运输层协议综述

运输层有一组很重要的功能：复用和分用
不同的应用进程可以使用同一个运输层协议传送数据
接收方运输层剥去报文首部后可以正确交付目的进程

网络层为主机之间提供逻辑通信
运输层为应用进程之间提供端到端的逻辑通信
运输层协议有无连接的UDP和面向连接的TCP
当运输层采用面向连接的TCP协议时，即使下面的网络是不可靠的，但这种逻辑通信信道就相当于一条全双工可靠信道；当运输层采用无连接的UDP协议时，这种逻辑通信信道仍然是一条不可靠的信道

用户数据报协议UDP： User Datagram Protocol
传输控制协议TCP： Transmission Control Protocol
协议栈层间抽象的协议端口是软件端口
软件端口是应用层各种协议进程和
运输实体进行层间交互的一种地址
端口号为16位，只有本地意义
服务器端使用的端口号：系统端口号和登记端口号
客户端使用的端口号：在客户进程运行时才动态选择

用户数据报协议

UDP只在IP数据报之上增加了很少一点功能
复用和分用的功能和差错检测的功能
①无连接 ②尽最大努力交付 ③首部开销小
④没有拥塞控制：适合用于实时应用
⑤支持一对一、一对多、多对一和多对多交互通信
⑥面向报文：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IP层，一次交付一个完整报文

首部格式：
①源端口 ②目的端口 ③长度 ④检验和
如果接收方UDP发现报文中的目的端口号不正确
丢弃该报文并由ICMP发送“端口不可达”差错报文
计算检验和时临时添加伪首部到UDP数据报的前面
IP数据报的检验和只检查IP数据报的首部
UDP的检验和把首部和数据部分一起检查
即检查源端口号、目的端口号、数据部分、
IP数据报的源IP地址和目的IP地址

传输控制协议TCP

TCP提供可靠交付、提供全双工通信
两端设有发送缓存和接收缓存
TCP面向字节流
TCP中的“流”指的是流入进程或从进程流出的字节序列
TCP把应用程序交下来的数据看成一连串无结构的字节流
TCP不关心应用进程一次把多长的报文发送到TCP缓存
根据对方给出的窗口值和网络拥塞程度决定一个报文应该包含多少字节，而UDP的报文长度是进程给出的

TCP连接的端点叫套接字（socket）或插口
端口号拼接到IP地址即构成了套接字

可靠传输的工作原理

理想的传输条件：
传输信道不产生差错
不管发送方以多快的速度发送，接受方总能来得及处理

当出现差错时让发送方重传出现差错的数据，在接收方来不及处理收到的数据时，及时告诉发送方适当降低发送数据的速度。这样，本来不可靠的传输信道就能实现可靠传输了。

停止等待协议：
停止等待就是每发送完一个分组就停止发送，等待对方的确认。在收到确认后再发送下一个分组。
发送方只要超过一段时间没有收到确认，就认为刚才发送的分组丢失了（也可能使确认丢失了），重传前面发送过的分组。这叫做超时重传。

①发送方发送完一个分组后，必须暂时保留分组的副本
②分组和确认分组都必须进行编号
③超时计时器设置的重传时间应当比数据在分组传输的平均往返时间更长一些

确认丢失：
当接收方收到了发送方发来同样的分组时，不向上交付
向发送方发送确认，此时的情况是最初的确认分组丢失

确认迟到：
发送方收到了接收方发来的重复确认时，收下后即弃
接收方收到重复的M1，同样丢弃，但要重传确认分组

停止等待协议的优点是简单，缺点是信道利用率太低
假设发送方发送分组需要时间T1，接收方发送确认分组需要时间T2，RTT是往返时间，则：
信道利用率 $U = T1/(T1 + T2 + RTT)$
为了提高传输效率，发送方可以不使用低效率的停止等待协议，采用流水线传输。

连续ARQ协议：发送方维持发送窗口，窗口的意义是发送窗口内的分组都可以连续发送出去而不需要等待确认
连续ARQ协议规定，发送方每收到一个确认，就把发送窗口向前滑动一个分组的位置。
接受方一般采用累积确认的方式，收到几个分组后，对按序到达的最后一个分组发送确认。

TCP报文首部格式

TCP报文首部前20个字节是固定的，后面是选项部分
因此TCP首部的最小长度是20字节

①源端口和目的端口：各2个字节
②序号：4字节 序号使用 mod 2的32次方运算
TCP连接中传送的字节流中的每一个字节都按顺序编号
首部中的序号字段值指本报文中第一个字节的序号
③确认号：4字节 确认号是期待收到对方下一个报文段的第一个数字字节的序号
若确认号为N，则N-1为止的所有数据都已正确收到
④数据偏移：4位，指出了TCP报文段的首部长度
数据偏移最大值为60字节，也是TCP首部最大长度
⑤保留：6位，今后使用
⑥6个控制位：
-----终止FIN（Finish）
----------释放TCP连接
-----紧急URG（Urgent）
----------高优先级的数据
-----复位RST（Reset）
----------TCP连接中出现了严重差错
-----确认ACK（Acknowledgement）
----------仅当ACK = 1时确认号字段才有效
-----推送PSH（Push）
----------两个进程进行交互通信时
----------有时一端的进程希望另一端能实时回应。
-----同步SYN（Synchronization）
----------在连接建立时用来同步序号。当SYN = 1
----------ACK = 0时表示这是一个连接请求报文段
----------若对方同意建立连接，则在相应中的报文置
----------SYN = 1，ACK = 1
⑦窗口：2字节 指的是发送报文段一方的接收窗口大小
窗口值是接收方让发送方设置其发送窗口大小的依据
⑧检验和：2字节 范围包括首部、伪首部和数据部分
⑨紧急指针：2字节 紧急指针仅在URG = 1时才有效
指出本报文段中紧急数据的字节数（其后是普通数据）
⑩选项：长度可变，0-40字节
-----最大报文段长度MSS
----------Maximum Segment Size
----------MSS是TCP报文中的数字字段的最大长度
----------MSS应尽可能大一些，只要
----------在IP层当中不需要分片传输即可
----------两个传送方向可以设置不同的MSS值
-----窗口扩大：扩大窗口用
-----时间戳：时间戳值字段和时间戳回送回答字段
----------计算往返时间RTT
----------防止序号绕回PAWS
----------Protect Against Wrapped Sequence
----------为了接收方能分开新的报文段和迟到很久的
----------报文段，可加上时间戳

TCP可靠传输的实现

发送窗口后沿后方表示已发送且收到了确认
发送窗口前沿前方表示目前不允许发送
发送窗口的位置由窗口前沿后沿共同决定

发送窗口后沿变化情况有两种：
不动（没有收到新的确认）和前移（收到了新的确认）

发送窗口前沿通常不断向前移动，也可能会不动
发送窗口前沿不动的情况有两种：
没有收到新的确认，对方通知的窗口大小不变
收到了新的确认，对方通知的窗口大小变小
发送窗口前沿也可能后退，原因是对方接收窗口变小了
但TCP标准强烈不赞成这样做

描述一个发送窗口的状态需要三个指针
P3 - P1 = 发送方的发送窗口
P2 - P1 = 已发送但尚未收到确认的字节数
P3 - P2 = 允许发送但尚未发送的字节数（可用窗口）

发送缓存用来暂时存放：
发送应用程序传送给发送方TCP准备发送的数据
TCP已发出但尚未收到确认的数据
发送窗口是发送缓存的一部分，通常二者后沿重合

接收缓存用来暂时存放：
按序到达的，但尚未被接收应用程序读取的数据
未按序到达的数据
TCP通常对未按序到达的数据先临时放在接受窗口当中
等待字节流中缺少的字节收到后按序交付上层应用进程

TCP要求接收方有累积确认功能，不过接受方不应过分推迟发送确认，否则会导致发送方不必要的重传
也可以在自己有数据要发送时把确认信息顺便捎带上

超时重传时间的选择：
TCP使用自适应算法，记录报文发送和收到确认的时间
两个时间差是该报文的往返时间RTT
TCP保留了RTT的加权平均往返时间RTTs
第一次测量到RTT时，RTTs = RTT
以后测量到新的RTT时
$RTTs = (1 - \alpha) \cdot RTTs + \alpha \cdot RTT$ 推荐 $\alpha = 1/8$

超时重传时间RTO（Retransmission Time-Out）
RTO应略大于RTTs
$RTO = RTTs + 4 \cdot RTTd$
RTTd是RTT偏差的加权平均，与新RTT和RTTs之差有关
第一次测量时，RTTd = 0.5RTT
以后测量到新的RTT时
$RTTd = (1 - \beta) \cdot RTTd + \beta \cdot |RTTs - RTT|$ 推荐 $\beta = 1/4$

在计算RTTs时，如果报文段重传了，则不采用其往返时间样本。这样得出的RTTs和RTO比较准确

选择确认SACK（Selective ACK）
若收到的报文段无差错，只是未按序号，中间缺少数据
可以只传送缺少的数据而不必重传已经正确到达的数据
和前后字节不连续的每一个字节快有两个边界
左边界和右边界
左边界：字节块第一个字节的序号
右边界：字节块最后一个字节的序号加1

TCP的流量控制

流量控制：
让发送方的发送速率不要太快，要让接收方来得及接收
TCP为每一个连接设有一个持续计时器
如果TCP连接的一方收到对方的零窗口通知，就启动持续计数器。若持续计时器设置的时间到期，就发送一个零窗口试探报文。对方在确认这个探测报文时给出现在窗口值。如果窗口仍然为0，那么重新设置持续计数器。如果窗口不是0，则打破了死锁的僵局。

TCP报文段的发送时机：
缓存中存放的数据大小达到MSS时组装发送
发送方的应用进程指明要求发送报文段（PSH）
发送方设置计时器，到时则将缓存装入报文段发送

Nagle算法：
发送应用进程把要发送的数据逐个字节送到TCP发送缓存
发送发就把第一个字节先发送去，把后面到达的字节缓存起来。当发送方收到对第一个字节的确认后，再把发送缓存中的所有数据组装成报文段发送出去。

当到达的数据已达到发送窗口大小的一半或已达到MSS时，则立刻发送一个报文段。

糊涂窗口综合征（Silly Window Syndrome）
让接收方等待一段时间，使得或者接收缓存足够空间容纳一个最长报文段，或者等待缓存有一半空闲空间。只要出现两种情况之一，接收方发出确认报文，通知发送方窗口大小。发送方也不发送太小的报文段，把数据积累成够大的报文或达到接收方缓存空间一半大小再发送

TCP的拥塞控制

拥塞的实质往往是整个系统各个部分不匹配，只有所有的部分都平衡了，问题才会解决。
拥塞引起的重传并不会缓解网络的拥塞，反而会加剧
拥塞控制是一个全局性过程
流量控制是端到端问题
想要拥塞控制就需要付出代价，需要获得网络内部流量分布的信息，在实施拥塞控制时，还需要在节点之间交换信息和命令，以便选择控制的策略和实施措施

吞吐量——负载图：
横坐标是提供的负载（Offered Load）
代表单位时间内输入给网络的分组数量
纵坐标是吞吐量（Throughput）
代表单位时间内从网络输出的分组数量
随着提供的负载增大，吞吐量的增长速率逐渐减小
当提供的负载增大到某一数值时，网络吞吐量下降为零
网络已无法工作，造成了死锁

分组的丢失是网络发生拥塞的先兆而不是原因
在许多情况下，正是拥塞控制机制成为引起
网络性能恶化甚至发生死锁的原因
从大的方面看，拥塞控制可以分为开环控制和闭环控制
开环控制：设计网络时把所有发生拥塞的因素考虑到
闭环控制：基于反馈环路概念，实时发现问题，解决问题

TCP进行拥塞控制的算法有四种
慢开始、拥塞避免、快重传和快恢复
慢开始（slow start）：
发送方维持一拥塞窗口 cwnd（congestion window）
拥塞窗口的大小取决于网络拥塞程度，动态变化
发送方让自己的发送窗口等于拥塞窗口
只要网络没有出现拥塞，拥塞窗口就可以再大一点
只要网络出现拥塞或可能拥塞，拥塞窗口就减小一点
判断网络拥塞的依据就是出现了超时
在慢开始中，每收到对一个新的报文段的确认
就可以把拥塞窗口增加最对SMSS的数值
每经过一个传输轮次，拥塞窗口cwnd就加倍
传输轮次：发送方把拥塞窗口内的报文都发送出去
并收到了对已发送的最后一个字节的确认
拥塞避免（congestion avoidance）：
慢开始门限ssthresh：
当cwnd < ssthresh：使用慢开始算法
当cwnd > ssthresh：改用拥塞避免算法
当cwnd = ssthresh：两种算法都可以
拥塞避免让拥塞窗口cwnd缓慢地增大，每经过一个往返时间RTT就把发送方cwnd加一，是线性规律缓慢增长，有“加法增大”AI（Additive Increase）的特点
当网络出现超时，调整门限值ssthresh = cwnd/2
同时修改拥塞窗口cwnd = 1，进入慢开始阶段

快重传（Fast Retransmit）：可以让发送方尽早知道发生了个别报文段的丢失。比如接收方收到了M1和M2都发出了确认，没有收到M3但收到了M4。本来接收方可以什么都不做，但是按照快重传算法接受方需要立即发送对M2的重复确认，让发送方尽早知道接收方没有收到。

快恢复（Fast Recovery）：发送方知道现在只是丢失了个别报文段，于是不启动慢开始而是执行快恢复算法。调整门限ssthresh = cwnd/2，设置拥塞窗口cwnd = ssthresh并开始执行拥塞避免算法。也有的快恢复的实现是把快恢复开始时的拥塞窗口cwnd再增大一点（3个报文段长度），cwnd = ssthresh + 3MSS

快重传（Fast Retransmit）：可以让发送方尽早知道发生了个别报文段的丢失。比如接收方收到了M1和M2都发出了确认，没有收到M3但收到了M4。本来接收方可以什么都不做，但是按照快重传算法接受方需要立即发送对M2的重复确认，让发送方尽早知道接收方没有收到。

快恢复（Fast Recovery）：发送方知道现在只是丢失了个别报文段，于是不启动慢开始而是执行快恢复算法。调整门限ssthresh = cwnd/2，设置拥塞窗口cwnd = ssthresh并开始执行拥塞避免算法。也有的快恢复的实现是把快恢复开始时的拥塞窗口cwnd再增大一点（3个报文段长度），cwnd = ssthresh + 3MSS

发送方窗口上限为：min（rwnd，cwnd）
rwnd<cwnd时是接收方的接受能力限制了发送窗口大小
rwnd>cwnd时是网络拥塞程度限制了发送窗口大小

网络层的策略对TCP拥塞控制影响最大的
就是路由器的分组丢弃策略
队列满时，再到达的分组都要被丢弃
这就是尾部丢弃策略
路由器的尾部丢弃往往使TCP进入拥塞控制的慢开始阶段
网络中通常有很多TCP连接，复用在IP数据报中传送
这会使很多TCP连接在同一时间突然进入慢开始阶段
这在TCP中称为全局同步（global synchronization）
为了避免网络中出现全局同步现象
使用主动队列管理（Active Queue Management）
在队列长度达到某个值得警惕的数值时主动丢弃
到达的分组，提醒发送方放慢发送速率

随机早期检测（Random Early Detection）
RED需要让路由器维持两个参数
即队列长度最小门限和最大门限
若队列长度小于最小门限就把新分组放入队列
若队列长度大于最大门限就把新分组丢弃
若队列长度介于最小门限和最大门限之间
按照某一丢弃概率p把新到达的分组丢弃