为应用层使用网络提供了抽象

The Transport Layer

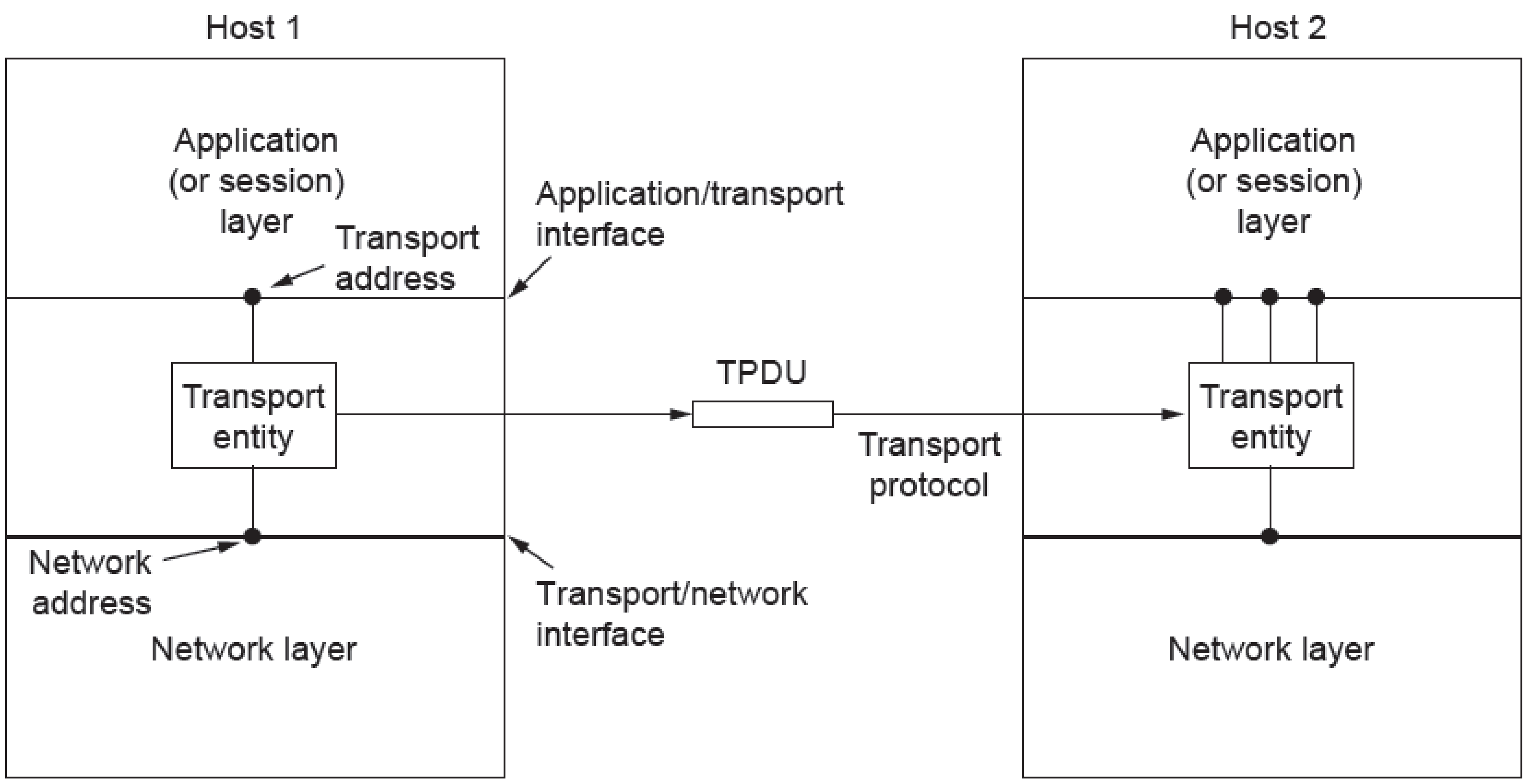
Chapter 6

- The transport layer is responsible for completing the services of the underlying network to the extent that application development can take place

6.1 Transport Service

- Upper Layer Services
- Transport Service Primitives
- Berkeley Sockets
- Example of Socket Programming: Internet File Server

Services Provided to the Upper Layers



The network, transport, and application layers

Services Provided to the Upper Layers

- **Services Provided to the Upper Layer**
  - provide reliable connection-oriented services
  - provide unreliable connectionless services
- **Important:** we’re talking about efficient and cost-effective services, in particular reliable connections..

为什么不和网
络层合并呢？

位置不一样，主机，路由器。
用户对网络层没有控制权

Services Provided to the Upper Layers

- **Consequence:** If we want to develop applications that are independent of the particular services offered by a carrier, we'll have to design a standard communication interface and implement that interface at the client's sites. The transport layer contains such implementations.

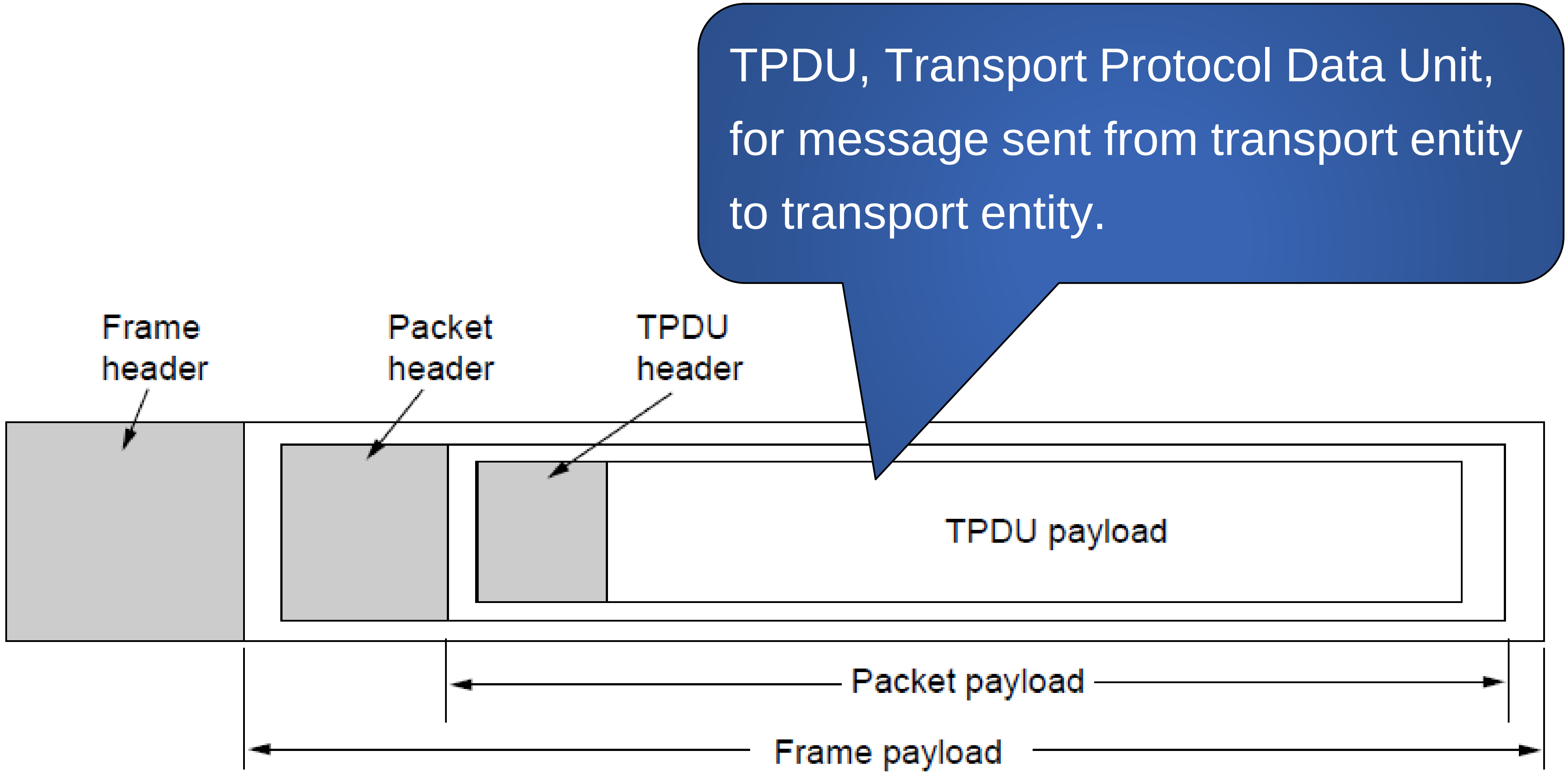
Transport Service Primitives (1)

- The transport service primitives allow transport users (e.g., application programs) to access the transport service. Each transport service has its own access primitives.
- Allows application programs to establish, use, and release connections.

Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

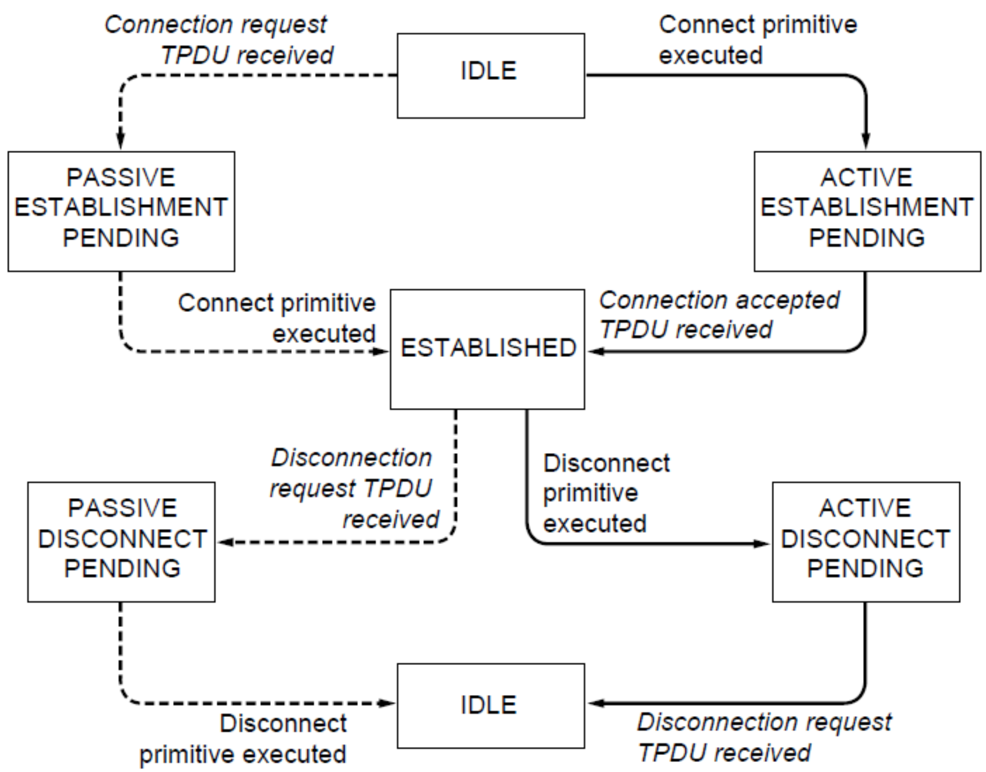
The primitives for a simple transport service

Transport Service Primitives (2)



Nesting of TPDUs, packets, and frames.

Berkeley Sockets (1)



A state diagram for a simple connection management scheme. Transitions labeled in italics are caused by packet arrivals. The solid lines show the client's state sequence. The dashed lines show the server's state sequence.

Berkeley Sockets (2)

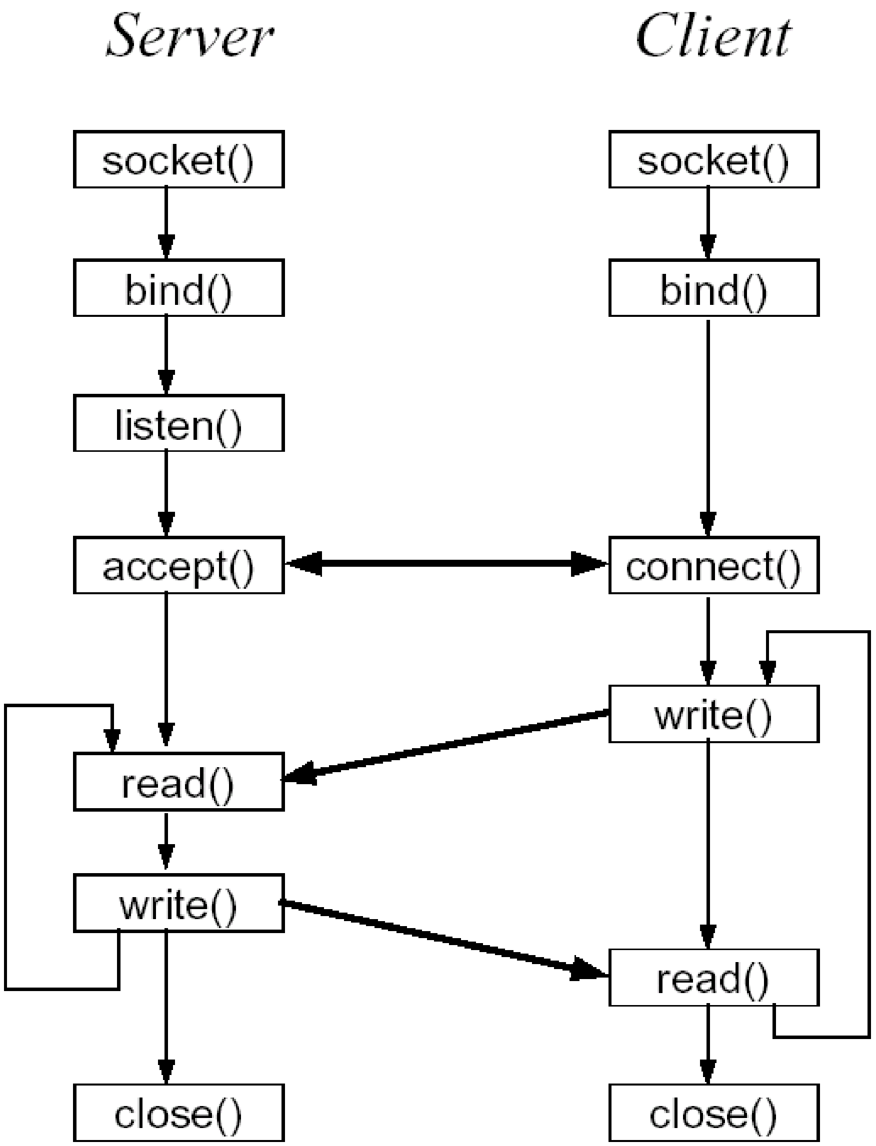
- **Example:** Consider the **Berkeley socket interface**, which has been adopted by most UNIX systems, as well as Windows 9X/NT:

分配表空间，明确地址格式、服务类型、协议

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Associate a local address with a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Passively establish an incoming connection
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

The socket primitives for TCP

Berkeley Sockets (3)



Example of Socket Programming: An Internet File Server (1)

```
/* This page contains a client program that can request a file from the server program
 * on the next page. The server responds by sending the whole file.
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345                /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096                    /* block transfer size */

int main(int argc, char **argv)
{
  int c, s, bytes;
  char buf[BUF_SIZE];                    /* buffer for incoming file */
  struct hostent *h;                     /* info about server */
  struct sockaddr_in channel;            /* holds IP address */

  . . .
```

Client code using sockets

Example of Socket Programming: An Internet File Server (2)

```
. . .

if (argc != 3) fatal("Usage: client server-name file-name");
h = gethostbyname(argv[1]);                    /* look up host's IP address */
if (!h) fatal("gethostbyname failed");

s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s <0) fatal("socket");
memset(&channel, 0, sizeof(channel));
channel.sin_family= AF_INET;
memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
channel.sin_port= htons(SERVER_PORT);

c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
if (c < 0) fatal("connect failed");


 . . .
```

Client code using sockets

Example of Socket Programming: An Internet File Server (3)

```
. . .

  c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
  if (c < 0) fatal("connect failed");

  /* Connection is now established. Send file name including 0 byte at end. */
  write(s, argv[2], strlen(argv[2])+1);

  /* Go get the file and write it to standard output. */
  while (1) {
       bytes = read(s, buf, BUF_SIZE);          /* read from socket */
       if (bytes <= 0) exit(0);                 /* check for end of file */
       write(1, buf, bytes);                    /* write to standard output */
  }
}

fatal(char *string)
{
  printf("%s\n", string);
  exit(1);
}
```

Client code using sockets

Example of Socket Programming: An Internet File Server (4)

```
#include <sys/types.h>                       /* This is the server code */
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345                    /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096                        /* block transfer size */
#define QUEUE_SIZE 10

int main(int argc, char *argv[])
{
  int s, b, l, fd, sa, bytes, on = 1;
  char buf[BUF_SIZE];                        /* buffer for outgoing file */
  struct sockaddr_in channel;                /* holds IP address */

        . . .
```

Server code

Example of Socket Programming: An Internet File Server (5)

```
. . .

/* Build address structure to bind to socket. */
memset(&channel, 0, sizeof(channel));       /* zero channel */
channel.sin_family = AF_INET;
channel.sin_addr.s_addr = htonl(INADDR_ANY);
channel.sin_port = htons(SERVER_PORT);

/* Passive open. Wait for connection. */
s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);    /* create socket */
if (s < 0) fatal("socket failed");
setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));

b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
if (b < 0) fatal("bind failed");

l = listen(s, QUEUE_SIZE);                  /* specify queue size */
if (l < 0) fatal("listen failed");
 . . .
```

Server code

Example of Socket Programming: An Internet File Server (6)

```
 . . .

/* Socket is now set up and bound. Wait for connection and process it. */
while (1) {
    sa = accept(s, 0, 0);                      /* block for connection request */
    if (sa < 0) fatal("accept failed");

    read(sa, buf, BUF_SIZE);                   /* read file name from socket */

    /* Get and return the file. */
    fd = open(buf, O_RDONLY);                  /* open the file to be sent back */
    if (fd < 0) fatal("open failed");

    while (1) {
        bytes = read(fd, buf, BUF_SIZE); /* read from file */
        if (bytes <= 0) break;                 /* check for end of file */
        write(sa, buf, bytes);                 /* write bytes to socket */
    }
    close(fd);                                 /* close file */
    close(sa);                                 /* close connection */
}
}
```

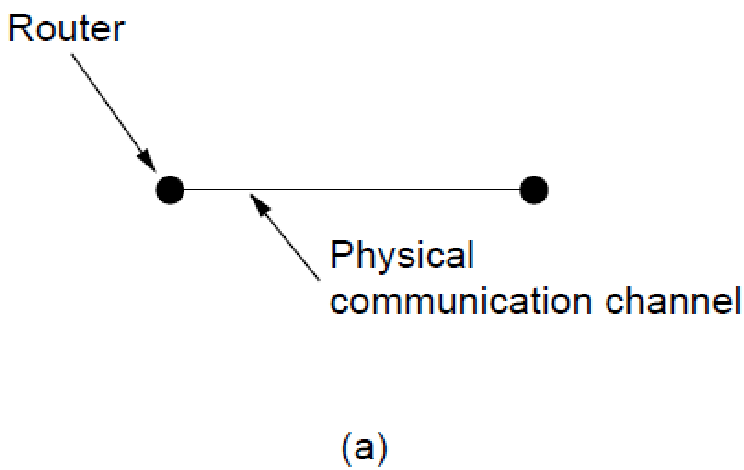
Server code

6.2 Elements of Transport Protocols (1)

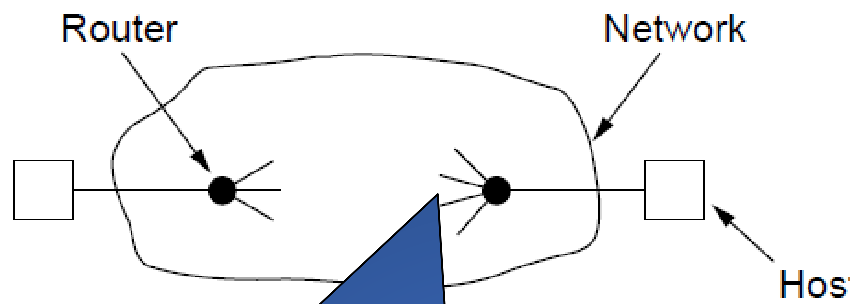
- Addressing
- Connection establishment
  - Problem: Delayed and duplicate packets
  - Solution: Three-way handshake
- Connection release
- Error control and flow control
- Multiplexing
- Crash recovery

Elements of Transport Protocols (2)

区别？



(a)



1） 需要地址

2）连接建立复杂

3）处理网络的延时和乱序

e.g. 处理延时重复包，银行取钱

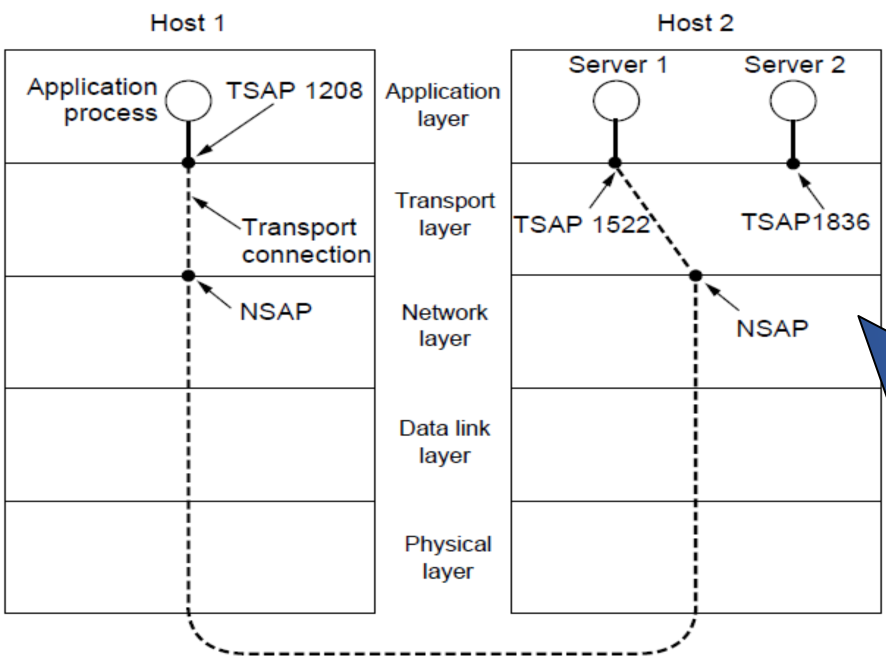
4）大量可变的连接，所以缓冲和流量控制方法不同。

(a) Environment of the data link layer.

(b) Environment of the transport layer.

Addressing (1)

- **Note:** Each layer has its own way of dealing with addresses. a **transport service access point** is **an IP address with a port number**.



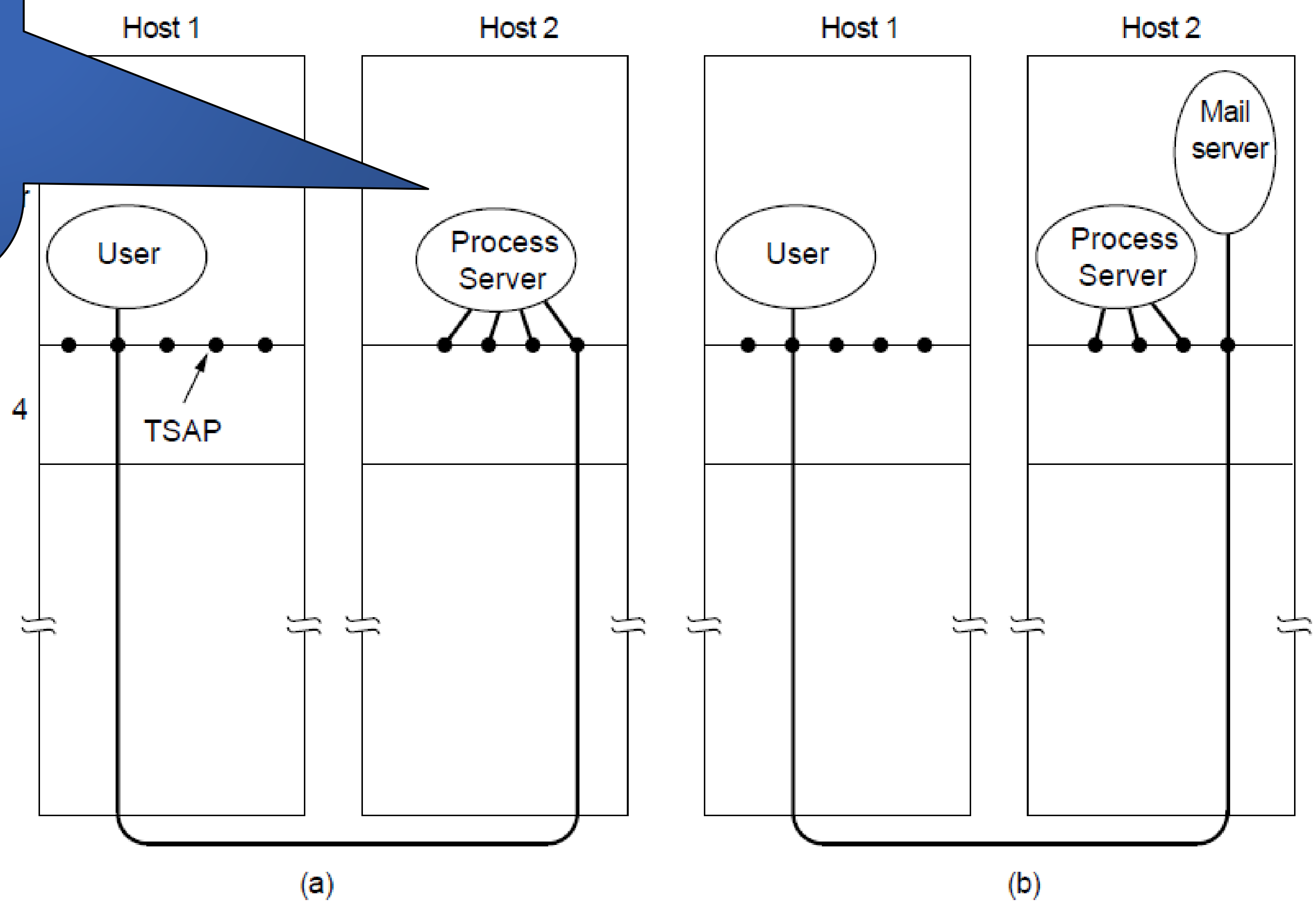
TSAPs, NSAPs, and transport connections

连接问题:

- 端口Well Know。
- 端口映射器（在一个知名端口，新服务要注册）。相当于查号员。

Addressing (2)

多服务器进程效率问题：
初始连接协议：进程服务器充当那些不频繁使用的服务器的代理。同时监听一组端口。



How a user process in host 1 establishes a connection with a mail server in host 2 via a process server.

Connection Establishment (1)

- **Basic idea:** To establish a connection, you send off a connection request to the other end. The other end then accepts the connection, and returns an acknowledgment.
- **Big problem**: Suppose you don’t get an answer, so you do another request.

Connection Establishment (2)

- **Main cause:** The network has storage capabilities, and unpredictable delays. This means that things can pop up out of the blue.

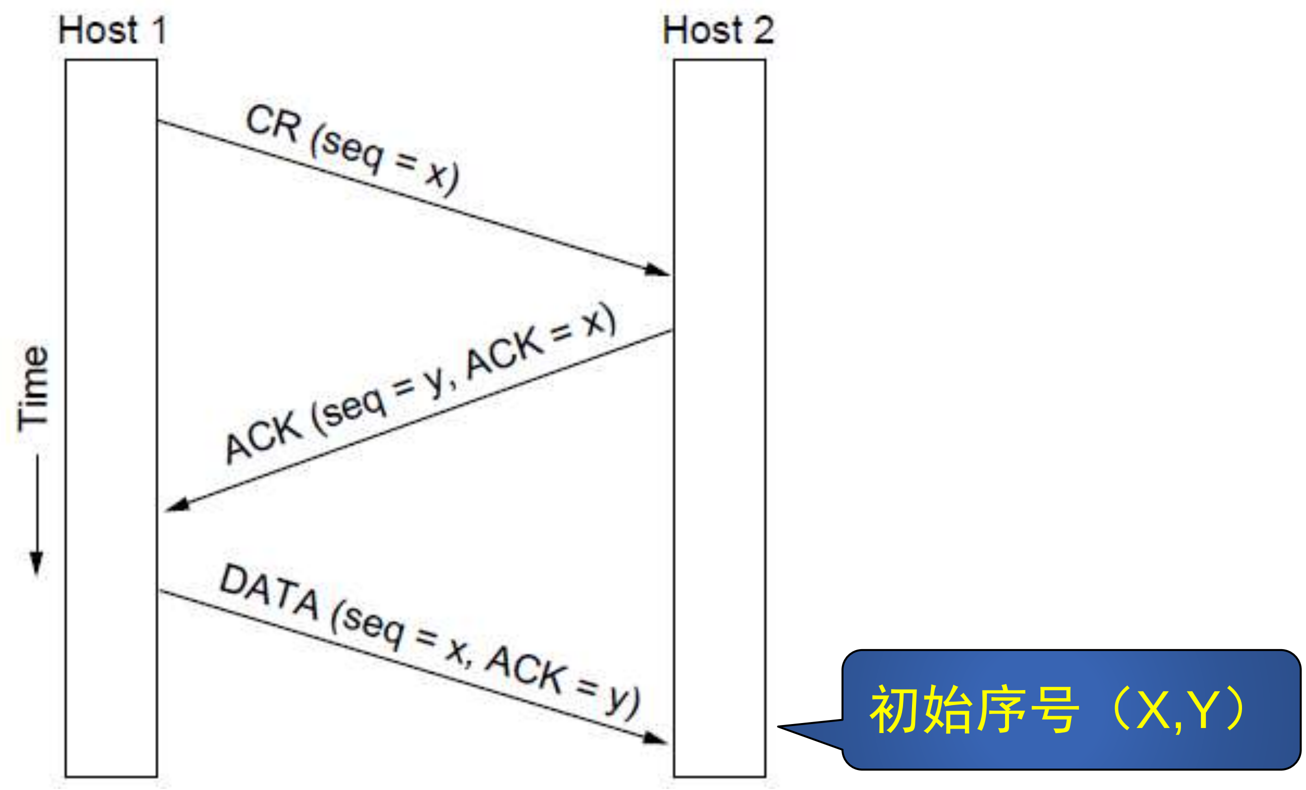
- **Attacking Duplicates**
  - Assign sequence numbers to TPDUs, and let the sequence number space be so large that no two outstanding TPDUs can have the same number.
  - Restrict the lifetime of TPDUs – if the maximum lifetime is known in advance, we can be sure that a previous packet is discarded and that it won't interfere with successive ones.

Connection Establishment (3)

Techniques for restricting packet lifetime

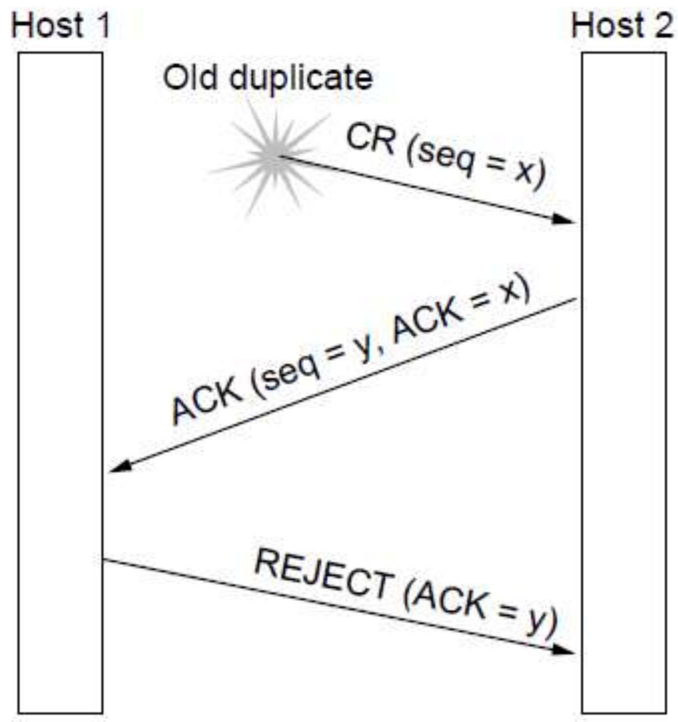
- Restricted network design.
- Putting a hop counter in each packet.
- Timestamping each packet.

Connection Establishment (4)



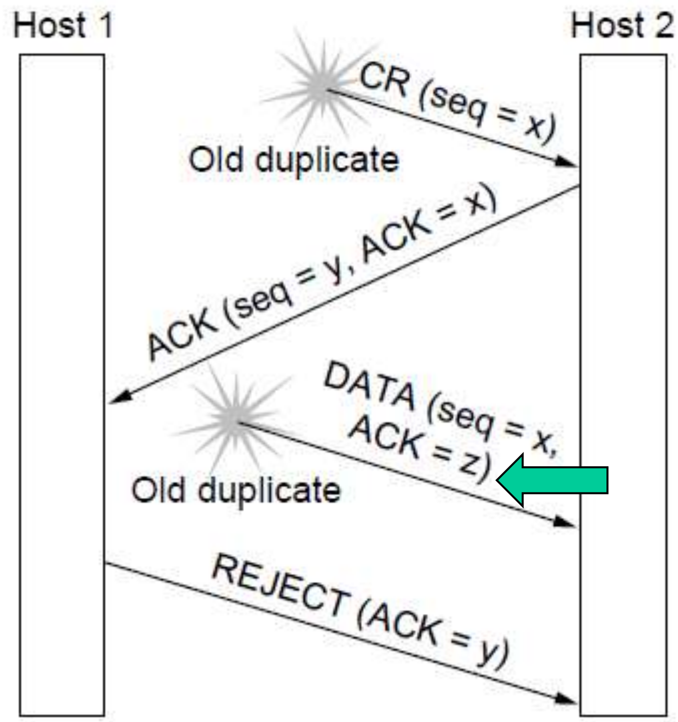
Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST. Normal operation.

Connection Establishment (5)



Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST. Old duplicate CONNECTION REQUEST appearing out of nowhere.

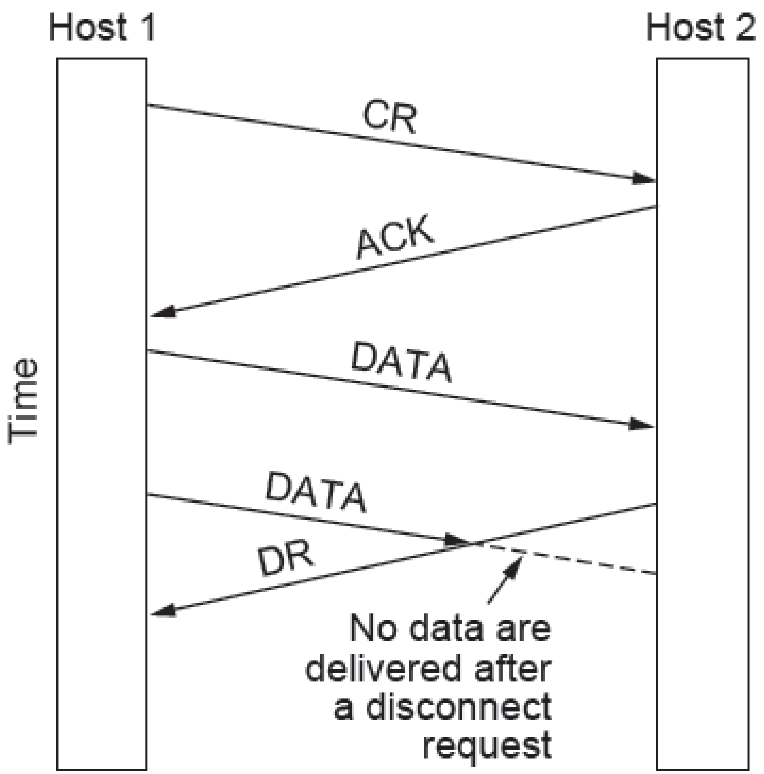
Connection Establishment (6)



Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST. Duplicate CONNECTION REQUEST and duplicate ACK

Connection Release (1)

- 非对称释放：挂电话。
- 对称释放：看成两个独立的连接，发DISCONECT后还可以收数据。



Abrupt disconnection with loss of data

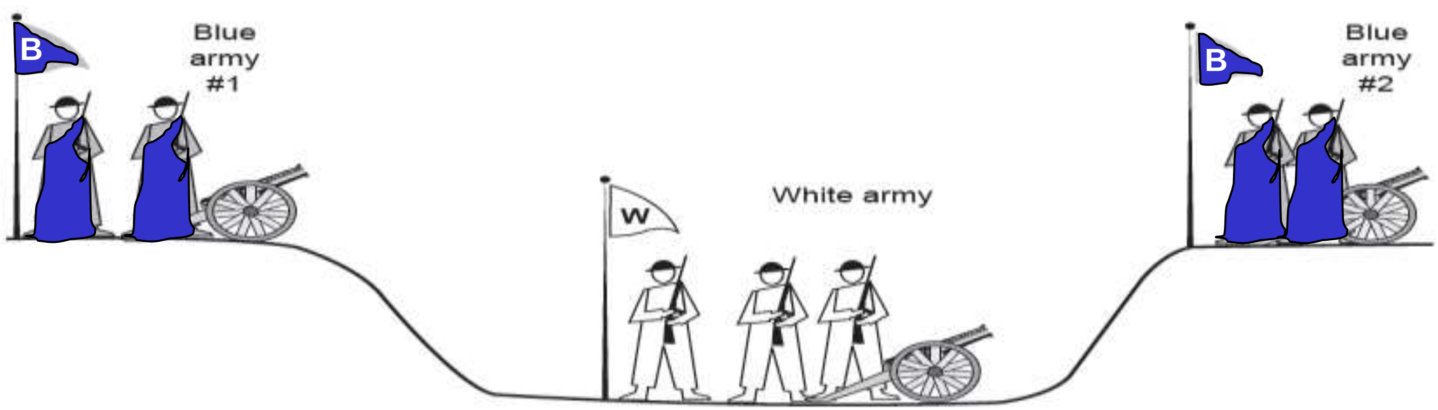
Connection Release (1)

- **Big problem:** Can we devise a solution to release a connection such that the two parties will *always* agree. The answer is simple:

  - **Normal case:** Host 1 sends disconnect request (DR). Host 2 responds with a DR. Host 1 acknowledges, and ACK arrives at host 2.
  - **ACK is lost:** What should host 2 do? It doesn't know for sure that its DR came through.
  - **Host 2's DR is lost:** What should host 1 do? Of course, send another DR, but this brings us back to the normal case. This still means that the ACK sent by host 1 may still get lost.

Connection Release (2)

- **Practical solution:** Use timeout mechanisms. This will catch most cases, but it is never a fool-proof solution: the initial DR and all retransmissions may still be lost, resulting in a **half-open connection**.

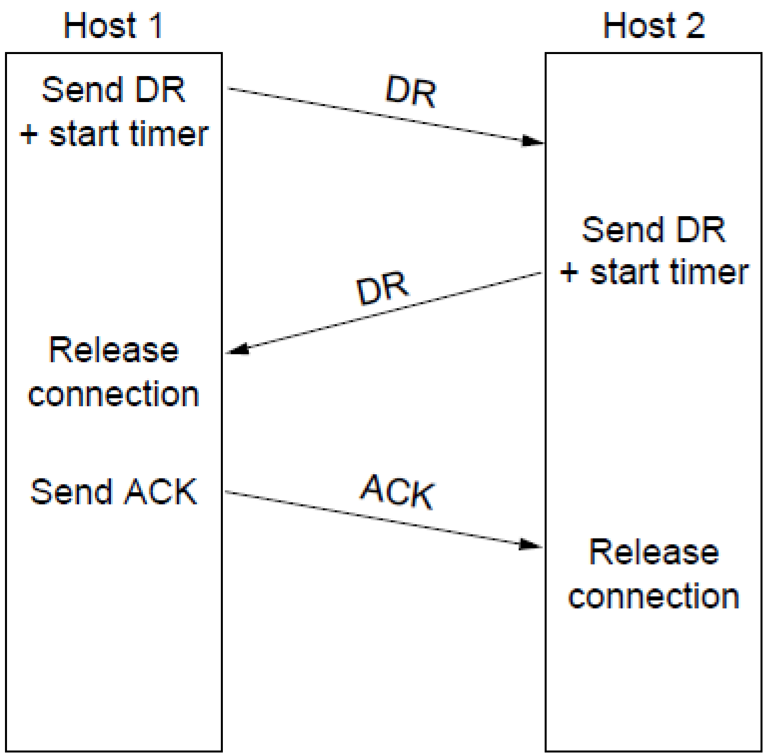


The two-army problem

维度	两军对垒	拜占庭
通信信道	信道不可靠	信道可靠
有无叛徒	无	有

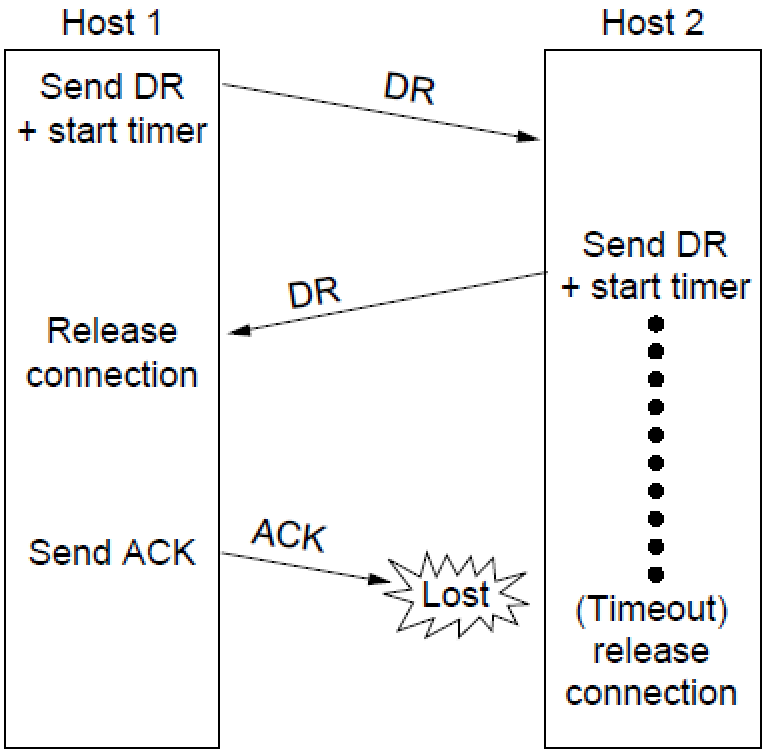
所有忠诚的将军都能够让别的将军接收到自己的真实意图，并最终一致行动。

Connection Release (3)



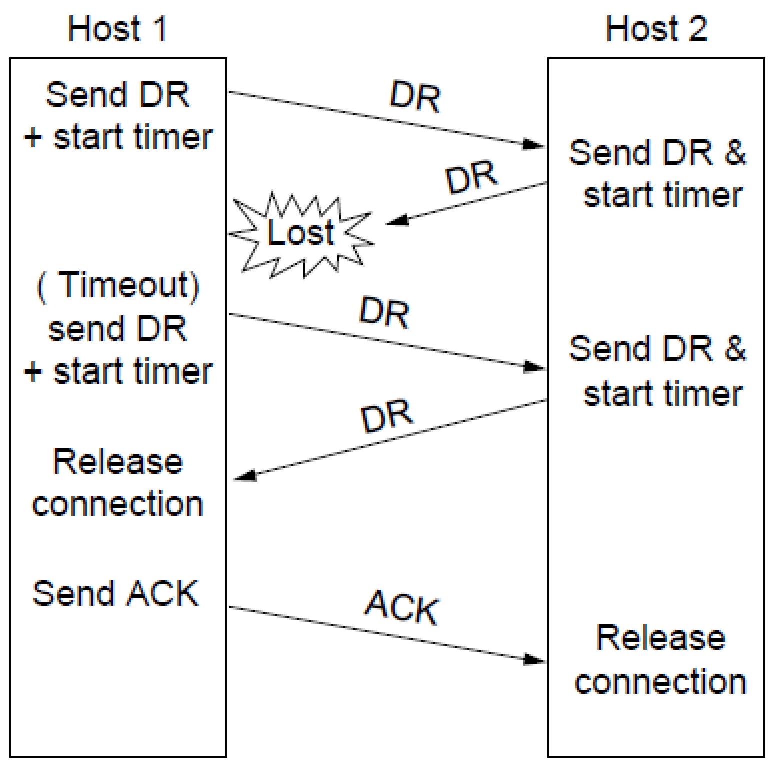
Four protocol scenarios for releasing a connection.
(a) Normal case of three-way handshake

Connection Release (4)



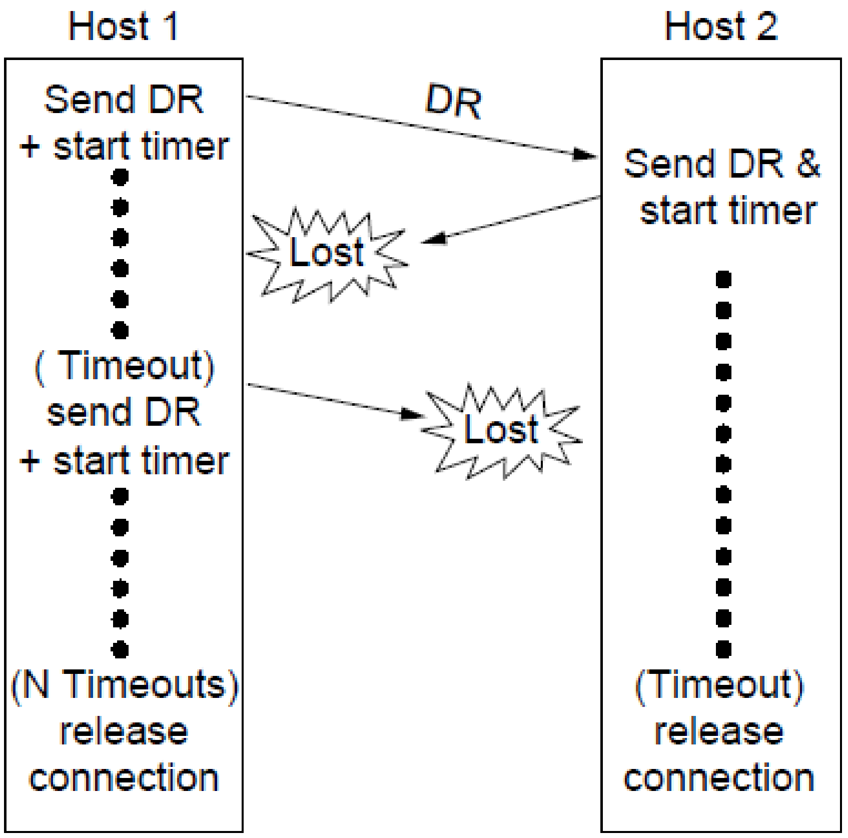
Four protocol scenarios for releasing a connection.
(b) Final ACK lost.

Connection Release (5)



Four protocol scenarios for releasing a connection.
(c) Response lost

Connection Release (6)



Four protocol scenarios for releasing a connection.
(d) Response lost and subsequent DRs lost.

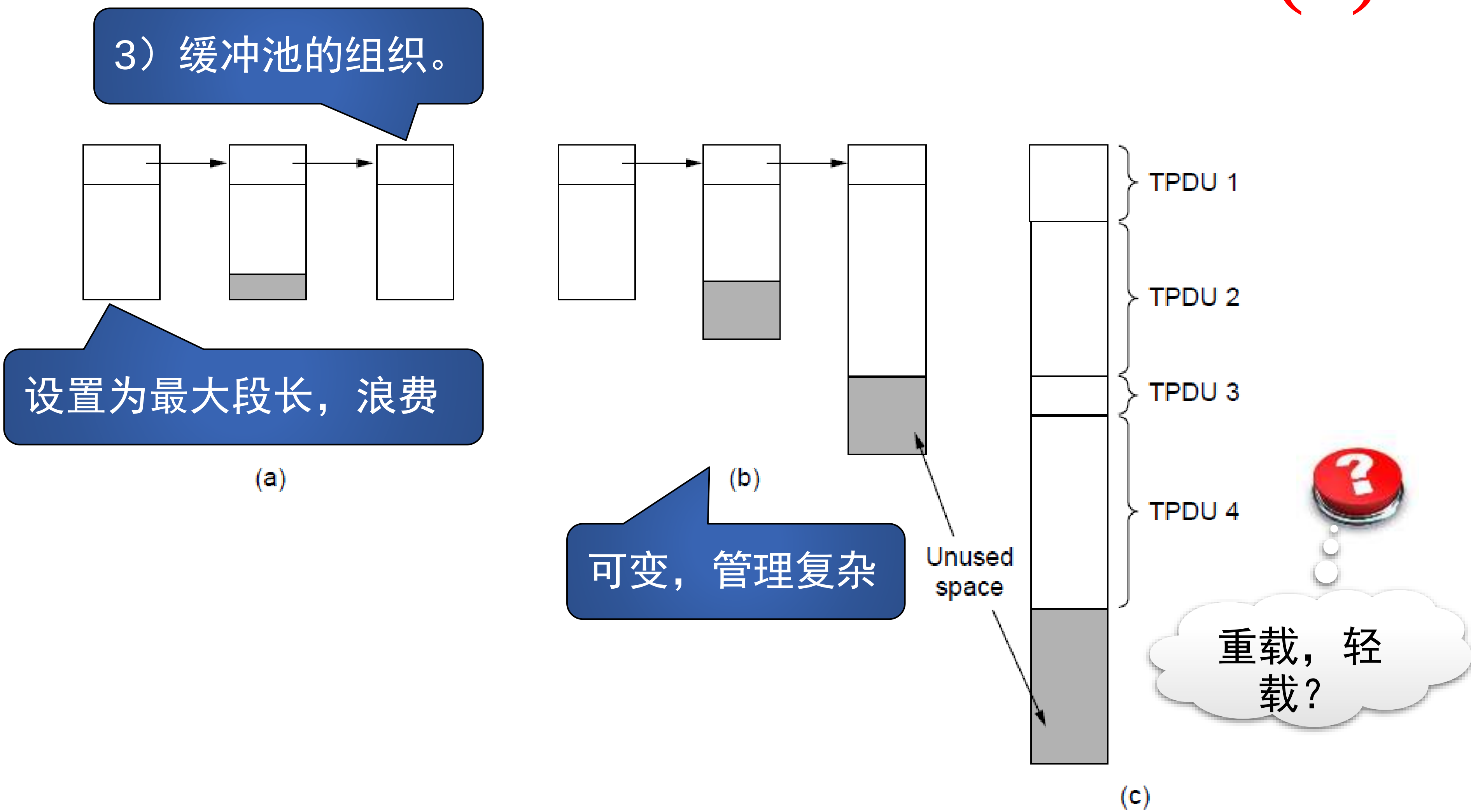
Error Control and Flow Control (1)

链路层只是链路，传输层还有路由器

- **Main problem:** Hosts may have so many connections that it becomes infeasible to allocate a fixed number of buffers per connection to implement a proper sliding window protocol, we need a **dynamic buffer allocation scheme**.

- 链路层带宽延时积很小，窗口小，e.g. 802.11。
- 1）传输层多用大的窗口（缓冲区）。
- 2）更进一步缓冲区可能需要多连接共享。接收端有专门的满窗口缓冲区最好。TCP

Error Control and Flow Control (2)



(a) Chained fixed-size buffers. (b) Chained variable-sized buffers. (c) One large circular buffer per connection.

Error Control and Flow Control (3)

几个蜗牛？

	A	Message	B	Comments
1	→	< request 8 buffers>	→	A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	B grants messages 0-3 only
3	→	<seq = 0, data = m0>	→	A has 3 buffers left now
4	→	<seq = 1, data = m1>	→	A has 2 buffers left now
5	→	<seq = 2, data = m2>	• • •	Message lost but A thinks it has 1 left
6	←	<ack = 1, buf = 3>	←	B acknowledges 0 and 1, permits 2-4
7	→	<seq = 3, data = m3>	→	A has 1 buffer left
8	→	<seq = 4, data = m4>	→	A has 0 buffers left, and must stop
9	→	<seq = 2, data = m2>	→	A times out and retransmits
10	←	<ack = 4, buf = 0>	←	Everything acknowledged, but A still blocked
11	←	<ack = 4, buf = 1>	←	A may now send 5
12	←	<ack = 4, buf = 2>	←	B found a new buffer somewhere
13	→	<seq = 5, data = m5>	→	A has 1 buffer left
14	→	<seq = 6, data = m6>	→	A is now blocked again
15	←	<ack = 6, buf = 0>	←	A is still blocked
16	• • •	<ack = 6, buf = 4>	←	Potential deadlock

缓冲与确认机制相分离

Dynamic buffer allocation. The arrows show the direction of transmission. An ellipsis (...) indicates a lost TPDU

动态缓冲区管理：可变大小的窗口。

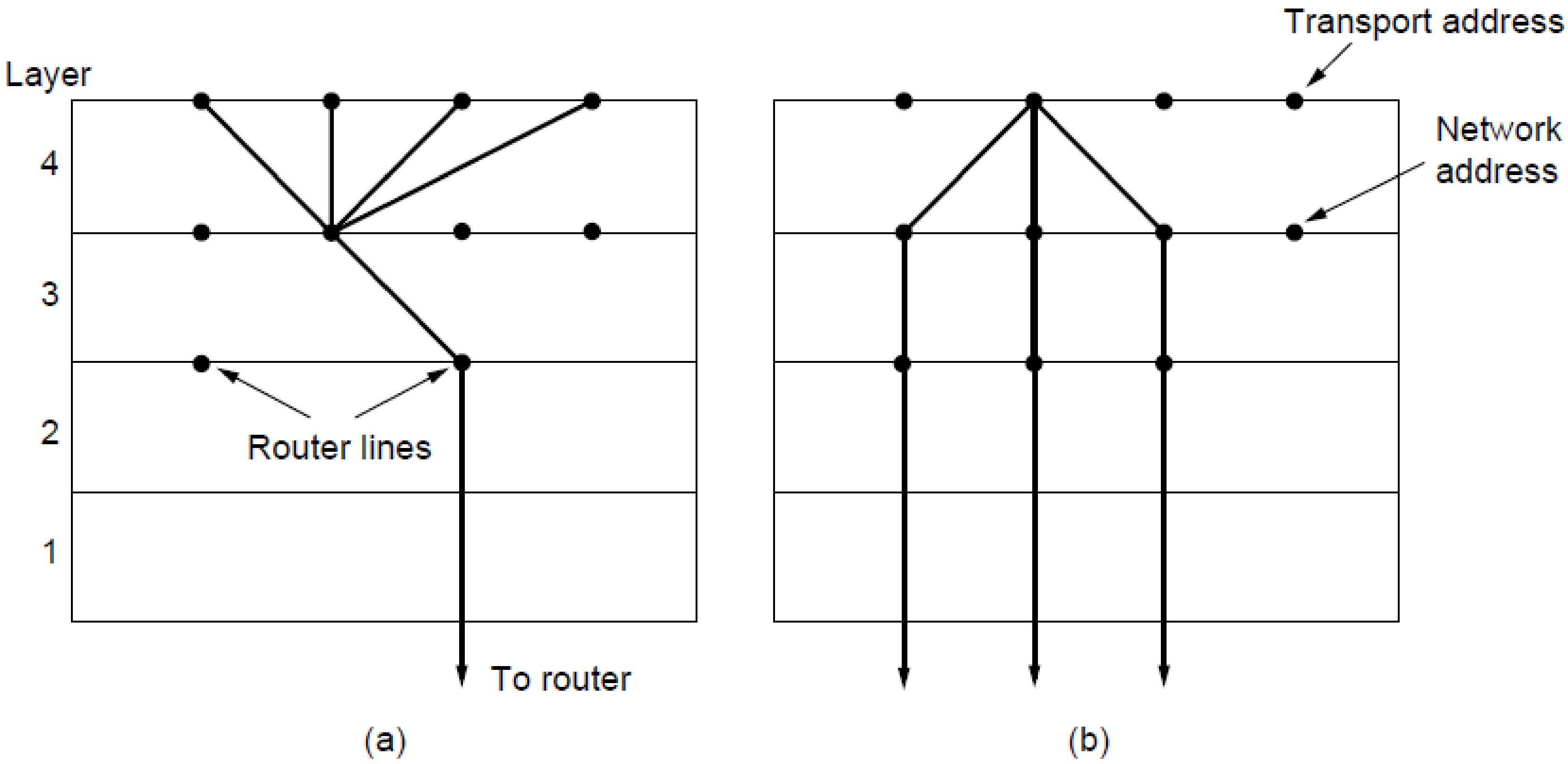
- 发送端根据需求请求缓冲区；
- 接收端尽可能分配缓冲区；
- 发送端发一个段，减少分配给它的缓冲区；
- 接收端在逆向流量中捎带确认和缓冲区数。

e.g. TCP window size.

w Control (3)

	A	M	B	Comments
1	→	< request 8 buffers>	→	A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	B grants messages 0-3 only
3	→	<seq = 0, data = m0>	→	A has 3 buffers left now
4	→	<seq = 1, data = m1>	→	A has 2 buffers left now
5	→	<seq = 2, data = m2>	• • •	Message lost but A thinks it has 1 left
6	←	<ack = 1, buf = 3>	←	B acknowledges 0 and 1, permits 2-4
7	→	<seq = 3, data = m3>	→	A has 1 buffer left
8	→	<seq = 4, data = m4>	→	A has 0 buffers left, and must stop
9	→	<seq = 2, data = m2>	→	A times out and retransmits
10	←	<ack = 4, buf = 0>	←	Everything acknowledged, but A still blocked
11	←	<ack = 4, buf = 1>	←	A may now send 5
12	←	<ack = 4, buf = 2>	←	B found a new buffer somewhere
13	→	<seq = 5, data = m5>	→	A has 1 buffer left
14	→	<seq = 6, data = m6>	→	A is now blocked again
15	←	<ack = 6, buf = 0>	←	A is still blocked
16	• • •	<ack = 6, buf = 4>	←	Potential deadlock

Multiplexing



(a) Multiplexing. (b) Inverse multiplexing.

Crash Recovery

从第N层崩溃中的恢复工作只能由N+1层完成

Strategy used by receiving host

Strategy used by sending host	First ACK, then write			First write, then ACK		
	AC(W)	AWC	C(AW)	C(WA)	W AC	WC(A)
Always retransmit	OK	DUP	OK	OK	DUP	DUP
Never retransmit	LOST	OK	LOST	LOST	OK	OK
Retransmit in S0	OK	DUP	LOST	LOST	DUP	OK
Retransmit in S1	LOST	OK	OK	OK	OK	DUP

S0：没有未完成的段，有确认才重传
S1：发出一个段，但是没有确认

OK = Protocol functions correctly
DUP = Protocol generates a duplicate message
LOST = Protocol loses a message

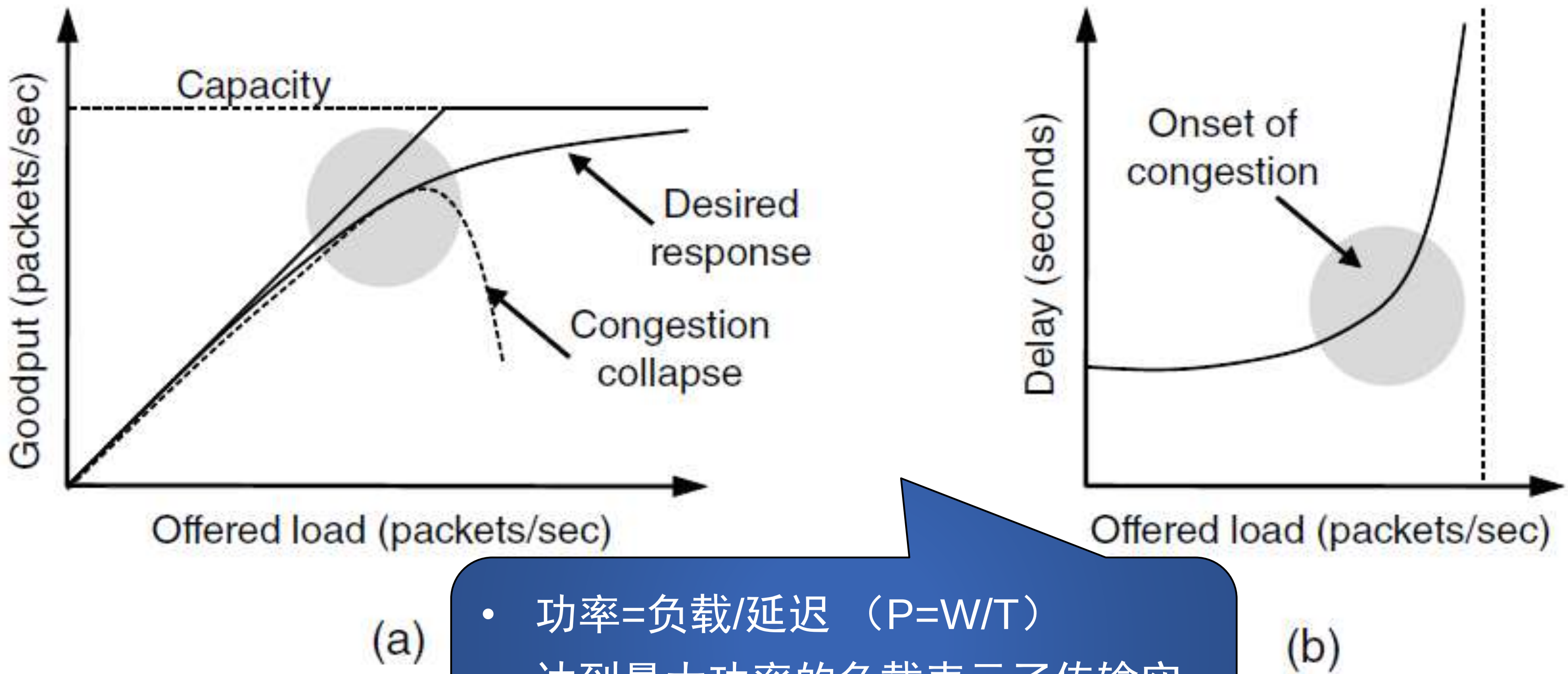
Different combinations of client and server strategy

6.3 Congestion Control

- 利用所有可用带宽，却能避免拥塞。
- 对整个竞争实体是公平的。
- 并能快速跟踪流量的变化。（收敛）

- Desirable bandwidth allocation
  - Efficiency and power
  - Max-min fairness
  - Convergence
- Regulating the sending rate
- Wireless Issues

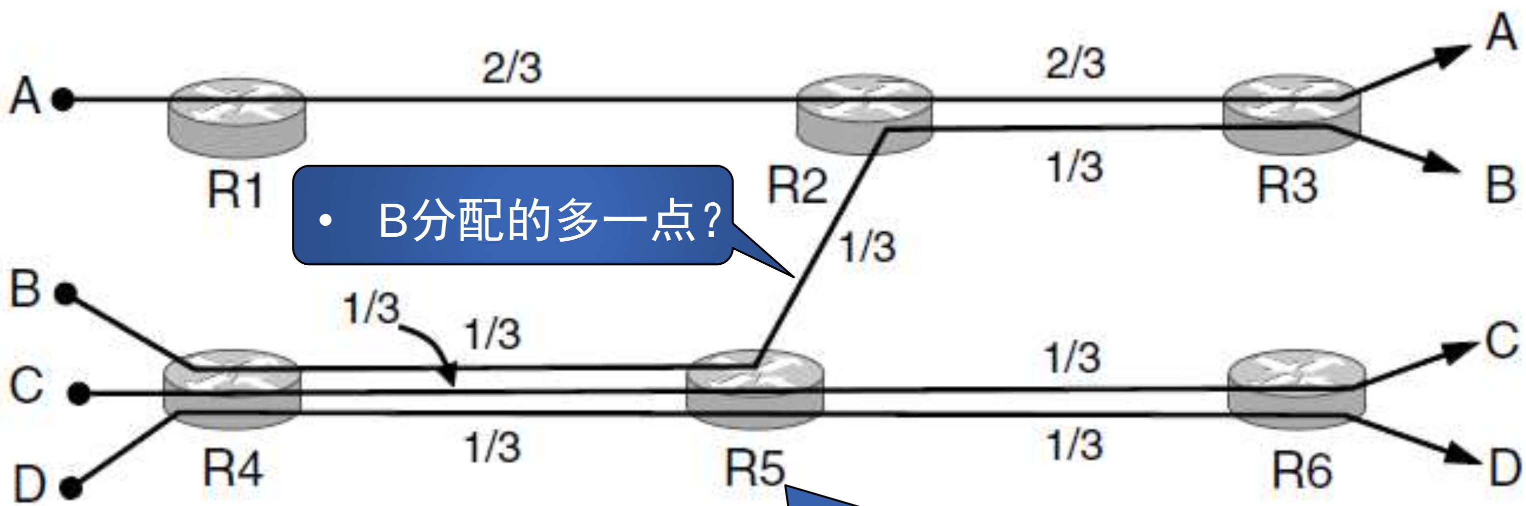
Desirable Bandwidth Allocation (1)



- 功率=负载/延迟 （P=W/T）
- 达到最大功率的负载表示了传输实体放置在网络上的有效负载

(a) Goodput and (b) delay as a function of offered load

Desirable Bandwidth Allocation (2)

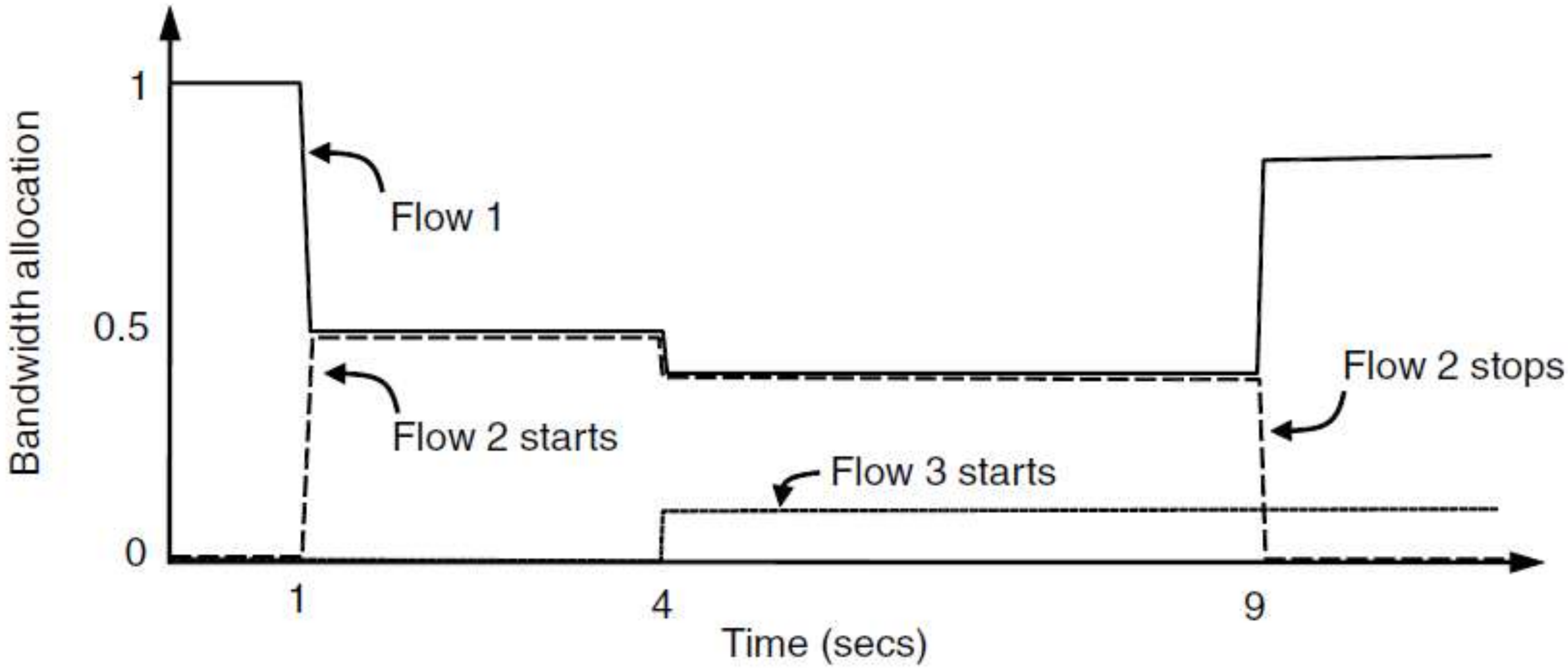


- 最大最小公平：分配给一个流的带宽在不减少分配给另一个流带宽的前提下无法得到进一步增长，那么就不给这个流更多带宽

连接级别的公平？服务器级别得公平（多连接）

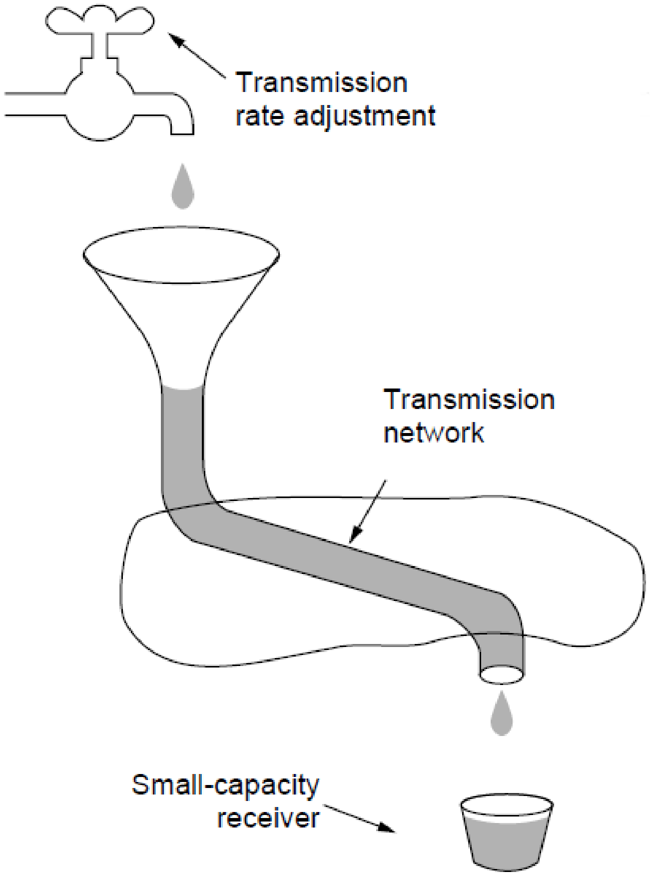
Max-min bandwidth allocation for four flows

Desirable Bandwidth Allocation (3)



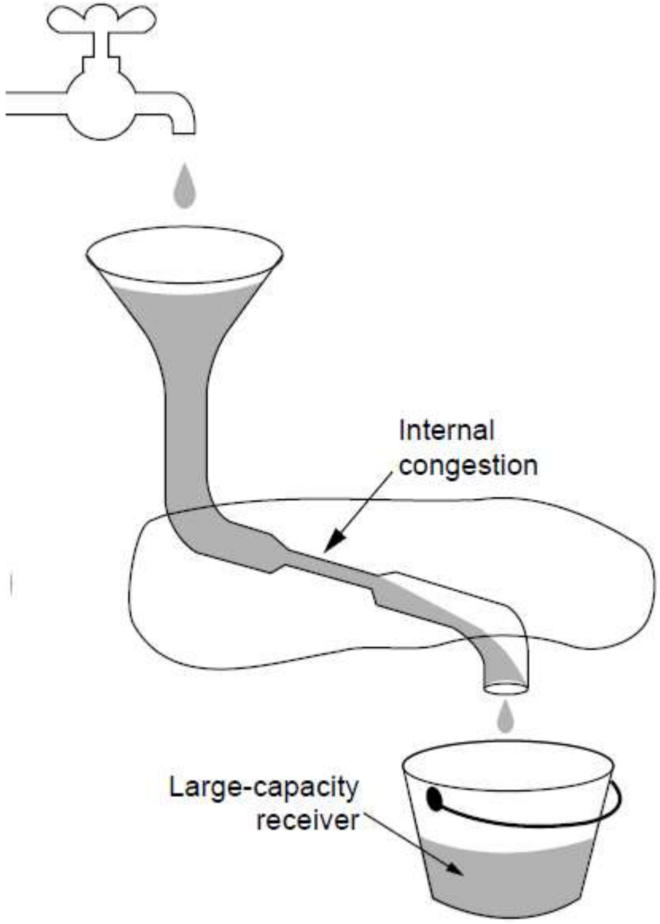
Changing bandwidth allocation over time

Regulating the Sending Rate (1)



A fast network feeding a low-capacity receiver

Regulating the Sending Rate (2)



A slow network feeding a high-capacity receiver

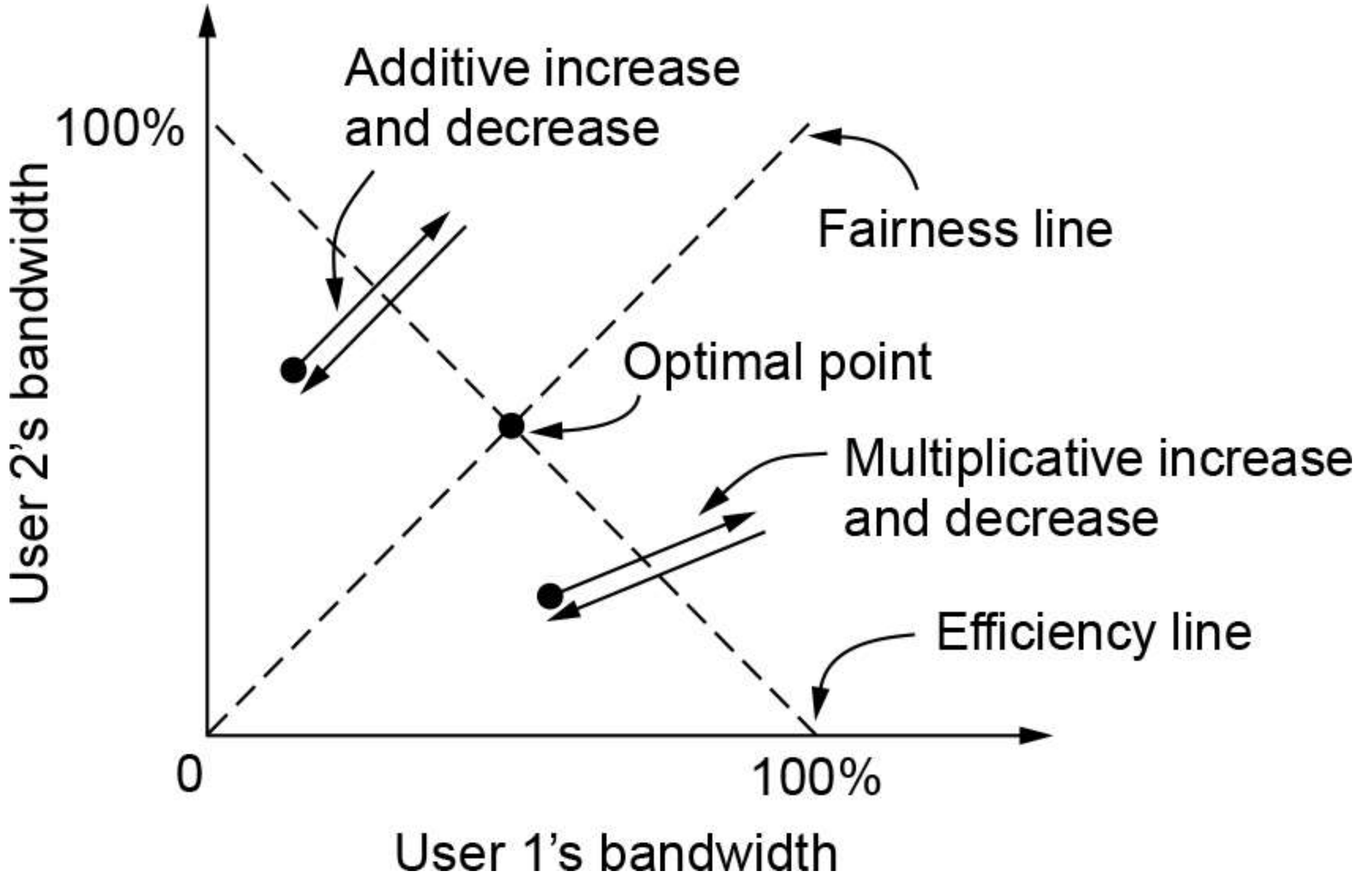
Regulating the Sending Rate (3)

eXplicit Congestion Control

Protocol	Signal	Explicit?	Precise?
XCP	Rate to use	Yes	Yes
TCP with ECN	Congestion warning	Yes	No
FAST TCP	End-to-end delay	No	Yes
CUBIC TCP	Packet loss	No	No
TCP	Packet loss	No	No

Some congestion control protocols

Regulating the Sending Rate (4)



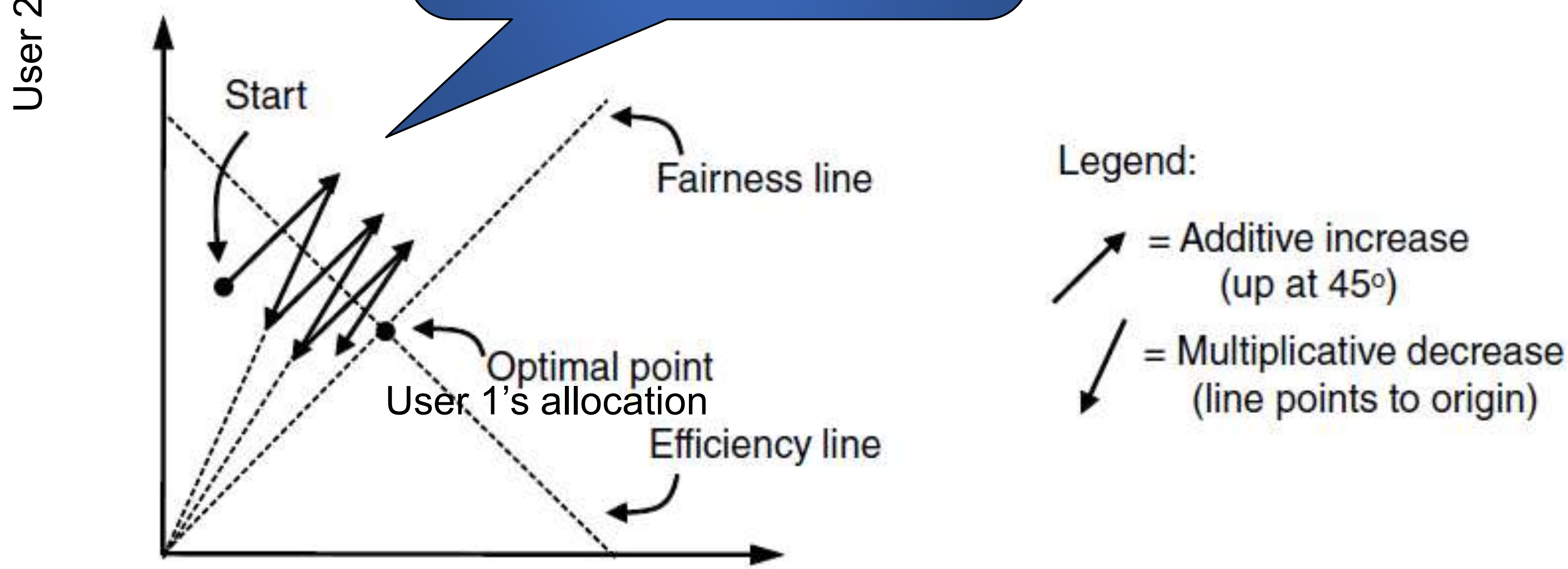
Additive and multiplicative bandwidth adjustments

Regulating the Sending Rate (5)

e.g.

（1，3）加性增（2，4）

（2，4）乘性减（1，2）



Additive Increase Multiplicative Decrease (AIMD) control law.

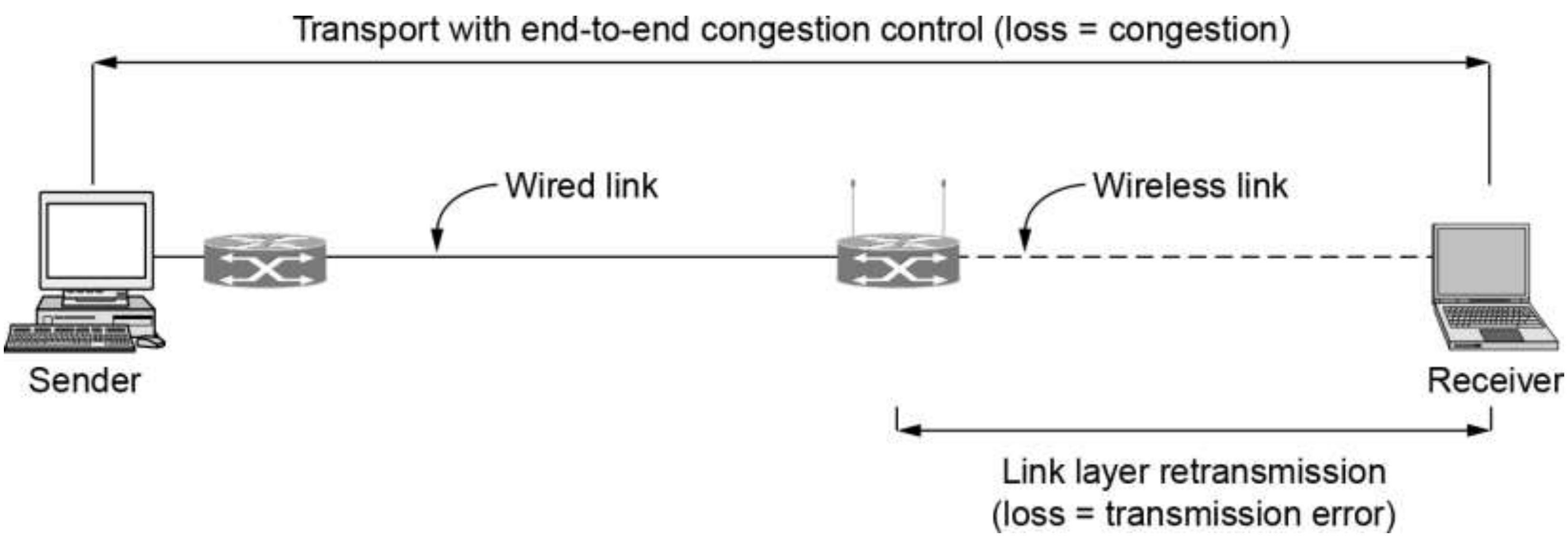
Wireless

- 无线网络链路重传在微秒，毫秒级。 但是传输层在毫秒，秒级。链路层的帧重传和传输层的拥塞控制作用在不同的时间尺度

1, 拥塞， 误码？

2，链路的容量变化。

- 不特别处理



Congestion control over a path with a wireless link

6.4 The Internet Transport Protocols: UDP

User Datagram Protocol

- Introduction to UDP
- Remote procedure call
- Real-time transport
  - RTP—the Real-time Transport Protocol
  - RTCP—the Real-time Transport Control Protocol
  - Playout with buffering and jitter control

Introduction to UDP (1)

- UDP provides a way for applications to send encapsulated raw IP datagram without having to establish a connection (RFC 768).
- It does not do flow control, error control, or retransmission upon receipt of a bad segment.
- It **does** provide an interface to the IP protocol with the added feature of demultiplexing multiple processes using the ports.

Introduction to UDP (1)

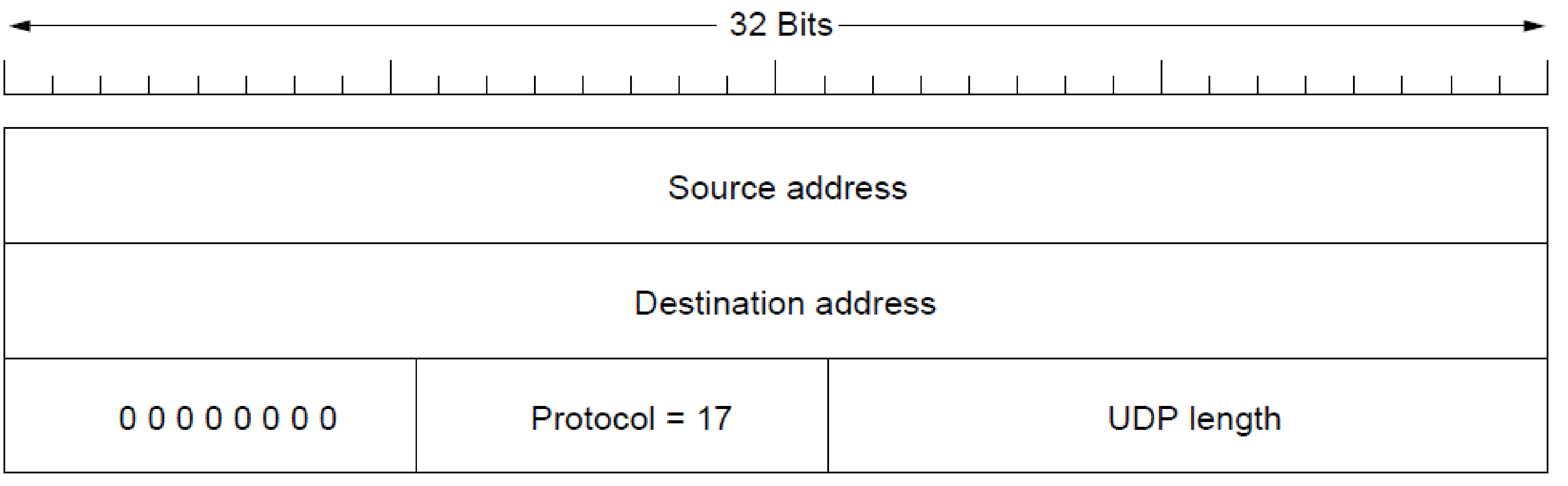
32 Bits

Source port	Destination port
UDP length	UDP checksum

校验 头，数据，IP伪头

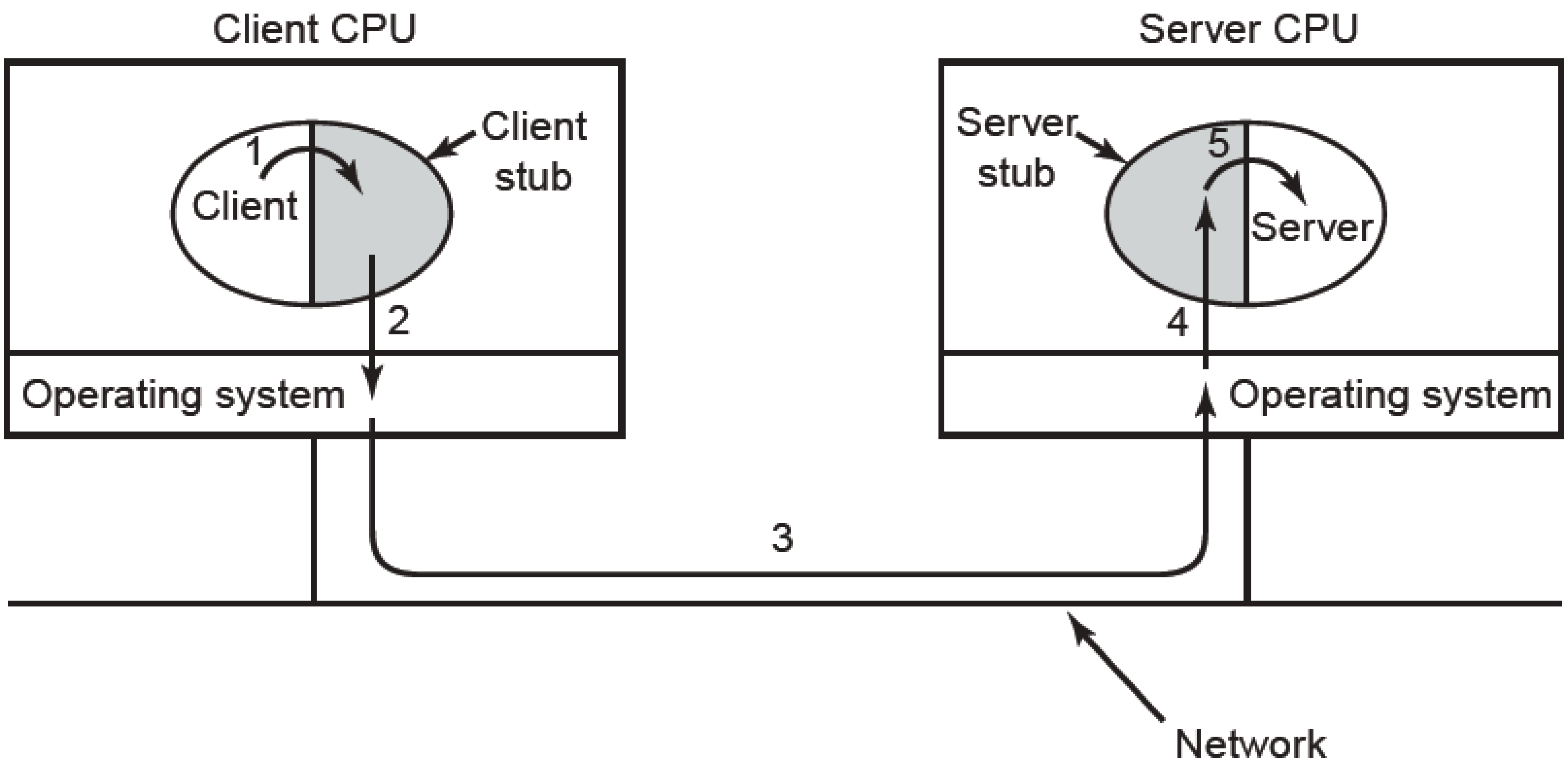
The UDP header.

Introduction to UDP (2)



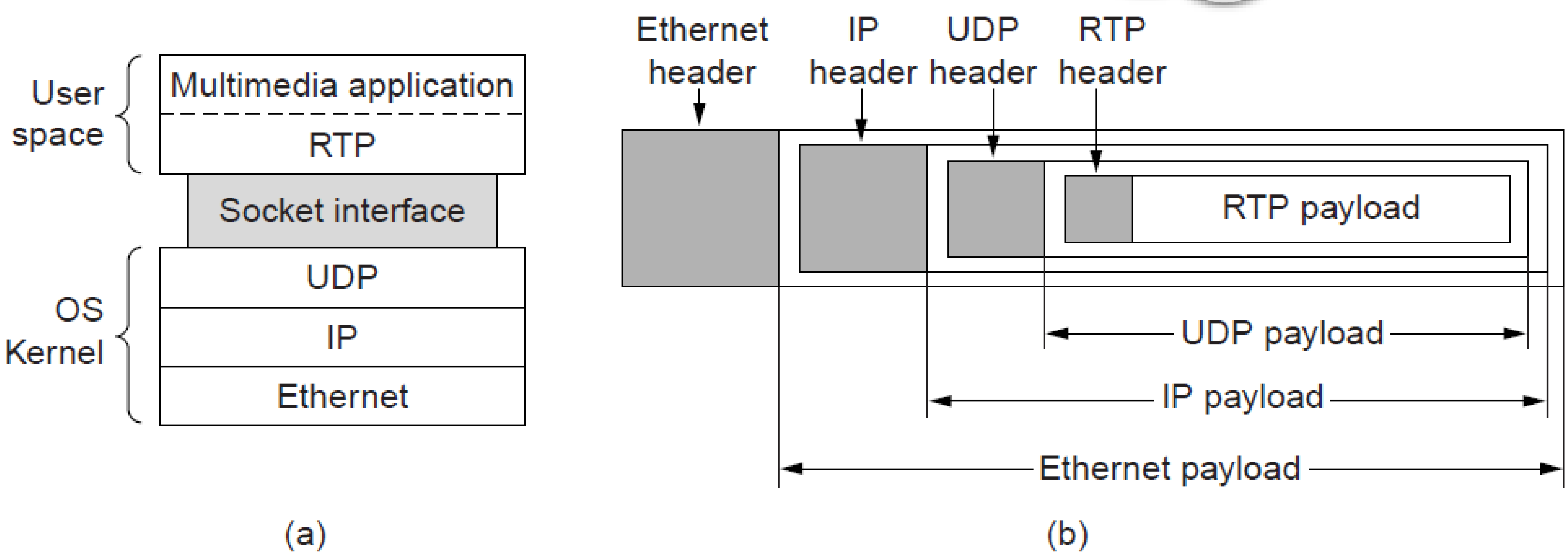
The IPv4 pseudoheader included in the UDP checksum.

Remote Procedure Call



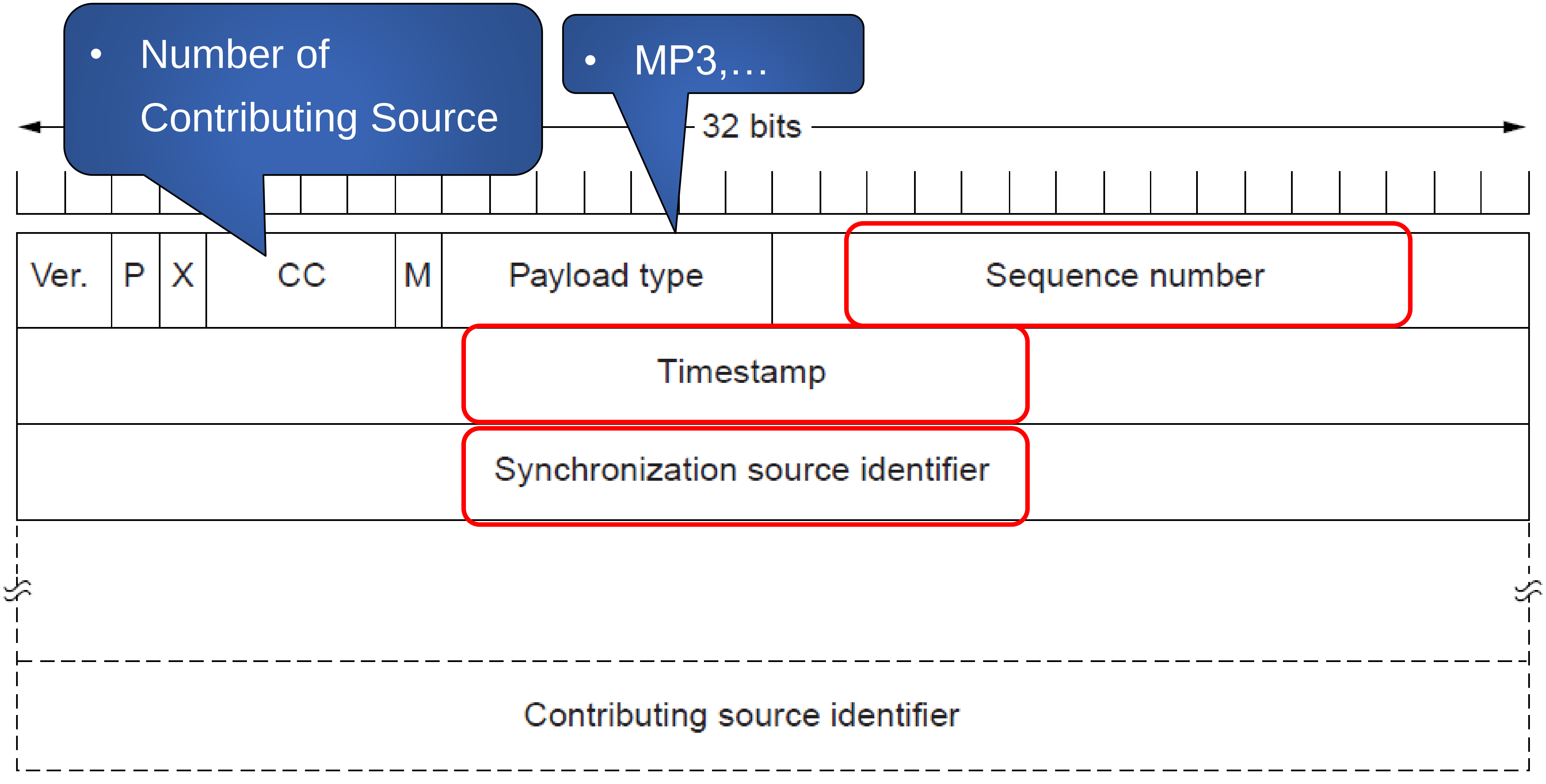
Steps in making a remote procedure call. The stubs are shaded.

Real-Time Transport (1)



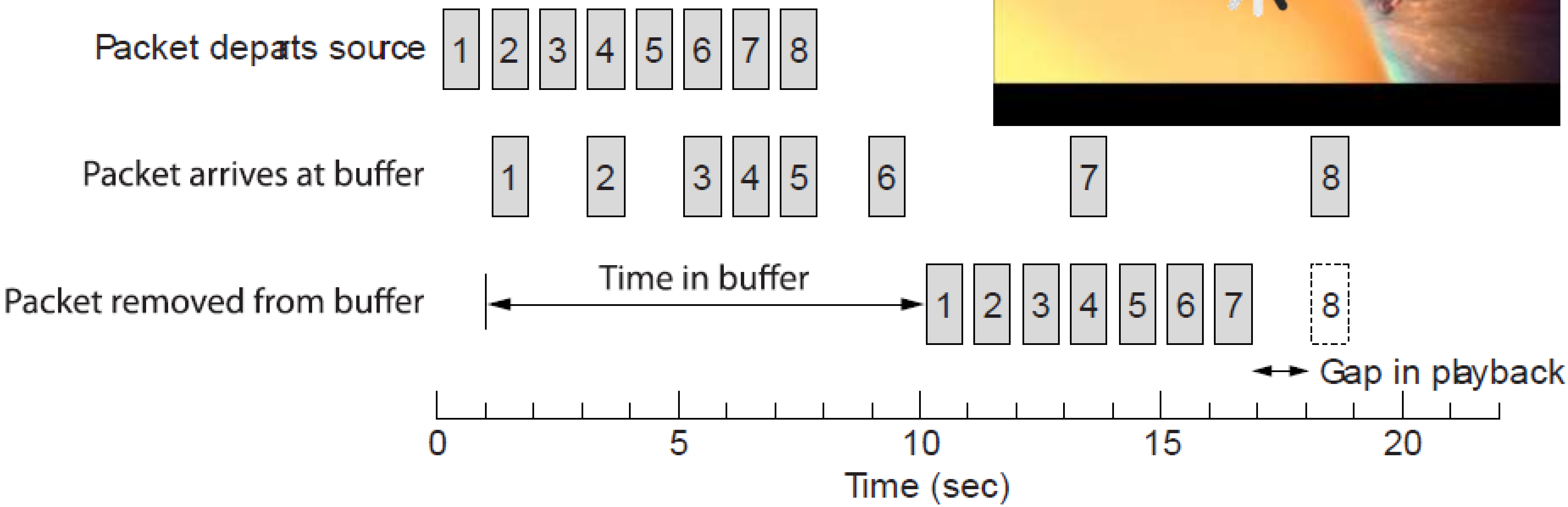
(a) The position of RTP in the protocol stack. (b) Packet nesting.

Real-Time Transport (2)



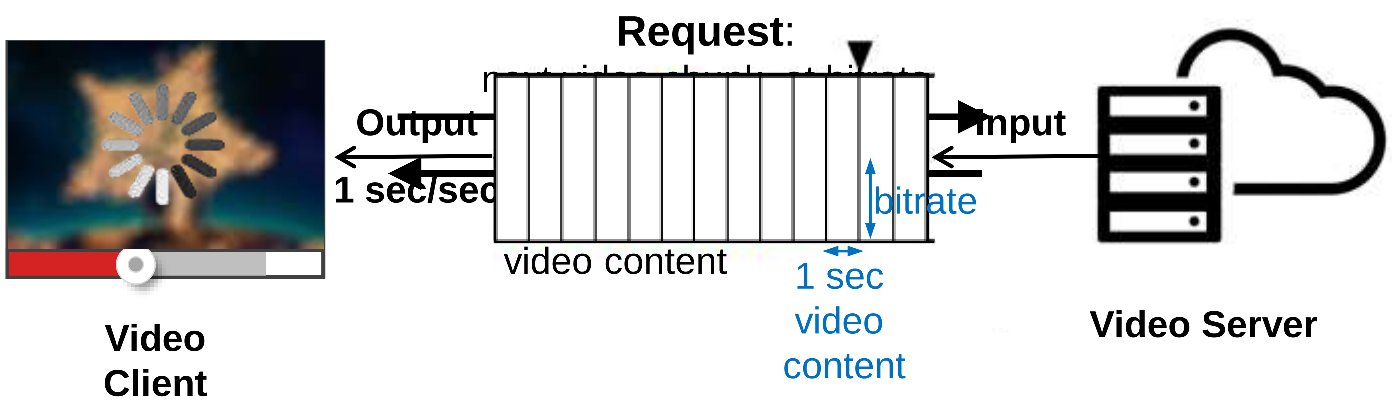
The RTP header

Real-Time Transport (3)



Smoothing the output stream by buffering packets

Real-Time Transport (3)



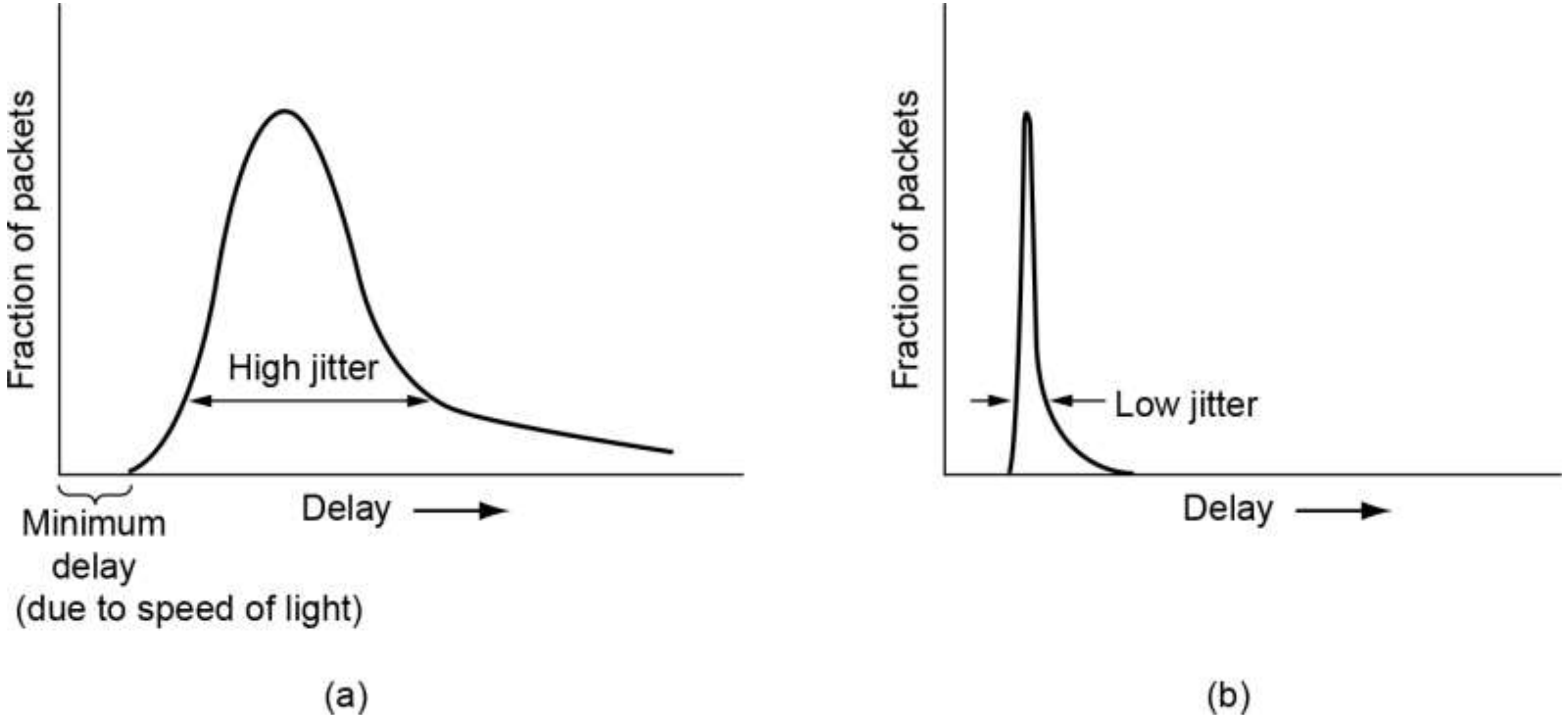
Animation borrowed from Te-Yuan Huang (SIGCOMM '14) http://conferences.sigcomm.org/sigcomm/2014/doc/slides/38.pdf

Real-Time Transport (3)

Video
Client

Video Server

Real-Time Transport (4)



(a) High jitter. (b) Low jitter.

6.5 The Internet Transport Protocols: TCP (1)

Transmission Control Protocol

- Introduction to TCP
- The TCP service model
- The TCP protocol
- The TCP segment header
- TCP connection establishment
- TCP connection release

The Internet Transport Protocols: TCP (2)

- TCP connection management modeling
- TCP sliding window
- TCP timer management
- TCP congestion control
- TCP CUBIC

The Internet Transport Protocols: TCP (3)

- TCP ensures reliable, point–to–point connections. No support for multicasting or broadcasting.
- A TCP TPDU is called a **segment**, consisting of (minimal) 20-byte header, and maximum total length of 65,535 bytes. A segment is fragmented by the network layer when it is larger than the network's **maximum transfer unit** (MTU).

The Internet Transport Protocols: TCP (4)

- TCP service is obtained by both the sender and receiver creating end points, called **sockets**, which has a socket number (address) consisting of the IP address of the host and a 16-bit number local to that host, called a port. A socket may be used for multiple connections at the same time. Two or more connections may terminate at the same socket

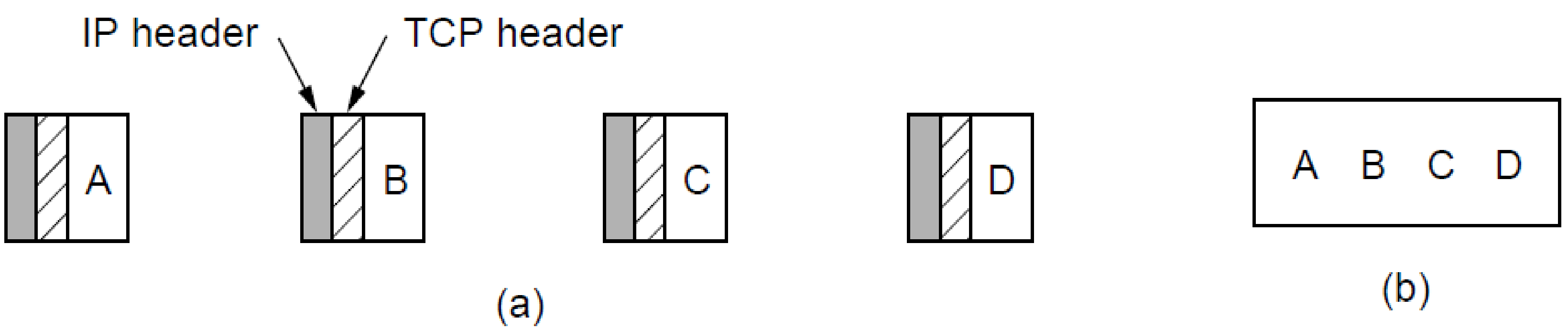
The TCP Service Model (1)

Port	Protocol	Use
20, 21	FTP	File transfer
22	SSH	Remote login, replacement for Telnet
25	SMTP	Email
80	HTTP	World Wide Web
110	POP-3	Remote email access
143	IMAP	Remote email access
443	HTTPS	Secure Web (HTTP over SSL/TLS)
543	RTSP	Media player control
631	IPP	Printer sharing

Some assigned ports

The TCP Service Model (2)

- A TCP connection **is a byte stream**, not a message stream. Message boundaries are not preserved end to end. If sending process does four 512-byte writes to a TCP stream, these data may be delivered as four 512-byte chunks, two 1024-byte chunks, one 2048-byte chunk.

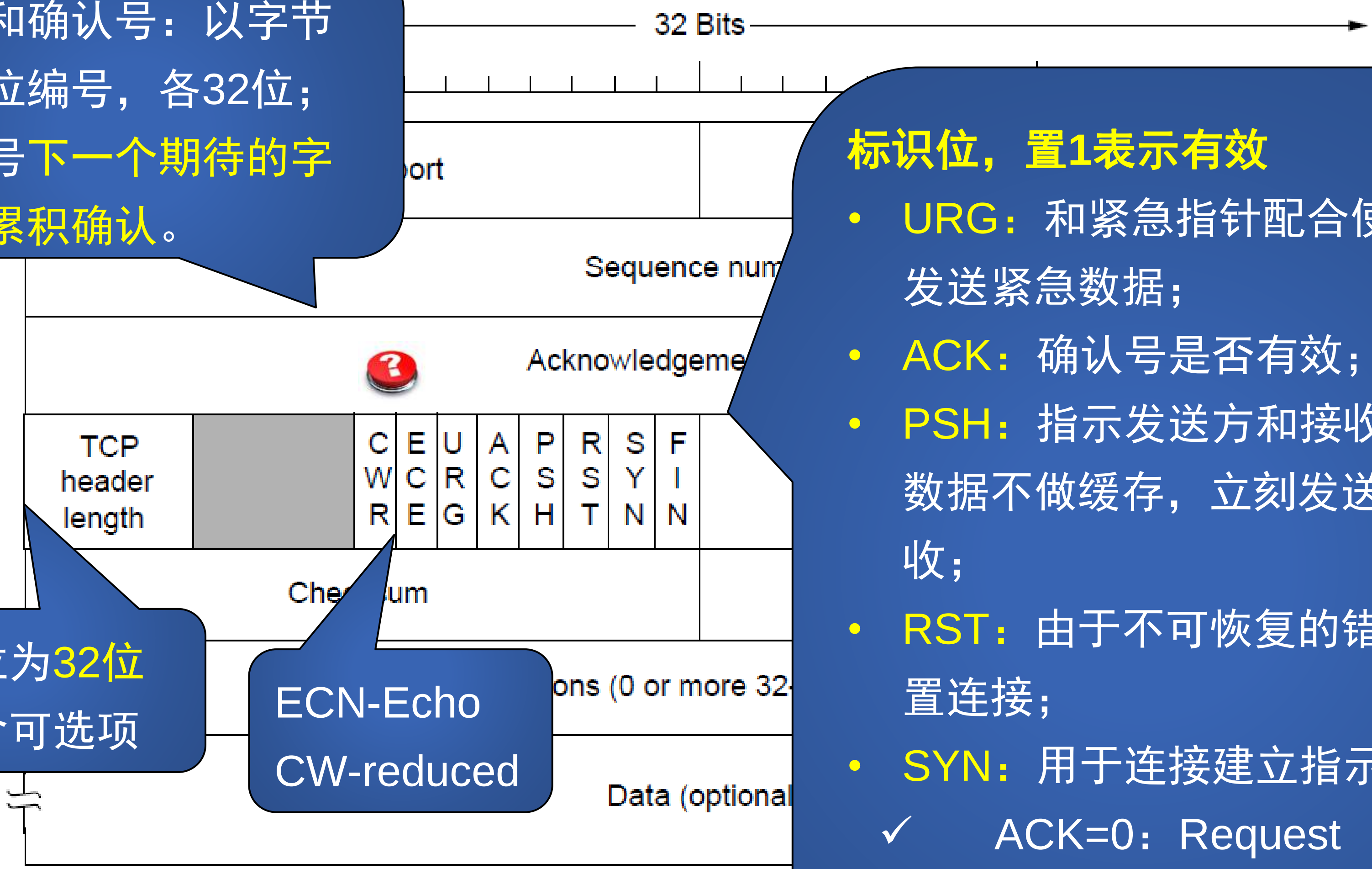


(a) Four 512-byte segments sent as separate IP diagrams

(b) The 2048 bytes of data delivered to the application in a single READ call

The TCP Segment Header

序号和确认号：以字节为单位编号，各32位；确认号下一个期待的字节；累积确认。



长度单位为32位字，包含可选项

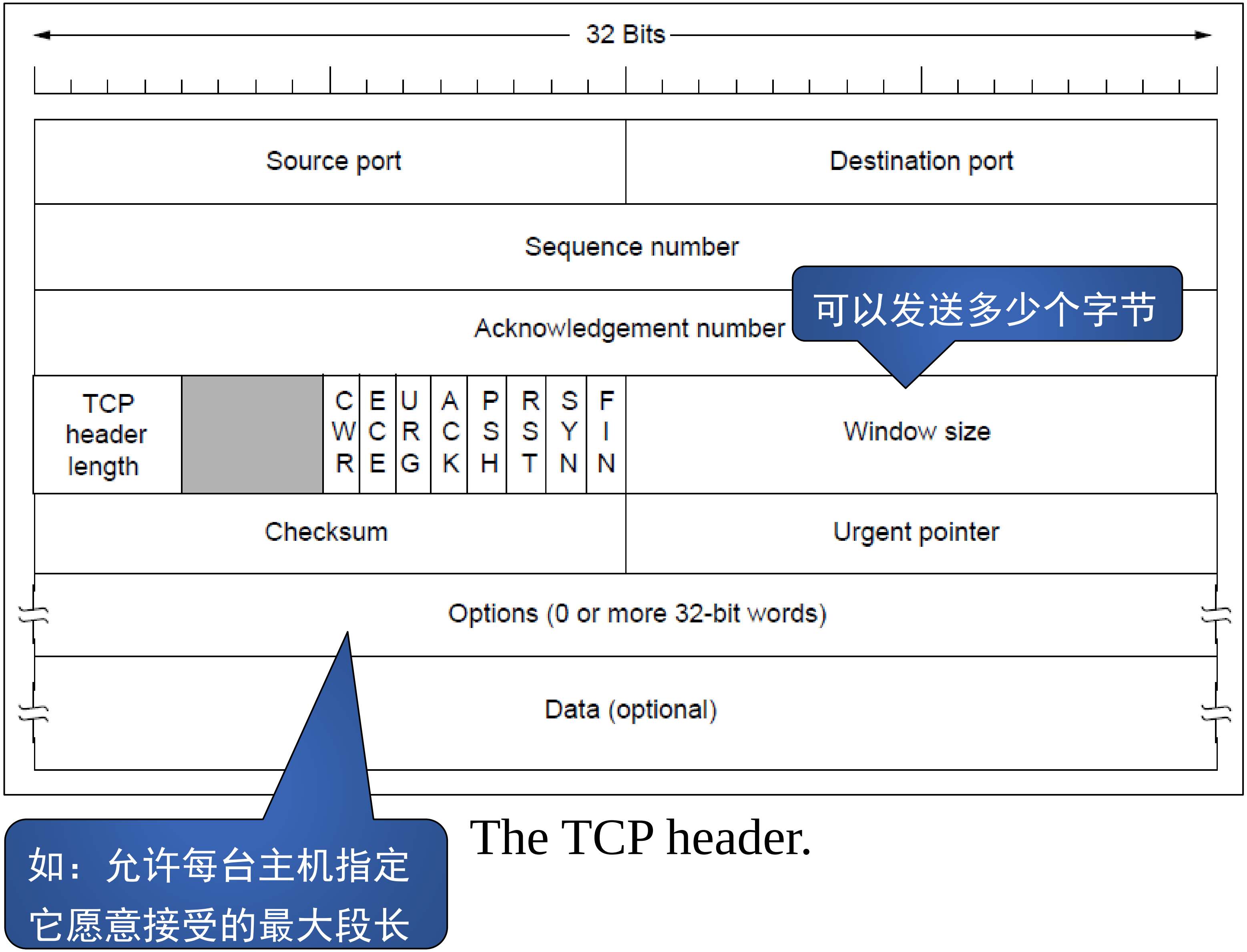
ECN-Echo
CW-reduced

标识位，置1表示有效

- URG：和紧急指针配合使用，发送紧急数据；
- ACK：确认号是否有效；
- PSH：指示发送方和接收方将数据不做缓存，立刻发送或接收；
- RST：由于不可恢复的错误重置连接；
- SYN：用于连接建立指示；
  - ✓ ACK=0：Request
  - ✓ ACK=1：Accepted
- FIN：用于连接释放指示

The TCP he

The TCP Segment Header



The TCP header.

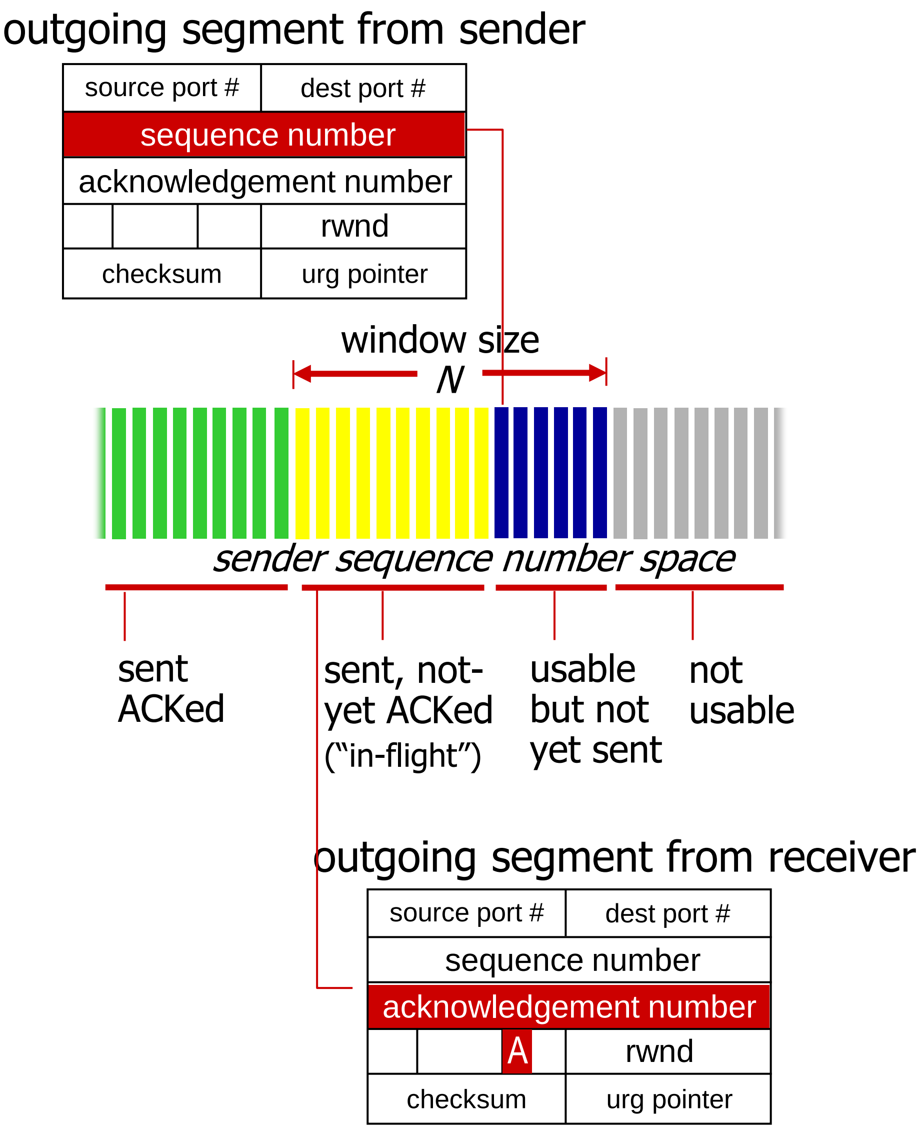
TCP sequence numbers, ACKs (补充1)

Sequence numbers:

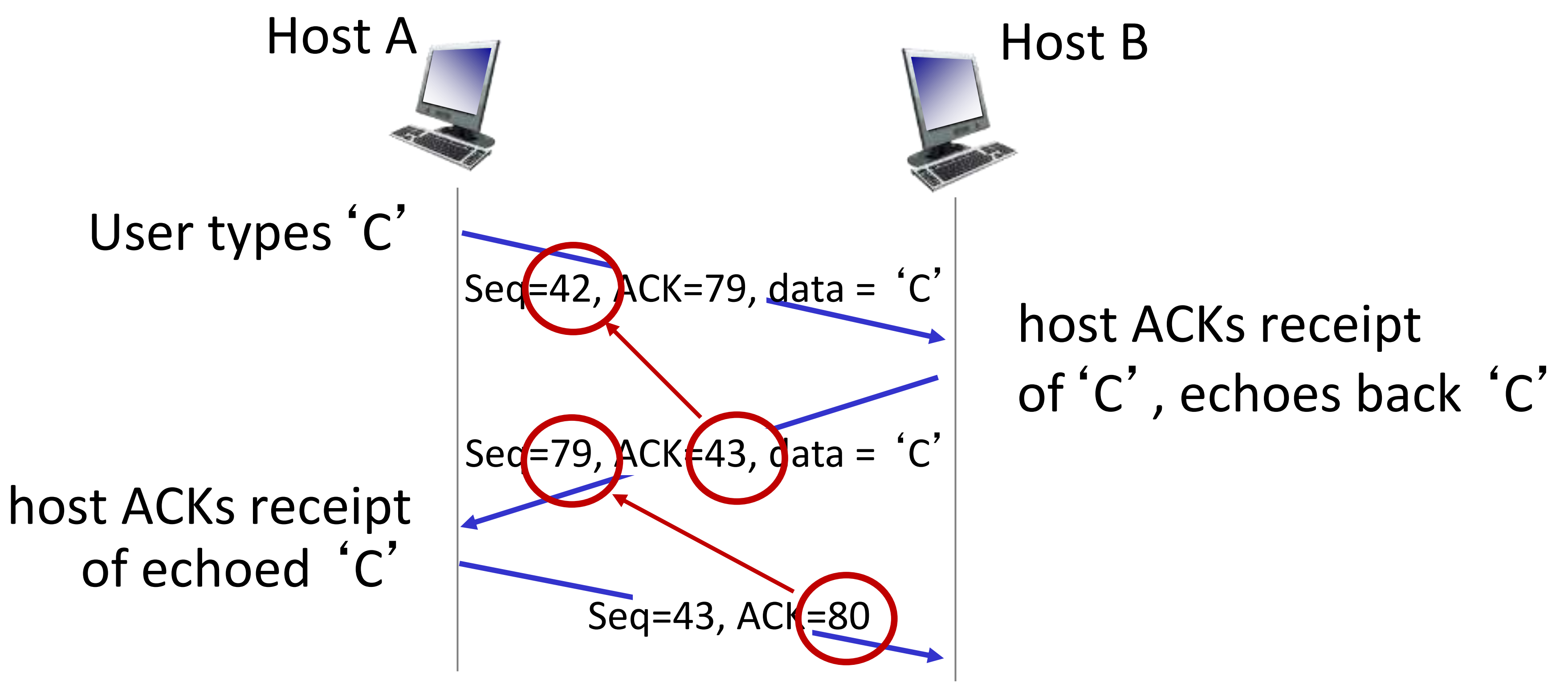
- byte stream "number" of first byte in segment's data

Acknowledgements:

- seq # of next byte expected from other side
- cumulative ACK



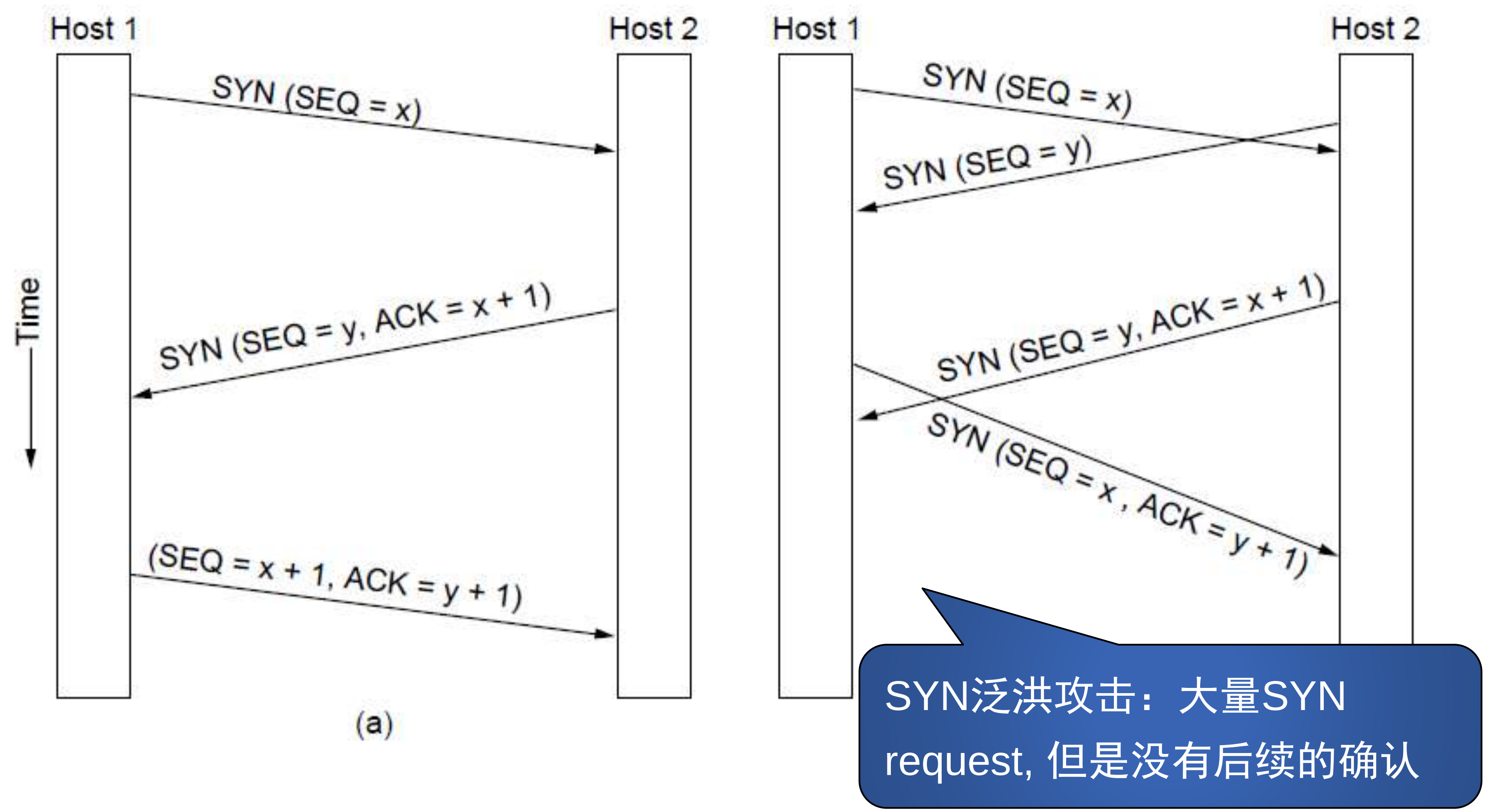
TCP sequence numbers, ACKs (补充2)



TCP Connection Establishment

- 三次握手建立连接
  - 服务器方执行LISTEN和ACCEPT原语，被动监听；
  - 客户方执行CONNECT原语，产生一个SYN为1和ACK为0的TCP段，表示连接请求；
  - 服务器方的传输实体接收到这个TCP段后，首先检查是否有服务进程在所请求的端口上监听，若没有，回答RST置位的TCP段；
  - 若有服务进程在所请求的端口上监听，该服务进程可以决定是否接受该请求。在接受后，发出一个SYN置1和ACK置1的TCP段表示连接确认，并请求与对方的连接；
  - 发起方收到确认后，发出一个SYN置0和ACK置1的TCP段表示给对方的连接确认；

TCP Connection Establishment

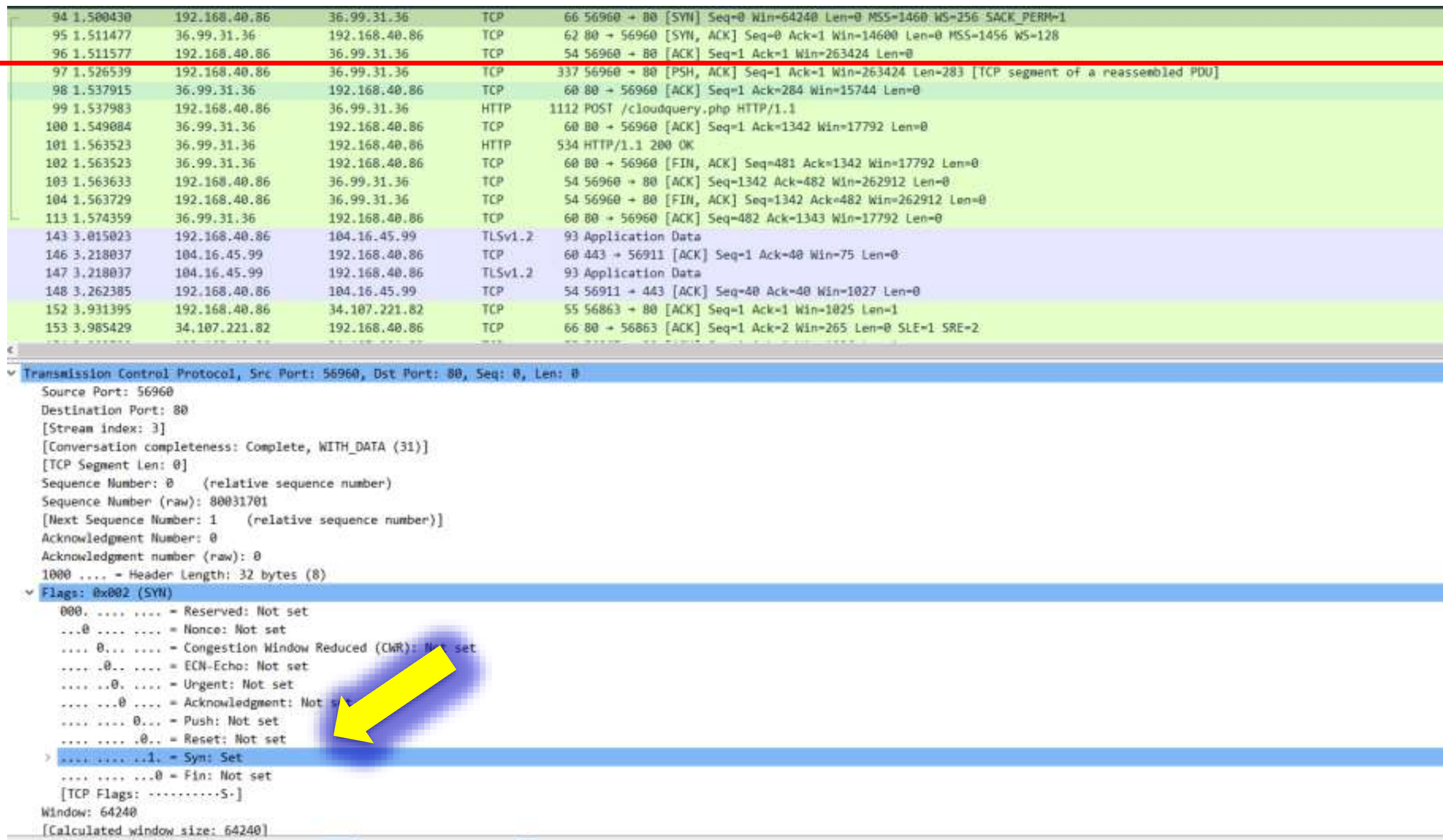


SYN泛洪攻击：大量SYN request, 但是没有后续的确认

(a) TCP connection establishment in the normal case.

(b) Simultaneous connection establishment on both sides.

TCP Connection Establishment

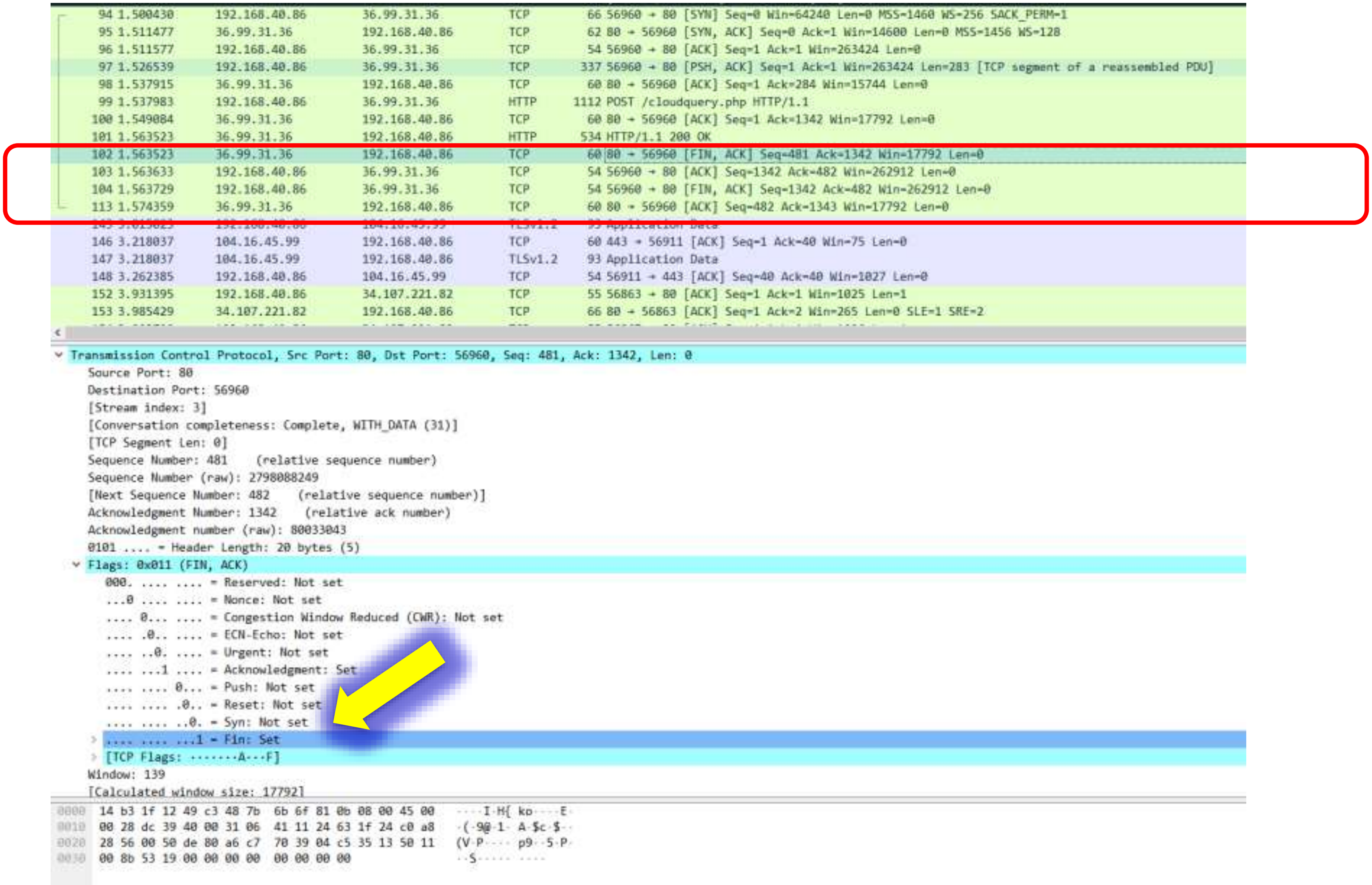


TCP Connection Management Modeling (1)

State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIME WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

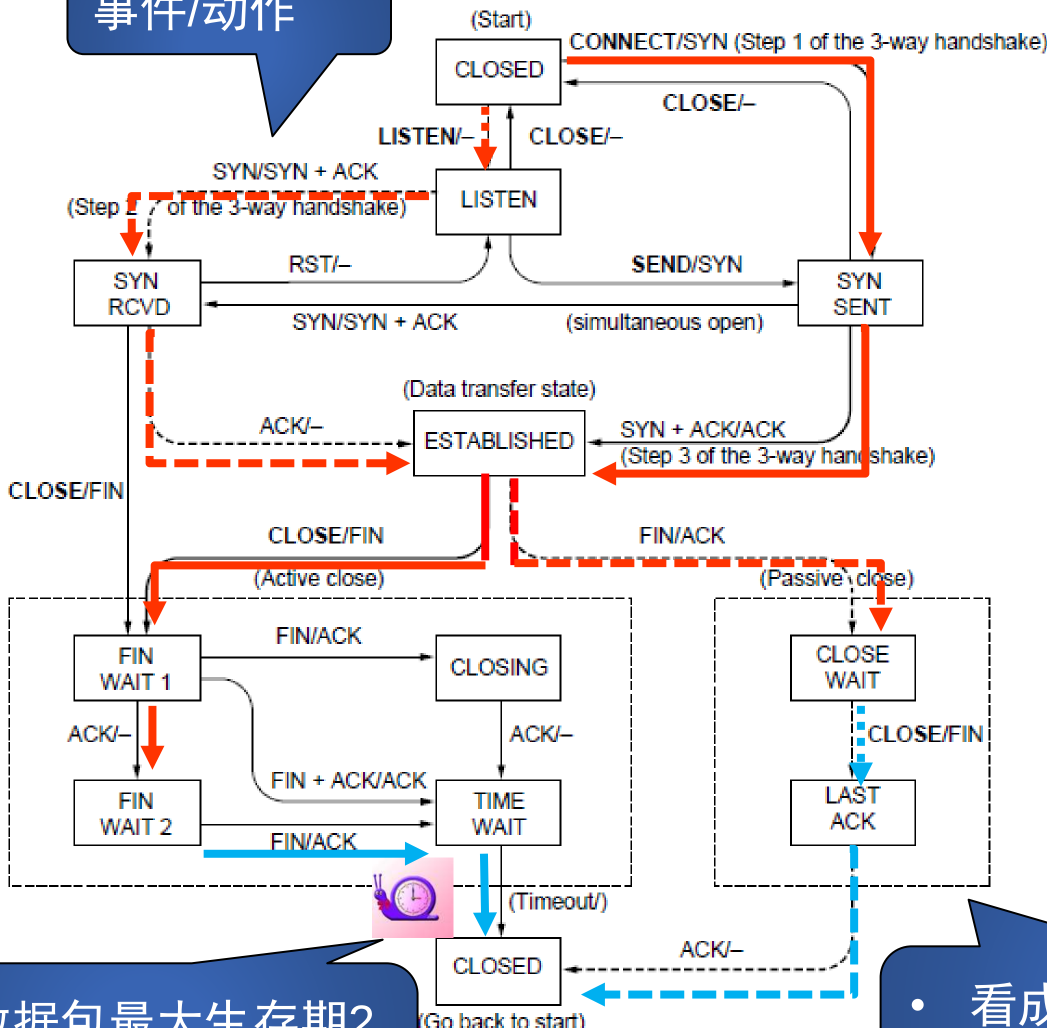
The states used in the TCP connection management finite state machine.

TCP Connection Establishment



TCP Connection Management Modeling (2)

事件/动作



- **TCP connection management finite state machine.**
- The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events. Each transition is labeled by the event causing it and the action resulting from it, separated by a slash.

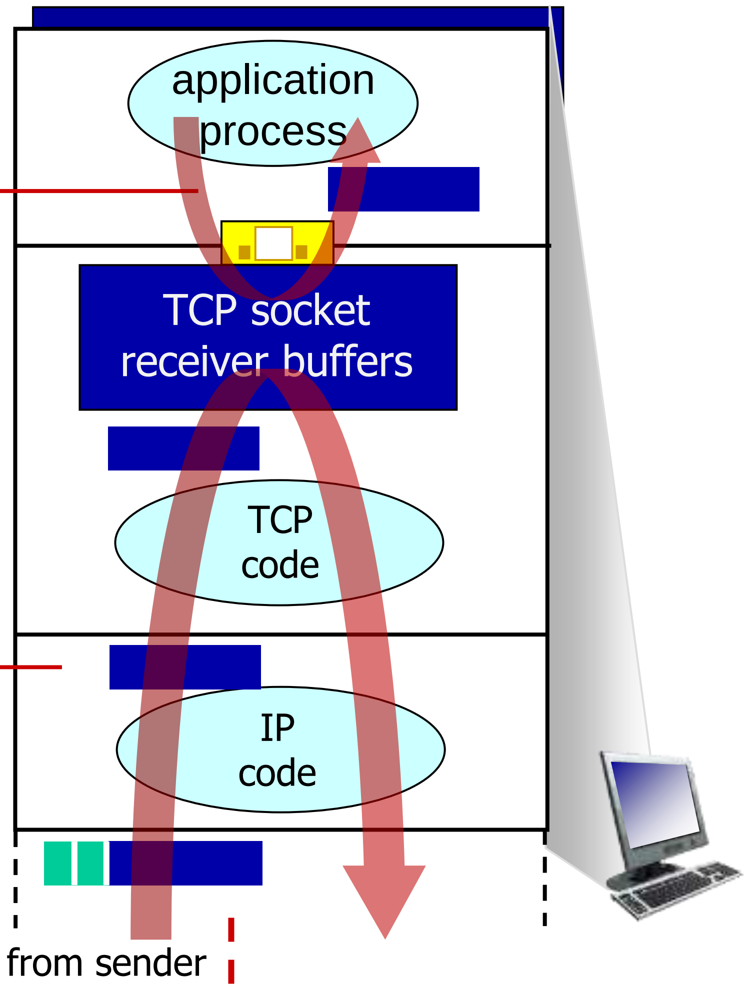
- 数据包最大生存期2倍的

- 看成一对单工连接单独释放 FIN-ACK

TCP Sliding Window (补前传1)

Application removing data from TCP socket buffers

Network layer delivering IP datagram payload into TCP socket buffers



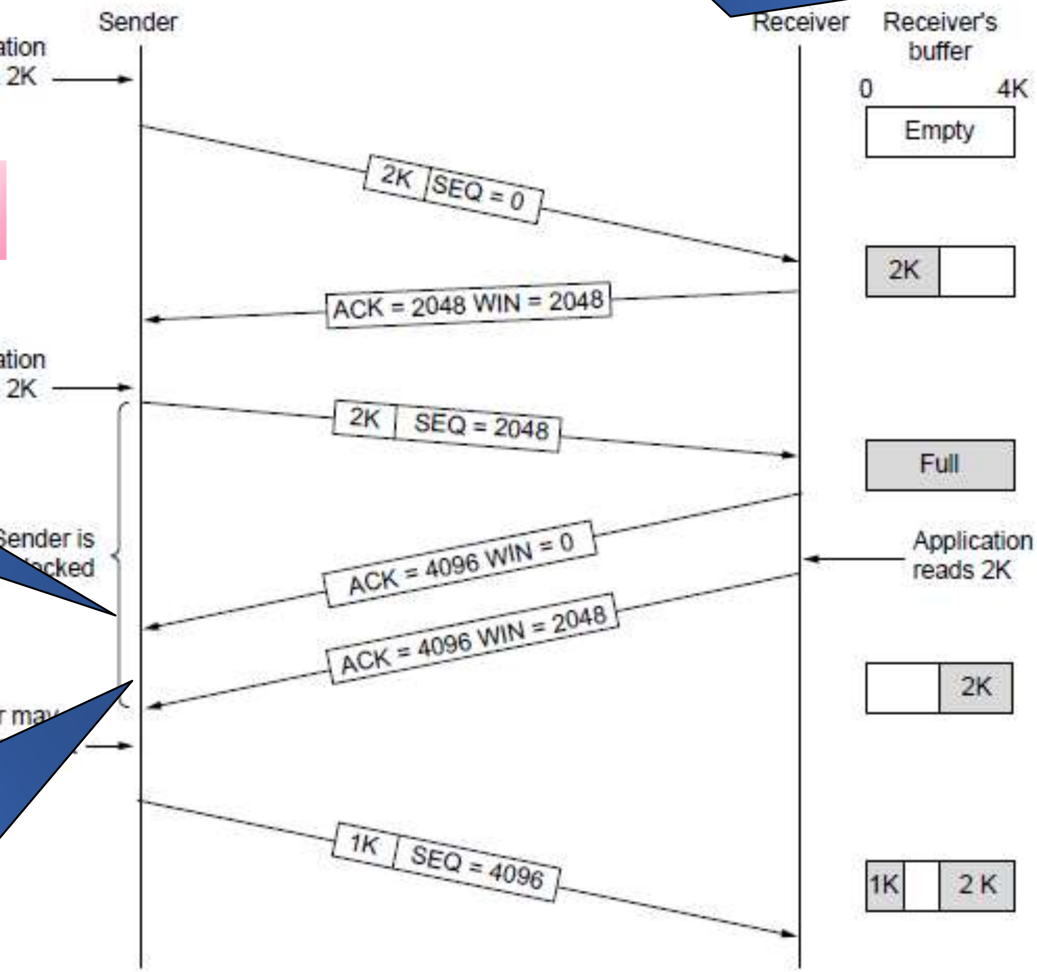
receiver protocol stack

TCP Sliding Window (补前传2)

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?

TCP Sliding Window (1)

延迟确认：将确认延迟,完全可以等4K再发确认



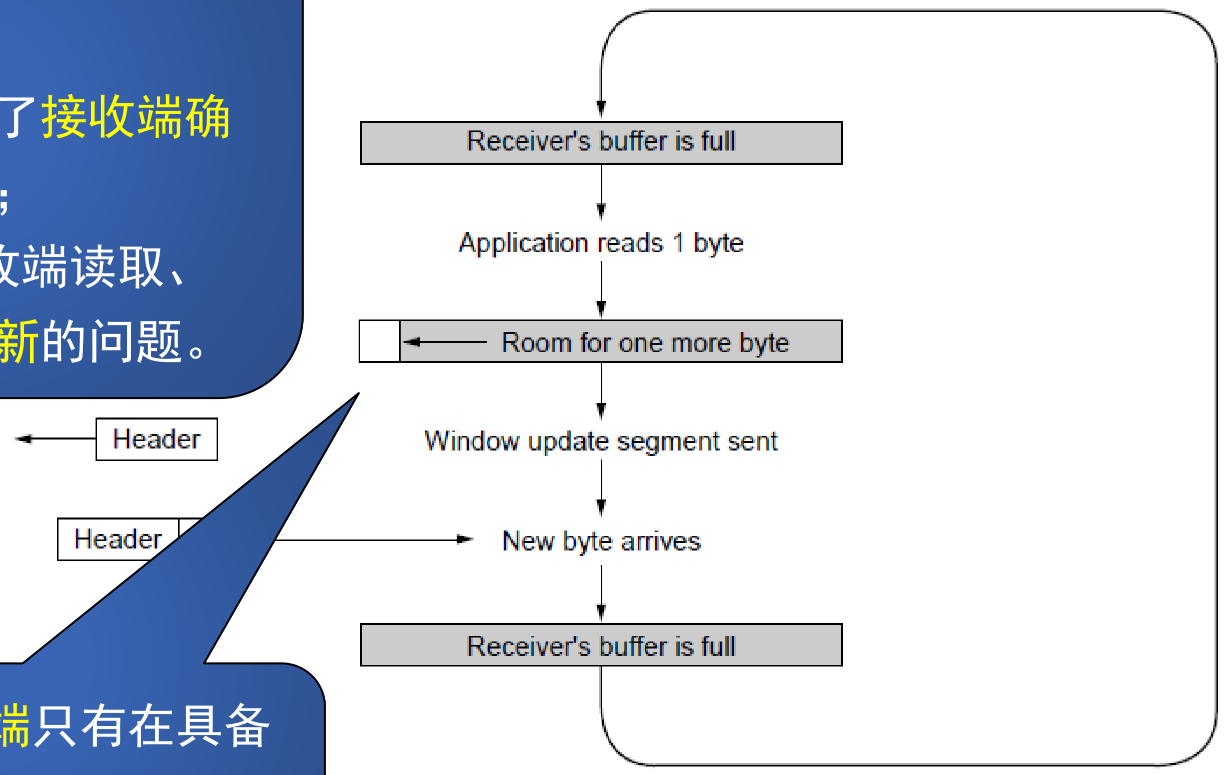
Window management in TCP

发送端可以窗口探测，以便让接收方重新声明确认号和窗口大小

Nagle：发送端立刻发开销大。只发第一次到达的数据字节；其余的缓冲直到前面的被确认，或者可以填满一个最大数据段。

TCP Sliding Window (2)

- Nagle解决发送端发一个字节的问题；
- 延迟确认解决了接收端确认太多的问题；
- Clark解决接收端读取、接收端窗口更新的问题。

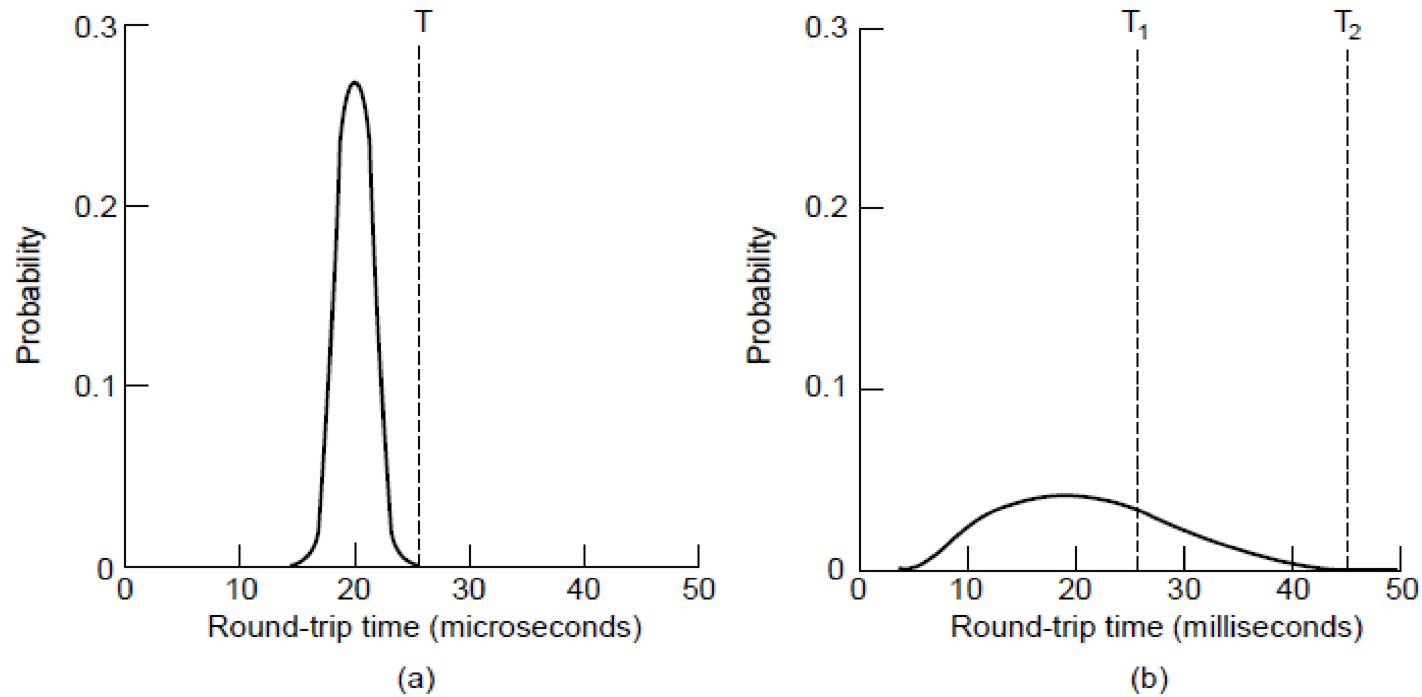


Clark：限制接收端只有在具备一半的空缓存或最大段长的空缓存时，才产生一个窗口更新

Silly window syndrome

TCP Timer Management

- **Main issue:** How do we determine the best timeout value for retransmitting segments in the face of a large standard deviation of round-trip delays:



(a) Probability density of acknowledgment arrival times in data link layer. (b) ... for TCP

TCP Timer Management

- Dynamic algorithm for adjusting the timeout interval, based on continuous measurements of network performance.

RTT	best current estimate of round-trip delay
D	estimate of deviation of round-trip delays
M	measured round-trip delay

$$\begin{aligned} RTT &= \alpha RTT + (1-\alpha)M \\ D &= \alpha D + (1-\alpha)\lvert RTT - M\rvert \\ timeout &= RTT + 4 \cdot D \end{aligned}$$


1，重传
2，持续计时器 --死锁
3，终止-time wait
4，保活计时器 (这个可以不算)—半开连接

His work redesigning TCP/IP's flow control algorithms (Jacobson's algorithm)[5][6] to better handle congestion is said to have saved the Internet from collapsing in the late 1980s and early 1990s.[7] He is also known for the TCP/IP Header Compression protocol described in RFC 1144: *Compressing TCP/IP Headers for Low-Speed Serial Links*, popularly known as Van Jacobson TCP/IP Header Compression.

He is co-author of several widely used network diagnostic tools, including traceroute, tcpdump, and pathchar. He was a leader in the development of the multicast backbone (MBone)[8] and the multimedia tools vic,[9] vat,[10] and wb.[11]

Jacobson worked at the Lawrence Berkeley Laboratory from 1974 to 1998 as a Research scientist in the Real-time Controls Group and later group leader for the Network Research Group.[12] He was Chief Scientist at Cisco Systems from 1998 to 2000.[13] In 2000 he became Chief Scientist for Packet Design, Inc. and in 2002 for a spin-off, Precision I/O.[14] He joined PARC as a research fellow in August 2006.

In January 2006 at Linux.conf.au, Jacobson presented another idea about network performance improvement, which has since been referred to as *network channels*.[15] Jacobson discussed his ideas on Content-centric networking, the focus of his current work at PARC, in August 2006 as part of the Google Tech Talks.[16][17]

Named data networking
Content-centric networking

面向主机→面向内容（where → what）

- named host → named data，以内容为中心
- 以“content name”定位内容，不需要地址

Software-Defined Information-Centric Networks

TCP Congestion Control (1)

- **Problem:** Question is how to detect and react to congestion.

- **Solution:** use a **congestion window** next to the **window granted by the receiver**. The actual window size is **the minimum of the two**.

TCP Congestion Control (补前传1)

Congestion:

- informally: “too many sources sending too much data too fast for *network* to handle”
- manifestations:
  - long delays (queueing in router buffers)
  - packet loss (buffer overflow at routers)

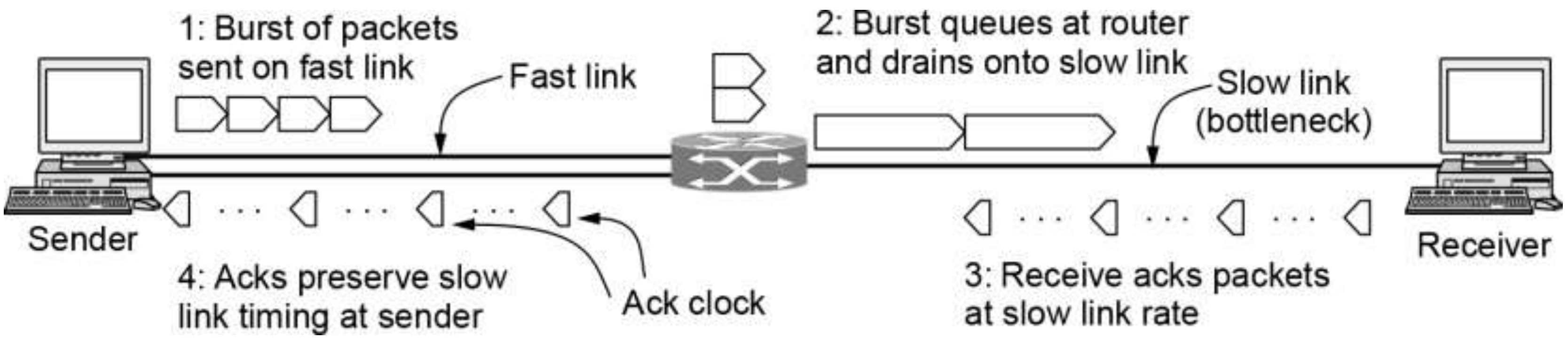
congestion control:
too many senders, sending too fast

- different from flow control!

flow control: one sender too fast for one receiver

TCP Congestion Control (1)

确认时钟：确认返回到发送端的速度反映了最慢链路的速度，从而平滑输出流量和避免不必要的路由器队列。

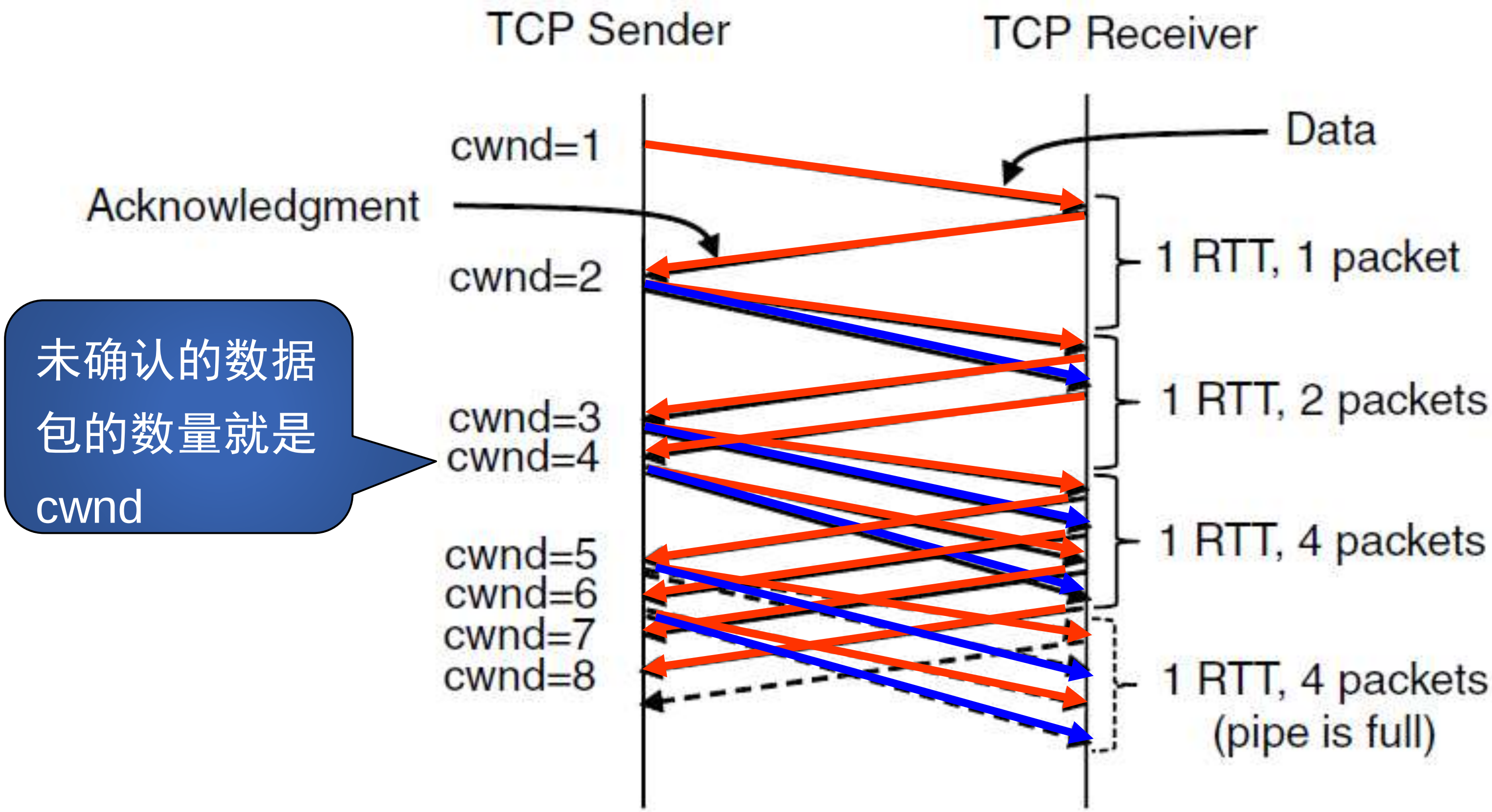


A burst of packets from a sender and the returning ack clock

TCP Congestion Control (1)

- Initialize congestion window to a maximum segment size to be used in the connection. Send it off. **If it gets acknowledged, double the size.** Repeat until failure. Leads to initial congestion window size (**slow start,慢启动**).

TCP Congestion Control (1)



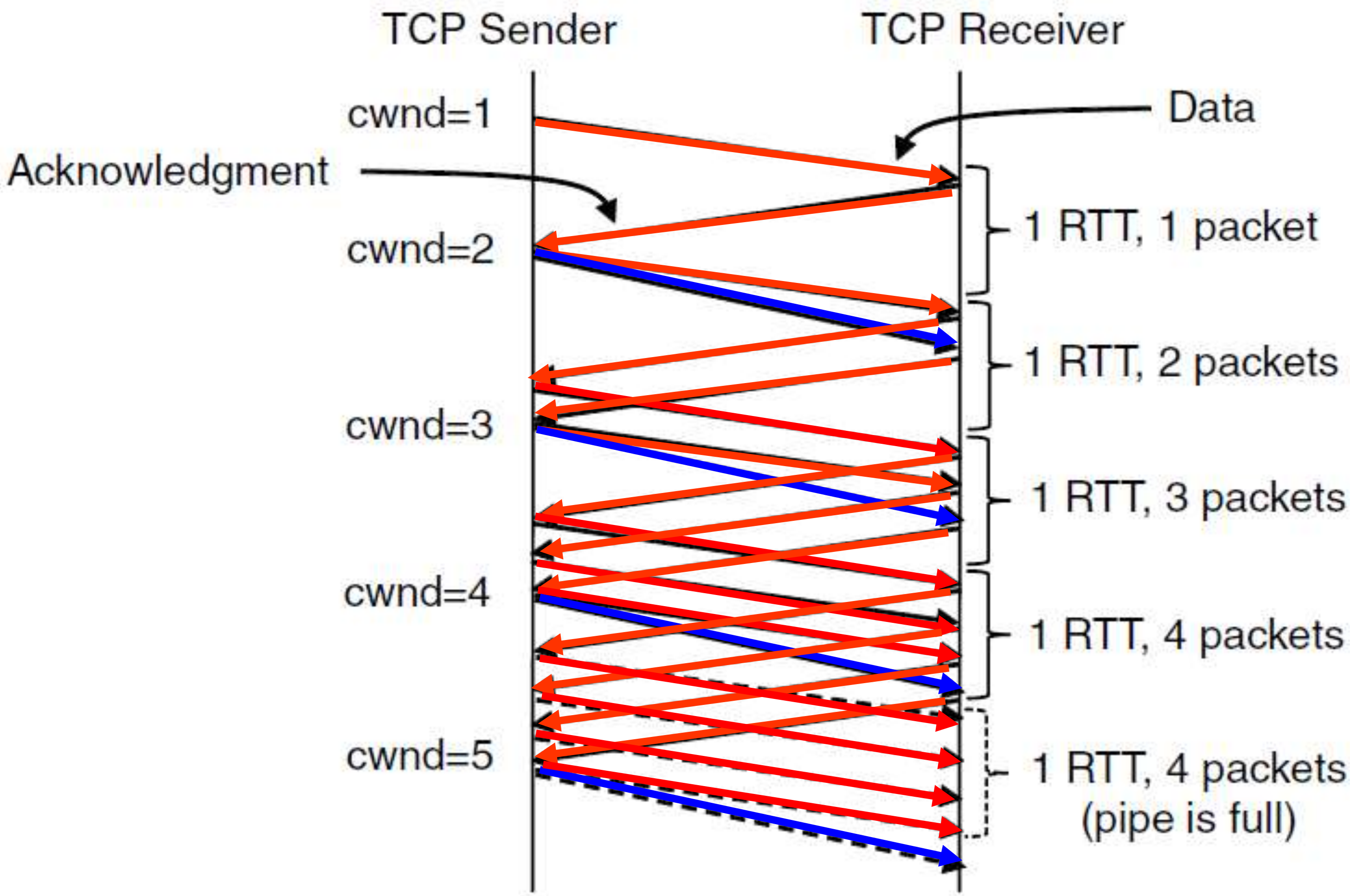
Slow start from an initial congestion window of 1 segment

TCP Congestion Control (1)

- In addition, use a **threshold**. On a timeout, lower the threshold to 50 % of the congestion window size, do a slow start (exponential) until new threshold, and *add* maximum segment size to congestion window size after that (**linear growth，拥塞避免**).

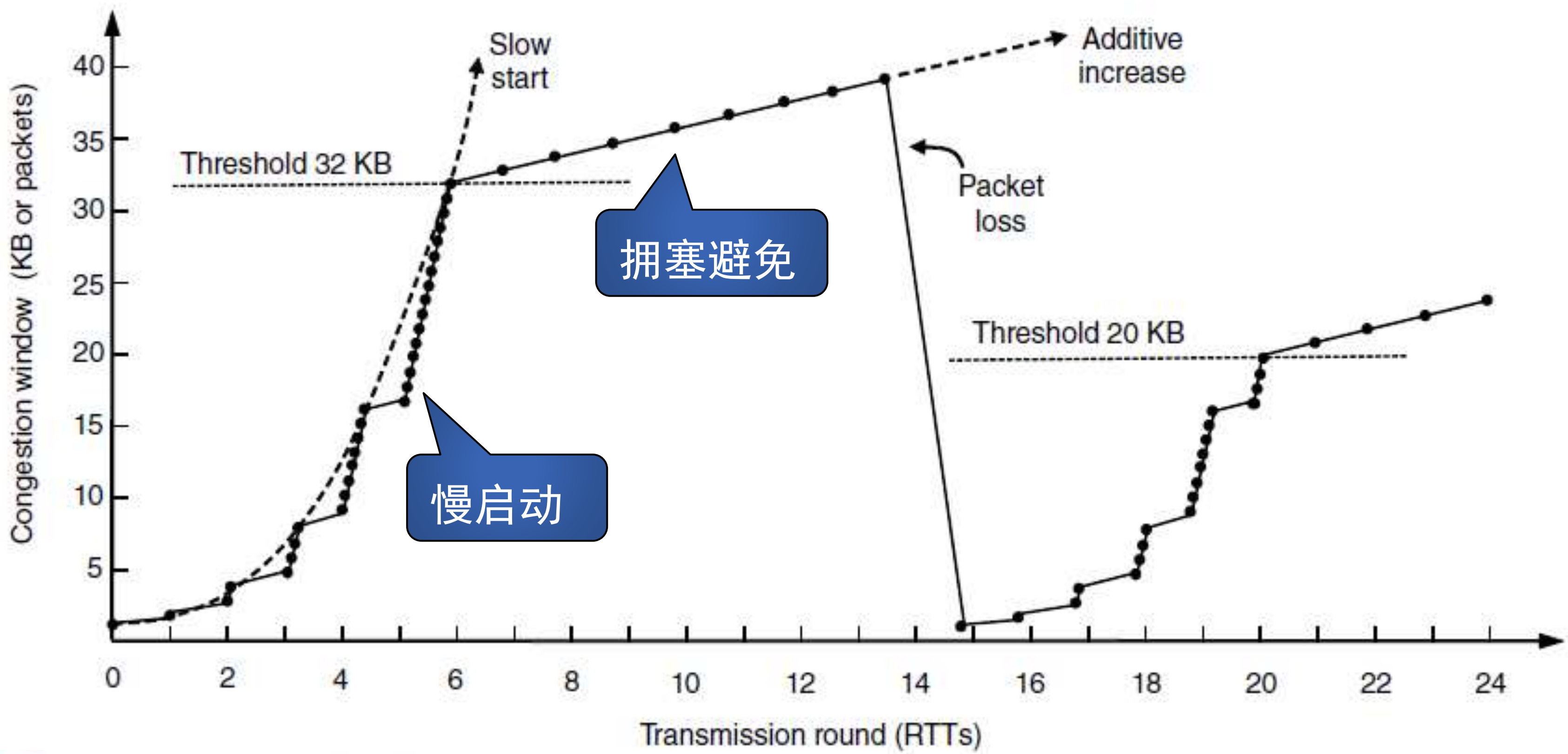
每个RTT加1个段

TCP Congestion Control (2)



Additive increase from an initial congestion window of 1 segment.

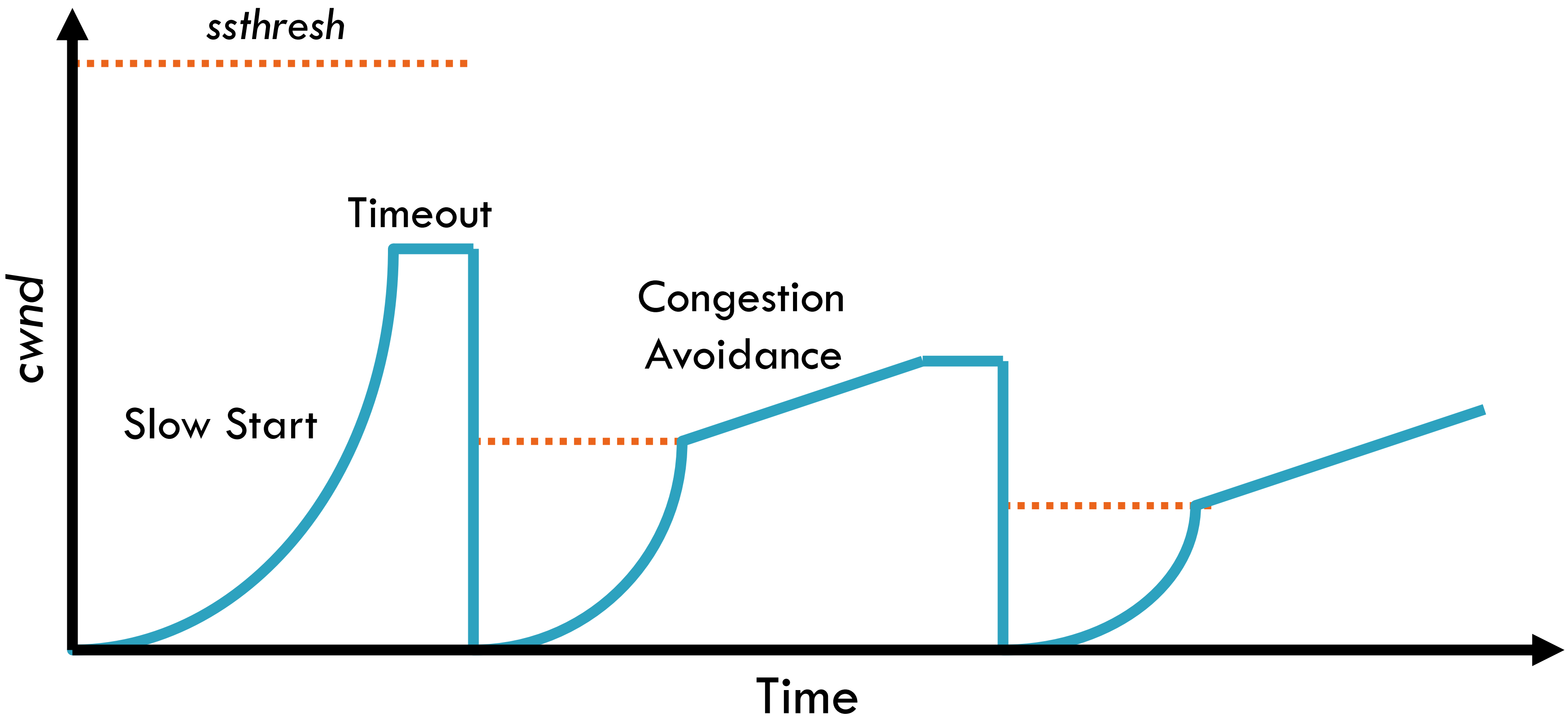
TCP Congestion Control (3)



慢启动？

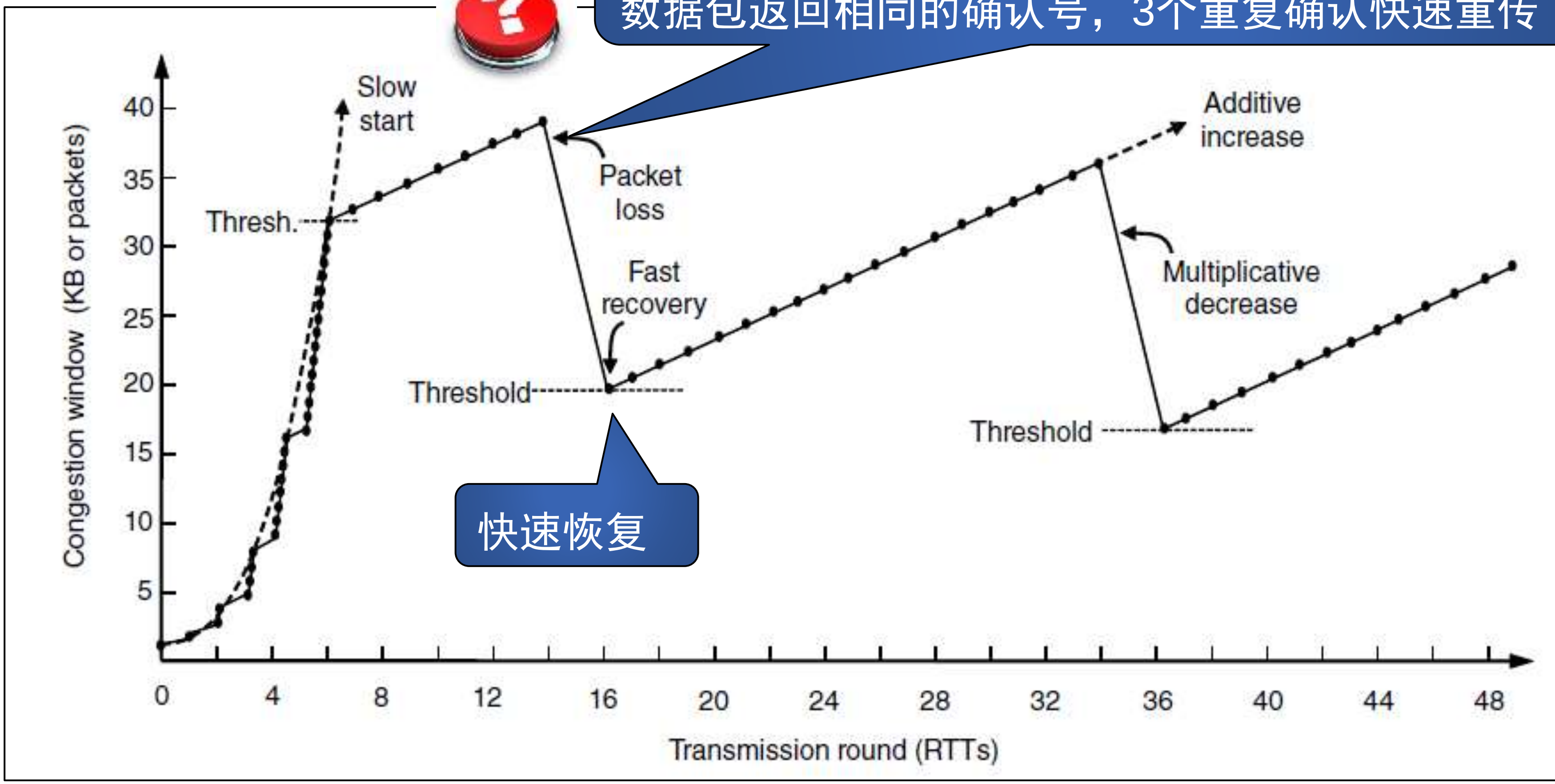
Slow start followed by additive increase in TCP Tahoe.

TCP Congestion Control (补前传3)



TCP Congestion Control (4)

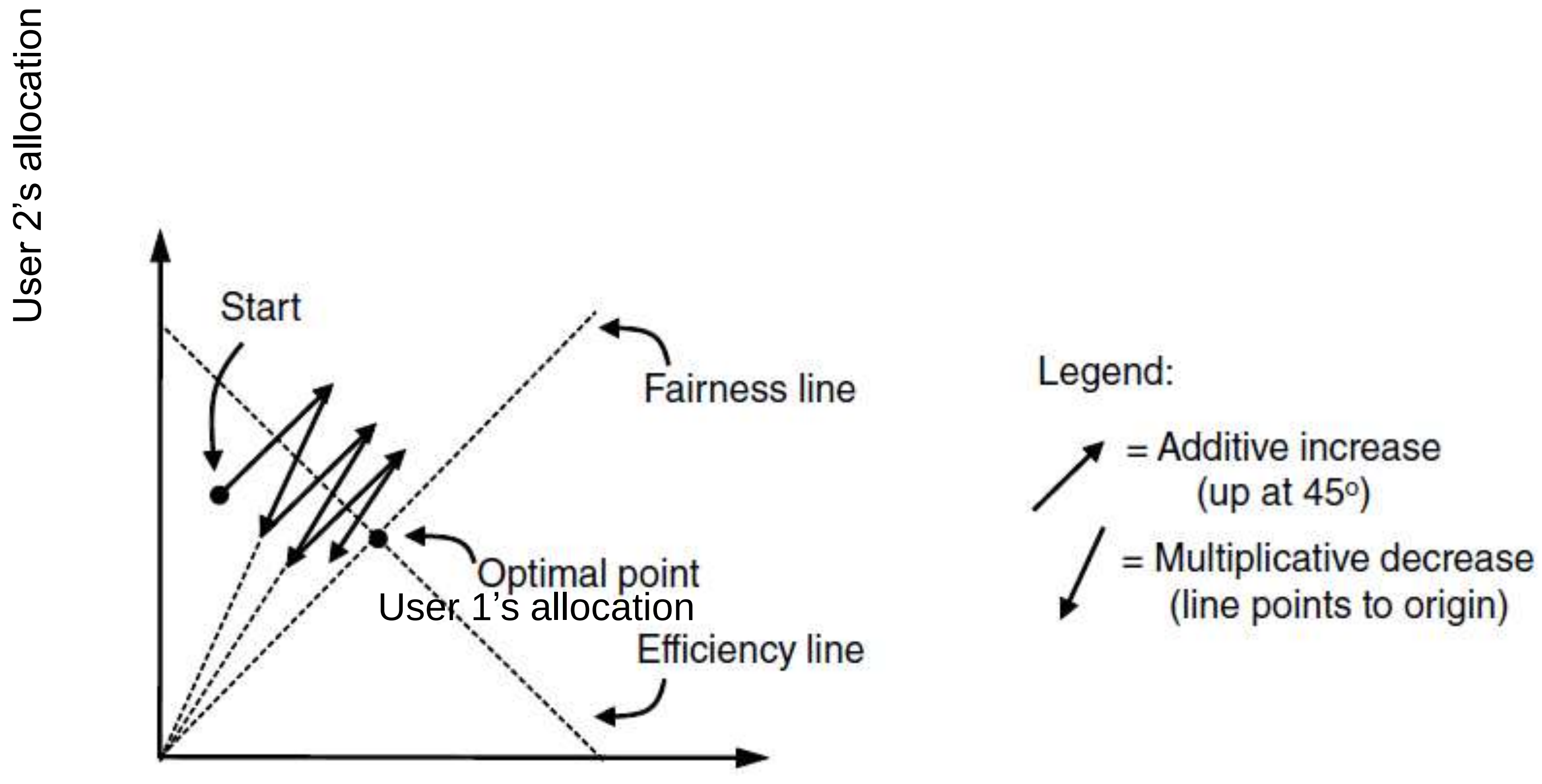
TCP是累积确认，产生重复确认，丢失数据包的后续数据包返回相同的确认号，3个重复确认快速重传



快速恢复

Fast recovery and the sawtooth pattern of TCP Reno.

TCP Congestion Control (5)



Additive Increase Multiplicative Decrease (AIMD) control law.

TCP Congestion Control (补充6)

```
// 当ack时一个可疑的ack，如sack，或者路由发送的显示拥塞控制，或者当前拥塞状态不是正常状态时。
if (tcp_ack_is_dubious(sk, flag)) {
    /* Advance CWND, if state allows this. */
    if ((flag & FLAG_DATA_ACKED) && !frto_cwnd &&
        tcp_may_raise_cwnd(sk, flag))
        // 当窗口仍然满足可以增长的条件时，进入拥塞控制，
        // 这是一个钩子函数，具体实现由具体拥塞控制算法来实现，
        // 对于reno而言可能是慢启动，可能是拥塞避免。
        tcp_cong_avoid(sk, ack, prior_in_flight);
    // 处理拥塞状态机，暂时不展开
    tcp_fastretrans_alert(sk, prior_packets - tp->packets_out,
                 flag);
} else {
    // 当这个ack是一个正常的数据确认包，进入拥塞控制
    if ((flag & FLAG_DATA_ACKED) && !frto_cwnd)
        tcp_cong_avoid(sk, ack, prior_in_flight);
}
```

TCP Congestion Control (补充6)

```
/*
 * TCP Reno congestion control
 * This is special case used for fallback as well.
 */
/* This is Jacobson's slow start and congestion avoidance.
 * SIGCOMM '88, p. 328.
 */
void tcp_reno_cong_avoid(struct sock *sk, u32 ack, u32 in_flight)
{
    struct tcp_sock *tp = tcp_sk(sk);

    if (!tcp_is_cwnd_limited(sk, in_flight))
        return;

    /* In "safe" area, increase. */
    // 小于阈值会进入慢启动环节，不重置窗口的慢启动。
    if (tp->snd_cwnd <= tp->snd_ssthresh)
        tcp_slow_start(tp);

    /* In dangerous area, increase slowly. */
    else if (sysctl_tcp_abc) {
        /* RFC3465: Appropriate Byte Count
         * increase once for each full cwnd acked
         */
        // RFC3465的拥塞避免算法，使用bytes_acked来作为修改拥塞窗口的判断条件
        if (tp->bytes_acked >= tp->snd_cwnd*tp->mss_cache) {
            tp->bytes_acked -= tp->snd_cwnd*tp->mss_cache;
            if (tp->snd_cwnd < tp->snd_cwnd_clamp)
                tp->snd_cwnd++;
        }
    } else {
        // 拥塞避免
        tcp_cong_avoid_ai(tp, tp->snd_cwnd);
    }
}
```

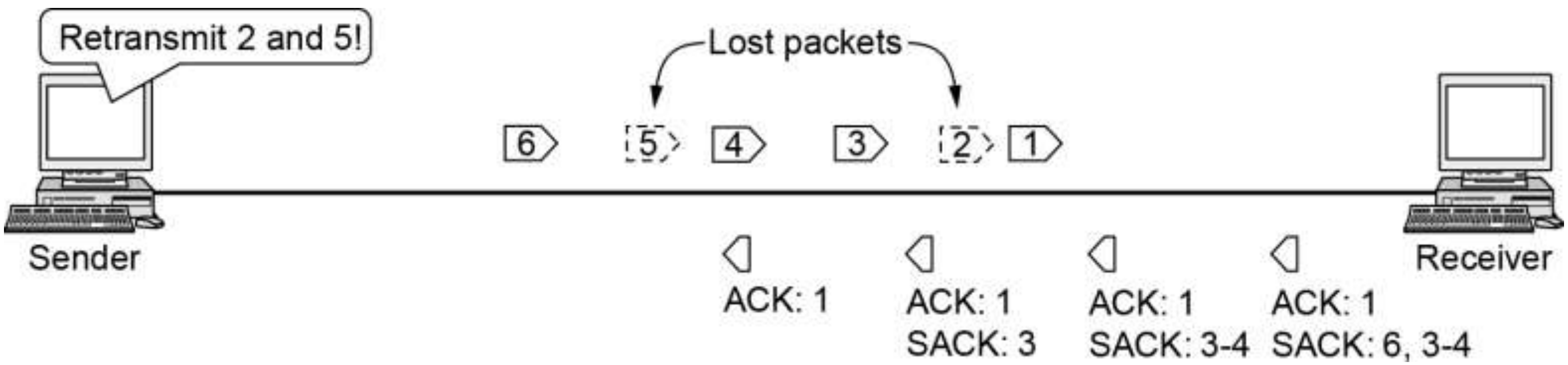
TCP Congestion Control (补充6)

```c
void tcp_slow_start( struct tcp_sock *tp )
{
 int cnt ; /* increase in packets */

/* RFC3465 : ABC slow start
 * Increase only after a full MSS of bytes is acked
 *
 * TCP sender SHOULD increase cwnd by the number of
 * previously unacknowledged bytes ACKed by each incoming
 * acknowledgment , provided the increase is not more than L
 */
 /* ack的数据少于MSS */
 if ( sysctl_tcp_abc && tp->bytes_acked < tp->mss_cached )
        return ;

/* 此时不是应该进入拥塞避免? */
 if ( sysctl_tcp_max_ssthresh >0 && tcp->snd_cwnd >sysctl_tcp_max_ssthresh)
        cnt = sysctl_tcp_max_ssthresh >> 1 ; /* limited slow start */
 else
        cnt = tp->snd_cwnd ; /* exponential increase */

/* RFC3465 : ABC
 * We MAY increase by 2 if discovered delayed ack
 */
/* 如果接收方启用了延时确认，此时收到的确认代表两个MSS数据报*/
if ( sysctl_tcp_abc >1 && tp->bytes_acked >= 2*tp->mss_cache )
        cnt <<= 1 ;

tp->bytes_acked = 0 ;
tp->snd_cwnd_cnt += cnt ; /* 此时snd_cwnd_cnt等于snd_cwnd或2*snd_cwnd */

 while( tp->snd_cwnd_cnt >= tp->snd_cwnd ) {
        tp->snd_cwnd_cnt -= tp->snd_cwnd ;
        if( tp->snd_cwnd < tp->snd_cwnd_clamp )
                tp->snd_cwnd++ ;
 }
}
EXPORT_SYMBOL_GPL( tcp_slow_start ) ;
```

TCP Congestion Control (补充6)

```
/* In theory this is tp->snd_cwnd += 1 / tp->snd_cwnd (or alternative w) */
void tcp_cong_avoid_ai(struct tcp_sock *tp, u32 w)
{
    // 每次cnt++，直到w次后snd_cwnd++，即单位 1 / w
    if (tp->snd_cwnd_cnt >= w) {
        if (tp->snd_cwnd < tp->snd_cwnd_clamp)
            tp->snd_cwnd++;
        tp->snd_cwnd_cnt = 0;
    } else {
        tp->snd_cwnd_cnt++;
    }
}
```

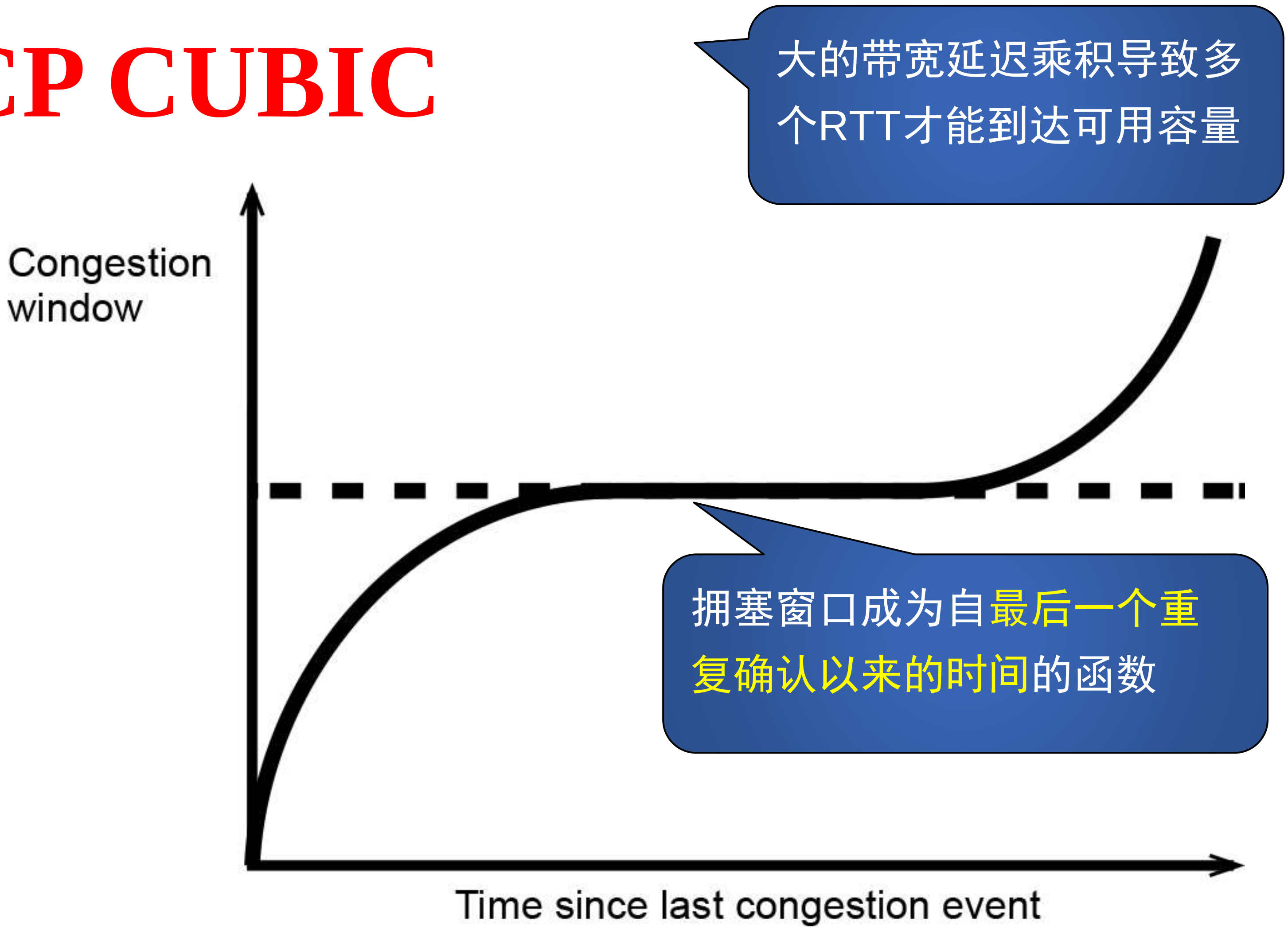
TCP Congestion Control



选择确认：TCP选项字段

Selective acknowledgements

TCP CUBIC



Evolution of TCP CUBIC Congestion Window

6.6 Transport Protocols and Congestion Control

- QUIC: Quick UDP Internet Connections
- BBR: Congestion control based on bottleneck bandwidth
- The future of TCP

QUIC(引言1)

1.连接建立延时低
2.改进的拥塞控制
3.基于stream和connection级别的流量控制
4.没有队头阻塞的多路复用

"车同轨、书同文"，掌握网络国际标准非常重要！需要艰苦奋斗、自主创新。

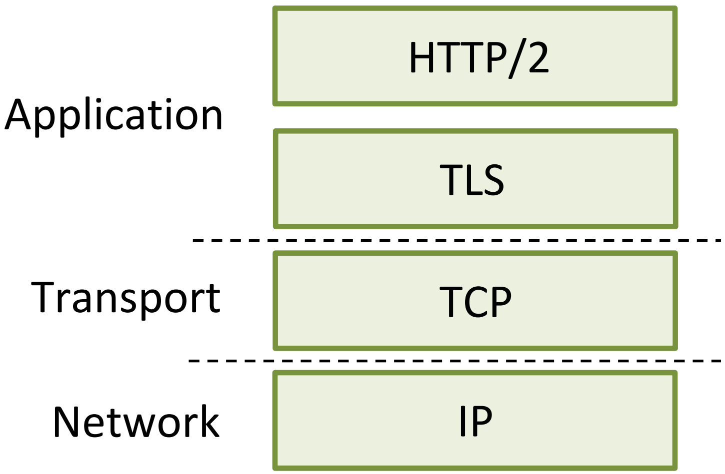
- Quick UDP Internet Connections
  - Deployed by Google in 2013
  - Now an Internet standard (part of HTTP/3)
    draft-ietf-quic-http,19 July 2021
  - Provides reliable in-order byte streams, but using UDP
- Motivation
  - TCP is difficult to do quickly at scale (OS changes)
  - Privacy/integrity are critical, so encrypt everything

传输层安全协议

- QUIC is intended to replace TCP, TLS, and HTTP/2

QUIC (补充2)

- application-layer protocol, on top of UDP
  - increase performance of HTTP
  - deployed on many Google servers, apps (Chrome)

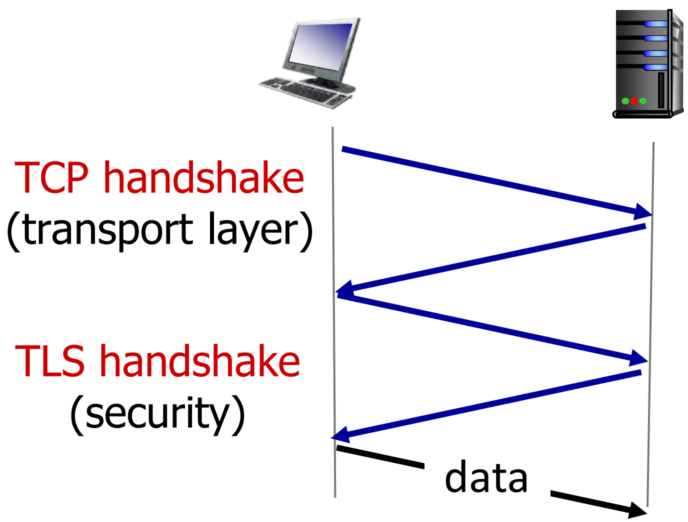


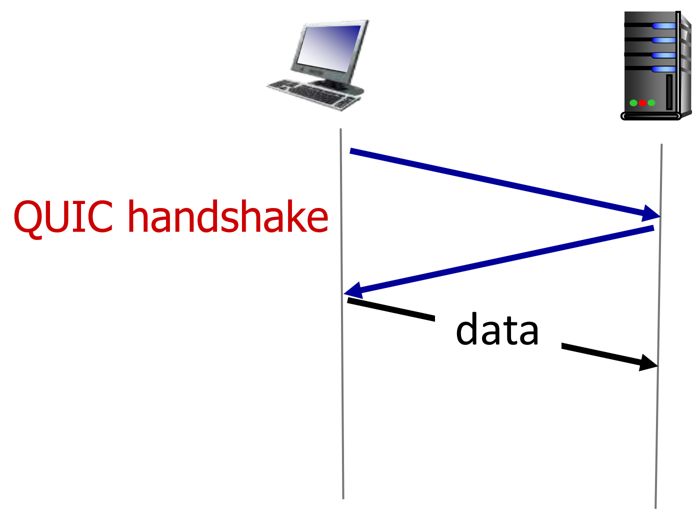
HTTP/2 over TCP

QUIC (补充3)

- “0-RTT” connection establishment
  - Allows resumption of connections if client/server have communicated before
  - No expensive 3-way handshake or TLS connection setup
- Reduced “head of line” blocking
  - Packet loss impacting one HTTP/2 stream does not affect others in the same connection
- Improved congestion control

QUIC (补充4)





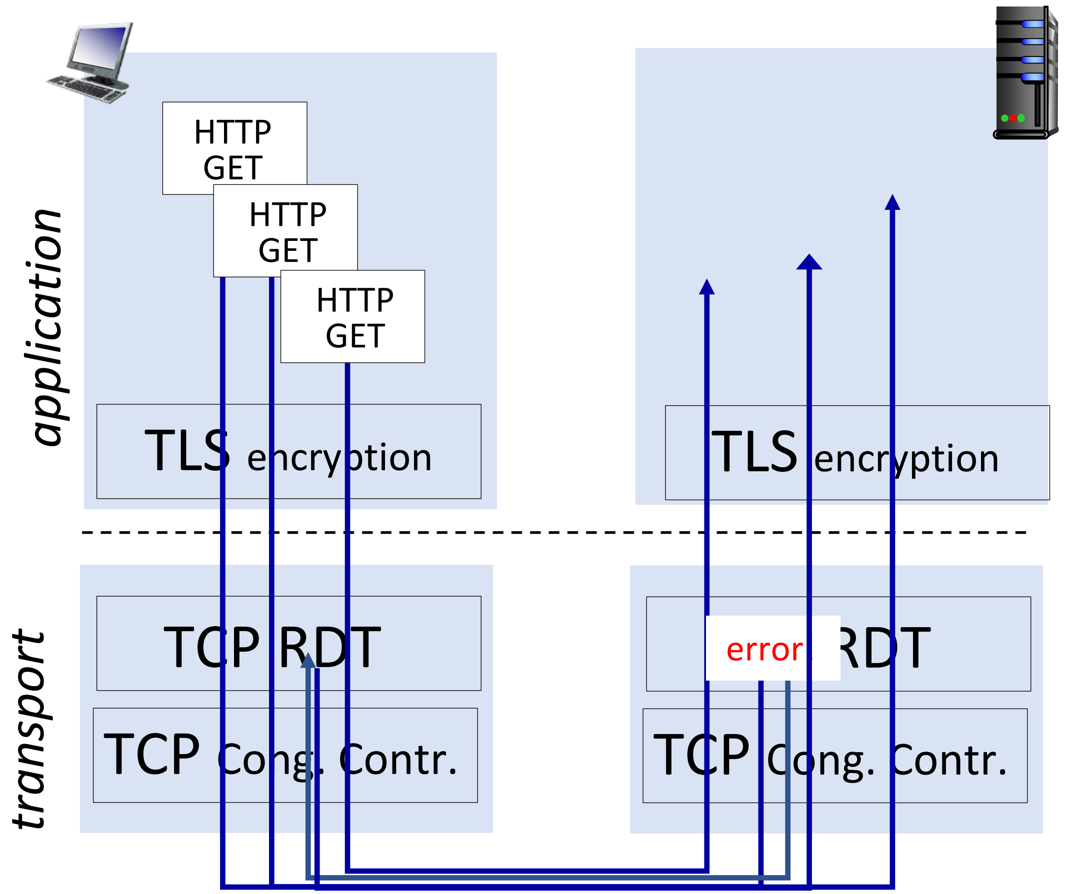
TCP (reliability, congestion control state) + TLS (authentication, crypto state)

- 2 serial handshakes

QUIC: reliability, congestion control, authentication, crypto state

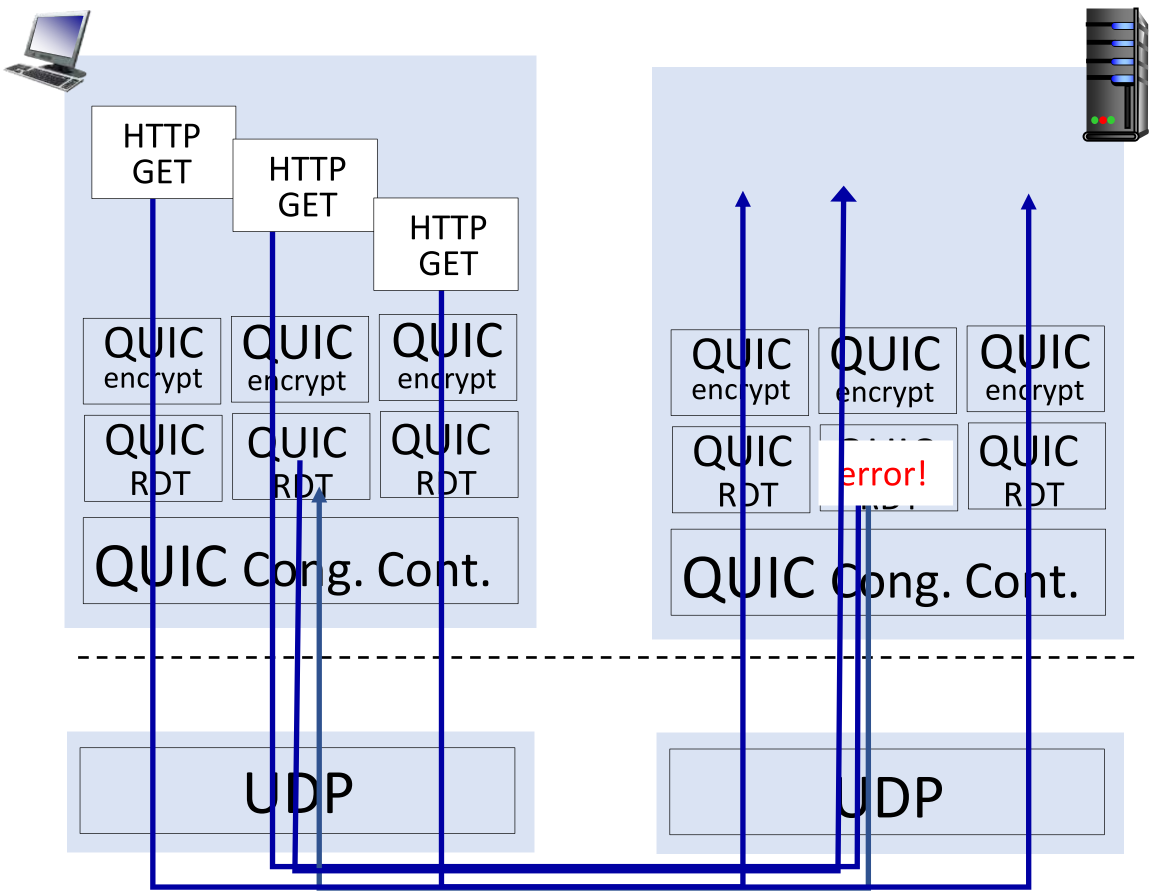
- 1 handshake

QUIC (补充5)



(a) HTTP 1.1

QUIC (补充5)



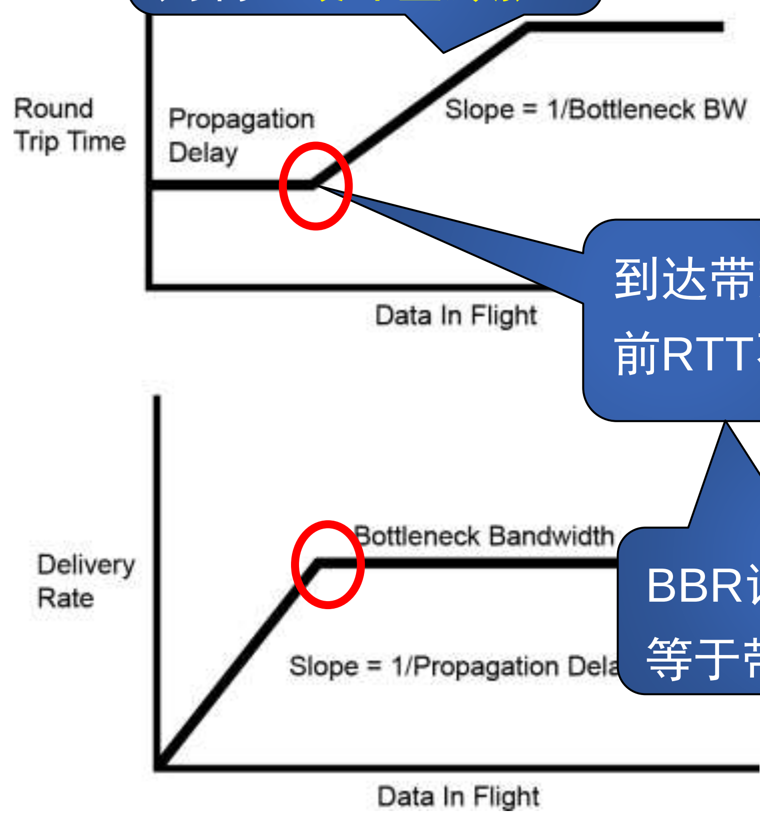
(b) HTTP/2 with QUIC: no HOL blocking

BBR: Congestion Control Based on Bottleneck Bandwidth

对于瓶颈缓冲区很大时，引起缓冲区膨胀，发送太快的发送方的拥塞事件被延迟了。数据本身也延迟

排队延迟增加到队列满（缓冲区膨胀）

BBR跟踪瓶颈带宽与延迟RTT。



到达带宽延迟积之前RTT不会增加

BBR让途中的包等于带宽延迟积

BBR Operating Point

The Future of TCP

- TCP will continue to evolve
- TCP issues
  - Does not provide transport semantics applications want
  - Application must deal with problems not solved by TCP
- Proposals providing a slightly different interface
  - SCTP and SST
- Must deal with “If it ain’t broke, don’t fix it” mentality

Stream Control Transmission Protocol
Structured Stream Transport

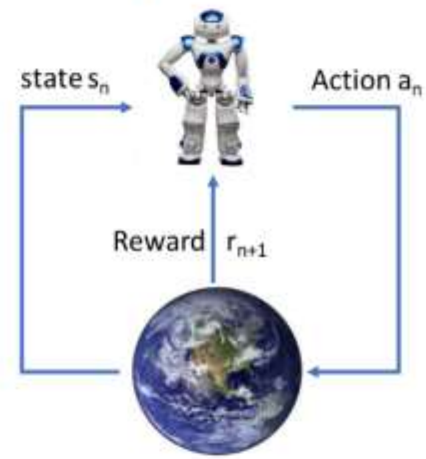
如果不打破就不能解决VS
用户要求更多的功能

TCP futures (补充1)

Improving TCP Congestion Control with Machine Intelligence

Reinforcement learning based TCP (RL-TCP)

Our RL-TCP
- Add variable to state
- Tailor action space to under-buffered bottleneck
- Propose a new temporal credit assignment of reward



- Objective of RL-TCP
  - Learn to adjust cwnd to increase an utility function

$$U = \log\left(\frac{tp}{B}\right) - \delta_1 \log(d) + \delta_2 \log(1-p)$$

Bottleneck bandwidth throughput delay Packet loss rate

6.7 Performance Issues

- Performance problems in computer networks
- Network performance measurement
- System design for better performance
- Fast TPDU processing
- Protocols for high-speed networks

Network Performance Measurement (1)

Steps to performance improvement

- Measure relevant network parameters, performance.
- Try to understand what is going on.
- Change one parameter.

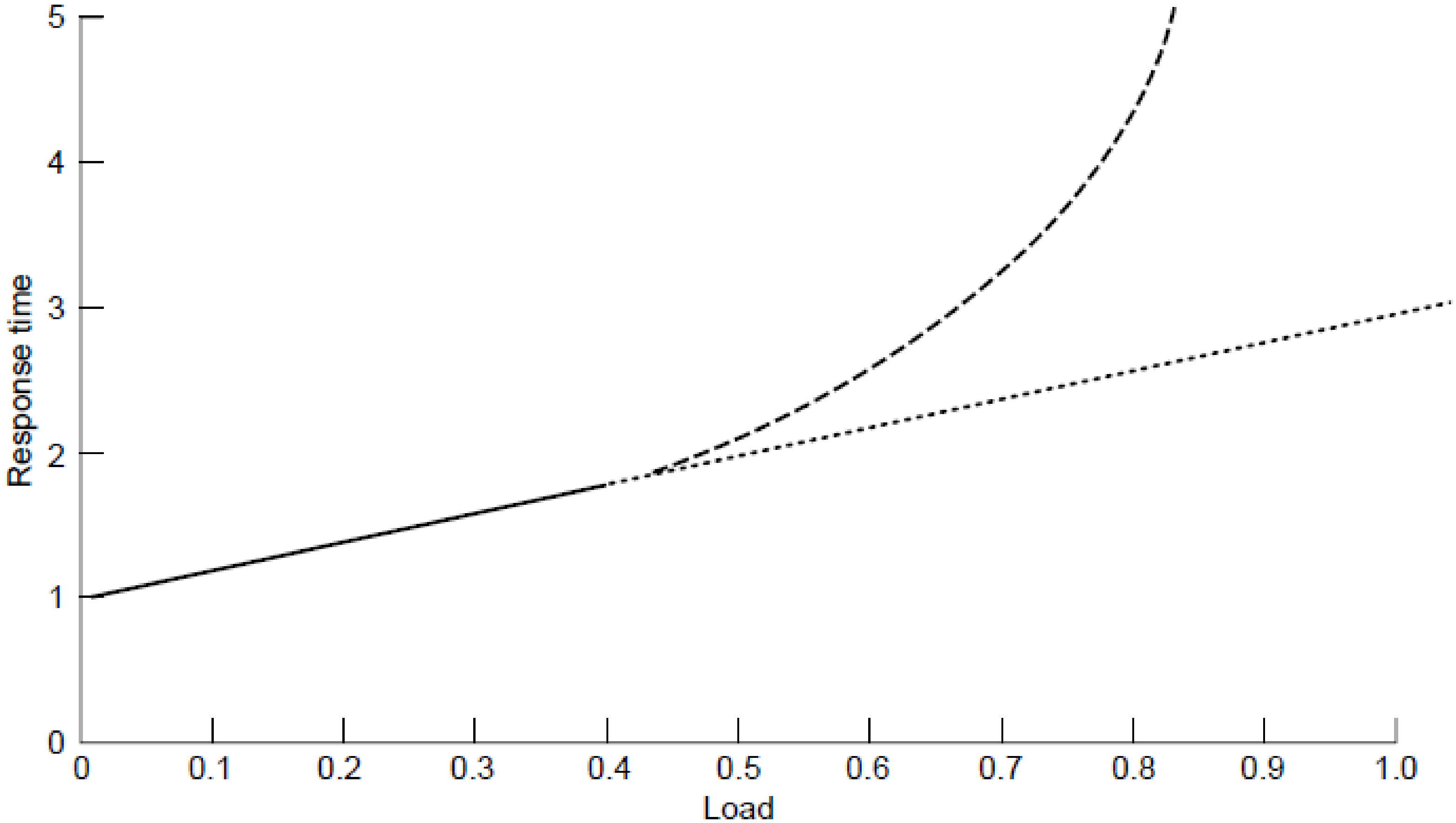
Network Performance Measurement (2)

Issues in measuring performance

- Sufficient sample size
- Representative samples
- Clock accuracy
- Measuring typical representative load
- Beware of caching
- Understand what you are measuring
- Extrapolate with care

小心推断

Network Performance Measurement (3)

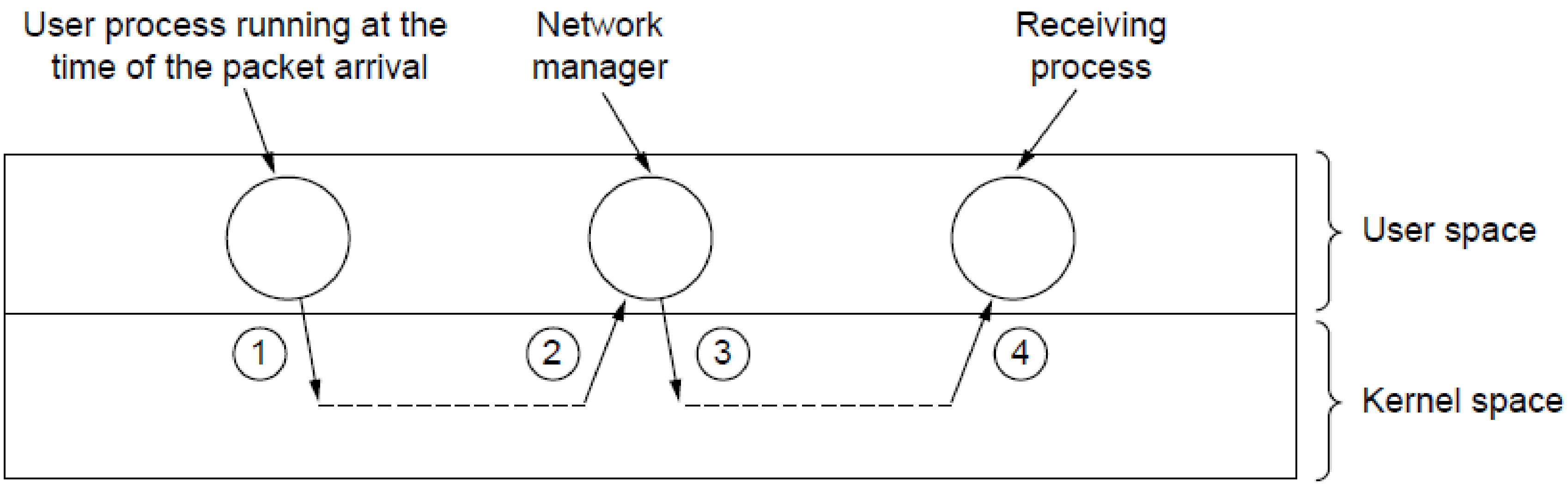


Response as a function of load.

System Design for Better Performance (1)

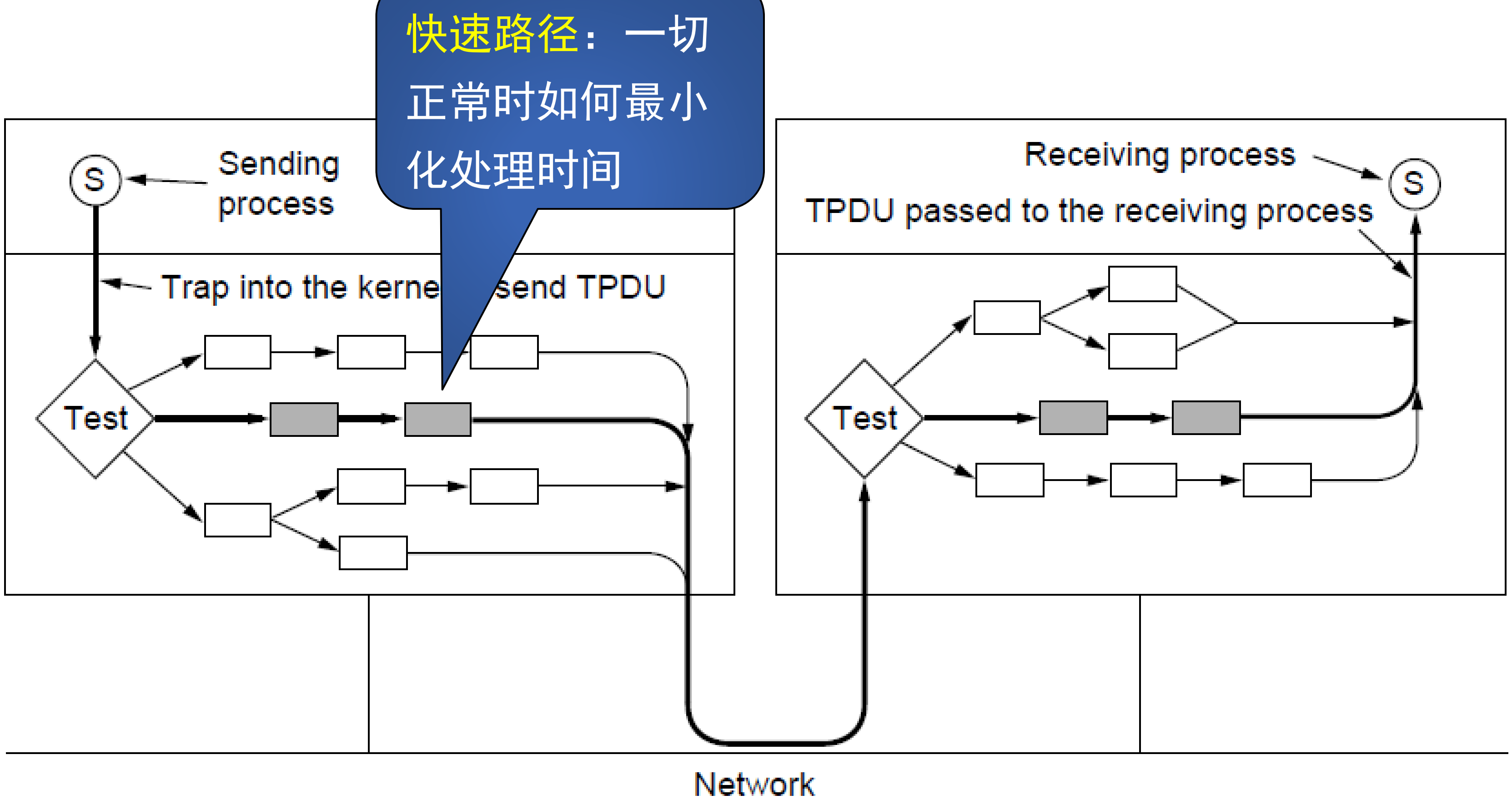
- CPU speed more important than network speed
- Reduce packet count to reduce software overhead
- Minimize data touching
- Minimize context switches
- Minimize copying
- You can buy more bandwidth but not lower delay
- Avoiding congestion is better than recovering from it
- Avoid timeouts

System Design for Better Performance (2)



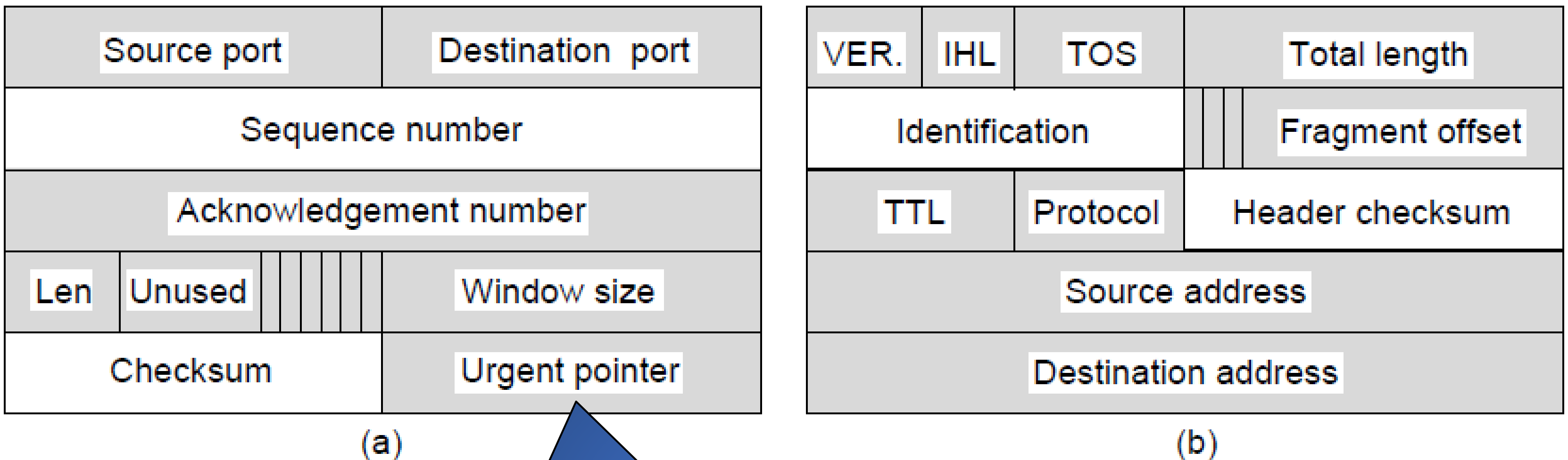
Four context switches to handle one packet
with a user-space network manager.

Fast TPDU Processing (1)



The fast path from sender to receiver is shown with a heavy line. The processing steps on this path are shaded.

Fast TPDU Processing (2)

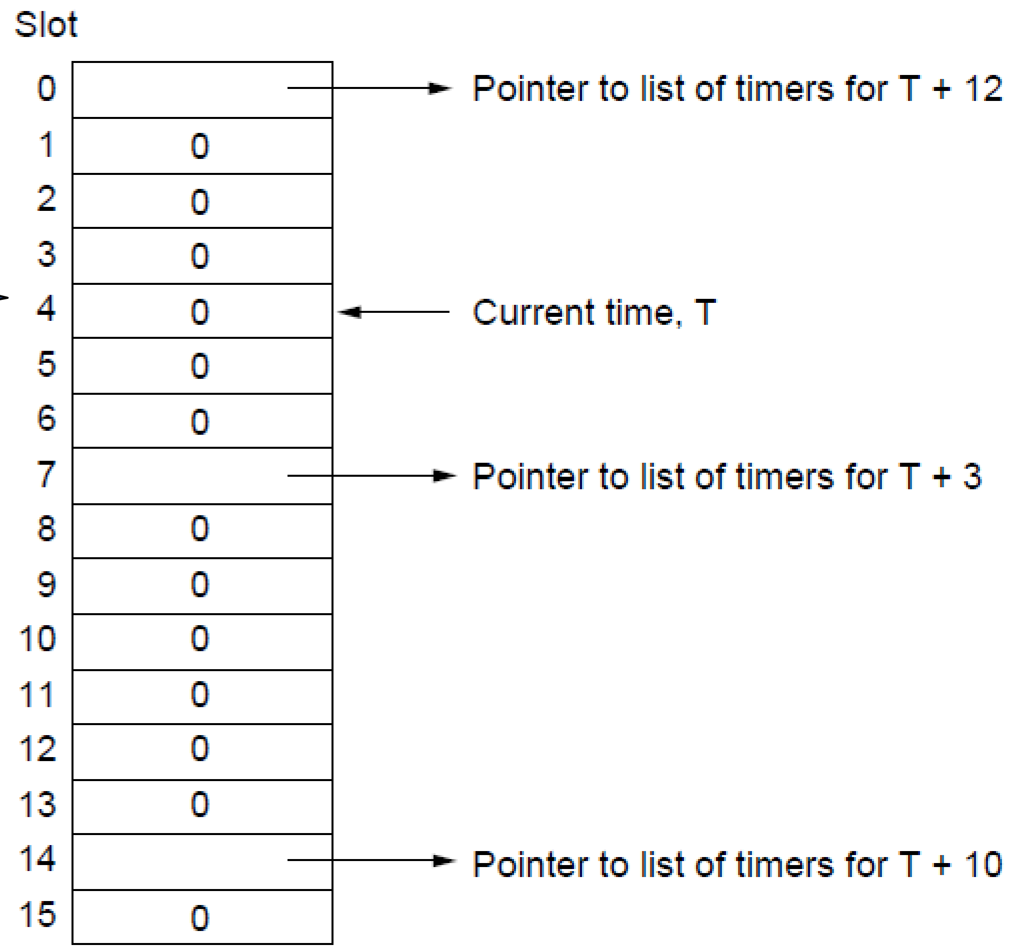


1) 缓冲区管理优化：
在临时缓冲区替换

(a) TCP header. (b) IP header. In both cases, the shaded fields are taken from the prototype without change.

Protocols for High-Speed Networks (1)

2) 定时器优化：
计时轮



A timing wheel

Performance Problems in Computer
etworks

1 序号绕回问题

2^{32}，32位 10M，57分钟; 1Gbps, 34秒，小于Internet最大包生存期120s

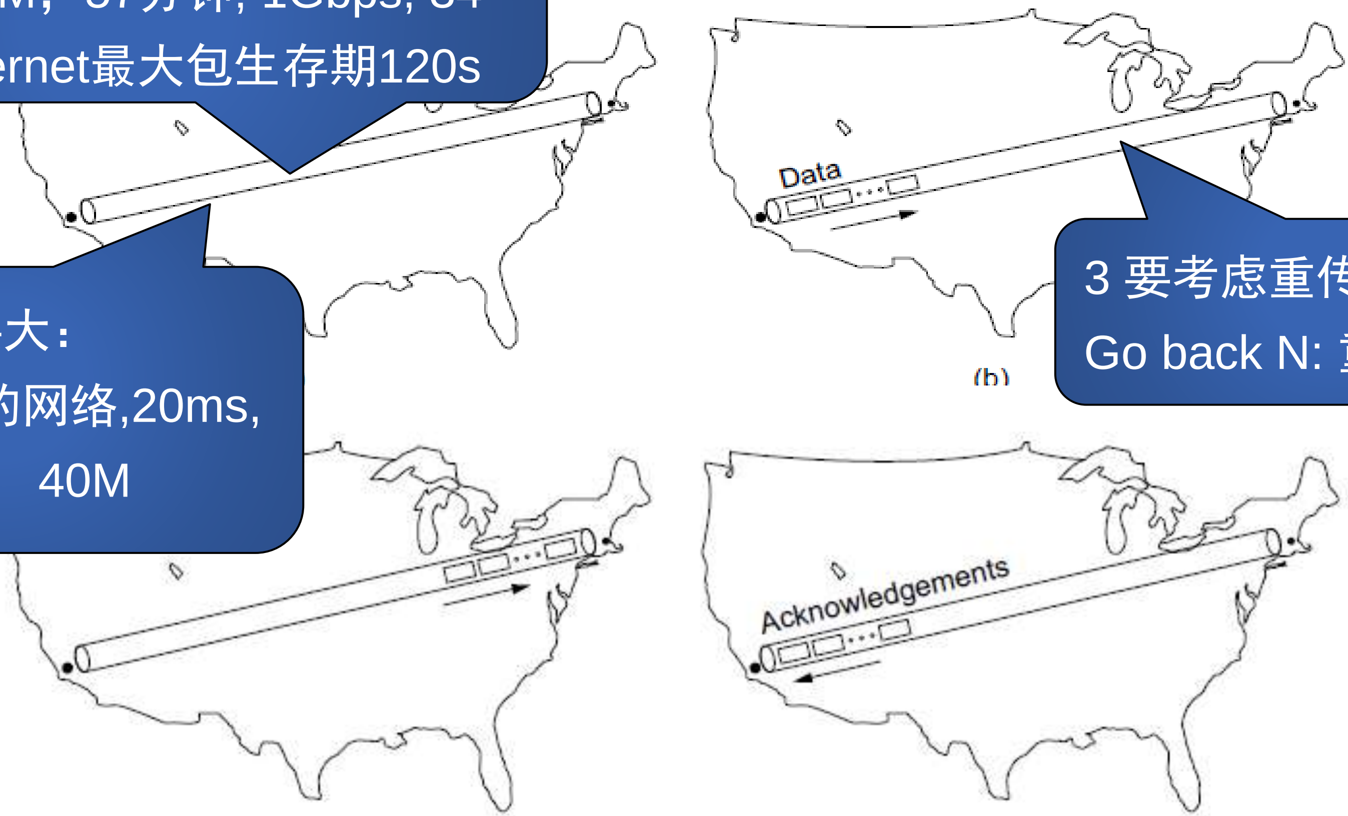
2流量窗口要大：

对与1Gbps的网络,20ms,

带宽延时积： 40M

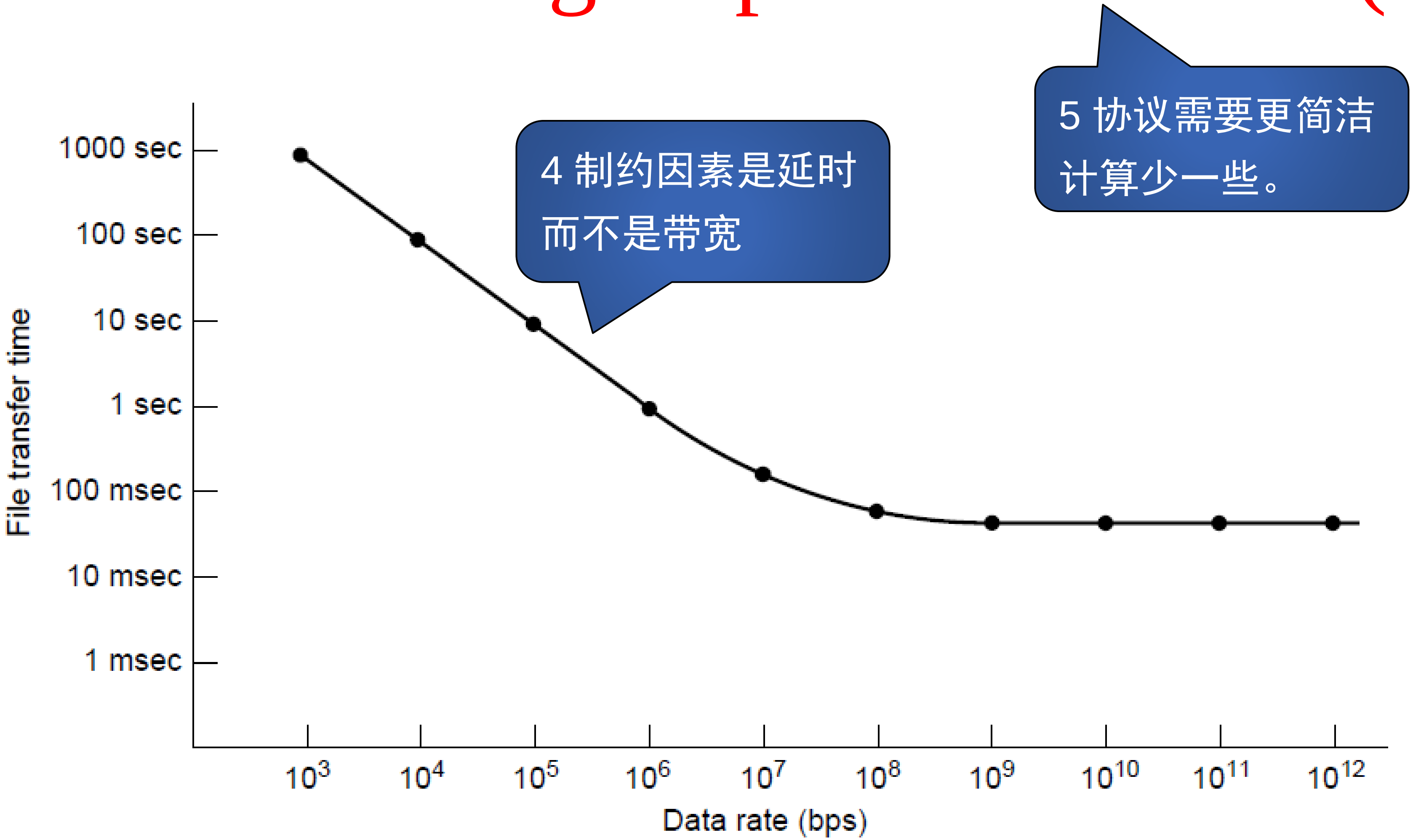
3 要考虑重传策略

Go back N: 重传 40M



The state of transmitting one megabit from San Diego to Boston. (a) At *t = 0.* (b) *After 500 μ sec.*
(c) *After 20 msec.* (d) *After 40 msec.*

Protocols for High-Speed Networks (2)



Time to transfer and acknowledge a 1-megabit file over a 4000-km line

The End