1. 丢包的原因

因为采用包交换，每个包都要在中间设备里采用存储转发方式工作，转发不是直接把包发到传输线路上，在输入链路和输出链路设置了缓冲，而缓冲是以包为单位的，设置了一些缓冲区，当数据包到达的速度快于输出链路时，输出缓冲区就耗尽，此时有新的数据包就无法接收新的数据包，这是直接将数据包丢掉。

1. 分层的原因

人们处理复杂系统的一种有效模式，设计模式、分析模式

软件架构有分层架构—4层：

表现层

应用层

业务层

数据服务层

网络分层：

* ISO/OSI RM：International Standard Organiztion Open System Interconnection Reference Model分7层—参考模型
  + 与TCP/IP比较多了两层
    - 会话层
      * 一次通信时有多个事情协商
    - 表示层
      * 加密、压缩等
* TCP/IP分层5层—事实上的工业标准
  + 物理层—二进制位的传输，位流
    - 1、位如何表示0 1如何表示
    - 2、位同步问题
    - 3、发送端实现并—串的转换
    - 4、接收端实现串-并的转换
    - 5、物理接口—接插件的尺寸和功能规范
    - 6、线路上传输的是基带信号还是模拟信号—传输的就是二进制信号
  + 链路层—数据链路层
    - 物理层解决的是位的传输
    - 以帧（frame）为单位的传输—字节流
      * 八位串—字节流
      * 起始串、结束串
      * 控制信息
        + 包括源地址、目的地址—MAC地址（局部使用的地址，是同一个网段，采用的是直接传送方式送到的）
    - 帧的匹配—识别帧
    - 地址匹配—是否传输给本节点的帧
  + 网络层—以包（datagram）为单位，解决节点标识问题，解决从源到目的路径的决策问题
    - 全局地址（IP地址）--可以用来表示范围，范围可以分为大范围，大范围由可以分为小范围，邮政编码二位一组，但是IP地址的分组非常灵活，不是固定的位数
      * IPv4—32bit
      * IPv6—128bit
      * 蹭热度的IPv9—忽悠人
    - 解决寻径问题
      * 范围如何标识，以节点为目标的话，路由器中必须配置大量存储器存储路径信息2^32=4G\*4，查找的代价也很大，范围标识方式IP加上掩码，掩码用来指定范围所用的位数
        + 掩码有两种方式指定，标识从左到右连续1的个数

IP加上255.255.255.0

IP/数字，如172.16.1.234/24

* + - * 网络的拓扑结构
        + 表示的问题
        + 网络设备如何获取的拓扑结构—Routing Algorithm
  + 传输层—以segment（段）
    - 传输层以下的层在什么地方实现—在网络边界、网络核心中的所有设备上实现，而传输层是在端系统中实现
    - 传输主体的标识—端系统上运行的网络应用
      * 16bit端口号—port number—发送者和接收者
    - 收发室类似的功能
      * 发送端收集数据
      * 接收端分发数据
    - 传输功能
      * 可靠传输
        + 超长的数据自动拆拼
        + 按顺序到达
        + 保证送达
      * 不可靠
  + 应用层—message
    - 命令、数据
      * 编码
        + 数据和分开
        + 放在一起编码
* 网络细节
  + 网络设备
    - 两层设备—交换机
    - 三层设备—路由器

ATM(银行)—Automatic Teller Machine

ATM(网络) Asynchronous Transfer Mode

网络发展的相关弯路：

三网合一，最初的三网（电话网、数据网、视频网）合一合到电话网，因为电话网有传输质量保障，传输质量（QoS，Quality of Service）：延时、带宽、可靠，固定电话网对上述三个都可以保障，计算机网络无法保障。提出解决方法就是ATM。数据包以53字节为单位的包—Cell（信元），屏蔽两个不同电话模式之间的差异—北美、欧洲模式。

1997年IP电话投入试用，中国到美国的每分钟电话费18元，网络带宽不断改善，通过IP电话实验后人们发现数据网既便宜还能达到电话网的效果，人们改变了看法，现在三网合一合到数据网—计算机网络（Internet）

Best effort

部署TCP/IP—TCP/IP详解，卷2

BSD UNIX，OpenBSD、FreeBSD

开发模式：如何利用云计算提供的计算资源，开发分为前端和后端，后端运行于云平台，前端运行在浏览器或者以Native App方式安装在各种平台上。

浏览器上运行，屏幕尺寸差异太大，衍生出响应式编程（Responesive）可以适应各种不同尺寸的屏幕。

Native App：

运行平台：

Desktop：

Windows：

* MFC（GUI）、C++/Qt + C++
* Java + Swing

Linux：

* Qt + C++
* Java

macOS:

* Objective-C
* Swift
* Java
* Qt + Cplusplus

Mobile：

iOS

* Objective-C
* Swift
* Qt + Cplusplus

Android:

* Java
* Kotlin

要实现跨平台：以前方案都有限制

* Eletron + JavaScript—Visual Studio Code

现在可选增加了Flutter，编程语言Dart，可以跨越浏览器、Desktop、Mobile。目前使用Flutter编写的程序可以运行在浏览器、iOS、Android、macOS上，Windows和Linux上正在完善。

后端采用云计算：

采用的架构：微服务（microservices），Cloud Native（原生云），对编程语言没有限制、编程时仅考虑业务，不用考虑微服务治理和有关微服务的安全。微服务治理和微服务的安全—service mesh（服务网格）。