The Transport Layer

Computer Networking, A Top-Down Approach, 5th Edition

任志诚 2023212020

2025-06-08 15:04:40

Contents

End-of-chapter exercises **2**

R.4 2
P.4 2
P.11 3
P.14 3
P.22 4
P.24 4
P.25 5
P.31 7
P.32 7
P.34 8
P.37 9
P.43 11

End-of-chapter exercises

R.4

Question:

Describe why an application developer might choose to run an application over UDP rather than TCP.

Answer:

An application developer might choose to run an application over UDP rather than TCP for the following reasons:

1. **Low Latency**: UDP is a connectionless protocol, meaning it does not establish a connection before sending data. This reduces latency, making it suitable for real-time applications like video streaming, online gaming, and voice over IP (VoIP).
2. **No Overhead for Reliability**: Unlike TCP, UDP does not provide reliability mechanisms such as retransmissions, acknowledgments, or congestion control. This reduces overhead and allows applications to handle reliability themselves if needed.
3. **Broadcast/Multicast Support**: UDP supports broadcasting and multicasting, which is useful for applications like live video streaming or network discovery.
4. **Simple Protocol**: UDP is simpler and requires fewer resources, making it ideal for lightweight applications or systems with limited processing power.
5. **Custom Error Handling**: Some applications prefer to implement their own error handling and flow control mechanisms tailored to their specific needs, which is easier with UDP.
6. **Unordered Data Delivery**: For applications where the order of data packets is not critical (e.g., DNS queries), UDP is a better choice as it does not enforce packet ordering.

P.4

Question: a. Suppose you have the following 2 bytes: 01011100 and 01010110. What is the 1s complement of the sum of these 2 bytes? b. Suppose you have the following 2 bytes: 11011010 and 00110110. What is the 1s complement of the sum of these 2 bytes? c. For the bytes in part

(a), give an example where one bit is flipped in each of the 2 bytes and yet the 1s complement doesn’t change.

Answer: - **a.** 10110010. - **b.** 00010001. - **c.** 01011110 and 01010100.

P.11

Question:

The sender side of rdt3.0 simply ignores (that is, takes no action on) all received packets that are either in error or have the wrong value in the acknum field of an acknowledgment packet. Suppose that in such circumstances, rdt3.0 were simply to retransmit the current data packet. Would the protocol still work? (Hint: Consider what would happen if there were only bit errors; there are no packet losses but premature timeouts can occur. Consider how many times the n^{th} packet is sent, in the limit as n approaches infinity.)

Answer:

在这种情况下，rdt 3.0 仍然是有效的。首先，该协议接收方可以通过检查 **sequence number** 来判断当前的包是否重复，这个检查方式一直有效。原本的 rdt3.0 协议在超时后会重传，但现在只要检查出 **acknowledgment packet** 中出错就重传，这显然会增加网络负载，比如：在极端情况下，发送方可能会多次重传一个包，若 ACK 错误率为 p，则每个数据包平均需要发送 $1/(1-p)$ 次，直到接受到正确的 **ACKs**。总结：rdt3.0 协议在这种情况下仍然是可靠的，但立即重传错误数据包会导致性能问题，因此这种修改并不是一个好的设计选择。

P.14

Question:

Consider the cross-country example shown in Figure 3.17. How big would the window size have to be for the channel utilization to be greater than 95 percent? Suppose that the size of a packet is $1,500$ bytes, including both header fields and data.

Complement: 1 Gbps link. RTT, is approximately 30 milliseconds.

Answer:

$$\mathrm{d}_{trans} = \frac{L}{R} = \frac{1500 \times 8}{1 \times 10^9} = 1.2 \times 10^{-5}\,\mathrm{s}$$

$$\text{Utilization} = \frac{\text{rwnd} \times \frac{L}{R}}{\text{RTT} + \text{rwnd} \times \frac{L}{R}} \geq 0.95$$

解得，$\text{rwnd} \geq 4750$。

P.22

Questions and Answers:

Answer true or false to the following questions and briefly justify your answer:

a. With the SR protocol, it is possible for the sender to receive an ACK for a packet that falls outside of its current window.

True; 在 **SR** 协议中，由于网络延迟等原因，发送方可能收到对应于已经滑出当前窗口的数据包的 ACK。

b. With GBN, it is possible for the sender to receive an ACK for a packet that falls outside of its current window.

False; 在 **GBN** 协议中，**ACK** 是累积的，表示所有序列号小于等于 ACK 的包已经被正确接收。

c. The alternating-bit protocol is the same as the SR protocol with a sender and receiver window size of 1**.**

True; 交替位协议是 SR 协议的一个特例，其中发送方和接收方的窗口大小均为 1。它只允许发送一个未确认的包，并通过序列号（0 或 1）来区分包。

d. The alternating-bit protocol is the same as the GBN protocol with a sender and receiver window size of 1**.**

True; 交替位协议也可以看作是 GBN 协议的一个特例，其中发送方和接收方的窗口大小均为 1。在这种情况下，GBN 的行为与交替位协议完全一致，因为它只允许发送一个未确认的包，并在超时或收到 NAK 时重传该包。

P.24

Questions and Answers:

Consider transferring an enormous file of L bytes from Host A to Host B. Assume an MSS of 536 bytes.

a. What is the maximum value of L such that TCP sequence numbers are not exhausted? Recall that the TCP sequence number field has 4 bytes.

TCP segment structure 中 sequence number 有 32 bits，从 0 开始。所以最大的 $L = 2^{32} - 1$。

b. For the L you obtain in (a), find how long it takes to transmit the file. Assume that a total of 66 bytes of transport, network, and data-link header are added to each segment before the resulting packet is sent out over a 155 Mbps link. Ignore flow control and congestion control so A can pump out the segments back to back and continuously.

- MSS = 536 bytes
- 每报文附加头部 = 66 bytes
- 总报文尺寸 (536 + 66 = 602) bytes
- 报文数量

$$N = \left\lceil \frac{L_{\text{max}}}{\text{MSS}} \right\rceil = \left\lceil \frac{2^{32} - 1}{536} \right\rceil = 8\,012\,999$$

- 总传输位数

$$B = N \times 602 \text{ bytes} \times 8 \, \tfrac{\text{bits}}{\text{byte}} = 8\,012\,999 \times 602 \times 8 \approx 3.85906 \times 10^{10} \text{ bits}$$

- 链路速率 (R = 155 Mbps)
- 传输时间

$$T = \frac{B}{R} = \frac{3.85906 \times 10^{10}}{155 \times 10^{6}} \approx 249 \text{ 秒} \approx 4.15 \text{ 分钟}$$

P.25

Host A and B are communicating over a TCP connection, and Host B has already received from A all bytes up through byte 126. Suppose Host A then sends two segments to Host B back-to-back. The first and second segments contain 70 and 50 bytes of data, respectively. In the first segment, the sequence number is 127, the source port number is 302, and the destination port number is 80. Host B sends an acknowledgement whenever it receives a segment from Host

A.

Questions and Answers:

a. In the second segment sent from Host A to B, what are the sequence number, source port number, and destination port number? - The sequence number is 197, the source port number is 302 and the dest port number is 80.

b. If the first segment arrives before the second segment, in the acknowledgement of the first arriving segment, what is the acknowledgment number, the source port number, and the destination port number? - The acknowledgement number is 197, the source port number is 80 and the dest port number is 302.

c. If the second segment arrives before the first segment, in the acknowledgement of the first arriving segment, what is the acknowledgment number? - The acknowledgement number is 127.

d. Suppose the two segments sent by A arrive in order at B. The first acknowledgement is lost and the second acknowledgement arrives after the first timeout interval. Draw a timing diagram, showing these segments and all other segments and acknowledgements sent. (Assume there is no additional packet loss.) For each segment in your figure, provide the sequence number and the number of bytes of data; for each acknowledgement that you add, provide the acknowledgement number.

- 假设：
- A→B 连续发送两段
- B 收到每段即发 ACK
- 第一条 ACK(197) 丢失，第二条 ACK(247) 成功到达
- 超时重传未触发（因为第二条 ACK 收到后 A 已完成确认）

```
1
2  Time →
3  Host A                              Host B
4    |                                   |
5    |-- [Seq=127, Len=70] ---------->|   接收 127-196，B 发送 ACK
       (197) → 丢失
```

```
6    |                                    |
7    |-- [Seq=197, Len=50] ---------->|   接收 197-246，B 发送 ACK
        (247)
8    |                                    |
9    |       [超时等待 ACK(197)]          |
10   |                                    |
11   |-- [Seq=127, Len=70] ---------->|   重复包，B 再次发送 ACK(247)
12   |                                    |
13   |<----------- ACK(247) ----------|   A 收到确认，确认两段都已接收
```

P.31

Question and Answer:

What is the relationship between the variable SendBase in Section 3.5.4 and the variable LastByteRcvd in Section 3.5.5?

在理想网络条件下（无丢包、延迟较小）：

$$\text{SendBase} - 1 = \text{LastByteRcvd}$$

这是因为：- 接收方接收到字节序号为 n 的数据后，LastByteRcvd = n - 接收方发送 ACK(n+1) - 发送方收到 ACK(n+1) 后，更新 SendBase = n+1

由于网络延迟、丢包等因素：- LastByteRcvd 可能小于 SendBase-1（发送方已收到更新的确认）- LastByteRcvd 可能大于 SendBase-1（某些已接收数据未被确认接收）

P.32

Question and Answer:

What is the relationship between the variable LastByteRcvd in Section 3.5.5 and the variable y in Section 3.5.4?

由上题所述：

$$\text{acknowledgement number} = \text{LastByteRcvd} + 1$$

P.34

Compare GBN, SR, and TCP (no delayed ACK). Assume that the timeout values for all three protocols are sufficiently long such that 5 consecutive data segments and their corresponding ACKs can be received (if not lost in the channel) by the receiving host (Host B) and the sending host (Host A) respectively. Suppose Host A sends 5 data segments to Host B, and the 2^{nd} segment (sent from A) is lost. In the end, all 5 data segments have been correctly received by Host B.

Question and Answer:

a. How many segments has Host A sent in total and how many ACKs has Host B sent in total? What are their sequence numbers? Answer this question for all three protocols.

题目没有指定 sequence number，这里默认从 0 开始，所有 segment 大小为 1 byte。首先，receiver 收到多少个 segment 就会回多少个 ACK。

首先明确各协议的 ACK 机制：

协议	确认机制	**ACK** 含义
GBN	累积确认	ACK(n) 表示期望收到序号 n 的段
SR	选择确认	ACK(n) 表示已收到序号 n 的段
TCP	累积确认 +SACK	ACK(n) 表示期望收到序号 n 的段

分析 GBN 协议，第一个 segment 的 sequence number 为 0，以此类推，接下来的 sequence number 分别为 1 2 3 4，相应的 ACK 为 1 2 3 4 5。但由于第二个包丢失，实际上的 ACK 为 1 丢失 1 1 1，再重发，2 3 4 5。再分析剩下两个协议，最终得到

Protocol	**Segments Sent**	**ACKs Received**	**Their Sequence**
GBN	9	8	1 1 1 1 2 3 4 5
SR	6	5	0 2 3 4 1
TCP	6	5	1 1 1 1 5

b. If the timeout values for all three protocol are much longer than 5 RTT, then which protocol successfully delivers all five data segments in shortest time interval?

SR 和 TCP，因为这两种协议都只发送了 6 次包。

P.37

Questions and Answers:

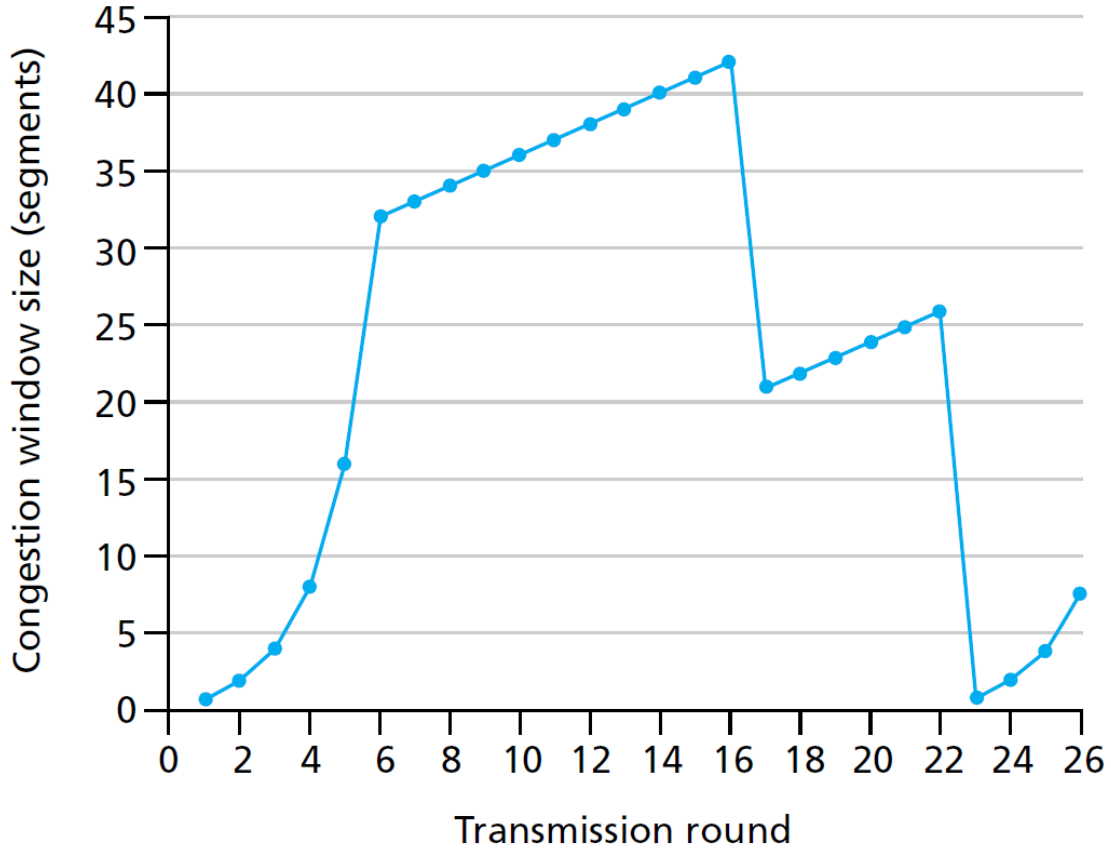

Figure 1: TCP window size as a function of time

Assuming TCP **Reno** is the protocol experiencing the behavior shown above, answer the following questions. In all cases, you should provide a short discussion justifying your answer.

a. Identify the intervals of time when TCP slow start is operating.

根据 TCP congestion control 机制，slow start phase 期间 **cwnd** 是以指数函数增长，从图中可以得出大概是在 Transmission round $\in [1, 6] \cup [23, 26]$ 期间。

b. Identify the intervals of time when TCP congestion avoidance is operating

TCP congestion avoidance 期间的特点是 **cwnd** 以 $ k = 1 $ MSS 的速率线性增长，从图中可以得出大概是在 Transmission round $\in [6, 16] \cup [17, 22]$ 期间。

c. After the 16th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?

从图中可以分析出，ssthresh $= \frac{\text{cwnd}}{2}$，cwnd $=$ ssthresh，然后线性增长，所以 detected by a triple duplicate ACK。

d. After the 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?

cwnd 归一，进入 slow start phase，所以 detected by a timeout。

e. What is the initial value of ssthresh at the first transmission round?

大概是 32 MSS，当 cwnd $\geq$ ssthresh 会进入 congestion avoidance phase，然后线性增长。

f. What is the value of ssthresh at the 18^{th} transmission round?

减半了，所以是，ssthresh $= \frac{\text{cwnd}}{2} = 21$ MSS

g. What is the value of ssthresh at the 24^{th} transmission round?

减半了，所以是，ssthresh $= \frac{\text{cwnd}}{2} = 13$ MSS

h. During what transmission round is the 70^{th} segment sent?

第一次 slow start phase 阶段后传了 63 个，所在第 6 和 7 轮之间。

i. Assuming a packet loss is detected after the 26^{th} round by the receipt of a triple duplicate ACK, what will be the values of the congestion window size and of ssthresh?

ssthresh $= \frac{\text{cwnd}}{2}$，cwnd $=$ ssthresh，都是 4 segment.

j. Suppose TCP Tahoe is used (instead of TCP Reno), and assume that triple duplicate ACKs are received at the 16^{th} round. What are the ssthresh and the congestion window size at the 19^{th} round?

特性	TCP Tahoe	TCP Reno
三次重复 **ACK** 时	cwnd=1MSS, 进入慢启动	cwnd=ssthresh, 进入拥塞避免
超时事件时	cwnd=1MSS, 进入慢启动	cwnd=1MSS, 进入慢启动
恢复速度	较慢 (需要重新慢启动)	较快 (可能直接进入拥塞避免)

特性	TCP Tahoe	TCP Reno
快速恢复机制	不支持	支持

第 16^{th} 出现了 triple duplicate ACKs，所以，第 17^{th} ssthresh $= \frac{\text{cwnd}}{2} = 21$，cwnd $= 1$，然后进入 slow start 阶段，第 19^{th} cwnd $= 4$

k. Again suppose TCP Tahoe is used, and there is a timeout event at 22^{nd} round. How many packets have been sent out from 17^{th} round till 22^{nd} round, inclusive?

需要注意的是，在第 16^{th} 收到了三个 ACKs，同时，在第 22^{th} 到达 ssthresh 所以实际收到 $1 + 2 + 4 + 8 + 16 + 21 = 52$ MSS。

P.43

Consider that only a single TCP (Reno) connection uses one 10 Mbps link which does not buffer any data. Suppose that this link is the only congested link between the sending and receiving hosts. Assume that the TCP sender has a huge file to send to the receiver, and the receiver's receive buffer is much larger than the congestion window. We also make the following assumptions: each TCP segment size is 1 500 bytes; the two-way propagation delay of this connection is 100 msec; and this TCP connection **is always in congestion avoidance phase, that is, ignore slow start**.

Questions and Answers:

a. What is the maximum window size (in segments) that this TCP connection can achieve?

最大窗口大小由 带宽延迟积 **(BDP)** 决定：

首先计算 RTT：- 传播延迟 = 100 毫秒 - 传输延迟 = 段大小/带宽 = (1500×8) 比特/(10×10^6) 比特/秒 = 1.2 毫秒 - 总 **RTT** = 101.2 毫秒

带宽延迟积：

$$\text{BDP} = 10 \text{ Mbps} \times 0.1012 \text{ s} = 1.012 \times 10^6 \text{ bits}$$

也就是 PPT 上的公式：

$$\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}}$$

最大窗口大小 (段数)：

$$\text{最大窗口} = \frac{\text{BDP}}{\text{段大小}} = \frac{1.012 \times 10^6}{1500 \times 8} \approx 84.33 = \boxed{84 \text{ 段}}$$

b. What is the average window size (in segments) and average throughput (in bps) of this TCP connection?

根据 TCP Reno 特性，窗口大小在拥塞避免阶段会在 $\frac{W}{2}$ 和 W 之间周期变化。

平均窗口大小：

$$\text{平均窗口} = 0.75 \times W = 0.75 \times 84 = \boxed{63 \text{ 段}}$$

平均吞吐量：

$$\text{平均吞吐量} = 0.75 \times \frac{W \times \text{段大小} \times 8}{\text{RTT}} = 0.75 \times \frac{84 \times 1500 \times 8}{0.1012} \approx \boxed{7.5 \text{ Mbps}}$$

c. How long would it take for this TCP connection to reach its maximum window again after recovering from a packet loss?

在 TCP Reno 中，丢包后窗口大小减半，然后每个 RTT 增加 1 个 MSS：

- 初始窗口（丢包后）= $\frac{W}{2} = 42$ 段
- 每个 RTT 增加 1 段
- 需要增加的段数 = $\frac{W}{2} = 42$ 段
- 所需 RTT 数 = 42

$$\text{恢复时间} = 42 \times 0.1012 = \boxed{4.25 \text{ 秒}}$$