Tcp的相关知识梳理

一、TCP 头部的详细格式

Field	Length (bits)	Length (bytes)	Description
Source Port	16	2	发送端口号
Destination Port	16	2	接收端口号
Sequence Number	32	4	数据流的序列号
Acknowledgment Number	32	4	确认号
Data Offset	4	0.5	头部长度
Reserved	3	0.375	保留字段
Flags	9	1.125	标志位（URG, ACK, PSH, RST, SYN, FIN）
Window Size	16	2	接收窗口大小
Checksum	16	2	校验和
Urgent Pointer	16	2	紧急指针
Options	Variable	0-40	选项字段（如最大报文段大小）
Data	Variable	0 - (Maximum size)	实际数据

标志位（Flags） - 9 位 **具体含义**：

- **URG**：如果该位为 1，表示紧急指针字段有效，接收方应优先处理该数据包。
- **ACK**：如果该位为 1，表示确认号字段有效。
- **PSH**：如果该位为 1，表示数据应立即传递给应用层。
- **RST**：如果该位为 1，表示连接重置，通常用于关闭连接或发生错误时。
- **SYN**：用于建立连接时的同步序列号。
- **FIN**：用于表示发送方的数据已经发送完毕，可以关闭连接。

二、tcp的seq/ack作用

在 TCP 协议中，**SEQ（Sequence Number）** 和 **ACK（Acknowledgment Number）** 是两个非常重要的字段，它们用于数据包的顺序控制、数据完整性保障以及流量控制。它们在 TCP 连接的建立、数据传输和关闭过程中起着关键作用。

1. SEQ（Sequence Number）作用及计算

SEQ 代表一个 TCP 数据流中的字节流的顺序号。每一个 TCP 数据包（Segment）都携带一个 `SEQ` 字段，表示该数据包中的第一个字节在整个连接中的位置。

- **作用**：
  - 确保数据的顺序：通过 `SEQ` 号，接收端可以按照顺序重组数据包。
  - 处理数据丢失：如果某些数据包丢失，接收端可以通过 `SEQ` 号来请求重传。
  - 支持流量控制：在一些情况下，TCP 连接可以使用 `SEQ` 号来进行流量控制，例如接收方的缓冲区容量限制。
- **如何计算和增加**：
  - **连接建立时的初始化**：在连接建立时（即 TCP 三次握手期间），每一方会随机选择一个初始的 **序列号（ISN, Initial Sequence Number）**，并在后续的数据包中使用。
  - **数据包中的 SEQ 号**：`SEQ` 号代表该数据包中第一个字节的序列号。在发送数据时，`SEQ` 号会增加，每发送一个字节，`SEQ` 就会加 1。例如，假设初始的 SEQ 为 1000，发送的第一个数据包携带的 SEQ 为 1000，第一个字节的 `SEQ` 号就是 1000，接下来的字节依次是 1001、1002 等等。
- **什么时候增加**：
  - `SEQ` 号会随着每个数据包的发送而增加，具体增加的数值是数据包中有效负载的字节数。例如，如果数据包中包含 500 字节数据，那么 `SEQ` 号会增加 500。

2. ACK（Acknowledgment Number）作用及计算

ACK 代表接收方期望下一个数据字节的序列号。接收方通过这个字段告知发送方，自己已经成功接收到的字节流的最后一个字节的下一个字节的序列号。

- **作用**：
  - 确保可靠传输：通过 ACK 机制，发送方知道哪些数据已经成功接收到。如果接收方未能成功接收到某些数据包，它会通过丢失的 `SEQ` 号发起重传。
  - 流量控制和拥塞控制：ACK 包含的信息也可以帮助发送方了解接收方的接收能力，从而进行流量控制。
- **如何计算和增加**：
  - **初始化**：在连接建立时，接收方会发送一个带有初始 `SEQ` 号的 `SYN` 包，并在 ACK 中确认对方的 `SEQ` 号。此时接收方会将其 ACK 的 `ACK` 号设为对方发送的 `SEQ` + 1，表示接收方已经成功接收到该 `SEQ` 号的数据。
  - **每个数据包的 ACK 号**：当接收方收到数据包后，会将 `ACK` 号设置为下一个期望接收的字节的 `SEQ` 号。例如，发送方发送的数据包的 `SEQ` 是 1000-1500，那么接收方会返回一个 `ACK` 包，`ACK` 号就是 1501，表示接收方已经成功接收了从 1000 到 1500 的字节，并期望接收到下一个字节（即 1501）。
  - **累计确认**：TCP 使用 **累计确认（Cumulative Acknowledgment）** 机制，这意味着接收方通常只确认最后一个接收到的字节的下一个字节的 `SEQ` 号，而不需要每次确认每个字节。例如，如果接收方已经收到了字节 1000 到 2000，那么它只会返回 `ACK = 2001`。
- **什么时候增加**：
  - `ACK` 号在接收到数据包后增加。每当接收方成功接收数据并发回确认，它就会根据接收到的字节数增加 ACK 号。例如，如果接收方收到了 1000 字节的数据，且初始的 `ACK` 号为 1001（表示下一个期望的字节是 1001），那么当接收到 1000 字节后，`ACK` 号会增加为 2001。

3. TCP 序列号和确认号的工作流程

3.1. 三次握手（建立连接）

在 TCP 连接建立的三次握手过程中，`SEQ` 和 `ACK` 的使用如下：

1. **第一次握手**：客户端向服务器发送一个 SYN 包，选择一个初始的 `SEQ` 号（比如 1000）。此时，`SEQ` 为 1000，`ACK` 字段为空。
2. **第二次握手**：服务器响应客户端的 SYN 包，选择自己的初始 `SEQ` 号（比如 2000），并对客户端的 `SEQ` 号进行确认，即 `ACK` = 1001（表示客户端的第一个字节已经接收）。此时，服务器的 `SEQ` 为 2000，`ACK` 为 1001。
3. **第三次握手**：客户端收到服务器的 SYN-ACK 包后，向服务器发送 ACK 包，确认服务器的 `SEQ` 号。此时，客户端的 `SEQ` 号为 1001（客户端已经发送了一个字节），`ACK` 为 2001。

3.2. 数据传输（确认和序列号的变化）

在数据传输过程中，`SEQ` 和 `ACK` 的变化如下：

- 发送方发送数据时，`SEQ` 会逐渐增加。
- 接收方通过 `ACK` 告知发送方已收到的最后一个字节的 `SEQ`，并期望接收到下一个字节。
- 发送方根据接收到的 `ACK` 号，确认哪些数据已经被接收，从而决定是否进行重传或继续发送数据。

3.3. 连接关闭

当连接关闭时，TCP 使用四次挥手来断开连接。在关闭过程中，`SEQ` 和 `ACK` 仍然在工作，确保所有的数据都被成功传输并确认。

4. 总结

- **SEQ（序列号）**：用于标识 TCP 数据流中的字节，保证数据的顺序和完整性。每发送一个字节，`SEQ` 会增加。
- **ACK（确认号）**：用于确认接收到的数据，表示接收方下一个期望接收的字节的序列号。`ACK` 号会根据接收到的字节数逐渐增加。
- **增加方式**：`SEQ` 每发送一个字节增加，而 `ACK` 会根据接收到的数据更新，表示下一个期望接收的字节。

通过 `SEQ` 和 `ACK` 的交互，TCP 能够保证数据的可靠传输，确保数据按顺序到达并正确无误。

三、syn/ack标志对于seq/ack号有什么影响吗

在 TCP 协议中，**SYN** 和 **ACK** 标志对于 **SEQ** 和 **ACK** 号的设置有着重要影响，尤其是在连接建立和连接关闭过程中。我们可以通过解析三次握手和四次挥手过程，来看一下这两个标志如何影响 `SEQ` 和 `ACK` 号的变化。

1. SYN 标志

`SYN`（Synchronize）标志用于启动一个新的连接。在 TCP 连接的建立过程中，`SYN` 标志用于同步双方的序列号。以下是 `SYN` 对 `SEQ` 和 `ACK` 号的影响：

- **连接建立（SYN 过程）**：

- 当客户端向服务器发送 SYN 包时，它将选择一个 **初始序列号（ISN, Initial Sequence Number）**。例如，假设客户端选择的 ISN 为 `1000`。
- 客户端发送的 SYN 包的 `SEQ` 号是该初始序列号 `1000`，但是它的 `ACK` 字段是空的（因为这只是发起连接的一方，尚未收到对方的响应）。
- 服务器收到这个 SYN 包时，会：
  1. 选择一个 **初始序列号（ISN）**，例如 `2000`。
  2. 发送一个 **SYN-ACK** 包，表示它收到了客户端的连接请求，并同意建立连接。这个包的 `SEQ` 号是它选择的 ISN，即 `2000`，同时它会发送 `ACK` **号 = 客户端的** `SEQ` **+ 1**，即 `1001`，表示它已经成功接收到客户端的 SYN 包，并期望下一个数据包的 `SEQ` 号为 `1001`。
- 客户端收到服务器的 SYN-ACK 包后，会：
  1. 确认连接并返回一个 **ACK** 包，表示它收到了服务器的 SYN 包。
  2. 该 ACK 包的 `SEQ` **号** 仍然是客户端的初始 `SEQ` + 1（即 `1001`），并且它的 `ACK` **号** 为 `2001`，表示客户端已经收到服务器的 SYN 包，期望接收下一个字节的 `SEQ` 号为 `2001`。

- **总结**：
  `SYN` 标志在连接建立时用于初始化序列号，`SEQ` 号从 ISN 开始。在发送 SYN 包时，`ACK` 号未被设置；而在收到 SYN 包后，`ACK` 号被设置为对方的 `SEQ` + 1。

---

2. ACK 标志

`ACK` 标志用于确认数据包的接收。当接收方确认已经成功接收到数据后，会发送一个带有 **ACK** 标志的数据包，`ACK` 号指示接收方期望接收的下一个字节的 `SEQ` 号。

- **连接建立时的 ACK**：
  - 在 TCP 三次握手的第二次（服务器回应客户端的 SYN 包）和第三次（客户端回应服务器的 SYN-ACK 包）中，`ACK` **标志是必须的**。通过这个标志，客户端和服务器能够确认对方的序列号，确保连接的双方都已经收到对方的初始 `SEQ` 号。
  - 具体来说：
    - 在客户端发出的 SYN 包中，`SEQ` 号是客户端的 ISN，`ACK` 字段为空。
    - 在服务器的 SYN-ACK 包中，`SEQ` 号是服务器的 ISN，`ACK` 号是客户端的 `SEQ` + 1。
    - 在客户端的 ACK 包中，`SEQ` 号是客户端的 `SEQ` + 1，`ACK` 号是服务器的 `SEQ` + 1。
- **数据传输时的 ACK**：
  - 当数据传输开始后，发送方发送的数据包会包含一个 `SEQ` **号**，表示该数据包中的第一个字节的位置，接收方会通过 `ACK` 号来确认已经接收到的字节数。
  - 每次接收到一个数据包后，接收方会返回一个 `ACK` **包**，并将其 `ACK` **号设置为下一个期望接收的字节的** `SEQ` **号**。
  - 例如，假设客户端发送的数据包的 `SEQ` 是 1000，接收方成功接收到 1000 到 1500 字节的数据，那么它会返回一个 `ACK` 包，其中 `ACK` 号为 1501，表示接收方已经成功接收到字节 1000 到 1500，并期望接收下一个字节（即 1501）。
- **总结**：`ACK` 标志用于确认接收到的数据并告诉对方下一个期望的 `SEQ` 号。它是在连接建立时通过 `SYN-ACK` 来同步双方的 `SEQ` 号，在数据传输过程中，则通过返回带有 `ACK` 标志的数据包来确认接收的字节。

---

3. SYN 和 ACK 对 SEQ/ACK 号的具体影响

- **SYN**：
  - 在发送 SYN 包时，`SEQ` 号会被设置为初始序列号（ISN），此时没有有效的数据，因此 `ACK` 号为空。
  - 在响应 SYN 包时，`SEQ` 号设置为自己的 ISN，并且 `ACK` 号设置为接收到的 `SEQ` 号 + 1，表示确认对方的 SYN 包。
- **ACK**：
  - 在收到数据后，接收方通过返回 `ACK` 包确认已收到的字节，并将 `ACK` 号设置为下一个期望接收的字节的 `SEQ` 号。
  - 在 TCP 连接建立时，`ACK` 号用于确认对方的 `SEQ` 号，并且 `ACK` 号总是等于对方的 `SEQ` + 1。

4. 总结

- **SYN** 标志主要用于连接的初始化，影响 `SEQ` 号的设置。发送 SYN 包时，`SEQ` 为 ISN，且 `ACK` 号为空；接收 SYN 包时，`ACK` 号为收到的 `SEQ` 号 + 1。
- **ACK** 标志用于确认数据包的接收，`ACK` 号表示接收方期望接收的下一个字节的 `SEQ` 号。

在三次握手中，`SYN` 和 `ACK` 标志共同作用于 `SEQ` 和 `ACK` 号的交换，确保双方可以同步序列号并可靠地建立连接。而在数据传输过程中，`ACK` 机制保证数据的可靠传递，防止丢包和乱序问题。

四、什么是 TCP 拥塞?

TCP 拥塞是指由于网络中存在过多的数据流量，导致网络设备（如路由器、交换机）无法处理所有数据，进而引发的网络性能下降的现象。TCP 是一种 **面向连接** 的协议，负责在发送端和接收端之间可靠地传输数据。为了确保在网络出现拥堵时依然能够有效地传输数据，TCP 采用了 **拥塞控制**（Congestion Control）机制。

拥塞并不是由于接收方的能力不足（这由流量控制机制处理），而是由于 **网络中间节点（如路由器或交换机）** 的资源（如带宽、缓冲区）过载，无法有效处理通过它们的数据流。这种情况通常表现为网络延迟增加、数据包丢失、吞吐量下降等。

TCP 拥塞的原因

TCP 拥塞主要源于 **网络带宽不足** 或 **网络节点的缓冲区溢出。**以下是可能导致 TCP 拥塞的原因：

1. **带宽瓶颈**：网络的带宽资源有限。当多个流量竞争同一链路的带宽时，可能出现拥塞。
2. **路由器/交换机的缓冲区满**：网络中的路由器和交换机通常会缓存即将转发的数据包。如果这些设备的缓冲区被填满，它们将丢弃无法处理的数据包。
3. **流量突发**：某些应用可能在短时间内发送大量数据，超过了网络的承载能力，导致网络拥堵。
4. **带宽-延迟积**：高带宽和高延迟的组合（例如卫星通信、长距离网络连接）可能导致 **TCP 滞后**，从而导致拥塞窗口的调整不及时，增加了拥塞的可能性。

TCP 拥塞的影响

当发生拥塞时，网络中的数据包会遭遇丢失或延迟，这直接影响了数据传输的效率和可靠性。具体影响如下：

- **数据包丢失**：在拥塞的情况下，网络设备的缓冲区可能会溢出，导致数据包丢失。丢失的数据包会导致 **TCP 重传**，从而增加网络负担。

- **吞吐量下降**：当网络出现拥塞时，TCP 会降低发送速率，以避免进一步的丢包。这会导致应用程序的吞吐量下降，无法充分利用可用带宽。
- **延迟增加**：网络设备的缓冲区满时，数据包会被丢弃或延迟转发，导致传输延迟增加。这会影响延迟敏感的应用（如实时通信、视频流等）的性能。
- **连接的断开**：在极端的拥塞情况下，数据丢失频繁、重传超时，可能导致 TCP 连接超时或被强制关闭。

TCP 拥塞控制的目标

TCP 的 **拥塞控制** 机制的目标是 **动态调整发送速率**，以适应网络的当前状况，避免过度拥塞，并在网络恢复后尽可能提高传输速度。它通过以下几种方式来控制发送数据的速率：

- **避免发送过多的数据**：通过控制发送数据的速率，避免过多的数据同时进入网络，避免路由器或交换机的缓冲区溢出。
- **响应网络的变化**：根据网络的反馈动态调整发送速率。例如，如果发生数据包丢失（通常是由于拥塞引起的），TCP 会减少发送速率，反之如果网络状况变好，TCP 会逐渐增加速率。

TCP 的 **拥塞控制** 是通过维护一个 **拥塞窗口（cwnd）** 来实现的。拥塞窗口控制了发送方可以发送的数据量，确保不会超过网络的承载能力。

总结

TCP 拥塞是由于网络中的过量数据流量引起的网络资源不足，导致的数据丢失和性能下降问题。通过 **拥塞控制**，TCP 能够根据网络的状况动态调整数据的发送速率，避免发生严重的网络拥堵和丢包。了解并控制 TCP 拥塞是优化网络性能和提高数据传输效率的关键。

拥塞窗口在tcp中是如何设置与控制的

五、拥塞窗口 (Congestion Window, cwnd) 在 TCP 中的设置与控制

拥塞窗口（cwnd） 是 TCP 协议中控制数据流量的重要机制之一，它用于管理发送方在没有收到确认（ACK）之前可以发送的最大数据量。拥塞窗口的大小直接影响 TCP 连接的吞吐量与网络拥塞控制。

在 **TCP 拥塞控制** 中，cwnd 结合 **接收窗口（rwnd）** 一起决定了 TCP 发送方能在网络中传输的最大数据量。通常，cwnd 主要由网络的拥塞状况（即网络的带宽、延迟、丢包等因素）动态调整，而 rwnd 由接收方的接收缓存大小来决定。

cwnd 的工作原理

cwnd 是一个动态的窗口，它根据网络的当前状态（特别是网络拥塞情况）来调节，具体地说，cwnd 在不同的网络条件下通过不同的算法进行调整。通常，cwnd 的值会随网络的拥塞状况而变化，主要根据以下几个阶段和算法来设置和控制：

1. 慢启动 (Slow Start)

慢启动是 TCP 拥塞控制的初始阶段。当连接刚开始时，cwnd 会从一个小值（通常是 1 个最大报文段 MSS）开始增大。慢启动的目的是快速增大发送速率，但又要避免一开始就对网络造成过大的负担。

- **初始值**：在连接建立时，cwnd 被初始化为一个小值（例如，1 或 2 个 MSS）。

- **增加规则**：在每个往返时间（RTT）内，cwnd 会加倍。每当收到一个数据包的确认 ACK 时，cwnd 就增加 1 个 MSS（也就是说，在每一 RTT 中，cwnd 会加倍）。

  示例：假设 MSS 为 1460 字节：

  - 初始时：cwnd = 1 * MSS
  - 第一次 RTT 后：cwnd = 2 * MSS
  - 第二次 RTT 后：cwnd = 4 * MSS
  - 依此类推。

慢启动会一直进行，直到网络中的拥塞开始出现或达到某个阈值。

2. 拥塞避免 (Congestion Avoidance)

当 cwnd 达到一个 **慢启动阈值**（ssthresh）时，TCP 会进入拥塞避免阶段，增长速率会减慢，目的是避免网络拥塞过度。

- **cwnd 增长规则**：在拥塞避免阶段，每经过一个 RTT，cwnd 增加 1 个 MSS，而不是每个 ACK 增加 1 个 MSS（像慢启动阶段那样），这样能更平稳地增加 cwnd，避免网络突然发生拥塞。
- **阈值设置**：ssthresh 的设置通常是在慢启动阶段发生丢包后，根据丢包发生的时刻将 cwnd 降低一半，并将该值设置为 ssthresh。

3. 拥塞控制阶段的处理（丢包、快速重传与快速恢复）

如果在网络中发生了拥塞（通常表现为丢包），TCP 会触发 **快速重传**（Fast Retransmit）和 **快速恢复**（Fast Recovery）过程来重新调整 cwnd 和控制网络流量。

- **丢包检测**：TCP 丢包通常是通过接收到三个重复的 ACK 来触发的。
- **快速重传**：当发送方收到三个重复的 ACK 时，它认为中间某个数据包丢失，会立即重传丢失的数据包，而不是等待超时。
- **快速恢复**：快速恢复机制会将 cwnd 设置为丢失数据包的数量，并进入新的发送阶段。在此过程中，cwnd 会减少并且线性增加，以便在恢复阶段尽量避免过度拥塞。

4. 慢启动阈值的设置与调整 (ssthresh)

慢启动阈值（ssthresh）是一个关键的参数，它决定了 TCP 连接从慢启动到拥塞避免的过渡点。每当出现拥塞或丢包时，ssthresh 会被动态调整。具体来说：

- 在发生 **丢包** 后，TCP 会将 **ssthresh** 设置为当前 **cwnd** 的一半。
- 然后，cwnd 会重新设置为 1 或 2 MSS，并进入慢启动阶段。此后，cwnd 会逐渐增大，直到再次达到 ssthresh。

5. TCP 拥塞控制的五个主要算法

TCP 拥塞控制包含多个算法，以下是主要的五个算法，它们会影响拥塞窗口的计算和增长方式：

1. **慢启动**（Slow Start）
2. **拥塞避免**（Congestion Avoidance）
3. **快速重传**（Fast Retransmit）
4. **快速恢复**（Fast Recovery）
5. **拥塞窗口减小**：每当丢包发生，cwnd 会减半。

6. 拥塞窗口的计算示意

假设在 TCP 连接的开始时，cwnd 初始值为 1 MSS。

- **第一轮 RTT**：cwnd 增加到 2 MSS。
- **第二轮 RTT**：cwnd 增加到 4 MSS。
- **第三轮 RTT**：cwnd 增加到 8 MSS。

如果在第 3 RTT 时丢包发生，cwnd 会根据丢包位置减少并进入 **快速恢复** 阶段，ssthresh 会调整为丢包发生时的 cwnd 的一半。

总结：

- **慢启动**：刚开始时快速增大发送速率。
- **拥塞避免**：网络状态好时平稳增大 cwnd。
- **快速重传和快速恢复**：当丢包发生时，通过快速重传与恢复调整 cwnd，避免网络进一步拥塞。
- **慢启动阈值（ssthresh）**：决定何时从慢启动过渡到拥塞避免，并在丢包后动态调整。

这些机制和算法共同作用，通过动态调整拥塞窗口，帮助 TCP 在变化的网络环境中平衡流量，避免拥塞并提高数据传输效率。