传输层协议

yangzhang@whut.edu.cn
yzhang.org

端到端通信

- 点到点（Point to point）与端到端（End to end）
  - 直接相连节点之间对等实体的通信，叫点到点通信
  - 端到端：源主机上数据来源于其网络应用程序，最终要（通过IP层）送到目标主机上某个特定网络应用程序
    - 这样在源主机和目标主机之间好像有一条直接的数据传输通路，
    - 直接把源主机应用程序产生的数据，传输到目标主机使用这些数据的应用程序

IP是点到点还是端到端?

一般说TCP/IP提供端到端的通信

端口（Port）

- 端口是TCP/IP协议族中，应用层进程与传输层协议实体间的通信接口
- 在OSI七层协议描述中，将其称为应用层进程与传输层协议实体间的服务访问点（SAP）
  - 应用层进程通过系统调用与某个传输层端口进行绑定，然后通过该接口接收或发送数据。
  - 类似于文件描述符，每个端口都拥有一个叫作端口号（port number）的16位整数型标识符。
  - 可以用端口标识通信的网络应用程序

- 在TCP/IP协议中，传输层使用的端口号用一个16位的二进制数表示

- 在传输层如果使用TCP协议进行进程通信，则可用的端口号共有64K个

- UDP也是传输层一个独立于TCP的协议，因此使用UDP协议时也有64K个不同的端口

UDP

源端口	目标端口
UDP长度	UDP校验和

TCP

源端口								目标端口
序　号								
确认号								
数据偏移	保留	URG	ACK	PSH	RST	SYN	FIN	窗口
校验和								紧急指针
选项								填充

UDP header and TCP header

- 从实现的角度讲，端口是一种抽象的软件机制，包括一些数据结构和I/O缓冲区。
    - 进程通过系统调用与某端口建立绑定关系后，传输层传给该端口的数据都被相应进程接收，相应进程发给传输层的数据都通过该端口输出。
- 在TCP/IP实现中端口操作类似于一般的I/O操作。
    - 进程获取一个端口，相当于获取本地唯一的I/O文件，可以用一般的读写原语访问。

端口号的分配

- 网络进程通信前必须获知对方的进程地址。
  - 由于网络应用程序大多采用C/S模式开发，通信总是由客户机发起，因此事先只需让客户机知道服务器的进程地址即可。
- Internet中为客户服务的众所周知的服务有限。
- TCP/IP协议采用了全局分配（静态分配）和本地分配（动态分配）相结合的分配方法。
- 对于TCP或UDP，将它们的全部65535个端口号分为保留端口号和自由端口号两部分

- 保留端口号
  - 范围是0－1023，只占少数
  - 采用全局分配或集中控制的方式，由Internet号分配机构IANA（Internet Assigned Numbers Authority）根据需要进行统一分配
  - 256～1023之间的端口号通常都是由Unix系统占用，以提供一些特定的Unix服务

- 众所周知（Well known）端口
  - 0-255
  - 端口0不使用/特殊情况才使用
  - 客户要使用的由服务器进程提供的服务，客户已知道它们的端口号。如FTP服务器的TCP端口号是21

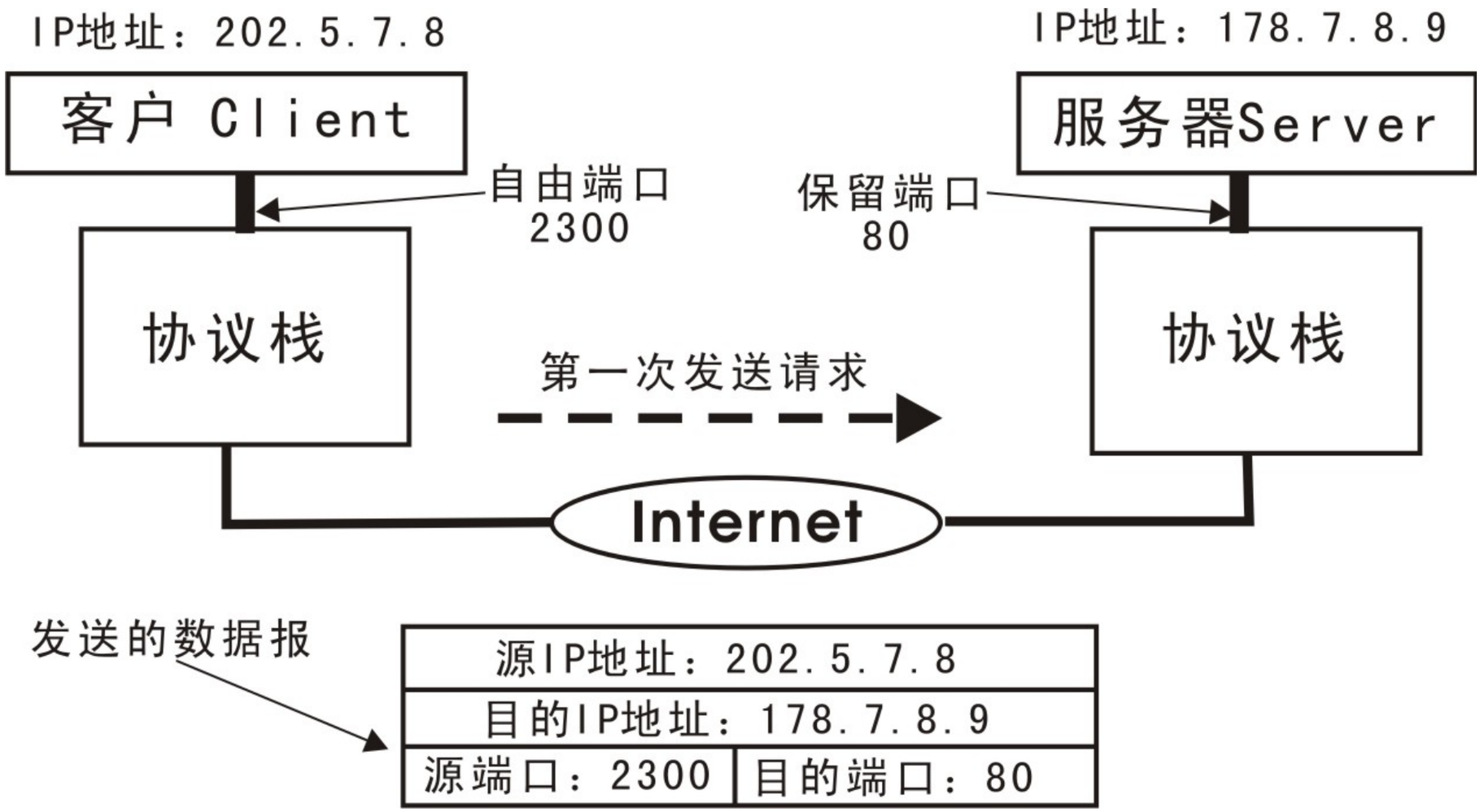
TCP端口号	关键词	描 述
20	FTP-DATA	文件传输协议（数据连接）
21	FTP	文件传输协议（控制连接）
23	Telnet	远程登录协议
25	SMTP	简单邮件传输协议
53	DOMAIN	域名服务器
80	HTTP	超文本传输协议
110	POP3	邮局协议3
119	NNTP	新闻传输协议

- 常见的保留端口（TCP）

UDP端口号	关键词	描 述
53	**DOMAIN**	域名服务器
67	**BOOTPS**	引导协议服务器
68	**BOOTPC**	引导协议客户机
69	**TFTP**	简单文件传输协议
161	**SNMP**	简单网络管理协议
162	**SNMP-TRAP**	简单网络管理协议陷阱

- 常见的保留端口（UDP）

- 客户端口号 / 临时端口号
  - 在客户程序要进行通信之前，动态的从系统申请分配一个端口号
  - 客户以该端口号为源端口，使用某个众所周知的端口号为目标端口号（如在TCP协议上要进行文件传输时使用21）进行客户端到服务器端的通信
  - 通信完成后，客户端的端口号就被释放掉



- 客户端与服务器第一次通信

- 大多数TCP/IP实现时，给临时端口分配1024～5000之间的端口号。
- 大于5000的端口号是为其它服务预留的，为Internet上并不常用的服务

- 端口总结：

  - 两台要通信的主机，每一端要使用一个二元地址（IP地址，端口号），才可以完成它们之间的通信。端到端之间的一条通信就可能表示为

（源IP地址，源端口；目标IP地址，目标端口）

  - IP地址用来标识互联网中的两台通信的特定主机，端口号用来标识特定主机上通信的进程。(网络程序设计最基本知识)

用户数据报协议 - UDP
User datagram protocol

- 网络层之上常用的简单协议，只能保证进程之间通信的最基本要求，

- 没有提供数据传输过程中的可靠性保证措施，无连接，不可靠

- UDP从进程产生的输出，对每次输出都生成一个UDP数据报直接封装在IP数据报中进行传输。因此传输层使用UDP协议时，发送端不需要发送缓冲区

- UDP数据报到达接收端主机的IP层后，由目标主机的UDP层根据目标端口号送到相应进程

UDP报文结构

UDP源 端 口 号（16位）	**UDP目 标 端 口 号（16位）**
UDP长 度（16位）	**UDP 校 验 和（16位）**
数 据 区	

- UDP报文结构：校验码与伪首部
  - UDP头部校验和是一个16位二进制数表示的错误检查字段，
  - 为了提高UDP协议的工作效率，该字段可直接填入0
  - 计算UDP校验和的时候除了包含UDP首部和UDP数据区外，还包含了一个12个字节长的伪首部（pseudo header）
  - 伪首部它并不是UDP的真正组成部分，只是为了差错检查时包含更多的信息（主要是IP信息）因为如图4-4所示的UDP数据报中并不包含与IP地址有关的信息，如果只以UDP数据报为依据计算校验和，就无法对目标地址的正确性进行检查。

源 端IP地 址（32位）		
目 标 端IP地 址（32位）		
填充域（8位，全0）	协议(8位,UDP值为17)	UDP 长 度（16位）

- 伪首部：如果只以UDP数据报为依据计算校验和，就无法对目标地址的正确性进行检查

<table>
<tr><td colspan="3">源 端IP地 址（32位）</td></tr>
<tr><td colspan="3">目 标 端IP地 址（32位）</td></tr>
<tr><td>填充域（8位，全0）</td><td>协议(8位,UDP值为17)</td><td>UDP 长 度（16位）</td></tr>
</table>

- 伪首部包含IP首部的一些字段（地址）
- 填充域全填0，目的是使伪首部为16位二进制数的整数倍
- UDP长度为UDP数据报总长（不包括伪首部）

总结：UDP特点

- 无连接、不可靠的数据报传输协议。不保持端对端连接，仅发送/接收据报
- UDP传输过程中惟一的可靠保证措施是进行差错校验
- 接收时若UDP数据报中的目标端口号不匹配，则抛弃
- UDP设计简单，保证了高效性和低延时性。在服务质量较高的网络中（如局域网），UDP可以高效地工作
- 常用于传输延时小，对可靠性要求不高，传输数据少的情况，如DNS、TFTP等

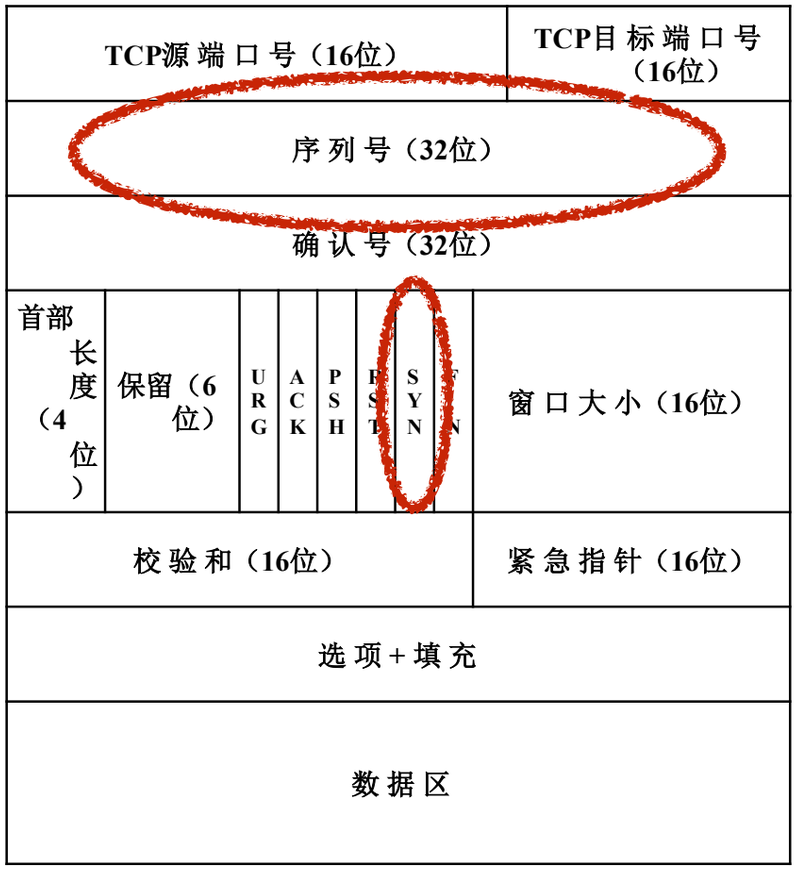
传输控制协议 - TCP
Transmission Control Protocol

- TCP提供一种面向连接的、可靠的数据流服务
- TCP协议成为传输层最常用的比较复杂的协议
- TCP报文段与UDP数据报一样是封装在IP中传输的，只是数据区为TCP报文段

TCP报文段结构

<table>
<tr><td colspan="9">TCP源端口号（16位）</td><td colspan="7">TCP目标端口号（16位）</td></tr>
<tr><td colspan="16">序列号（32位）</td></tr>
<tr><td colspan="16">确认号（32位）</td></tr>
<tr><td>首部长度（4位）</td><td colspan="3">保留（6位）</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="7">窗口大小（16位）</td></tr>
<tr><td colspan="12">校验和（16位）</td><td colspan="4">紧急指针（16位）</td></tr>
<tr><td colspan="16">选项+填充</td></tr>
<tr><td colspan="16">数据区</td></tr>
</table>

- TCP源端口号：
  - 长度为16位，标识发送方通信进程的端口。目标端在收到TCP报文段后，可以用源端口号和源IP地址标识报文的返回地址
- TCP目标端口号
  - 长度为16位，标识接收方通信进程的端口。
  - 源端口号和IP首部中的源端IP地址，目标端口号和目标端IP地址，惟一确定从源端到目标端的一对TCP连接



- 序列号SEQ
  - 序列号长度为32位，用于标识TCP发送端向TCP接收端发送数据字节流的序号
  - 序号的作用：保证顺序

- 如果SYN标记为1：SEQ序列号为初始的序列号，ACK=SEQ+1，后续实际发送数据时，第一个TCP数据段的SEQ被设置为初始序列号+1

- 如果SYN标记为0：则当前的数据段是正在传送过程中的TCP数据的一部分，SEQ表示当前正在传输的序号?

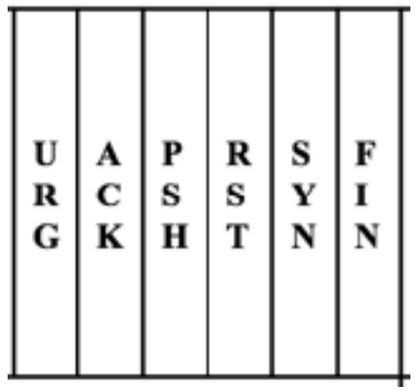
- 相对序号/绝对序号?

- 序号容量/序号用完怎么办

<table>
<tr><td colspan="6">TCP源端口号（16位）</td><td colspan="4">TCP目标端口号（16位）</td></tr>
<tr><td colspan="10">序列号（32位）</td></tr>
<tr><td colspan="10">确认号（32位）</td></tr>
<tr><td>首部长度（4位）</td><td>保留（6位）</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="2">窗口大小（16位）</td></tr>
<tr><td colspan="6">校验和（16位）</td><td colspan="4">紧急指针（16位）</td></tr>
<tr><td colspan="10">选项+填充</td></tr>
<tr><td colspan="10">数据区</td></tr>
</table>

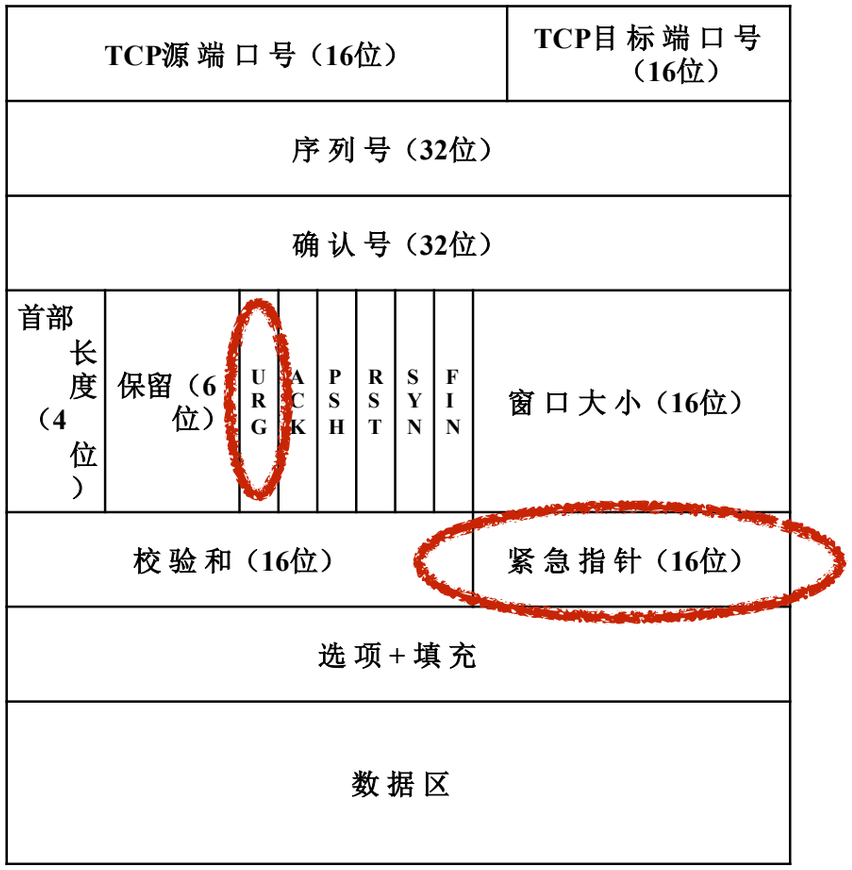
- 确认号ACK
  - 长度为32位，用于确认已经收到的数据的序号
  - 需要ACK标记（Flag）位置1才有效

- ACK确认号的用途
  - 确认了已经收到的数据的字节序
  - 期待的下一个将收到的字节序
  - （特殊情况，如前述，当收到第一个SYN时，只SEQ+1）



- 标志位
  - URG：紧急指针（urgent pointer）标志，置1时，紧急指针有效（下面介绍）。
  - ACK：确认号标志
  - PSH：Push操作标志，当置1时，表示不等缓冲区饱和，直接对数据进行Push操作，数据将立即被传输或者接受并发送给相应端口的应用程序。

- 标志位（续）
  - RST：连接复位标志，表示由于主机崩溃或其它原因而出现错误时的连接。可以用它来表示非法的数据段或拒绝连接请求
  - SYN：同步序列号标志，发起一个连接的建立（握手），只有在连接建立的过程中SYN才被置1
  - FIN：连接终止标志，当一端发送FIN标志置1的报文时，告诉另一端已完成了数据发送任务



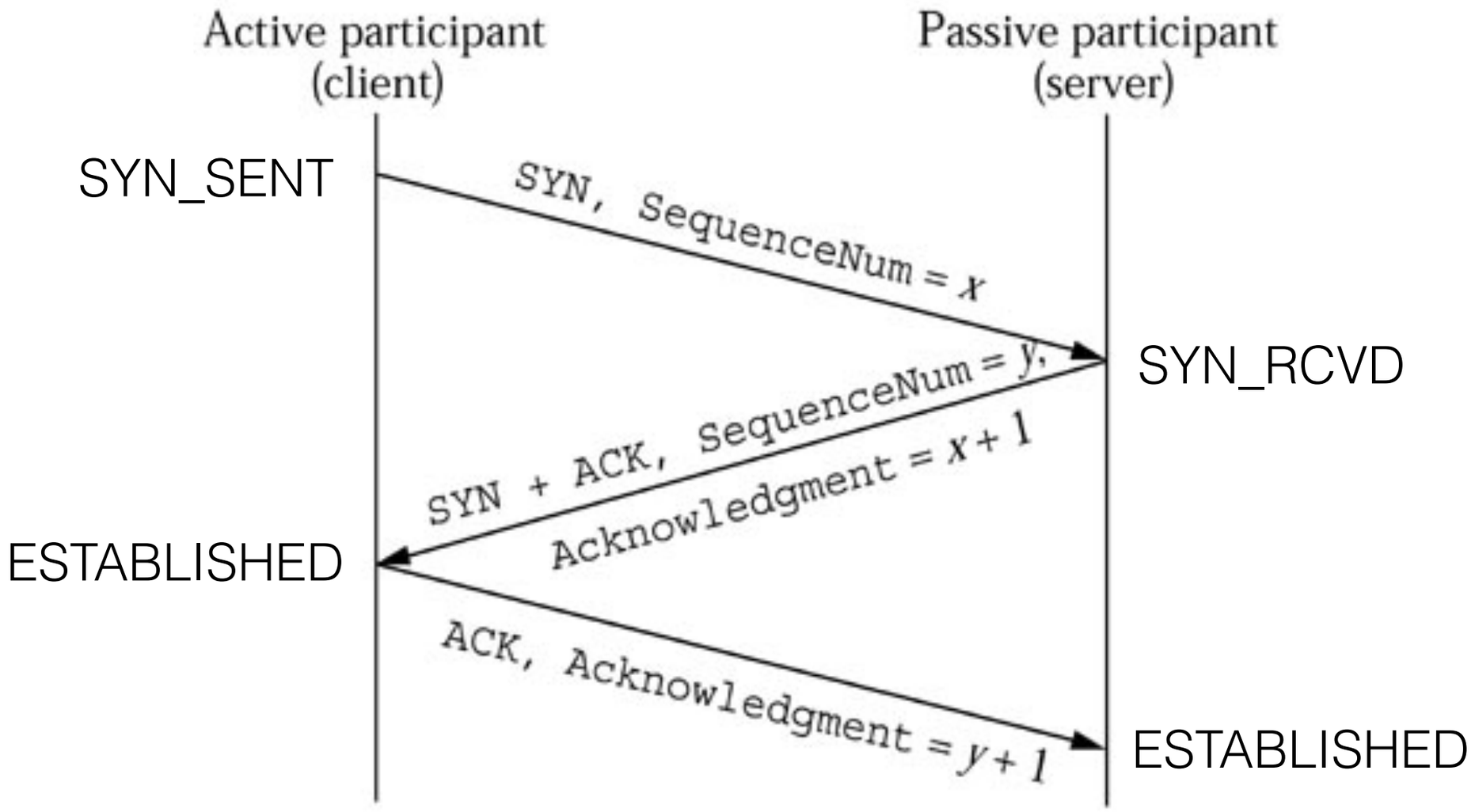
- 紧急指针字段，配合URG Flag使用
- 长度为16位，它的值指向紧急数据最后一个字节的位置
- TCP协议层提供了这种紧急模式(urgent mode)，但TCP并不关心

TCP连接的建立：三次握手

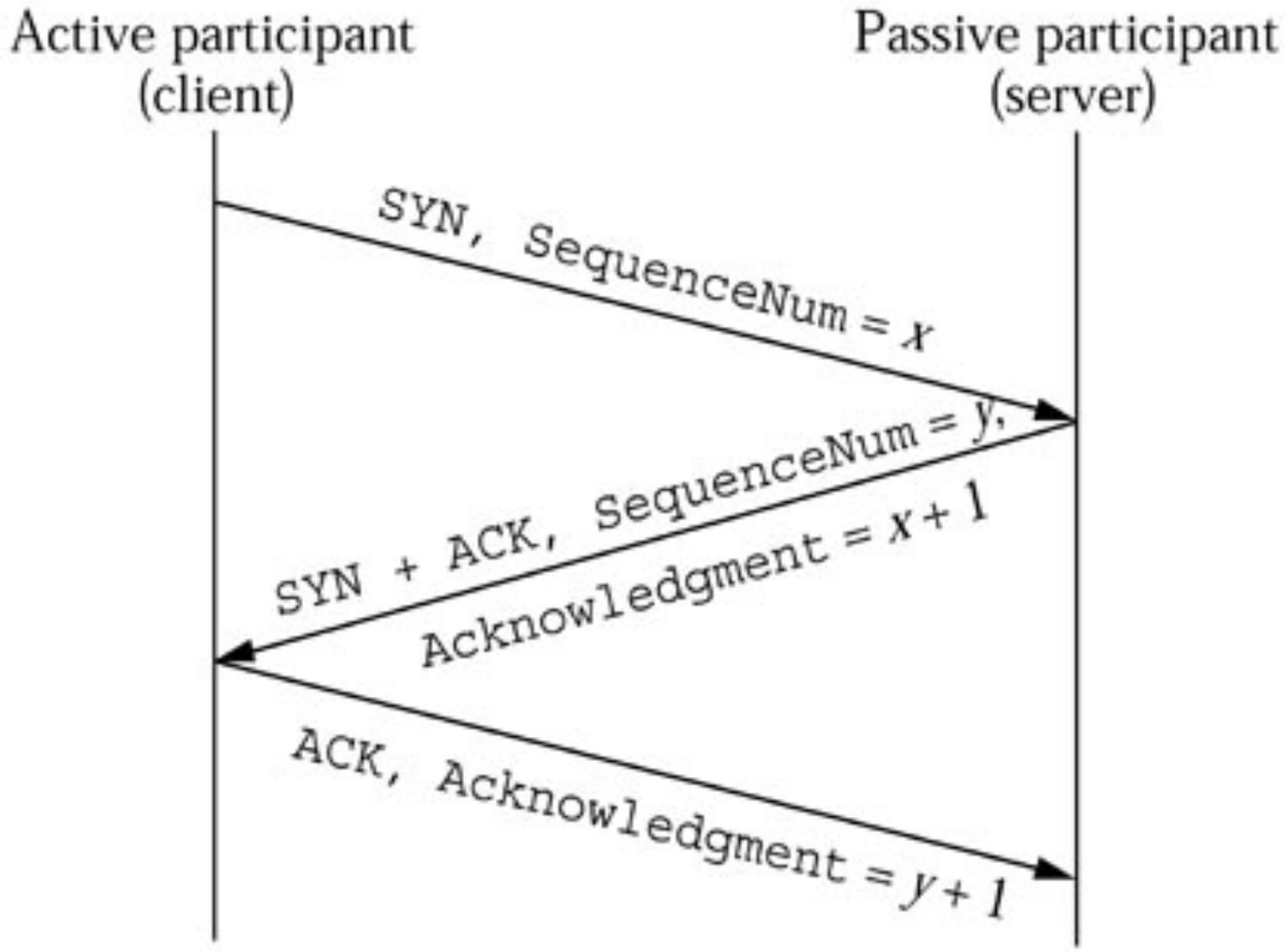
- 建立TCP连接时，一台主机主动提出通信的请求（客户机），另一台被动的响应请求（服务器）

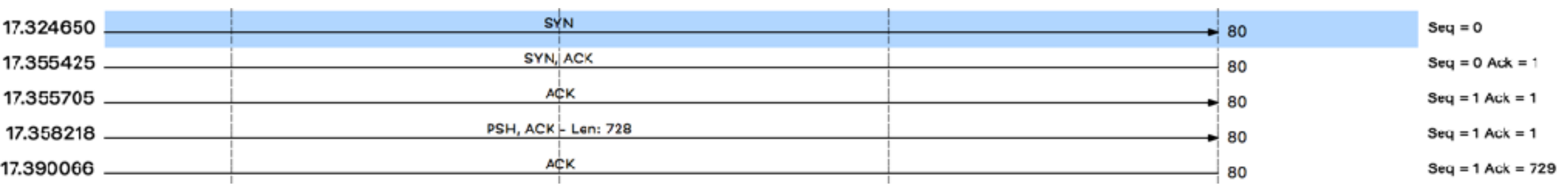
- TCP使用“3次握手”（3-way handshake）法来建立一条连接

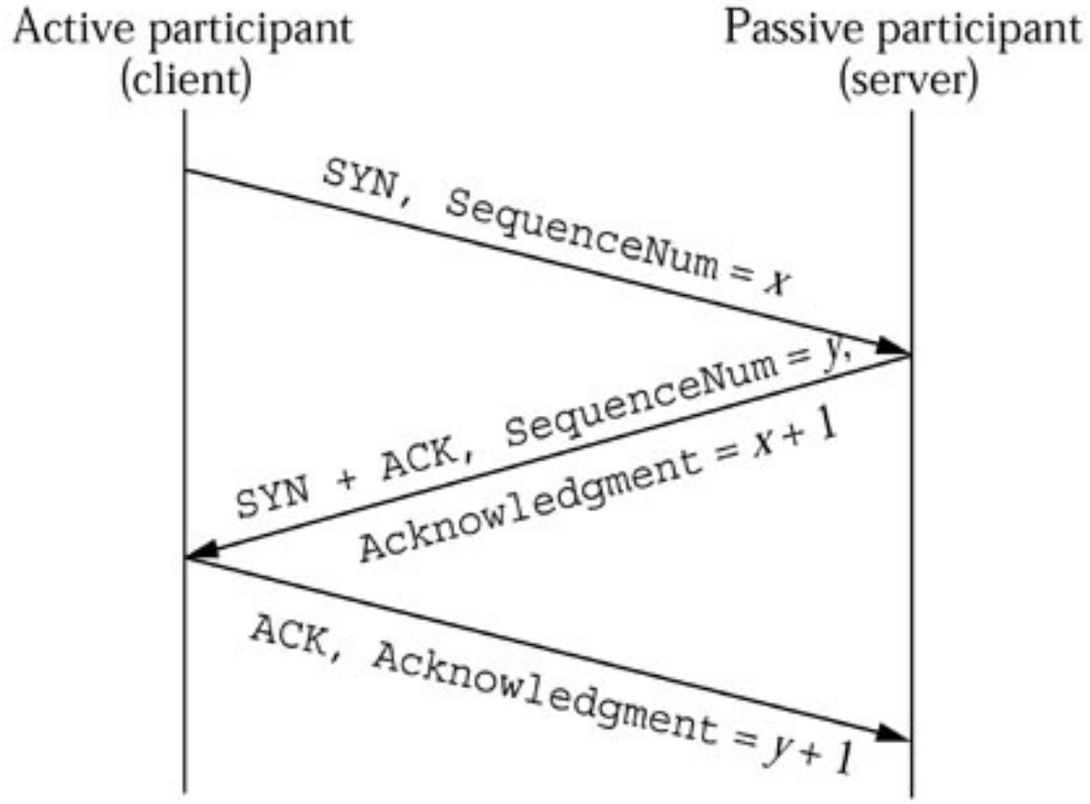
- 所谓3次握手就是指在建立一条连接时通信双方要交换3次报文


Active participant
(client)
Passive participant
(server)
SYN_SENT
SYN, SequenceNum = x
SYN_RCVD
SYN + ACK, SequenceNum = y,
Acknowledgment = x + 1
ESTABLISHED
ACK, Acknowledgment = y + 1
ESTABLISHED

- 三次握手


Active participant
(client)
Passive participant
(server)
SYN, SequenceNum = x
SYN + ACK, SequenceNum = y,
Acknowledgment = x + 1
ACK, Acknowledgment = y + 1


17.324650
SYN
80
Seq = 0
17.355425
SYN, ACK
80
Seq = 0 Ack = 1
17.355705
ACK
80
Seq = 1 Ack = 1
17.358218
PSH, ACK - Len: 728
80
Seq = 1 Ack = 1
17.390066
ACK
80
Seq = 1 Ack = 729



17.730212	SYN		80	Seq = 0
17.730212	SYN		80	Seq = 0
17.730322	SYN		80	Seq = 0
17.730373	SYN		80	Seq = 0
17.730454	SYN		80	Seq = 0
17.730562	SYN		80	Seq = 0
17.753092	SYN, ACK		80	Seq = 0 Ack = 1
17.753261	ACK		80	Seq = 1 Ack = 1
17.754261	SYN, ACK		80	Seq = 0 Ack = 1
17.754452	SYN, ACK		80	Seq = 0 Ack = 1
17.754855	SYN, ACK		80	Seq = 0 Ack = 1

为什么是三次握手?

理论上三次是能够在不可靠信道上双方达成一致的最小值

“你瞅啥？”

“咋？瞅你咋地？”

“再瞅一个试试？！”

——TCP三次握手的一个形象类比（来自知乎）

- 连接建立后通信的双方可以相互传输数据，并且双方的地位是平等的。

- 如果在建立连接的过程中握手报文段丢失，则可以通过重发机制进行解决

- 客户端按某种机制重发建立连接的请求报文段若干次后，就通知应用进程，连接不能建立（超时）；服务器端没有相应的端口时，服务器端以复位报文应答（RST），连接不能建立

- 建立连接的TCP报文段中（SYN），只有报文头没有数据区

TCP连接的关闭：四次挥手

- 第1次挥手（或者说握手）：客户向服务器发送一个FIN（即FIN=1的TCP数据段）

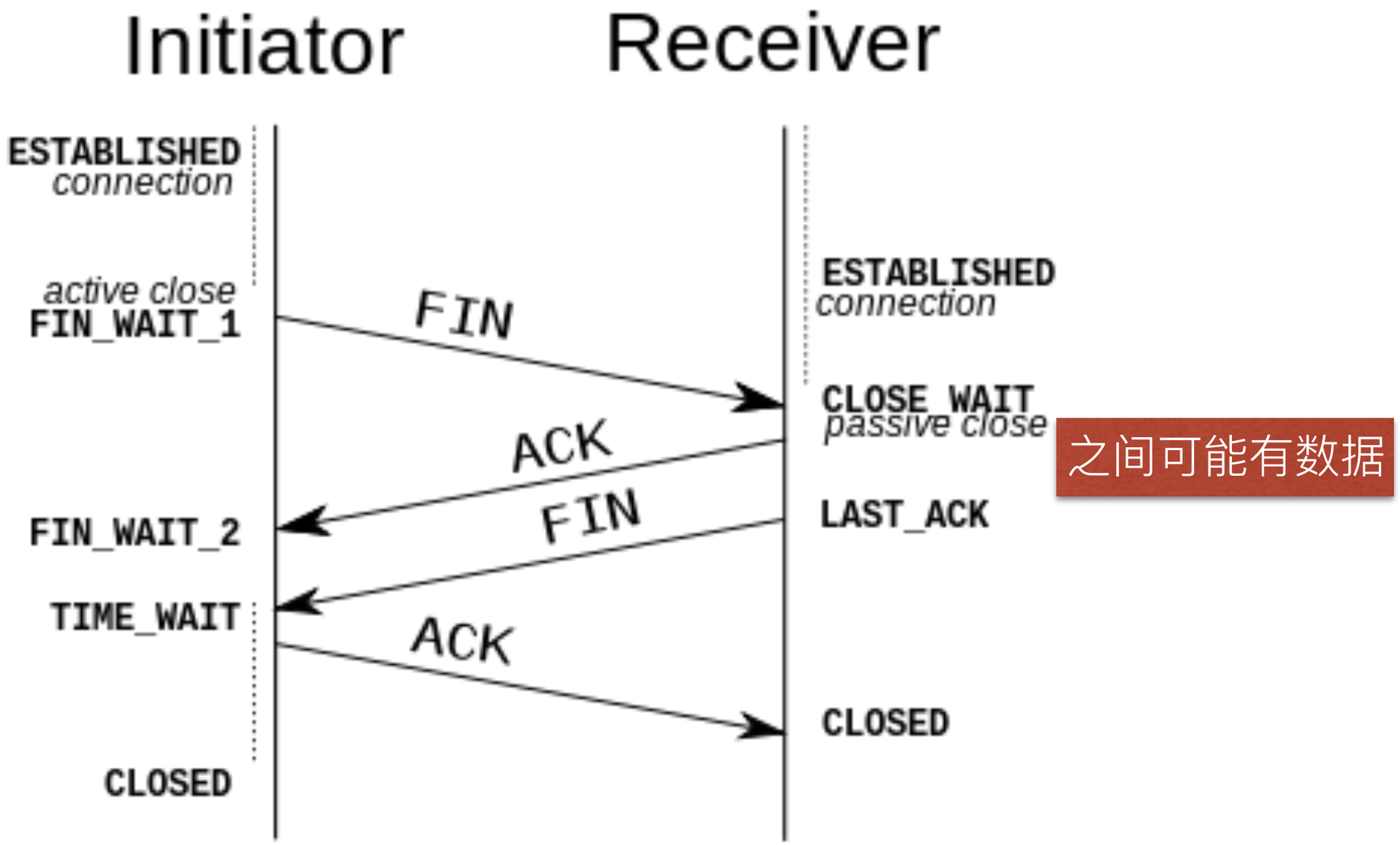
- 第2次挥手：服务器（的TCP协议层）收到FIN，就发出ACK确认，确认号为已收到的最后一个字节的序列号加1

- 在发送完ACK后，服务器可以继续向客户机发送数据，这种状态叫半关闭（half-close）状态，只是客户方已无数据发向服务器了（为何如此设置？）

- 第3次挥手：服务器数据发送完后，向客户机发送一个FIN，要求关闭连接。

- 第4次挥手：客户机收到关闭连接的FIN报文段后，向服务器发送一个ACK确认，确认号为已收到数据的序列号加1

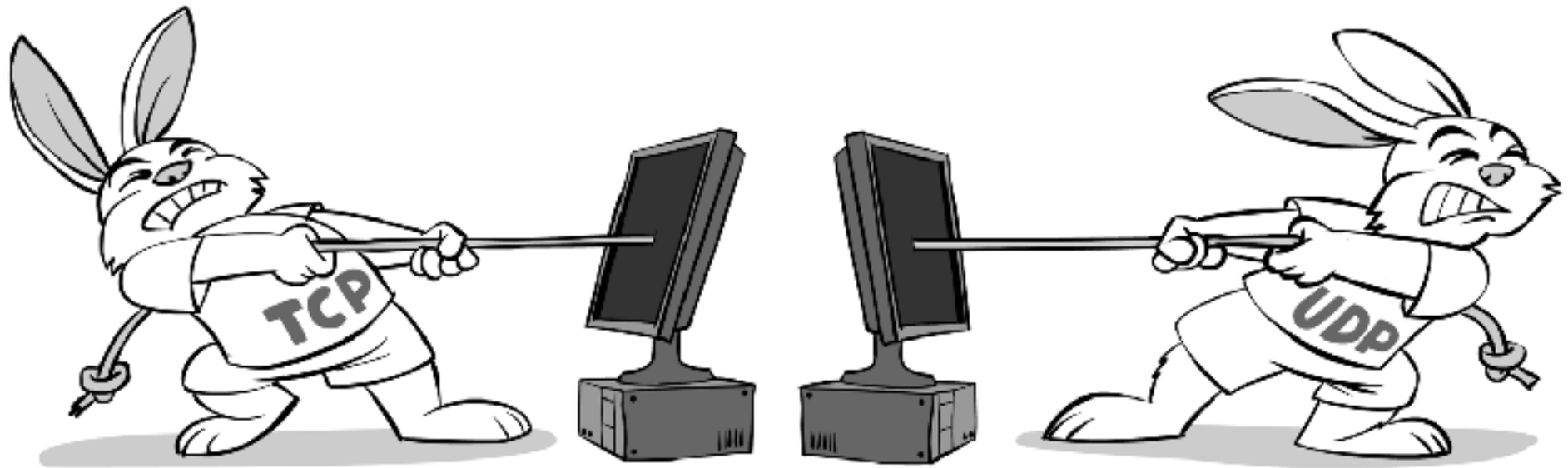
- 当服务器收到确认后，整个连接被完全关闭。


Initiator
Receiver
ESTABLISHED
connection
active close
FIN_WAIT_1
FIN
ESTABLISHED
connection
CLOSE_WAIT
passive close
ACK
FIN_WAIT_2
FIN
LAST_ACK
TIME_WAIT
ACK
CLOSED
CLOSED
之间可能有数据

总结：TCP与UDP的比较

比较项目	TCP	UDP
建立的连接与关闭	有	无
数据传输效率（当网络可靠时）	低	高
对数据的确认	有	无
流量控制	有（滑动窗口）	无
丢失分组的重发	有	无（由高层应用程序负责）
协议复杂性	复杂	简单
发送端缓冲	有	无
分组排序	有	无
对重复分组的检测	有	无
校验和	有	有（且算法相同）
在低层被分片的情况	可能性小（因为在连接建立时，双方通知各自的MSS，每个TCP报文段的长度不超过MSS）	可能性大（因为应用程序每次输出都产生一个UDP报文，当一次有大量数据要输出时，常在低层被分片）
广播与多播	不支持（因它要建立一对一的连接）	支持
适用场合	可靠性要求高，有大量数据要连续传输，该协议在互联网应用多	对可靠性要求一般，但要高效传输数据，或数据传输量小的应用场合

总结：重提TCP和UDP之争



- UDP的优势
    - Akamai报告从2008年到2015年，各国网络平均速率由1.5Mbps提升为5.1Mbps。网络环境变好，延迟、稳定性改善，UDP的丢包率低于5%，配合应用层重传，UDP可确保传输的可靠性。[知乎]
    - UDP不用握手，不提供流控、拥塞等功能，传输不可靠，因此有时更实时简单有效。（奥卡姆剃刀？）
    - UDP打洞，绕过NAT
    - UDP协议简单，冗余功能少，提升空间大
    - 网络连接质量特别差的时候，TCP反而效果不好?

- TCP的优势
  - 自带可靠连接属性
  - 适合不知道该用UDP还是用TCP的场景

TCP/IP网络通信原理总结

- 在因特网中，用一个三元组可以在全局中唯一地标识一个应用层进程：

  应用层进程地址 =（传输层协议，主机的IP地址，传输层的端口号）

- 这个三元组叫做一个半相关（half-association），它标识了因特网中，进程间通信的一个端点，也把它称为进程的网络地址

- 一个完整的网间通信需要一个五元组在全局中唯一地来标识：

  (传输层协议，本地机IP地址，本地机传输层端口，远地机IP地址，远地机传输层端口)

- 这个五元组称为一个全相关（association），即两个协议相同的半相关才能组合成一个合适的全相关，或完全指定一对网间通信的进程

完