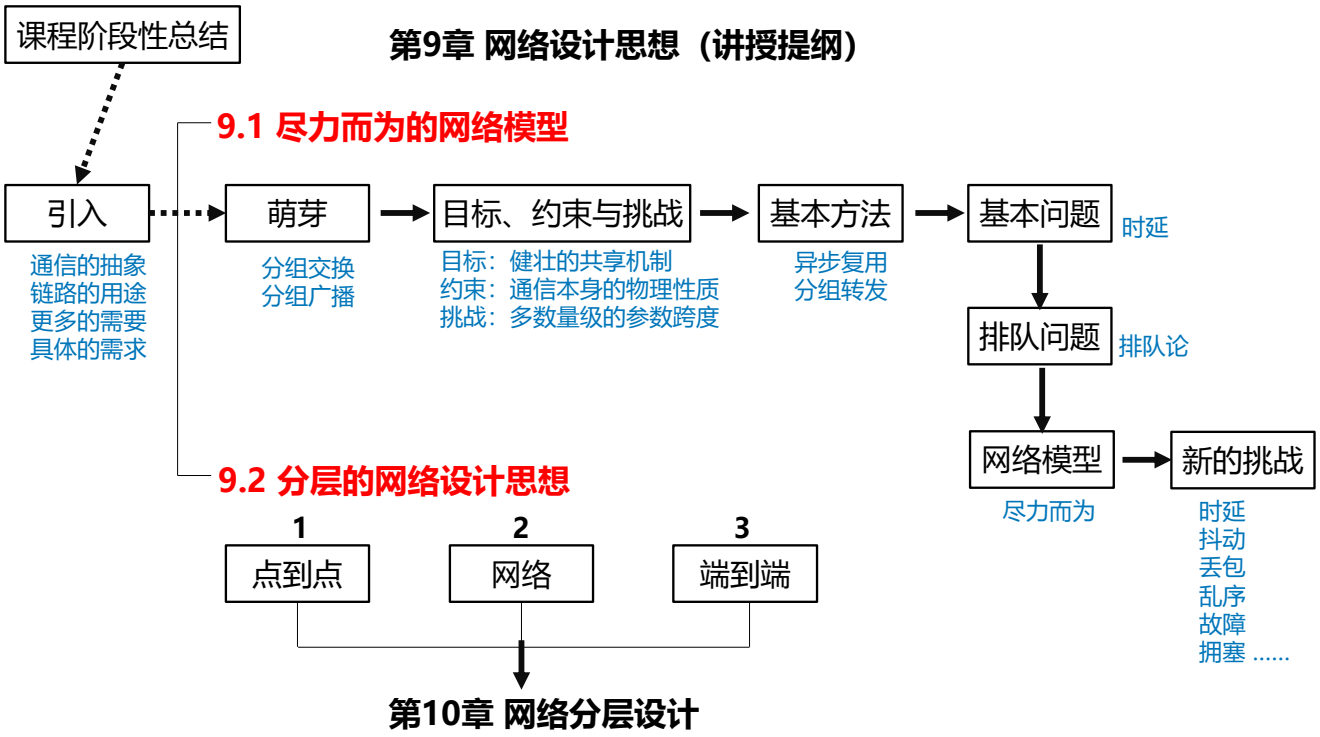
10. 分层设计实现

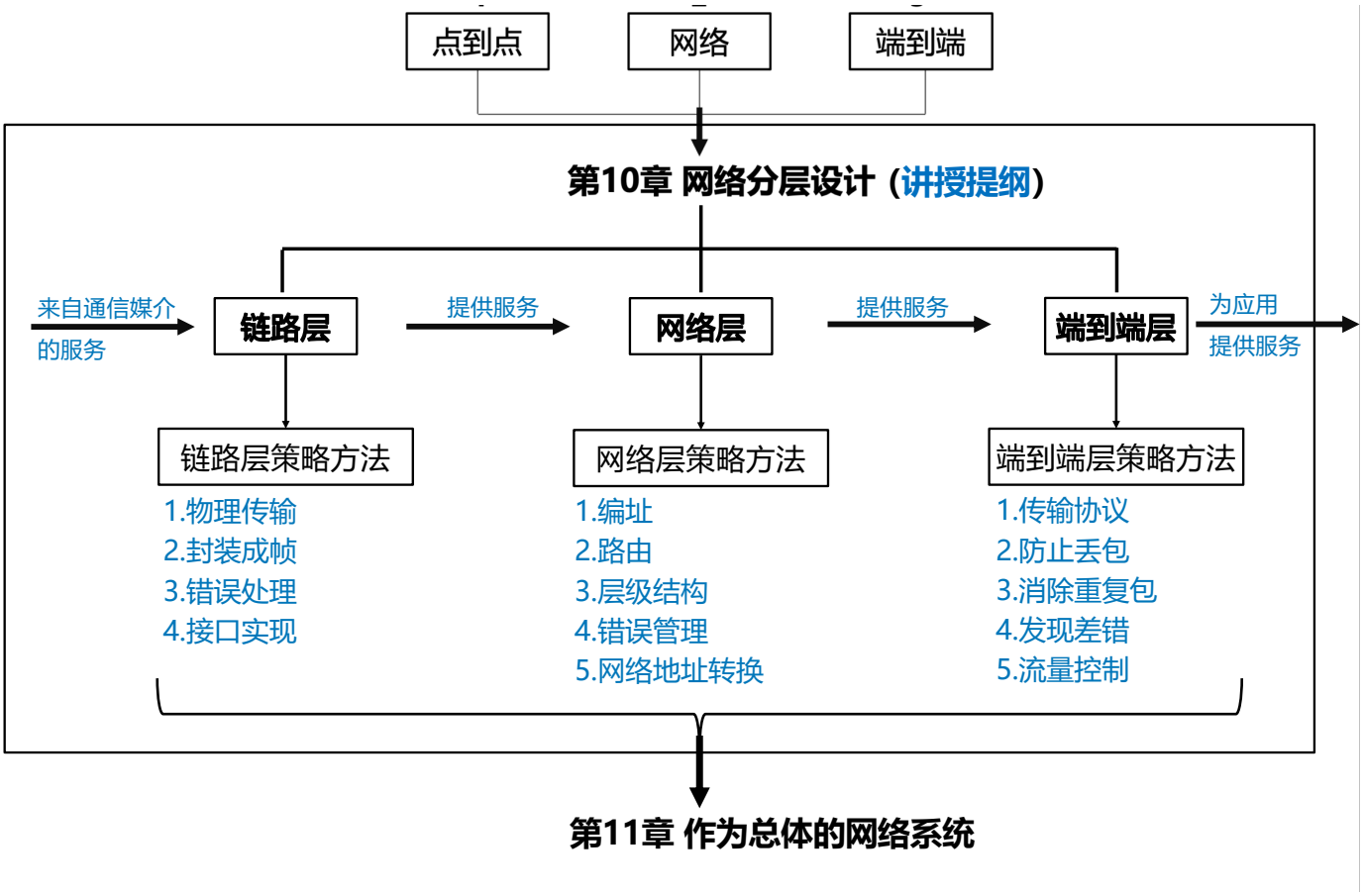
20

本章相关的参考文献

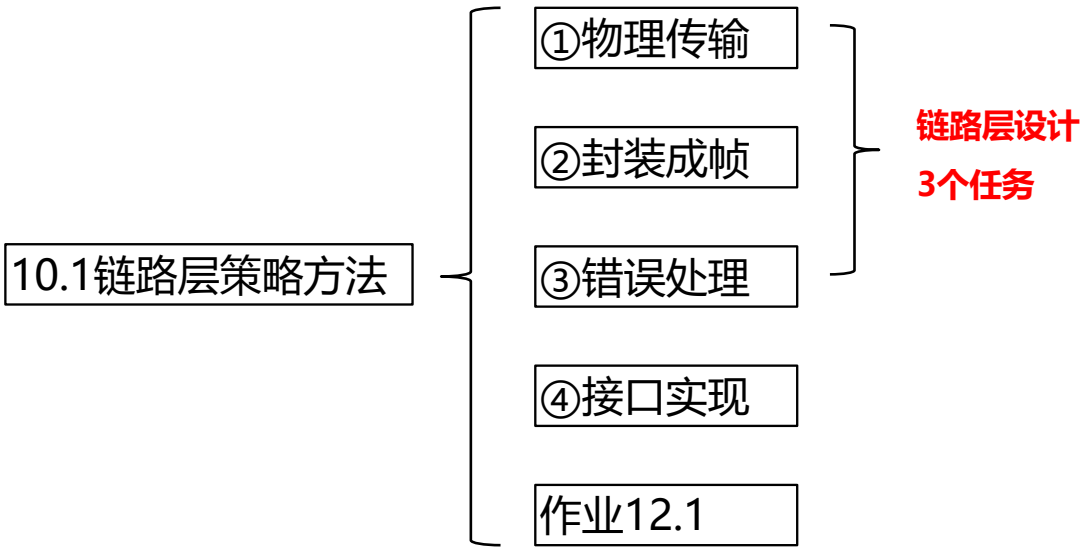
- Claude E. Shannon. A mathematical theory of communication. Bell System Technical Journal 27 (1948), 379–423 & 623–656.
- Clark D. The design philosophy of the DARPA Internet protocols[C]//Symposium proceedings on Communications architectures and protocols. 1988: 106-114.
- David D. Clark and David L. Tennenhouse. Architectural considerations for a new generation of protocols. ACM SIGCOMM ’91 Conference: Communications Architectures and Protocols, in Computer Communication Review 20, 4 (September 1990), 200–208.
- Vinton G. Cerf and Peter T. Kirstein. Issues in packet-network interconnec- tion. Proceedings of the IEEE 66, 11 (November 1978), 1386–1408.

回　顾


课程阶段性总结
第9章 网络设计思想（讲授提纲）
9.1 尽力而为的网络模型
引入
通信的抽象
链路的用途
更多的需要
具体的需求
萌芽
分组交换
分组广播
目标、约束与挑战
目标：健壮的共享机制
约束：通信本身的物理性质
挑战：多数量级的参数跨度
基本方法
异步复用
分组转发
基本问题
时延
排队问题
排队论
网络模型
尽力而为
新的挑战
时延
抖动
丢包
乱序
故障
拥塞 .....
9.2 分层的网络设计思想
1
点到点
2
网络
3
端到端
第10章 网络分层设计


点到点
网络
端到端
第10章 网络分层设计（讲授提纲）
来自通信媒介
的服务
链路层
提供服务
网络层
提供服务
端到端层
为应用
提供服务
链路层策略方法
1.物理传输
2.封装成帧
3.错误处理
4.接口实现
网络层策略方法
1.编址
2.路由
3.层级结构
4.错误管理
5.网络地址转换
端到端层策略方法
1.传输协议
2.防止丢包
3.消除重复包
4.发现差错
5.流量控制
第11章 作为总体的网络系统

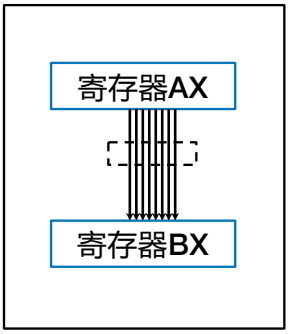
10.1 链路层策略方法



① 物理传输

1.同步：收发者在约定的时间片段，同时设置/观测物理量

- 在同一芯片上，可用同步传输
- 例如：MOV AX BX
- 正确性假设
  1. 电压在公差内
  2. 无电气干扰
  3. 时钟周期间隔满足传播时间要求

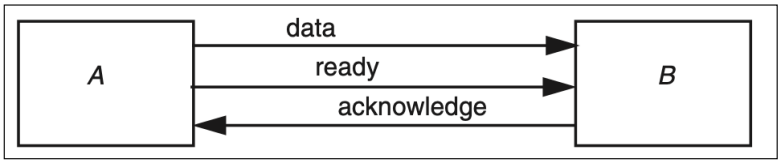



需要几根线?
是否适用远距离传输?

① 物理传输

2.异步：收发者未约定时间片，接收方完成后给予反馈

- 对于不同模块，常用异步传输
- 例
  - 3线异步协议
    1. 发送方设置ready 信号
    2. 接收方完成后设置acknowledge信号



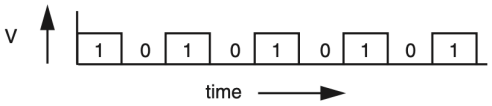

是否适用远距离传输?

- 数据率（单位时间可传输的数据）与传播距离的关系?
  - 距离增长，传播时间增长，数据率下降!

① 物理传输

串行同步：适用于远距离传输

- 同步：双方都知道时间片大小
- 串行：单线传输**比特流**
- 基本方案：高电平1，低电平0



问题：

- 距离和干扰导致信号：衰减、时间偏移
- 如何提高抗干扰能力？

方案

- 相位编码：1和0都包含高电平，1和0都有电平跳变

相位编码

相位编码 - 曼彻斯特编码：

- 0=低电平-高电平，1=高电平-低电平
- 收益：抗干扰、自同步、实现简单
- 开销：最大数据率减半

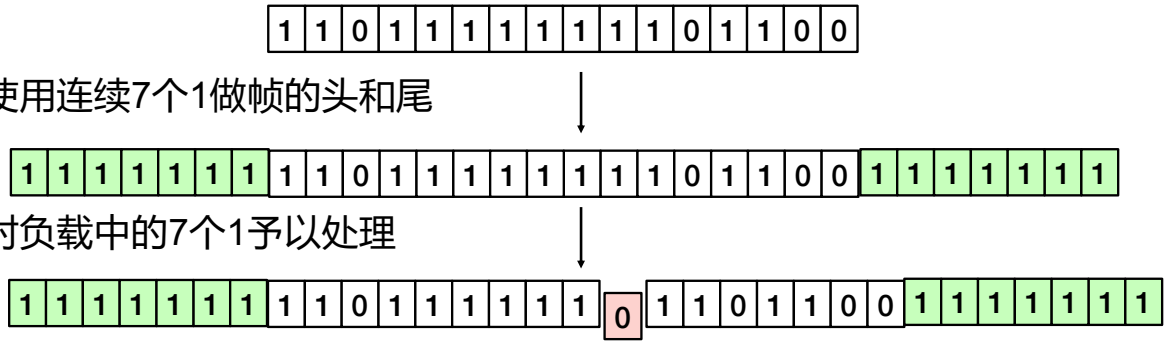
更多关于物理传输的问题，通过讲义自学

- 带宽、C/W、BER（bit error rate）

② 封装成帧

成帧：帧是链路层的数据单元，如何从比特流中提取出帧?

- 常用方法（定界）：
  - 为帧加上特定模式的首和尾
  - 对帧内数据进行位填充
- 例：



  - 使用连续7个1做帧的头和尾
  - 对负载中的7个1予以处理

③ 错误处理

使用检错码可发现错误，
但检出的错误如何处理?

1. 丢弃
2. 请求重发
3. 使用纠错码

思考：开销是什么?

实际应用：

- 联合使用，简单纠错+丢弃，取得可靠性和性能的折中

④ 接口实现

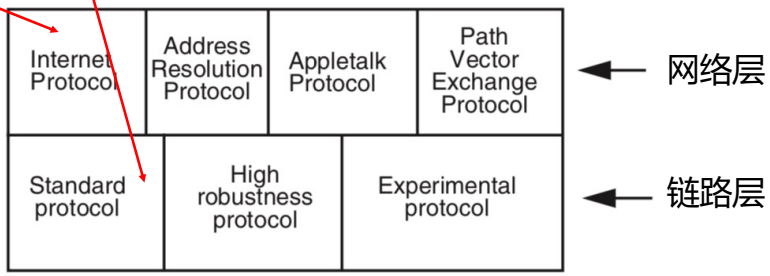
基本原语：link_send 和 network_handle

1. link_send (data_buffer, link_identifier)
2. network_handle (data_buffer, link_identifier)

- 假设上下层各只有唯一的协议
- 如果有多个网络层和链路层协议，如何标示请求者和服务者呢?

载荷

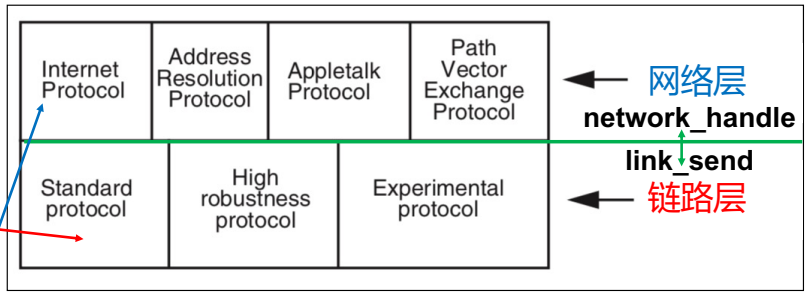
什么含义?
如何体现了从网络到链路的转换?



④ 接口实现

复用原语

1. **link_send** (data_buffer, link_identifier, **link_protocol**, **network_protocol**)
2. network_handle (...)



参数含义

- link_protocol:
  - 用哪个链路层协议
  - 支持多链路层协议
- network_protocol:
  - 网络层哪个协议调用的
  - 传给对等方时，network_handle处理时分派给正确的模块
  - 支持链路复用

④ 接口实现 - link_send代码

讨论：1.功能？2.考虑？

```
structure frame
   structure checked_contents
      bit_string net_protocol                    //多路复用参数
      bit_string payload                         //载荷数据
   bit_string checksum

procedure link_send (data buffer, link_identifier, link_protocol, network_protocol)
   frame instance outgoing_frame
   outgoing_frame.checked_contents.payload ← data_buffer
   outgoing_frame.checked_contents.net_protocol ← network_protocol
   frame_length ← length(data buffer) + header_length
   outgoing_frame.checksum ← checksum (frame.checked_contents, frame_length)
   sendproc ← link_protocol_proc[link.protocol]              //选择链路协议
   sendproc (outgoing_frame, frame_length, link_identifier)  //发送帧
```


④ 接口实现 - network_handle代码

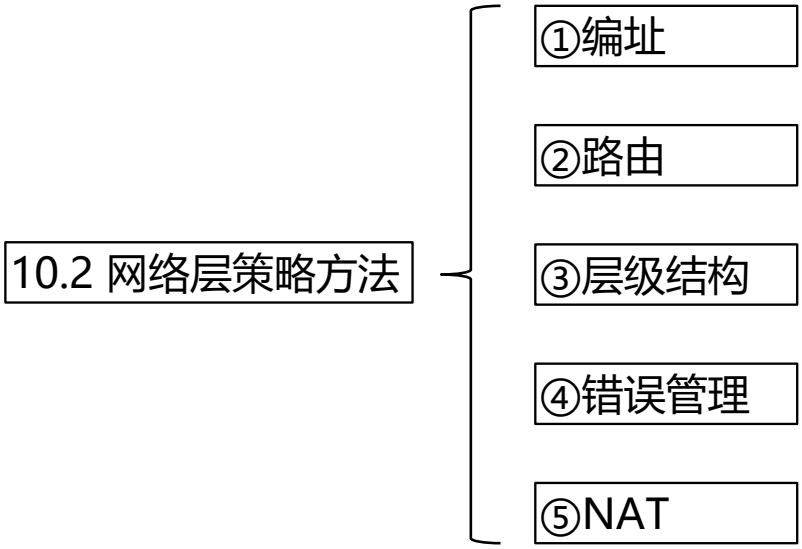
课下思考：如何实现network_handle?

作业12.1

MTU与BER

- 链路协议通常会根据链路情况设置MTU数值，即：最大传输单元（maximum transmission unit），其发送的帧均不能大于MTU。
- 问题：
  - 为什么需要设置MTU?
  - 如果该链路协议的BER（比特误码率）是稳定和已知的，它是如何影响MTU的?

10.2网络层策略方法



网络层

（因特网）网络层最大的挑战是什么?

- 一种设计，面对悬殊的参数（性能指标、流量负载、接入点数量）
  - **KIS系统设计原则，E2E网络设计原则**

网络层概念

- 网络（network）
- 网络地址（net. addr.）
- 网络接入点（net. addr. point）
- **源（source）**
- **目的（destination）**

① 编址（addressing）

将目的地址与数据进行关联

1. **network_send** (segment_buffer, destination, network_protocol, end_layer_protocol)
2. **network_handle** (packet, network_protocol)

仍然多对多

这里需解决什么问题?

下一页代码寻找答案

链路层：link_send (data_buffer, link_identifier, link_protocol, network_protocol)

从编址到路由

```
procedure network_handle(net_packet, net_protocol
   packet instance net_packet
   if net_packet.destination ≠ MY_NETWORK_ADDRESS
      next_hop ←lookup (net_packet.destination,
      link_send (net_packet, next_hop, link_proto
   else
      give_to_end_layer (net_packet.payload, net_
```

```
procedure network_handle(net_packet, net_protocol)
   packet instance net_packet
   if net_packet.destination ≠ MY_NETWORK_ADDRESS then
      next_hop ←lookup (net_packet.destination, forwarding_table)
      link_send (net_packet, next_hop, link_protocol, net_protocol)
   else
      give_to_end_layer (net_packet.payload, net_packet.end_protocol,
net_packet.source)
```

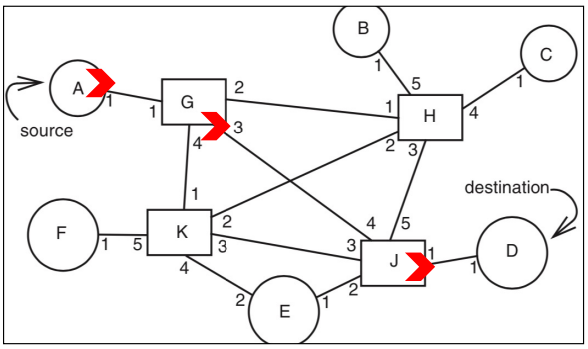
路由的转发表举例

A的转发表

destination	link
A	end-layer
all other	1

G的转发表

destination	link
A	1
B	2
C	2
D	3
E	4
F	4
G	end-layer
H	2
J	3
K	4



使用举例：A→D

- A的转发表：通过链路1发往G
- G的转发表：选择通过链路3发往J（目标D）
- J的转发表：选择链路1到D

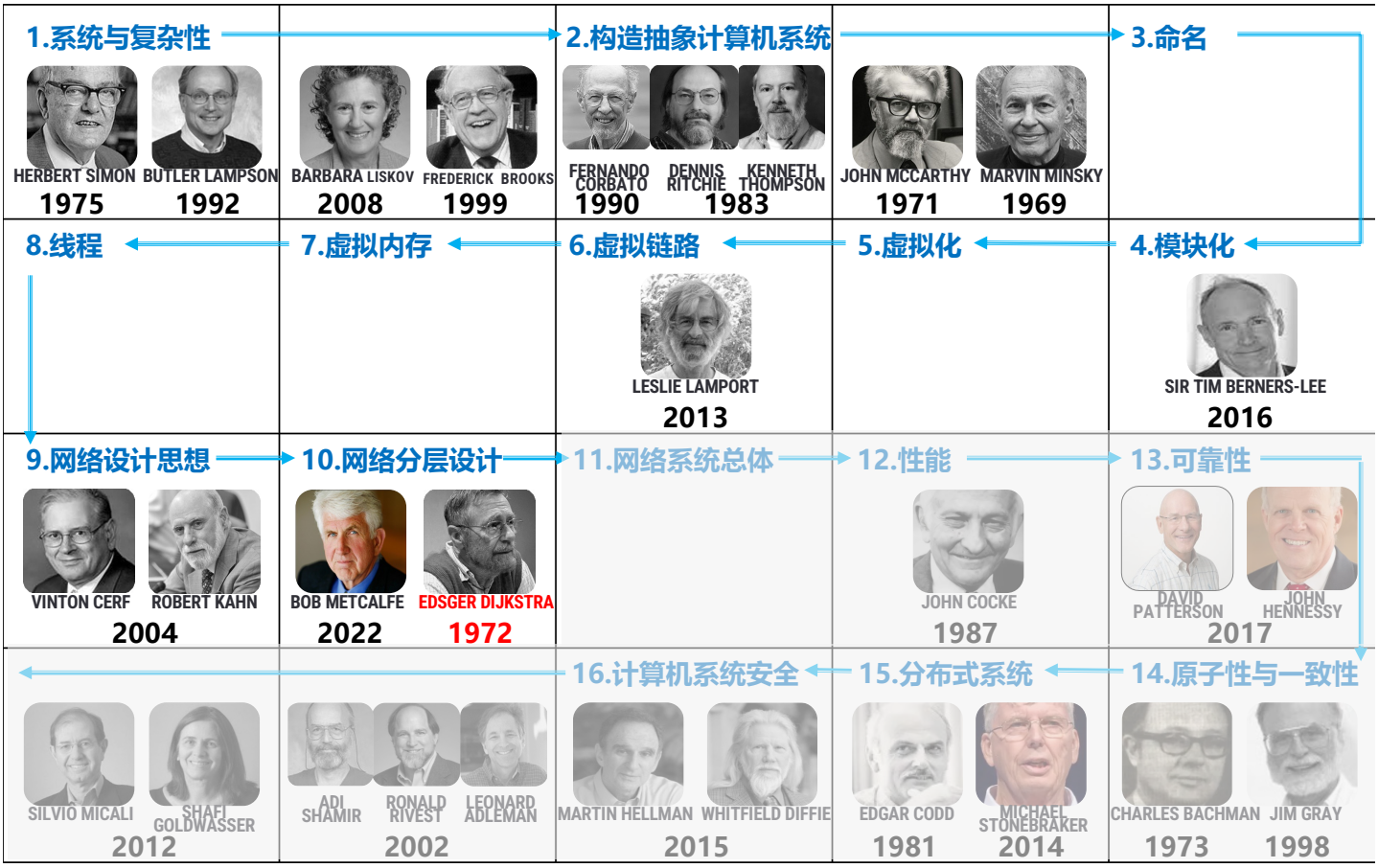
② 路由

问题：转发表如何建立?

- 人工计算（Dijkstra）
  - 收集网络链路的所有信息，计算出每个节点的最优转发表
  - 思考：计算量~规模？计算频率~规模？突发故障~怎么办？

人工建立转发表的问题

- 1
- 2
- 3



② 路由

问题：转发表如何建立?

- 自动路由算法
  1. 静态路由（static routing）：不考虑变化和故障
     - 在网络情况已知的情况下，计算当前转发表
     - 能否分布式计算?
  2. 自适应路由（adaptive routing）：考虑变化、故障
     - 如何随时计算和分发?
- 实现方案
  - 可计算路由的转发器 → 路由器（router）= 转发器 + 路由计算器
  - 路由计算器需要交换数据? 路由协议（routing protocol）

路由算法

建立转发表的关键问题之一：协调

- 为什么？如何算出全局协调的转发表?

是互联网最重要的算法之一，很多尝试，很多失败，并没有终极答案。

路由算法

小网络的好算法：path vector exchange（路径向量交换）

- 路径向量：节点间的**完全**路径，用节点序列（向量）表示
  - 例如：<A，H，J，E>
- 算法步骤：
  1. 路径广播（path advertising）
  2. 路径选择（path selection）
  3. 迭代
  4. 完成并写入

第1步：路径广播

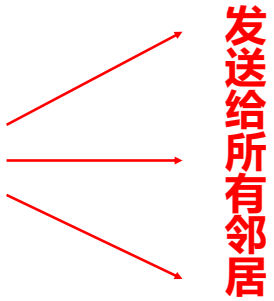
1. 发送 <自己的地址, 路径向量>给所有直接邻居

- 链路：沿所有链路
- 协议：使用PATH_EXCHANGE协议

- 以G为例：

to	path
G	< >

G的转发表的开始状态



第2-4步

2. 将收到的信息合并进自己的路径向量

- 合并方法：收到的路径前置上本节点，是否纳入？两种情况：

1. 新终点的路径全纳入
2. 旧终点路径选择更优

问题：更优的标准是什么？

- 跳数少？
- 数据率高？
- 负载或排队少？

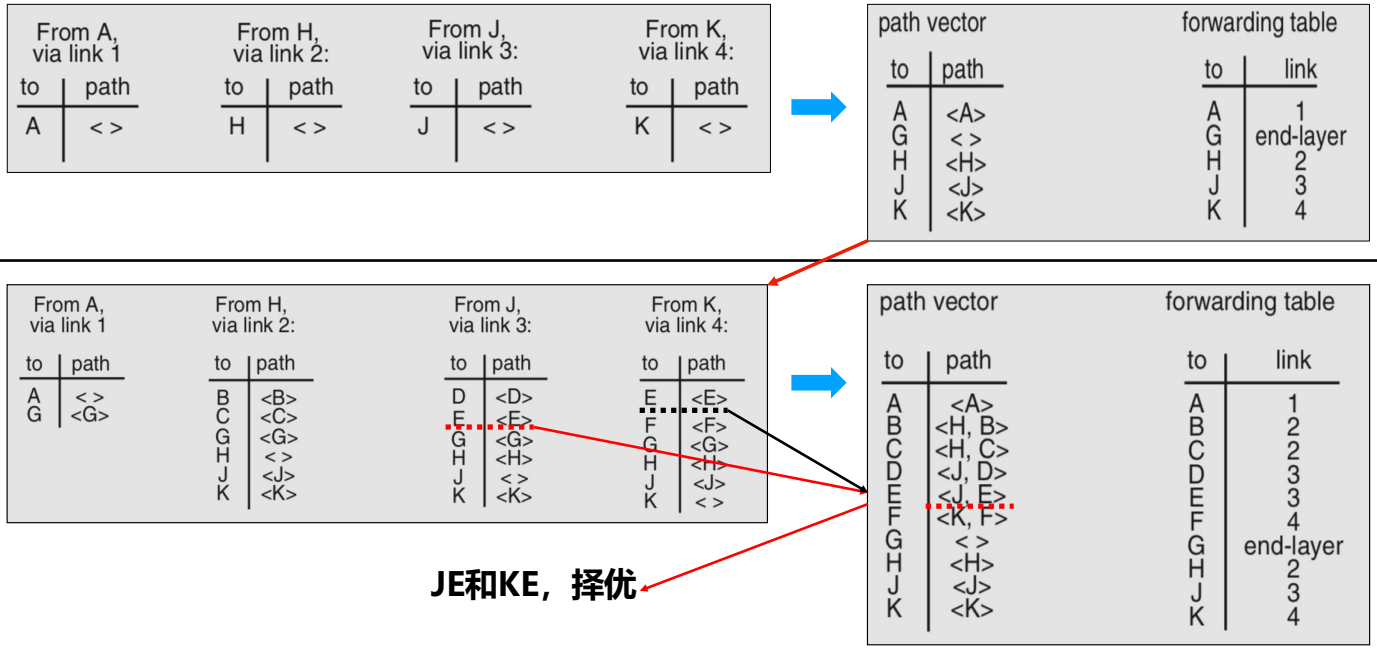
3. 迭代

- 重复1、2
- 直到找到全部终点

4. 写入

- 写入（静态）转发表

路径向量交换过程



自适应算法

- 基本算法：判断出网络拓扑变化时，重新、从头进行计算

- 优化算法：使用当前路径向量作为基础进行计算
  - 删除：
    - 删除包含失效链路的路径，并广播所有路径向量
    - 邻居节点根据此信息删除包含该链路的路径，如何获得新路径？
  - 新增：
    - 增加包含新链路的更优路径，并广播该路径向量
    - 邻居节点根据此信息决定是否更新自己的路径

链路失效收敛问题

课下复现该算法：

```
// Maintain routing and forwarding tables.

vector associative array        // vector[d_addr] contains path to destination d_addr
neighbor_vector instance of vector  // A path vector received from some neighbor
my_vector  instance of vector   // My current path vector.
addr associative array          // addr[j] is the address of the network attachment
                                // point at the other end of link j.
                                // my_addr is address of my network attachment point.
                                // A path is a parsable list of addresses, e.g. {a,b,c,d}

procedure main()                        // Initialize, then start advertising.
    SET_TYPE_HANDLER (HANDLE_ADVERTISEMENT, exchange_protocol)
    clear my_vector;                    // Listen for advertisements
    do occasionally                     // and advertise my paths
        for each j in link_ids do       // to all of my neighbors.
            status ← SEND_PATH_VECTOR (j, my_addr, my_vector, exch_protocol)
            if status ≠ 0 then          // If the link was down,
                clear new_vector        // forget about any paths
                FLUSH_AND_REBUILD (j)   // that start with that link.

procedure HANDLE_ADVERTISEMENT (advt, link_id)   // Called when an advt arrives.
    addr[link_id] ← GET_SOURCE (advt)            // Extract neighbor's address
    neighbor_vector ← GET_PATH_VECTOR (advt)     //    and path vector.
    for each neighbor_vector.d_addr do           // Look for better paths.
        new_path ←{addr[link_id], neighbor_vector[d_addr]}      // Build potential path.
        if my_addr is not in new_path then       // Skip it if I'm in it.
            if my_vector[d_addr] = NULL) then    // Is it a new destination?
                my_vector[d_addr] ← new_path     // Yes, add this one.
            else                                 // Not new; if better, use it.
                my_vector[d_addr] ← SELECT_PATH (new_path, my_vector[d_addr])
    FLUSH_AND_REBUILD (link_id)

procedure SELECT_PATH (new, old)        // Decide if new path is better than old one.
    if first_hop(new) = first_hop(old) then return new  // Update any path we were
                                                        // already using.
    else if length(new) ≥ length(old) then return old   // We know a shorter path, keep
    else return new                                     // OK, the new one looks better.

procedure FLUSH_AND_REBUILD (link_id)   // Flush out stale paths from this neighbor.
    for each d_addr in my_vector
        if first_hop(my_vector[d_addr]) = addr[link_id] and new_vector[d_addr] = NULL
            then
            delete my_vector[d_addr]    // Delete paths that are no longer advertised.
    REBUILD_FORWARDING_TABLE (my_vector, addr)          // Pass info to forwarder.
```

路由算法的评价

1. **算法收敛吗？**
2. **如果收敛，收敛速度多快？**
3. **如果很快，对链路失效的更新也很快吗？**
4. **如果收敛，算法收敛前转发流量是安全的吗？**
   - 不安全的例子：loop
   - 应对loop的方法：hop limit。
5. **算法适用于多大的网络规模？**

进一步学习

安全用途

- 可用于实施路由限制策略（restrictive routing policie）-国家安全

路由算法与协议

- 路由协议复用在链路层上
- 因特网上同时运行多个路由协议
- 至少再自学一种其他路由算法，并与路径向量交换做比较

③ 层级结构

问题：大规模网络

1. G规模的唯一名称
2. G规模的路径向量

方案：层级结构（hierarchy）

- 分而治之
  - 命名
  - 寻址
- 网络地址变成：区域(region) + 站点(station)
  - 例：211.64.142.7 [ping一下试试看]

③ 层级结构

1.分配：

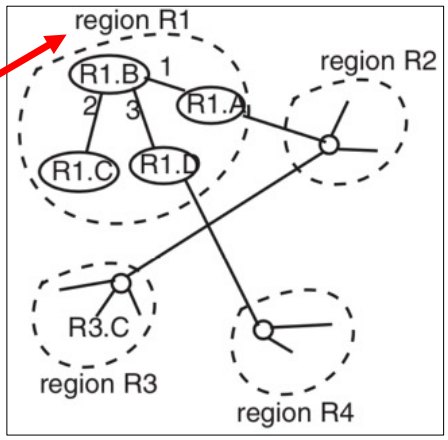
- 权威机构分配区域号，区域分配站点号
- ABCD四类区域号

2.转发：

- 区域至少需要1个边界跨区域节点
- 转发表区分内部和外部转发

3.矛盾：

- 区域划分与转发器设置



forwarding table in R1.B

region forwarding section	
to	link
R1	local
R2	1
R3	1
R4	3

local forwarding section	
to	link
R1.A	1
R1.B	end-layer
R1.C	2
R1.D	3

IP地址的不均衡与中国网络的发展

2019年，全球40多亿个IPv4地址全部分配完毕。

中国拥有343,881,984个，约占所有地址的8%。

但中国储备了全球最多的IPv6地址。

IPv6为中国互联网发展打开了一个新的创新空间……

但未来的发展，仍要靠技术获得话语权！

④ 错误管理

常见错误

1. 路由器缓冲区满
2. 目标不存在
3. 参数不能识别
4. 跳数超长（TTL到零）

为什么包损坏不报告?

是否应报告?

- 尽力而为的网络：不报告
- 工程折中：少量报告

网络层的错误管理
产生了端到端的服务
产生了端到端的监听

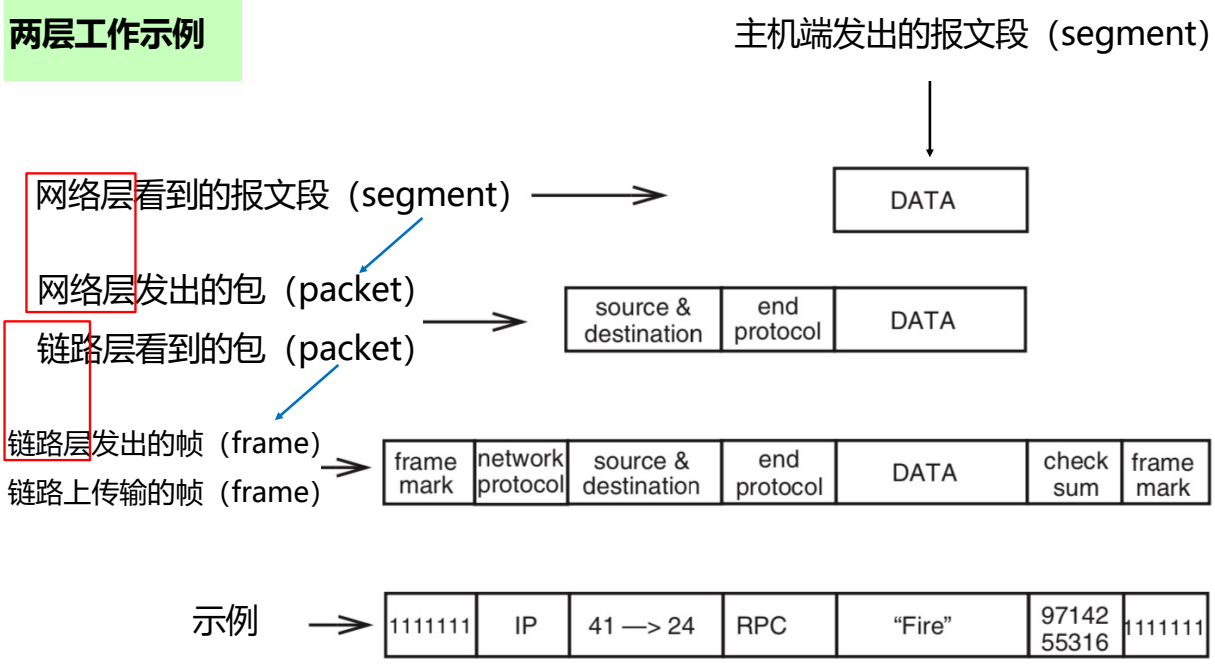
错误管理机制带来的设计机会

利用已有设计	获得额外收益
询问包	进行主机存活判断
跳数超限错误	探测途径点的地址（traceroute）
询问包的时延	进行网络拥塞判断
询问包的大小	发现MTU

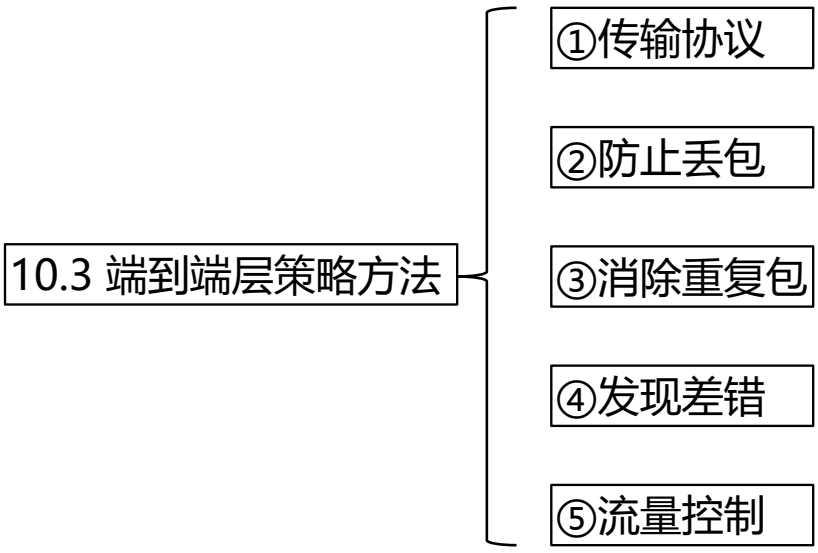
⑤ 网络地址转换（NAT）

- 现象观察：
  - 因特网地址分成多个层级
  - 层级内部可以有自己的上下文
  - 下层分配对上层是隐藏的
  - 区域的边界存在对内和对外两个方向的转发
- 设计机会：
  - 因特网地址只用于边界外；
  - 边界内使用内部地址
  - 在边界进行地址转换
- 设计收益：资源共用、隐藏拓扑、安全

两层工作示例



10.3 端到端层策略方法



尽力而为的网络 → 应用的各种需求

不保证：

1. 时间
2. 顺序
3. 正确性
4. 目的地
5. 成功

需要：

1. 可靠性
2. 性能
3. 安全性

- 特别是
  - **不同程度保障端到端双向传输**
- 传输协议（transport protocol）

① 传输协议

原语V1：非复用版本

- **send_message (destination, message)**
- upcall：deliver_message (message)

原语V2：服务复用版本

- **send_message (destination, service_port, message)**
  - 端口（port）是什么？对应了应用还是应用实例？

原语V3：请求复用版本

- **send_message (destination, service_port, reply_port, message)**

典型协议

UDP（User datagram protocol）

1. 复用：port
2. 校验：checksum

- 用于
  - 简单的请求/应答服务，如NTP、DNS
  - 应用自己构建的消息传输协议

TCP（Transmission control protocol）

RTP（Real-time transport protocol）

典型协议

UDP（User datagram protocol）

TCP（Transmission control protocol）

1. 顺序
2. 不丢包
3. 不重复
4. 适度完整性
5. 流量控制（flow control）

RTP（Real-time transport protocol）

典型协议

UDP（User datagram protocol)

TCP（Transmission control protocol)

RTP（Real-time transport protocol)

- 在UDP之上（也可看做插接层）
- (－)关闭校验
- (＋)增加时间戳
- 重视时间
- 容忍差错
- 丢弃乱序

如何学习更多协议?

RFC有最详细协议描述

应多关注原理而非细节

②防止丢包：计时器

重复发送请求直到收到确认

- 发送者
  1. 发送数据包，包含一次性随机数（nonce）
  2. 保存副本
  3. 设置计时器 > 往返时间（round-trip time，RTT）
- 接收者
  - 发送回应包，包含收到包中的nonce
- 发送者
  - 如收到回应，清计时器，丢弃副本
  - 计时器超时，重发

计时策略

固定计时

- 大于rtt

自适应计时

- 用rtt的指数移动加权平均（EWMA）代替rtt
- 改进：收到相同确认，立即增大计时器

否定应答（NAK）

- 接收者推算丢包，发出NAK
- NAK包含丢包的序号集合
  - 发包的nonce改为序列号

权衡：接收者怎么判定丢包?

讨论：还需要timer吗?

指数移动加权平均

动态序列值的平均值的一种（最常用）计算方法

$$A = \left(M_0 + M_1 \times \alpha + M_2 \times \alpha^2 + M_3 \times \alpha^3 + \ldots\right) \times (1-\alpha)$$

- α：衰减因子，α<1
- 下标/指数：表示时间
- 1-α：用作归一化，绝对值关系转为相对值关系

特点

- 旧的观测值权重迅速下降
- 易于计算

$$A_{new} \leftarrow (\alpha \times A_{old} + (1-\alpha) \times M_{new})$$

$$\frac{A_{new}}{(1-\alpha)} \leftarrow \left(\alpha \times \frac{A_{old}}{(1-\alpha)} + M_{new}\right)$$

③消除重复包

问题：防止丢包导致的副作用——重复包

方法：在接收端设置状态

- 问题：无穷增长的nonce列表
- 改进：发送包携带已收到确认的最大序号max
  - 接收端可丢弃max之下的所有nonce
- 问题：最后1个nonce?
  - 关闭端口，旧nonce可以安全抛弃
  - 接受不完美，根据可能重发次数n，设置n倍rtt为丢弃时间

④ 发现差错

保障数据完整性

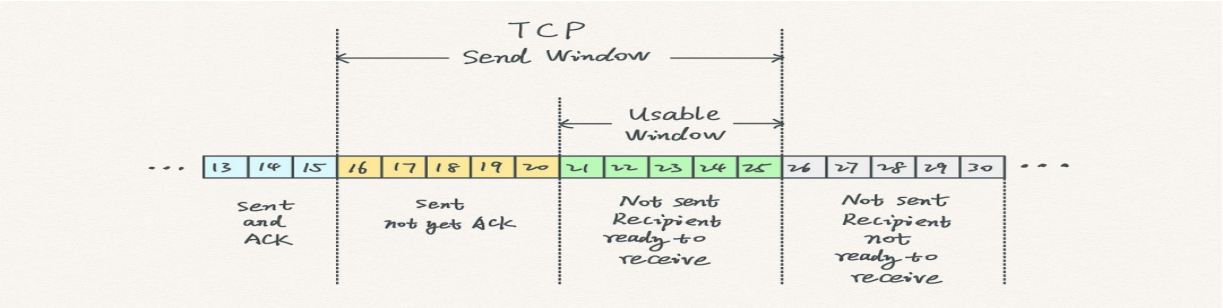
方法：校验和（checksum）= 不变量

- 讨论1：收到差错包，是否需要立即发送NAK?
  - 不发送，仅丢弃
  - 但链路层一般会立即告知，为什么?
- 讨论2：链路层有校验和，端到端层有无必要?
  - 错误层次不同
- 讨论3：校验和可靠吗
  - 算法与错误种类决定

⑤ 流量控制

连续发送问题

1. 不连续发送（lock-step，锁步）：性能低
2. 理想的连续发送（overlapping，交叠）：形成流水线
   - 性能带来复杂性（回顾第1章）
3. 实际方案：接收端避免瓶颈，设置“接收窗口”（window）



身份鉴别与机密性

TCP/IP协议的设计假设：善意环境

新假设：任何组件均可能有恶意

- 方法：
  - 签名与验证：sign & verify
  - 加密与解密：encrypt & decrypt

- 教训：错误的安全机制不如没有安全机制

- 第16章专题讲授

传输协议之外的端到端协议

建立在传输协议之上的（绝大多数）

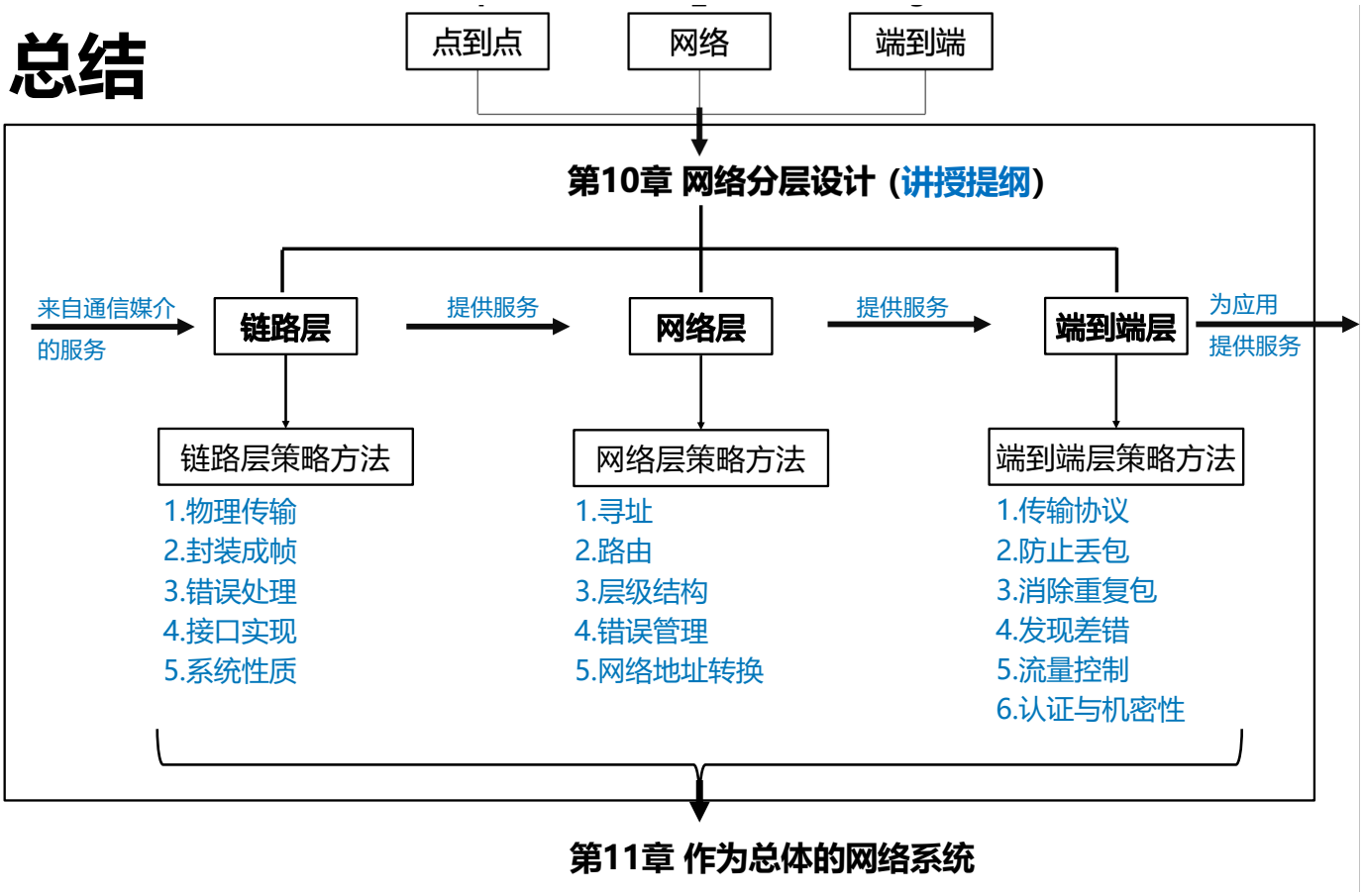
1. 文件传输、共享与访问：FTP、SMB、NFS
2. 远程过程调用（RPC）：SOAP、gRPC

- 这两种概念上归于表示层（presentation protocol）
  - 为什么？在不同平台进行数据格式转换

3. HTTP、SMTP、TLS、IMAP、SSH、DNS

建立在网络层协议之上的（少数）

- 路由协议：OSPF、RIP……
- 网络层错误报告协议：ICMP

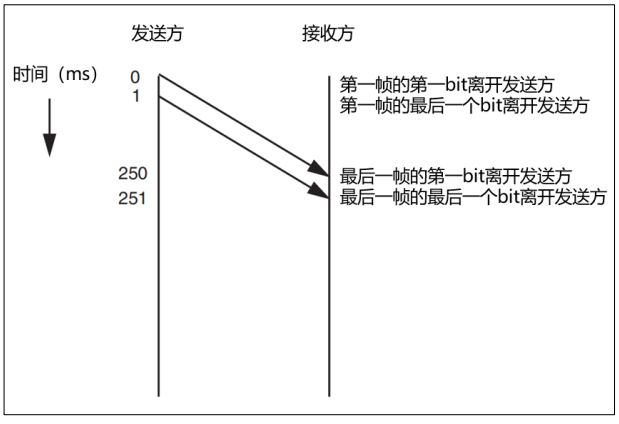


习题

- IBM的SNA协议族使用pacing流量控制机制。机制如下：发送者可发送一定数目的报文段，之后须暂停发送。接收者收到所有报文段后，发送pacing响应给发送者，发送者收到响应后可发送新的一组报文段。
- 假设将该方案用于卫星链路。地面站之间的时延是250毫秒。帧大小是1000比特，每次暂停前可发送4个报文段，卫星信道的数据率为1Mb/秒。

习题

- 时序图表示了第1个帧的发送



  - 问题
    - 1：填写接下来的6个帧。假设：没有帧丢失，延迟是统一的，发送方和接收方没有内部延迟
    - 2：卫星容量可用的最大比例是多少?
    - 3：如果想把信道利用率提高到50%，但是不能增加帧的大小，需要在响应前发送多少报文段才能实现?

重要术语中英文对照

封装成帧：framing

并行传输：parallel transmission

串行传输：serial transmission

压控振荡器：VCO

锁相环：PLL

相位编码：phase encoding

曼彻斯特编码：Manchester code

位错误率：BER（bit error rate）

位填充：bit stuffing

纠错码：error correction code

重要术语中英文对照

单工链路：simplex link

双工链路：duplex link

半双工链路：half-duplex link

全双工链路：full-duplex link

广播：broadcast

最大传输单元：MTU

端口：port

UDP协议：User datagram protocol

TCP协议：Transmission control protocol

RTP协议：Real-time transport protocol

重要术语中英文对照

协议：protocol

应用协议：application protocol

表示协议：presentation protocol

传输协议：transportation protocol

链路层：link layer

帧：frame

网络层：network layer

包/分组：packet

端到端层：end-to-end layer

报文段：segment

消息：message

流：stream

重要术语中英文对照

流量控制：flow control

往返时延：RTT

指数移动加权平均：exponentially weighted moving average

否定应答：negative acknowledgment

数据完整性：data integrity

衰减因子：decay factor

归一化：normalize

可靠交付协议：reliable delivery protocol

签名与验证：sign & verify

加密与解密：encrypt & decrypt

本章相关的参考文献

- Claude E. Shannon. A mathematical theory of communication. Bell System Technical Journal 27 (1948), 379–423 & 623–656.
- Clark D. The design philosophy of the DARPA Internet protocols[C]//Symposium proceedings on Communications architectures and protocols. 1988: 106-114.
- David D. Clark and David L. Tennenhouse. Architectural considerations for a new generation of protocols. ACM SIGCOMM ’91 Conference: Communications Architectures and Protocols, in Computer Communication Review 20, 4 (September 1990), 200–208.
- Vinton G. Cerf and Peter T. Kirstein. Issues in packet-network interconnec- tion. Proceedings of the IEEE 66, 11 (November 1978), 1386–1408.