第三章 运输层

1 运输层提供的服务

- 在两个**不同主机**上运行的**应用程序**之间体统**逻辑通信**
- **发送**方将应用**程序报文**分成**数据段**传输给网络层
- **接收方**将**数据段**重新组装成**报文**传递到**应用层**

1.1 传输协议

- TCP
- UDP

1.2 运输层和网络层的区别

- **网络层**：两个**主机**之间的逻辑通信
- **运输层**：两个**进程**之间的逻辑通信

1.3 多路复用 & 多路分解

- 在**接收**主机多路**分解**：将接受的数据段传输到**正确的套接字**。
- 在**发送**的主机多路**复用**：在多个套接字收集数据，用**首部**封装数据，然后将报文段传递到**网络层**
- 主机用**IP地址**和**端口号**指明数据段属于哪个合适的**套接字**

1.4 无连接的多路分解

- 当主机收到**UDP数据段**
  - 检查数据段中的**目的端口号**
  - 用端口号指示UDP数据段属于哪个**套接字**
- 具有不同的**源IP地址**或**源端口号**，但有着相同**目的端口号**和**目的IP地址**的数据报指向同样的套接字

1.5 面向连接的多路分解

- **TCP套接字**由4部分指定：
  - 源IP地址
  - 源端口号
  - 目的IP地址
  - 目的端口号
- 接收主机使用所有**四个值**将数据段定位到**合适的套接字**

2 UPD协议特性

- **无修饰**、**不加渲染**的英特网**传输层**协议
- UDP数据段可能
  - **丢失**
  - 传递**失序**报文
- **无连接**
  - 在UDP接收者和发送者之间**没有握手**
  - 每个UDP数据段的处理**独立于**其他数据段

2.1 为什么有UDP

- 低延迟：无需建立连接
- 简单：无连接状态
- 数据段首部小
- 速度快：无拥塞控制

2.2 UDP是面向报文的

- UDP的**首部**开销小，只有**8个字节**。
- **发送方UDP**对应程序交下来的报文，在**添加首部**后就向下交付**IP层**。UDP对**应用层**交下来的报文，**既不合并，也不拆分**，保留这些报文的**边界**
- **接收方UDP**对**IP层**交上来的UDP用户数据报，在**去除首部**后就原封不动的交付上层**应用层**

2.2 UDP首部格式

- **源端口 2**
- **目的端口 2**
- **长度 2**
- **校验和 2**

3 校验和的实现思想

- 将数据每**16位**分为一组
- 将所有**数据组**相加

- 例如：

> 0110 0110 0110 0000
> 0101 0101 0101 0101
> 1000 1111 0000 1100

- 依次相加，在**第二次相加后**得到了

> **1** 0100 1010 1100 0001
> 注意出现了第**17位**

- 将第17位的1加至第一位得到

> 0100 1010 1100 0001 + 1
> 0100 1010 1100 0010

- 取反码

> 1011 0101 0011 1101

4 可靠数据传输原理

- 有易到难，从最完美的情况做假设

4.1 版本一 rdt 1.0

- **完全可靠**的信道上的信道上的**可靠数据传输**
  - 无**Bit错误**
  - 无**分组丢失**
- 此时发送方正常发送，接收方正常接收

4.2 版本二 rdt 2.0

- 下层信道可能会让传输分组中的**Bit受损**
  - 使用**校验和**检测Bit错误

方法

- **确认（ACKs）**：接收方告诉发送方分组**正确接收**
- **否认（NAKs）**：接收方明确告诉发送方分组**接受出错**
- 发送方收到NAK之后**重发分组**

rdt 2.1

- ACK/NAK有混淆风险
  - 发送方并不知道接收方**发生了什么**
  - 不能**正确重发**
- **发送方**
  - 为每个分组交替加上**序号（0/1）**
  - 记住当前**报文状态**号
  - 收到**1号报文确认**则发送**0号报文**，反之**重发**1号报文
  - 收到**0号报文确认**则发送**1号报文**，反之**重发**0号报文
- **接收方**
  - 检查是否收到**重复报文**

rdt 2.2

- 不再使用**NAKs**
- 使用**ACKs**表示被**确认的报文序号**

- 当**发送方**收到**重复**的ACKs，说明该分组没有正确送达，需要**重新发送**该分组

4.3 版本三 rdt 3.0

- 存在**Bit出错**
- 存在**报文丢失**

方法

- 发送者等待**一定确认时间**
- 如果在这个时间没有收到确认就**重发**

4.4 网络利用率的计算

$$U_sender = \frac{L/R}{RTT+L/R}$$

- **L**：Packet Length in **bits**
- **R**：Transmission Rate, **bps**

4.5 流水线技术

- 发送方允许发送多个**还在路上**的**未确认报文**
  - 序号数目的**范围**必须增加
  - 发送方和接收方必须设置**缓冲区**
- 流水线技术可增加**网络利用率**

4.5.1 Go-Back-N

- 在发送完一个帧后，**不用停下来等待确认**，而是可以**连续发送多个数据帧**。这样就减少了等待时间。
- 如果**前一个帧在超时时间内未得到确认**，就认为丢失或被破坏，**需要重发出错帧及其后面的所有数据帧**。

- 线路很差时，使用退后N帧的协议会**浪费大量的带宽重传帧**。

- **发送方**

    - 在分组头规定一个**k位的序号**
    - **窗口：**允许**连续未确认**的报文
    - **ACK（n）：**确认从该组第**一个报文到n个报文**已被接收
    - **超时：**重新发送**n以后的报文**。

- **接收方：**

    - 总是位正确接收的**最高序号的分组**发送ACK
    - **丢弃**接收的失序分组

4.5.2 选择性重传（Selective Repeat，SR）

- 发送方有**发送窗口**接收方有**接收窗口**

- **发送方**

    - 发送**发送窗口**内的报文
    - 收到**ACKs**时，若该确认时发送窗口内的**第一个报文**，则向后滑动。

    - **超时**之后，只重传位**未被确认**的那一帧。

- **接收方**

    - 正常接收，滑动接收窗口
    - **缓存**收到的**非期望帧**
    - 返回所收到帧的**ACK**

- **窗口大小应小于或等于序号空间的一半！！**

5 面向连接传输：TCP

- **全双工数据**
    - 同一个连接上的**双向数据流**
    - **MSS**：最大报文段长
- **面向连接**
    - 在数据交换前**握手**
    - 初始化**发送方**和**接收方**的状态
- **流量控制机**
    - 发送方**不会淹没**接收方

5.1 TCP往返时延的估计

指数加权移动平均

$EstimatedRTT = (1-\alpha) \times EstimatedTT + \alpha \times sam$
$\alpha = 0.125$

设置超时

$$DevRTT = (1-\beta) \times DevRTT + \beta \times |SampleRTT -$$
$$\beta = 0.25$$
$$TimeoutInterval = EstimatedRTT + 4\backslash \text{Times} DevRT$$

- 初始**TimeInterval**设置为**1秒**
- 获得第一个**样本RTT后**

$$EstimatedRTT = SampleRTT$$
$$DevRTT = SampleRTT \div 2$$
$$TimeoutInterval = Esti, atedRTT + max(G, K \times De$$

- **K** = 4
- **G**是用户设置的**时间粒度**

5.2 可靠数据传输

快速重传

- 如果发送方收到对**同样报文**的**三次重复确认**，则发送方认为**该报文之后**的数据段丢失，在定时器超时之**前快速重新发送**该报文。

TCP ACK

- 所期望的报文段到达，且在期望序号**之前的报文段都被确认**
    - **先等500ms**，看看还有没有报文，再发送ACK

- 期望序号的报文按序到达，另一个报文段**正在准备发送ACK**
  - 立即发送**单个累计ACK**确认两个有序报文段
- 收到**失序报文段**，监测到**缝隙**
  - 立即发送**重复ACK**，指出**期望序号**
- 到达报文段**填充间隙**
  - **立即**发送ACK

5.3 流量控制

- 发送发不能发送**太多太快**，让**接收缓冲区**溢出
- **流量控制**使用**接收窗口**接收缓冲区中的剩余空间
- **接收方**在**报文**中宣告**窗口中**的剩余空间
- **发送方**限制没有确认的数据不超过**接收窗口**

5.4 连接管理

建立连接-三次握手

- **客户**发送**TCP SYN**报文到服务器
  - 指定**初始序号**
  - **无数据**
- **服务器**接收**SYN**，回复**SYNACK报文**
  - 分配**缓冲区**
  - 指定**初始序号**
- **客户**收到**SYNACK**，回复**ACK报文**，可能包含**数据**

关闭连接

- **客户**发送**TCP FIN**控制报文到**服务器**
- **服务器**接收**FIN**，回复**ACK**，**半关闭连接**，并发送**FIN**到客户
- **客户**接收**FIN**，回复**ACK**
- **服务器**接收**ACK**，关闭连接

6 拥塞控制原理

端到端的拥塞控制

- 没有从网络中获取的**明确反馈**
- 从**端系统**观察的**丢失和延迟**判断出拥塞
- TCP的方法

网络辅助的拥塞控制

- **路由器**给端系统提供反馈

7 TCP采用的拥塞控制

- 端到端，无**网络辅助**

发送方如何感知拥塞

- **丢失事件**：超时或者3个重复的ACKs
- **TCP发送方**在丢失事件发生之后降低发送速率

三个机制

- AMID
- 慢启动
- 对超时事件作出反应

TCP AIMD

- **乘性递减**：发生丢失事件之后将**拥塞窗口减半**
- **加性递增**：每个RTT内如果没有**丢失性事件**发生，拥塞窗口增加一个**MSS（最大报文段长）**

TCP 慢启动

- 连接开始的时候
  $CongWin = 1MSS$
- 连接开始的时候以**2的指数**方式增加速率，直到第一个**丢失事件**发生

TCP对拥塞事件的反应

- 三个**重复**的确认之后
  - **CongWin减半**后加上**3个MSS**
  - 窗口**线性增长**
- **超时事件**后，TCP进入慢**启动**过程

    - **Congwin**设置为1
    - 窗口开始**指数增长**
    - 到达一个值数之后再**线性增长**

什么时候从指数增长变为先行增长?

- 当CongWin达到**超时前**的**一半**的时候