清華大學

Tsinghua University

# 软件实验及理论背景详解

路由器实验团队

2024 年 10 月

# 目录

Contents

# 计算机如何上网?

- 快递如何送到家中?
- 网上购物时，需要填写收件人地址
- 卖家只需要知道收件人地址，就可以交给快递寄出
- 卖家和买家不需要知道快递是如何运输的，就可以享受快递带来的服务

# 计算机如何上网?

- 快递（网站内容）如何送到家（本地浏览器）中?
- 网上购物（浏览网站）时需要填写收件人地址（本地需要有 IP 地址）
- 卖家（网站服务器）只需要知道收件人地址（本地 IP 地址），就可以把快递（网站内容）寄出

# 计算机如何上网?

- 卖家（服务器）和买家（浏览器）不需要知道快递（网站内容）是如何运输的，就可以享受快递（网络）带来的服务
- 在网络里的 IP 地址就对应了现实世界里的地址
- 北京市海淀区清华大学紫荆公寓
- 北京市.海淀区.清华大学.紫荆公寓
- 紫荆公寓 2 号楼：59.66.131.*

# 地址分配

- 现实生活中，如何获得地址？
- 入学清华大学：获得清华大学的宿舍地址
- 出生在家里：从父母继承自己的地址
- 在网络世界中，如何获得 IP 地址？
- 连上清华的 Wi-Fi，由清华给你分配 IP 地址
- 连上家里的 Wi-Fi，由家里的路由器给你分配 IP 地址

# 地址的划分

- 下面每个地址都可以用于快递的收件地址：
  - 清华大学紫荆学生公寓
  - 清华大学紫荆学生公寓 2 号楼
  - 清华大学紫荆学生公寓 2 号楼 XXX A/B
- 但实际上快递都会送到紫荆 14 号楼后的快递站或者紫荆篮球场的快递柜，需要自己去取，不会送到宿舍楼下，也不会送到房间

# 地址的划分

- 负责清华片区的快递员只需要知道你住在紫荆还是南区，就知道要送到哪边的快递站/快递柜
- 不需要知道你住哪栋楼，叫什么名字
- 快递员需要看的：北京海淀清华大学紫荆公寓
- 快递员不需要看，留给快递站/快递柜鉴别身份的：姓名，取件号，手机号

# 路由

- 负责清华片区的快递员需要做的是：
  - 收取所有收件人地址属于清华的包裹
  - 根据收件人住址，分发到不同宿舍区的快递站/快递柜
- 写成匹配逻辑（假设清华划分为紫荆和南区）：
  - 北京/海淀/清华/紫荆/*: 送到紫荆的快递站/快递柜
  - 北京/海淀/清华/南区/*: 送到南区的快递站/快递柜
  - 其他：送错了，退回到海淀区的快递中转站

# 路由

- 负责海淀片区的快递员需要做的是：
  - 北京/海淀/清华/*：转交负责清华的快递员
  - 北京/海淀/北大/*：转交负责北大的快递员
  - 其他情况类似
  - 如果都没有匹配，说明不是海淀区的包裹，退回到北京的快递中转站

# 路由

- 快递员要做的事情，就是：
  - 根据一些预设的规则，对收件人地址进行匹配，决定把包裹转给哪一个快递员/快递站
  - 如果规则都不匹配，那就交给负责更大范围的快递员
- 对于最后一级的紫荆 14 号楼快递站/紫荆篮球场快递柜，它只需要等收件人本人来取件，不需要把包裹转给其他快递站/快递柜

# 路由

- 这就是路由，对地址进行前缀匹配，匹配时，可能的操作有：
  - 第一种情况，直达收件人，叫做直连路由
  - 第二种情况，非直达收件人，还需要转交给下一个快递员/快递站/快递柜，叫做间接路由
- 形式化“如果规则都不匹配”：设置一个通配的规则（*），无论什么地址都可以匹配

# 路由

- 清华片区快递员的路由：
  - 北京/海淀/清华/紫荆/*: 送到紫荆的快递站/快递柜
  - 北京/海淀/清华/南区/*: 送到南区的快递站/快递柜
  - *：送到海淀区的快递中转站
- 紫荆快递站的路由：
  - 北京/海淀/清华/紫荆/*：直连路由，直接给收件人本人
  - *：送到清华片区快递员

# 路由

- 收件没问题了，怎么寄件呢？
  - 广东省深圳市南山区清华大学深圳国际研究生院 -> 北京市海淀区清华大学
- 深研院的快递员拿到快递，能直接交给清华的快递员吗？
- 不能。怎么办？

# 默认路由

- 深研院的快递员的路由：
  - 广东省/深圳市/南山区/清华深研院/*：直连路由，直接交给收件人
  - *：送到南山区的快递员，称为默认路由
- 南山区快递员的默认路由：深圳市
- 深圳市快递员的默认路由：广东省
- 广东省快递员的默认路由：？

# 路由

- 广东省快递员收到送往北京的快递，应该：
  - 送往中国快递员
  - 送往华南片区快递员
  - 送往北京市快递员
- 答案是都可以，但是从路由来看，并不一样：
  - 送往中国/华南片区：送到更高一级，可以用默认路由
  - 送往北京：送到同一级，要给同一级的每个地方设置一个路由

# 路由

- 假如最高层级是中国，那么快递走的路径是：
- 深研院->南山区->深圳市->广东省->中国->北京市->海淀区->清华大学
- 那么中国快递员路由会是：
  - 北京/*：到北京快递员
  - 广东/*：到广东快递员
  - 依此类推

# 路由

- 假如最高层级是华南/华北，那么快递走的路径是：
- 深研院->南山区->深圳市->广东省->华南->华北->北京市->海淀区->清华大学
- 那么华南快递员路由会是：
  - 北京/*，河北/* 等等：到华北快递员
  - 广东/*：到广东快递员

# 路由

- 假如最高层级是广东/北京，那么快递走的路径是：
- 深研院->南山区->深圳市->广东省->北京市->海淀区->清华大学
- 那么广东快递员路由会是：
  - 北京/*：到北京快递员
  - 广西/*：到广西快递员

# 路由

- 那么哪种设计更好呢?
  - 从路由简单来说，最高级设到中国是最简单的，但是所有跨省包裹都要经过中国一级
  - 如果跨省包裹都是一步到位，那么路径会变短，但是全国每两个省之间都需要建立一个快递线路，并且每个省的路由条目会很多

# 路由

- 假如使用省间直达的方案，并且把国外考虑进来：
- 北京市的路由包括三类：
  - 北京市内的：中国/北京/海淀/*：到海淀区的快递站
  - 中国内各省的：中国/广东/*：到广东省的快递站
  - 剩下的到国外的：*：到海关
- 最长匹配原则：如果匹配多条规则，选择其中匹配长度最长的

# 路由

- 类似地，清华大学的快递员也可以设计这样三类路由：
  - 北京/海淀/清华/紫荆/*：到清华内的快递站
  - 北京/海淀/北大/*：到清华附近片区的快递站
  - *：到海淀区的快递站
- 此时从清华到北大的路径变短了：
  - 从 紫荆->清华->海淀->北大->未名湖 变成 紫荆->清华->北大->未名湖

# 路由

- 回归计算机网络，路由的查询对象从收件人地址变成了目的地的 IP 地址
- 如何查看本地的 IP 地址？
  - Windows：打开命令行提示符，运行 ipconfig /all
  - Linux：打开 Terminal，运行 ip addr
  - macOS：打开 Terminal，运行 ifconfig

# IPv4 地址

- IPv4 地址：a.b.c.d
  - 32 位分为四段，每一段 8 位，用十进制表示，用小数点连接（点分十进制）
  - 以网络字节序（大端序）保存为 32 位数
- 1.2.3.4 保存为 01 02 03 04（十六进制），在小端序机器上会解释为 0x04030201

# 字节序转换

- 主机字节序：小端序（常见）或大端序
- 网络字节序：大端序
- 主机->网络：htons（16 位）htonl（32 位）
- 网络->主机：ntohs（16 位）ntohl（32 位）
- 1.2.3.4 网络字节序 01 02 03 04（十六进制），在小端序机器上会解释为 0x04030201；如果想要得到 0x01020304，需要用 ntohl

# 公网 IPv4 地址

- 清华常见公网 IPv4 地址：
  - 59.66.x.x：如宿舍
  - 183.172.x.x，183.173.x.x：如无线网
  - 166.111.x.x：如计算机系
  - 101.5.x.x，101.6.x.x：如李兆基
- 也可以访问网站来看自己的公网 IP 地址
  - ipip.net
  - ip.chinaz.com

# IPv6 地址

- 把 IPv4 地址扩充到 128 位
- 每 16 位一段，每一段用 16 进制表示，用冒号连接，例子:
- 2001:0db8:85a3:0000:0000:8a2e:0370:7334
- 2001:db8:85a3::8a2e:370:7334
- 如果有连续的 16 位全为 0，可以忽略并改成 ::

# IPv6 地址

- fd00::1:0 对应
  - fd00:0000:0000:0000:0000:0000:0001:0000
  - fd 00 00 00 00 00 00 00 00 00 00 00 01 00 00
  - 没有字节序的问题：char [16]
- fe80::1 对应
  - fe80:0000:0000:0000:0000:0000:0000:0001

# 计算机网络中的路由

- 如何看本机的路由表?
  - Windows：命令行提示符中运行 route print
  - Linux：Terminal 中运行 ip route
  - macOS：Terminal 中运行 netstat -nr
- 输出可能会很多，可能需要往回翻才能看到 IPv4 的路由

# 计算机网络中的路由

- 真实世界的地址匹配：
  - 北京/海淀/清华/*
  - 实际上可以认为是把清华后面的部分清空，然后和 北京/海淀/清华 进行比较
  - 假设地址最多四段，那么地址 X 匹配上述规则等价于：
  - 把 X 最后一段地址改成空 == 北京/海淀/清华/空
- 这样做方便计算机处理 IP 地址的路由

# 计算机网络中的路由

- 计算机世界的地址匹配：
  - 假如要匹配 59.66.*.* 的 IPv4地址，那就等价于：
  - 地址 & 255.255.0.0 == 59.66.0.0
  - 注意 IPv4 地址就是一个 32 位数字，可以用按位与操作，把最后 16 位部分清零
  - 用十六进制表示：
    - 0x3B421234 & 0xFFFF0000 = 0x3B420000
  - 把 255.255.0.0 称为地址掩码（Netmask）

# 地址掩码

- 地址掩码：描述一个匹配规则的模糊/精确程度
  - 255.255.0.0 代表只匹配前 16 位
  - 255.0.0.0 代表只匹配前 8 位
  - 255.255.255.255 代表 32 位都要匹配，即精确匹配
  - 255.128.0.0（0xFF800000） 代表只匹配前 9 位

# 地址掩码

- 观察地址掩码的二进制表示：
  - 255.255.0.0: 11111111111111110000000000000000
  - 255.0.0.0: 11111111000000000000000000000000
  - 255.128.0.0: 11111111100000000000000000000000
- 二进制上，连续的 1 拼接连续的 0
- 因此地址掩码和它前缀 1 的个数一一对应
  - 习惯用 /16 表示 255.255.0.0，用 /8 表示 255.0.0.0

# 路由前缀

- 回顾匹配规则：59.66.x.x
  - 地址 & 255.255.0.0 == 59.66.0.0
- 地址掩码：255.255.0.0，又可以表示为 /16
- 那么匹配规则可以写成：59.66.0.0/16
- 如果只匹配一个 IPv4 地址：59.66.131.1/32
- 匹配所有 IPv4 地址：0.0.0.0/0，通常称为 default

# Windows 路由表

- Windows 系统的路由表：

```
IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0      192.168.0.1    192.168.0.72     25
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    331
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    331
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    331
     172.31.128.0    255.255.240.0         On-link      172.31.128.1   5256
     172.31.128.1  255.255.255.255         On-link      172.31.128.1   5256
   172.31.143.255  255.255.255.255         On-link      172.31.128.1   5256
      192.168.0.0    255.255.255.0         On-link      192.168.0.72    281
     192.168.0.72  255.255.255.255         On-link      192.168.0.72    281
    192.168.0.255  255.255.255.255         On-link      192.168.0.72    281
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    331
        224.0.0.0        240.0.0.0         On-link      192.168.0.72    281
        224.0.0.0        240.0.0.0         On-link      172.31.128.1   5256
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    331
  255.255.255.255  255.255.255.255         On-link      192.168.0.72    281
  255.255.255.255  255.255.255.255         On-link      172.31.128.1   5256
===========================================================================
```

# Windows 路由表

- 路由表的前三列：
- Network Destination：192.168.0.0
- Netmask：255.255.255.0
- Gateway：On-link
- 表示一条 192.168.0.0/24 的直连（On-link）路由
- 规则：如果匹配 192.168.0.0/24，则直接访问

# Windows 路由表

- 路由表的前三列：
- Network Destination：0.0.0.0
- Netmask：0.0.0.0
- Gateway：192.168.0.1
- 表示一条 0.0.0.0/0 的路由，网关是 192.168.0.1，是间接路由中的默认（0.0.0.0/0）路由
- 默认路由的网关也被叫做默认网关

# Linux 路由表

- Linux 系统的路由表：

```
default via 192.168.111.1 dev wlan0 proto dhcp src 192.168.111.97 metric 600
169.254.0.0/16 dev vmbr0 scope link metric 1000
192.168.0.0/24 dev vmbr0 proto kernel scope link src 192.168.0.1
192.168.1.0/24 dev vmbr1 proto kernel scope link src 192.168.1.1
192.168.111.0/24 dev wlan0 proto dhcp scope link metric 600
192.168.111.0/24 dev wlan0 proto kernel scope link src 192.168.111.97 metric 600
```

# Linux 路由表

- default via 192.168.111.1 dev wlan0
- default：0.0.0.0/0，表示默认路由
- 192.168.111.1：下一跳地址，或者叫网关
- dev wlan0：通过无线网卡访问
- 默认规则：通过无线网卡，发送数据给 192.168.111.1

# Linux 路由表

- 192.168.0.0/24 dev vmbr0
- 192.168.0.0/24：192.168.0.x
- 没有 via + IP 地址：说明是直连路由
- 规则：如果匹配 192.168.0.0/24，则直接访问

# macOS 路由表

- macOS 系统的路由表：

```
Internet:
Destination        Gateway            Flags           Netif Expire
default            192.168.1.1        UGScg             en0
127                127.0.0.1          UCS               lo0
127.0.0.1          127.0.0.1          UH                lo0
169.254            link#15            UCS               en0      !
169.254.169.254    link#15            UHLSW             en0      !
192.168.1          link#15            UCS               en0      !
192.168.1.1/32     link#15            UCS               en0      !
192.168.1.1        f4:e4:d7:d2:e3:c0  UHLWIir           en0   1163
192.168.1.2/32     link#15            UCS               en0      !
192.168.1.5        de:23:c4:8e:b2:6e  UHLWIi            en0    661
192.168.123        utun0              Uc              utun0
224.0.0/4          link#15            UmCS              en0      !
224.0.0.251        1:0:5e:0:0:fb      UHmLWI            en0
239.255.255.250    1:0:5e:7f:ff:fa    UHmLWI            en0
255.255.255.255/32 link#15            UCS               en0      !
```

# macOS 路由表

- Destination: default，也就是 0.0.0.0/0
- Gateway：192.168.1.1
- Netif：en0（无线网卡）
- 默认路由，通过 en0（无线网卡）发给网关 192.168.1.1

# macOS 路由表

- Destination：192.168.1，只写了三段，表示的是 192.168.1.x，也就是 192.168.1.0/24
- Gateway：link#15，表示直连路由
- Netif：en0
- 规则：如果匹配 192.168.1.0/24 ，直接访问

# 路由表小结

- 路由表由多条路由组成，每条路由至少包括：
  - 前缀：a.b.c.d/len，或者 a.b.c.d 和 netmask 的组合
  - 网关/下一跳 IP 地址：a.b.c.d，如果没有则是直连
  - 网络接口：通过哪个物理网卡或者虚拟网卡发送
- 对应到真实世界：
  - 前缀：匹配的地址规则，如北京/海淀/清华/*
  - 网关：要转交的下一个快递员
  - 网络接口：要走哪条路去送快递

# 路由表的生成

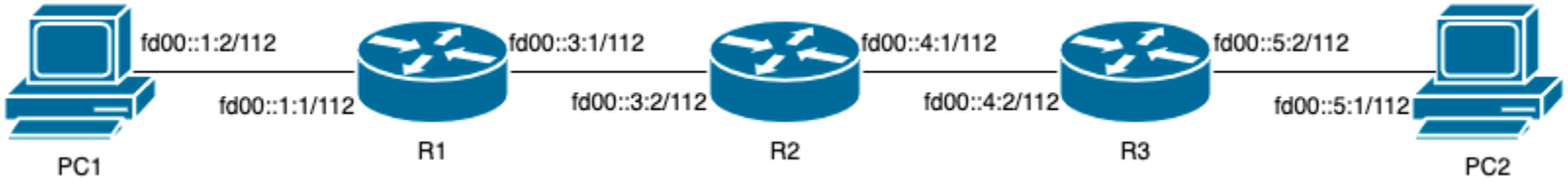
- 如何生成路由表?
  - 法一：人手动设置
  - 法二：对于终端来讲，只需要默认路由和直连路由
  - 法三：自动发现其他路由器，并计算路由表
- 终端：获取 IP 地址和默认网关即可生成路由表
  - 默认路由：0.0.0.0/0 via 默认网关
  - 直连路由：192.168.0.0/24

# 路由协议

- 能否自动生成路由表?
- 理想情况：把网络设备都连在一起，就可以互联互通（IP 地址和直连路由需要提前设好）
- 现实：网络设备能直接访问的只有邻居，也就是直连路由所能访问的设备范围

# 路由协议

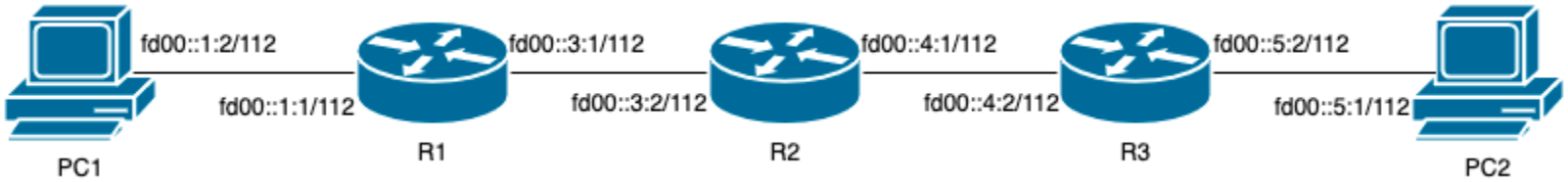
- 初始状态：每个网络设备配置好了 IP 地址和直连路由




- 例如 R1 的路由表有：
  - fd00::1:0/112 dev r1pc1（r1pc1 表示 R1 到 PC1）
  - fd00::3:0/112 dev r1r2（r1r2 表示 R1 到 R2）
  - 但并不知道 R3 和 PC2 的存在，也不知道如何访问

# 路由协议

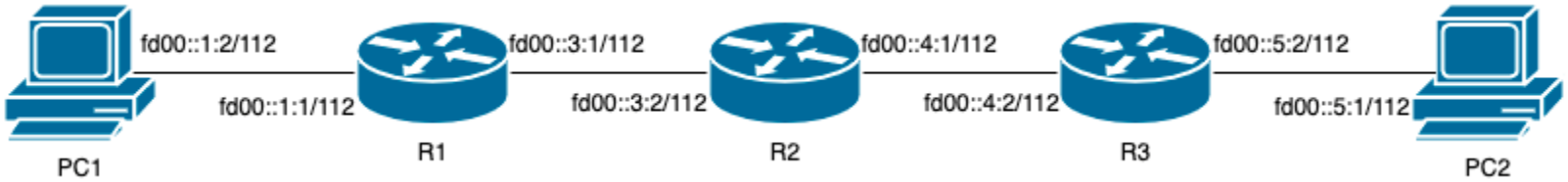
- 如何把路由信息传播到整个网络?




- R1：我有 fd00::1:0/112 和 fd00::3:0/112 的路由
- R2：我有 fd00::3:0/112 和 fd00::4:0/112 的路由
- R1 收到了 R2 发送的信息，学习到 fd00::4:0/112
- R2 收到了 R1 发送的信息，学习到 fd00::1:0/112

# 路由协议

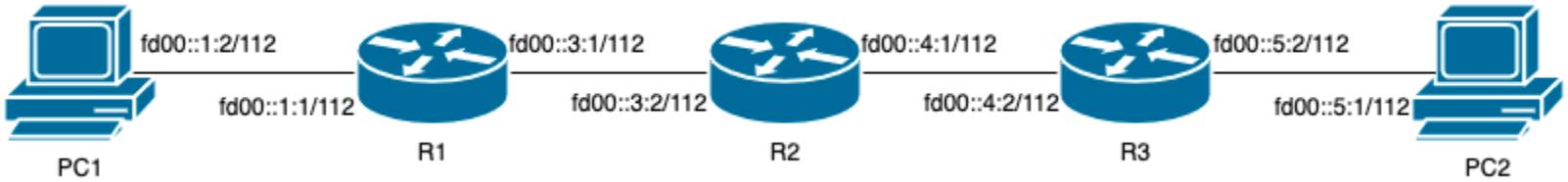
- 如何把路由信息传播到整个网络?




- R2：我有 fd00::1:0/112、fd00::3:0/112 和 fd00::4:0/112 的路由
- R3 收到 R2 发送的信息，学习到 fd00::1:0/112 和 fd00::3:0/112 的路由，知道如何访问原来无法访问的 PC1 和 R1

# 路由协议

- 通过不断地传播，整个网络实现了互联




- 进一步：为了更快到达目的地，能否找到一条最短路径?
- 回忆数据结构课上讲的单源最短路径算法
  - Bellman-Ford 算法：对应 RIPng 路由协议
  - Dijkstra 算法：对应 OSPF 路由协议

# 路由协议

- 软件实验第二、第三阶段中的两个选项
  - RIPng 协议：距离向量路由协议，Bellman-Ford 算法
  - OSPF 协议：链路状态路由协议，Dijkstra 算法
- 推荐选择 OSPF 协议，通过实验帮助理解路由协议中最广泛使用的链路状态路由协议

# 公网地址 vs 内网地址

- 公网地址：类比，快递上要填写，保证可以送到你的地址
  - 清华大学紫荆学生公寓
- 内网地址：内部用的地址，快递员不需要知道，也可以送到目的地
  - 几号床位
- 那么紫荆几号楼哪个房间，属于公网还是内网？

# 地址分配

- 连上校园网后，如何分配到 IPv4/IPv6 地址?
- IPv4 地址通过 DHCP 协议分配，IPv6 地址通过 DHCPv6 等协议分配
- 清华公网 IP 比较多，因此能够分配公网 IP 地址
- 家庭带宽通常会分到内网 IP 地址，此时看到的公网 IP 地址是运营商设备的公网 IP 地址，中间采用了 NAT 技术

# 动态地址分配

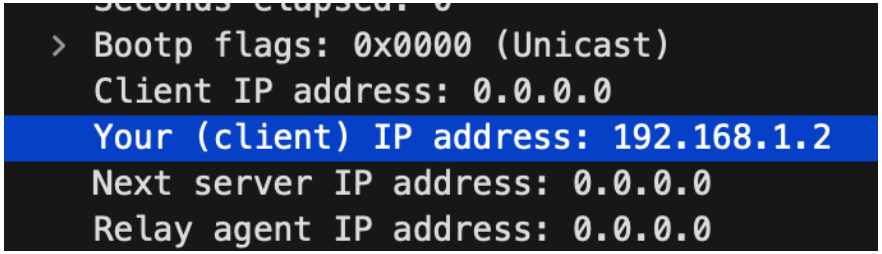
- DHCP 协议动态分配 IPv4 地址
- DHCPv6 协议动态分配 IPv6 地址
- 实验：
  - 安装 wireshark
  - 监听 Wi-Fi，过滤条件输入：dhcp or dhcpv6 回车
  - 断开无线网再连接，或者在设置里 Renew DHCP

# DHCP 协议

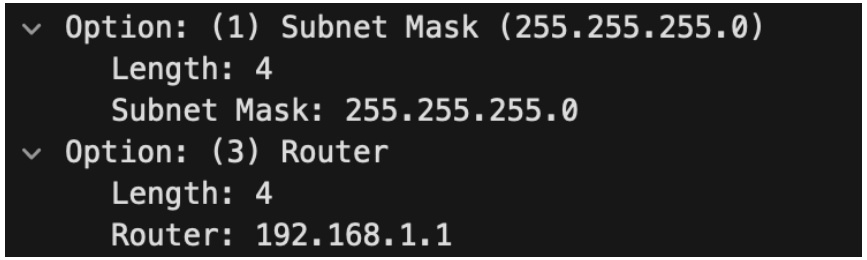
```
Seconds elapsed: 0
> Bootp flags: 0x0000 (Unicast)
  Client IP address: 0.0.0.0
  Your (client) IP address: 192.168.1.2
  Next server IP address: 0.0.0.0
  Relay agent IP address: 0.0.0.0
```

```
v Option: (1) Subnet Mask (255.255.255.0)
    Length: 4
    Subnet Mask: 255.255.255.0
v Option: (3) Router
    Length: 4
    Router: 192.168.1.1
```

# DHCP 协议




- 分配到 192.168.1.2
- 默认网关 192.168.1.1
- 网络掩码 255.255.255.0
- 对应的路由表：
  - 0.0.0.0/0 via 192.168.1.1 dev wlan0 默认路由
  - 192.168.1.0/24 dev wlan0 直连路由



# DHCPv6 协议

- DHCPv6 协议用于动态分配 IPv6 地址
- 软件实验第二、三阶段的选项之一
- 实现一份 DHCPv6 协议的路由器

# TFTP 协议

- 除了上文所述的三个实现选项：DHCPv6，RIPng 和 OSPF，还提供了 TFTP 的选项
- TFTP 是一个简单的文件传输协议，实现在客户端和服务端之间的文件传输
- 在应用层实现类似课上讲的数据链路层协议
  - 联系作业一 End to End Argument：课上为啥在数据链路层上讲，实际又在应用层上实现

# 软件实验

- 第一阶段：编程作业
- 第二阶段：真机评测（单人）
- 第三阶段：真机评测（互联）

# 实验框架

- 登录 TANLabs 实验平台，平台自动创建实验仓库
  - 平台将自动导入实验框架到实验仓库
- 编程作业为 Homework 目录下的四个题目
  - `eui64`：生成 EUI-64 格式的 IPv6 Link Local 地址
  - `internet-checksum`：校验和计算和检验
  - `lookup`：路由表维护、查询
  - `protocol`：RIPng 或 OSPF 报文解析
  - 实验者可暂不关注其他目录的代码
  - 在题目各自目录下运行 `make` 编译，运行 `make grade` 进行本地评测

# 实验内容

- 每个题目目录下的 `README.md` 描述了题目的要求和实现方法
- 代码（包括头文件）中的注释也需要阅读
- 不支持 Windows，请用 WSL（建议 WSL2）
- 注意如何读写预定义的结构化数据
  - 注意题目中提示的各个结构体

# 实验内容：eui64

- 给定一个 MAC 地址，采用 EUI-64 规则生成对应的 IPv6 Link Local 地址
- 尝试理解 MAC 地址和 IPv6 地址结构体的内容
- 接触 IPv6 地址的表示方法
- 阅读 RFC 4291

# 实验内容：internet-checksum

- 给定一个 IPv6 分组，检查校验和是否正确并计算出正确的校验和
- 用途
  - 接收 IPv6 分组时检查校验和是否正确
  - 发送 IPv6 分组时计算出正确的校验和
- 注意 IPv6 Pseudo Header 的处理
  - 长度的端序是什么？提示：可以用 ntohl htonl 函数
- 阅读 RFC 1071

# 实验内容：lookup

- 维护一个路由表，实现基本的查询和更新功能
- 查询算法为最长前缀匹配
  - 可能有多个路由条目匹配目标 IPv6 地址，算法需要选取其中前缀长度最长的
- 更新算法为精确匹配
  - 插入、更新或删除路由条目
- 本次作业不对性能做过多要求
- 可参考的实现方法：线性查找、字典树等

# 实验内容：protocol

- 解析 RIPng 或 OSPF 报文，二选一
- 检查各种常见的错误
- 注意部分字段取值固定
- 使用 Wireshark 打开生成的 pcap 文件观察，有助于调试
- 阅读 RFC 2080（RIPng）或 RFC 5340（OSPFv3）
- 注意 OSPF 版本的作业在 protocol-ospf 目录下

# 实验平台

- 实验平台 TANLabs 地址：

  https://lab.cs.tsinghua.edu.cn/tan

- 实验文档地址：

  https://lab.cs.tsinghua.edu.cn/router/doc

- 访问实验平台后会跳转到 `git.tsinghua.edu.cn`
- 签署 honor code 后，平台自动创建实验仓库
- 实验者需要
  - 将实验仓库克隆到本地，修改后通过 Git Push 上传
  - 在 TANLabs 上查看评测结果并标记最终结果

# 实验要求

- 学术道德
  - 参考网上代码请注明出处
  - 横向（同学代码）+纵向（往届代码）查重
  - **严禁抄袭！抄袭被认定后实验计零分！**
- 务必阅读题目中提到的 RFC 文档
- 第一阶段截止时间：UTC+8 第十一周周日（11 月 24 日）**晚上 10 点整**
  - 请在截止时间前完成代码提交、在线评测并标记最终结果

# 实验目标

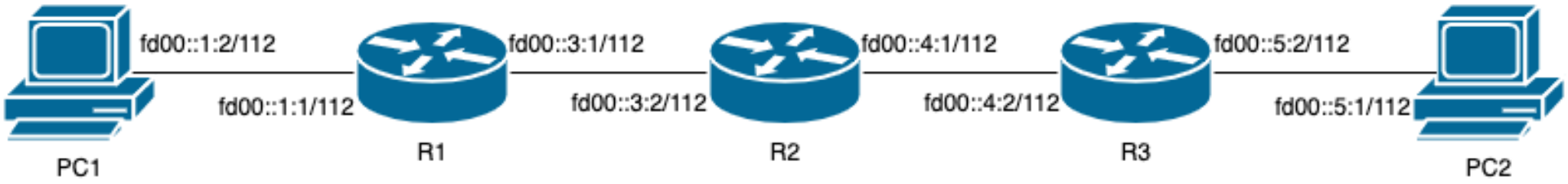
- 第二、三阶段从以下四个选项中选择一个完成
  - RIPng 协议的路由器
  - OSPF 协议的路由器
  - DHCPv6 协议的路由器
  - TFTP 协议的客户端和服务端
- 第二阶段：与已有的成熟软件进行测试
- 第三阶段：与其他八名同学的软件进行测试
- 学术道德
  - 参考网上代码请注明出处
  - 横向（同学代码）+纵向（往届代码）查重
  - **严禁抄袭！抄袭被认定后实验计零分！**

# 实验内容

- **真机评测（个人+互联各 40% 分数）**
  - 个人：第十二周到第十四周 2024.12.15
  - 互联：第十五周到第十六周 2024.12.29
  - 截止日期都是**当周周日北京时间晚上 10 点整**
  - 在云端**真实硬件**上运行和测试
  - 个人：与成熟的标准实现进行测试
  - 互联：与其他八名同学的实现进行测试
- **实验前请更新代码仓库！**
- **第一阶段实验第 11 周周日（11.24）晚 22 点截止**

# 实验内容

- 选项一、二：实现 RIPng 或 OSPF 协议的路由器
  - 初始时，R1、R2 和 R3 都只有自身的直连路由
    - 如 R1 有 fd00::1:0/112 dev r1pc1 路由
    - 表示目标 IP 地址可达
  - 目标：R1 获得到 PC2 的路由、R3 获得到 PC1 的路由
    - 比如 R1 学习到 fd00::5:0/112 via fd00::3:2 dev r1r2
  - 表现：PC1 可以访问 PC2

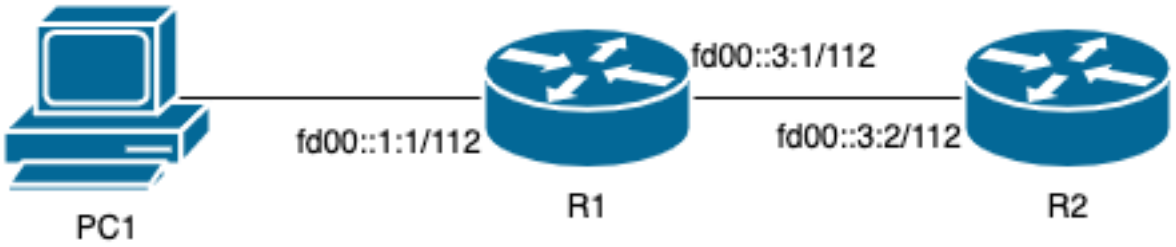


# 实验内容

- 选项一、二：实现 RIPng 或 OSPF 协议的路由器
  - 评测内容：
    - 网络中的设备能否互相访问（Ping，TCP）
    - 路由器的路由表是否正确
    - ICMP 处理是否正确（Hop Limit，Echo Reply）
    - 转发性能
    - 见实验文档
  - 如何调试：按照文档讲述的流程，在本地启动一个虚拟的网络环境，在其中重现评测的流程

# 实验内容

- 选项三：实现 DHCPv6 协议的路由器
  - 初始时，PC1 没有 IPv6 地址
  - R1 负责给 PC1 分配动态的 IPv6 地址
  - PC1 上运行 DHCPv6 客户端
  - 目标：PC1 获得 IPv6 地址和默认路由
    - PC1 得到 IPv6 地址 fd00::1:2/112，默认网关 fd00::1:1
  - 表现：PC1 可以访问 R2

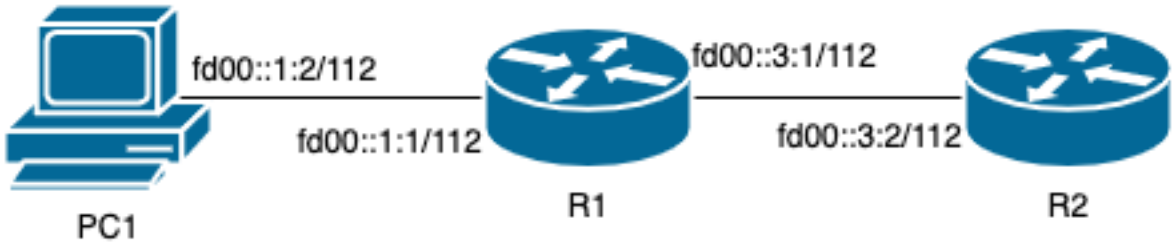


# 实验内容

- 选项三：实现 DHCPv6 协议的路由器
  - 评测内容：
    - 是否正确响应 ICMPv6 RS
    - 能否正确通过 DHCPv6 协议分配 IPv6 地址
    - PC1 能否访问 R2（Ping）
    - 转发性能
    - 见实验文档
  - 如何调试：按照文档讲述的流程，在本地启动一个虚拟的网络环境，在其中重现评测的流程

# 实验内容

- 选项四：实现 TFTP 协议的客户端和服务端
  - TFTP 协议是基于 UDP 的文件传输协议
  - 带有简单的重传机制
  - 目标：
    - 客户端可以从服务端下载文件
    - 客户端可以向服务端上传文件
  - 客户端运行在 PC1，服务端运行在 R2



# 实验内容

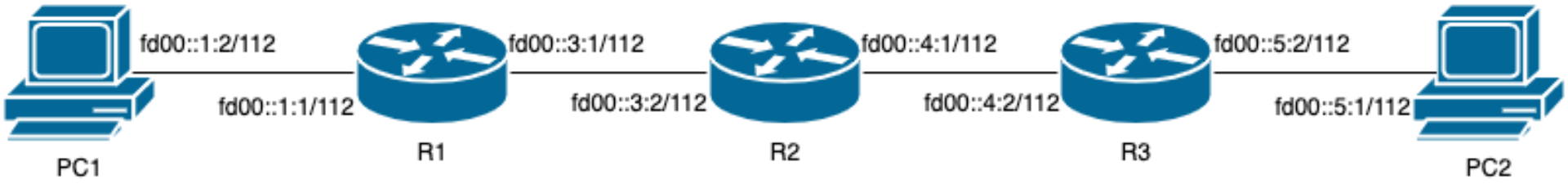
- 选项四：实现 TFTP 协议的客户端和服务端
  - 评测内容：
    - 自己的客户端能否访问标准服务端
    - 标准客户端能否访问自己的服务端
    - 自己的客户端能否访问自己的服务端
    - 文件传输性能
    - 见实验文档
  - 如何调试：按照文档讲述的流程，在本地启动一个虚拟的网络环境，在其中重现评测的流程

# 实验内容

- 四个实验选项的对比：
  - OSPF：理解难度最大，代码量最小
  - RIPng：理解难度较大，代码量较小
  - DHCPv6：理解难度中等，代码量中等
  - TFTP：理解难度最小，代码量最大
- 三个选项都有什么收获：
  - RIPng 或 OSPF：学习路由协议，了解当今互联网是如何工作的
  - DHCPv6：理解手机电脑是如何连上互联网的
  - TFTP：实现一个简单的文件传输协议，实践重传协议，帮助未来完成计算机网络专题训练课程的实验

# 实验内容

- 第三阶段：互联测试
  - PC1: 运行第一名同学的 TFTP 客户端
  - R1: 运行第二名同学的 DHCPv6 路由器
  - R2: 运行第三名同学的 RIPng 或 OSPF 路由器
  - R3: 运行第四名同学的 RIPng 或 OSPF 路由器
  - PC2: 运行第五名同学的 TFTP 服务器
- 挑选其他四名同学进行评测，最终选择人员不重合的同一份代码的两次评测结果

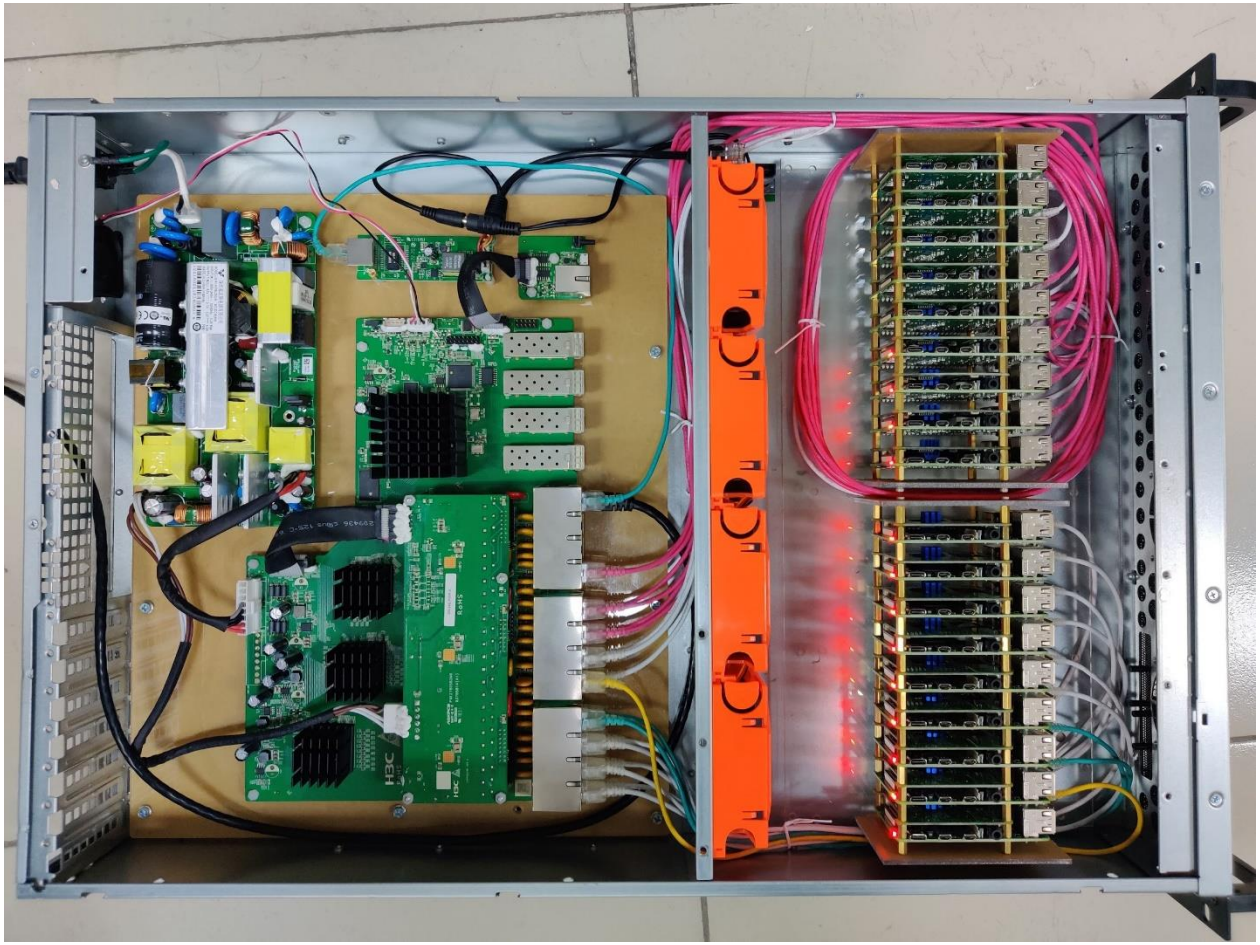


# 实验平台

- 清华高级网络实验平台（TANLabs）
  - https://lab.cs.tsinghua.edu.cn/tan
  - 在线进行个人/互联评测
  - 同样需要标记 master 分支上的最终评测
  - 每次评测的性能结果可能有 5% 左右的波动
  - 同学可以多次尝试提交最好的一次
  - 但注意评测资源也是有限的，不要交太多次
  - **重申：学术道德**
  - **实验前请更新代码仓库！！！**

# 实验平台

- 一个实验节点（交换机+18x树莓派）：



# 实验文档

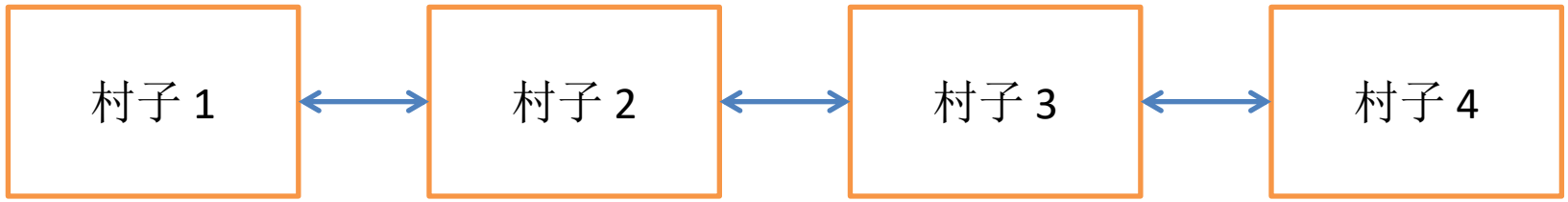
- 实验文档涵盖了实验的所有信息
  - https://lab.cs.tsinghua.edu.cn/router/doc
  - 在线评测的各个环节
  - 如何搭建本地的评测环境
  - Linux 网络的配置方法
  - 常见的错误
  - 路由器的调试方法
- Read before you ask anything!

# 查重与抄袭认定

- 在实验平台上创建仓库前，请仔细阅读诚信守则
- 不可以抄代码
- 不要给别人发代码
- 不可以用 AI 补全代码，但可用于辅助
- 所有代码或 AI 辅助参考应当按照格式写在代码注释或 HONOR CODE 文件中
- 完整版见实验文档，实验前必读

# 交换机 vs 路由器

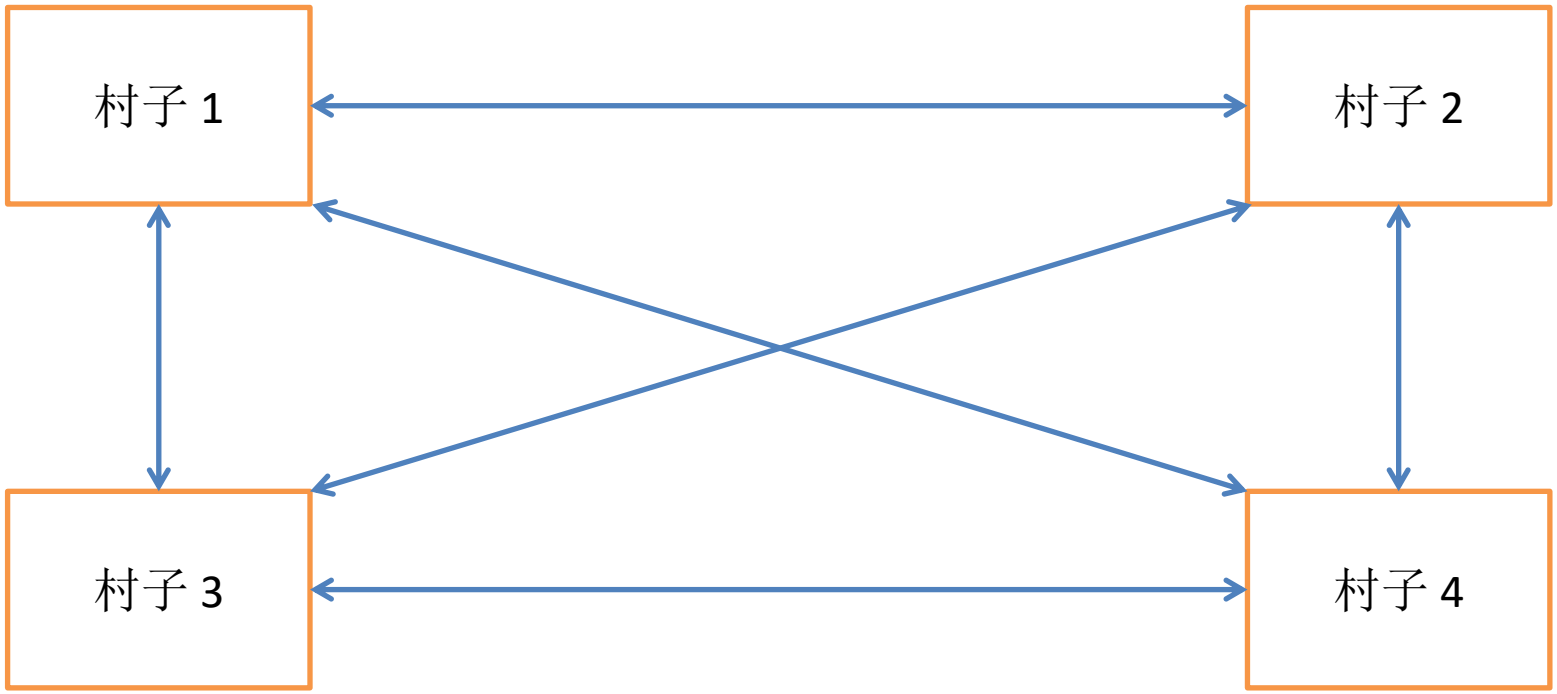
- 多个村子要修路连接起来，怎么修?
- 第一种方法:




- 把村子串联起来，村子 1 要到村子 4，需要经过村子 2 和 村子 3。
- 但假如每次进出村子都要给过路费（转发开销），怎么办?

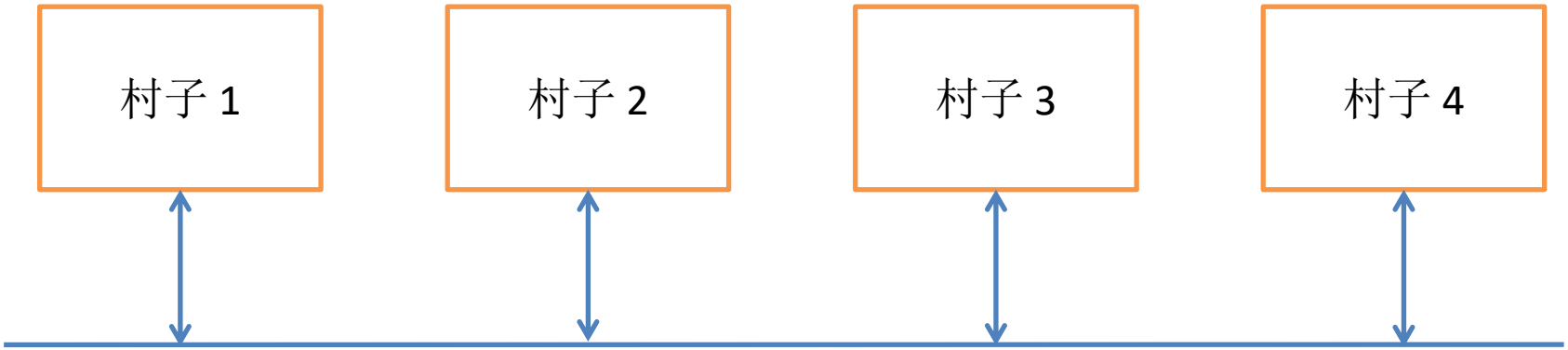
# 交换机 vs 路由器

- 第二种方法：村子之间两两全相连，道路太多



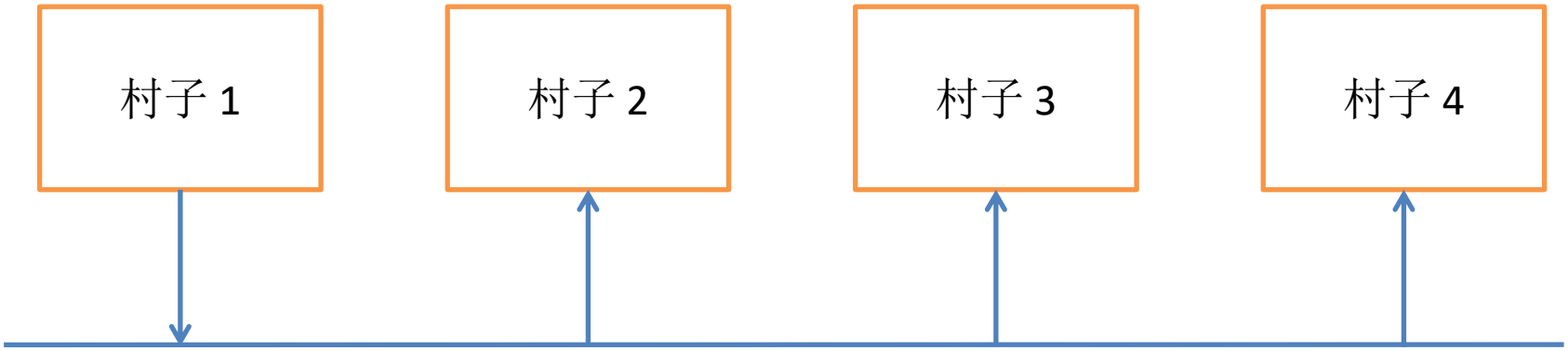
# 交换机 vs 路由器

- 第三种方法：
- 设置一段公共的道路，每个村子都连接到公共道路上，不需要经过其他村子也可以达到目的地



# 交换机 vs 路由器

- 但在网络里面，道路就是网线。在网线上发送的数据，所有连接到该网线上的设备都会收到
- 村子 1 发送的，村子 2-4 都会收到

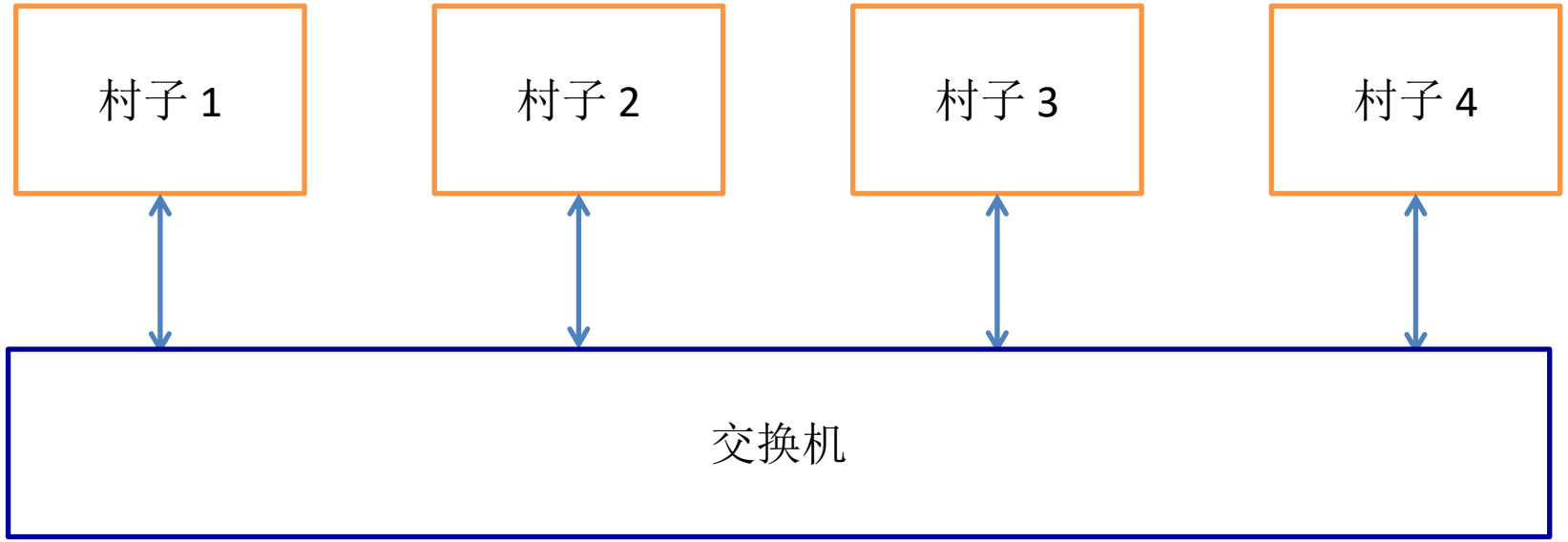


# 交换机 vs 路由器

- 共享介质下，来自不同村子的车流量会冲突，如何优化?

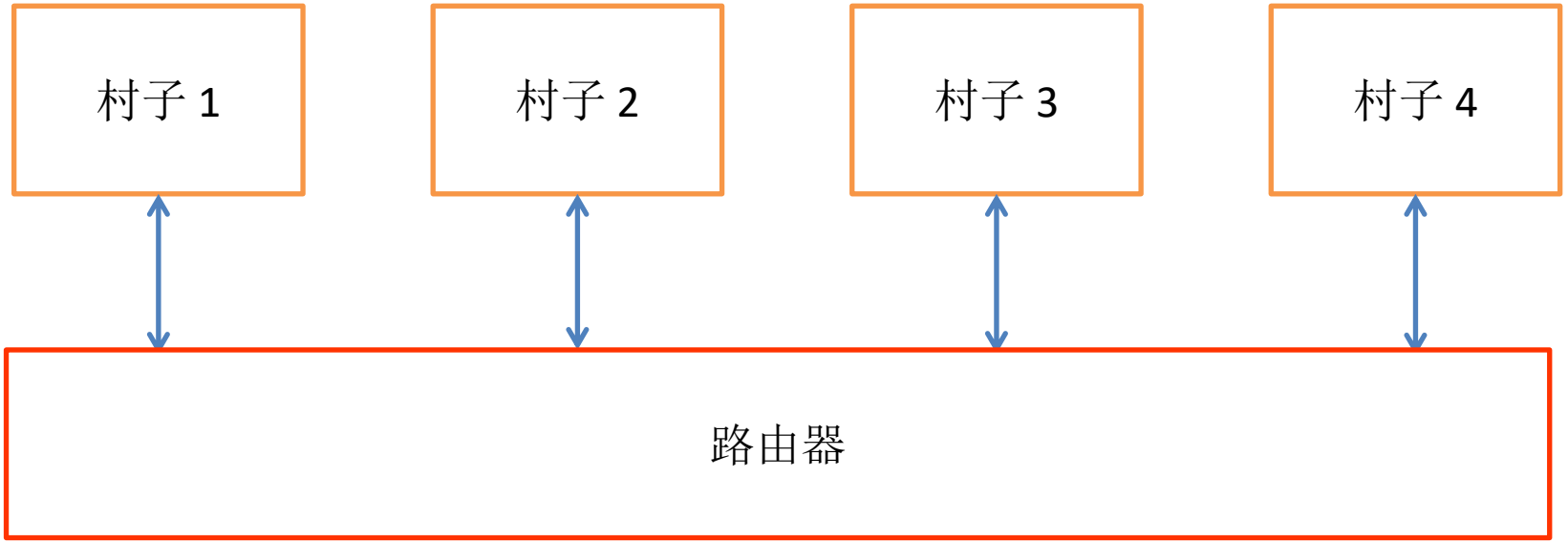
# 交换机 vs 路由器

- 引入交换机作为“交警”：只负责引导车流（流量），工作量比较小，不需要收过路费（开销），对村子来说是透明的



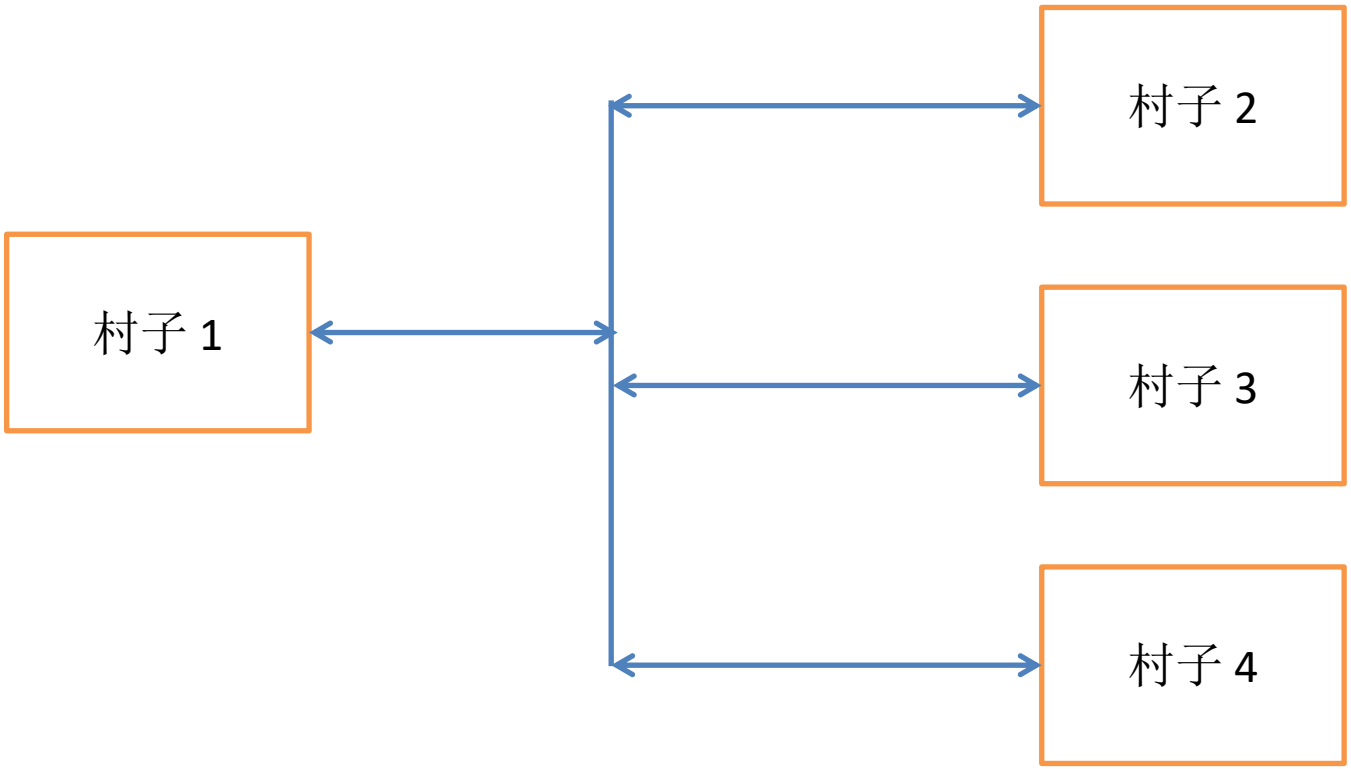
# 交换机 vs 路由器

- 那么能否用路由器做交警呢？可以，但是需要收过路费（转发开销），对村子来说是不透明的，路由器就像是一个只用来过路的村子



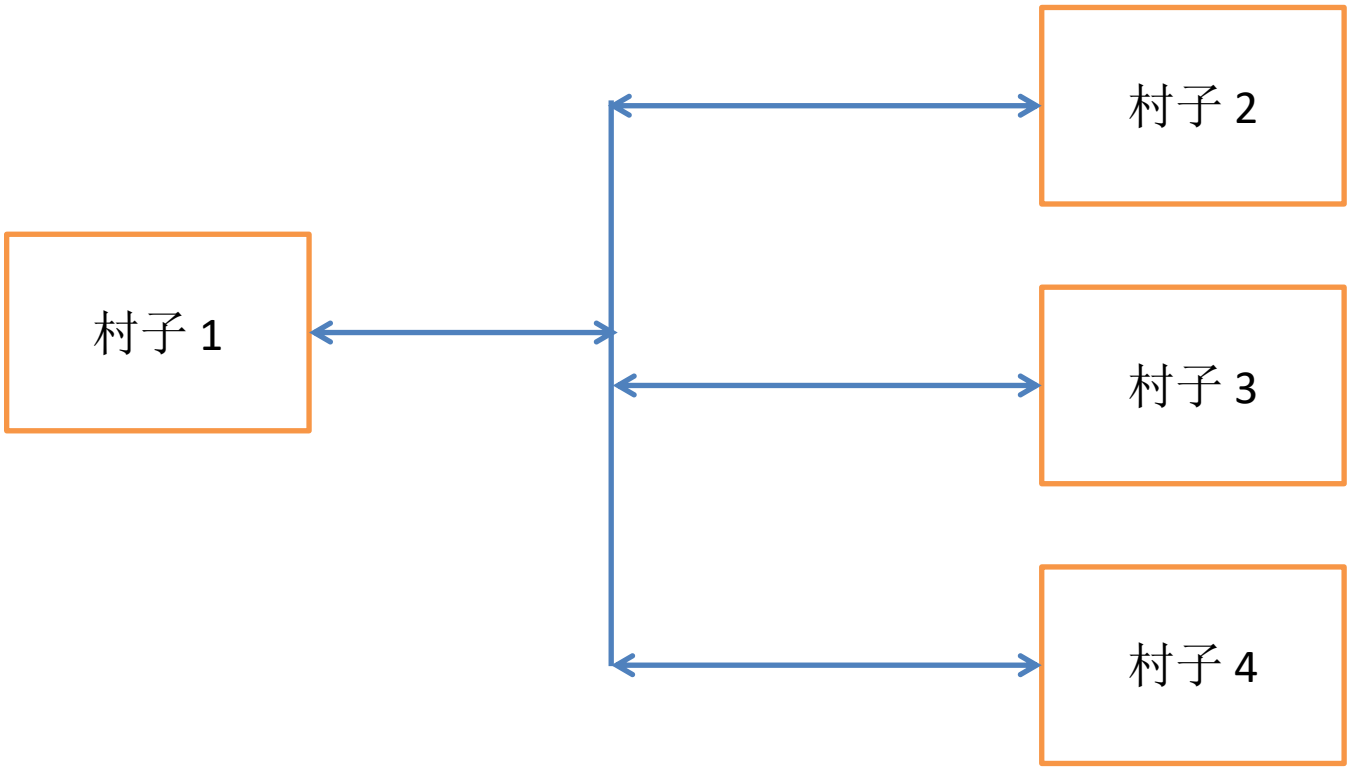
# 交换机 vs 路由器

- 不使用交换机或路由器时，村子 1 看到的视角：



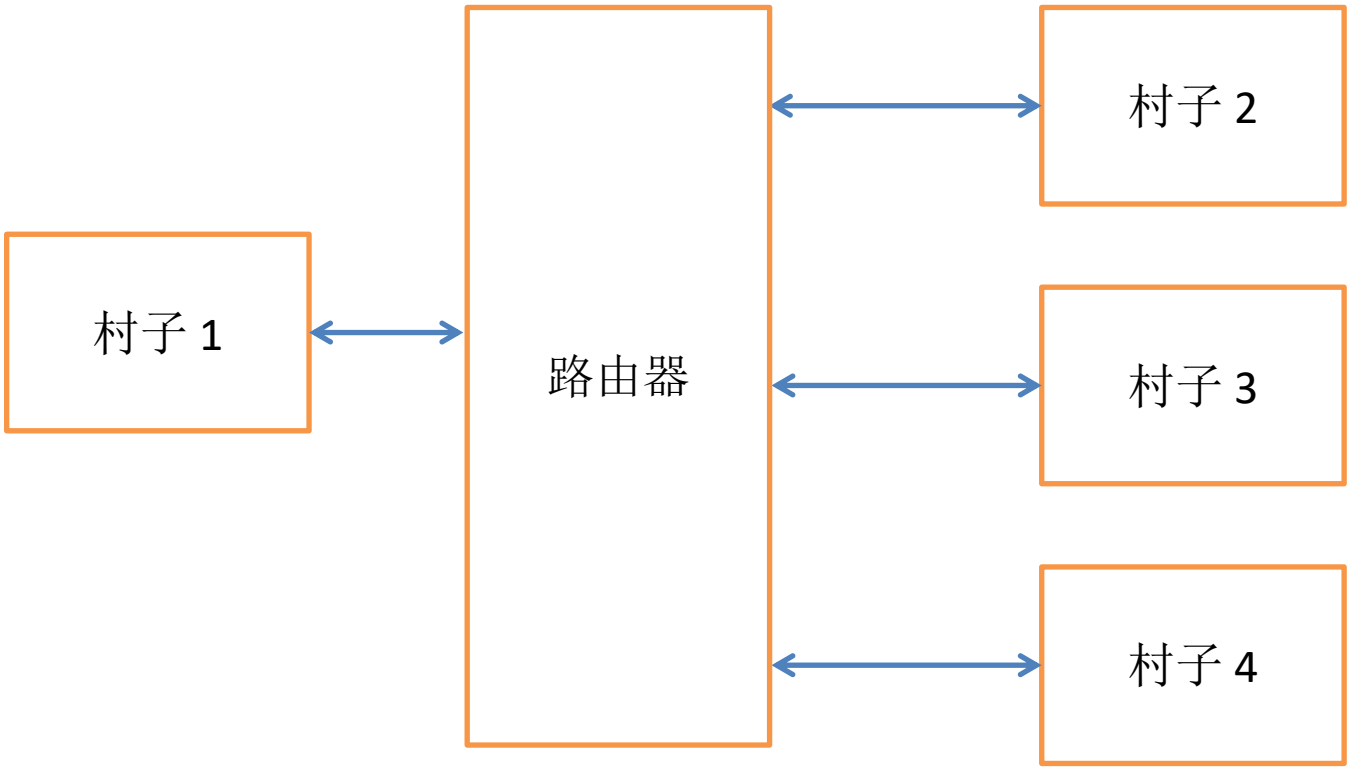
# 交换机 vs 路由器

- 引入交换机后，村子 1 看到的视角不变：



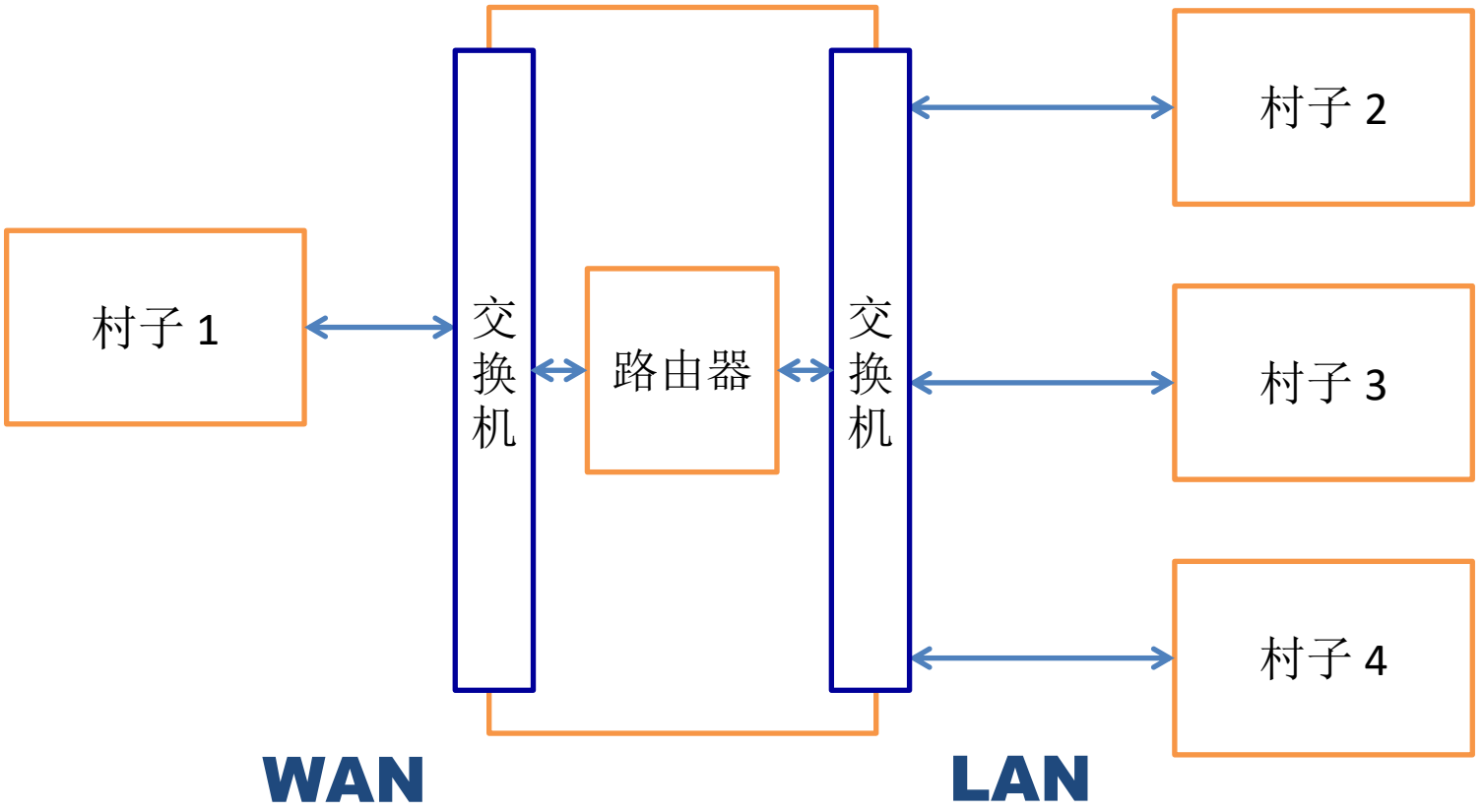
# 交换机 vs 路由器

- 引入路由器后，村子 1 看到的视角变成了：



# 交换机 vs 路由器

- 平常生活中的路由器经常是 交换机 + 路由器



# Linux 网络

- 第二、三实验需要在 Linux 环境下进行
- 为什么?
  - Linux 对底层网络编程友好
  - Linux 提供了网络隔离机制（netns），可以在一个系统中虚拟出多个网络，这也是 Docker 等容器化技术所采用的机制

# 网络接口

- 什么是网络接口（Network Interface）？
- 它可能对应一个实际的网卡：
  - eth0 有线网卡，wlan0 无线网卡
- 也可能对应虚拟网卡：
  - lo: 本地环回
  - veth：虚拟以太网，成对出现，虚拟了一根网线的两端

# 基础命令

- 前面已经出现过的有：
  - ip addr：显示所有网络接口和地址信息，可写为 ip a
  - ip route：显示当前的系统路由表，可写为 ip r
- 增删 IP 地址：
  - ip addr add 1.2.3.4/24 dev en0：给 en0 网络接口添加 1.2.3.4 的 IP 地址，同时配置直连路由 1.2.3.0/24 dev en0
  - ip addr del 1.2.3.4/24 dev en0

# 基础命令

- 增删路由表：
  - ip route add default via 192.168.0.1 dev en0：新增默认路由（default 表示 0.0.0.0/0），下一跳 IP 地址是 192.168.0.1，网络接口是 en0
  - ip route add 192.168.0.0/24 dev en0：新增直连路由
  - ip route del default via 192.168.0.1 dev en0：删除路由

# Linux 下发送 IP 分组的流程

- 在 Linux 下，发送 IP 分组的时候，需要：
  - 根据目的 IP 地址，查询路由表，找到下一跳 IP 地址和出网络接口
  - 根据下一跳 IP 地址查询邻居表，查询得到下一跳 IP 地址对应的 MAC 地址
  - 将上一步查询得到的 MAC 地址填入目的 MAC 地址，向第一步查询到的出网络接口发送 IP 分组
- 什么是 MAC 地址？Ch6 MAC Sublayer 会讲

# Linux 下发送 IP 分组的例子

- 例子：访问 Baidu
  - 第一步：通过 DNS 解析，得到 baidu 的 IP 地址，作为目的 IP 地址
  - 第二步：使用目的 IP 地址查询路由表，得到默认路由：default via 默认网关 dev 网络接口0
  - 第三步：查询默认网关的 MAC 地址，填入目的 MAC 地址
  - 第四步：向网络接口 0 发送该 IP 分组

# 实验框架中发送 IP 分组

- 在实验框架中，虽然在 Linux 下运行，但是绕过了 Linux 的网络栈
  - 其实直接关闭了 Linux 的网络栈，避免干扰实验程序
- 因此自己决定 IP 分组的内容，以及发到哪个网络接口
  - HAL_SendIPPacket 函数
  - 前述查路由表+邻居表的流程已经在框架中实现

# 虚拟网络（netns）

- Linux 实现了 network namespace（netns）技术，可以在系统中虚拟出多个网络
- 默认情况下，所有进程都在默认 netns 中
- 创建 netns：
  - ip netns add test
  - 就会创建名为 test 的 netns，它里面只有自动创建的 lo 本地环回网络接口，无法访问外面的网络

# 虚拟网络（netns）

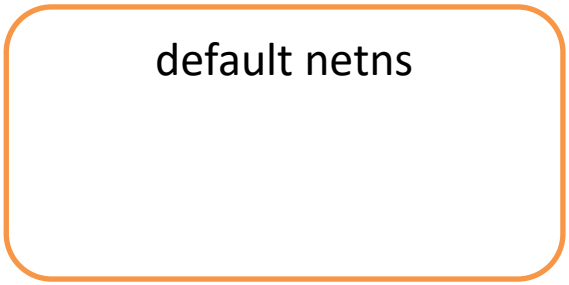
- 把进程放到 netns 中执行：
  - ip netns exec test command args ...
  - 在 test 这个 netns 中运行 command
  - 如果不用 ip netns exec test，直接运行 command 就是在默认 netns 中
- 查看 netns 中的网络接口的地址：
  - ip netns exec test ip addr

# 虚拟网络（netns）

- 如何给新建的 netns 添加网络呢?
  - 虚拟一个网线出来：veth
  - 网线有两头，一头放在 test netns 中，另一头放默认 netns 中
  - 首先创建 veth：ip link add veth0 type veth peer name veth1
  - 把 veth1 放到 test netns 中：ip link set veth1 netns test

# 虚拟网络（netns）

- 初始情况下，只有 default netns



# 虚拟网络（netns）

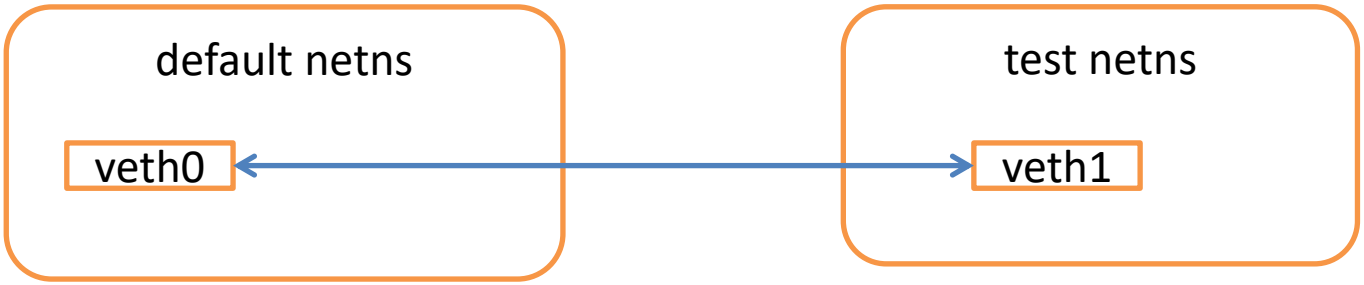
- 首先创建 test netns：

# 虚拟网络（netns）

- 创建一根虚拟网线，网线两端是 veth0 和 veth1，都在 default netns 中

# 虚拟网络（netns）

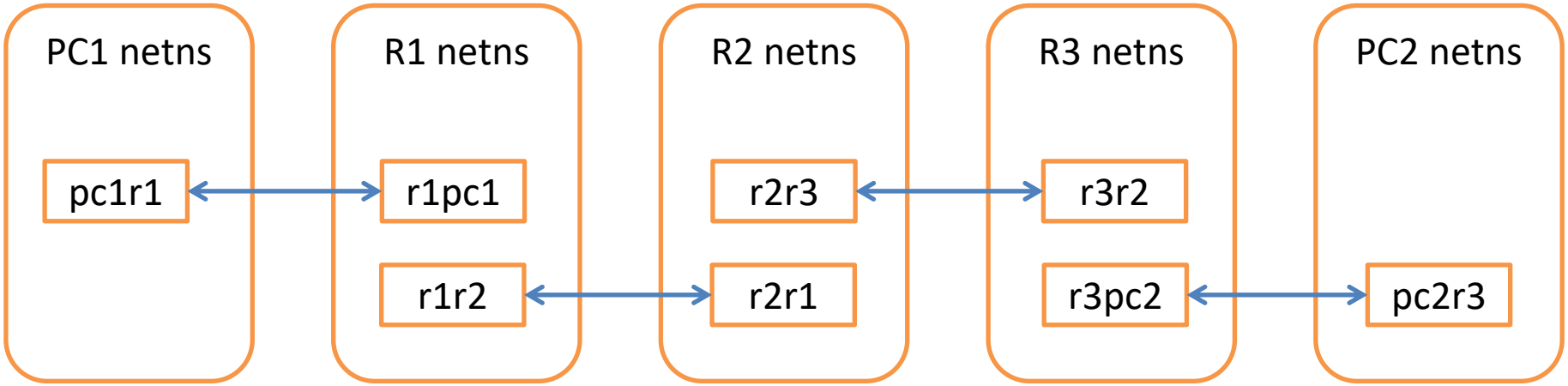
- 把 veth1 放到 test netns 中，在默认 netns 和 test netns 之间连接了一个虚拟的网线：

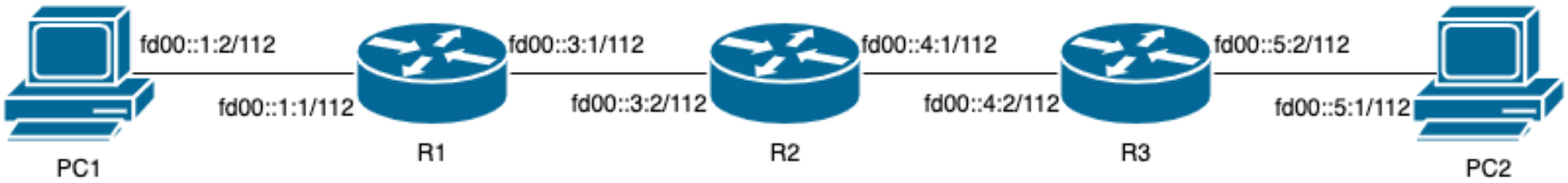



- 默认 netns 通过 veth0 可以访问 test netns
- test netns 通过 veth1 可以访问默认 netns

# 虚拟网络（netns）

- 依此类推，就可以把实验中所使用的五个设备的拓扑在 Linux 中用 netns 实现出来：



# 本地测试

- 针对第二、三阶段，提供了在虚拟机网络中的测试脚本
  - 按照实验拓扑，创建并配置 netns
  - 按照测试步骤，在虚拟网络中复现测试场景
  - 详细内容见实验文档
- 重要：虚拟网络在一些小细节上与真实网络不一样，详见实验文档

# 调试思路

- 介绍一个通用的调试思路
- 学习了这个调试思路，可以有目的地去调试，而不是盲目地去猜问题会在哪里
- 首先举一个简单的例子：假设现在实现了一个功能，并且这个功能需要用多个步骤完成，后一个步骤依赖前一个步骤。例如一个功能分三步：
- A - B - C：B 依赖 A，C 依赖 B

# 调试思路

- 现在这个功能不工作，怎么调试？
- 就按照 A-B-C 的顺序，一个一个检查：
  - A 步骤是否完成？
  - A 步骤是否输出了正确的结果？
  - B 步骤是否从 A 步骤获取了正确的输入？
  - B 步骤是否完成？
  - B 步骤是否输出了正确的结果？
  - C 步骤是否从 B 步骤获取了正确的输入？
  - C 步骤是否完成？

# 调试思路

- 这个检查方法看起来很麻烦，步骤很多，感觉很容易就可以想到
- 但人总是倾向于偷懒，不愿意逐步去排查问题
- 寄希望于自己能够从茫茫代码中猜到代码哪里有问题，直接把问题解决了
- 然而往往事与愿违，越是心急越是难以找到问题所在

# 调试思路

- 而在网络领域里（如本实验）这种方法特别好用：
  - 网络内的各个节点自然形成了一个多步的逻辑链
  - 例如网络拓扑是 A - B - C，那么 A 要发送数据给 C，一定会经过以下过程：
  - A 生成数据，发送给 B
  - B 收到 A 的数据，检查是否正确
  - B 发送数据给 C
  - C 收到 B 的数据，检查是否正确
  - C 进行数据的处理

# 调试思路

- 在网络里面，拓扑上怎么连，数据就怎么走，你只要去路径上每一个点看一看：
  - 它收到数据了没
  - 它收到数据以后，是否认为数据是正确的
  - 数据正确的话，它是否准备把数据再发出去
  - 要发送出去的话，它把数据发送给了谁
- 按照这个方法，很容易就可以定位到是具体哪个节点出现了问题

# 调试思路小结

- 当你知道这个东西是怎么工作的，它要工作需要 ABCD 哪几步
- 那你就直接按照这个顺序去排查，看看到底卡在哪一步上了
- 只要你的理论知识足够扎实，就可以逐步抽丝剥茧把实际的问题找到
- 如果你没有掌握现象背后的原理，很可能就会走向思维的误区，调试不出来

# 节点 vs 结点

- 结点是抽象的概念，例如图里面的结点，链表里的结点，在数据结构和数学领域使用
- 节点指的是实际的设备，例如计算机，路由器，交换机等等
- 网络拓扑中，网络设备物理上真实存在，可以称为节点；另一方面，把拓扑看成图的话，网络设备对应图中的一个结点


清華大學

Tsinghua University

# 谢谢