计算机网络期中复习

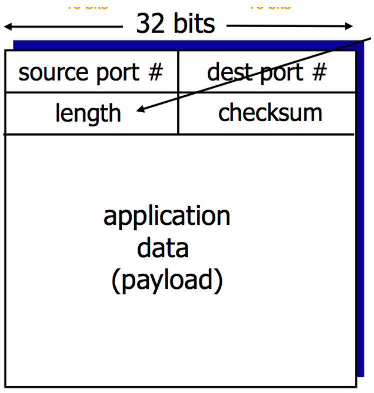
> 一台主机运行多个网络应用进程, 每个进程有一个或多个套接字, 每生成一个套接字，就为他分配一个称为**端口号**的标志符。
>
> - UDP套接字由一个二元组全面标识（目的IP + 目的端口号）
> - TCP套接字由一个四元组全面标识（源IP + 源端口号 + 目的IP + 目的端口号）

CH3

3.3 Connectionless transport: UDP

UDP: User Datagram Protocol

1. Connectionless:
    - no handshaking between UDP sender and receiver(不需要任何准备即可进行数据传输)
    - each UDP segment handled independently of others
2. UDP is used in:
    - streaming multimedia 流式多媒体
    - DNS
    - SNMP 简单网络管理协议
3. UDP是可以实现可靠数据传输的：
    - add reliability at application layer
    - Application-specific error recovery
4. UDP 报文段结构

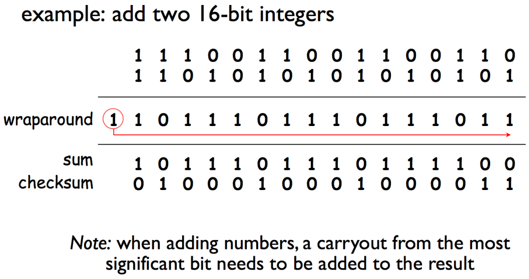


UDP segment format

- length: in bytes of UDP segment, including header

    > **why there is a UDP?**
    >
    > - no connection establishment(which can add delay)
    > - Simple: no connection state at sender, receive ❓
    > - small header size(8 bytes *TCP has 20 bytes header*)
    > - no congestion control ❓

5. UDP checksum:



example: add two 16-bit integers

```
              1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
              1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

wraparound  1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1

       sum    1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
  checksum    0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1
```

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

UDP对报文段中的所有16bit字进行加和，然后再将和进行反码运算，求和时遇到的所有溢出都被回卷。

3.4 Principles of reliable data transfer

1. rdt: Reliable data transfer

2. rdt 1.0: reliable transfer over a reliable channel

- just send & receive

3. rdt 2.0: channel with bit errors

    - error detection: checksum

    - feedback: control msgs(ACK, NAK) from receiver to sender

        - (Acknowledgements)ACK: tell sender that packet received ok
        - (negative acknowledgements)NAK: tell sender that packet has errors
        - sender retransmits pkt on receipt of NAK

    - rdt 2.0 has a **fatal flaw:**

        如果ACK/NAK的传输中断了，那么sender重新transmit是不行的，因为会产生duplicate。

4. rdt 2.1

    handling duplicate的方法:

    i. sender retransmits current pkt if ACK/NAK corrupted

    ii. sender adds sequence number to each pkt

    iii. receiver discards duplicate pkt

    ❗ *Notes*:

    - For sender:

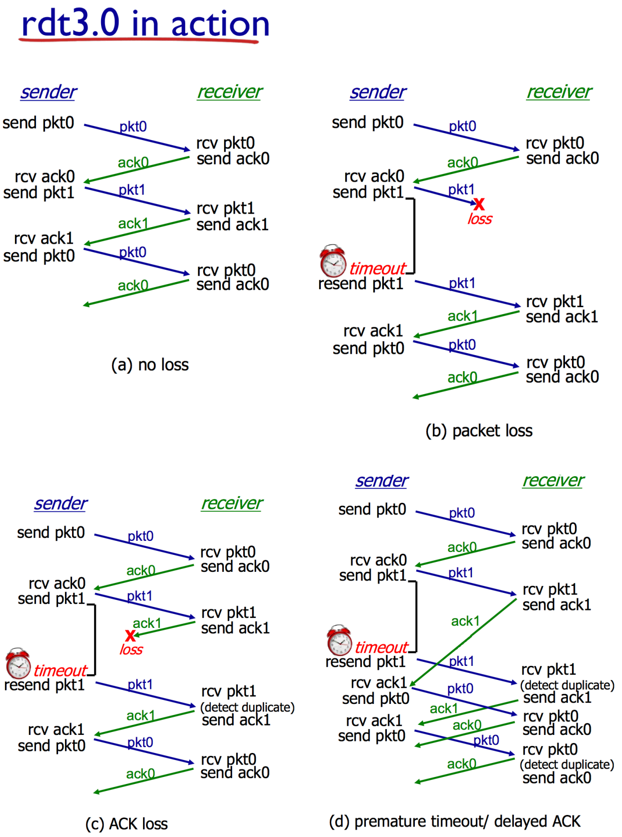
        - two seq # (0,1) added to pkt will suffice
        - must check if received ACK/NAK corrupted
        - States must "remember" whether "expected" pkt should have seq # of 0 or 1

    - For receiver:

        - must check if received pkt is duplicate

        - Receiver cannot know if ACK/NAK received OK at sender

5. rdt 2.2: use ACKs only
6. rdt 3.0: channels with errors and loss



(a) no loss

(b) packet loss

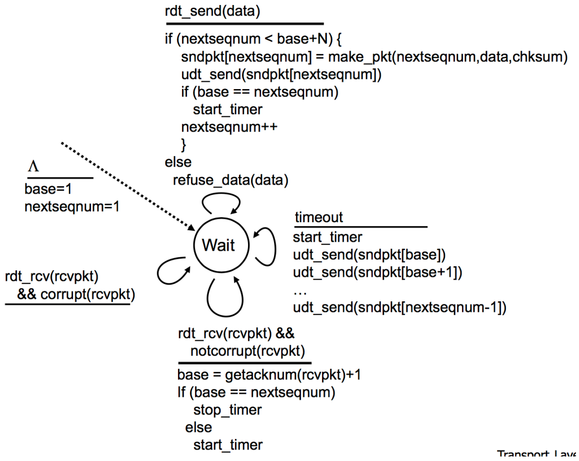
(c) ACK loss

(d) premature timeout/ delayed ACK

7. rdt 3.1: pipelined protocols

(1). Go-Back-N

Sender:



```
rdt_send(data)
_______________
if (nextseqnum < base+N) {
    sndpkt[nextseqnum] = make_pkt(nextseqnum,data,chksum)
    udt_send(sndpkt[nextseqnum])
    if (base == nextseqnum)
      start_timer
    nextseqnum++
    }
else
  refuse_data(data)
```

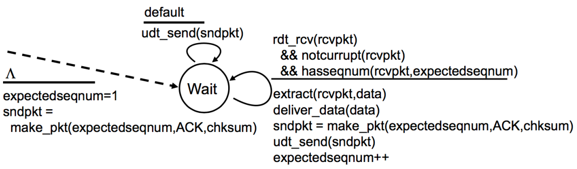
```
Λ
_______________
base=1
nextseqnum=1
```

```
timeout
_______________
start_timer
udt_send(sndpkt[base])
udt_send(sndpkt[base+1])
…
udt_send(sndpkt[nextseqnum-1])
```

```
rdt_rcv(rcvpkt)
  && corrupt(rcvpkt)
_______________
```

```
rdt_rcv(rcvpkt) &&
  notcorrupt(rcvpkt)
_______________
base = getacknum(rcvpkt)+1
If (base == nextseqnum)
    stop_timer
  else
    start_timer
```

Receiver:



```
default
_______________
udt_send(sndpkt)
```

```
Λ
_______________
expectedseqnum=1
sndpkt =
  make_pkt(expectedseqnum,ACK,chksum)
```

```
rdt_rcv(rcvpkt)
  && notcurrupt(rcvpkt)
  && hasseqnum(rcvpkt,expectedseqnum)
_______________
extract(rcvpkt,data)
deliver_data(data)
sndpkt = make_pkt(expectedseqnum,ACK,chksum)
udt_send(sndpkt)
expectedseqnum++
```

  - 优点：receiver 不需要缓存任何失序分组，接收方需要维护的唯一信息就是下一个按序接收的分组的序号，即FSM中的 `expectedseqnum`.
  - 缺点：会丢弃已经正确接收的分组。对该分组的重传也许会丢失或出错，因此甚至需要更多的重传。也会加大对整个网络的压力。

(2). Selective repeat

  - 仅重传那些它怀疑在接收方出错的分组
  - 每个分组拥有自己的定时器，超时发生后只能发送一个分组

3.5 Connection-oriented transport: TCP

> **TCP & UDP协议只在端系统中运行！** 🌓

1. TCP 连接服务的特点：
   - Full-duplex service: 若一台主机上的进程A与另一台主机上的进程B存在一条TCP连接，则应用层的数据可以从进程B流向进程A，也可从进程A流向进程B。
   - Point-to-point: 单个发送方与单个接收方之间的连接

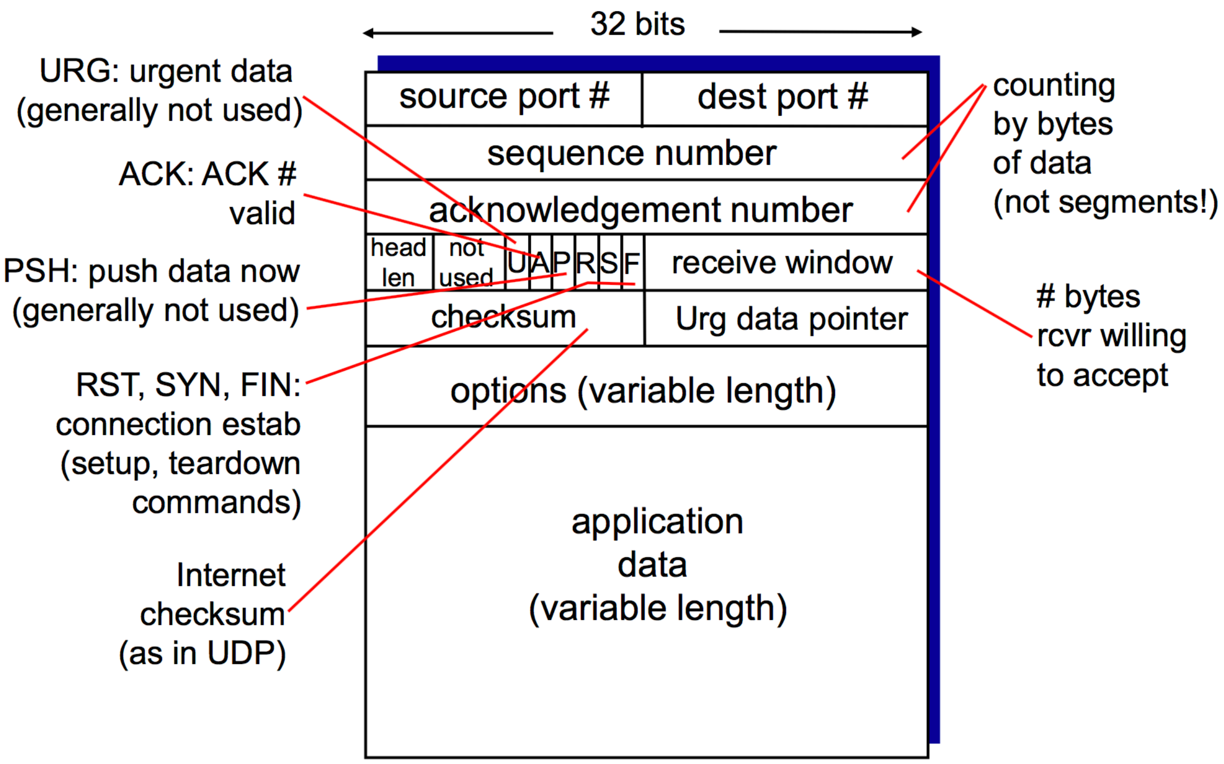
`clientSocket.connect((serverName, serverPort))`

三次握手：

- client首先发送一个特殊的报文段
- server用另一个特殊的TCP报文段来响应
- 最后，client再用第三个特殊的报文段作为响应

Notes: 前两个报文段不包含应用层数据，第三个可以包含。

2. TCP报文段结构（TCP segment）


32 bits
source port #
dest port #
sequence number
acknowledgement number
head len
not used
U
A
P
R
S
F
receive window
checksum
Urg data pointer
options (variable length)
application data (variable length)
URG: urgent data (generally not used)
ACK: ACK # valid
PSH: push data now (generally not used)
RST, SYN, FIN: connection estab (setup, teardown commands)
Internet checksum (as in UDP)
counting by bytes of data (not segments!)
# bytes rcvr willing to accept

- 报文段序号(sequence number for a segment)：是该报文段**首字节的字节流编号**

  （即我发送的首字节的序号）
- 确认号(acknowledgement number): 是我想要的下一字节的序号

TCP提供**累计确认**，即TCP只确认该流中至第一个丢失字节为止的字节 **——** cumulative ACK

问：当主机在一条TCP连接中收到失序报文段时该怎么办?

答：取决于implementer （1）接收方立即丢弃失序报文段（2）接收方保留失序的字节，并等待缺少的字节以填补该间隙