

第08章 传输层



讲师：韩立刚
51CTO学院金牌讲师
51CTO学院教学顾问
微软最有价值专家MVP
河北师大软件学院讲师
河北地质大学客座教授
计算机图书作者

本章重点讲传输层两个

TCP/IPv4协议组

HTTP	FTP	SMTP	POP3	Telnet	DHCP	DNS	TFTP
TCP				UDP			
IPv4						ICMP	IGMP
ARP							
CSMA/CD		X.25	HDLC	Frame Relay		PPP	

本章内容

- 8.1 传输层的两个协议
- 8.2 用户数据报协议UDP
- 8.3 传输控制协议TCP
- 8.4 可靠传输
- 8.5 流量控制
- 8.6 拥塞控制
- 8.7 TCP连接管理

8.1传输层的两个协议

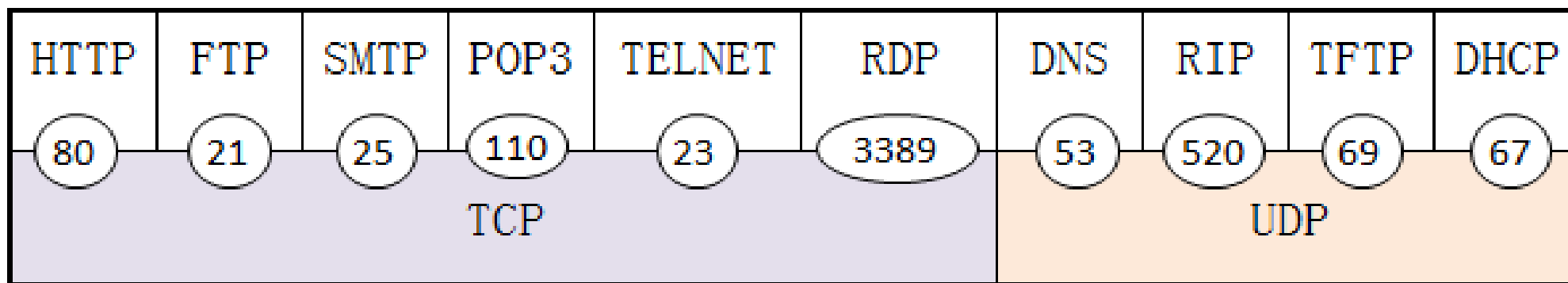
- 8.1.1 TCP和UDP协议的应用场景
- 8.1.2 传输层协议和应用层协议之间的关系
- 8.1.3 服务和端口之间的关系
- 8.1.4 实战：服务器端口冲突造成服务器启动失败
- 8.1.5 实战：更改服务使用的默认端口
- 8.1.6 端口和安全的关系
- 8.1.7 实战：Windows防火墙和TCP/IP筛选实现网络安全

8.1.1 TCP和UDP协议的应用场景

- 网络中的计算机通信无外乎有以下两种情况：
 - 1.要发送的内容多，需要将发送的内容分成多个数据包发送。
 - 2.要发送的内容少，一个数据包就能发送全部内容。
- 针对这两种情况，在传输层有两个协议，TCP（Transmission Control Protocol 即传输控制协议）和UDP（User Datagram Protocol即用户数据报协议）。

8.1.2传输层协议和应用层协议之间的关系1

- 应用层协议很多，传输层就两个协议，如何使用传输层两个协议标识应用层协议呢？
- 传输层协议加一个端口号来标识一个应用层协议，展示了传输层协议和应用层协议之间的关系。



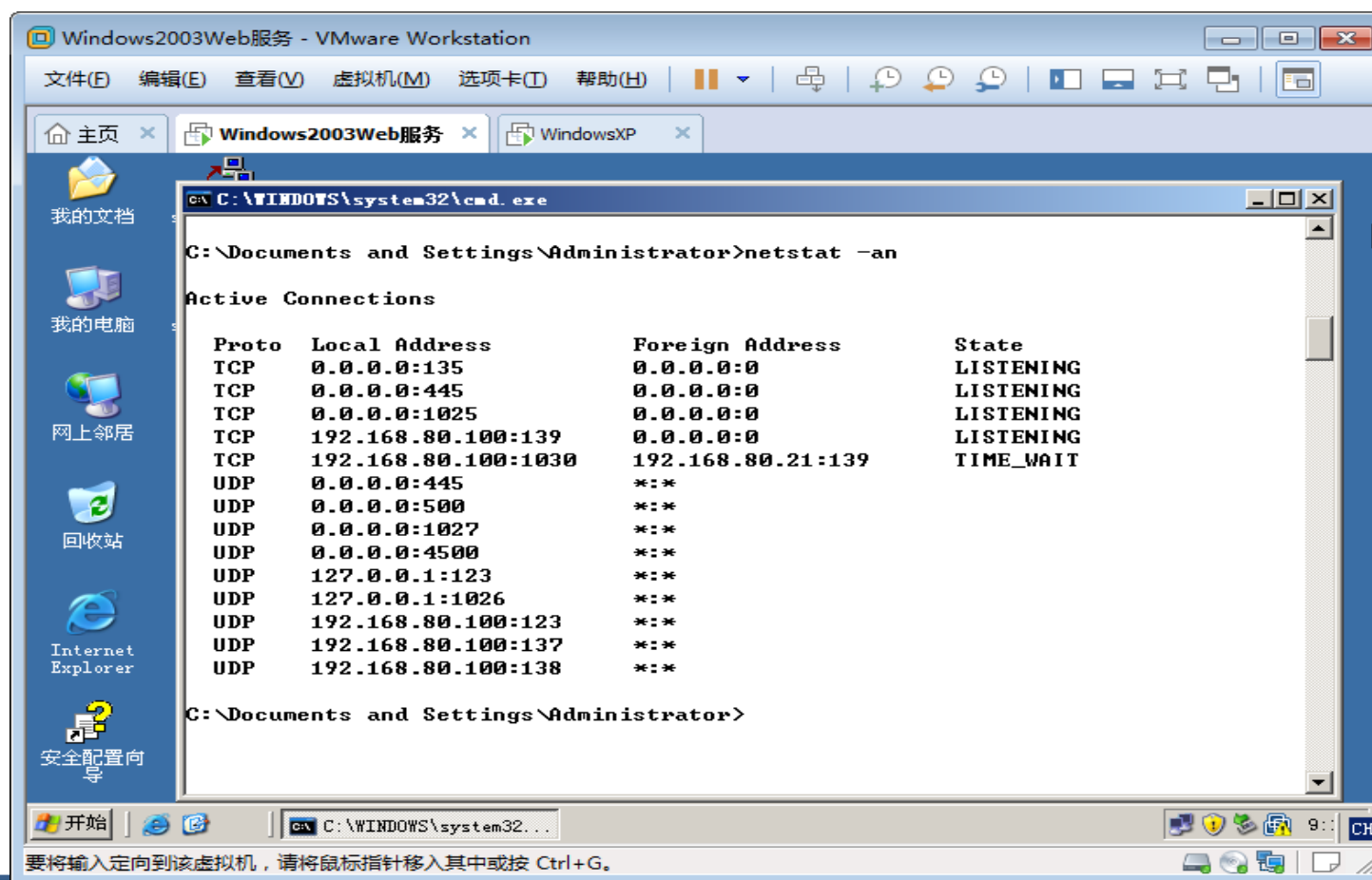
8.1.2传输层协议和应用层协议之间的关系2

■ 一些常见的应用层协议和传输层协议，以及它们之间的关系。

- HTTP默认使用TCP的80端口标识。
- FTP默认使用TCP的21端口标识。
- SMTP默认使用TCP的25端口标识。
- POP3默认使用TCP的110端口。
- HTTPS默认使用TCP的443端口。
- DNS使用UDP的53端口。
- 远程桌面协议（RDP）默认使用TCP的3389端口。
- Telnet使用TCP的23端口。
- Windows访问共享资源使用TCP的445端口。
- 微软SQL数据库默认使用TCP的1433端口。
- mySQL数据库默认使用TCP的3306端口。

8.1.3服务和端口之间的关系

- Windows和Linux操作系统有些服务为本地计算机提供服务，有些服务为网络中的计算机提供服务。
- 为网络中计算机提供服务的服，一旦启动就会使用TCP或UDP的某个端口侦听客户端的请求。



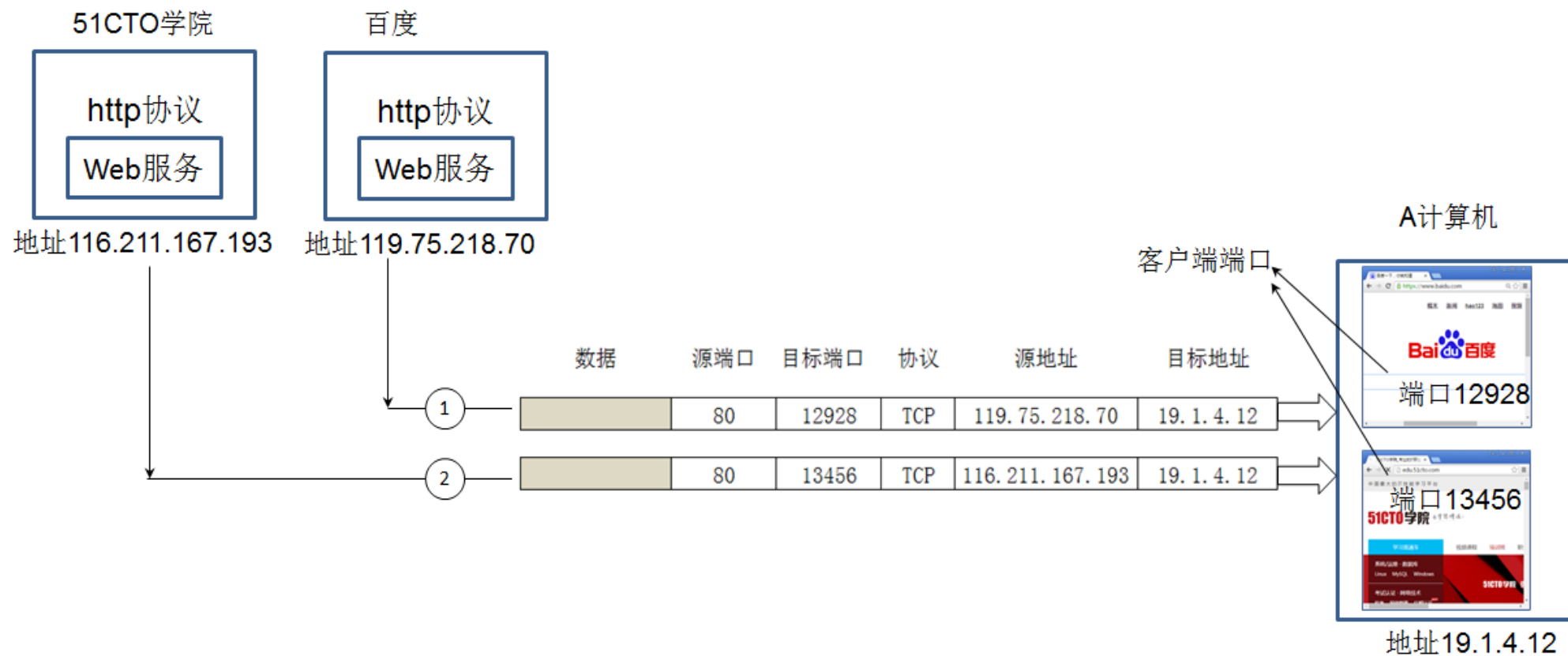
服务和端口的关系

- 客户端通过IP地址定位服务器，通过协议和端口号定位服务器上的服务。



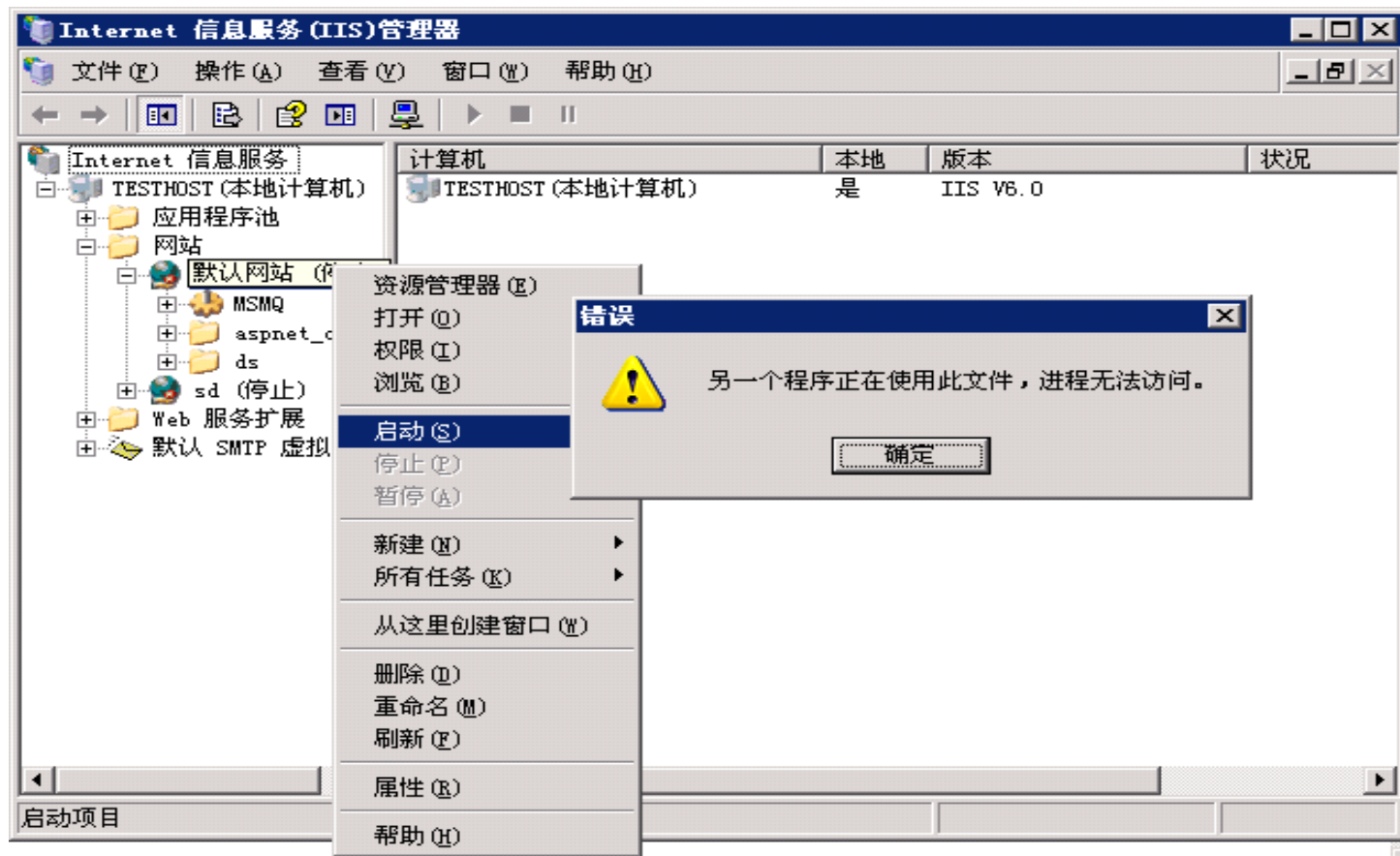
客户端端口的作用

- 客户端软件可以同时访问多个服务器，客户端会为每个出去的流量分配一个唯一的源端口。



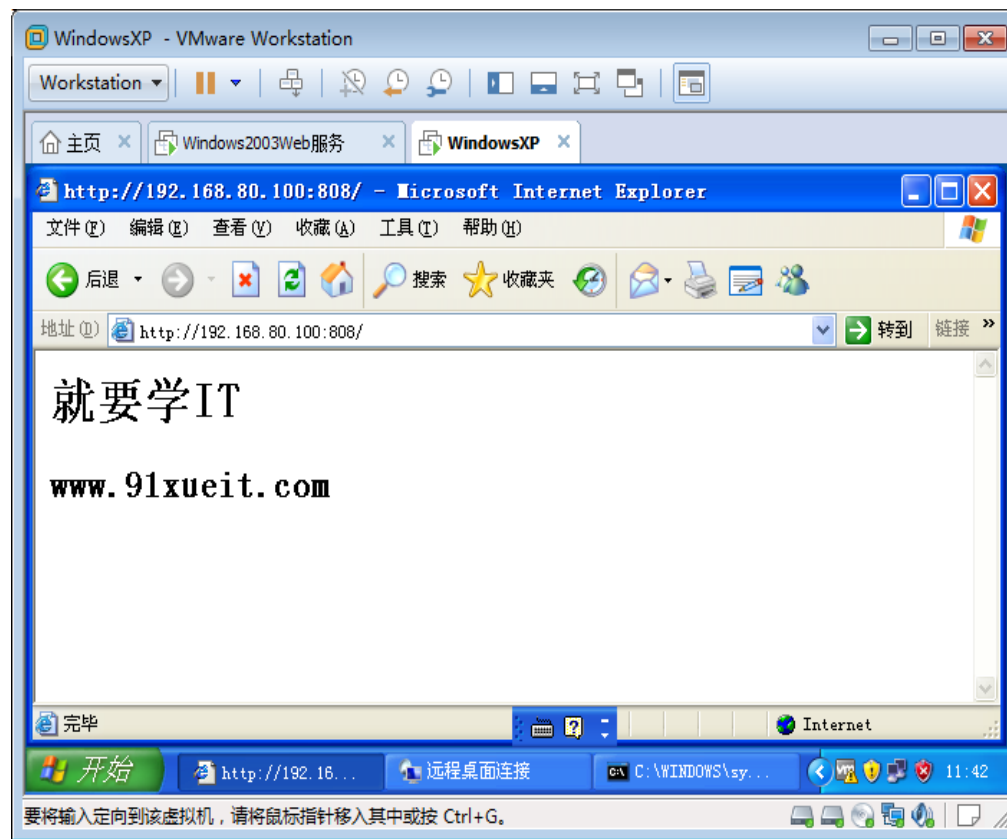
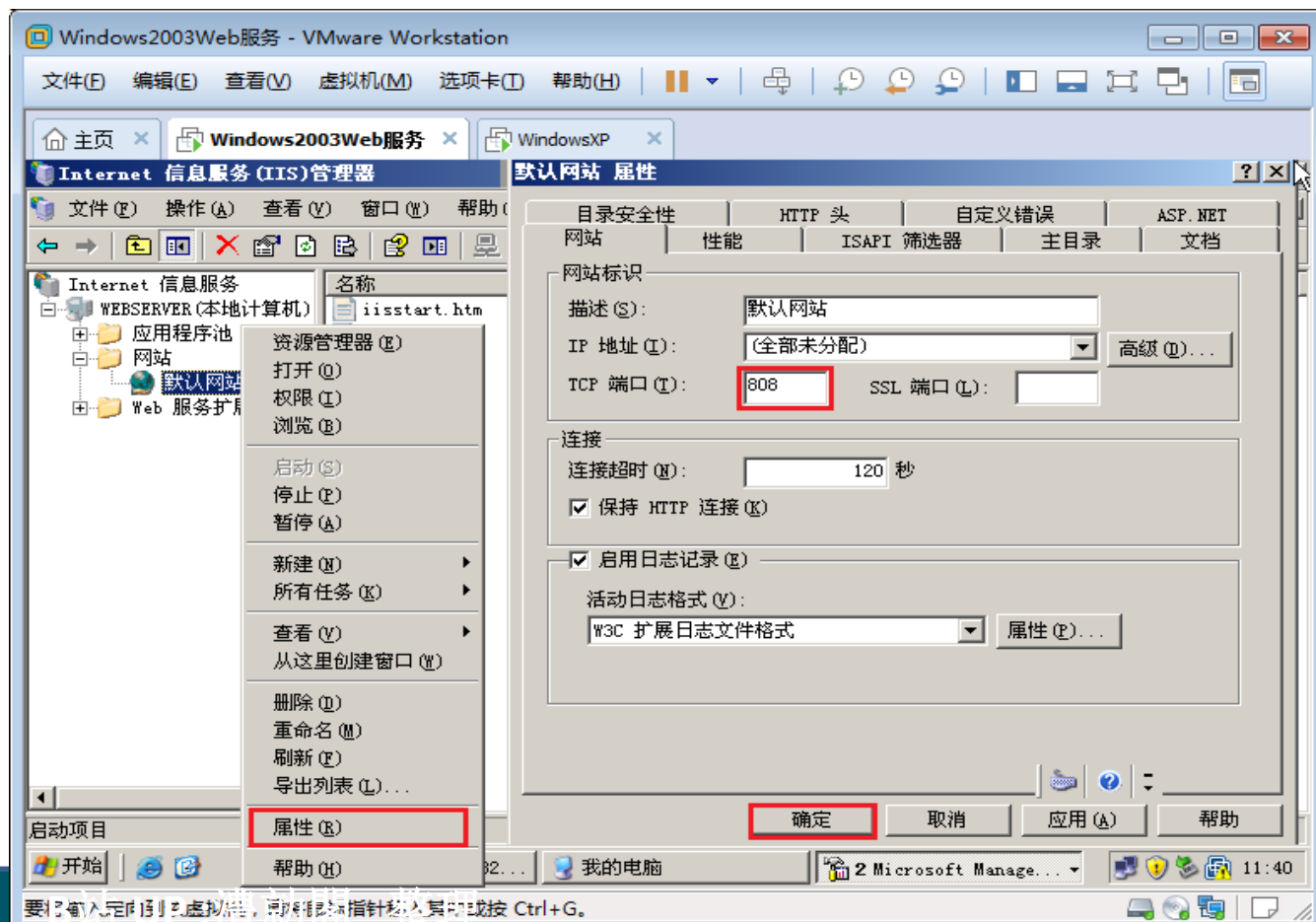
8.1.4实战：服务器端口冲突造成服务启动失败

- 服务器上的服务侦听的端口不能冲突，否则将会造成服务启动失败。



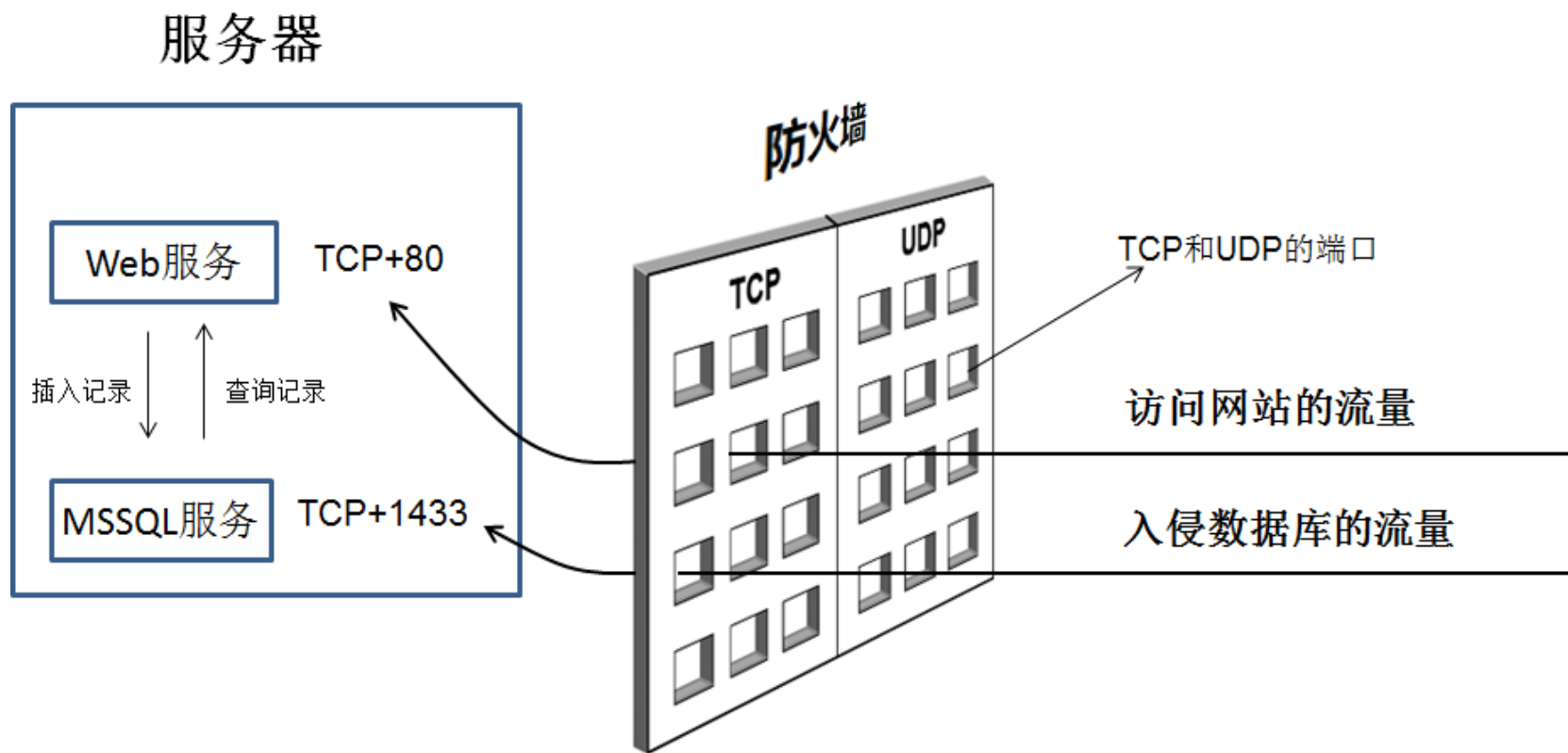
8.1.5实战：更改服务使用的默认端口

- 应用层协议也可以不使用默认端口和客户端通信。
- 如果不适用默认端口通信，客户度需要指明使用的端口。



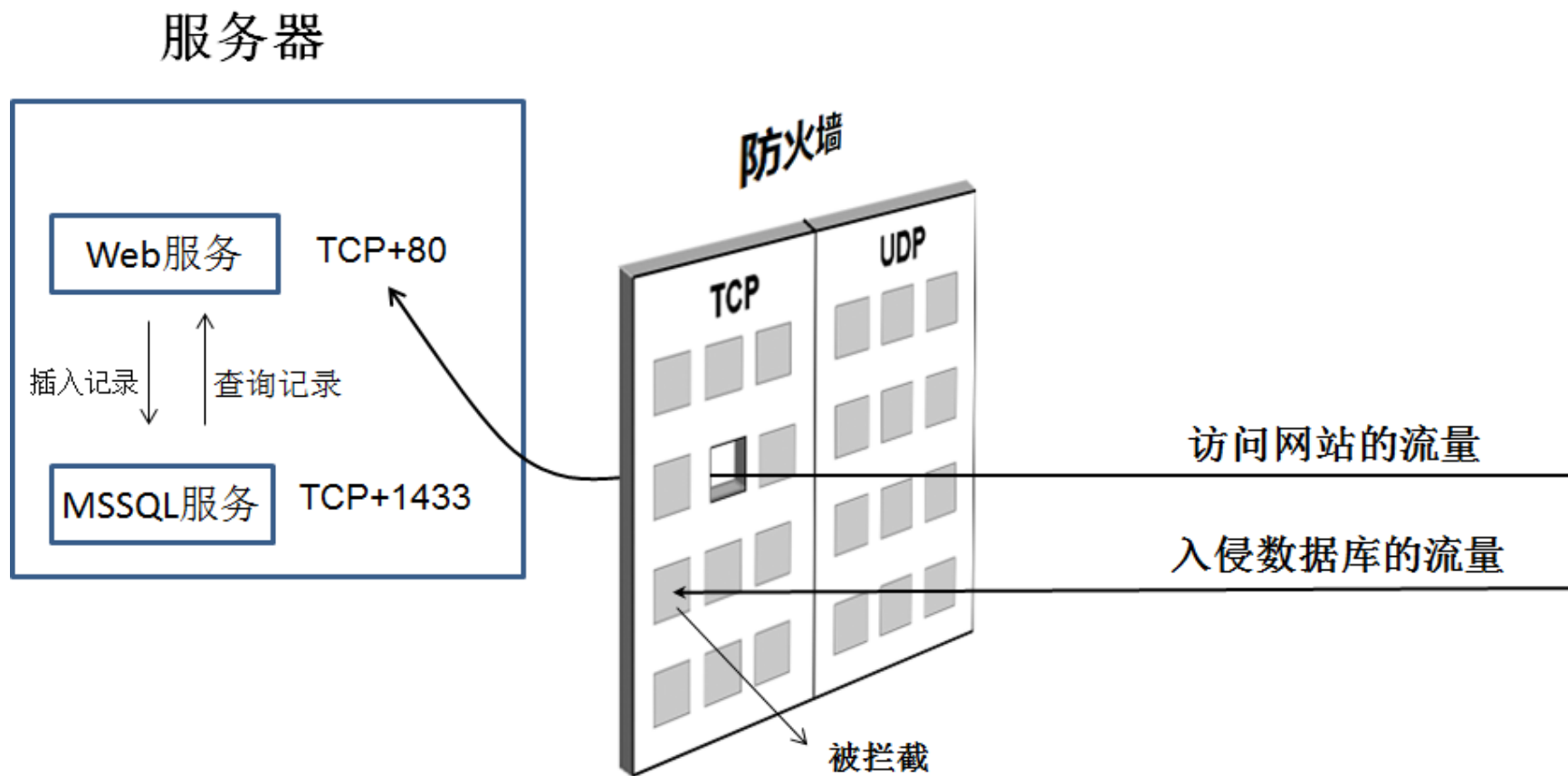
8.1.6端口和网络安全的关系1

- 客户端和服务端之间的通信使用应用层协议，应用层协议使用传输层协议+端口标识，如果在网络设备封掉TCP或UDP的某个端口，就不能访问其对应的服务，就可以实现网络安全。



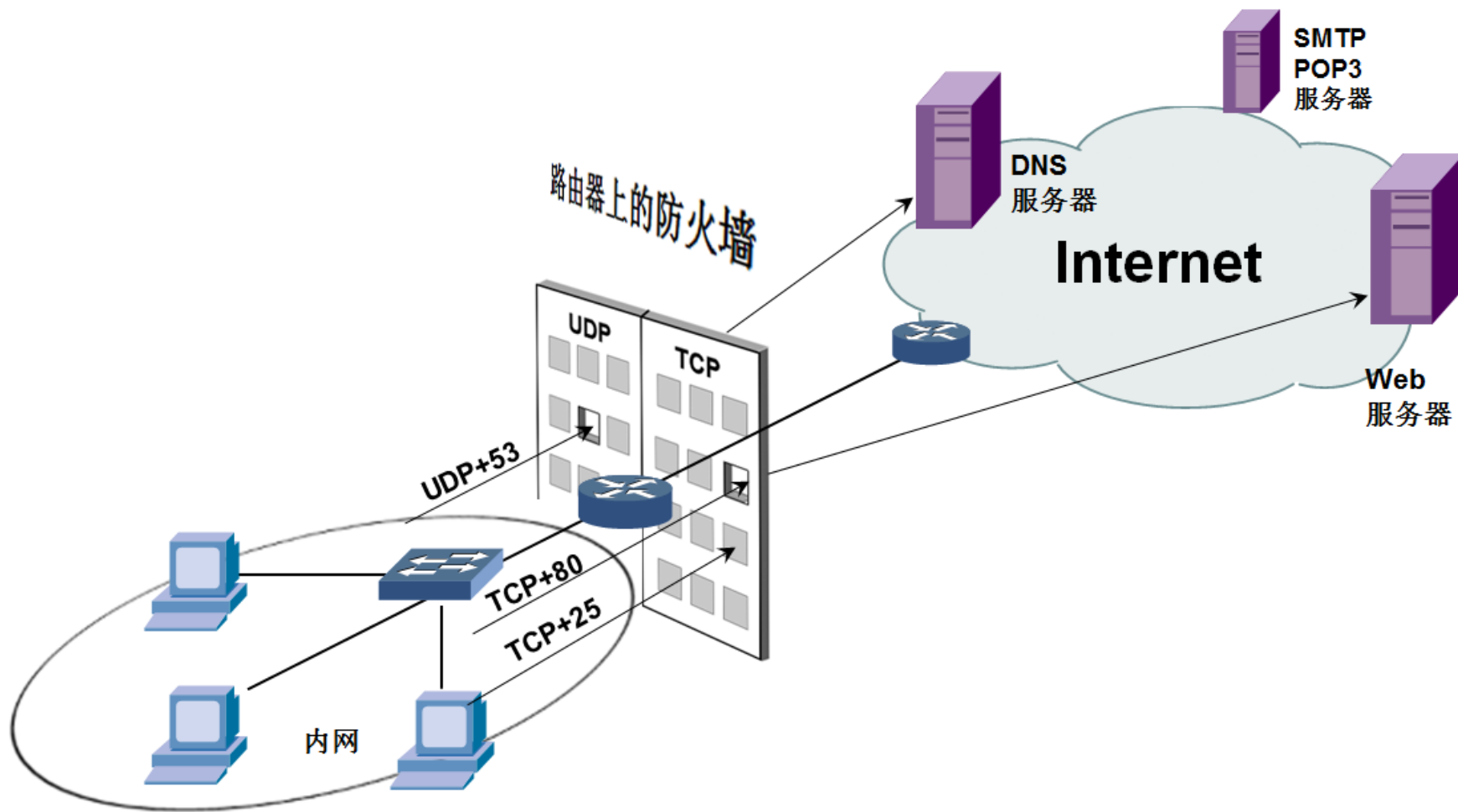
设置服务器网络安全

- 设置服务器网络安全，只开放必要的端口。
- 在Windows上可以通过设置TCP/IP筛选和Windows防火墙来实现。

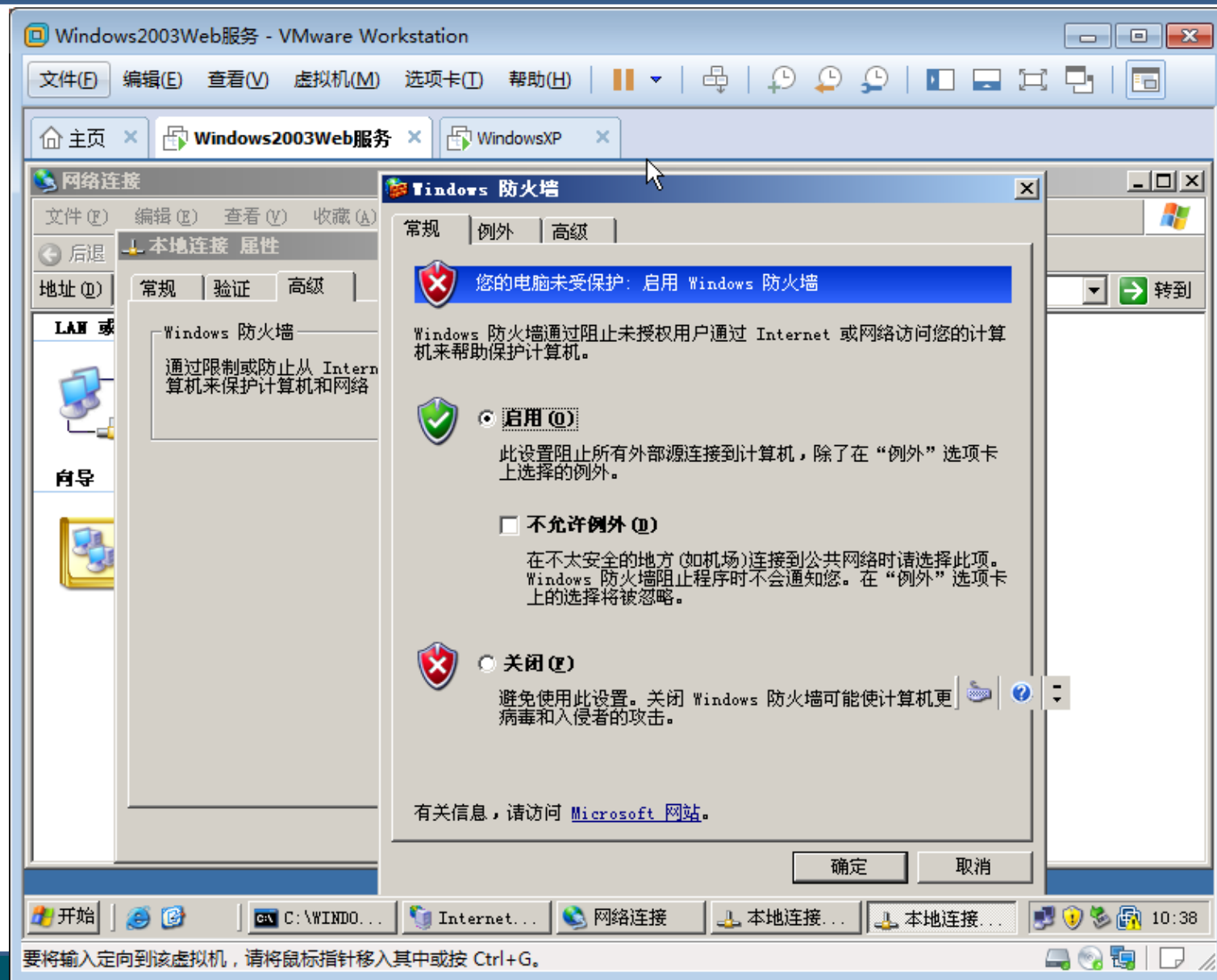


8.1.6端口和网络安全的关系2

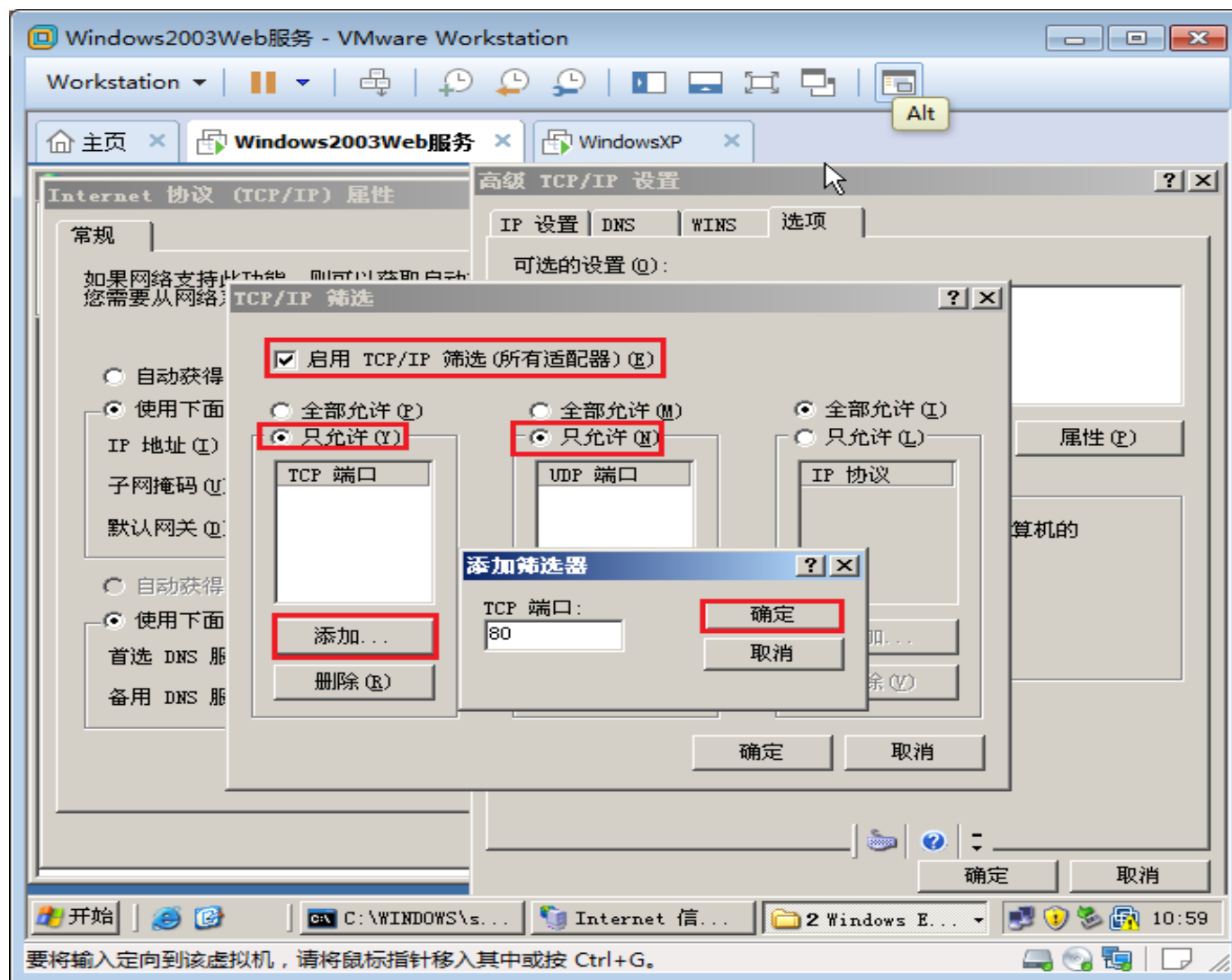
■在网络上设备上控制端口



8.1.7实战：Windows防火墙和TCP/IP筛选实现网络安全1



8.1.7 实战：Windows防火墙和TCP/IP筛选实现网络安全2



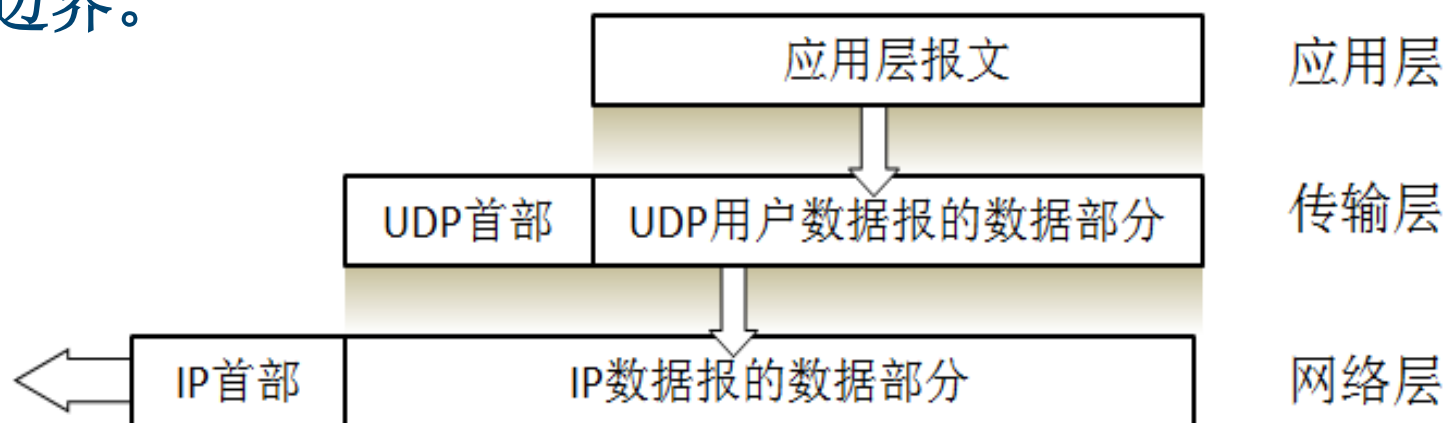
8.2用户数据报协议UDP

■8.2.1UDP协议的特点

■8.2.2UDP的首部格式

8.2.1 UDP协议的特点1

- (1) **UDP**是无连接的，即发送数据之前不需要建立连接（当然发送数据结束时也没有连接可释放），因此减少了开销和发送数据之前的时延。
- (2) **UDP**使用尽最大努力交付，即不保证可靠交付，因此主机不需要维持复杂的连接状态表（这里面有许多参数），通信的两端不用保持连接，因此节省系统资源。
- (3) **UDP**是面向报文的，发送方的**UDP**对应用程序交下来的报文，在添加首部后就向下交付给网络层。**UDP**对应用层交下来的报文，既不合并，也不拆分，而是保留这些报文的边界。



8.2.1 UDP协议的特点2

- (4) **UDP**没有拥塞控制，因此网络出现的拥塞不会使源主机的发送速率降低。这对某些实时应用是很重要的。
- (5) **UDP**支持一对一、一对多、多对一和多对多的交互通信。
- (6) **UDP**的首部开销小，只有8个字节，比TCP的20个字节的首部要短。

8.2.2 UDP 的首部 格式1

源端口

目标端口

长度

校验和

Wireshark 1.12.4 (v1.12.4-0-gb4861da from master-1.12)

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Length	Info
1	0.00000000	192.168.80.100	114.114.114.114	DNS	75	Standard query 0xee
2	0.03560600	114.114.114.114	192.168.80.100	DNS	91	Standard query response
3	0.03599700	192.168.80.100	59.46.80.160	TCP	62	1061→80 [SYN] Seq=0
4	0.05697300	59.46.80.160	192.168.80.100	TCP	60	80→1061 [SYN, ACK]
5	0.05700500	192.168.80.100	59.46.80.160	TCP	54	1061→80 [ACK] Seq=1
6	0.05733700	192.168.80.100	59.46.80.160	HTTP	362	GET / HTTP/1.1

Frame 1: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface 0

Ethernet II, Src: Vmware_14:bf:02 (00:0c:29:14:bf:02), Dst: Vmware_f4:11:93 (00:50:56:f4:11:93)

Internet Protocol Version 4, Src: 192.168.80.100 (192.168.80.100), Dst: 114.114.114.114 (114.114.114.114)

User Datagram Protocol, Src Port: 1040 (1040), Dst Port: 53 (53)

Source Port: 1040 (1040)

Destination Port: 53 (53)

Length: 41

Checksum: 0x0ec8 [validation disabled]

[Good Checksum: False]

[Bad Checksum: False]

[Stream index: 0]

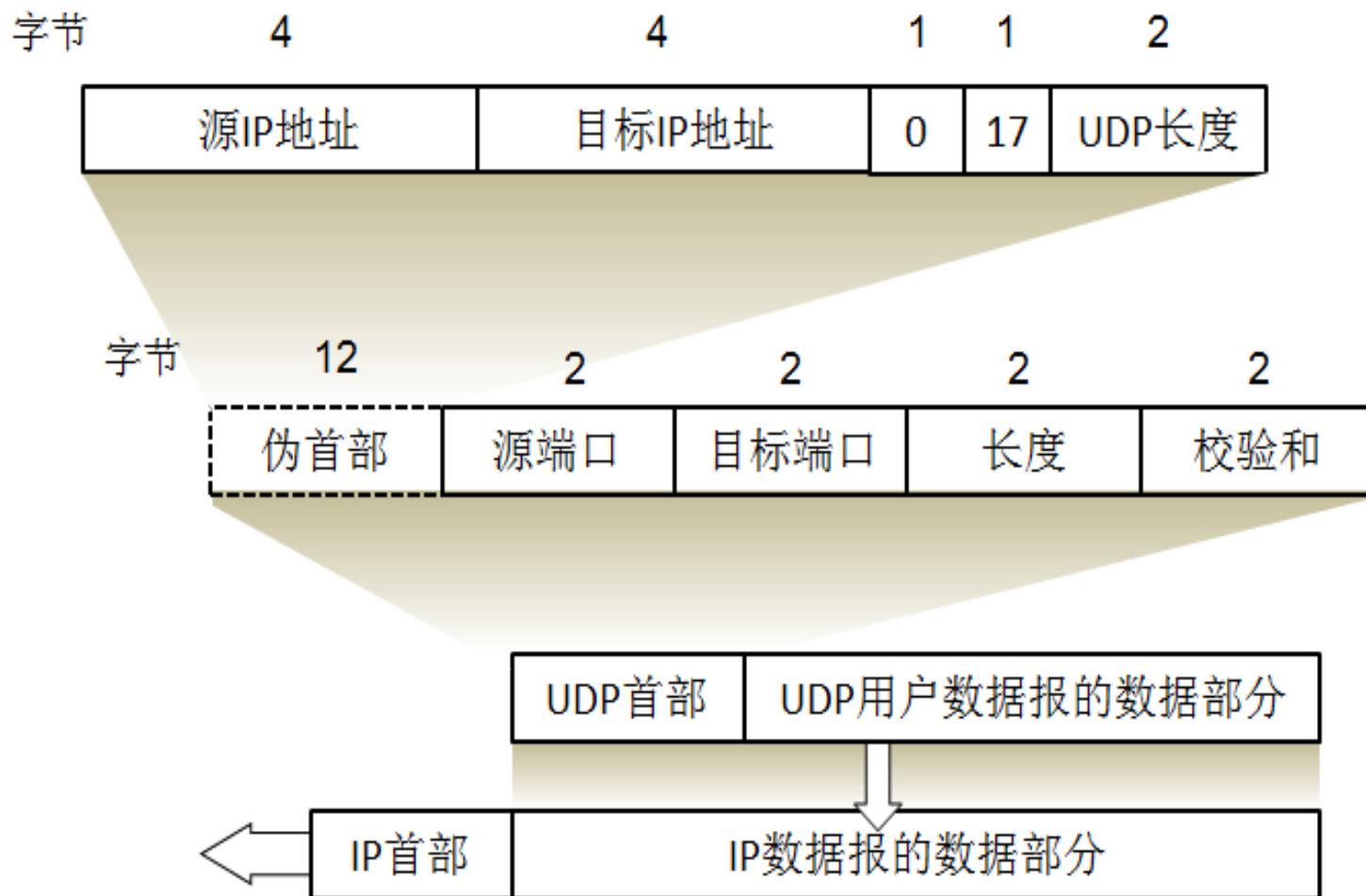
Domain Name System (query)

0000 00 50 56 f4 11 93 00 0c 29 14 bf 02 08 00 45 00 .PV.....).....E.
0010 00 3d 08 2b 00 00 80 11 3c 94 c0 a8 50 64 72 72 .=.+.<...Pdrr
0020 72 72 04 10 00 35 00 29 0e c8 ee 52 01 00 00 01 rr...5.)...R....
0030 00 00 00 00 00 00 03 77 77 77 07 39 31 78 75 65w ww.91xue
0040 69 74 03 63 6f 6d 00 00 01 00 01 it.com.. ...

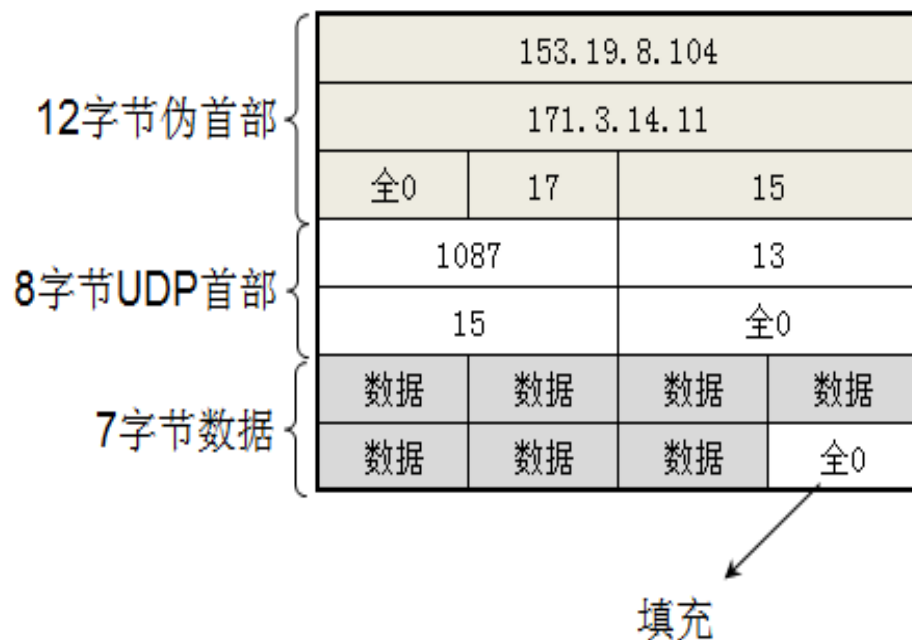
User Datagram Protocol (udp), 8 bytes Packets: 220 · Displayed: 220 (1... Profile: Default

8.2.2UDP的首部格式2

- UDP的首部包括四个字段，源端口、目标端口、长度和校验和，每个字段的长度是两个字节。
- 伪首部包括：源地址、目的地址、UDP数据长度、协议类型（0x11），协议类型就一个字节，但需要补一个字节的0x0，构成12个字节。



计算UDP校验和的例子



1001100100010011 → 153.19
0000100001101000 → 8.104
1010101100000011 → 171.3
0000111000001011 → 14.11
0000000000010001 → 0 和 17
0000000000001111 → 15
0000010000111111 → 1087
0000000000001101 → 13
0000000000001111 → 15
0000000000000000 → 0 (校验和)
0101010001000101 → 数据
0101001101010100 → 数据
0100100101001110 → 数据
0100011100000000 → 数据和 0 (填充)

按二进制反码运算求和
将得出的结果求反码

1001011011101101 → 求和得出的结果
0110100100010010 → 校验和

8.3传输控制协议TCP

■8.3.1TCP协议的主要特点

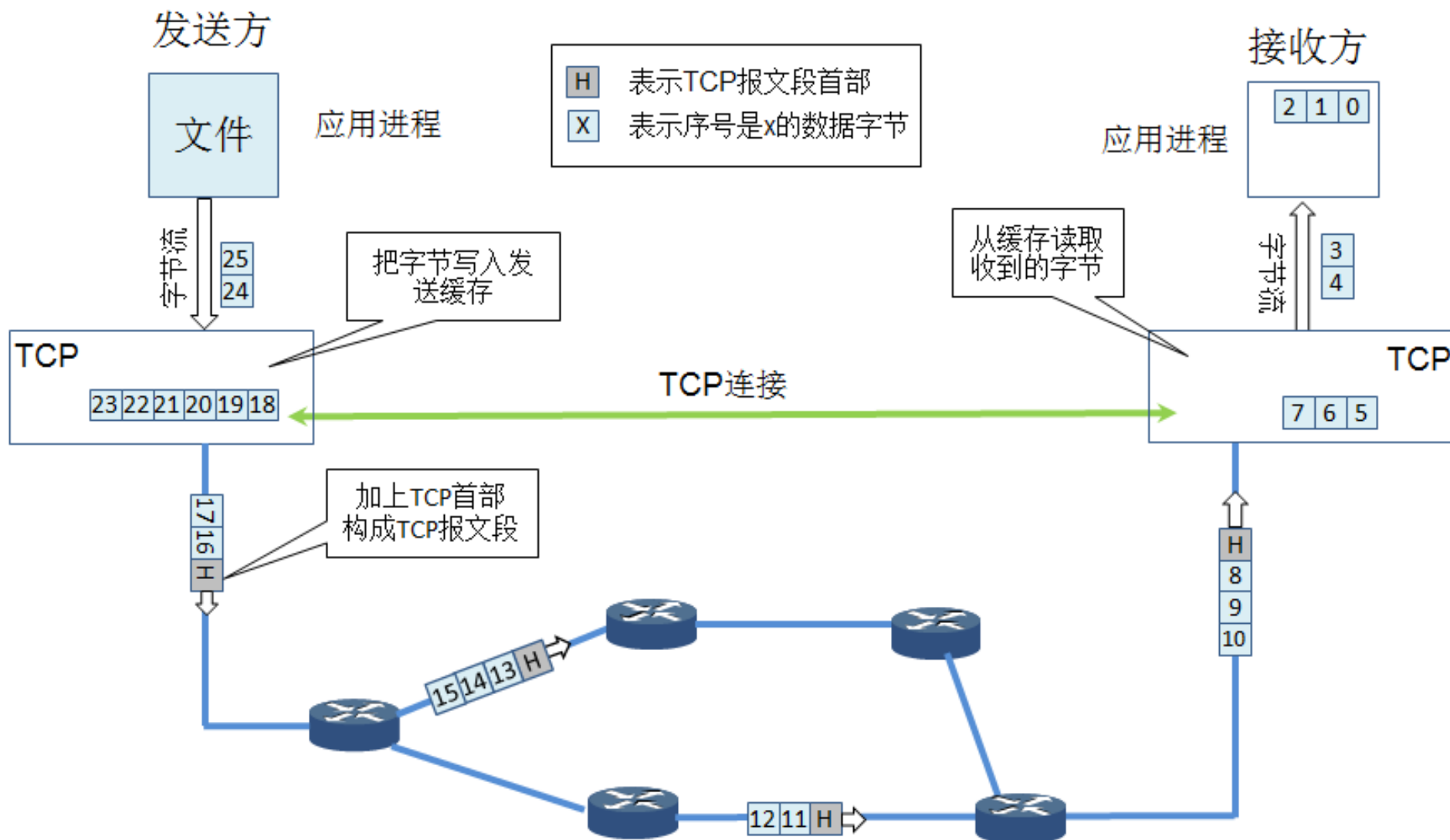
■8.3.2TCP报文的首部格式

8.3.1 TCP协议的主要特点

- (1) **TCP**是面向连接的传输层协议。这就是说，应用程序在使用**TCP**协议之前，必须先建立**TCP**连接。在传送数据完毕后，必须释放已经建立的**TCP**连接。这就是说，应用进程之间的通信好像在“打电话”：通话前要先拨号建立连接，通话结束后要挂机释放连接。
- (2) 每一条**TCP**连接只能有两个端点（**endpoint**），每一条**TCP**连接只能是点对点的（一对一）。
- (3) **TCP**提供可靠交付的服务。也就是说，通过**TCP**连接传送的数据，无差错、不丢失、不重复、且按序发送。
- (4) **TCP**提供全双工通信。**TCP**允许通信双方的应用进程在任何时候都能发送数据。**TCP**连接的两端都设有发送缓存和接收缓存，用来临时存放双向通信的数据。在发送时，应用程序把数据传送给**TCP**的缓存后，就可以做自己的事，而**TCP**在合适的时候把数据发送出去。在接收时，**TCP**把收到的数据放入缓存，上层的应用进程在合适的时候读取缓存中的数据。
- (5) 面向字节流。**TCP**中的“流”（**stream**）指的是流入到进程或从进程流出的字节序列。

面向字节流

TCP面向流的概念



8.3.2 TCP报文的首部格式1

- TCP协议是能够实现数据分段传输、可靠传输、流量控制、网络拥塞避免等功能，因此TCP报文的首部要比UDP报文首部字段要多，并且首部长度的不固定。

传输层首部

源端口

目标端口

序号

确认号

数据偏移

保留

紧急URG

确认ACK

推送PSH

复位RST

同步SYN

终止FIN

窗口

检验和

紧急指针

选项

Wireshark 1.12.4 (v1.12.4-0-gb4861da from master-1.12)

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Length	Info
5	0.05700500	192.168.80.100	59.46.80.160	TCP	54	1061→80 [ACK] Seq=
6	0.05733700	192.168.80.100	59.46.80.160	HTTP	362	GET / HTTP/1.1
7	0.05769200	59.46.80.160	192.168.80.100	TCP	60	80→1061 [ACK] Seq=
8	0.08015400	59.46.80.160	192.168.80.100	HTTP	452	HTTP/1.1 302 Found

Frame 6: 362 bytes on wire (2896 bits), 362 bytes captured (2896 bits) on interface...

Ethernet II, Src: Vmware_14:bf:02 (00:0c:29:14:bf:02), Dst: Vmware_f4:11:93 (00:50...

Internet Protocol Version 4, Src: 192.168.80.100 (192.168.80.100), Dst: 59.46.80.1...

Transmission Control Protocol, Src Port: 1061 (1061), Dst Port: 80 (80), Seq: 1, A...

Source Port: 1061 (1061)

Destination Port: 80 (80)

[Stream index: 0]

[TCP segment Len: 308]

Sequence number: 1 (relative sequence number)

[Next sequence number: 309 (relative sequence number)]

Acknowledgment number: 1 (relative ack number)

Header Length: 20 bytes

.... 0000 0001 1000 = Flags: 0x018 (PSH, ACK)

000. = Reserved: Not set

...0 = Nonce: Not set

... 0... = Congestion window Reduced (CWR): Not set

.... .0.. = ECN-Echo: Not set

.... ..0. = Urgent: Not set

.... ...1 = Acknowledgment: set

.... 1... = Push: set

....0.. = Reset: Not set

....0. = Syn: Not set

....0 = Fin: Not set

window size value: 64240

[Calculated window size: 64240]

[window size scaling factor: -2 (no window scaling used)]

Checksum: 0x9e29 [validation disabled]

[Good Checksum: False]

[Bad checksum: False]

urgent pointer: 0

[SEQ/ACK analysis]

[RTT: 0.021008000 seconds]

[Bytes in flight: 308]

Hypertext Transfer Protocol

0020 50 a0 04 25 00 50 84 2b a2 6e 3f cd 49 b8 50 18 P..%.P.+ .n?.I.P.

0030 fa f0 9e 29 00 00 47 45 54 20 2f 20 48 54 54 50 ...).GE T / HTTP

0040 2f 31 2e 31 0d 0a 41 63 63 65 70 74 3a 20 69 6d /1.1..Ac cept: im

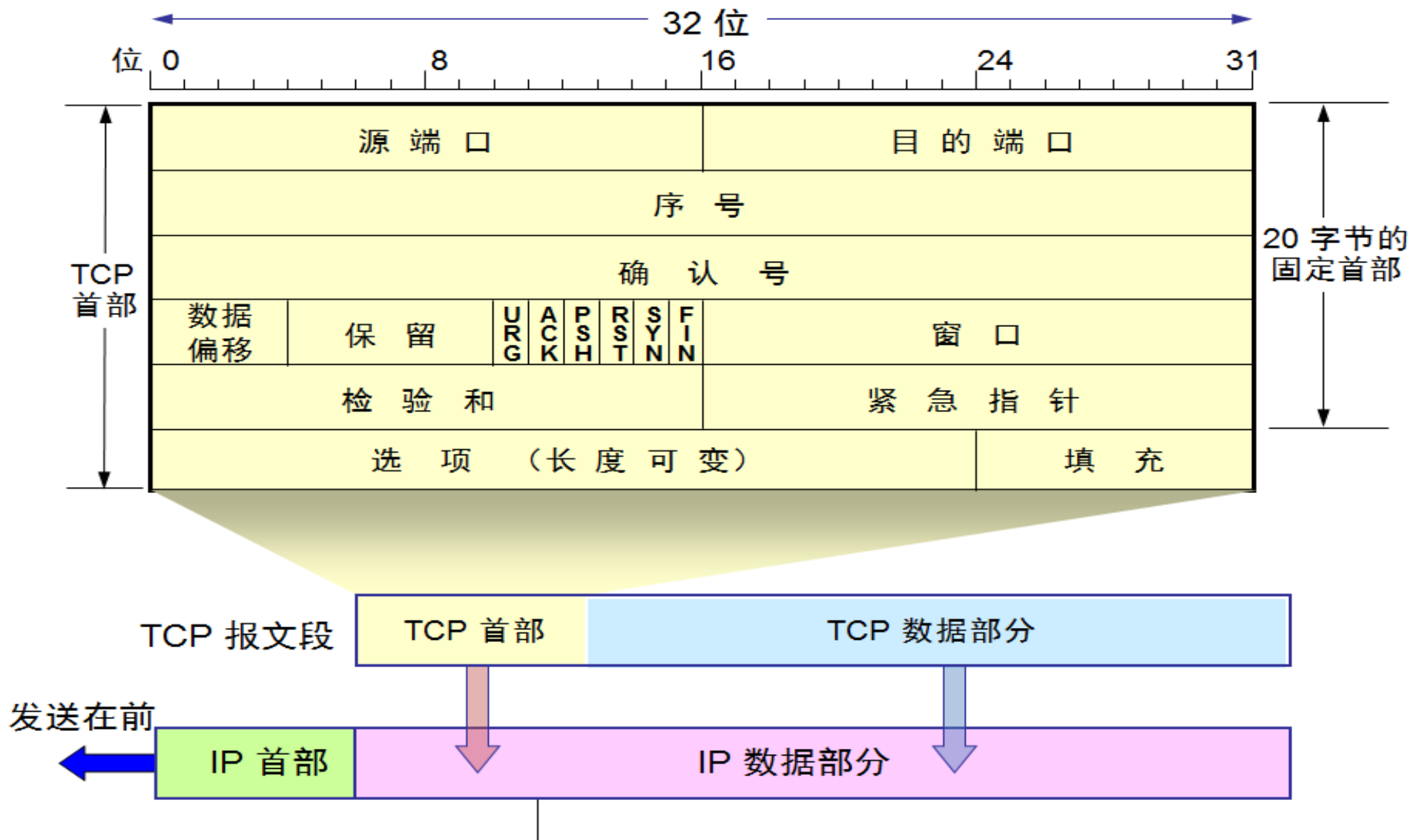
0050 61 67 65 2f 67 69 66 2c 20 69 6d 61 67 65 2f 78 age/gif, image/x

0060 2d 78 62 69 74 6d 61 70 2c 20 69 6d 61 67 65 2f -xbitmap , image/

0070 63 70 65 67 2c 20 69 6d 61 67 65 2f 70 63 70 65 image/

Transmission Control Protocol (tcp), 20 b... Packets: 220 · Displayed: 220 (Profile: Default

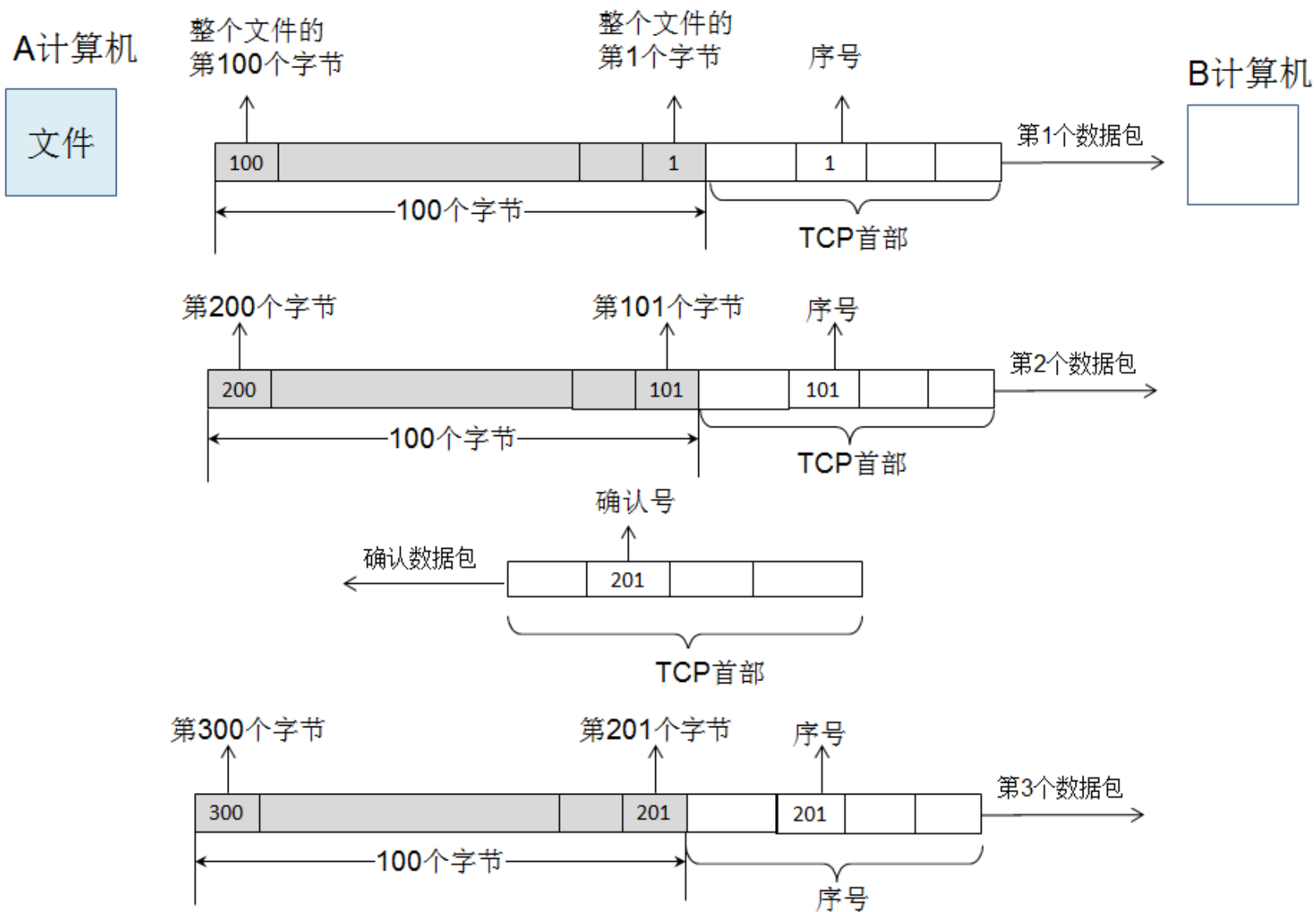
8.3.2 TCP报文的首部格式2



8.3.2 TCP报文的首部格式3

- (1) 源端口和目的端口各占2个字节，分别写入源端口号和目的端口号。和前面图所示的UDP的分用相似，TCP的分用功能也是通过端口实现的。
- (2) 序号占4字节。序号范围是 $[0, 2^{32}-1]$ ，共 2^{32} （即4 294 967 296）个序号。序号增加到 $2^{32}-1$ 后，下一个序号就又回到0。TCP是面向字节流的。在一个TCP连接中传送的字节流中的每一个字节都按顺序编号

序号 字段的意义



8.3.2 TCP报文的首部格式4

- (3) 确认号 占4字节，是期望收到对方下一个报文段的第一个数据字节的序号。
 - TCP协议能够实现可靠传输，接收方收到几个数据包后，就会给发送方一个确认数据包，告诉发送方下一个数据包该发第多少个字节了。
 - 若确认号是N，则表明：到序号N-1为止的所有数据都已正确收到。
- (4) 数据偏移 占4位，它指出TCP报文段的数据起始处距离TCP报文段的起始处有多远。这个字段实际上是指出TCP报文段的首部长度。由于首部中还有长度不确定的选项字段，因此数据偏移字段是必要的。但请注意，“数据偏移”的单位为4字节，由于4位二进制数能够表示的最大十进制数字是15，因此数据偏移的最大值是60字节，这也是TCP首部的最大长度，这也就意味着选项长度不能超过40字节。

8.3.2 TCP报文的首部格式5

- (6) 保留 占6位，保留为今后使用，但目前应置为0。
- (7) 紧急URG (URGent) 当URG=1时，表明紧急指针字段有效。它告诉系统此报文段中有紧急数据，应尽快传送（相当于高优先级的数据），而不要按原来的排队顺序来传送。
- 确认ACK (ACKnowledgment) 仅当ACK=1时确认号字段才有效。当ACK=0时，确认号无效。TCP规定，在连接建立后所有传送的报文段都必须把ACK置1。
- (9) 推送PSH (PuSH) 当两个应用进程进行交互式的通信时，有时在一端的应用进程希望在键入一个命令后立即就能够收到对方的响应。

8.3.2 TCP报文的首部格式6

- **(10) 复位RST (ReSeT)** 当RST=1时, 表明TCP连接中出现严重差错(如由于主机崩溃或其他原因), 必须释放连接, 然后再重新建立运输连接。
- **(11) 同步SYN (SYNchronization)** 在连接建立时用来同步序号。当SYN=1而ACK=0时, 表明这是一个连接请求报文段。对方若同意建立连接, 则应在响应的报文段中使SYN=1和ACK=1。因此, SYN置为1就表示这是一个连接请求或连接接受报文。
- **(12) 终止FIN (FINish意思是“完”、“终”)** 用来释放一个连接。当FIN=1时, 表明此报文段的发送方的数据已发送完毕, 并要求释放传输连接。

8.3.2 TCP报文的首部格式7

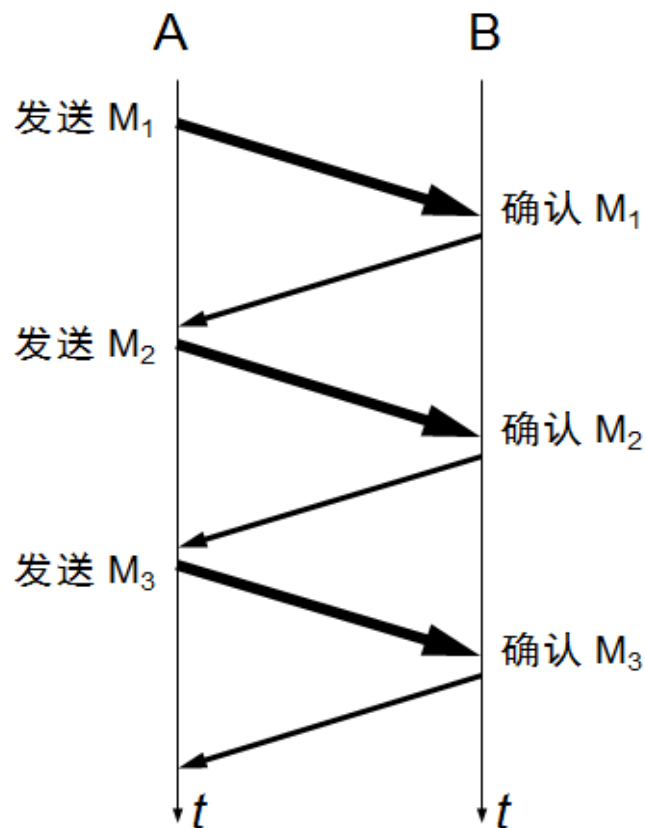
- (13) 窗口 占2字节。窗口值是 $[0, 2^{16}-1]$ 之间的整数。TCP协议有流量控制功能，窗口值告诉对方：从本报文段首部中的确认号算起，接收方目前允许对方发送的数据量（单位是字节）。
- (14) 检验和 占2字节。检验和字段检验的范围包括首部和数据这两部分。和UDP用户数据报一样，在计算检验和时，要在TCP报文段的前面加上12字节的伪首部。
- (15) 紧急指针 占2字节。紧急指针仅在URG=1时才有意义，它指出本报文段中的紧急数据的字节数（紧急数据结束后就是普通数据）。因此紧急指针指出了紧急数据的末尾在报文段中的位置。
- (16) 选项 长度可变，最长可达40个字节。当没有使用选项时，TCP的首部长度是20字节。TCP最初只规定了一种选项，即最大报文段长度MSS（Maximum Segment Size）。

8.4可靠传输

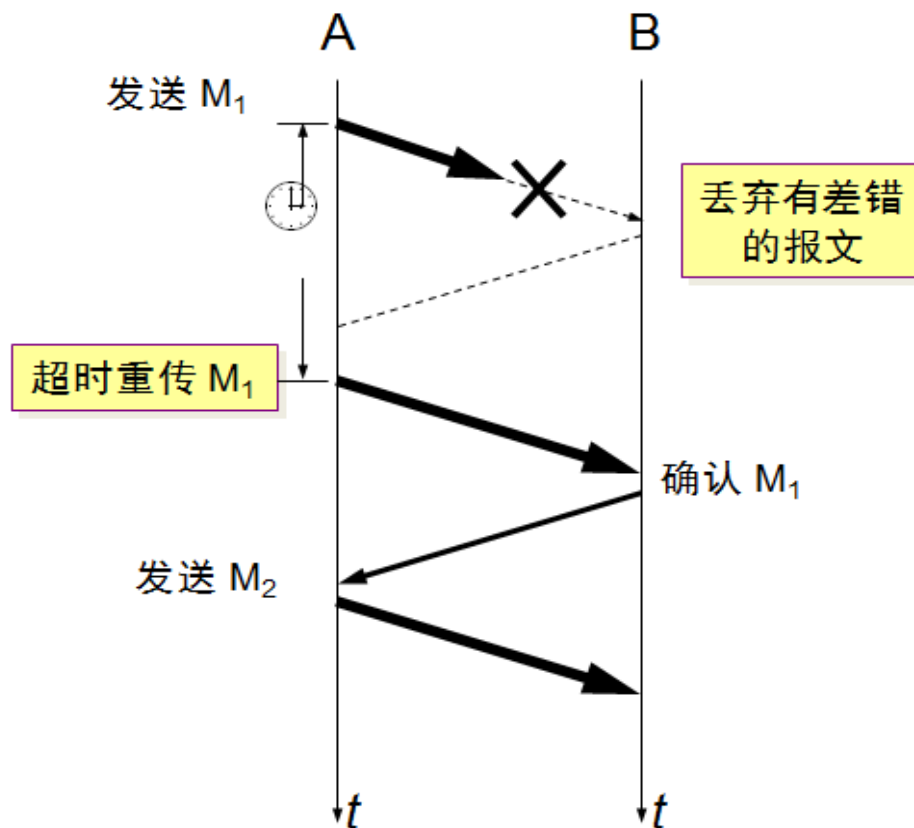
- 8.4.1TCP可靠传输的实现-停止等待协议
- 8.4.2连续ARQ协议和滑动窗口协议-改进的停止等待协议
- 8.4.3以字节为单位的滑动窗口技术详解
- 8.4.4改进的确认-选择确认（SACK）
- 8.4.5超时重传的时间调整

8.4.1 TCP可靠传输的实现-停止等待协议1

■无差错情况



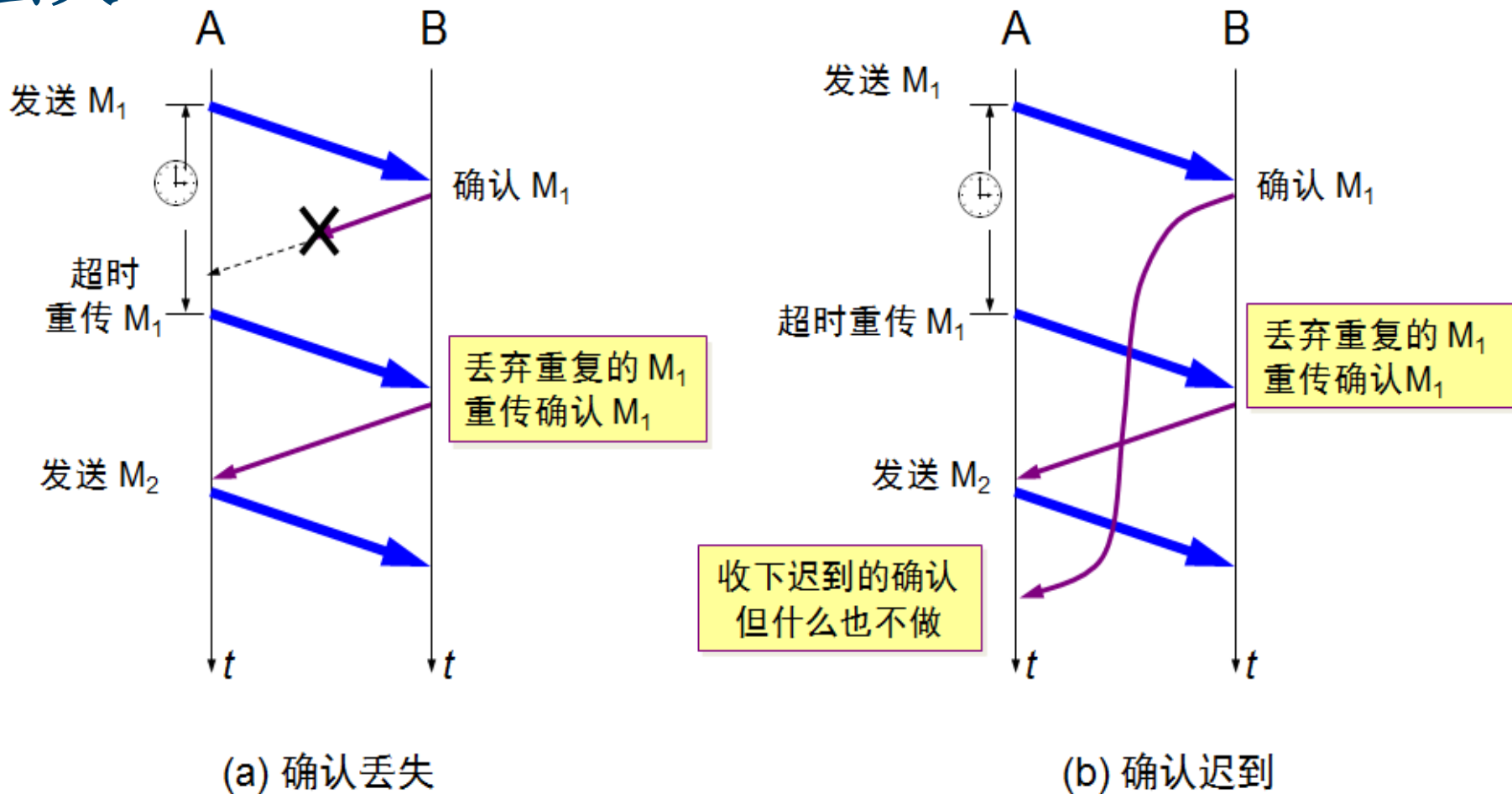
(a) 无差错情况



(b) 超时重传

8.4.1 TCP可靠传输的实现-停止等待协议2

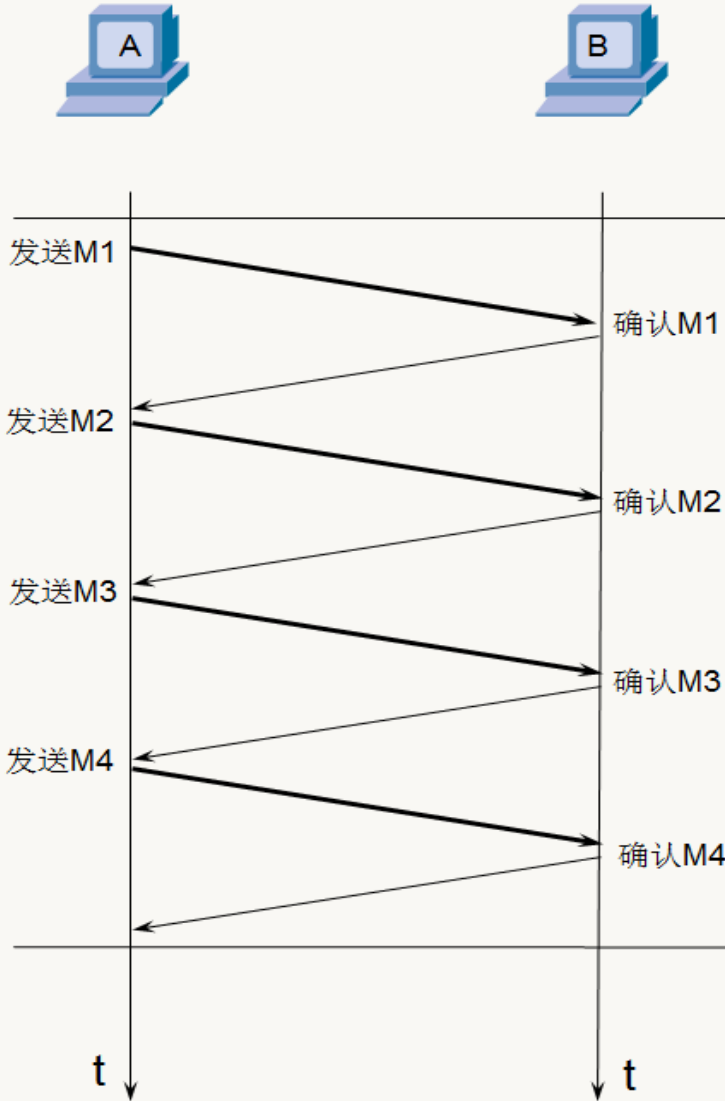
■出现差错或丢失



连续ARQ协议和滑动窗口协议-改进的停止等待协议

停止等待协议

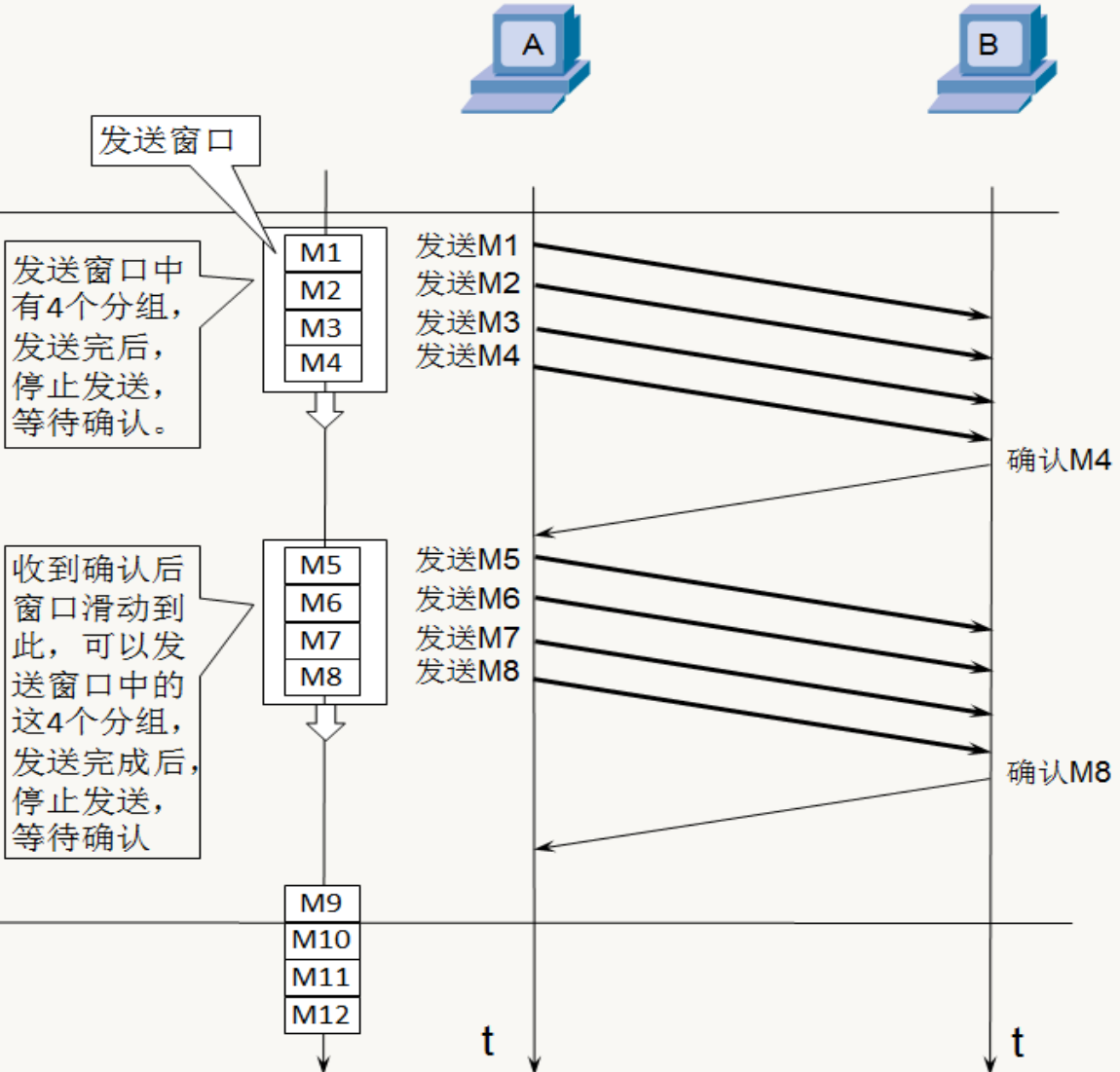
发送一个分组就停止发送等待确认



(a)

连续ARQ协议和滑动窗口协议

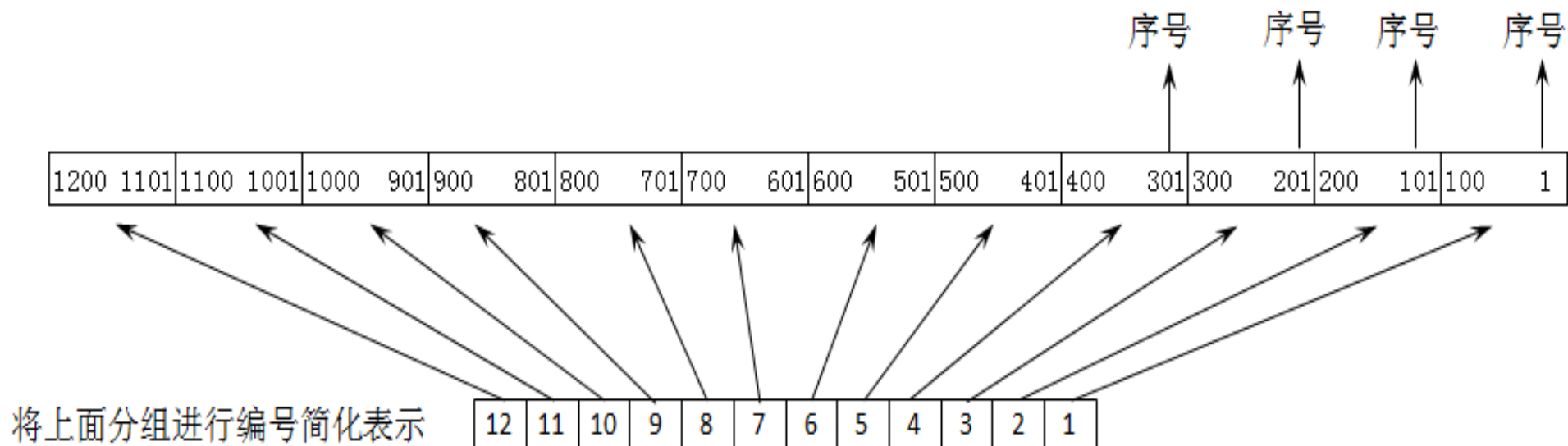
发送窗口中的分组连续发送，发送完后，停止等待确认

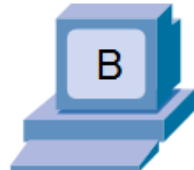
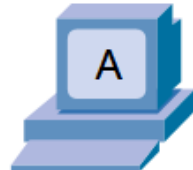


(b)

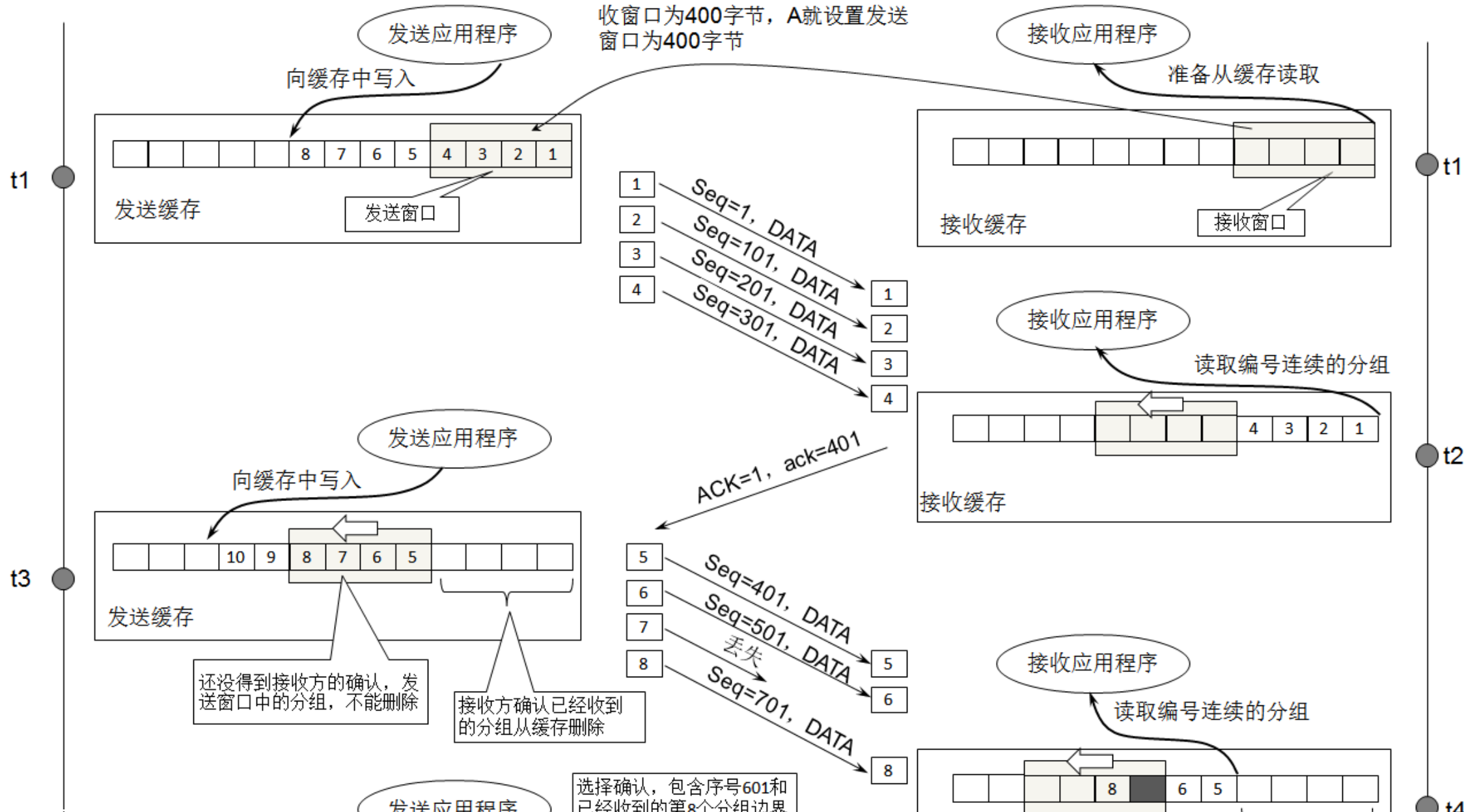
8.4.3以字节为单位的滑动窗口技术详解

- 滑动窗口是面向字节流的，为了方便大家记住每个分组的序号，下面的讲解每一个分组就假设**100个字节**，为了方便画图表示，将分组进行编号简化表示，如图所示，不过你要记住，每一





建立TCP连接时B告诉A字节的接收窗口为400字节，A就设置发送窗口为400字节



8.4.4改进的确认-选择确认 (SACK)

■TCP通信时，如果发送序列中间某个数据包丢失，TCP会通过重传最后确认的分组后续的分组，这样原先已经正确传输的分组也可能重复发送，降低了TCP性能。为改善这种情况，发展出SACK（Selective Acknowledgment，选择确认）技术，使TCP只重新发送丢失的包，不用发送后续所有的分组，而且提供相应机制使接收方能告诉发送方哪些数据丢失，哪些数据已经提前收到等。

8.4.4改进的确认-选择确认 (SACK)

The image shows a Wireshark packet capture analysis of a TCP segment. The packet is identified as Frame 1514, 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0. It is an Ethernet II frame from Src: AsustekC_2e:6e:1e (c8:60:00:2e:6e:1e) to Dst: 50:da:00:ce:11:3c (50:da:00:ce:11:3c). The Internet Protocol Version 4 header shows Src: 10.7.10.18 and Dst: 210.32.92.135. The Transmission Control Protocol header shows Src Port: 14921 and Dst Port: 80. The TCP segment has Seq: 1105, Ack: 49641, and Win: 65700. The segment length is 0 bytes. The flags are 0x010 (ACK). The window size value is 16425, and the calculated window size is 65700. The checksum is 0x42e7. The urgent pointer is 0. The options list includes No-Operation (NOP), No-Operation (NOP), and SACK: 51101-51454. The SACK option is highlighted in blue, and its details are shown in the right pane: Kind: SACK (5), Length: 10, left edge = 51101 (relative), right edge = 51454 (relative), and [TCP SACK Count: 1].

确认号 → Acknowledgment number: 49641 (relative ack number)

首部长度的 → Header Length: 32 bytes

接收窗口 → window size value: 16425
[Calculated window size: 65700]
[window size scaling factor: 4]

TCP选项 → Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), SACK

指明是SACK → SACK: 51101-51454

该选项长度 → Kind: SACK (5) → 1个字节
Length: 10 → 1个字节

左边界 → left edge = 51101 (relative) → 4个字节

右边界 → right edge = 51454 (relative) → 4个字节

[TCP SACK Count: 1]

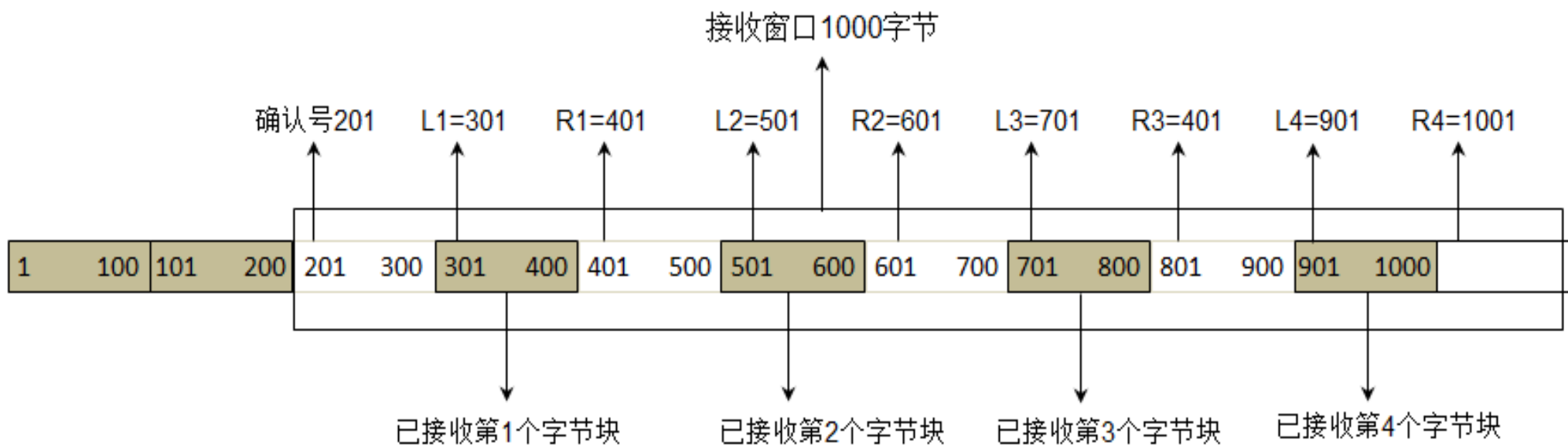
[SEQ/ACK analysis]

确认数据包没有数据部分 只有TCP首部（包括选项）

0000 50 da 00 ce 11 3c c8 60 00 2e 6e 1e 08 00 45 00 P...<...n...E.
0010 00 34 7e f1 40 00 80 06 00 00 0a 07 0a 12 d2 20 .4~.@... ..
0020 5c 87 3a 49 00 50 c7 fb 98 a0 df 66 dd c0 80 10 \.:I.P... ..f...
0030 40 28 42 e7 00 01 01 05 0a df 66 e3 74 df 66 @)B...df...f+f

选择性确认最多表示4个边界

- 由于TCP首部选项最长40个字节，而指明一个边界需要用掉4个字节（因为序号有32位，需要使用4个字节表示），因此在TCP选项中一次最多只能指明4个字节块的边界信息。这是因为4个字节块有8个边界，一个边界占用4个字节，占用32个字节，另外还需要2个字节，一个字节用来指明是SACK选项，另一个字节指明这个选项占多少字节。



8.4.5 超时重传的时间调整

- TCP的发送方在规定的时间内没有收到确认就要重传已发送的报文段。这种重传的概念是很简单的，但重传时间的选择却是TCP最复杂的问题之一。
- 传输层的超时计时器的超时重传时间究竟应设置为多大呢？TCP往返传输时间（RTT）的测量可以采用两种方法：
- TCP Timestamp选项
 - TCP时间戳选项可以用来精确的测量RTT。发送方在发送报文段时把当前时钟的时间值放入时间戳字段，接收方在确认该报文段时把时间戳字段值复制到时间戳回送回答字段。因此，发送方在收到确认报文后，可以准确地计算出RTT来。 $RTT = \text{当前时间} - \text{数据包中Timestamp选项的回显时间}$ 。
- 重传队列中数据包的TCP控制块
 - 在TCP发送窗口中保存着发送而未被确认的数据包，数据包skb中的TCP控制块包含着一个变量，`tcp_skb_cb->when`，记录了该数据包的第一次发送时间，当收到该数据包确认，就可以计算RTT， $RTT = \text{当前时间} - \text{when}$ 。这就意味着发送端收到一个确认，就能计算新的RTT。

抓包计时

建立TCP连接请求

建立TCP连接响应

第5个包的确认

计算出的RTT

Wireshark 1.12.4 (v1.12.4-0-gb46b1da from master-1.12.4)

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Vmware_14:bf:02	Broadcast	ARP	42	who has 192.168.80.1
2	0.002666000	Vmware_f4:11:93	Vmware_14:bf:02	ARP	60	192.168.80.1 is at
3	0.002680000	192.168.80.100	114.114.114.114	DNS	75	standard query 0x51
4	0.068124000	114.114.114.114	192.168.80.100	DNS	91	standard query respo
5	0.068758000	192.168.80.100	59.46.80.160	TCP	62	1616->80 [SYN] Seq=0
6	0.101120000	59.46.80.160	192.168.80.100	TCP	60	80->1616 [SYN, ACK] s
7	0.101154000	192.168.80.100	59.46.80.160	TCP	54	1616->80 [ACK] Seq=1

Frame 6: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0

Ethernet II, Src: Vmware_f4:11:93 (00:50:56:f4:11:93), Dst: Vmware_14:bf:02 (00:0c:29:14:bf:02)

Internet Protocol Version 4, Src: 59.46.80.160 (59.46.80.160), Dst: 192.168.80.100 (192.168.80.100)

Transmission Control Protocol, Src Port: 80 (80), Dst Port: 1616 (1616), Seq: 0, Ack: 1, Len: 0

Source Port: 80 (80)

Destination Port: 1616 (1616)

[Stream index: 0]

[TCP Segment Len: 0]

Sequence number: 0 (relative sequence number)

Acknowledgment number: 1 (relative ack number)

Header Length: 24 bytes

.... 0000 0001 0010 = Flags: 0x012 (SYN, ACK)

window size value: 64240

[Calculated window size: 64240]

Checksum: 0x31d5 [validation disabled]

Urgent pointer: 0

Options: (4 bytes), Maximum segment size

[SEQ/ACK analysis]

[This is an ACK to the segment in frame: 5]

[The RTT to ACK the segment was: 0.032362000 seconds]

[RTT: 0.032396000 seconds]

0000 00 0c 29 14 bf 02 00 50 56 f4 11 93 08 00 45 00 ..).P.V.....E.

0010 00 2c fe de 00 00 80 06 9f 12 3b 2e 50 a0 c0 a8P...

0020 50 64 00 50 06 50 7a a4 75 09 73 56 64 d1 60 12 Pd.P.Pz. u.svd..

0030 fa f0 31 d5 00 00 02 04 05 b4 00 00 ..1.....

Transmission Control Protocol (tcp), 24 b... Packets: 320 · Displayed: 320 (100.0%) Profile: Default

RTT的调整

- RTT是随着网络状态动态的变化的，TCP保留了RTT的一个加权平均往返时间RTTs（这又称为平滑的往返时间，S表示Smoothed。因为进行的是加权平均，因此得出的结果更加平滑）。每当第一次测量到RTT样本时，RTTs值就取为所测量到的RTT样本值。但以后每测量到一个新的RTT样本，就按下式重新计算一次RTTs：

$$\text{新的RTTs} = (1-\alpha) \times (\text{旧的RTTs}) + \alpha \times (\text{新的RTT样本})$$

- 在上式中， $0 \leq \alpha < 1$ 。若 α 很接近于零，表示新的RTTs值和旧的RTT值相比变化不大，而对新的RTT样本影响不大，RTT值更新较慢）。若选择 α 接近于1，则表示新的RTTs值受新的RTT样本的影响较大（RTT值更新较快），RFC2988推荐的 α 值为1/8，即0.125。用这种方法得出的加权平均往返时间RTTs就比测量出的RTT值更加平滑。

超时计时器设置的超时重传时间RTO

- 超时计时器设置的超时重传时间RTO（RetransmissionTime-Out）应略大于上面得出的加权平均往返时间RTTs。RFC2988建议使用下式计算RTO：

$$RTO = RTTs + 4 \times RTT_D$$

- 而 RTT_D 是RTT的偏差的加权平均值，它与RTTs和新的RTT样本之差有关。RFC2988建议这样计算 RTT_D 。当第一次测量时， RTT_D 值取为测量到的RTT样本值的一半。在以后的测量中，则使用下式计算加权平均的 RTT_D 。

$$\text{新的}RTT_D = (1-\beta) \times (\text{旧的}RTT_D) + \beta \times |RTTs - \text{新的}RTT\text{样本}|$$

- 这里 β 是个小于1的系数，它的推荐值是1/4，即0.25。

8.5流量控制1

Capturing from 本地连接 [Wireshark 1.12.4 (v1.12.4-0-gb4861da from master-1.12)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.80.100	114.114.114.114	DNS	75	Standard query 0xc045 A www.91xueit.com
2	0.040048000	114.114.114.114	192.168.80.100	DNS	91	Standard query response 0xc045 A 59.46.80.160
3	0.040916000	192.168.80.100	59.46.80.160	TCP	62	1059→80 [SYN] Seq=0 win=64240 Len=0 MSS=1460 SACK_PERM=1
4	0.067417000	59.46.80.160	192.168.80.100	TCP	60	80→1059 [SYN, ACK] Seq=0 Ack=1 win=64240 Len=0 MSS=1460
5	0.067444000	192.168.80.100	59.46.80.160	TCP	54	1059→80 [ACK] Seq=1 Ack=1 win=64240 Len=0
6	0.067585000	192.168.80.100	59.46.80.160	HTTP	309	GET / HTTP/1.1
7	0.068125000	59.46.80.160	192.168.80.100	TCP	60	80→1059 [ACK] Seq=1 Ack=256 win=64240 Len=0
8	0.106718000	59.46.80.160	192.168.80.100	HTTP	452	HTTP/1.1 302 Found (text/html)
9	0.107711000	192.168.80.100	59.46.80.160	HTTP	329	GET /91xueit/default.html HTTP/1.1
10	0.108973000	59.46.80.160	192.168.80.100	TCP	60	80→1059 [ACK] Seq=399 Ack=531 win=64240 Len=0
11	0.138373000	59.46.80.160	192.168.80.100	TCP	1514	[TCP segment of a reassembled PDU]
12	0.138543000	59.46.80.160	192.168.80.100	TCP	1514	[TCP segment of a reassembled PDU]
13	0.138551000	59.46.80.160	192.168.80.100	TCP	606	[TCP segment of a reassembled PDU]
14	0.138579000	192.168.80.100	59.46.80.160	TCP	54	1059→80 [ACK] Seq=531 Ack=3871 win=64240 Len=0
15	0.164748000	59.46.80.160	192.168.80.100	TCP	1514	[TCP segment of a reassembled PDU]
16	0.164775000	59.46.80.160	192.168.80.100	TCP	1514	[TCP segment of a reassembled PDU]
17	0.164781000	59.46.80.160	192.168.80.100	TCP	606	[TCP segment of a reassembled PDU]
18	0.164809000	192.168.80.100	59.46.80.160	TCP	54	1059→80 [ACK] Seq=531 Ack=7343 win=64240 Len=0
19	0.189943000	59.46.80.160	192.168.80.100	TCP	1514	[TCP segment of a reassembled PDU]
20	0.189958000	59.46.80.160	192.168.80.100	TCP	1514	[TCP segment of a reassembled PDU]
21	0.189963000	59.46.80.160	192.168.80.100	TCP	1514	[TCP segment of a reassembled PDU]
22	0.189965000	59.46.80.160	192.168.80.100	TCP	1514	[TCP segment of a reassembled PDU]
23	0.189967000	59.46.80.160	192.168.80.100	TCP	1158	[TCP segment of a reassembled PDU]
24	0.189984000	192.168.80.100	59.46.80.160	TCP	54	1059→80 [ACK] Seq=531 Ack=14287 win=63136 Len=0
25	0.190897000	59.46.80.160	192.168.80.100	TCP	1514	[TCP segment of a reassembled PDU]
26	0.190925000	59.46.80.160	192.168.80.100	TCP	1514	[TCP segment of a reassembled PDU]
27	0.190927000	59.46.80.160	192.168.80.100	TCP	606	[TCP segment of a reassembled PDU]
28	0.190936000	192.168.80.100	59.46.80.160	TCP	54	1059→80 [ACK] Seq=531 Ack=17759 win=59664 Len=0
29	0.192073000	192.168.80.100	59.46.80.160	TCP	54	[TCP window update] 1059→80 [ACK] Seq=531 Ack=17759 win=63424

0000 00 50 56 f4 11 93 00 0c 29 14 bf 02 08 00 45 00 .PV.....).....E.
0010 00 30 02 55 40 00 80 06 5b 98 c0 a8 50 64 3b 2e .0.U@...[...Pd;.
0020 50 a0 04 23 00 50 12 b9 4e ef 00 00 00 00 70 02 P..#.P..N....p.
0030 fa f0 85 38 00 00 02 04 05 b4 01 01 04 02 ...8....

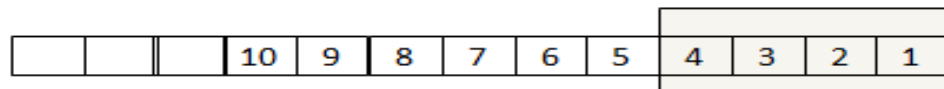
Ready to load or capture Packets: 70993 · Displayed: 70993 (100.0%) Profile: Default

客户端建立TCP连接告诉发送方接收窗口

确认数据包中Win用来调整发送端窗口大小

8.5流量控制2

发送窗口400

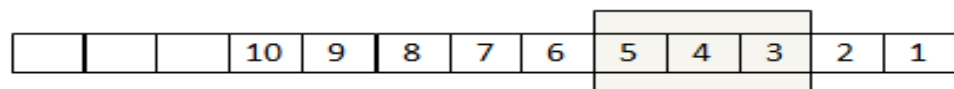


A发送序号1-100, 还能发送300字节

A发送序号101-200, 还能发送200字节

A发送序号201-300, 还能发送100字节

发送窗口300

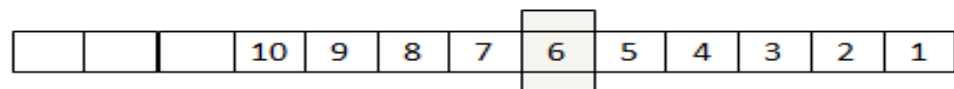


窗口前移, A发送序号301-400, 还能发送100字节

A发送序号401-500, 停止发送

A超时重传丢失的数据, 不能发送新的数据

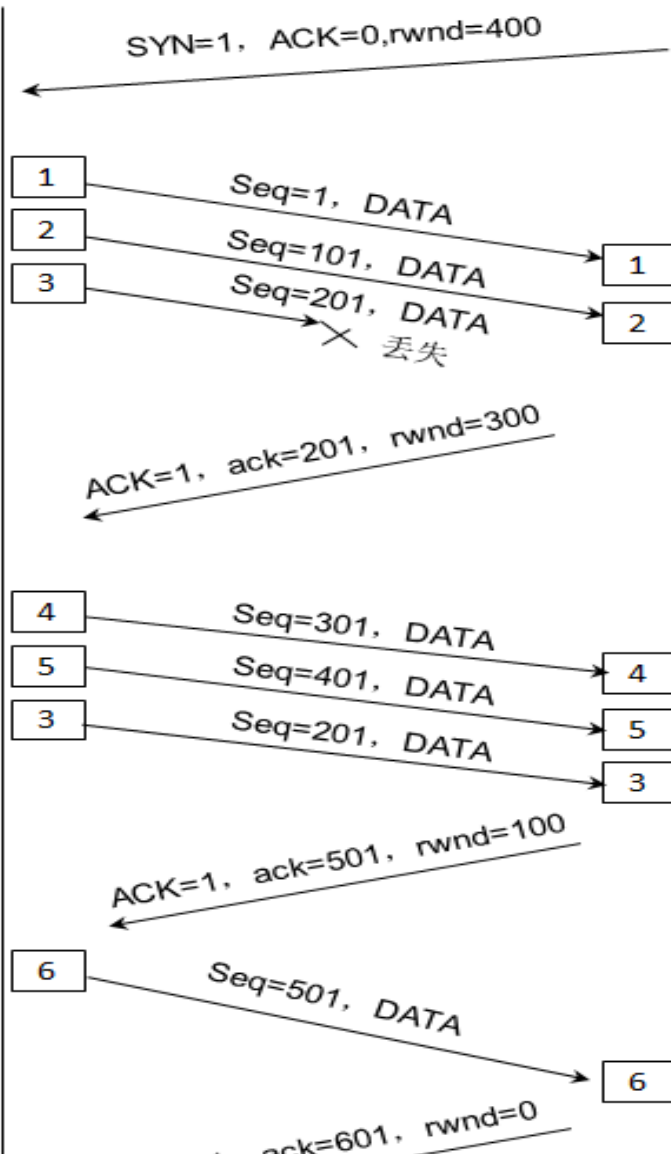
发送窗口100



窗口前移, 调整发送窗口大小, 只能发送100字节

发送窗口0

A



B

B向A发起建立TCP连接的请求
win指明窗口大小

第3个分组丢失, 发送确认,
同时调整窗口大小为300

发送确认,
同时调整窗口大小为100

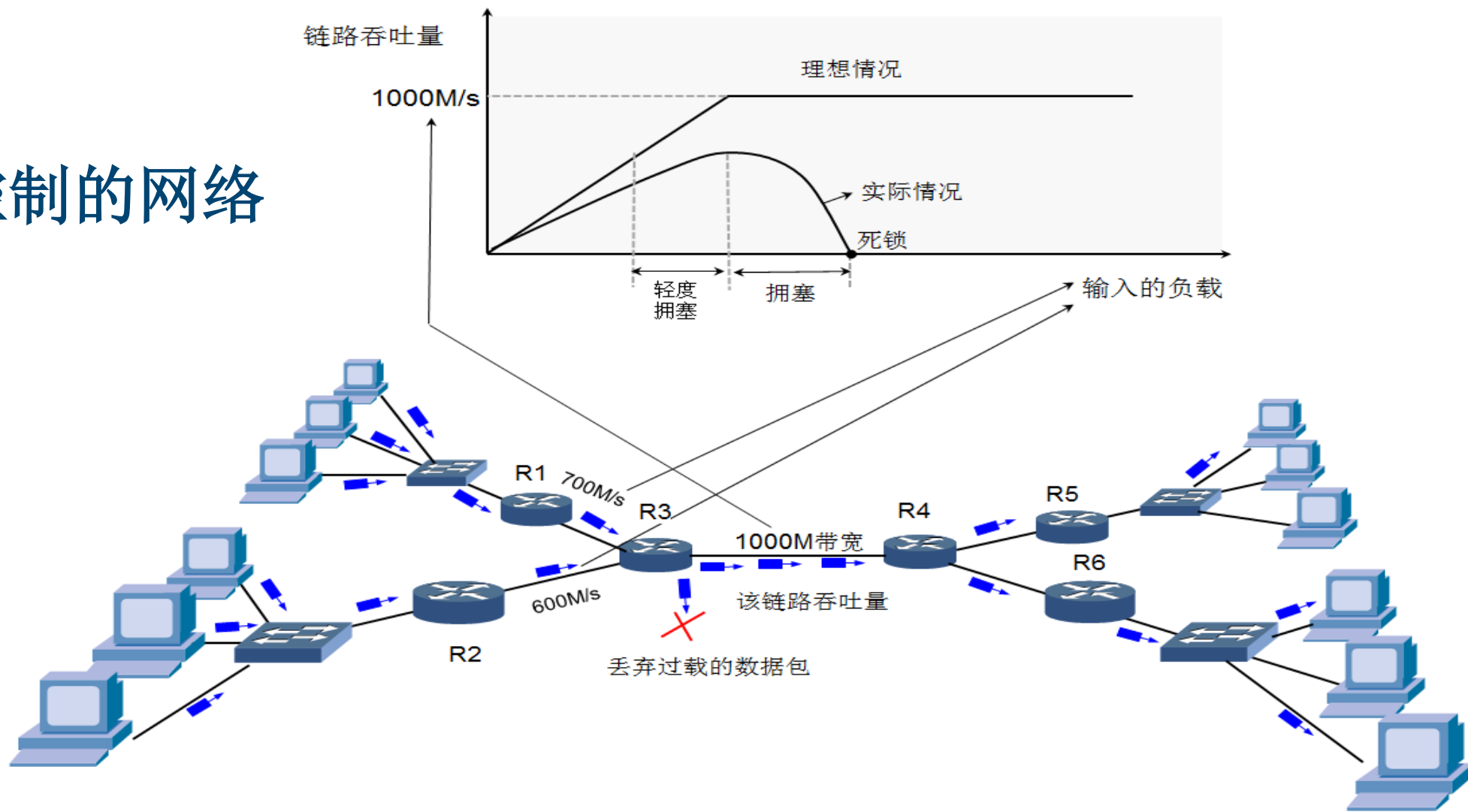
B接收缓存满了,
打算让A停止发送,
发送确认,
设置窗口大小为0

8.6拥塞控制

- 8.6.1拥塞控制的原理
- 8.6.2拥塞控制方法-慢开始和拥塞避免
- 8.6.3拥塞控制方法-快重传和快恢复
- 8.6.4发送窗口的上限

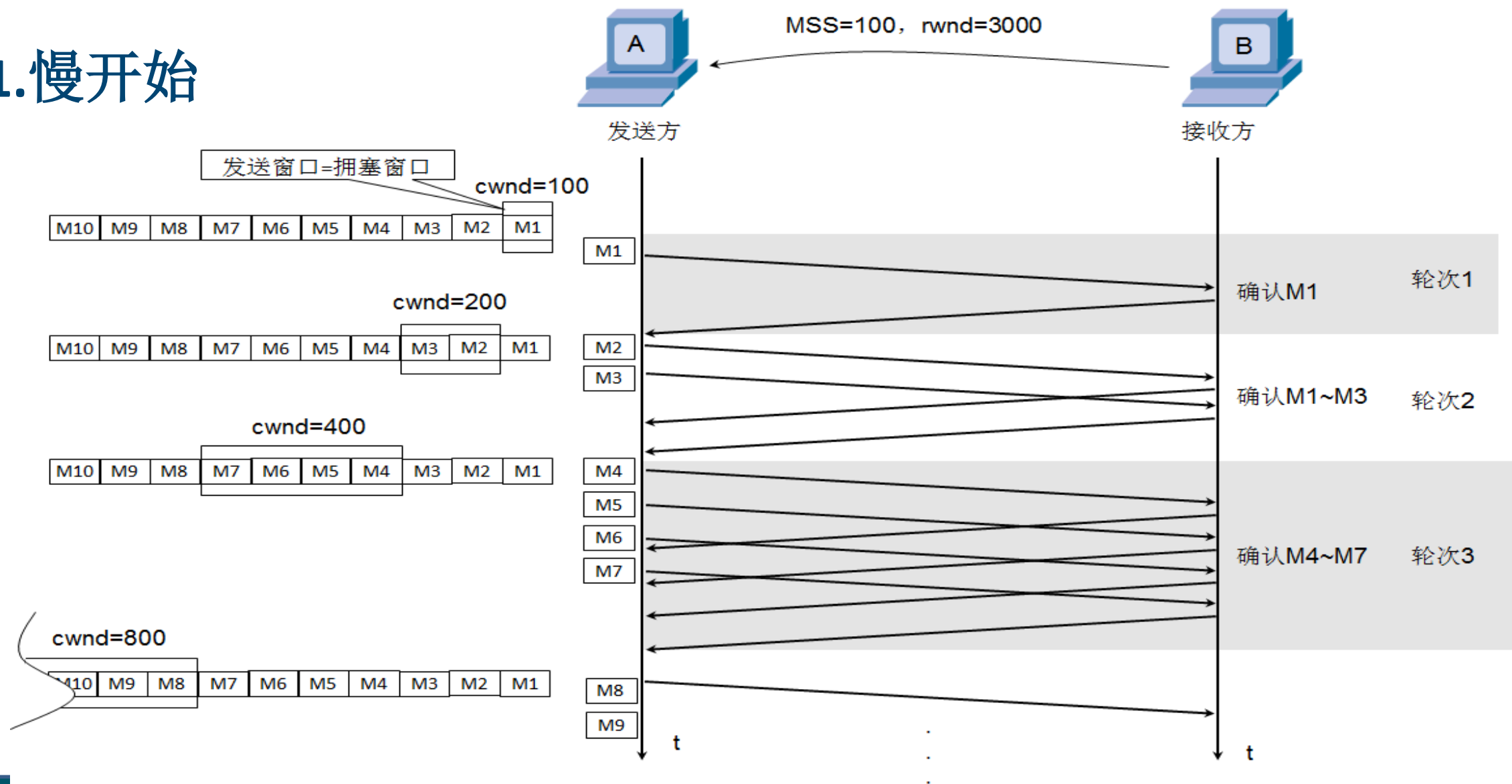
8.6.1 拥塞控制的原理

■ 有拥塞控制的网络



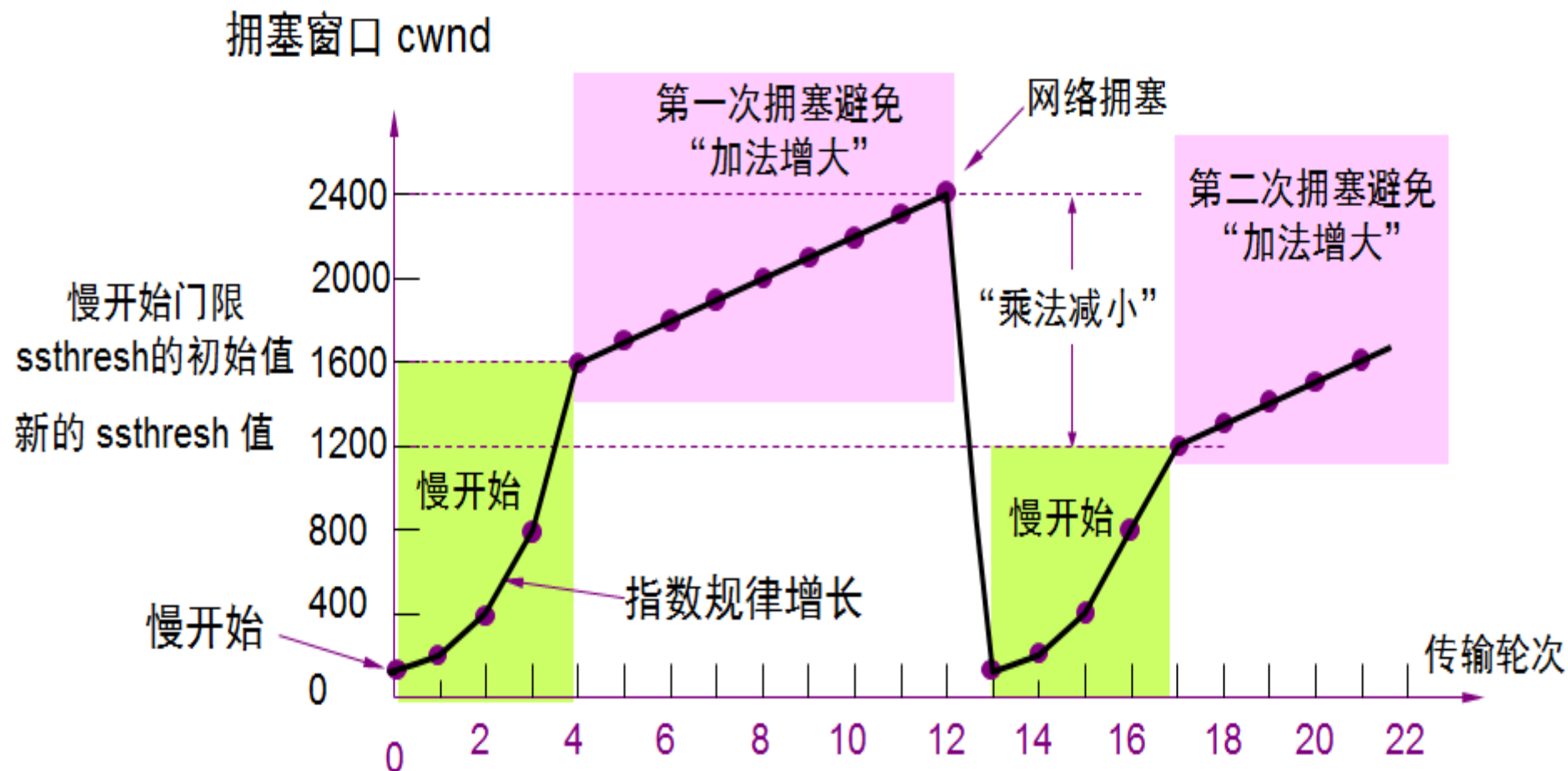
8.6.2 拥塞控制方法-慢开始和拥塞避免1

1. 慢开始



8.6.2 拥塞控制方法-慢开始和拥塞避免2

■ 2. 拥塞避免算法

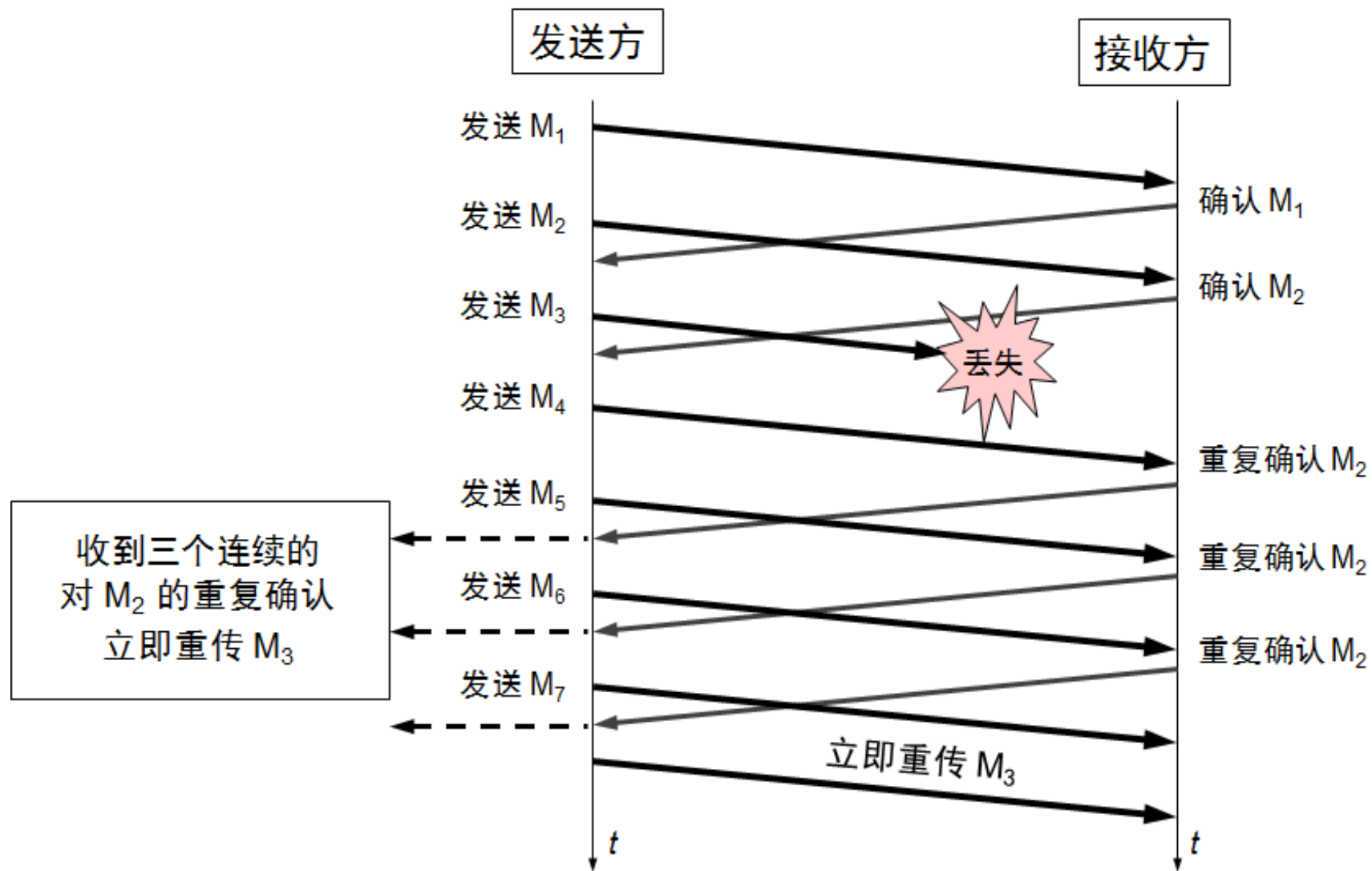


8.6.3 拥塞控制方法-快重传和快恢复

■ 快重传

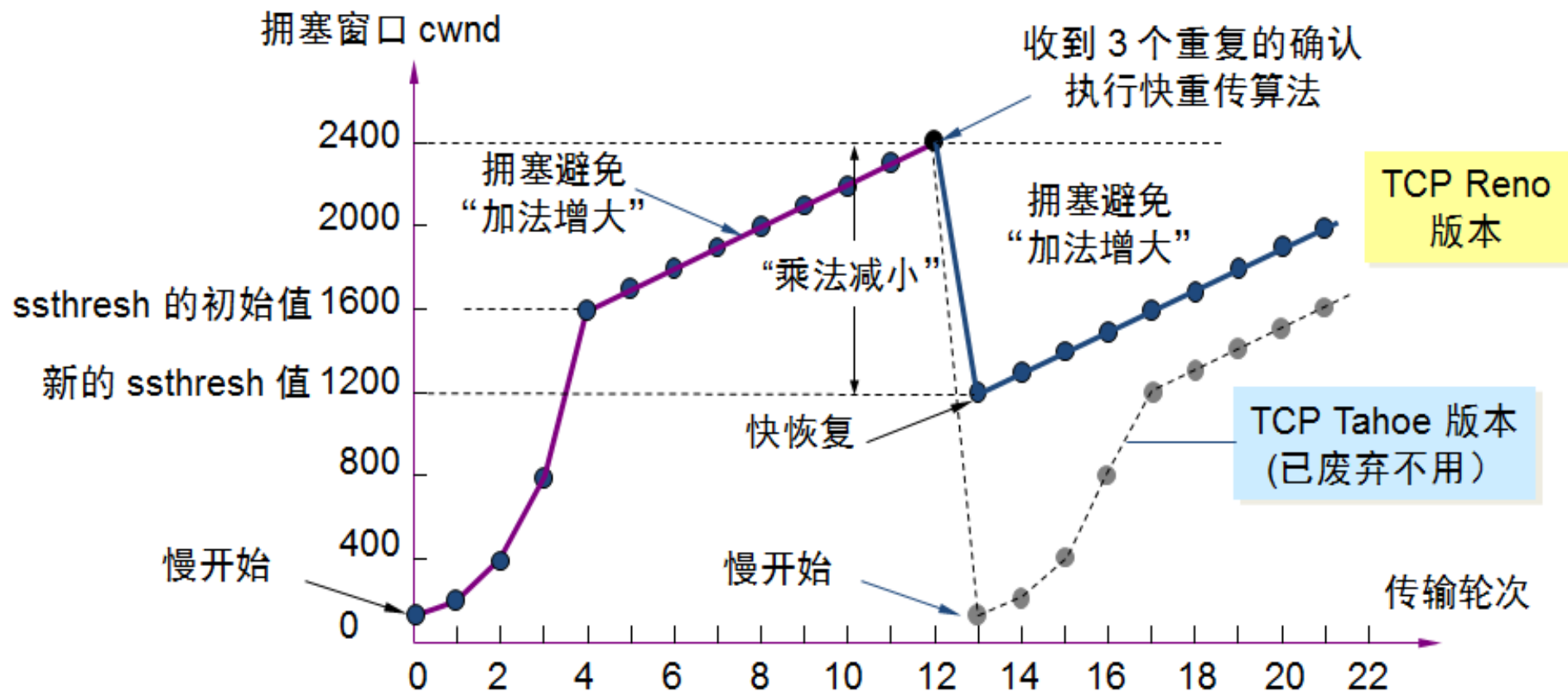
- 快重传算法首先要求接收方每收到一个失序的分组后就立即发出重复确认（为的是使发送方及早知道有分组没有到达对方）而不要等待自己发送数据时才进行捎带确认。
- 快重传算法规定，发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段M3，而不必继续等待为M3设置的重传计时器到期。

快重传



8.6.3 拥塞控制方法-快重传和快恢复

■ 与快重传配合使用的还有快恢复算法



8.6.4发送窗口的上限

- 如果把本节所讨论的拥塞控制和接收方对发送方的流量控制一起考虑，那么很显然，发送方的窗口的上限值应当取为接收方窗口rwnd和拥塞窗口cwnd这两个变量中较小的一个，也就是说：

发送方窗口的上限值 = $\text{Min}[\text{rwnd}, \text{cwnd}]$

- 当 $\text{rwnd} < \text{cwnd}$ 时，是接收方的接收能力限制发送方窗口的最大值。
- 反之，当 $\text{cwnd} < \text{rwnd}$ 时，则是网络的拥塞限制发送方窗口的最大值。
- 也就是说，rwnd和cwnd中较小的一个控制发送方发送数据的速率。

8.7 TCP连接管理

- 8.7.1 TCP的连接建立
- 8.7.2TCP连接释放
- 8.7.3实战：查看TCP释放连接的数据包
- 8.7.4实战：SYN攻击

8.7.1 TCP建立连接-请求建立TCP连接的数据包

Frame 3: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0

Ethernet II, Src: VMware_14:bf:02 (00:0c:29:14:bf:02), Dst: VMware_f4:11:93 (00:50:56:f4:11:93)

Internet Protocol Version 4, Src: 192.168.80.100 (192.168.80.100), Dst: 59.46.80.160 (59.46.80.160)

Transmission Control Protocol, Src Port: 1033 (1033), Dst Port: 80 (80), Seq: 0, Len: 0

Source Port: 1033 (1033)

Destination Port: 80 (80)

[Stream index: 0]

[TCP Segment Len: 0]

Sequence number: 0 ← (relative sequence number)

Acknowledgment number: 0

Header Length: 28 bytes

... 0000 0000 0010 = Flags: 0x002 (SYN)

000. = Reserved: Not set

...0 = Nonce: Not set

.... 0... = Congestion window Reduced (CWR): Not set

.... .0.. = ECN-Echo: Not set

.... ..0. = Urgent: Not set

.... ...0 = Acknowledgment: Not set

.... 0... = Push: Not set

.... 0.. = Reset: Not set

... ..1. = Syn: Set

[Expert Info (Chat/Sequence): Connection establish request (SYN): server port 80]

[Connection establish request (SYN): server port 80]

[Severity level: chat]

[Group: Sequence]

....0 = Fin: Not set

window size value: 64240

[Calculated window size: 64240]

Checksum: 0x07d0 [validation disabled]

urgent pointer: 0

Options: (8 bytes), Maximum segment size, No-Operation (NOP), No-Operation (NOP), SACK permitted

Maximum segment size: 1460 bytes

Kind: Maximum Segment Size (2)

Length: 4

MSS Value: 1460

No-Operation (NOP)

No-Operation (NOP)

TCP SACK Permitted Option: True

Kind: SACK Permitted (4)

Length: 2

这是客户端发给服务器的第一个数据包所以seq=0

序号seq=0

确认号ack=0

请求连接的特征

ACK=0

SYN=1

MSS=1460

允许选择确认

选项部分

8.7.1 TCP建立连接- TCP连接确认数据包

Frame 4: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0

Ethernet II, Src: Vmware_f4:11:93 (00:50:56:f4:11:93), Dst: Vmware_14:bf:02 (00:0c:29:14:bf:02)

Internet Protocol Version 4, Src: 59.46.80.160 (59.46.80.160), Dst: 192.168.80.100 (192.168.80.100)

Transmission Control Protocol, Src Port: 80 (80), Dst Port: 1033 (1033), Seq: 0, Ack: 1, Len: 0

Source Port: 80 (80)

Destination Port: 1033 (1033)

[Stream index: 0]

[TCP segment Len: 0]

Sequence number: 0 ← (relative sequence number)

Acknowledgment number: 1 ← (relative ack number)

Header Length: 24 bytes

.... 0000 0001 0010 = Flags: 0x012 (SYN, ACK)

000. = Reserved: Not set

...0 = Nonce: Not set

.... 0... = Congestion Window Reduced (CWR): Not set

.... .0.. = ECN-Echo: Not set

.... ..0. = Urgent: Not set

.... ...1 = Acknowledgment: Set

.... 0... = Push: Not set

....0.. = Reset: Not set

....1. = Syn: Set

[Expert Info (Chat/Sequence): Connection establish acknowledge (SYN+ACK): server port 80]

[Connection establish acknowledge (SYN+ACK): server port 80]

[Severity level: Chat]

[Group: Sequence]

....0 = Fin: Not set

window size value: 64240

[Calculated window size: 64240]

Checksum: 0x3ff4 [validation disabled]

Urgent pointer: 0

Options: (4 bytes), Maximum segment size

Maximum segment size: 1460 bytes

Kind: Maximum Segment Size (2)

Length: 4

MSS value: 1460

[SEQ/ACK analysis]

序号seq=0

确认号ack=1

确认连接的特征

ACK=1

SYN=1

服务器支持的MSS=1460

这是服务器发给客户机的第一个数据包所以seq=0

服务器收到连接请求seq=0, 现发送确认号为0+1

选项部分

8.7.1 TCP建立连接-确认的确认

Frame 5: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0

Ethernet II, Src: Vmware_14:bf:02 (00:0c:29:14:bf:02), Dst: Vmware_f4:11:93 (00:50:56:f4:11:93)

Internet Protocol Version 4, Src: 192.168.80.100 (192.168.80.100), Dst: 59.46.80.160 (59.46.80.160)

Transmission Control Protocol, Src Port: 1033 (1033), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 0

Source Port: 1033 (1033)

Destination Port: 80 (80)

[Stream index: 0]

[TCP Segment Len: 0]

Sequence number: 1 (relative sequence number)

Acknowledgment number: 1 (relative ack number)

Header Length: 20 bytes

.... 0000 0001 0000 = Flags: 0x010 (ACK)

000. = Reserved: Not set

...0 = Nonce: Not set

.... 0... = Congestion window Reduced (CWR): Not set

.... .0.. = ECN-Echo: Not set

.... ..0. = Urgent: Not set

.... ...1 = Acknowledgment: Set

.... 0... = Push: Not set

....0.. = Reset: Not set

....0. = Syn: Not set

....0 = Fin: Not set

window size value: 64240

[calculated window size: 64240]

[window size scaling factor: -2 (no window scaling used)]

Checksum: 0x9cf5 [validation disabled]

Urgent pointer: 0

[SEQ/ACK analysis]

序号seq=1

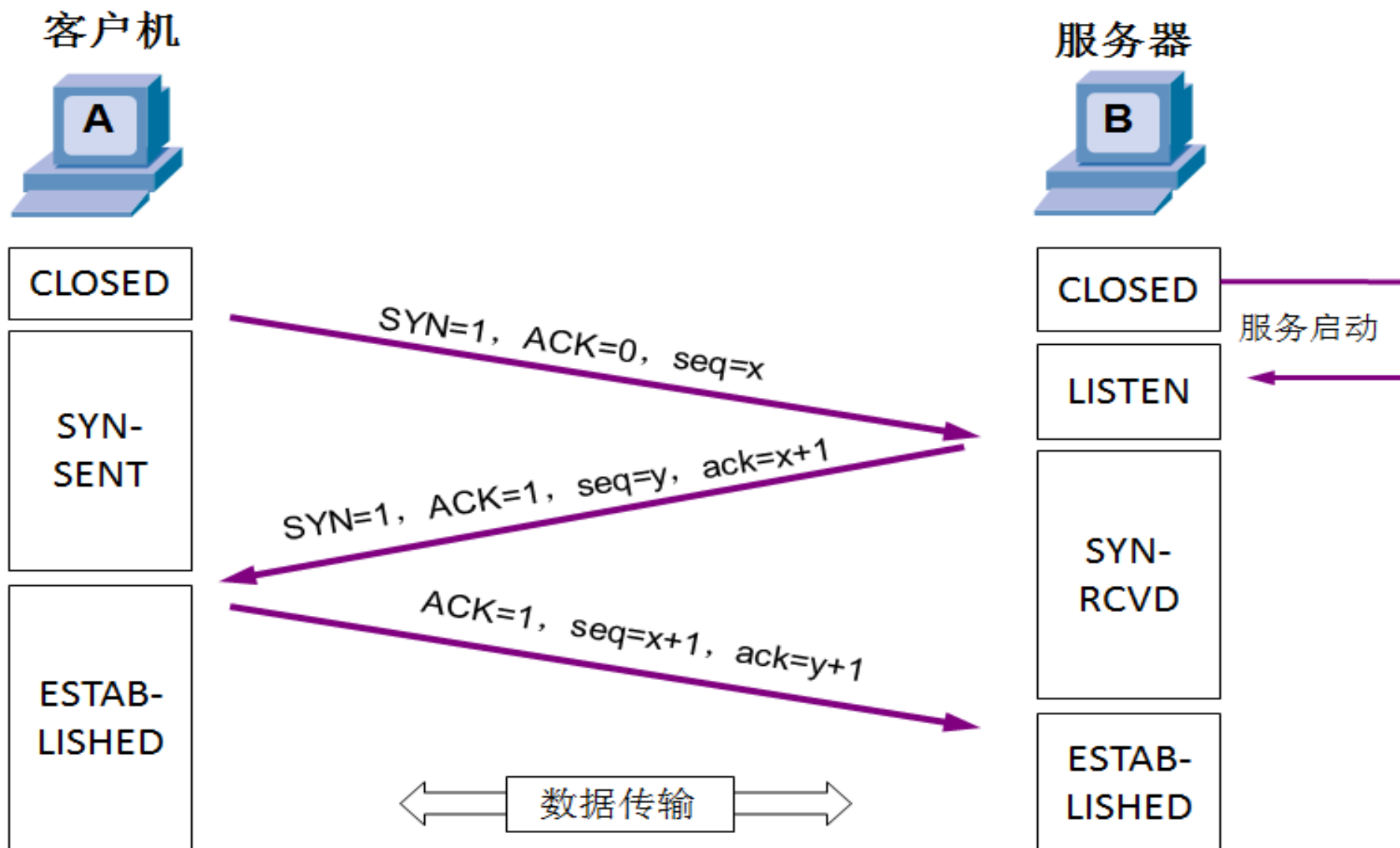
确认号ack=1

ACK=1

这是客户机给确认的一个确认seq=1

客户机收到确认连接seq=0, 现发送确认号为0+1

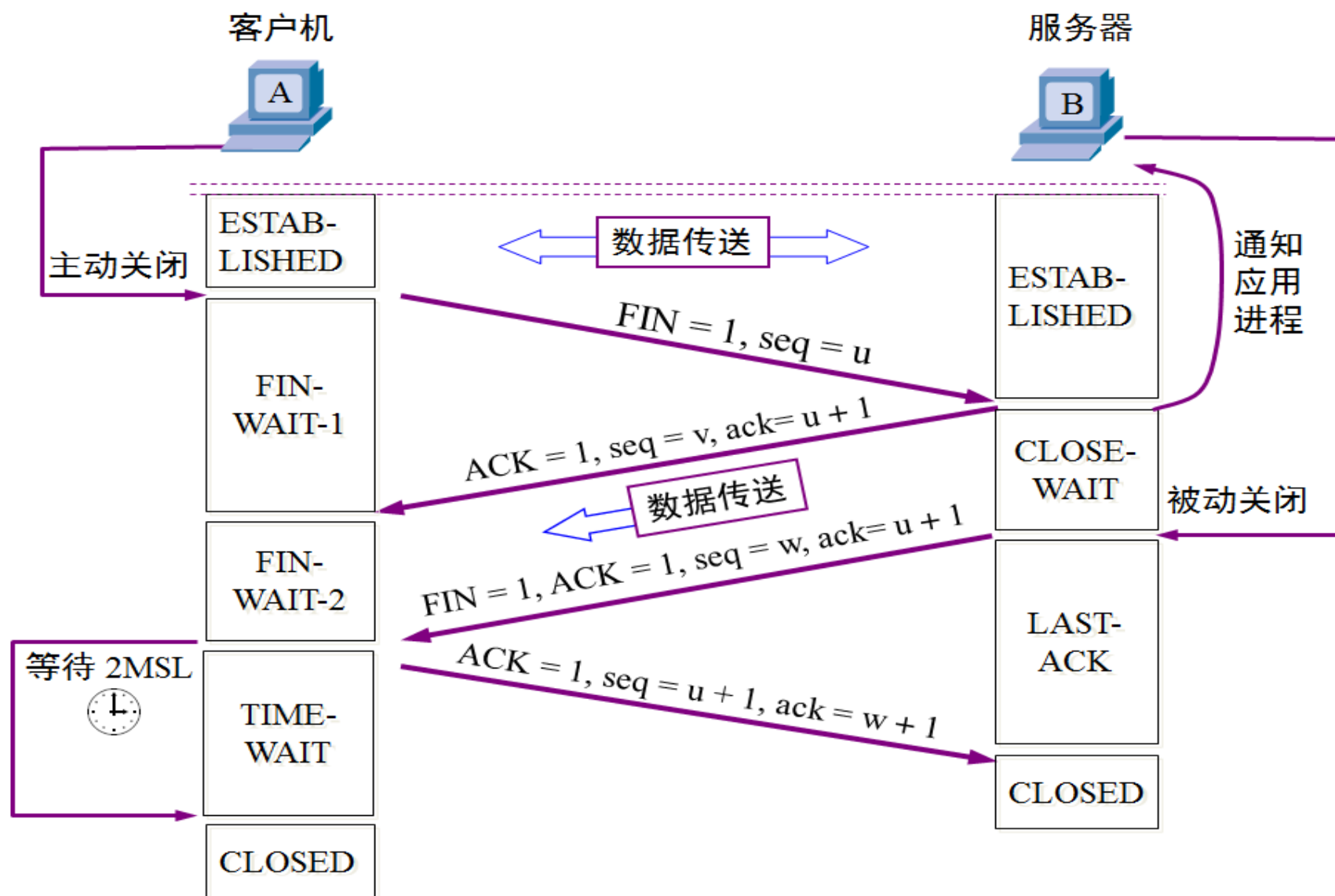
8.7.1 TCP的连接建立的过程



8.7.2 TCP连接释放

- TCP协议通信结束后，需要释放连接，TCP连接释放过程比较复杂，我们仍结合双方状态的改变来阐明连接释放的过程。
- 数据传输结束后，通信的双方都可释放连接。如图8-74所示，现在A和B都处于ESTABLISHED状态，A的应用进程先向其TCP发出连接释放报文段，并停止再发送数据，主动关闭TCP连接。A把连接释放报文段首部的FIN置1，其序号seq=u，它等于前面已传送过的数据的最后一个字节的序号加1。这时A进入FIN-WAIT-1（终止等待1）状态，等待B的确认。

8.7.2 TCP连接释放



8.7.3实战：查看TCP释放连接的数据包1

释放连接的
4个数据包

Wireshark 1.12.4 (v1.12.4-0-gb4861da from master-1.12)

Filter: Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
8351	7.922817000	192.168.80.100	192.168.80.111	TCP	54	1649→21 [ACK] Seq=212 Ack=883 win=63332 Len=0
8352	7.924032000	192.168.80.111	192.168.80.100	FTP	78	Response: 226 Transfer complete.
8353	8.142919000	192.168.80.100	192.168.80.111	TCP	54	1649→21 [ACK] Seq=212 Ack=909 win=63332 Len=0
8354	15.392052000	192.168.80.100	192.168.80.111	TCP	54	1649→21 [FIN, ACK] Seq=212 Ack=909 win=63332 Len=0
8355	15.396766000	192.168.80.111	192.168.80.100	TCP	60	21→1649 [ACK] Seq=909 Ack=213 win=64029 Len=0
8356	15.396796000	192.168.80.111	192.168.80.100	TCP	60	21→1649 [FIN, ACK] Seq=909 Ack=213 win=64029 Len=0
8357	15.396820000	192.168.80.100	192.168.80.111	TCP	54	1649→21 [ACK] Seq=213 Ack=910 win=63332 Len=0

... 0000 0001 0001 = Flags: 0x011 (FIN, ACK)

- 000. = Reserved: Not set
- ...0 = Nonce: Not set
- 0... = Congestion window Reduced (CWR): Not set
-0.. = ECN-Echo: Not set
-0. = Urgent: Not set
-1 = Acknowledgment: Set
- 0... = Push: Not set
-0.. = Reset: Not set
-0. = Syn: Not set
-1 = Fin: Set**

window size value: 63332

0000 00 0c 29 b8 3c 9c 00 0c 29 14 bf 02 08 00 45 00 ..).<...).....E.
0010 00 28 3d 33 40 00 80 06 9b 78 c0 a8 50 64 c0 a8 .(=3@... .x..Pd..
0020 50 6f 06 71 00 15 84 2c 99 ac 95 36 4d ae 50 11 P0,q...,...6M.P.
0030 f7 64 22 3f 00 00 .d"?..

Flags (tcp.flags), 2 bytes

Packets: 8358 · Displayed: 8358 (100.0%) · Dropped: 0 (0.0%)

Profile: Default

8.7.3实战：查看TCP释放连接的数据包2

```
C:\Windows\system32\cmd.exe

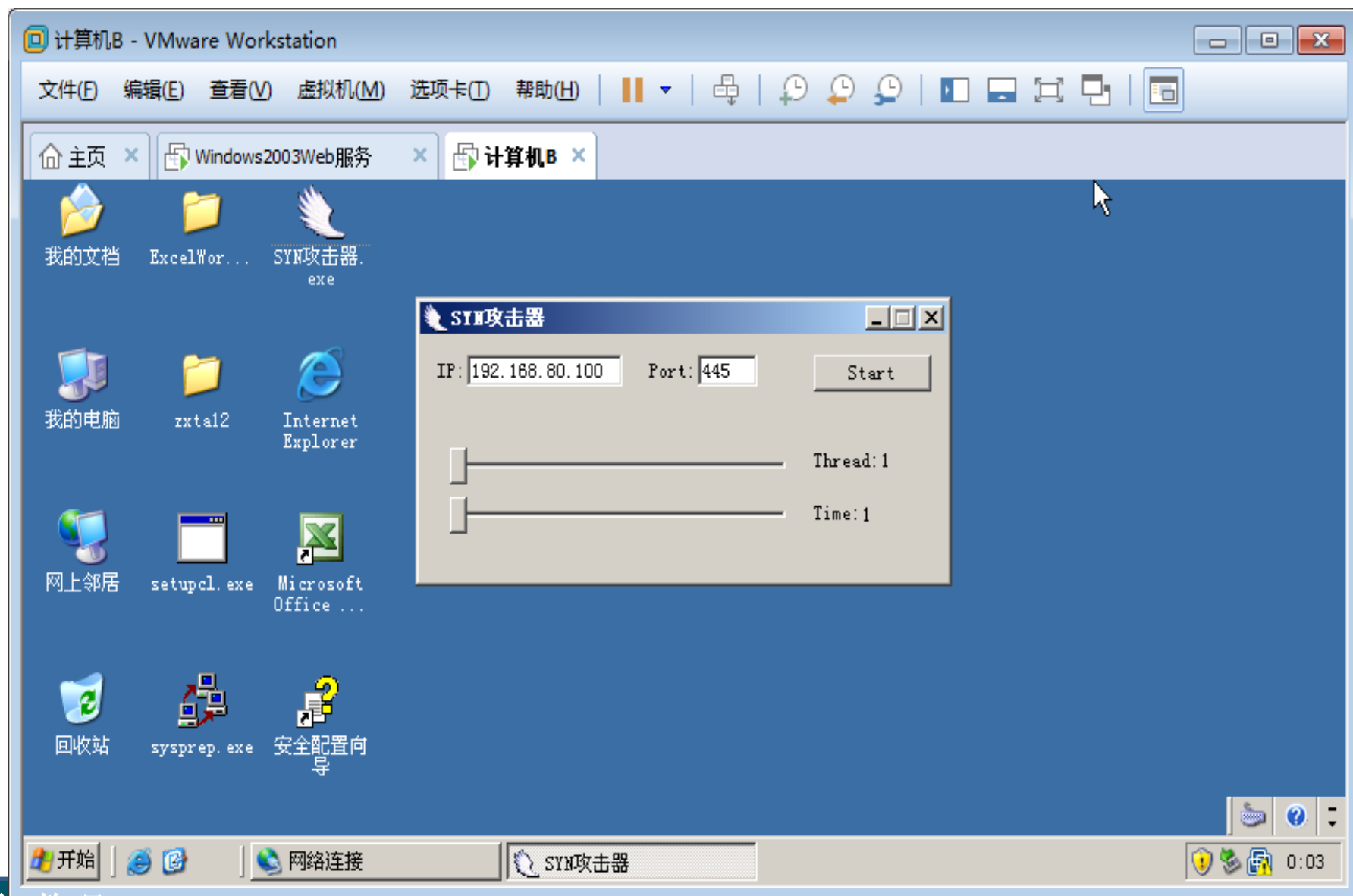
C:\Users\han>netstat -n

活动连接

  协议    本地地址           外部地址           状态
  TCP     127.0.0.1:1027      127.0.0.1:1028      ESTABLISHED
  TCP     127.0.0.1:1028      127.0.0.1:1027      ESTABLISHED
  TCP     127.0.0.1:1047      127.0.0.1:1048      ESTABLISHED
  TCP     127.0.0.1:1048      127.0.0.1:1047      ESTABLISHED
  TCP     127.0.0.1:4510      127.0.0.1:4511      ESTABLISHED
  TCP     127.0.0.1:4511      127.0.0.1:4510      ESTABLISHED
  TCP     127.0.0.1:4530      127.0.0.1:4531      ESTABLISHED
  TCP     127.0.0.1:4531      127.0.0.1:4530      ESTABLISHED
  TCP     127.0.0.1:4533      127.0.0.1:4534      ESTABLISHED
  TCP     127.0.0.1:4534      127.0.0.1:4533      ESTABLISHED
  TCP     192.168.0.103:4532  23.220.173.184:443  ESTABLISHED → 已建立连接
  TCP     192.168.0.103:6588  106.120.166.72:80   ESTABLISHED
  TCP     192.168.0.103:6785  220.181.132.153:80  ESTABLISHED
  TCP     192.168.0.103:6944  14.17.42.38:80      TIME_WAIT → 时间等待
  TCP     192.168.0.103:6951  123.151.41.219:80   ESTABLISHED
  TCP     192.168.0.103:6953  106.38.181.143:80   CLOSE_WAIT → 关闭等待
  TCP     192.168.0.103:6965  14.17.42.38:80      TIME_WAIT
  TCP     192.168.0.103:6972  74.125.23.139:443   SYN_SENT → 发送了连接请求
  TCP     192.168.0.103:6974  180.149.144.107:80  ESTABLISHED
  TCP     192.168.0.103:6976  106.120.163.52:80   TIME_WAIT
  TCP     192.168.0.103:6977  220.181.163.138:80  ESTABLISHED
  TCP     192.168.0.103:6983  183.57.48.18:80     ESTABLISHED

C:\Users\han>
半:
```

8.7.4实战：SYN攻击1



8.7.4实战：SYN攻击2

Capturing from 本地连接 [Wireshark 1.12.4 (v1.12.4-0-gb4861da from master-1.12)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
325357	8.620168000	111.143.78.157	192.168.80.100	TCP	78	[TCP Retransmission] 12082+445 [SYN] Seq=0 win=65535
325358	8.620172000	89.80.7.71	192.168.80.100	TCP	78	[TCP Retransmission] 26126+445 [SYN] Seq=0 win=65535
325359	8.620175000	133.113.214.43	192.168.80.100	TCP	78	[TCP Retransmission] 36774+445 [SYN] Seq=0 win=65535
325360	8.620210000	238.123.122.118	192.168.80.100	TCP	78	[TCP Retransmission] 18166+445 [SYN] Seq=0 win=65535
325361	8.620213000	228.109.161.123	192.168.80.100	TCP	78	[TCP Retransmission] 41760+445 [SYN] Seq=0 win=65535
325362	8.620216000	124.151.12.189	192.168.80.100	TCP	78	[TCP Retransmission] 14097+445 [SYN] Seq=0 win=65535
325363	8.620220000	115.144.58.26	192.168.80.100	TCP	78	[TCP Retransmission] 30383+445 [SYN] Seq=0 win=65535
325364	8.620224000	149.175.145.86	192.168.80.100	TCP	78	[TCP Retransmission] 43386+445 [SYN] Seq=0 win=65535
325365	8.620226000	172.107.5.157	192.168.80.100	TCP	78	[TCP Retransmission] 37562+445 [SYN] Seq=0 win=65535

.... 0000 0000 0010 = Flags: 0x002 (SYN)

- 000. = Reserved: Not set
- 0 = Nonce: Not set
- 0... = Congestion Window Reduced (CWR): Not set
-0.. = ECN-Echo: Not set
-0. = Urgent: Not set
-0 = Acknowledgment: Not set
- 0... = Push: Not set
-0.. = Reset: Not set
- +1. = Syn: Set
-0 = Fin: Not set

Window size value: 65535

0000 00 0c 29 14 bf 02 00 0c 29 b8 3c 9c 08 00 45 00 ..).).<...E.
0010 00 40 a3 2f 40 00 40 06 d4 73 ac 6b 05 9d c0 a8 .@./@.@. .s.k....
0020 50 64 92 ba 01 bd 61 9a 75 81 00 00 00 00 b0 02 Pd....a. u.....
0030 ff ff 07 6d 00 00 02 04 05 a0 01 03 03 00 01 01 ...m....
0040 08 0a 00 00 00 00 00 00 00 00 01 01 04 02

本地连接: <live capture in progress> File: ... Packets: 873128 · Displayed: 873128 (100.0%) Profile: Default