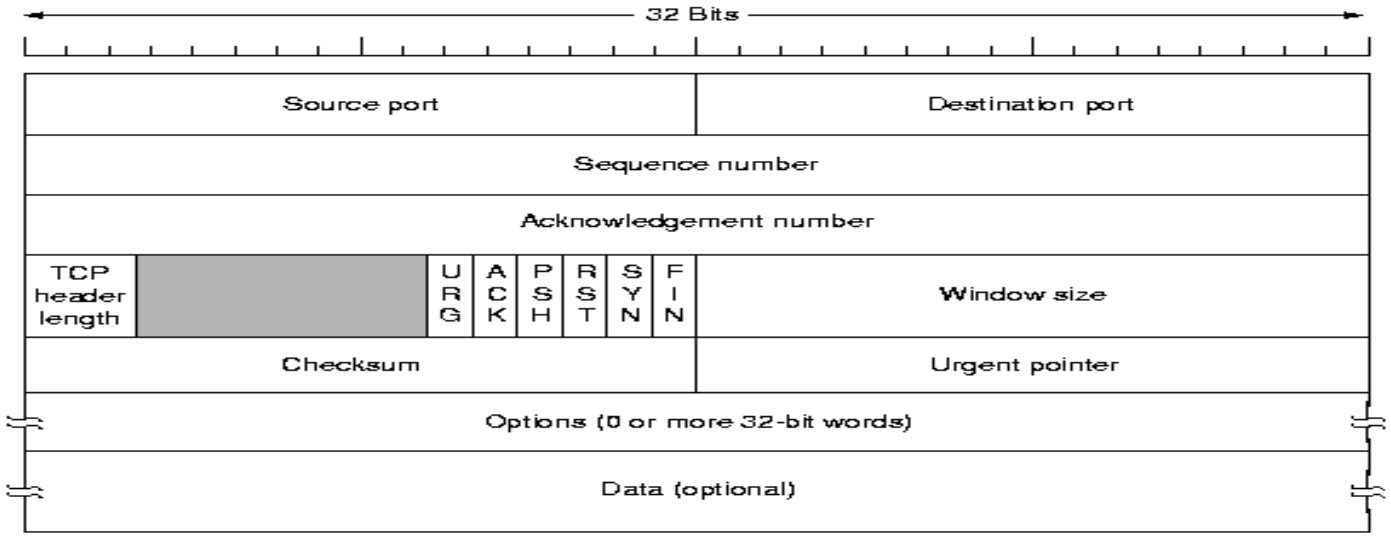
SW Receiver

- *Поддерживаются 3 переменных*
  - *Размер окна получения (SWR)*
  - *Наибольший допустимый номер сегмента (LAS)*
  - *Последний полученный сегмент (LRS)*
- *Всегда (LAS – LRS) ≤ SWR*
- *Если номер полученного сегмента < LAS, то шли подтверждение*
  - *Накопительный ack: если получены 1,2,3,5 – подтверждаем 3*
  - *TCP в уведомлении подтверждает номер ожидаемого сегмента (т.е. 4 в предыдущем примере)*

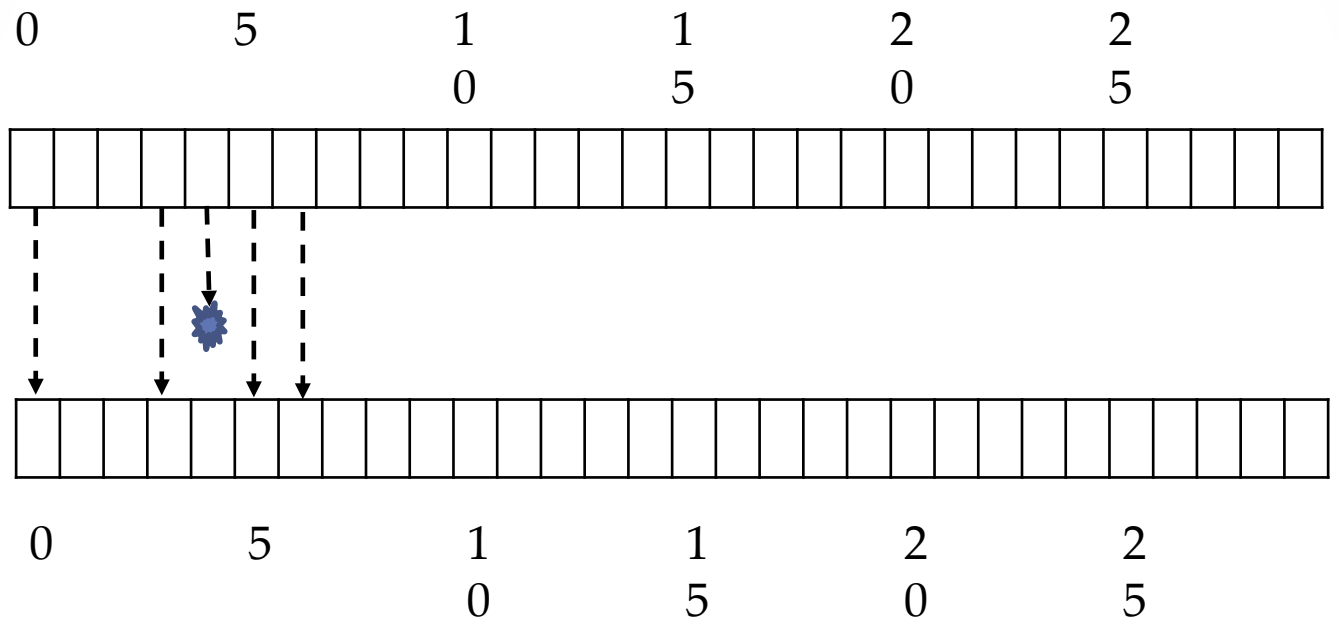


Управление потоком в TCP

- Receiver информирует о размере SWR через поле Window в TCP заголовке
- Sender может посылать данные с номерами не больше LAR+ window



SW пример



Управление потоком с SW

- *Допускает в окне только пакеты в пути, т.е. неподтвержденные*
- *Как только пришло уведомление, окно сдвигается*
- *Необходимое пространство последовательных номеров зависит от размера окна (поле window)*
- *Необходимо согласовать размеры SWS и SWR*

Интернет: модель ICMP сервисов

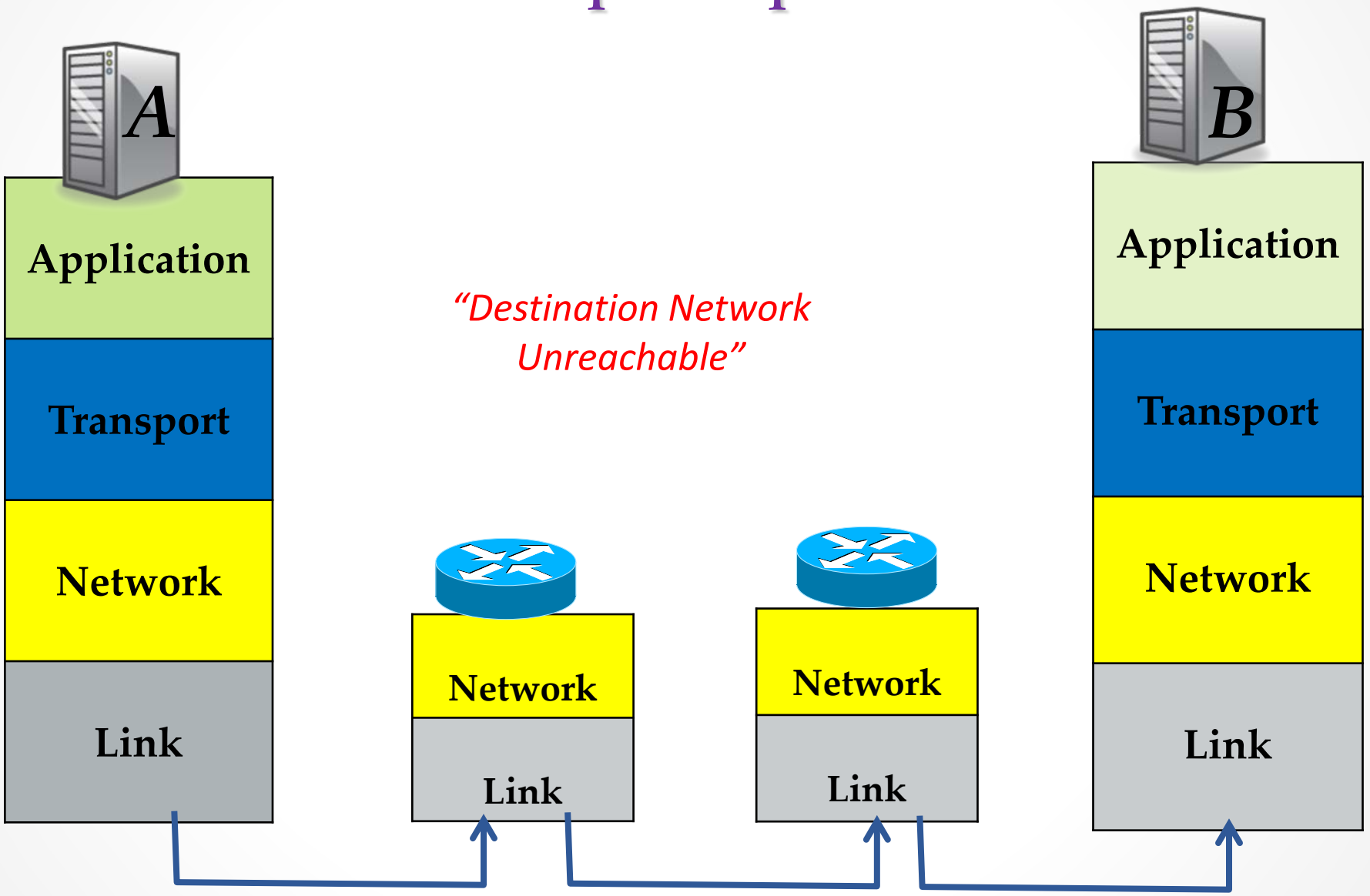
Введение в компьютерные сети

чл.-корр. РАН, проф. Смелянский Р.Л.
Кафедра АСВК
ф-т ВМК МГУ

ICMP протокол

- Предназначен для поддержки IP протокола
- IP протокол
  - Создает IP дейтаграммы
  - Передает их hop-by-hop
  - Формирует таблицы маршрутизации
  - Алгоритмы распространения таких таблиц
- ICMP протокол
  - Обеспечивает коммуникацию между сетевыми уровнями хостов и маршрутизаторов
  - Сообщения об ошибках
  - Диагностика проблем

Пример


A
Application
Transport
Network
Link
Network
Link
Network
Link
B
Application
Transport
Network
Link
"Destination Network Unreachable"

Модель ICMP сервиса

Свойство	Поведение
Сообщение о состоянии	*Самодостаточное сообщение об ошибке или исключительном состоянии*
Ненадежный	*Сервис без соединения и подтверждения*

Примеры типов и кодов ICMP сообщений

ICMP тип	ICMP код	Назначение
0	0	Echo Reply (ping)
3	0	Destination Network Unreachable
3	1	Destination Host Unreachable
3	3	Destination Port Unreachable
8	0	Echo Request (ping)
11	0	TTL Expired (traceroute)

RFC 792

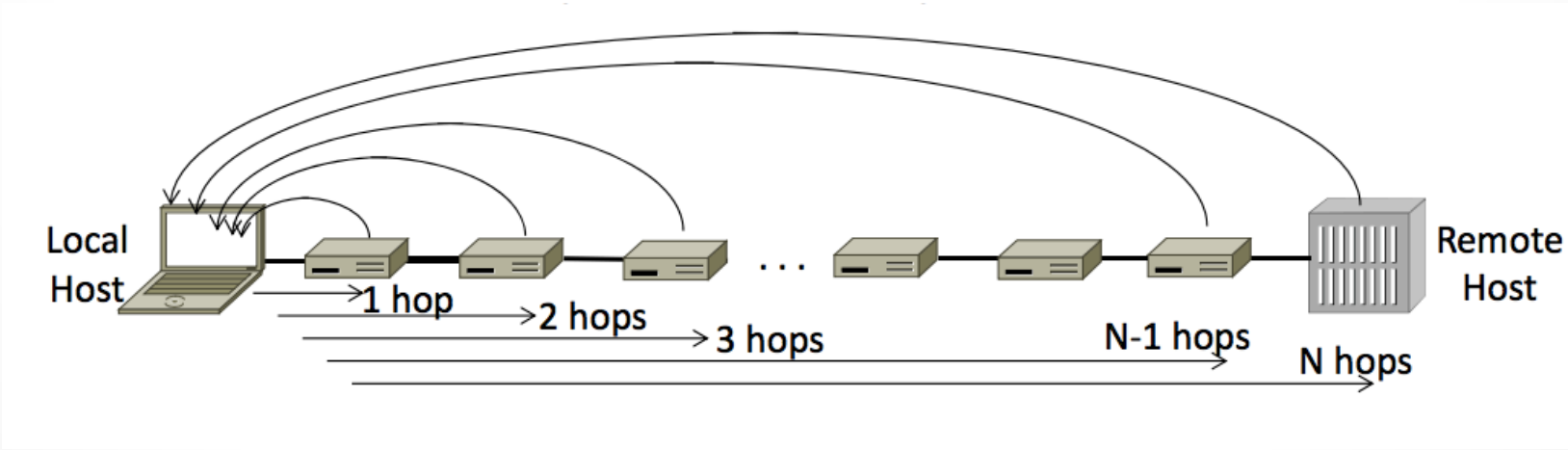
Структура заголовка ICMP сообщения

Заголовок ICMP-сообщения состоит из 8 байт:

- тип (1 байт) – числовой идентификатор типа сообщения:
  - 0 или 8, где 0 - **ICMP reply** (ответ), 8 - **ICMP request** (запрос);
- код (1 байт) – числовой идентификатор, точно определяющий тип ошибки
- контрольная сумма (2 байта) – вычисляется для всего ICMP-сообщения
- Оставшиеся 4 байта и поле данных зависит от значений полей типа и кода.

traceroute / tracert

- Отправляет пробные пакеты с TTL=1, увеличивая значение счетчика на каждой итерации
- Сообщения об ошибках ICMP идентифицируют узлы маршрута



Заключение

- ICMP предоставляет информацию о сетевом уровне хостам и маршрутизаторам
- ICMP работает над IP и относится к транспортному уровню
- ping и traceroute реализованы с помощью ICMP