



TCP(一个端口有一个连接在监听->n+1)

服务器使用多个套接字服务客户:

- 1 服务器进程在欢迎套接字上等待客户的连接请求; 客户进程需要通信时,创建与服务器欢迎套接字通信的客户套接字,在此过程中,客户TCP向服务器TCP发送连接请求
- 2 服务器进程创建一个临时套接字(称连接套接字)和一个新的服务器进程,与客户进程通信
- 3 服务器进程回到欢迎套接字上继续等待
- 4 允许服务器同时服务多个客户
- 4 客户服务结束后,服务器销毁进程,关闭连接套接字

UDP&TCP

UDP

- 报文传输服务
- 由于没有建立管道,应用程序发送每个报文必须给出远进程地址
- 服务器使用一个进程和一个套接字为所有客户服务,一次请求-响应完成一次服务

TCP

- 字节流传输服务
- 由于建立了管道,应用程序只需向套接字中写入字节序列,不需指出远进程地址
- 服务器为每个客户单独生成一个套接字和一个新进程,允许双方长时间通信

Chap3 传输层

3.1 概述和运输层服务

Sender:

- 假设最多允许N个已发送、未确认的分组
- 发送端看哪个序号(由序号长度决定)划分为以下4个区域:

send_base nextseqnum already ack'ed usable, not yet sent

sent, not yet ack'ed not usable

window size

N

- 已发送未确认序号 + 未发送可用序号 = 发送窗口 (包含N个序号)
- Ack分给组带所有确认的序号
- 使用累积确认:若ACK包含序号q,表明“序号至q的分组均正确收到”
- 若发送方收到ACK q,则更新基序号为 q+1,整体滑动发送窗口
- 发送方只对基序号分组使用一个定时器
- 超时:发送方重传发送窗口中从基序号开始的所有分组

网络层:提供主机之间的逻辑通信,尽力而为的服务,网络层尽最大努力在主机间交付分组,但不提供任何承诺-不保证交付,不保证按序交付,不保证数据完整

传输层:提供进程之间的逻辑通信,依赖并增强网络层服务,基本服务:将主机间交付扩展到进程间交付

1.传输协议运行在系统上:

发送方:将应用报文封装成报文段,交给网络层发送.

接收方:从收到的报文段中取出载荷,交给应用层

2.传输层不能提供的服务:延迟保证;带宽保证;传输层可以提供的服务:保证可靠,按序的交付;TCP;不保证可靠、按序的交付;UDP;

3.2 多路复用与多路分解(传输层必须提供的服务)

1发送端多路复用:从多个套接字收集数据交给网络层

2接收端多路分解:将接收到的报文段交付到正确的套接字

3.如何进行多路复用和多路分解:

为将报文段交付给正确的套接字:

主机中每个套接字应分配一个唯一的标识

报文段中有特殊字段指示要交付的套接字

“发送方”传输层需在报文段中包含目的套接字标识(多路复用)

“接收方”传输层需将报文段中的目的套接字标识与本地套接字标识进行匹配,将报文段交付到正确的套接字(多路分解)

4.端口号是套接字标识的组成部分,是16比特的数,其中0~1023保留给公共域协议使用

5.UDP套接字标识:(目的IP地址,目的端口号) UDP的源是用来发响应的报文的.TCP套接字标识(不同标识不同套接字)(源IP源端口号,目的IP的端口号)

3.3 UDP(流媒体,DNS,SNMP)

1.UDP提供的服务:进程到进程之间的报文交付,报文完整性检查(可选);检测并丢弃丢出的报文

2.UDP需要实现的功能:多路复用与解复用,报文检错

3.UDP不提供的服务:可靠/按序交付;延迟及带宽保证;【若要UDP实现可靠传输,在应用层实现可靠性】

4.UDP 报文结构:报头+载荷,报头如下:

用于多路复用/多路分解的字段:源端口号,目的端口号;用于检测报头错误性的字段:报文段长度,检核和;

5.UDP检查和/所有 16 位比特字相加,若溢出,则相加和+1,然后取反,接收方计算检核和,然后与checksum相加,若所有比特位都是1,正确,否则有错.

5.优点1没有建立连接的延迟2协议简单,发送端和接收端不需要保存连接状态3头部开销小[UDP8B TCP20B],承载效率高,网络带宽利用率高4没有拥塞控制和流量控制,可以尽可能快的发送报文.

6.适合应用:容忍丢包但对延迟敏感的应用;如流媒体

【单次请求/响应为主的应用如DNS

3.4 可靠数据传输(RDT)原理 有限状态机 (FSM)

数据通过可靠信道传输,虽有损坏和丢失,且按序接收.(有序无损)

Rdt1.0:底层信道完全可靠,只需发送和接收即可. 1-1

Rdt2.0:下层信道可能产生比特错误 **错误检测,接收方反馈** ACK/NAK,发送方发送后等待反馈,NAK重传 3-2

Rdt2.1:ACK/NAK 出错:发送方发现受损就重传,但接收方会出现冗余,发送方给分组添加序号[0,1],接收方发现序号相同就丢弃,重新发送ACK(A**CK不带序号,但带checksum**) 2-2

Rdt2.2:不用NAK,接收方收到出错的分组/非期待分组,重发最近一次ACK(带序号);发送方:若ACK的序号非期待(表明当前分组未被确认),重发当前分组.非期待重发(4-2)

Rdt3.0:可能丢包,检测丢包与恢复; 启动定时器,在超时前没收到期待的ACK,重发.受到非期待ACK不处理是停等协议,效率低,需要流水线机制.仅超时重传

传输时延(发送包用的时间)t=(L/R)*(N流水线),

信道占用时间 RTT+ L/R

发送方利用率:u=t/(RTT+L/R)

GBN:发送方只对基序号分组使用定时器,若ACK 不对,不管,仅超时重传整个发送窗口(确认更新).接收端不缓存失序分组,非期待重发,收到4,5, ACK=5

SR:自动调整窗口,避免不必要的重传.

发送窗口:包含已发送未确认和未发送可用的序号,可能有发送已确认的序号交织其中,基序号是已发送未确认或未发送可用的序号.

发送方收到的ACK不是最小未确认分组(基序号),则标记.若是,滑动窗口.

接收方一旦收到就ACK,失序就缓存.若收到窗口前冗余分组,说明发送方出错,发送ACK(n), 收方窗口大小限制在序号空间的一半以内**N<m/2**

接收窗口:包含期待但未收到和允许接收的序号可能有已确认已缓存的序号在其中基序号为期待但未收到或允许接收的序号,收到基序号分组时,按顺序交付分组

滑动接收窗口

发送方:从上层接收数据;若发送窗口不满,发送分组,启动定时器 n 超时:重传分组n,重启定时器

收到发送窗口内的ACK(n),标记分组n为已确认,若n=基序号, 滑动发送窗口,使基序号=最小未确认的序号或下一个序号//其余情形:忽略.

接收方:收到接收窗口内的分组n:发送ACK(n);若为失序分组:缓存该分组;若n=基序号交付从n开始的若干连续分组.滑动接收窗口,使基序号=下一个期待接收的序号;若收到[rcvbase-N,rcvbase-1]内的分组n:发送ACK(n) //其余情形:忽略

发送端滑动窗口的序号数目 + 接受端滑动窗口的序号数目 <= 分组编号数目

分组重排、序号重引用: 存活时间+长序号

3.5 TCP 协议

服务模型:点到点,全双工,面向连接,可靠有序字节流

需要机制:建立连接流水式发送报文(可靠)流量控制

2. TCP报文段结构:

源2,目的端口号2;序号:首字节在字节流中的序号,非报文段序号(seq4,确认号:希望从对方接收的下一字节序号,隐含累积确认(ack4);/首部长度:32bits为单位的首部长度(head len);接收窗口2;检验和2;紧急数据指针(指向最后一个字节Urg data pointer);数据段(data

GBN: 接收方FSM

初始化

expectseqnum=1

send pkt (0,ACK,chksum)

收到预期序号的分组

Wait

nd:rcv(seq)<pkts&& notcumack(rcvpkt)&& hasseqnum(rcvpkt,expectseqnum)

提取(pkt.data) deliver (data,data) send:make (pkt,expectseqnum,ACK,chksum) seq:send(seq) expectseqnum++ //预期序号加1

收到预期序号加1

收到失序的分组:

丢弃 (不在接收窗口的分组),意味着接收方只能按顺序接收分组

并重发前一次的ack

收到上层发回请求:

若发送窗口满:拒绝请求

若发送窗口不满:

构造分组并发送 (从下一个可用序号开始设置)

若原来发送窗口为空:对基序号启动一个定时器

收到ACK q:

更新基序号为q+1

若发送窗口变为空:终止定时器

若发送窗口不为空:对基序号启动一个定时器

收到出错的ACK:

不做处理,为什么?

1.若ack p出错,可由后续收到的ack q (pp) 进行累积确认

2.若ack p出错,且后续未收到ack,则超时后重传 pkt p

定时器超时:

启动定时器,从已发送未确认的分组开始,发送位于当前发送窗口内的所有分组

多路复用/分解:

标志位:URG:紧急数据 PSH:立即交给上层 RST:不接受连接 SYN:建立连接 FIN 拆除连接

接收端事件

收到一个期待的报文段,且之前的报文段均已发出确认

收到一个期待的报文段,且前一个报文段被延迟确认

收到一个失序的报文段(序号大于期待的序号),检测到序号问题

收到部分或全部期望回应的报文段

接收端动作

推迟发送确认,在500ms时间内若无下一个报文段到来,发送确认

立即发送确认(为准确估计RTT)

立即发送确认(快速重传的需要),重复当前的确认序号

若报文段始于间隔的低端,立即发送确认(以便发送超时),更新确认序号

接收窗口:接收端还可以接受的**字节数**

重要TCP选项:MSS:TCP段可以携带的**最大数据字节数**(缺省=536Byte);window scale:窗口比例因子,实际接收窗口大小=window size*2^n;window scale; SACK:选择确认,在累积确认的基础上允许接收端指出缺失的数据字节

5.往返时间估计

估计平均RTT 指数加权移动平均 a = 0.125

EstimatedRTT = (1-a)*Est+RTT+a*SampleRTT

安全距离 beta = 0.25 DevRTT = (1-beta)*DevRTT + beta*[SampleRTT - EsRTT]

超时值: TimeoutInterval=1s

TimeoutInterval = EstimatedRTT + 4*DevRTT

6. TCP 发送方处理的事件:

收到应用数据:(1)创建,发送TCP报文段;(2)若当前没有定时器运行(没有已发送未确认的报文段),启动定时器;

超时:重传包含最小序号,未确认的报文段;重启定时器;

-收到

data from IP

spare room

TCP data to buffer

RevWindow

application process

RevBuffer

ACK:(1)如果确认序号大于基序号,推进发送窗口(更新基序号,由于接收方采用累积确认机制,故发送窗口可一次推进多步,避免重发某些丢失了 ACK 的报文段);(2)如果还有未确认的报文段,启动定时器;(3)快速重传:当发送方收到对同一序号的3次重复确认时,立即重发包含该序号的报文段(why3次?Because,发送方收到冗余ACK不仅可能因为报文段丢失,还可能因为网络中的报文段重新排序,使达接收方顺序不同,此时接收方缓存失序报文段,可能下一个到达报文段就是期望值了.)

event: ACK received, with ACK field value of y

if (y > SendBase) { //收到更新的确认号

SendBase = y;

if (there are currently not-yet-acknowledged segments) start timer }

else { //收到重复的ACK

increment count of dup ACKs received for y

if (count of dup ACKs received for y = 3) {

resend segment with sequence number y }

定时器补偿-发送方每重传一个报文段,超时值就增大一倍.-若连续发生超时事件,超时值呈指数增长(规定的上限值).Karn算法& EstimateRTT & 定时器补偿算出超时 超时重传+快速重传

7.接收端处理的事情 非期待重发

推迟发送ACK:500ms 每隔一个正常确认

-优点:可以减少通信量;

-缺点:当延迟太大时,会导致不必要的重传

推迟确认造成RTT估计不准确.TCP协议规定:

Go-Back-N TCP SR

接收方: 接收方: 接收方:

使用累积确认 使用累积确认 缓存失序的分组

不缓存失序的分组 缓存失序的报文 每个分组使用一个定时器

对失序分组发 对失序报文段发 单独确认每个正重复ACK 重重复ACK 确认到的分组

发送方: 发送方: 发送方:

超时后重传 超时仅重传最早 超时后仅重传未基序号开始的 未确认的报文段 被确认的分组

所有分组 增加了快速重传

推迟确认的时间最多为500ms,接收方至少每隔一个报文段使用正常方式进行确认;

6. TCP 流量控制:调节发送速度,使 接收缓存不会溢出;接收缓存中的可用空间= RcvWindow //接收窗口 = RcvBuffer-[LastByteRcvd - LastByteRead]

接收方将RcvWindow放在报文段中,向发送方通告接收缓存的可用空间; LastByteRcvd >= SendBase-1;

发送方限制未确认的字节数不超过接收窗口的大小,即LastByteSent-LastByteAcked <= RcvWindow

接收窗口为0时,发送方必须停止发送.此时发送方启动定时器,超时后发送零窗口报文段(seq为上一段最后一个字节),从接收方的响应中获知窗口大小,仍0, 重定时糊涂窗口综合症:双方传输速度严重失衡;接收方不断发送微小窗口通告,引起发送方不断发送很小的数据分组,导致带宽浪费.接收方:当窗口大小显著增加(达到min(带宽空间(一半,MSS))之后才发送更新后的窗口

当窗口大小不满足以上策略时,推迟发送确认(但最多推迟500ms,且至少每隔一个报文段使用正常方式确认),仅当窗口大小满足以上时,再通告新的窗口大小;

发送方:Nagle算法连接后,数据到来时组成一个TCP段发送.收到确认后,后到来的数据放在发送缓存,当数据量达一个MSS或上次传输确认到来,将缓存中数据发适应网络延迟, MSS长度及应用速度的各种组合/常规情况下不会降低网络的吞吐量

连接管理:发送和接收方在交换数据前先握手.

-两次握手的不可行性-一个不可靠网络中存在许多干扰连接正常存在的因素:包传输延迟变化很大,存在重传的报文段,存在报文重排;

-三次握手建立连接:

Step1)不包含数据,2)不包含数据(服务器端分配缓存和变量)3)可能包含数据(客户端分配缓存和变量)

-四次挥手 closing a connection:

客户端:向服务器发送 FIN,等待服务器确认; 服务器:向客户端发送 ACK,确认请求;

服务器:向客户端发送 FIN,等待客户端确认; (可合并客户端:向服务器发送 ACK,等待一段时间)后结束.

Problems:服务器收到SYN段后,发送SYNACK段,分配资源,若未收到ACK段,服务器超时后重发该段.服务器等待一段时间(称SYN超时)后丢弃未完成连接.SYN超时类型为30~120s-SYN洪泛攻击:攻击者用伪造的源IP向服务器发送大量SYN段,不发送ACK段,服务器为维护巨大的半连接表耗尽资源,无法处理正常客户连接扫描-FIN扫描(ACK=1, RST=1/no rep)

7.拥塞控制原理: 网络辅助TCP ECN, ATM, Decbit 流量控制:限制发送速度,使不超过接收端处理能力;拥塞控制(多个源):限制发送速度, <= 网络的处理能力;拥塞造成(用来判断拥塞):1丢包(路由缓存溢出);2分组延迟增大(路由来不及转发,等待转发分组多);3拥塞指数:分组延迟大,网络负载重,但网络吞吐量很低-TCP采用从端到端的拥塞控制机制,发送方根据自己感知的网络拥塞程度,限制其发送速度.

发送方感知到拥塞后,改变其发送速率的算法是什么?

发送方使用拥塞窗口CongWin>=(限制)已发送未确认的数据量,其间按规则发送速率.

大概的:Rate = CongWin/RTT bytes/sec

TCP拥塞窗口的调节(拥塞控制):加性增,乘性减(AIMD)慢启动,每个RTT,CW加倍,(收到1ACK,CW增加1MSS)

发送方维护变量 Threshold

超过阈值,慢启动状态转为拥塞控制

发生丢包时,Threshold = CongWin/2 转为快速恢复

-收到3个冗余ACK:说明网络仍然有一定交付能力

Threshold = CongWin/2

Reno快速恢复 CongWin = Threshold + 3MSS;

-超时:网络的交付能力很差

Threshold=CongWin/2

CongWin = 1MSS(发送速率降至最低)

无条件到慢启动,增大CW, CW<threshold,每次加倍;否则只加1MSS(拥塞避免:缓慢增大CW).

8. TCP 吞吐量CW/RTT,丢包后= CW/2RTT(平均3/4)

吞吐量与丢包率L的关系:【1.22MSS/(RTT 根号 L)】

TCP的公平性:如果K条TCP连接共享某条带宽为R的瓶颈链路,每条连接具有平均速率R/K.

1. 面向连接服务的优点缺点?

优:可靠/有序传输;资源预置(使用)缺:需要全局信息

2. 无连接服务的优点与缺点?

优点:无需知道网络状态(包括网络资源)或只需知道局部网络状态;缺点:具有不确定性(是否有满足服务的网络资源不确定,能否完成服务;不确定)

3. 分层网络体系结构的不足:

上层协议的性能依赖于下层协议

4. 分组交换原理:

1存储转发;2动态路由(包括每个分组自带源地址、目的地址,拓朴发现、路由选择);3输出端由端系统处理

5.若一个WWW文档中除有文字外,还有7个图像.试问使用单个http/1.0,4个并行http/1.0与单个http/1.1各需要建立几次TCP连接? (1);8; (3); (3); (1)

6.假定要传送的报文共有k(单位bit),从源节点到目的节点共有k跳链路,每条链路的传播时延为d(单位s),链路带宽为b(单位bit/s);电路交换(包括连接建立与拆除)使用的控制帧(或信令)长度,在各节点的排队时延忽略不计;分组交换使用的分组头,分组长度分别为h、p(单位bit),分组在各节点的排队时延q(单位s).试分析在何种条件下电路交换的总时延要小于分组交换的总时延?

电路交换总时延D(c):

连接建立时间: kd;连接拆除时间: kd

数据传输时间: x/b;数据传输时间: kd

D(c)=3kd+x/b

-分组交换总时延D(p):

-单个分组传输时间: (p+h)/b

-第1跳传输时间: (x/p)/(p+h)/b (x/p为分组个数)

-传输时间每1跳增加1个分组的传输时间/总的传输-时间为x/p*(p+h)/b+(k-1)*(p+h)/b

-排队时间: kq;传播时间: kd

D(p)=x/p*(p+h)/b+(k-1)*(p+h)/b+kd+kq

若D(c)<D(p),则电路交换总时延小于分组交换总时延

1.停等,GBN,选择重传三个滑动窗口协议有关问题:

答: (1)三个协议的发送窗口、接收窗口大小分别是多少? (2)发送窗口最大有效值由什么因素决定?

答: (1)1、1/N、1: N、1/N

(2)网络缓冲能力(或RTT*瓶颈节点带宽,即BDP)

2. 实现TCP连接目标的三要素制.

(1)通过传输层地址(端口号)实现进程间通信(2)通过确认机制实现可靠传送(3)通过接收方缓存实现按序传送(4)流量控制(5)拥塞控制(6)连接建立与拆除机制

3.在TCP连接中,客户端的初始号215,客户打开连接,只发送一个携带有300字节数据的报文段,然后关闭连接下面从客户端发送的各个报文段的序号分别是多少?

SYN报段215,数据报段216,FIN报段5164.

在一条新建TCP连接上发送一个长度为40KB的文件,发送端每次发送一个最大长度的报文(MSS),MSS=1KB,接收端正确收到一个TCP段后立即给予确认,发送端的初始拥塞窗口T限设置为16KB.假设发送端尽可能快地传输数据,但只要发送窗口允许,发送端就发送一个MSS.

(1)发生第一次超时后,发送端将拥塞窗口门限调整为4KB.请问发生超时的时,发送端的拥塞窗口是多大? 发送端共发送了多少数据,有多少数据被成功确认了?

(2)发送端从未被确认的数据开始使用慢启动进行重传.假设此后未再发生超时,当文件全部发送完毕时,发送端的拥塞窗口是多大?

答: (1)发送端拥塞窗口大小 = 4KB*2 = 8KB

新建的TCP连接上,采用慢启动开始发送,当第一次超时发生时,已发送=1KB+2KB+4KB+8KB=15KB.

此时,最后一批8个TCP段未获确认,之前发送的TCP段都被确认,因此成功确认的数据量为7KB.

(2)此问题是在(1)问题的基础开始的,即已确认过7KB,在未发送与未发送有33KB,发送端采用慢启动重新开始发送,在拥塞窗口达到4KB时发送数据量=1KB+2KB+4KB=7KB.然后进入拥塞避免阶段:在收到全部4个MSS的确认后,拥塞窗口增至5KB,相应地发送端发送了5KB数据;收到全部5个MSS的确认后,拥塞窗口增至6KB,相应地发送端发送了6KB数据;收到全部6个MSS的确认后,拥塞窗口增至7KB,相应地发送端发送了7KB数据;收到全部7个MSS的确认后,拥塞窗口增至8KB,相应地发送端发送了8KB数据;此时刚好发完. 因此,文件发送结束时,发送端的拥塞窗口大小为8KB.

5. TCP如何发送紧急数据?

1)紧急标志位U(URG)置1;(2)紧急数据置于TCP段数据(载荷)前部;(3)紧急指针指向紧急数据的最后一个字节.

6. TCP接收方向何种情形需要立即进行确认?

答:(1)连续两个段按序到达,且前一个未确认;(2)收到失序段(序号比期望的序号大);(3)收到丢失分段;

7. TCP协议中ACK的作用.

(1)建立连接,拆除连接(2)差错控制(或可靠传送)

(3)流量控制(4)拥塞控制

一个基于重传的可靠传输协议通常包含以下要素:差错编码,确认,重传,定时器,分组序号,为什么需要这些要素

差错编码: 差错编码提供了一种发现传输错误的机制

确认:确认提供了反馈机制,使发送方可以得知接收方的接收情况;重传:重传提供了修正错误的方,即用新的正确的替换错误的;定时器:定时器提供了发现/定义超时时间的方法,避免死锁无限等待,分组序号/分组序号提供了区分不同分组的方式,流水线操作不

会引起混乱
当分组到达交换设备时，若输出链路的缓冲队列满，发生丢包。
DNS zone-不重叠区域-一个权威域的边界
报文放大
为什么DNS主要使用UDP：及时传递应用数据、无需建立连接（时延小）；无连接状态，支持更多活跃用户；分组首部开销小（UDP 8；TCP 20）
提供了一种按层次结构命名主机的方法
调用方式：向本地DNS代理的一个RPC调用
第一个P2P文件共享服务 Napster
Gnutella：完全分布；公共域协议；有许多Gnutella客户软件
发现对等方（客户软件自带列表）- 尝试建立
Ping- Pong
洪流：参与一个特定文件分发的对等方集合
跟踪器、列表
长为256KB，加入洪流时没有数据块
Tit for tat 10s-重新评估；30s-随机选择
视频分发面临问题：支持大量用户；网络环境异构
解决方案：采用分布式应用层基础设施
图像编码技术：帧内冗余（空间）+帧间冗余（时间）
CBR：编码速率固定//VBR



使用一个巨型服务器（不可行）
单点故障
网络拥塞点
远端用户传输距离长，跨多个ISP，
带宽低
同一条链路上传输多个视频拷贝，
浪费带宽
在地理上分布的多个站点存储和
提供服务（CDN）：
enter deep: 将CDN服务器深入
部署到大量接入网中，靠近用户
Akamai拥有1700个站点

bring home: 将少量（几十个）较大的集群部署在靠近接入网的POP中 Limelight采用此法
不可靠的数据报服务：由UDP协议实现 DGRAM
可靠的字节流服务：由TCP协议实现 STREAM
应用服务需求:reliability, bandwidth, delay, security
传输层提供的服务：传输服务
本地网络层接口 k基本服务-将主机间交付扩展到。。
TCP/UDP报文段格式
UDP套接字标识：<（目的）IP地址，（目的）端口号> 二元组
TCP四元组标识
网络层提供的服务加上不保证带宽及延迟要求
UDP报头： 8 bytes TCP报头： 20bytes
UDP补足16bit整数倍的长度不会包含在UDP数据包长度字段中，也不会被发送。

计算UDP检查和包括伪头、UDP头和数据三个部分。
检查和使用是可选的，若不计算检查和，该字段填入0。

