

IP routing

红茶三杯 CCIE 学习文档

文档版本： 2.0

更新时间： 2013-08-01

文档作者： 红茶三杯 <http://weibo.com/vinsoney>

文档备注： 请关注文档版本及更新时间。关注 <http://ccietea.com> 以便获得文档的最新版本。

原创技术文档，用于交流分享，可随意传播

红茶三杯（朱 SIR）版权所有，转载请保留原作者信息。

红茶三杯

网络工程 | 项目管理 | IT 服务管理 | CCIE 培训

学习 沉淀 成长 分享

微博：<http://weibo.com/vinsoney>

博客：<http://blog.sina.com.cn/vinsoney>

站点：<http://ccietea.com>

目录

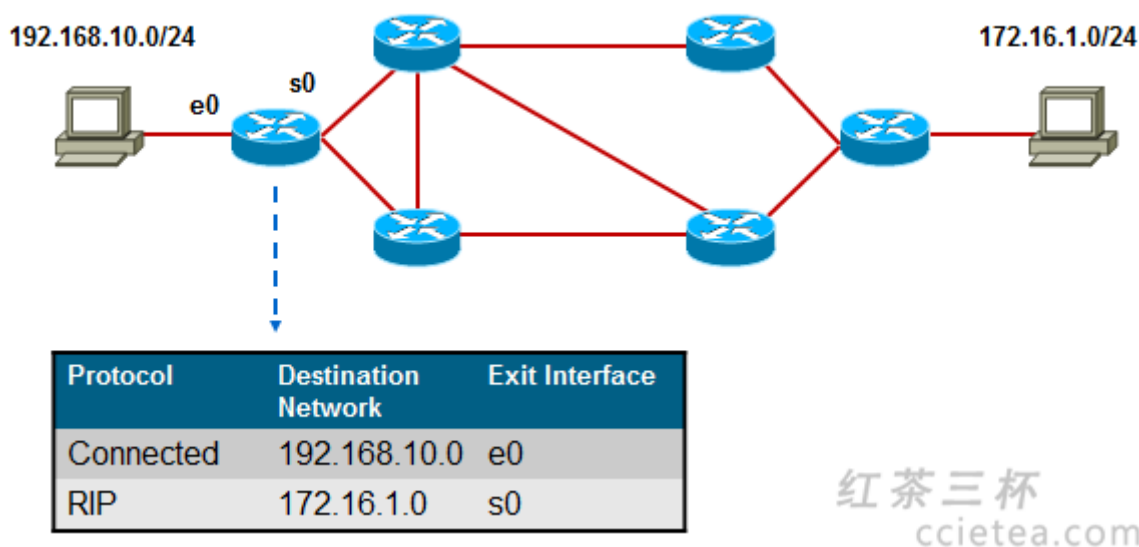
| | | |
|-------|--|----|
| 1 | IP 路由概述 | 5 |
| 1.1 | 关于 IP 路由 | 5 |
| 1.2 | IP 路由表 (IP Routing Table) | 5 |
| 1.3 | 管理距离 (Administrative Distance , 简称 AD) | 6 |
| 1.4 | 动态路由协议端口号或协议号 | 7 |
| 1.5 | ICMP 重定向 | 8 |
| 1.6 | 有类及无类路由查找方式 | 9 |
| 1.7 | 最长匹配原则 | 10 |
| 2 | CISCO 数据转发方式 | 13 |
| 2.1 | 数据转发方式 | 13 |
| 2.2 | 负载均衡方式 | 16 |
| 3 | 静态路由 | 20 |
| 3.1 | 概述 | 20 |
| 3.2 | 静态路由的配置 | 20 |
| 3.3 | 注意事项 | 21 |
| 3.4 | 浮动静态路由 | 23 |
| 3.5 | 路由汇总 | 23 |
| 4 | 距离矢量路由协议 | 28 |
| 4.1 | Basic | 28 |
| 4.2 | RIPv1 | 29 |
| 4.2.1 | Summary | 29 |
| 4.2.2 | Timers | 29 |
| 4.2.3 | RIP database | 30 |
| 4.2.4 | RIP 消息格式 | 31 |
| 4.2.5 | Ip rip trigger | 31 |
| 4.2.6 | RIPV1 操作行为 | 32 |
| 4.2.7 | 疑难解析 | 33 |
| 4.3 | RIPv2 | 35 |
| 4.3.1 | 改进 | 35 |
| 4.3.2 | 消息类型 | 35 |
| 4.3.3 | 认证 | 36 |
| 4.3.4 | 兼容性 | 37 |
| 4.3.5 | 高级特性 | 38 |
| 4.3.6 | 疑难解析 | 39 |
| 4.4 | RIPng | 39 |
| 4.5 | 参考资料 | 39 |
| 5 | 路由重发布 (Redistributing Routing Protocols) | 40 |
| 5.1 | 技术背景 | 40 |
| 5.2 | 实施要点 | 42 |
| 5.2.1 | 路由 feedback | 42 |
| 5.2.2 | 管理距离问题 | 43 |

| | | |
|-------|---|-----|
| 5.2.3 | Metric 问题 | 44 |
| 5.2.4 | 有类、无类路由选择协议的重发布 | 46 |
| 5.3 | 配置示例 | 46 |
| 5.3.1 | 配置命令 | 46 |
| 5.3.2 | 配置示例 | 46 |
| 5.3.3 | 重发布的常见问题 | 49 |
| 6 | 路由策略 | 50 |
| 6.1 | Passive-interface | 50 |
| 6.1.1 | 特性概述 | 50 |
| 6.1.2 | 相关要点 | 51 |
| 6.1.3 | Passive-interface 的配置 | 51 |
| 6.2 | 单播更新 | 53 |
| 6.3 | 调整路由协议的管理距离 | 54 |
| 6.4 | Route-map | 55 |
| 6.4.1 | Route-map 概述 | 55 |
| 6.4.2 | 配置命令 | 57 |
| 6.4.3 | 配置示例 | 58 |
| 6.4.4 | 难点案例 | 60 |
| 6.5 | distribute-list | 62 |
| 6.5.1 | 工具概述 | 62 |
| 6.5.2 | 部署要点 | 63 |
| 6.5.3 | 配置命令 | 65 |
| 6.5.4 | 应用场合 | 66 |
| 6.6 | 路由匹配工具 | 70 |
| 6.6.1 | ACL | 70 |
| 6.6.2 | Prefix-list | 70 |
| 6.6.3 | 路由标记 Tag | 73 |
| 7 | 路径控制 | 77 |
| 7.1 | 路径控制概述 | 77 |
| 7.2 | Offset-list 偏移列表 | 78 |
| 7.3 | Policy-based Routing (PBR) 策略路由 | 80 |
| 7.3.1 | 关于 PBR | 80 |
| 7.3.2 | 命令汇总 | 81 |
| 7.3.3 | 实验验证 | 82 |
| 7.3.4 | PBR 案例 | 88 |
| 8 | 双点双向路由重发布 | 91 |
| 8.1 | 概述 | 91 |
| 8.2 | 双点双向路由重发布存在的问题 | 92 |
| 8.3 | 双点双向路由重发布的实现 | 98 |
| 8.3.1 | 解决方案：修改路由协议管理距离 | 98 |
| 8.3.2 | 解决方案：采用重发布静态汇总路由的方式规避次优路由等问题 | 98 |
| 8.3.3 | 解决方案：使用分发列表规避次优路径问题 | 101 |
| 8.3.4 | 思考题 | 104 |

| | | |
|-----|--------------------------|-----|
| 9 | 缺省路由 | 108 |
| 9.1 | ip default-gateway | 108 |
| 9.2 | ip default-network | 108 |
| 9.3 | RIP 重发布默认路由 | 110 |
| 10 | 参考书目 | 110 |

1 IP 路由概述

1.1 关于 IP 路由



在一个 IP 网络中，路由（Routing）是个非常非常基本的概念。网络的基本功能，是使得处于网络中的两个 IP 节点能够进行通信，而通信实际上就是数据交互的过程，数据交互则需要网络设备帮助我们来将数据在两个通信节点之间进行传输。当路由器（或者其他三层设备）收到一个 IP 数据包，路由器会找出 IP 包三层头中的目的 IP 地址，然后拿着目的 IP 地址到自己的路由表中进行查找，找到“最匹配”的条目后，将数据包根据路由条目所指示的出接口或下一跳 IP 转发出去，这就是 **IP 路由（IP routing）**。而每台路由器都会在本地图维护一个**路由表（Routing Table）**，路由表中装载着路由器获知路由条目（Routes），路由条目由**路由前缀（路由所关联的目的地址）、路由信息来源、出接口或下一跳 IP** 等元素构成。路由器通过静态的或者动态的方式获取路由条目并维护自己的路由表。

1.2 IP 路由表（IP Routing Table）

每个路由表项最少必须包括下面三个项目：

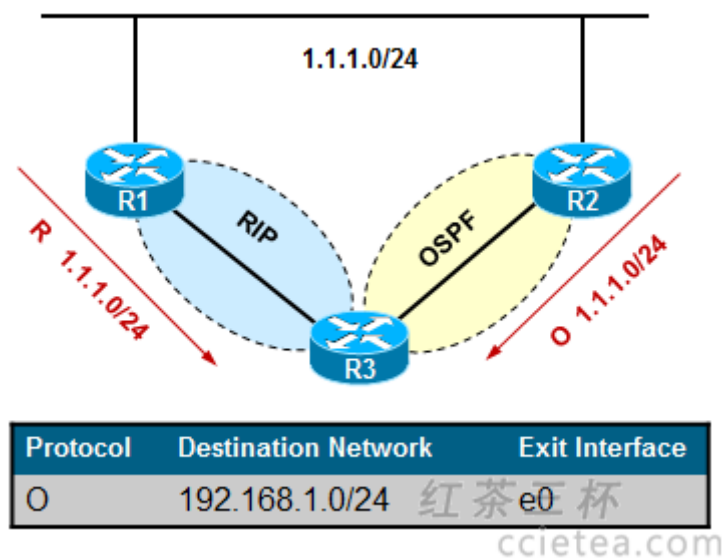
- **目标地址（路由前缀）：** 这是路由条目所关联的目的网络号。一条完整的路由前缀由：网络号+前缀长度构成

成，两者缺一不可，例如 192.168.1.0/24 与 192.168.1.0/25，虽然两者的网络号相同，都是 192.168.1.0，但是两者绝对是两条不同的路由、两个不同的路由前缀，因为他们的前缀长度不相同。

- **指向目标的指针：** 指针不是指向路由器的直连目标网络就是指向直连网络内的另一台路由器地址，或者是到这个链路的本地接口。更接近目标网络一跳的路由器叫下一跳(next hop)路由器。
- **路由信息的来源：** 本条路由是通过什么途径学习到的，例如是静态的，或者是通过 OSPF、IS-IS、EIGRP、BGP 等动态路由学习到的。

1.3 管理距离 (Administrative Distance , 简称 AD)

路由器可以通过多种途径获知路由条目：如静态手工配置、各种动态路由协议等等。当路由器从两种不同的途径获知去往同一个目的地的两条路由，那么路由器会比较这两条路由的 AD 值，也就是管理距离，优选 AD 值小的路由。如果 AD 值相等，例如是同种路由协议，则进一步比较 metric 值，当然，这其中还牵涉到不同的路由协议内在的工作机制问题，这就要针对不同的路由协议具体讨论了。如下图，R3 与 R1 运行的是 RIP 协议，R3 又通过 OSPF 与 R2 建立邻接关系。于是 R3 同时从 RIP 及 OSPF 学习到了去往目的地 1.1.1.0/24 的路由，这两条路由分别以 R1 和 R2 作为下一跳。那么 R3 最终选择 OSPF 的路由装载进路由表，也就是将 R2 作为实际去往 1.1.1.0/24 的下一跳，因为 OSPF 协议的 AD 值比 RIP 要小要更优。



针对不同的路由协议，对应的 AD 值见下表，这是个众所周知的约定：

| Routing Protocols | AD | 备注 |
|-------------------|----|----|
|-------------------|----|----|

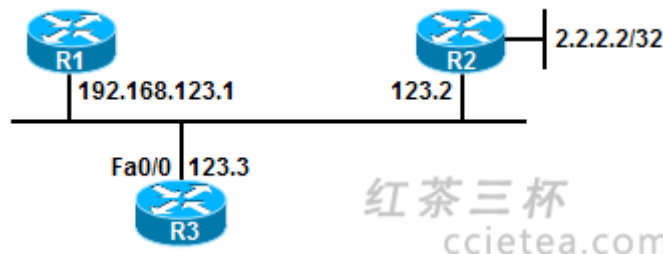
| | | |
|------------|-----|-----------|
| 直连接口 | 0 | |
| 关联出接口的静态路由 | 1 | Metric =0 |
| 关联下一跳的静态路由 | 1 | Metric =0 |
| EIGRP 汇总路由 | 5 | |
| 外部 BGP | 20 | |
| 内部 EIGRP | 90 | |
| IGRP | 100 | |
| OSPF | 110 | |
| RIPv1、v2 | 120 | |
| 外部 EIGRP | 170 | |
| 内部 BGP | 200 | |

PS：当同一台路由器，有两条去往同一目的地的静态路由、分别使用的是出接口和下一跳的方式，这两条路由由负载均衡（因为 AD 值和 metric 值都相等）。而使用关联出接口的方式配置的静态路由，该路由条目将作为直连网络输入到路由表中，具体请见本文档静态路由部分。

1.4 动态路由协议端口号或协议号

| Routing Protocols | 基于协议 | 备注 |
|-------------------|------|-----------|
| RIP | UDP | 端口号 520 |
| RIPng | UDP | 端口号 521 |
| EIGRP | IP | IP 协议号 88 |
| OSPF | IP | IP 协议号 89 |
| BGP | TCP | 端口号 179 |

1.5 ICMP 重定向



R3 的网关为 R1，R1 本身又有静态路由到 2.2.2.2，下一跳为 R2，注意这是个 MA 网络，三个接口都是同网段的，这个很关键，R3 将去往 2.2.2.2 时，将数据丢给自己的网关 192.168.123.1，这时 R1 经过路由表查找后，发现数据的下一跳是与本地入接口同一个网段的 123.2，因此他认为 123.2（也就是 R2）比自己距离目标更近，因此给源也就是 R3 发送了一个 **ICMP 重定向消息**（要求 R1 的以太网接口开启 ip redirects），告诉 R3 所 192.168.123.2 为更优的下一跳，那么 R2 后续的报文将发给 R2。

R1 的配置如下：

```
ip route 2.2.2.0 255.255.255.0 192.168.123.2
```

R3 的配置如下：

```
ip route 0.0.0.0 0.0.0.0 192.168.123.1
```

在 R3 上开启 debug ip icmp，现在 R3 去 ping 2.2.2.2：

```
*Mar  1 00:11:47.887: ICMP: redirect rcvd from 192.168.123.1- for 2.2.2.2 use gw 192.168.123.2
*Mar  1 00:11:47.891: ICMP: echo reply rcvd, src 2.2.2.2, dst 192.168.123.3
*Mar  1 00:11:47.951: ICMP: echo reply rcvd, src 2.2.2.2, dst 192.168.123.3
*Mar  1 00:11:47.999: ICMP: echo reply rcvd, src 2.2.2.2, dst 192.168.123.3
*Mar  1 00:11:48.023: ICMP: echo reply rcvd, src 2.2.2.2, dst 192.168.123.3
*Mar  1 00:11:48.043: ICMP: echo reply rcvd, src 2.2.2.2, dst 192.168.123.3
```

另一方面我们抓包得到：


```
Internet Protocol Version 4, Src: 192.168.123.1 (192.168.123.1), Dst: 192.168.123.3 (192.168.123.3)
  Version: 4
  Header length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Trans
  Total Length: 56
  Identification: 0x0000 (0)
  Flags: 0x00
  Fragment offset: 0
  Time to live: 255
  Protocol: ICMP (1)
  Header checksum: 0x446f [correct]
  Source: 192.168.123.1 (192.168.123.1)
  Destination: 192.168.123.3 (192.168.123.3)
Internet Control Message Protocol
  Type: 5 (Redirect)
  Code: 1 (Redirect for host)
  Checksum: 0x9b5d [correct]
  Gateway address: 192.168.123.2 (192.168.123.2)
  Internet Protocol Version 4, Src: 192.168.123.3 (192.168.123.3), Dst: 2.2.2.2 (2.2.2.2)
  Internet Control Message Protocol
```

这就是 R1 给 R3 发送的 ICMP 重定向消息，注意里头的 Gateway address 字段，填写的就是比自己距离目的地更近的下一跳 IP。

如果要关闭 ICMP 重定向，需在接口上，使用 no ip redirects。

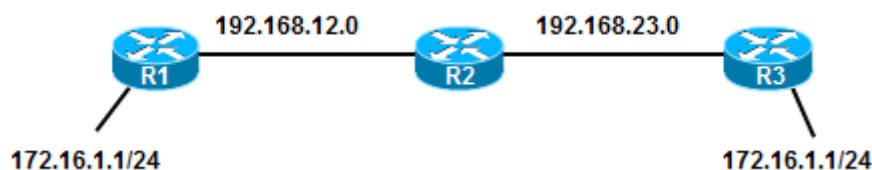
另外这里注意 ICMP 重定向的发送条件。同时注意，R3 上的这条默认路由，如果配置成关联出接口的方式，那么就无法观察到重定向的现象了，至于原因，你懂的，代理 ARP 嘛。

1.6 有类及无类路由查找方式

CISCO 路由器在路由的全局查找上有两种方式：有类 (Classful) 查找方式及无类 (Classless) 查找方式。

当路由器执行无类别路由查找时 (默认，ip classless)，它不会注意目的地址的类别，它会在目的地址和所有已知的路由之间逐位(bit by bit)执行最长匹配；

而如果有类路由查找 (no ip classless 且关闭 ip cef)，那么收到一个数据包时，路由器先查找目的地址所属主类，如果路由表中有主类路由，则再去找子网，如果有子网路由，则查询被限定在这些子网中，并进一步查找，如果最终查找失败 (没有任何子网匹配这条路由)，则丢弃数据包，即使有默认路由存在；如果本地没有该主类路由，则看是否有默认路由，如果有，则按默认路由转发，如果无，则丢弃数据包。



在 R2 上做如下配置并做测试 (**均在 no ip classless 且关闭 ip cef 环境下做的测试**):

- **【实验 1】有主类路由；有默认路由，走主类路由**

```
ip route 0.0.0.0 0.0.0.0 192.168.23.3
```

```
ip route 172.16.0.0 255.255.0.0 192.168.12.1    !! 走主类路由
```

- **【实验 2】有子网路由（匹配）；有默认路由，走子网路由**

```
ip route 0.0.0.0 0.0.0.0 192.168.23.3
```

```
ip route 172.16.1.0 255.255.255.0 192.168.12.1    !! 走子网路由
```

- **【实验 3】无主类网络；有默认路由，走默认路由**

```
ip route 0.0.0.0 0.0.0.0 192.168.23.3    !! 走默认路由
```

```
ip route 172.17.0.0 255.255.255.0 192.168.12.1
```

- **【实验 4】有主类网络；有子网路由（匹配），按最长匹配**

```
ip route 172.16.1.0 255.255.255.0 192.168.12.1
```

```
ip route 172.16.0.0 255.255.0.0 192.168.23.3
```

- **【实验 5】有子网路由，子网路由前缀长度不一样（但都匹配），按最长匹配**

```
ip route 172.16.1.0 255.255.255.0 192.168.12.1
```

```
ip route 172.16.1.0 255.255.255.224 192.168.23.3    !! 按最长匹配，走这条
```

- **【实验 6】有子网路由，子网路由前缀长度不一样（匹配及不匹配均有），走匹配路由**

```
ip route 172.16.1.0 255.255.255.224 192.168.12.1    !! 匹配，走这条
```

```
ip route 172.16.1.32 255.255.255.224 192.168.23.3
```

- **【实验 7】有子网路由（不匹配）；有默认路由**

```
ip route 0.0.0.0 0.0.0.0 192.168.12.1
```

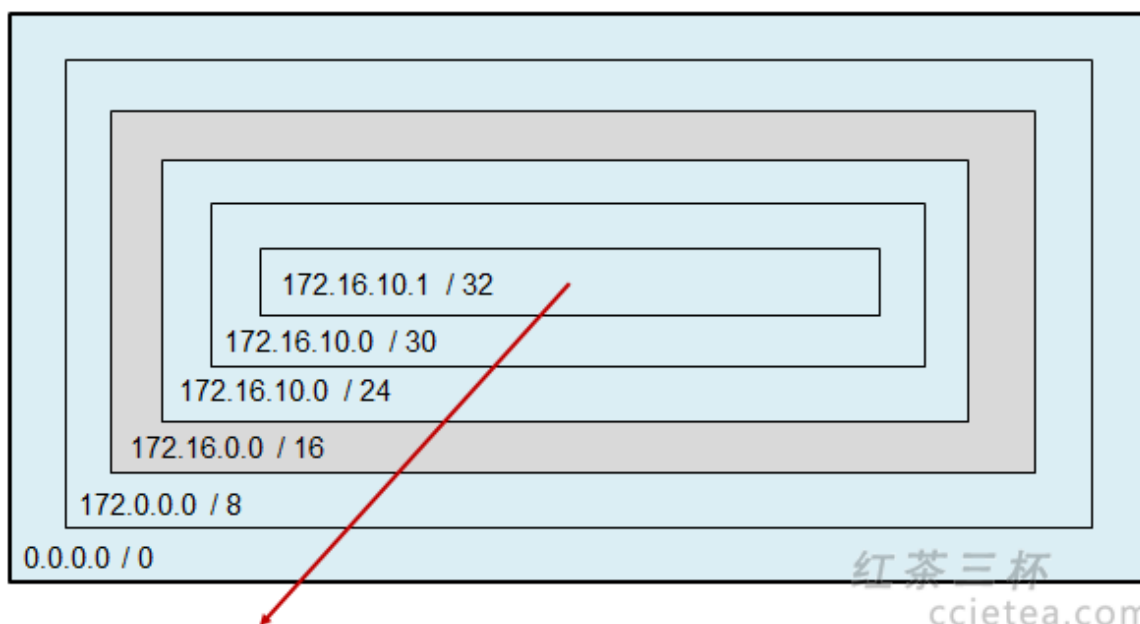
```
ip route 172.16.1.32 255.255.255.224 192.168.23.3
```

无法 ping 通，因为去往 172.16.1.1 查表后发现子网路由，因此查找被限定在子网路由中，然后却发现这条路由不匹配，因此直接丢弃报文，而不会走默认路由。

PS：要注意将有类、无类路由查找方式，与有类、无类路由选择协议区分开来。

1.7 最长匹配原则

最长匹配原则是 CISCO IOS 路由器默认的路由查找方式。当路由器收到一个 IP 数据包时，会将数据包的目的 IP 地址与自己本地路由表中的表项进行 bit by bit 的逐位查找，直到找到匹配度最长的条目，这叫最长匹配原则。



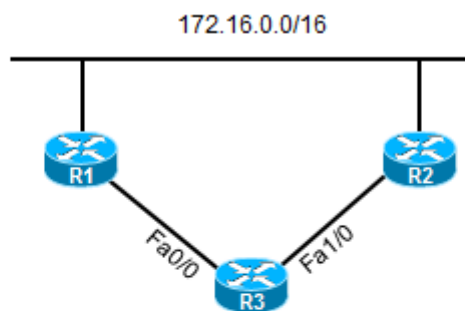
这里有几个概念要先搞清楚：

看上面的图，灰色的空间 172.16.0.0/16，这个网络号，我们称为**主类网络号**，所谓主类网络号，意思是该网络号，按照其所属的 IP 地址类别区分后，对应上的默认的子网掩码长度后得到的网络号。如 172.16.0.0 这是一个 B 类地址，B 类地址的默认子网掩码长度是 16 位，因此 172.16.0.0/16 本身就是一个主类网络号。再举过一个例子，10.1.12.0/24，首先 10 开头的，这是一个 A 类地址，A 类地址默认的掩码是 255.0.0.0，因此 10.1.12.0/24 它的主类网络号是 10.0.0.0/24。

我们首先顺着上面的图，从 172.16.0.0/16 开始往里走，下一个我们看到的网络号是 172.16.10.0/24，这很明显是应用了 VLSM 可变长子网掩码之后，得到的一个 172.16.0.0/16 这个主类网络的一个**子网**。所以所谓的子网，我们可以理解为是在网络号所属类别的默认掩码长度的基础上，将掩码“拉长”或者向主机位借位从而得到的一个网络号。实际上 172.16.0.0/16 是将 172.16.10.0/24 囊括在内的一个区间。那么在这里，如果我们有一个 IP：172.16.10.1，实际上这个 IP 既可以理解为在 172.16.0.0/16 网络内，也是在 172.16.10.0/24 网络内，当然，这里我们能看出来，谁更精确呢？很明显是 172.16.10.0/24 更精确，我们说，它的**匹配长度相比 172.16.0.0 更长**。

当然子网 172.16.0.0/16 还可以进一步划分子网，得到 172.16.10.0/30，甚至 172.16.10.1/32，那么如果这些前缀都存在的情况下，当我要去找 172.16.10.1，谁的匹配度最高呢？很明显，是 172.16.10.1/32 这条主机前缀，或者说，主机路由吧？这就是**最长匹配原则**。

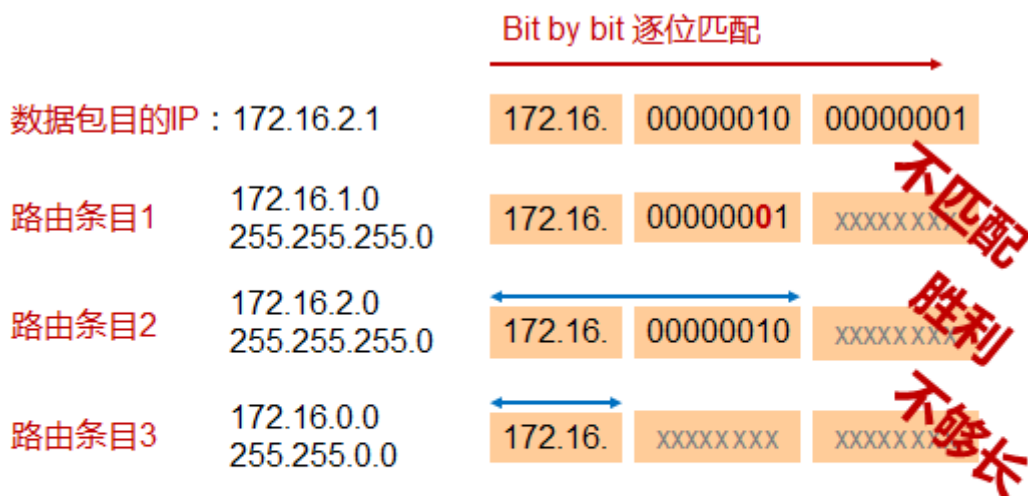
OK，现在回到 172.16.0.0/16 这个主类网络号，然后我们向外走，看上图。172.0.0.0/8 实际上是将这个 B 类地址的掩码向左移了 8bits，这样一来得到的这个网络号实际上是囊括了 172.16.0.0/16 在内的一个大的网络号，我们称其为**超网**。



```
S 172.16.1.0/24 [1/0] via 192.168.13.1
S 172.16.2.0/24 [1/0] via 192.168.13.2
S 172.16.0.0/16 [1/0] via 192.168.23.1
```

红茶三杯
ccietea.com

因此，当路由器的路由查找方式为 classless 也就是无类路由查找方式时，路由器默认的查找动作是最长匹配原则。例如上图，当 R3 收到一个数据包，去往 172.16.1.1，那么实际上，172.16.1.1 是“掉落”在 172.16.1.0/24 及 172.16.0.0/16 网络中的，两者貌似皆可，但是 172.16.1.0/24 显然，匹配度要更长，因此，最终这个数据包被丢给了 R1。同理若有数据包去往 172.16.2.1 呢？由于根据最长匹配原则，172.16.2.0/24 这个条目匹配度最高，因此数据被扔给了 R2。这个过程有点类似下面的样子：



```
S 172.16.1.0/24 [1/0] via 192.168.13.1
S 172.16.2.0/24 [1/0] via 192.168.13.2
S 172.16.0.0/16 [1/0] via 192.168.23.2
```

红茶三杯
ccietea.com

并且，当 R2 挂掉之后，172.16.2.0/24 的条目失效，去往 2.0 子网的数据此时匹配的路由条目是 172.16.0.0/16 这条路由，因此被送往了 R1。

这就是利用最长匹配原则，实施的一种简单的数据分流及路径冗余的方法。

下面我们总结一下路由器关于路由查找的几个重点内容：

- 不同的前缀（网络号+掩码，缺一不可），在路由表中属于不同的路由
- 相同的前缀，通过不同的协议获取，先比 AD，后比 metric
这是一般情况，当然有二般情况，这就要看特定的环境和特定的路由协议了
- 默认采用最长匹配原则，匹配，则转发；无匹配，则找默认路由，默认路由都没有，则丢弃
- 路由器的行为是逐跳的，到目标网络的沿路径每个路由器都必须有关于目的地的路由
- 数据是双向的，考虑流量的时候，要关注流量的往返。

2 CISCO 数据转发方式

2.1 数据转发方式

交换的概念：switching is the process of mapping layer 2 to layer 3 addresses and forwarding to a destination interface.

接下去我们来看看几种交换方式，本小节部分内容摘抄于网络，感谢原作者的分享。

1. Process Switching(进程交换)

在这种模式下，一条数据流(flow)中的第一个包(packet)将被置入系统缓存(system buffer).其目的地址将会拿到路由表中去查询比对,路由器的处理器(CPU or Processor)同时将进行 CRC 校验,检查包是否正确.然后数据包的二层 MAC 地址将会被重写,替换为下一跳接口的 MAC 地址,这样的过程将会继续,对这条数据流(flow)中的第 2 个、第 3 个数据包.....相同的操作,包括查询路由表、重写 MAC 地址,CRC 校验等。这种方式无疑是延迟最大的,因为它要利用 system buffer 以及 processor 去处理每个收到的包.但是我们仍然有机会使用这种交换方式,比如在进行基于每个包的负载分担时,或是 debug ip packet 时。

【启动进程交换】

默认情况下,思科路由器会启用 fast switching 或 optimum switching 或是 cef switching,而不是 process switching,所以我们只能通过:no ip route-cache 来禁用 fast switching,这在另一种意义上正是开启 process switching.

2. Fast Switching

快速交换要优于 process switching,它采用了 route cache(路由缓存)来存储关于某条数据流(flow)的特定信息,当然会包括诸如目的 MAC 地址,目的接口等内容.这时我们只需要对一条数据流(flow)中的第一个包做 process

switching,并把信息存入 cache,所有后续数据包,可以不必再中断 system processor 去执行查询等操作,直接从 cache 中提取目的接口,目的 MAC 地址等,这样大大加速了包转发速度。

【启动快速交换】

fast switching 在某些资料上可能被称为 route-cache switching

思科 1600、1700、2500、2600 系列路由器的 ethernet、fast ethernet、serial 接口默认采用的就是 fast switching.

我们可以用 ip route-cache 命令,在接口上启用 fast switching

show ip cache 来检查 fast switching 的相关信息.

3. Optimum and Distributed Switching

这两种交换模式,从原理上来讲都与 fast switching 极为相似,比如 optimum switching 其实采用了一种经过优化的交换缓存(optimumed switching cache),它的速度要较平常 cache 要快.distributed switching mode 需要使用 Versatile Interface Card 这种硬件卡,又称 VIP card.它会自己保存一份 route cache,这样在查询时就不必要等待使用共享的系统缓存了(shared system buffer),无论相对于 fast switching 还是 optimum switching 来讲,都是比较快的.

这两种模式一般只在思科高端设备上有所应用,比如 7200 系列路由器.或者 12000 系列路由器.

命令:ip route-cache optimum show ip cache optimum

4. Netflow switching

这种模式是最值得参考的,它完全基于其它 switching mode,重点在于对流经的数据包进行计费、监控、网管.但不得不提的是,这种模式因为也要存储相关信息,经过统计,大致 65536 条数据流(flow)会耗费 4MB 的 system buffer.

相关命令:

ip route-cache flow

show ip cache flow

ip flow-export 将 NETFLOW 审计的数据包转发到指定设备.

5. Cisco Express Forwarding

思科 CEF 是最为高效的一种三层协议。CEF 采用了基于硬件的平台,它不仅仅是将数据都存入 system buffer,而是将整个路由表、拓扑表,以及所有的下一跳地址、MAC 地址全部进行"预存",只要路由表、拓扑表中存在的条目,无论是否有数据请求发往其目的地址,都会提前预读取,预设置缓存.这样,当有新的数据请求发送时,就不需要 CPU 去查询目的接口,目的 MAC 地址等信息,而是直接从缓存中读取,从而使转发速度得以大大提高.

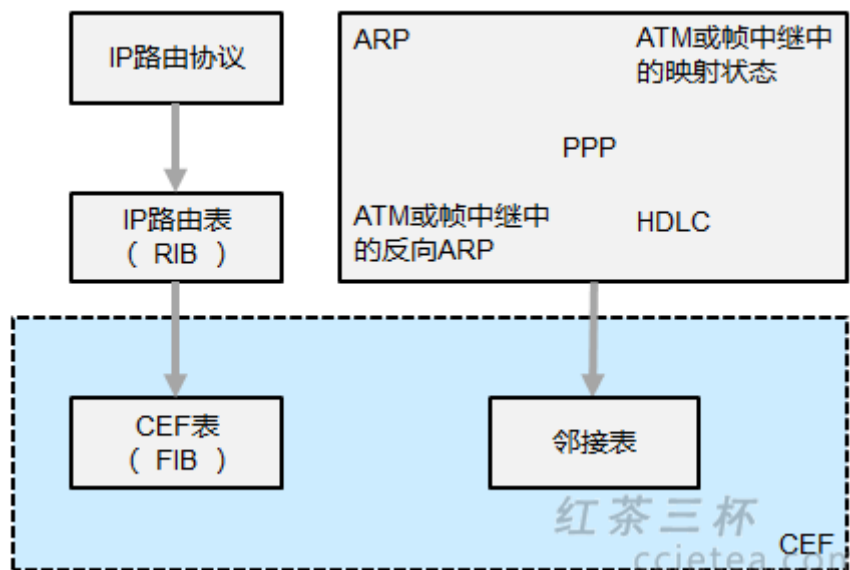
相关命令:ip route-cache cef

show ip cef

show ip cef detail

路由器通常根据入站接口和源与目的地址类型确定是否使用 CEF 交换. 对于考虑使用 CEF 的路由器来说,出站接口必须配置为 CEF 交换模式,如果接口上配置了 CEF,那么 CEF 将尝试交换数据包。否则,CEF 将会把数据包交付给仅次于最好的可用交换方法去处理。

CEF 有哪些组件：



邻接表

CEF 组件中的邻接表用于 MAC 或者第二层重写信息。

- 当路由器与主机邻接的话，他们通过某些方式学习。
- 当路由器通过点对点链路直连，每一个接口上只有一个邻居存在。
- 当路由器通过以太网这样的多路访问介质连接，需要一种动态的机制来发现对方，例如 ARP，ARP 可以将二层地址映射到 IP。如果是帧中继多点链路，那么邻接关系可以用过地址解析协议、映射表等学习到。

尽管决策如何转发报文的工作由 FIB 来实现，但是第二层的帧重写工作却是根据邻接表中的信息来完成，第二层需要重写的字段包括用于帧转发的第二层头部。对于以太网来说，就是新的源、目 MAC，以及类型字段。例如：

```

R2#show adjacency detail
Protocol Interface      Address
IP          FastEthernet0/0    10.1.12.1(10)
                                     14 packets, 7640 bytes
                                     epoch 0
                                     sourced in sev-epoch 1
                                     Encap length 14
                                     CA004FEC0008CA014FEC00080800
                                     ARP

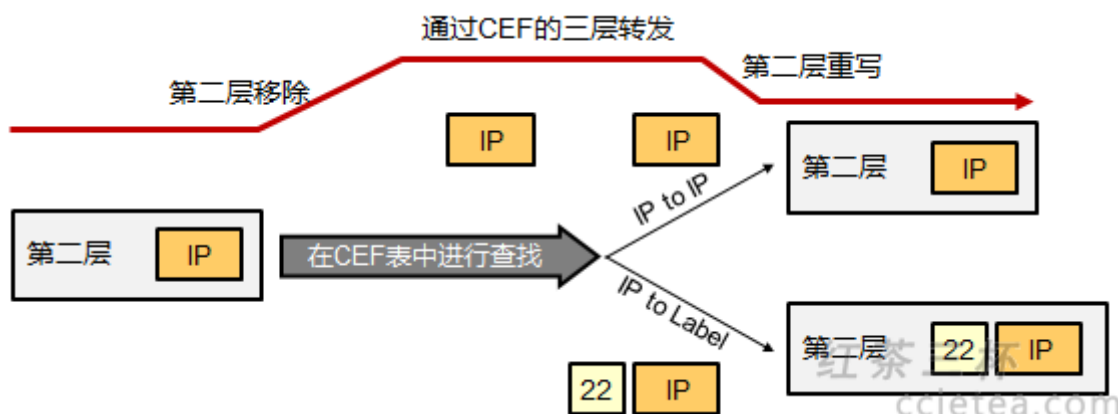
```

上面输出中的红色字体部分就是二层的 MAC 信息等等。

- **CEF 表**

进行三层转发决策。CEF 表维护着从路由表中提炼的核心信息，这些信息用于对接收到的报文执行转发决策，这些信息包括 IP 前缀、递归的下一跳地址和出站接口。CEF 表的一个重要的功能是它能够立即决策出递归前缀。

- **CEF 的实施**



当报文进入路由器的时候，路由器将第二层信息剥掉，随后在 CEF 表中查找目的 IP 地址，以进行转发。

转发决策的结果将会指向邻接表中的一条邻接条目，在邻接表中可以找到第二层需要重写的字符信息，路由器使用这些信息为数据帧构造一个新的第二层头部，然后再将该报文转发到指向下一跳的出站接口。

2.2 负载均衡方式

1. 基于 CEF

在 CEF 中，有两种主要的负载均衡方式，基于报文和基于目的。

- **(接口) ip load-sharing per-packet**

基于报文的负载均衡，所有报文在出站链路上进行轮循。如果你想要使用基于报文的 CEF 负载均衡的话，就需要在所有出站接口上都配置该命令。哥们实验验证过了，确实可以 per-packet 负载均衡。

- **(接口) ip load-sharing per-destination**

默认的负载均衡方式是基于目的地的负载均衡。CEF 的基于目的地负载均衡实际上是通过目的和源 IP 地址进行 HASH 后实现的。相反，在快速转发中，基于目的地负载均衡是严格按照目的 IP 地址来进行的。CEF 默认使用基于目的地址的负载均衡方式，因为基于报文的负载均衡方式可以持续性地在

不同路径中发送同一数据流的报文，那么在 IP 报文到达目的地的时候可能需要进行重新排队的工作。这可能会降低对诸如 VoIP 这样的流量的转发性能，或者报文如果不按顺序到达的话，服务质量也会降低，报文可能被丢弃，另外还会增加延迟抖动。

你可以使用命令 `show cef interface` 来检查使用了哪一种负载均衡模式，该命令可以给出在这个接口上配置的 CEF 信息。

```
R2#show cef interface f0/0
FastEthernet0/0 is up (if_number 4)
  Corresponding hwidb fast_if_number 4
  Corresponding hwidb firstsw->if_number 4
  Internet address is 10.1.12.2/24
  ICMP redirects are always sent
Per packet load-sharing is disabled
  IP unicast RPF check is disabled
  Inbound access list is not set
  Outbound access list is not set
  IP policy routing is disabled
  BGP based policy accounting on input is disabled
  BGP based policy accounting on output is disabled
  Hardware idb is FastEthernet0/0
  Fast switching type 1, interface type 18
  IP CEF switching enabled
  IP CEF switching turbo vector
  IP CEF turbo switching turbo vector
  IP prefix lookup IPv4 mtrie 8-8-8-8 optimized
  Input fast flags 0x0, Output fast flags 0x0
  ifindex 3(3)
  Slot Slot unit 0 VC -1
  Transmit limit accumulator 0x0 (0x0)
  IP MTU 1500
```

在 CISCO IOS 中，CEF 可以对源和目的 IP 地址进行哈希，将哈希的结果导入一张负载均衡表来实现负载均衡。这张表有 16 个哈希桶 bucket，每一个哈希桶都指向一个邻接关系，而多个桶可以同时指向一个邻接关系。可以使用 `show ip cef 路由前缀 internal` 这条隐藏命令来看：

```
R1#show ip cef 2.2.2.2 internal
```

2.2.2.2/32, version 55, epoch 0, per-destination sharing

0 packets, 0 bytes

tag information set

local tag: 100

via 10.1.13.3, FastEthernet1/0, 0 dependencies

traffic share 1

next hop 10.1.13.3, FastEthernet1/0

valid adjacency

tag rewrite with Fa1/0, 10.1.13.3, tags imposed: {}

via 10.1.12.2, FastEthernet0/0, 0 dependencies

traffic share 1

next hop 10.1.12.2, FastEthernet0/0

valid adjacency

tag rewrite with Fa0/0, 10.1.12.2, tags imposed: {}

0 packets, 0 bytes switched through the prefix

tmstats: external 0 packets, 0 bytes

internal 0 packets, 0 bytes

Load distribution: 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 (refcount 1)

| Hash | OK | Interface | Address | Packets | Tags imposed |
|------|----|-----------------|-----------|---------|--------------|
| 1 | Y | FastEthernet1/0 | 10.1.13.3 | 0 | none |
| 2 | Y | FastEthernet0/0 | 10.1.12.2 | 0 | none |
| 3 | Y | FastEthernet1/0 | 10.1.13.3 | 0 | none |
| 4 | Y | FastEthernet0/0 | 10.1.12.2 | 0 | none |
| 5 | Y | FastEthernet1/0 | 10.1.13.3 | 0 | none |
| 6 | Y | FastEthernet0/0 | 10.1.12.2 | 0 | none |
| 7 | Y | FastEthernet1/0 | 10.1.13.3 | 0 | none |
| 8 | Y | FastEthernet0/0 | 10.1.12.2 | 0 | none |
| 9 | Y | FastEthernet1/0 | 10.1.13.3 | 0 | none |
| 10 | Y | FastEthernet0/0 | 10.1.12.2 | 0 | none |
| 11 | Y | FastEthernet1/0 | 10.1.13.3 | 0 | none |
| 12 | Y | FastEthernet0/0 | 10.1.12.2 | 0 | none |
| 13 | Y | FastEthernet1/0 | 10.1.13.3 | 0 | none |

```

14    Y    FastEthernet0/0      10.1.12.2      0    none
15    Y    FastEthernet1/0      10.1.13.3      0    none
16    Y    FastEthernet0/0      10.1.12.2      0    none
refcount 6

```

要在基于目的的负载均衡中检验特定的流量使用的是哪一个出站接口，使用

Show ip cef exact-route src-addr dest-addr

例如前往一个目的地可能有多条等价路径，那么实际上可能在转发数据的时候，只是用了其中一条，用这条命令可以查看实际用于流量转发是哪一个接口。

2. 基于快速转发

快速交换意味着所有去往指定目的地的数据包都从相同的接口被发送出去，因此交换时间和处理器的占用率会大大降低。当去往相同网络内不同主机的数据包进入路由器且还一存在条可选路由时，路由器会在另一条路径上发送数据包到目的地。因此路由器能够做得最好的就是基于目标网络的均衡负载。

3. 过程交换

基于数据包的负载均衡和过程交换。过程交换就是对于每个数据包，路由器都要进行路由表查询和接口选择，然后再查询数据链路信息。因为每一次为数据包确定路由的过程都是相互独立的，所以不会强制去往相同目标网络的所有数据包使用相同的接口。为了在接口上打开过程交换功能，可以在接口下使用命令 no ip route-cache。对于 IPv6 什么也不需要，因为缺省情况下该功能是打开的。

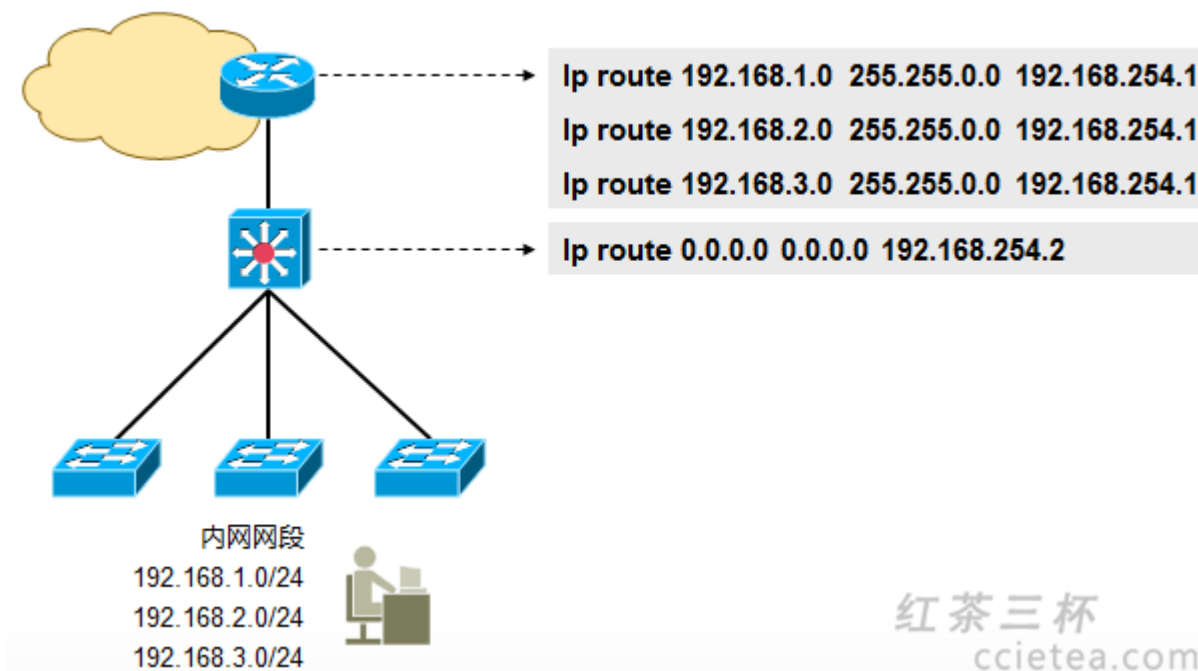
4. 哪一种交换方式会被用到

表 3-1 IOS 根据入站和出站接口的配置确定交换方法

| 入站配置 | 出站配置 | 所用的交换方法 |
|------|------|------------------|
| CEF | 过程 | CEF |
| CEF | 快速 | CEF |
| 过程 | CEF | 快速（如果是 IPv6 为过程） |
| 过程 | 快速 | 快速 |
| 快速 | CEF | 快速（如果是 IPv6 为过程） |
| 快速 | 过程 | 过程 |

3 静态路由

3.1 概述



路由器的天职，就是维护路由表以及利用路由表进行数据转发。而路由表中包含通过各种途径学习到的路由表项或路由条目，其中最简单最直接的方法，就是使用静态手工配置的方式，为路由器创建路由条目，这种方式最直接，可控性最高，配置也最简单。在小型的网络中，全网静态路由似乎没有什么问题，但是在一个大型网络中，如果纯用静态路由来做，工作量就非常大了，不仅仅工作量大，另外一个更重要的缺陷是静态路由无法根据网络拓扑结构的变更而做出调整，因此，在大规模网络中，我们往往采用静态+动态路由协议的方式来完成路由的部署。

3.2 静态路由的配置

```
ip route 192.168.10.0 255.255.255.0 192.168.1.1
```

- 使用指向下一跳的静态路由

```
ip route 192.168.10.0 255.255.255.0 fast 0/0
```

- 使用关联出接口的方式配置静态路由
- 该条目将作为直连网络输入到路由表中（如下）
- 如果出接口为广播型接口，可能会给接口下的节点造成额外的负担（ARP）或者，造成潜在的问题

```
Router#show ip route
```

```
Gateway of last resort is not set
```

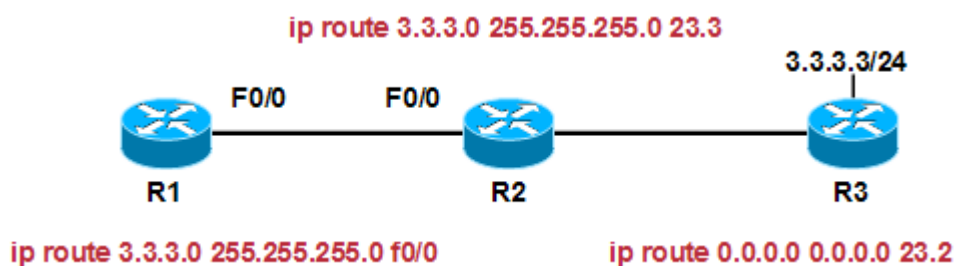
```
9.0.0.0/24 is subnetted, 2 subnets
```

```
C      9.9.12.0 is directly connected, FastEthernet0/0
```

```
S      9.9.23.0 is directly connected, FastEthernet0/0
```

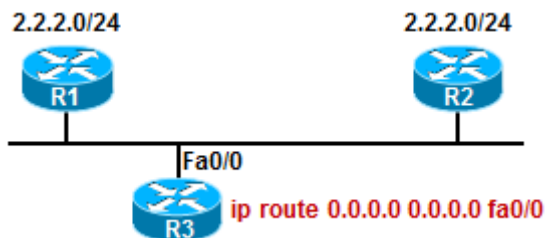
3.3 注意事项

1. 关联出接口的静态路由



正常情况下，R1 是能够 ping 通 3.3.3.3 的，但是如果把 R2 f0/0 口的代理 arp 关掉的话，就 ping 不通了。使用出口关联静态路由的话，路由器会将路由目的地址认为是本地链路，因此发 arp 请求 3.3.3.3 的 mac，而 R2 的 F0/0 口配置了代理 ARP，3.3.3.0 的网络 R2 本身又可达，因此 R2 会用自己 F0/0 口的 MAC 来回应这个 ARP 请求，实际上，这是一种 ARP 欺骗行为：)

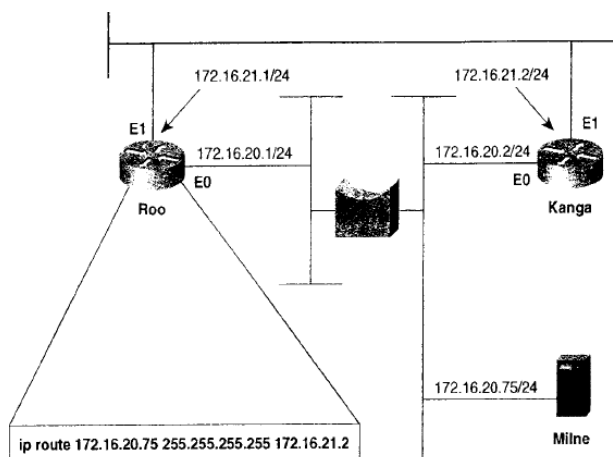
关联出接口的静态路由的另一个 CASE：



R1、及 R2 上均连接到 2.2.2.0 网络，此时 R3 去 ping 2.2.2.2，由于使用的出口关联的静态路由，R3 认为 2.2.2.0 是本地连接的网络，因此发 arp 请求，R1、R2、均能收到请求报文，且均回复请求报文（前提两

者的 f0/0 口都开了 arp 代理), 对于 R3, 会 R1、R2 中, 较晚到达的 arp 回应报文中的目的 MAC 放入 arp 表中, 关联 2.2.2.2 (架设为 R1), 此后但凡去往 2.2.2.2 的数据包, 均直接使用 R1 的 f0/0 口的 MAC 地址进行封装, 不再发 ARP 请求。而当 R1 的 f0/0 口 DOWN 时, 在 R3 的 2.2.2.2 的 ARP 条目超时之前, R3 均无法 ping 通 2.2.2.2, 直到 arp 条目超时, 或手工清除 2.2.2.2 的 arp 条目。

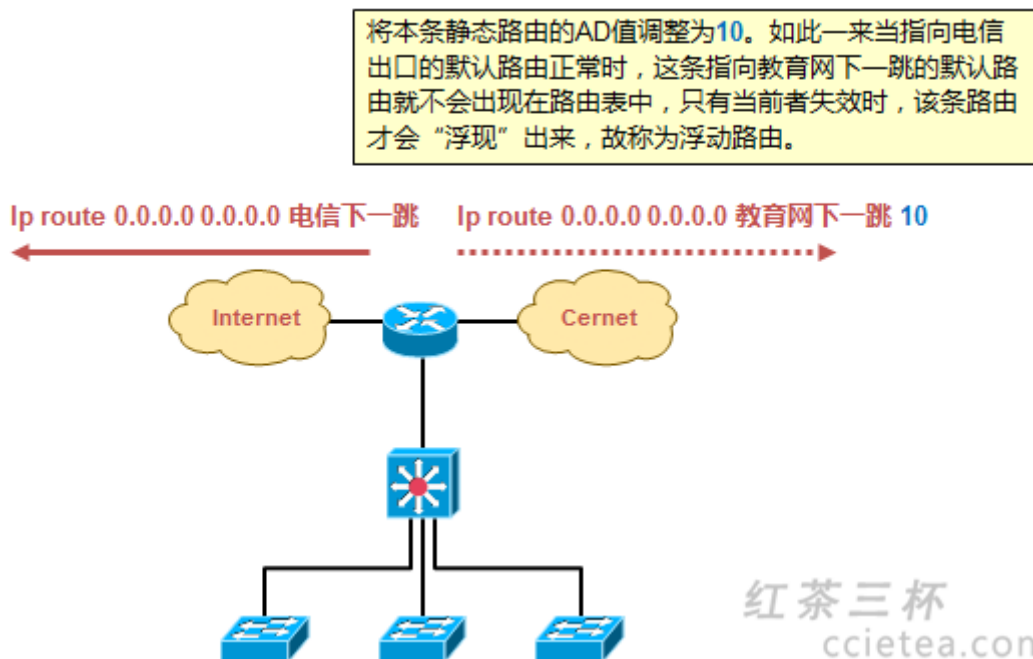
2. 代理 ARP 的另一个问题 (来自 TCP/IP 路由技术卷一)



ROO 上添加一条指向主机 Milne 的静态路由, 下一跳为 Kanga 的 E1 口, 目的是为了通过时常拥塞的网桥。但是却发现 Kanga 将发向主机 Milne 的数据包发给 Roo, 原因是学习到错误的 MAC 地址。

Kanga 发 arp 请求, 请求 Milne 的 mac, 这时 Roo 也收到该请求, 而其从 172.16.20.0 网段收到的请求, 同时查表发现自己通向主机 Milne 的路由所在的网络 (172.16.21.0) 与收到 arp 请求包的网路不一样, 并且接口开了 arp 代理, 因此回复自己的接口 mac。

3.4 浮动静态路由



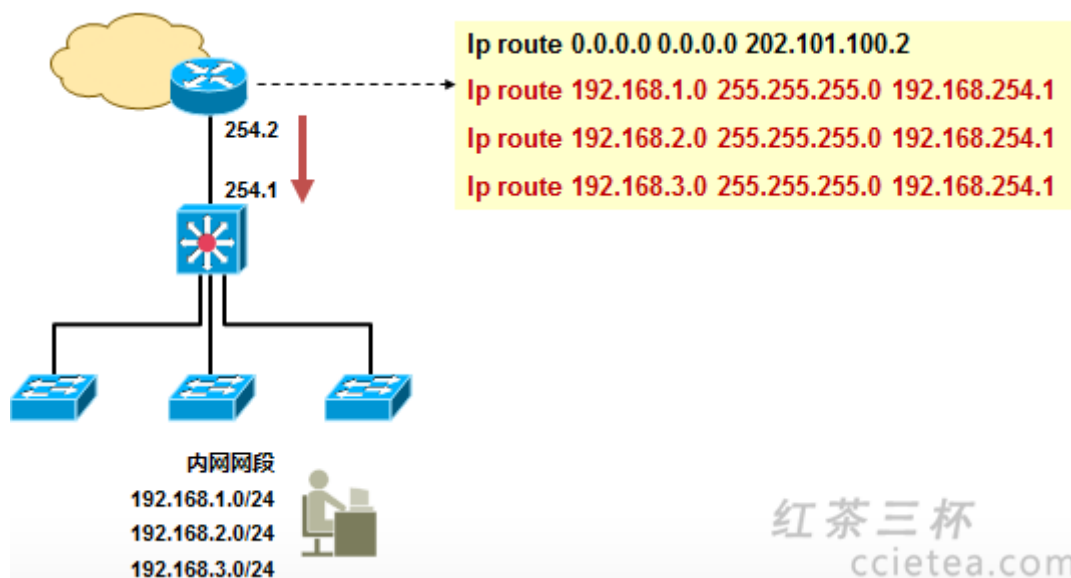
我们看上面的例子，在一个典型的教育园区网中，我们往往有两条以上的外网出口线路，假设一条为电信，一条为教育网出口。那么我们一般为出口路由器添加一条默认路由，指向电信的下一跳地址，为的是让内网用户能够通过电信线路访问 Internet 资源。然而，如果电信出口出现故障呢？我们可以在路由器上增加一条默认路由，我们知道，如果你配置两条默认路由，分别关联两个不同的下一跳，那么这两条路由将会在路由表中进行负载均衡，但是这里我们并不希望出现这个现象，我们希望一主一备，那么这条新增的默认路由就可以这么来配置：

```
Ip route 0.0.0.0 0.0.0.0 教育网下一跳 IP 10
```

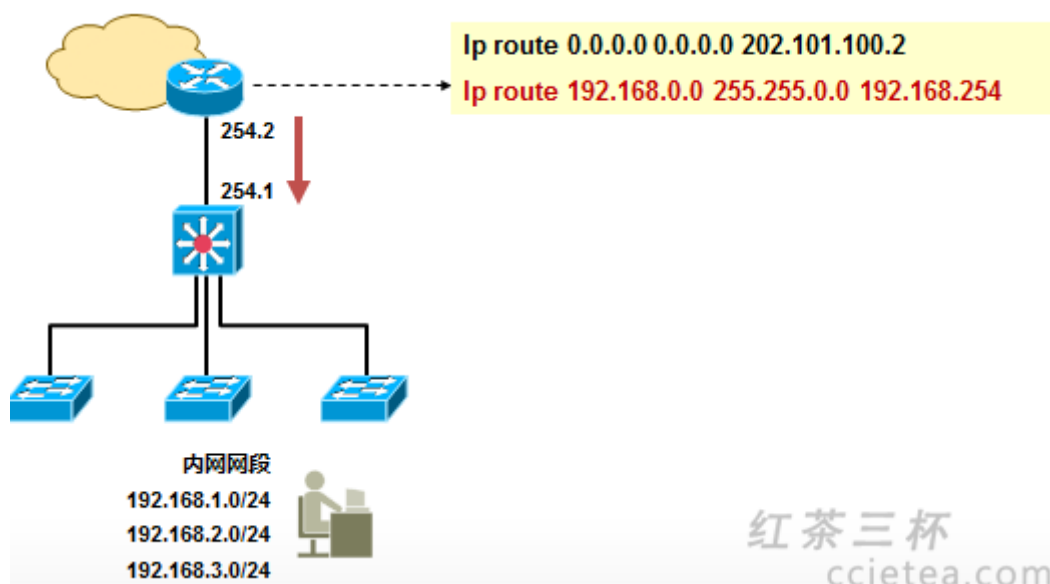
我们知道静态路由的 AD 值是 1，上面的配置方法，实际上是将该条静态路由的 AD 值修改为 10。这样一来我们有两条默认路由，一条指向电信出口，AD 值为默认的 1；另一条指向教育网出口，AD 值为 10。那么经过 PK 之后，毫无疑问指向电信的默认路由出现在了路由表里，而指向教育网这条默认路由，“猥琐”的躲了起来。当指向电信的默认路由失效的时候它就从路由表里消失了，那么这时候，指向教育网的这条默认路由，就“浮”了出来。

3.5 路由汇总

1. 路由汇总技术背景



我们看上面的场景，重点看出口路由器，这台出口路由器由于和三层交换机之间是三层链路，因此需配置到内网的回程路由，也就是上图中红色字体部分，这个场景中内网只有三个网段，因此配置了三条静态路由，但是如果有 100 个网段呢？岂不是要配 100 条路由？如此一来路由表就变的非常庞大和臃肿，维护和管理和非常不方便，更重要的是，这无疑浪费了设备的资源。因此从网络优化的角度，不管是何种网络场景何种网络模型，我们都需时刻关心网络中路由器路由表里的路由条目数量，是否足够优化，是否有可优化的空间。咋办？

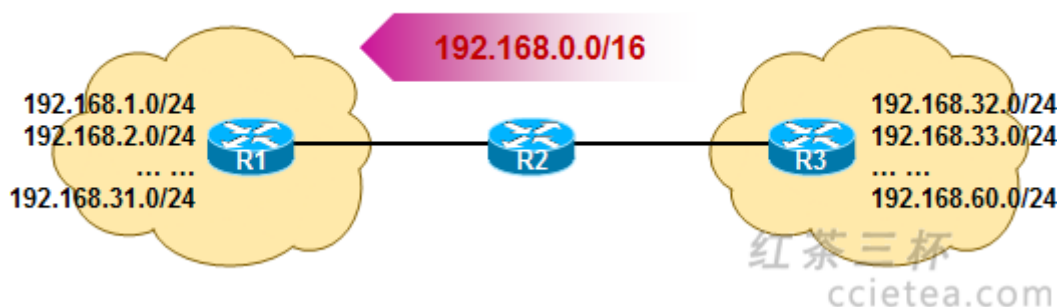


用路由汇总！看上图，前一个场景，我们需使用 3 条精细路由，而这里，我们却仅仅使用一条路由即可实现相同的效果，这条路由是上一个场景中三条明细路由的**汇总路由**。这样配置的一个直接好处就是，路由器的路由表条目大大减少了。这种操作方式我们称为路由汇总。路由汇总是一个非常重要的网络设计思想，通常在一个大中型的网络设计中，必须时刻考虑网络及路由的可优化性，路由汇总就是一个我们时常需要关

注的工具。这里实际上是部署了静态路由的汇总,当然除此之外我们也可以在动态路由协议中进行路由汇总,几乎所有的动态路由协议都支持路由汇总。

2. 路由精确汇总的算法

路由的汇总实际上是通过子网掩码的操作来完成的。对于下面的例子来说:



在 R2 上,为了到达 R1 下联的网络,R2 配置了使用路由汇总的工具,指了一条汇总路由:192.168.0.0/16 到 R1,虽然这确实起到了网络优化的目的,但是,这条汇总路由太“粗犷”了,它甚至将 R3 这一侧的网段也囊括在内,我们称这种行为不够精确。因此,一种理想的方式是,使用一个“刚刚好”囊括这些明细路由的汇总路由,这样一来就可以避免汇总不够精确的问题。

这里不得不强调一点,网络可以部署路由汇总的前提是 IP 子网及网络模型设计具备一定的科学性和合理性,因此路由汇总和网络的 IP 子网及网络模型的设计是息息相关的。

那么如何进行汇总路由的精确计算呢?下面我们来看一个例子:

假设我们有这么几个子网:

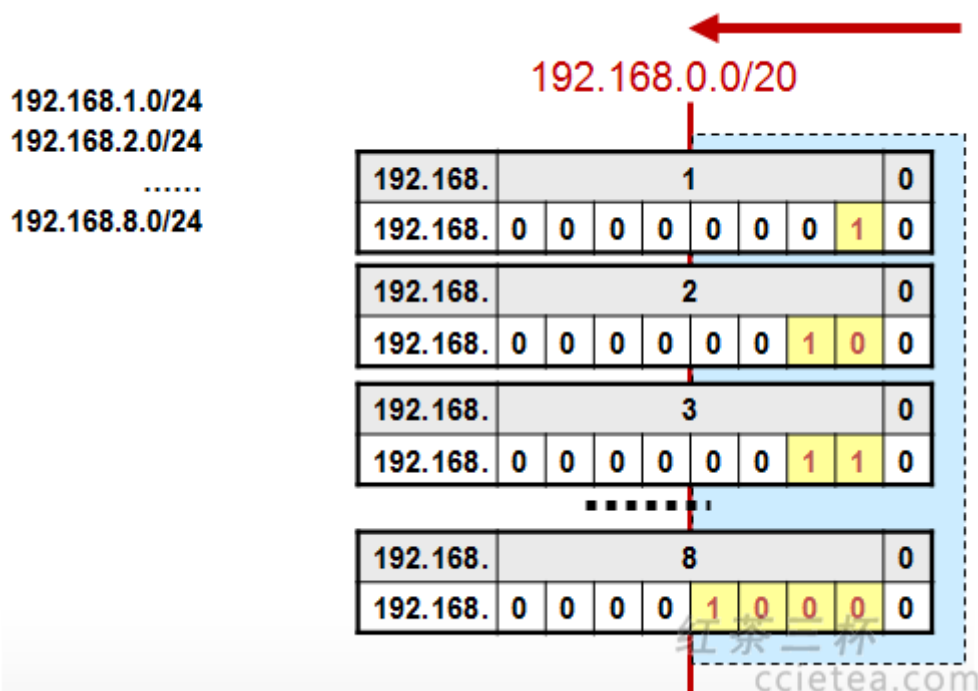
192.168.1.0/24

192.168.2.0/24

.....

192.168.8.0/24

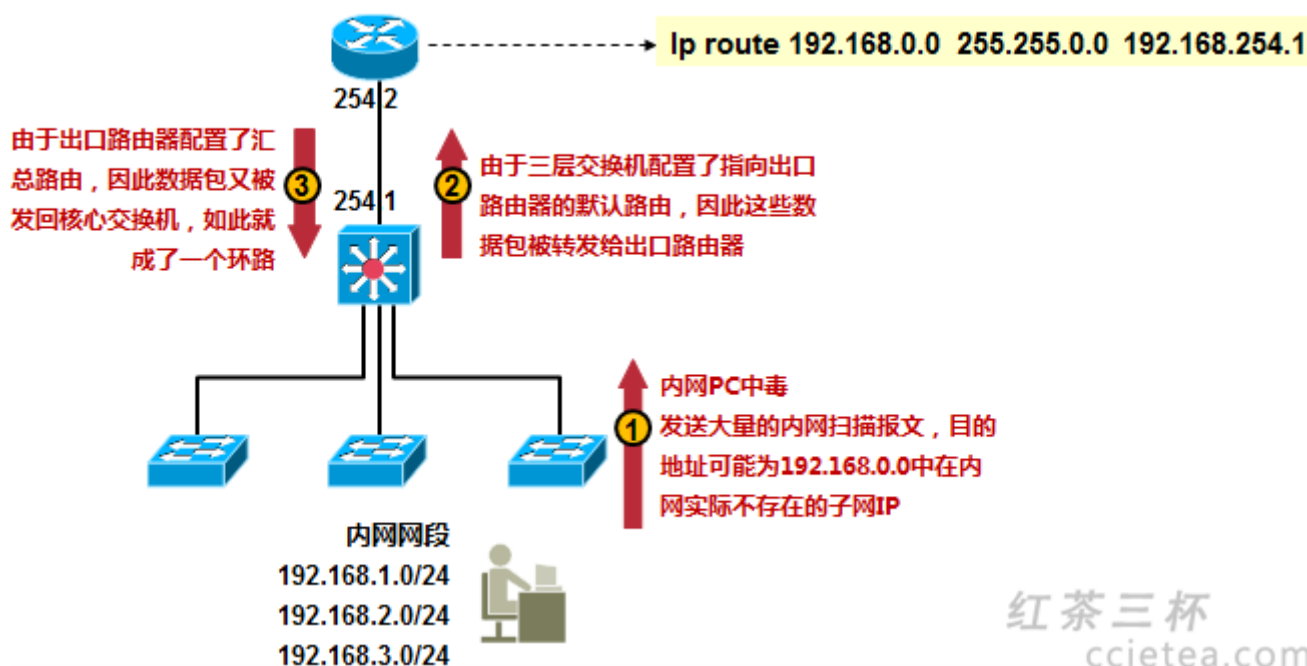
现在,我们要经过计算,得出刚刚好“囊括”这几个明细的汇总网段。



我们要做的事情非常简单,这些个明细子网是连续的,我们只要挑出首位的两到三个网络号来计算就足够了:

- 1) 将他们写成二进制形式,实际上,我们只要考虑第三个 8 位组即可,因为只有它是在变化的
- 2) 现在,我们要画一根竖线,这根线的左侧,每一个列二进制数都是一样的,线的右侧则无所谓,可以是变化的,注意这根竖线,可以从默认的掩码长度,也就是/24 开始,一格一格的往左移,直到你观察到线的左端每一列数值都相等,即可停下,这时候,这根线,所处的位置就是刚刚好。
- 3) 如上图,线的位置是 16+4=20,所以我们得到汇总地址:192.168.0.0/20,这就是一个最精确的汇总地址。

3. 路由汇总需注意的问题



路由汇总虽然确实是一个非常非常重要的思想和工具，但是使用起来要持谨慎态度，毕竟减少路由条目的同时，也降低了路由的颗粒度和精确性。看上图，在出口路由器上配置了静态汇总路由，下一跳是三层交换机。而三层交换机为了将访问外网的流量送到出口路由器，配置了一条默认路由，下一跳是出口路由器。

这个网络在流量正常的情况下不会有问题，但是，现在内网用户中毒了，于是这些 PC 开始疯狂的发送内网的扫描报文，这些报文的地址是一些 192.168 开头的不知名地址，甚至根本不存在的地址。数据包被送到了网关也就是三层交换机上，由于三层交换机配置了默认路由，因此这些数据包目的地被默认路由匹配并被引导到了出口路由器上，而出口路由器上部署了汇总路由，这些数据包的目的地址虽然在内网中不存在，但是却是这个汇总路由里的一个 IP，因此又被出口路由器转发回给三层交换机，接下去三层交换机又根据默认路由，将数据包转发回出口路由器，这就形成了数据的环路。

因此，从这里我们可以看出来，路由汇总，是有产生环路的风险的，解决上述问题的一个办法就是，我们在三层交换机上，增加一条静态路由：ip route 192.168.0.0 255.255.0.0 null 0，这样一来，当它收到访问 192.168 开头的、不存在的地址的数据包，就会直接丢弃。而正常的访问 192.168 内网其他子网的流量会根据最长匹配原则被正常转发。

这个思想被应用到了诸如 OSPF 等这类动态路由协议上，细心的童鞋会发现，你在 OSPF 中部署了路由汇总后，它会自动在本地产生一条指向 null0 的汇总路由，道理跟上面讲解的是一样的。

4 距离矢量路由协议

4.1 Basic

1. 距离矢量名称的由来是因为路由以矢量（距离，方向）的方式被通告出去的，其中距离是根据度量定义的，方向是根据下一跳路由器定义的
2. 定期更新、广播更新、路由表更新；依照传闻；距离矢量路由协议并不了解网络拓扑
3. 定义一个最大值：定义跳数的最大值 15 跳，16 跳为不可达，以避免路由选择环路
4. 通过水平分割消除路由选择环路

简单水平分割的规则是，从某接口发送的更新消息不包含从该接口收到的更新所包含的网络，换成人话说，就是我从这个接口收到的路由，就不会再从这个接口发回去。

5. 路由中毒



当 10.4.0.0 挂掉的时候，C 会立即发一条路由中毒消息（10.4.0.0 16 跳）然后通告出去；

B 收到这条中毒消息后，将 10.4.0.0 从路由表里抹去，但仍存在在 rip database 里，状态是 possible down，垃圾收集时间（Garbage collection CISCO 默认 60S）到后，路由被从 B 的 database 抹去。

6. 毒性逆转

同上图，4.0 挂掉后，C 会发送中毒消息，理论上 4.0 的路由是 C 通告给 B 的，根据水平分割原则，B 不能向 C 通告 4.0 的信息，但是带毒性逆转的水平分割打破了这个原则，B 会定期向 C 发送 4.0 的毒性逆转消息，以让 C 知道，他的邻居晓得了 4.0 挂掉的消息并且在眼巴巴的等着 4.0 原地满血复活，这样做的另一个好处是避免环路，至少 C 不会从 B 再去访问 4.0 了。

7. 用触发更新避免路由选择环路

新的路由表一般是定期发送给邻居路由器的，而触发更新(triggered update) 则是立即发送以响应路由表的变化。

8. 用抑制定时器(hold-down timer)防止路由选择环路

当一个 router 从邻居 router 收到一条更新，指示以前可达的网络现在不可达了，或有一个更大跳数的路由，则这个 router 将该路由为不可达并启动一个抑制定时器，如果在定时器满以前收到该路由又可达的更新，或者比以前的记录有更好的度量值，则该 router 标识这个路由可达并删除定时器。

4.2 RIPv1

4.2.1 Summary

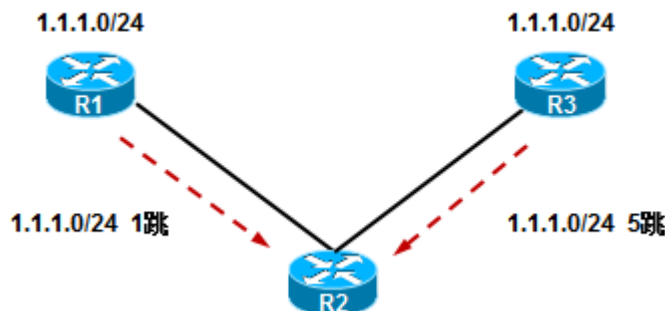
- V1 使用广播，默认每 30s 发送一次，采用 UDP 520 端口（源、目的端口）
V2 使用组播 224.0.0.9
- RIP 定义了请求消息和响应消息两种类型
请求消息：可以请求整张路由表，也可以请求具体路由信息。
 - 请求整个路由表：请求消息含有一个地址族标识字段为 0（地址为 0.0.0.0），度量值为 16 的单条路由，接收到这个请求的设备将通过单播方式向发出请求的地址回送它的整个路由表。
 - 请求具体路由信息：当需要获知某个或某些具体路由的信息，请求消息将与特定地址的路由条目一起发送。接受到请求的设备将根据请求消息逐个处理这些条目，并构成一个响应消息。
 更新消息：包含了整张路由表，周期性地更新
- RIP V1 支持最大 6 条，默认 4 条等价负载均衡。并且不支持验证。 V2 支持验证。

4.2.2 Timers

| | |
|--------------------------------|--|
| Update timer 更新计时器 | 默认 30s，RIP 更新周期 |
| Invalidation timer 无效计时器 | 默认 180s，也可称为 expiration timer 限时计时器或 timeout timer 用来限制停留在路由表中的路由未被更新的时间。 这个时间到了后，路由条目会变成 16 跳，标记不可达路由 possibly down。 Gateway of last resort is not set R 3.0.0.0/8 is possibly down , routing via 9.9.12.2, FastEthernet0/0 这个时候，他自己到如果收到数据包去 3.0.0.0 依然转发，但是不会向其他 rip 路由器去更新 3.0.0.0 的路由 |
| Garbage collection/flush timer | 一般为比无效计时器多 60-240 秒，如果连这个时间也超时了，那么路由条目彻底删除。思科默认 60s（注意是比 invalidtimer 多 60s），RFC 默认 120s |
| Holddown timer | 默认 6 个更新周期，即 180s。当收到一个更大跳数的条目时，该路由条目标记为不可达，同时启动抑制计时器，如果计时器超时后，同一个邻居仍然通告该 |

路由，则接受更新。

● 计时器验证



如上图，R1 向 R2 更新 1.1.1.0，1 跳，当 R1 DOWN 掉，而 R3 向 R2 更新 1.1.1.0 为 5 跳时，R2 上去往 R1 的路由会随着 invalid 计时器的到期变成 pdown 状态，随后进 holddown timer 的计时器

```

C    192.168.12.0/24 is directly connected, Serial0/0
     1.0.0.0/24 is subnetted, 1 subnets
R    1.1.1.0/24 is possibly down,
     routing via 192.168.12.1, Serial0/0
  
```

另外，如果 R1 更新 1.0.0.0 给 R2，如果 R1 passive 掉与 R2 互联的接口，invalid 到期后，R2 上关于 1.0.0.0 的路由会 pdown，这时 R2 仍然可以去往 1.0.0.0，但是不会将本路由传递给其他邻居，相反是仍然不停的发送中毒消息。这时候如果 no passive 掉 R1 的这个接口，R2 继续收到更新，但它不会马上恢复该路由，而是进入 holddown 计时器，直到 holddown 计时器超时后，才会接受该路由的更新。

Holddown timers 在 IOS 某些版本中被忽略，也就是即使设置了也不生效。

4.2.3 RIP database

每个运行 RIP 的路由器管理一个路由数据库，该路由数据库包含了到网络所有可达信宿的路由项，这些路由项包含下列信息：

- 目的地址：主机或网络的地址。
- 下一跳地址：为到达目的地，需要经过的相邻路由器的接口 IP 地址。
- 接口：转发报文的接口。
- Metric 值：本路由器到达目的地的开销，是一个 0~15 之间的整数。
- 路由时间：从路由项最后一次被修改到现在所经过的时间，路由项每次被修改时，路由时间重置为 0。

- 路由标记：区分内部路由协议路由和外部路由协议路由的标记。

4.2.4 RIP 消息格式

| | | | | |
|------------------------|------------------|----|-----|---|
| | 8 | 8 | 8 | 8 |
| 路由 条目 最大 25 条 | 命令 | 版本 | 未使用 | |
| | 地址族 | | 未使用 | |
| | IP 地址 | | | |
| | 未使用（ 设置为全 0 ） | | | |
| | 未使用（ 设置为全 0 ） | | | |
| | METRIC | | | |
| |（ 最大 25 条 ） | | | |

| 字段名 | 长度 | 含义 |
|--------------------------------|----------|---|
| Command | 8 比特 | 标识报文的类型： 1：Request 报文，向邻居请求全部或部分路由信息； 2：Response 报文，发送自己全部或部分路由信息，一个 Response 报文中最多包含 25 个路由表项。 |
| Version | 8 比特 | RIP 的版本号： 1：RIP-1； 2：RIP-2。 |
| Must be zero | 16/32 比特 | 必须为零字段。 |
| AFI(Address family identifier) | 16 比特 | 地址族标识，其值为 2 时表示 IP 协议。 |
| IP Address | 32 比特 | 该路由的目的 IP 地址，只能是自然网段的地址。 |
| Metric | 32 比特 | 路由的开销值。 |

每个 RIP 消息包含路由条目，最多 25 条，超过则需多条消息。

4.2.5 Ip rip trigger

在接口上配置了该命令后，便不在该方向周期性更新路由表，而是触发更新。路由表的更新将会变得最少，仅仅包括路由表最初的交换信息和路由表发生变化时的更新信息。这条命令仅仅在串行链路上有效，并且必须在链路的两端同时配置才会产生效果。

因为 RIP 出现的比较早，早期的路由器之间使用串行链路互联（点到点链路），所以 RIP 触发更新只支持在点到点链路上开启，对于 Frame-Relay 和以太网这样的多路访问接口中，不支持 RIP 触发更新，但是 Frame-Relay 点到点子接口被 RIP 认为是点到点链路，可以开启触发更新。

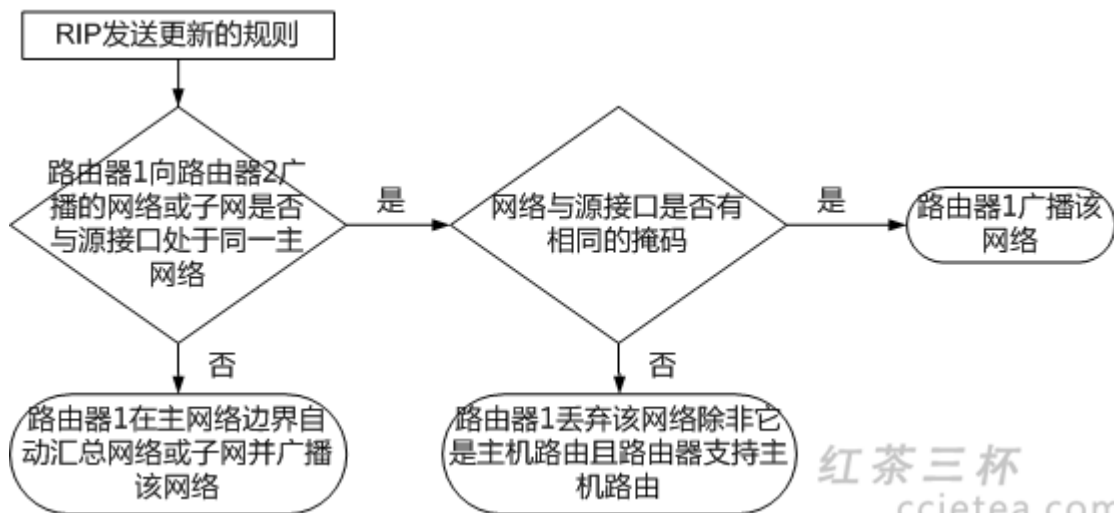
当开启触发更新后，链路两端的路由器之间不再周期性的发送路由更新，当然也带来另一个问题，路由表里的路由如果生存时间超时了也就挂了，因此要注意触发更新必须在链路两端接口上都进行配置，如此一来，路由更新会被标注 permanent 永久有效。

配置命令：接口模式下，在链路两端的接口上都要配置，完成后可 show ip rip database 查看路由详情

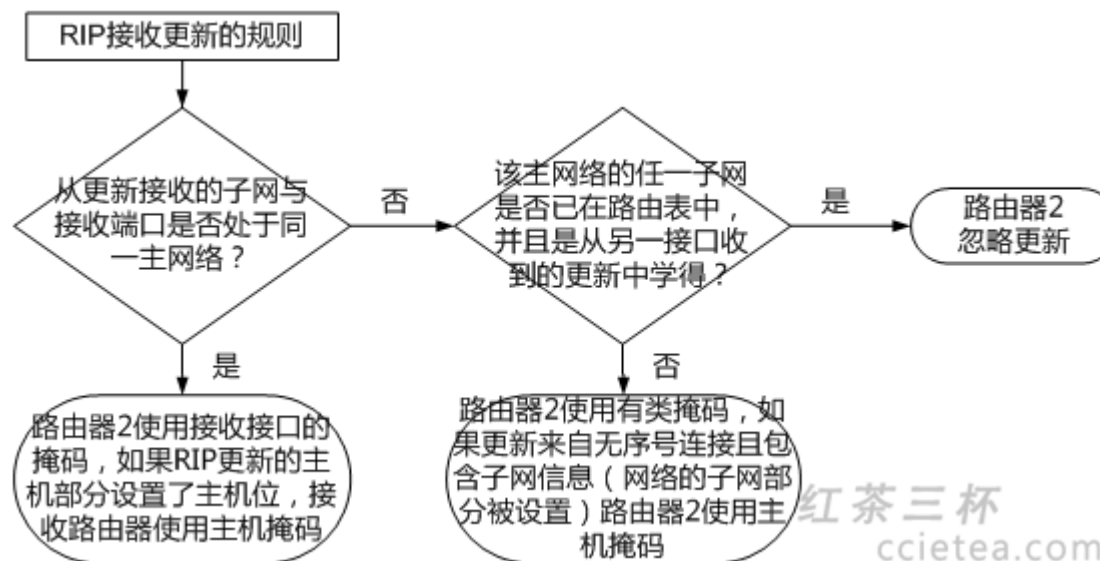
ip rip triggered

4.2.6 RIPV1 操作行为

1. RIP 更新行为

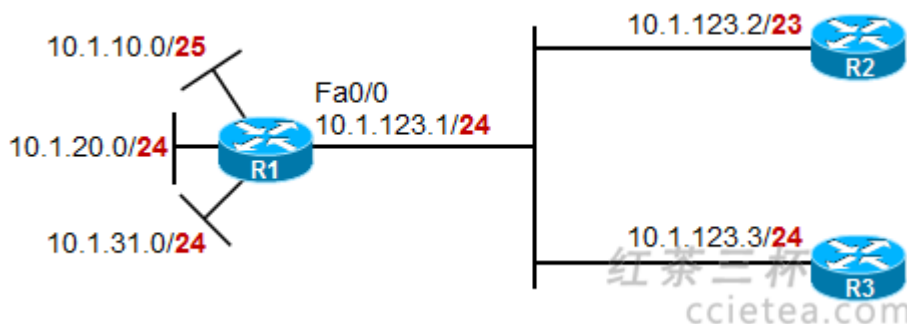


2. 接收行为



4.2.7 疑难解析

1. RIPv1 更新及接收规则验证



● 更新行为

10.1.10.0/25 这个子网, 不能被更新, 这是因为子网与更新源接口属于同一主类网络, 但掩码不同。

10.1.20.0/24 这条路由被 R1 以 “10.1.20.0” 更新出去, 这是因为子网掩码与更新接口掩码/24 一致。注意, v1 的更新包不含掩码, 更新的只有子网号。

10.1.31.0/24 这个子网, 由于和 R1 的更新源接口同主类同掩码, 因此被顺利更新 “10.1.31.0”

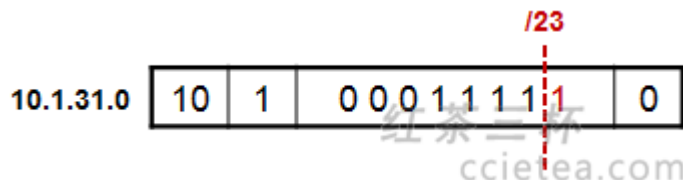
● 接收行为

“10.1.20.0” 在传递给 R2 后, 由于更新的子网与 R2 接口主网一致, 因此 R2 接收更新, 并且使用接口的掩码/23 作为接收的 RIP 子网的掩码, 那么 R2 本地装载的这条 RIP 路由就是 10.1.20.0/23。这里其实路由的前缀长度就错误了, 我们可以在 R2 接口上再配置个 secondary address, 这样一来, RIP 将使用 secondary address (如果配置了的话) 的掩码进行路由接收操作。例如给 R2 配置个 10.1.123.222/24,

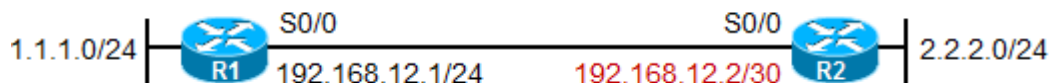
那么收到的路由就变成了 10.1.20.0/24，掩码就对了。

“10.1.20.0”在传递给 R3 后，由于 RIP 更新的子网与 R3 接口子网属同一主类网络，因此 R3 接收更新，并且用接口的掩码作为 RIP 路由的掩码，于是 R3 路由表中装载的路由就是：10.1.20.0/24

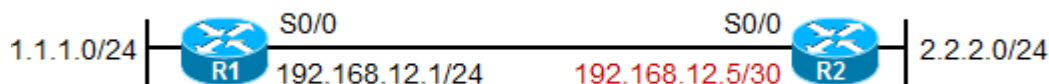
“10.1.31.0”在传递给 R2 后，由于 R2 拿接收接口的掩码去运算，发现 10.1.31.0 在主机部分有位置 1，因此将该子网存进路由表，并将其视为主机路由：10.1.31.0/32



2. RIP 更新源问题



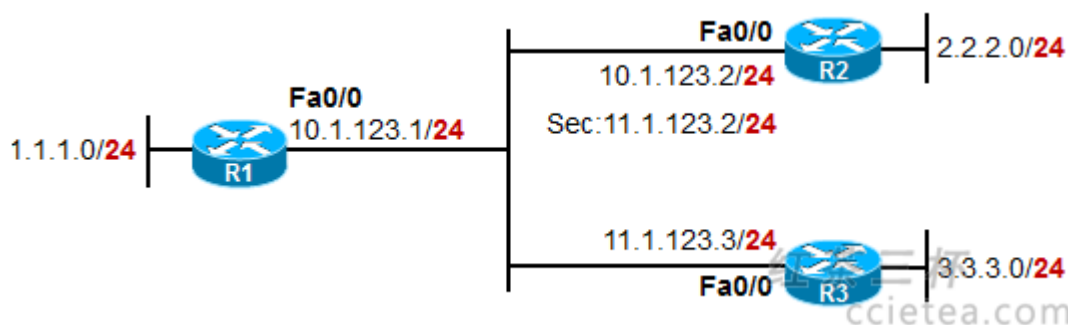
R1、R2 之间跑 RIP（宣告直连及各自的 LOOPBACK），这时 R1 发出来的更新包，源地址为 192.168.12.1，R2 收到包检查后发现，这是与自己接受更新包的接口同一子网的 IP，因此接受这个更新。所以这个测试，虽然两边接口掩码不同，但是互相之间都能接受对方的更新，同时 1.1.1.1 及 2.2.2.2 之间也都能 ping 通。



变更拓扑后，R2 的接口 IP 变了。这时候 R1 发过来的 RIP 更新，源地址仍是 192.168.12.1，R2 收到后，发现与本地接受更新的接口 IP 不在一个子网，因此 R2 忽略 R1 的更新消息。另外，R2 发给 R1 的消息，被 R1 接受了，因为 R1 认为 R2 发过来的 RIP 更新消息源地址 192.168.12.5 与自己的接口在同一个子网。

当 R2 上配置：no validate-update-source 后，R2 则不会对 RIP 更新消息源进行校验，因此 R2 会接受 R1 发来的更新，这时候双方各有对方的路由，但是 1.1.1.1 无法与 2.2.2.2 ping 通。

3. RIP secondary address 问题



IP 配置如上，R2 接口配置了两个 IP 地址，并都做了宣告全网跑 RIP。

- R2 会同时用主地址和辅助地址进行 validate-update-source ,因此 R2 能学习到 2.2.2.0 及 3.3.3.0 路由。
- 当同时存在主地址和辅助地址 , RIPV1 路由器将使用本地 “接收路由更新包” 的接口的辅助地址的掩码进行接收操作 (这点在上面的实验已经验证过了)。
- 当同时存在主地址和辅助地址 , RIP 将同时使用主、辅助地址作为源来发送路由更新。所以 R1 和 R3 都会学习到 2.2.2.0 的路由更新。但是 R3 学习不到 1.1.1.0 , 一方面是因为 R1 发给 R3 的更新被 R3 忽略 (更新源校验失败) ,另一方面 R2 也不会将从 R1 收到的 1.1.1.0 的路由 ,再经过相同的接口、以 11.1.123.2 作为更新源 , 发给 R3。
- 另一点 , R2 会将 10.0.0.0、11.0.0.0 (辅助地址所在网段) 以及 1.0.0.0、3.3.3.0 都从自己的 loopback 更新出去 ,然而 ,它不会将自己的辅助地址 11.0.0.0 网段从 fast0/0 发送出去给 R1(除非关闭水平分割)。

4.3 RIPv2

4.3.1 改进

在 RIPv1 的基础上增加了

1. 外部路由标记

通过在 RIP 中使用路由标记 , 就能在其他协议中 , 控制相关路由的重发布。当重发布到其他协议时 , RIP 路由只需比较赋予他们的标记而不用比较整个路由。

2. VLSM 支持

3. 组播能力

使用 224.0.0.9

4. 认证

5. 下一跳

下一跳(Next Hop)—如果存在的话,它标识一个比通告路由器的地址更好的下一跳地址。换句话说,它指出的下一跳地址,其度量值比在同一个子网上的通告路由器更靠近目的地。如果这个字段设置为全 0(0.0.0.0),说明通告路由器的地址是最优的下一跳地址。

4.3.2 消息类型

| | | | | |
|------------------------|-------------------|----|------|---|
| 路由 条目 最大 25 条 | 8 | 8 | 8 | 8 |
| | 命令 | 版本 | 未使用 | |
| | 地址族 | | 路由标记 | |
| | IP 地址 | | | |
| | 子网掩码 | | | |
| | 下一跳 | | | |
| | METRIC | | | |
| | (最大 25 条) | | | |

| 字段名 | 长度 | 含义 |
|-----------------------------------|-------|---|
| Command | 8 比特 | 标识报文的类型： 1：Request 报文，向邻居请求全部或部分路由信息； 2：Response 报文，发送自己全部或部分路由信息，一个 Response 报文中最多包含 25 个路由表项。 |
| Version | 8 比特 | RIP 的版本号： 1：RIP-1； 2：RIP-2； |
| Must be zero | 16 比特 | 必须为零字段。 |
| AFI (Address Family Identifier) | 16 比特 | 地址族标识，其值为 2 时表示 IP 协议。 |
| Route Tag | 16 比特 | 外部路由标记。 |
| IP Address | 32 比特 | 该路由的目的 IP 地址，可以是自然网段的地址，也可以是子网地址或主机地址。 |
| Subnet Mask | 32 比特 | 目的地址的掩码。 |
| Next Hop | 32 比特 | 提供一个更好的下一跳地址。如果为 0.0.0.0，则表示发布此路由的路由器地址就是最优下一跳地址。 |
| Metric | 32 比特 | 路由的开销值。 |

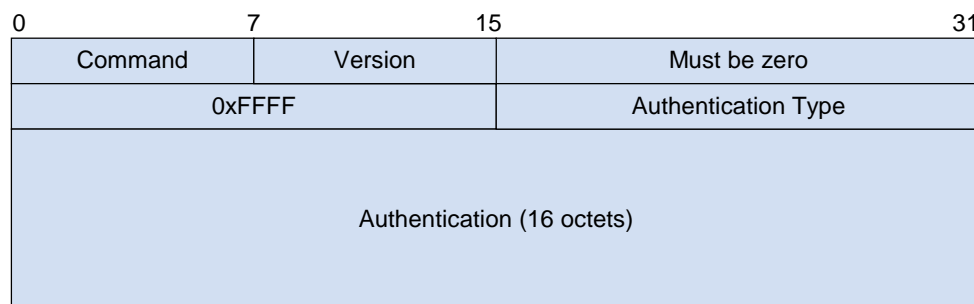
其中地址族消息默认总是 2，当请求整个路由表时为 0

关于下一跳及路由标记，详见卷一 189 页

4.3.3 认证

RIPv2 是通过更改 RIP 消息中正常情况下应该是第一个路由条目的字段来支持认证的,在含有认证的单个更新消息中,最大可以携带的路由条目被减少到了 24 个。

认证是通过设置地址族标识字段为全 1 (0xFFFF) 来标识的。



| 字段名 | 长度 | 含义 |
|---------------------|-------|--|
| Command | 8 比特 | 标识报文的类型： 1：Request 报文，向邻居请求全部或部分路由信息； 2：Reponse 报文，发送自己全部或部分路由信息，一个 Response 报文中最多包含 25 个路由表项。 |
| Version | 8 比特 | RIP 的版本号： 1：RIP-1； 2：RIP-2； |
| Must be zero | 16 比特 | 必须为零字段。 |
| 0xFFFF | 16 比特 | 验证项标识，表示整个路由报文需要验证。 |
| Authentication Type | 16 比特 | 验证类型： 2：明文验证； 3：MD5 验证。 |
| Authentication | 16 字节 | 验证字，当使用明文验证时包含了密码信息。 |

4.3.4 兼容性

默认情况下 CISCO 路由器上运行的 RIP 发送 v1，接收 v1 v2，如果显示声明为 v1（在 RIP 进程中使用 version 1 命令）则只发送和接收 v1 报文，v2 默认只发送和接收 v2 报文

RFC1723 定义了几个兼容性开关

- RIP-1 只发送 RIPv1 更新
- RIP-1 兼容性 RIPv2 使用广播的形式发送更新，使得 V1 能收到
- RIP-2 RIPv2 使用组播更新
- None 不发送更新（passive）

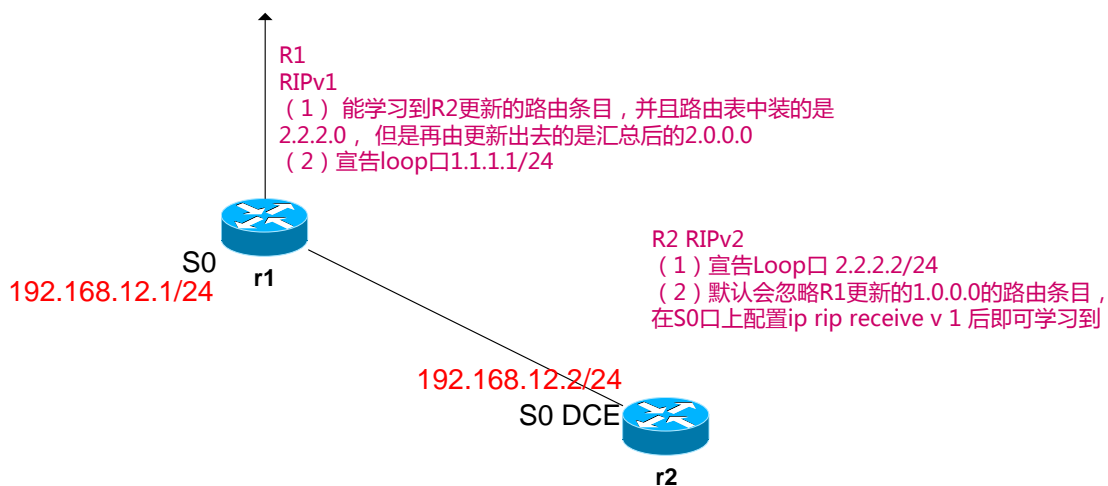
RFC1723 定义了一个接受控制开关

- RIP-1 Only

- RIP-2 Only
- Both
- None 使用分发列表或访问控制列表 deny 521 端口。

v1 默认发送 v1 接收 v1,v2

v2 默认发送 v2 接收 v2



4.3.5 高级特性

1. 被动接口

```
Passive-interface Fa0/0
```

接口将只收更新，不发更新

2. 单播更新

```
Passive-interface Fa0/0
```

```
neighbour xxxxxx
```

3. 偏移列表

手动修改 RIP 的跳数。首先配置匹配的路由

```
Offset-list Acl 号 in/out 偏移值 接口
```

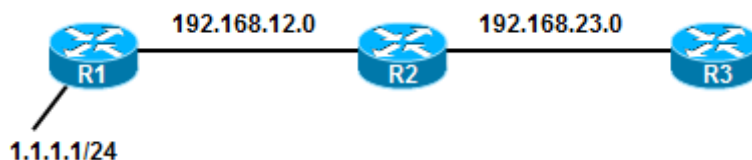
//该偏移值是增加在正常路由条目中显示的值上，是在原来的基础上增加而不是替换原来的跳数。

4. RIPv2 手工汇总

```
Ip summary-address rip xxxx yyyy      //注意，这个汇总不支持 CIDR 超网。
```

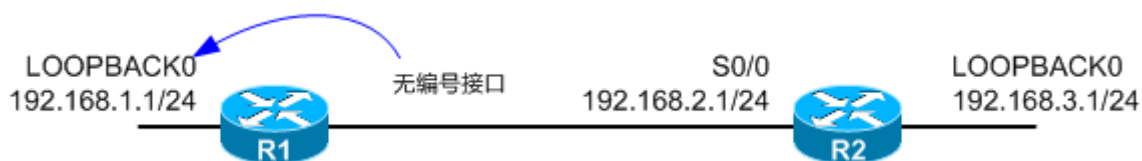
4.3.6 疑难解析

1. RIP 基于传闻的更新



运行 RIPv2，默认 R1 更新 1.0.0.0/8 的汇总给 R2，在 R1 上关闭自动汇总，则 R2 学习到 1.1.1.0/24，同时由于自己仍然开启自动汇总，于是更新给 R3 的是 1.0.0.0/8。这时候如果在 R2 上 `ip route 1.1.1.0 255.255.255.0 null0`，则 R2 上路由表关于 1.1.1.0 就变成了 static 静态路由了，则 R2 不再向 R3 更新 RIP 的 1.0.0.0/8 的路由了

2. 使用无编号地址传递 RIP 更新(无效源)



R2 会忽略 R1 发送过来的更新，因为“有源检查”，从 R1 发送过来的 RIP 更新与 R2 的 S0/0 口不在同一子网，所以 R2 忽略 R1 发来的路由更新。这个时候只要在 R2 的 RIP 进程下 `no validate-update-source` 即可。

4.4 RIPng

请看红茶三杯《IPv6 笔记》，访问 ccietea.com 获取最新版本

4.5 参考资料

| 文档编号 | 描述 |
|---------|---|
| RFC1058 | Routing Information Protocol |
| RFC1723 | RIP Version 2 Carrying Additional Information |
| RFC1721 | RIP Version 2 Protocol Analysis |

| 文档编号 | 描述 |
|---------|--|
| RFC1722 | RIP Version 2 Protocol Applicability Statement |
| RFC1724 | RIP Version 2 MIB Extension |
| RFC2082 | RIP-2 MD5 Authentication |
| RFC2080 | RIPng for IPv6 |

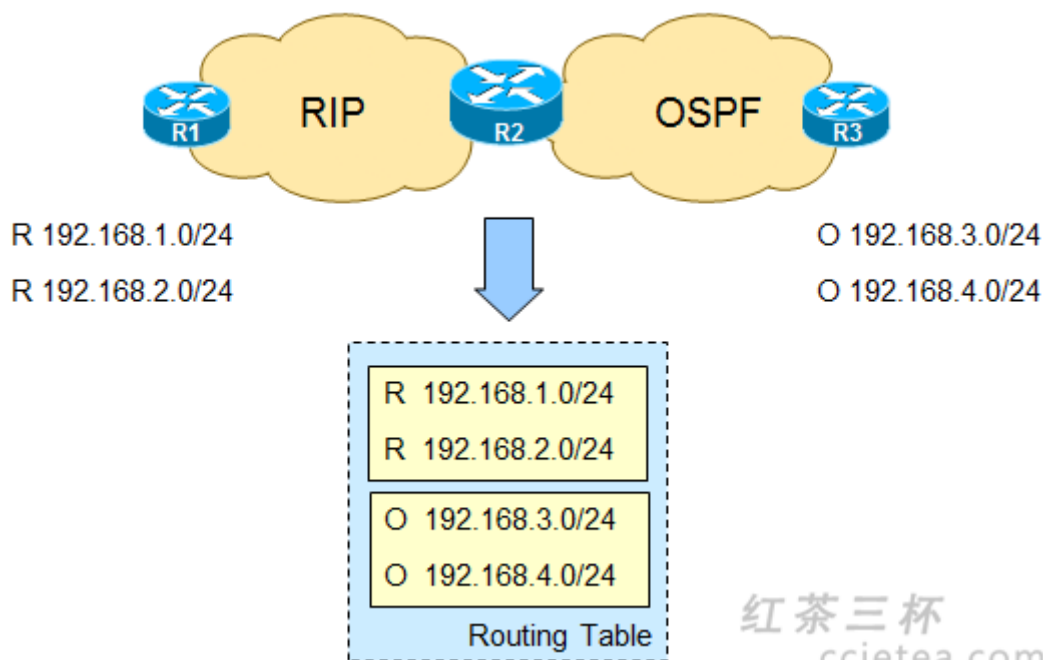
5 路由重发布 (Redistributing Routing Protocols)

5.1 技术背景

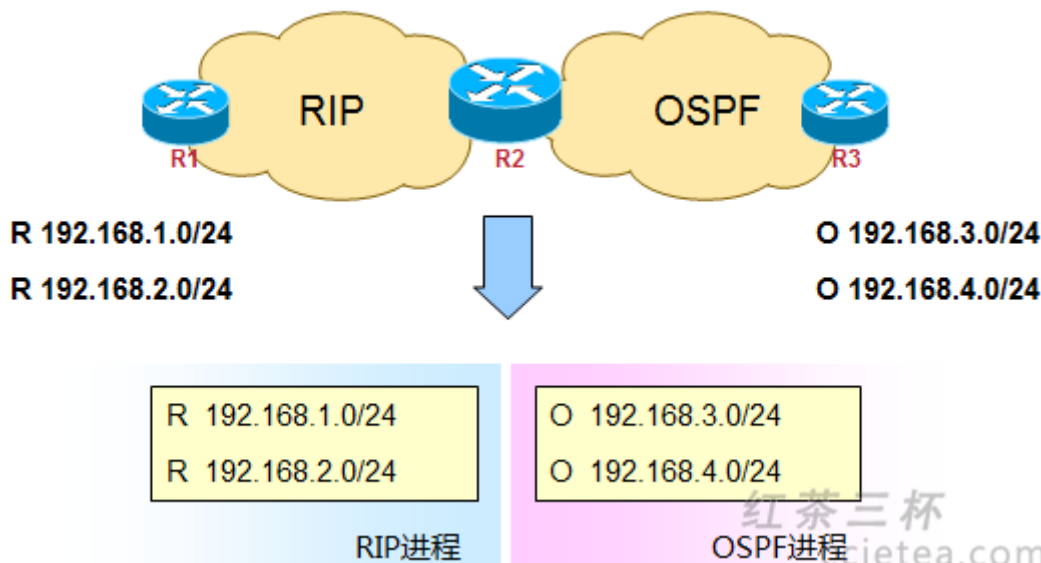
在现实的网络环境中，我们可能会遇到，一个网络环境中，同时存在两种或两种以上的路由协议的情况，例如：

- 多厂商的路由环境
- 网络合并（同一协议或是不同协议）
- 从旧的路由协议过渡到新的路由协议
- 路由策略的需要（可靠性、冗余性、分流模型等）

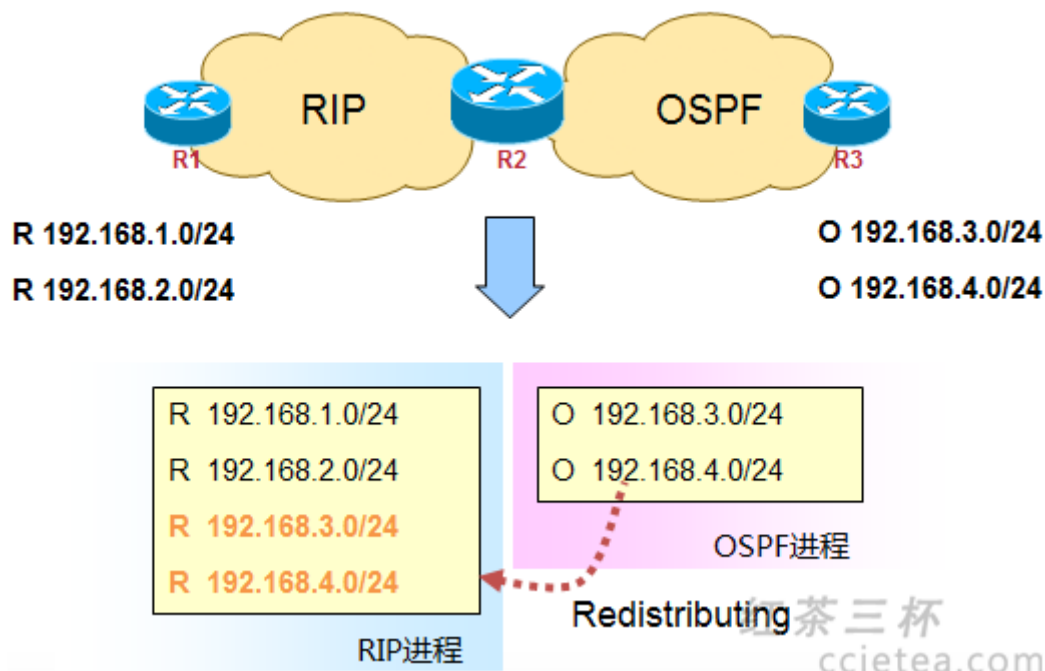
在同一个网络拓扑结构中，如果存在两种不同的路由协议，由于不同的路由协议的机理各有不同，对路由的理解也不相同，这就在网络中造成了路由信息的隔离，然而由于这很有可能是同一个自治系统内的网络，全网需要互通，这时候咋办？这就需要使用路由重发布了。



我们看上图，R1 与 R2 之间运行 RIP 来交互路由信息，R2 通过 RIP 学习到了 R1 发布过来的 192.168.1.0/24 及 2.0/24 的路由信息，装载进路由表，标记为 R。同时 R2 与 R3 又运行 OSPF，建立起 OSPF 邻接关系，R2 也从 R3、通过 OSPF 学习到了两条路由：3.0 及 4.0/24，也装载进了路由表，标记为 O。那么这样一来，对于 R2 而言，它自己就有了去往全网的路由，但是，在 R2 内部，我们可以这么形象的理解：它不会将从 RIP 学习过来的路由，变成 OSPF 路由告诉给 R3，也不会将从 OSPF 学习来的路由，变成 RIP 路由告诉给 R1。对于 R2 而言，虽然它自己的路由表里有完整的路由信息，但是，就好像冥冥之中，R 和 O 的条目之间有道鸿沟，无法逾越。而 R2 也就成了 RIP 及 OSPF 域的分界点。那么如何能够让 R1 学习到来自 OSPF 的路由，让 R3 学习到来自 RIP 的路由呢？关键点在于 R2 上，通过在 R2 上部署**路由重发布**，可以实现路由信息在不同路由选择域间的传递。



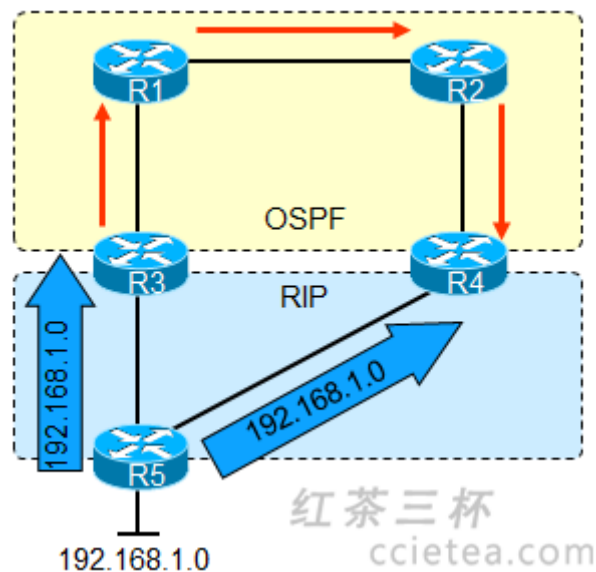
上图是初始状态。



上图中，我们开始在 R2 上执行重发布的动作，我们将 OSPF 的路由“注入”到了 RIP 进程之中，如此一来，R2 就会将 3.0/24 及 4.0/24 这两条 OSPF 路由“翻译”成 RIP，然后传递给 R1。R1 也就能够学习到 3.0 和 4.0 路由了。注意重发布的执行地点，是在 R2 上，也就是在路由选择域的分界点上执行，另外，路由重发布是有方向的，例如刚才我们执行完相关动作后，R3 还是没有 R1 的路由的，需进一步在 R2 上，将 RIP 路由重发布进 OSPF，才能让 R3 学习到 1.0/24 及 2.0/24 路由。

5.2 实施要点

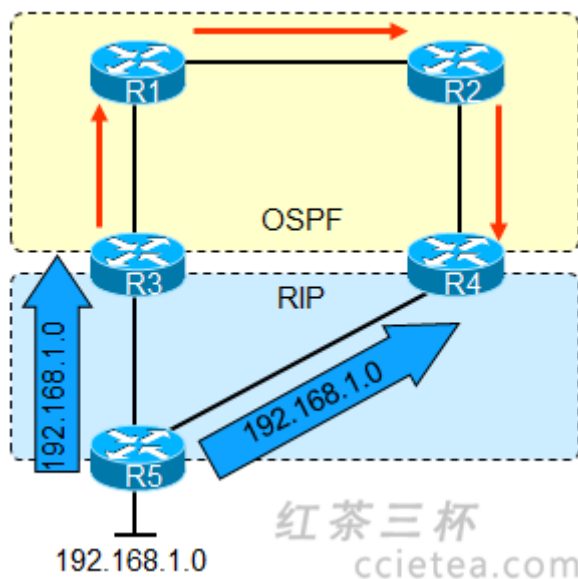
5.2.1 路由 feedback



路由的 Feedback（回馈）是一个在部署路由重发布时需要非常注意的一个现象。如上图所示，R5 将 192.168.1.0 宣告进了 RIP，R3 及 R4 都能够学习到这条路由，并且装载进自己的全局路由表。那么如果我们在 R3 上部署 RIP 到 OSPF 的双向重发布，会发生什么事情呢？我们假设在 R3 上先完成的配置，192.168.1.0 这条路由将被 R3 注入到 OSPF 中，并被更新给 R1，再由 R2 更新给 R4，此刻，R4 同时从 OSPF 及 RIP 都学习到了这条路由，它会作何优选？当然是优选 OSPF 的，因为 AD 小，所以它的路由表里，关于 192.168.1.0 的路由是 OSPF 的。这样一来，对于 R4 而言，它去往 192.168.1.0，就存在**次优路径**，也就是说，绕远路了，走 R2-R3-R3-R5 这条路径。并且由于路由表里没了 RIP 路由，自然 RIP 向 OSPF 重发布就失败了，更糟糕的是，R4 上关于 192.168.1.0 的 OSPF 路由更会被重新注入 RIP（因为我们部署的是双向重发布），这就是路由 Feedback，路由被灌回来了。

因此在部署重发布时，这个问题是需要格外注意的，至于问题如何规避，在后面的内容中，我再做介绍。

5.2.2 管理距离问题

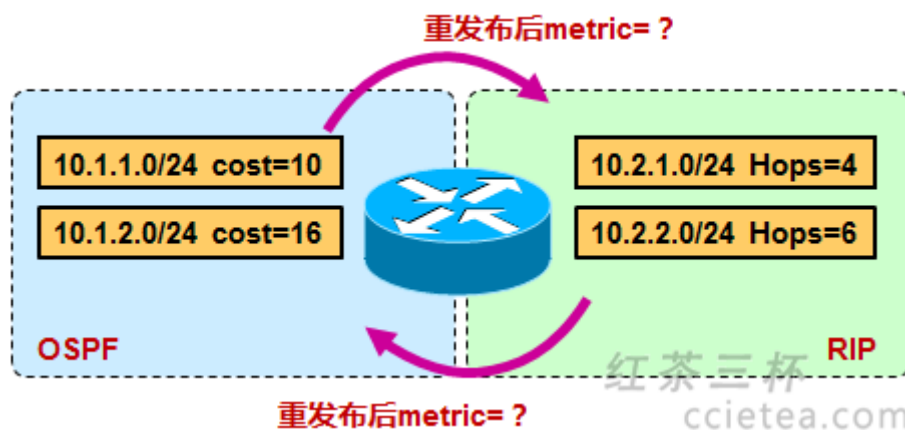


在上面这个例子中，我们提到的现象，R4 会同时从 OSPF 及 RIP 都获知到 192.168.1.0/24 的路由，最终 R4 会选择 OSPF 的路由。这是我们在这个环境中，不愿看到的现象，因为这样一来就造成了次优路径。几种常见的路由协议其 AD 值列举如下：

| 路由来源 | 管理距离 |
|-------------|------|
| 直连接口 | 0 |
| 静态路由 | 1 |
| EIGRP汇总路由 | 5 |
| 外部BGP | 20 |
| 内部EIGRP | 90 |
| OSPF | 110 |
| IS-IS | 115 |
| RIPv1、RIPv2 | 120 |
| 外部EIGRP | 170 |
| 内部BGP | 200 |
| 未知 | 255 |

值得注意的是我们可以通过在 R4 上，特定的协议进程中手工修改该路由协议的管理距离，从而达到影响路由器本身路由选择的目的，例如在 R4 上，我们将 OSPF 针对外部路由的默认管理距离 110，修改为 130，比 RIP 的管理距离 120 更大，这样一来，R4 在这个环境中，针对 192.168.1.0 就会优选 RIP 的路由，就可以规避次优路径以及路由 feedback 的问题了。

5.2.3 Metric 问题



要知道，每一种路由选择协议，对于路由 Metric 度量值的理解是不同的，OSPF 是用 cost 开销来衡量一条路由，RIP 是用跳数，EIGRP 是用混合的各种元素，那么当我将一些路由，从某一中路由协议重发布到另一种路由协议中，这些路由的 metric 会作何变化呢？方式之一是，你可以在执行重发布的动作的时候，手工进行修改，具体改成什么值，要看具体的环境需求，这个我们后面会举例别担心，哥们就这么实诚 :)。方式之二是，采用默认的动作，也就是在路由协议之间重发布时所定义的种子度量。

所谓种子度量，指的就是当，我将一条路由，从外部路由选择协议重发布到本路由选择协议中时，如果没有手工指定路由的 metric，而使用的默认的 metric。看下表（下表是一个公认的默认值，可在路由进程中使用 default-metric 修改）：

| 将路由重分发到该协议 | 默认种子度量值 |
|------------|-----------------------------------|
| RIP | 0，视为无穷大 |
| IGRP/EIGRP | 0，视为无穷大 |
| OSPF | BGP 进来为 1，其他路由为 20，OSPF 之间度量值保持不变 |
| IS-IS | 0 |
| BGP | BGP 度量值被设置为 IGP 度量值 |

注意，以上是从其他动态路由协议重发布进本路由协议时的默认 metric。

而如果是重发布本地直连或静态路由，则情况就有变化了，如下：

- EIGRP 请见红茶三杯 EIGRP 笔记（访问 ccietea.com）
- RIP 重发布直连如果没有设置 metric，则默认 1 跳传给邻居（邻居直接使用这个 1 跳作为 metric）；重发布静态路由默认 metric=1，使用 default-metric 可以修改这个默认值，这条命令对重发布直连接口的 metric 无影响。
- OSPF 重发布直连接口默认 cost20；重发布静态路由默认 cost20；使用 default-metric 可以修改重发布静态路由以及其他路由协议的路由进 OSPF 后的默认 cost，只不过这条命令对重发布直连接口无效。

5.2.4 有类、无类路由选择协议的重发布

这个话题，个人觉得没必要研究了，对有类路由协议的研究个人感觉意义不大。无论是实际应用或是 LAB 考试，都不再涉及，有兴趣自己弄弄吧。

5.3 配置示例

5.3.1 配置命令

路由重发布是有方向的，将 A 路由选择域的路由信息注入到 B 路由选择域中，我们要在 B 路由协议的进程中进行配置，例如，要将其他路由协议重发布到 RIP，那么配置如下（重发布到其他路由协议大同小异）：

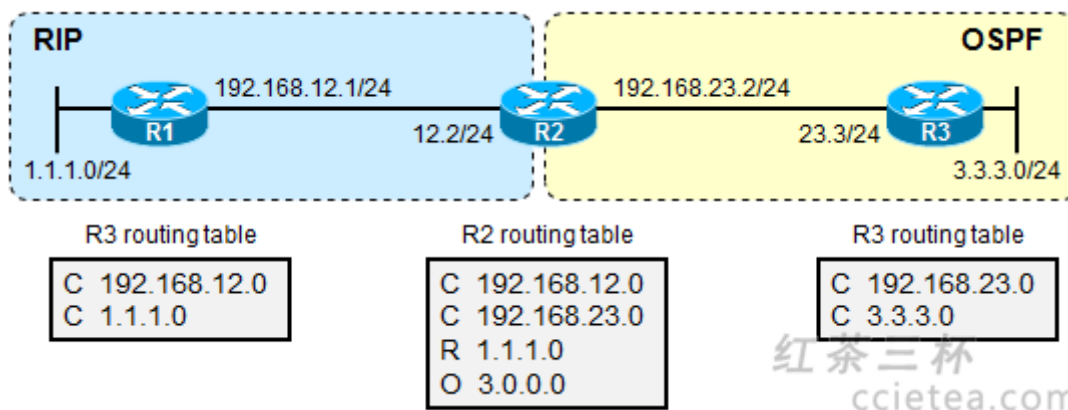
```
Router(config)#router rip
```

```
Router(config-router)#redistribute ?
```

| | |
|-----------|--|
| bgp | Border Gateway Protocol (BGP) |
| connected | Connected |
| eigrp | Enhanced Interior Gateway Routing Protocol (EIGRP) |
| isis | ISO IS-IS |
| iso-igrp | IGRP for OSI networks |
| metric | Metric for redistributed routes |
| mobile | Mobile routes |
| odr | On Demand stub Routes |
| ospf | Open Shortest Path First (OSPF) |
| rip | Routing Information Protocol (RIP) |
| route-map | Route map reference |
| static | Static routes |

5.3.2 配置示例

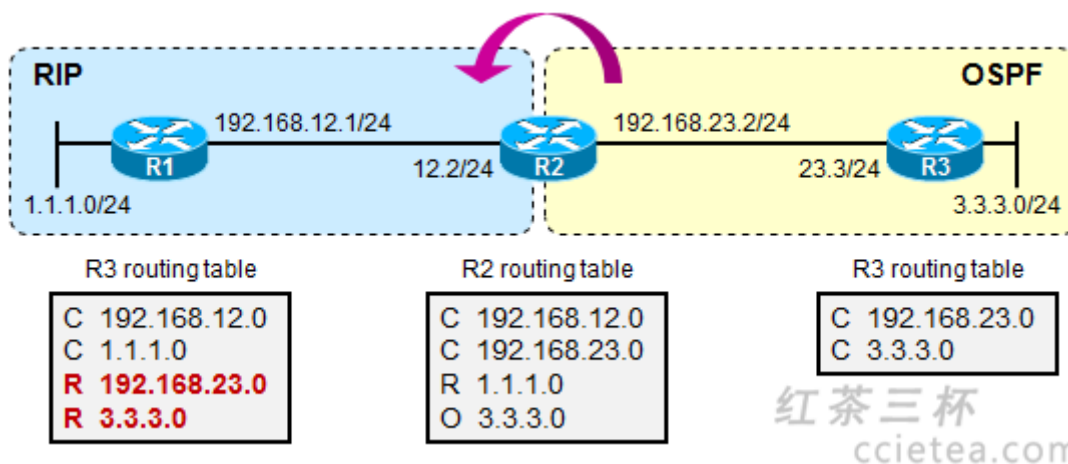
1. OSPF 与 RIP



R1 与 R2 运行 RIP；R2 与 R3 建立 OSPF 邻居关系。初始化情况下，R2 的路由表中有四个条目，如上图所示，而 R1 的路由表中，只有 2 个条目，也就是两个直连链路。那么现在，我们在 R2 上做重发布动作，将 OSPF 路由重发布到 RIP，那么配置如下：

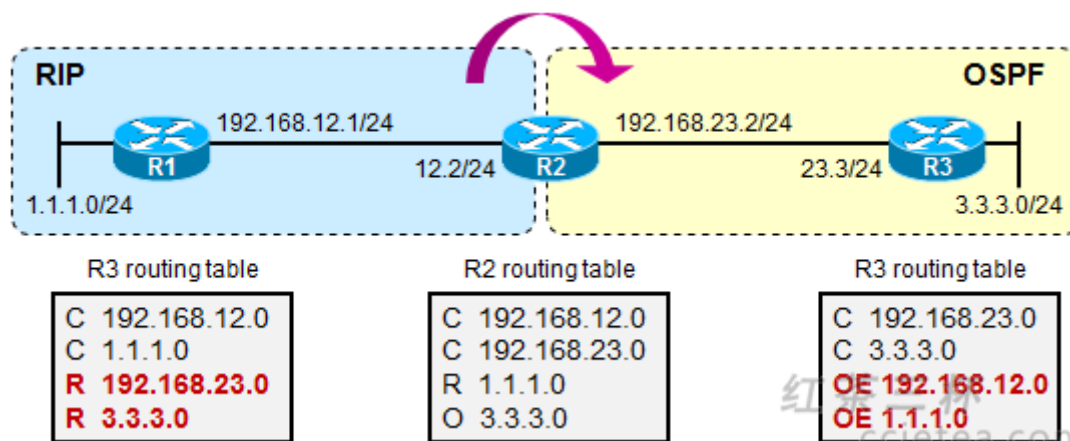
```
router rip
 redistribute ospf 1 metric 3
```

如此一来，R2 上，路由表中的 OSPF 路由 1.1.1.0，以及宣告进 OSPF 进程的 192.168.12.0 直连网段，都被宣告进了 RIP，而 R1 通过 RIP 就能够学习到这两条路由，如下图示红色粗体部分。



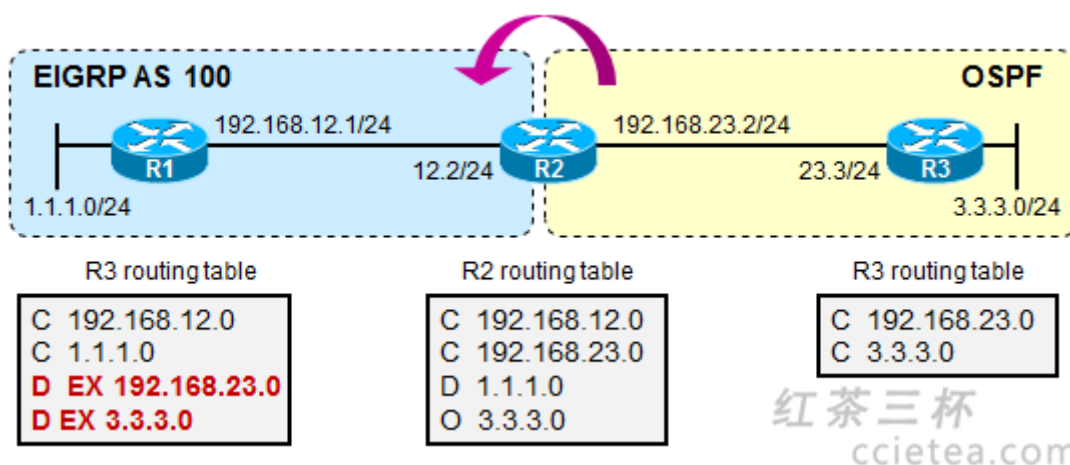
当然，这个时候 1.1.1.0 是无法访问 3.3.3.0 的，因为 R3 并没有 RIP 路由选择域中的路由（也就是说回程路由有问题，数据通信永远要考虑来回路径，记住了），所以如果要想实现全网互通，那么需在 R2 上：

```
R2(config)#router ospf 1
R2(config-router)#redistribute rip subnets
```

如此一来，就实现了全网互通。注意，当重发布路由到 OSPF 时，redistribute rip subnets，这个 **subnets** 关键字要加上，否则只会重发布主类路由。

2. OSPF 与 EIGRP 重发布



初始情况同上，我们先看看将 OSPF 路由重发布进 EIGRP AS 100，配置当然还是在 R2 上进行，进入 R2 的 EIGRP 路由进程：

```
R2(config)# router eigrp 100
```

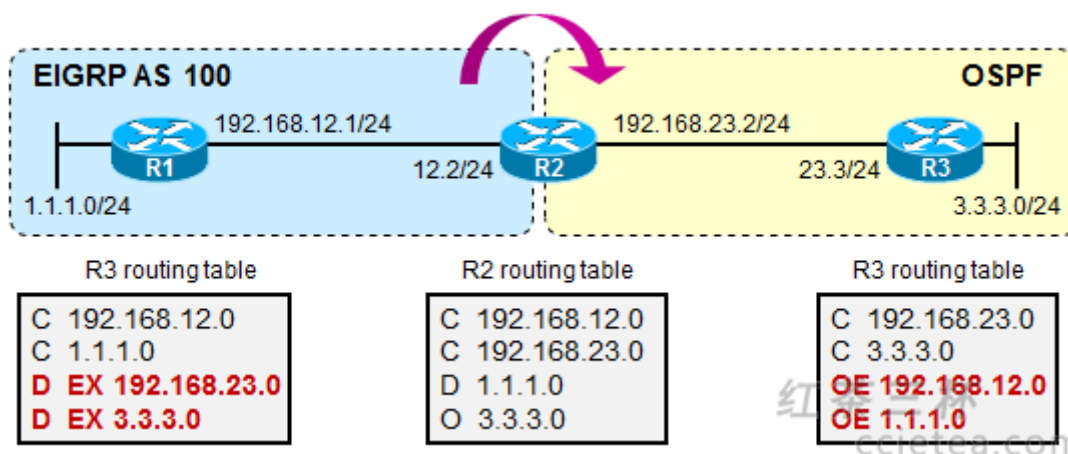
```
R2(config-router)# redistribute os 1 metric 100000 100 255 1 1500
```

注意，EIGRP 的 metric 是混合型的，**metric 100000 100 255 1 1500** 这里指定的参数，从左至右依次是带宽、延迟、负载、可靠性、MTU。可根据实际需要灵活的进行设定。上述配置完成后，R2 就会将路由表中 OSPF 的路由：包括 3.3.3.0，以及宣告进 OSPF 的直连网段 192.168.23.0/24 注入到 EIGRP 进程。这样 R1 就能够学习到这两条外部路由。

接下来是 EIGRP 到 OSPF 的重发布：

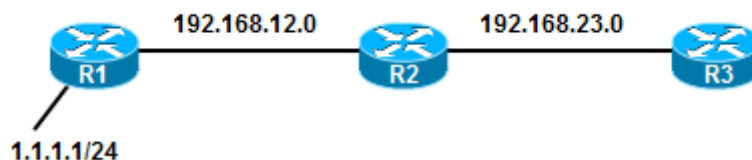
```
R2(config)# router ospf 1
```

```
R2(config-router)# redistribute eigrp 100 subnets
```

5.3.3 重发布的常见问题

1. 关联出接口的静态路由 在被 network 时的问题



R1、R2、R3 跑 **RIP**，R1 上 1.1.1.0/24 没有直接宣告，在 R2 上：

```
ip route 1.1.1.0 255.255.255.0 serial 0/0
```

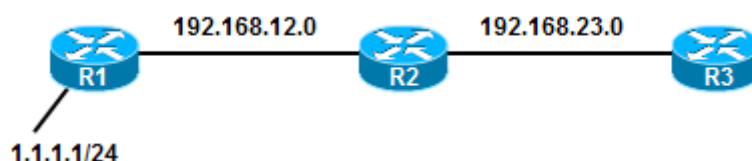
同时 network 1.0.0.0，这时 1.0.0.0 会被宣告出去

使用关联出接口的方式配置的静态路由，路由器会将目的网段视为**本地直连**，因此 RIP 在 network 的时候，会宣告出去。

如果以上换成 EIGRP，则现象与 RIP 一样，R1 会将 1.1.1.0 宣告进 EIGRP

如果以上换成 OSPF，则无效，即关联出接口的静态路由，在 OSPF 中 network 该路由的网络号时，并不会被宣告进 OSPF。

2. 关联出接口的静态路由 在重发布时的问题

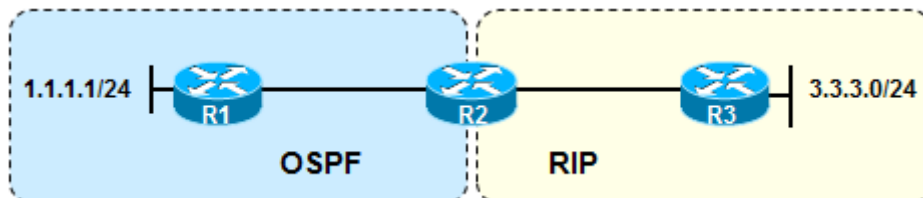


R1、R2、R3 跑 RIP，R1 上 1.1.1.0/24 没有直接宣告，在 R2 上：

ip route 1.1.1.0 255.255.255.0 serial 0/0

此时在 R2 上重发布直连接口，则发现 1.1.1.0 并没有被重发布进 RIP

3. 重发布只会将路由表中有的路由执行重发布动作



在 R2 上进行双向重发布，正常情况下 R1 能够学习到 3.3.3.0、R3 能学到 1.1.1.0

如果在 R2 上 ip route 1.1.1.0 255.255.255.0 null0，这个时候路由表中没有了 OSPF 的 1.1.1.0 路由了因此重发布不成功，所以 R3 无法学习到 1.1.1.0/24 的路由；

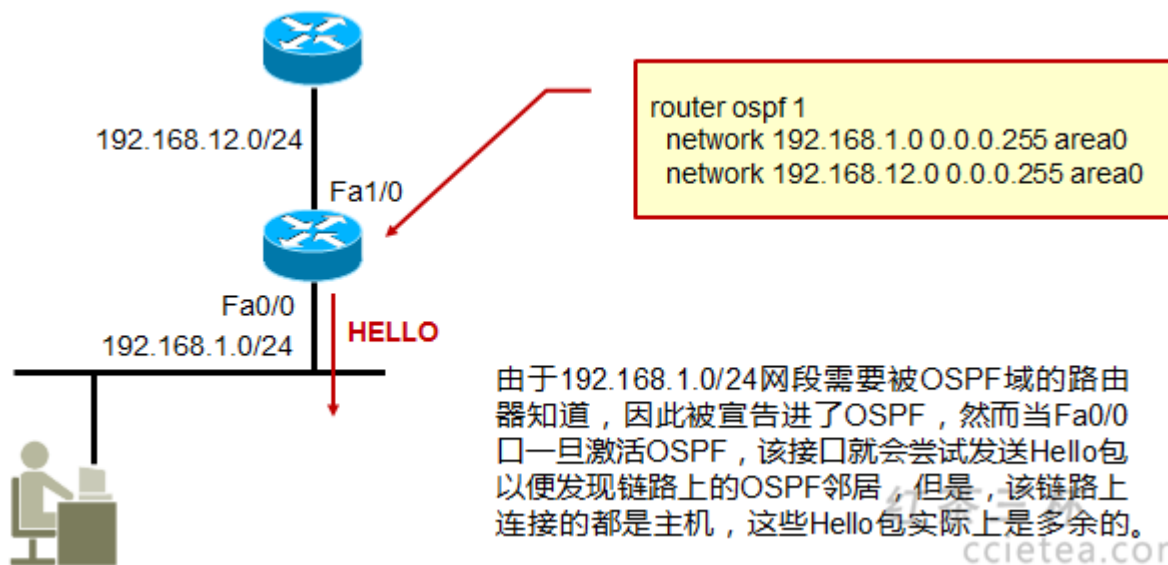
同样，如果 ip route 3.3.3.0 255.255.255.0 null0 也是一样的道理。

重发布是看路由表的，也就是说，例如我将 OSPF 重发布到 EIGRP，那么首先路由必须得在我路由表里有，而且必须是 OSPF 的路由，才能够被注入到 EIGRP。另外，这里有个小问题，R1、R2 之间的链路，虽然在 R2 的路由表中没有看到关于它的 OSPF 路由，但是却成功地被重发布进 RIP 且被 R3 学习到了，这是因为这个**直连链路(接口)已经被 R2 的 OSPF 进程 network 了**，也即通过 OSPF 学习到了，且是直连链路，因此能被重发布。

6 路由策略

6.1 Passive-interface

6.1.1 特性概述



针对上面的问题，我们可以将路由器上的 Fa0/0 口设置为 passive-interface，如此一来，该接口将不再发送 OSPF Hello 消息，同时，该接口关联的网段 192.168.1.0/24 仍然会被宣告进 OSPF，这样就能够避免不必要的 OSPF 组播包在 LAN 中泛洪。这就是 passive-interface 特性的一个最常见的应用，需要注意的是，不同的路由协议对 passive-interface 的操作有所不同。

6.1.2 相关要点

- RIP 和 IGRP 的 passive-interface 不发送路由更新，但是接受路由更新
- EIGRP 的 passive-interface 既不发送也不接收路由更新，并且也不发 HELLO 包
- OSPF passive-interface 既不发送也不接收路由更新，并且也不发 HELLO 包

【注意】接口如果被 passive 掉，但是同时在路由进程里 network 这个接口，则该接口虽不会尝试去发送更新或建立邻居关系，但是其所在网段，仍会被宣告进路由选择进程。

6.1.3 Passive-interface 的配置

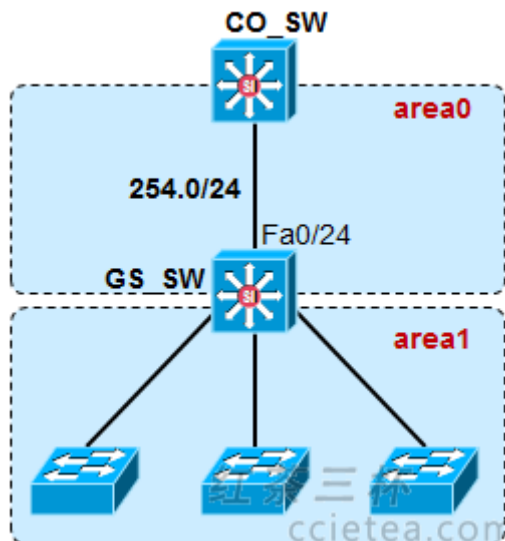
- 将某个接口配置为被动接口：

```
Router(config-router)# passive-interface int-type int-num
```

- 将所有接口配置为被动接口，并手动激活特定接口：

```
Router(config-router)# passive-interface default
Router(config-router)# no passive-interface int-type int-num
```

- 典型配置示例：



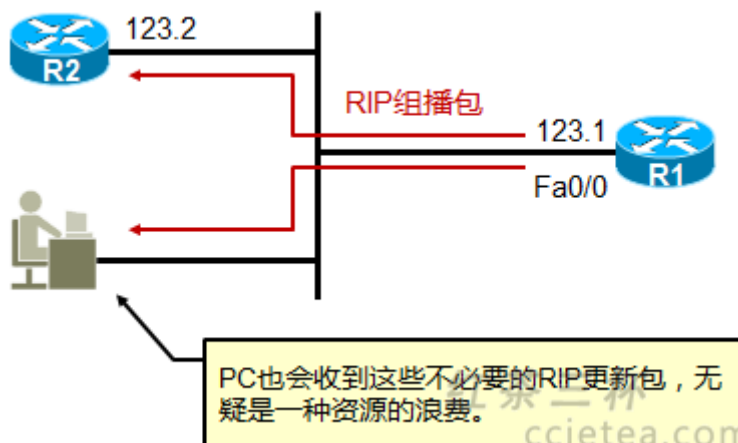
这是一个非常典型的场景，GS_SW 是汇聚层交换机，与核心交换机之间跑个 OSPF。GS_SW 需要通告下挂的 VLAN 所对应的网段，以便核心交换机能够获知相关的路由。然后一旦 GS_SW 在 OSPF 进程中宣告这些 VLAN 对应的网段，相应的 SVI 接口就会向 VLAN 中去泛洪 OSPF HELLO 消息，而这些消息，实际上是多余的。因此我们可以做如下配置：

```
interface vlan 10
  ip address 192.168.10.254 255.255.255.0
interface vlan 20
  ip address 192.168.20.254 255.255.255.0
router ospf 1
  network 192.168.10.0 0.0.0.255 area 1
  network 192.168.20.0 0.0.0.255 area 1
  network 192.168.254.0 0.0.0.255 area 1
passive-interface default
no passive-interface fast 0/24
```

由于实际部署的时候，SVI 口可能比较多，如果一个个的去 no passive-interface 配置量可能会比较大，因此可以选择先 passive-interface default 将所有接口全部 passive 掉，然后再单个接口去 no passive-interface。

6.2 单播更新

1. 配置 RIP 单播更新：



上图中，R1 及 R2 两台路由器，互相之间通过 RIP 交互路由信息。但是，由于 RIPv2 基于组播发送路由更新及相关报文，一旦两台路由器在接口上激活 RIP，PC 将不得不耗费资源处理这些它并不需要的组播消息。因此这里就可以使用到单播更新这个特性，R1 配置如下：

```
Router(config) router rip
Router(config-router)# passive-interface fast 0/0
Router(config-router)# neighbor 192.168.123.2
```

R2 的配置类似。配置完成后，R1、R2 之间交互 RIP 消息，就采用单播进行，这样 PC 就不会收到影响。

2. 配置 EIGRP 单播更新：

注意：如果是 EIGRP 环境，需实现单播更新，那么路由更新接口不能被 PASSIVE（这与 RIP 不一样），直接使用 neighbor 命令去指定邻居即可。如果接口一旦被 PASSIVE，则即使手工指定了 neighbor，也是无法正常建立 EIGRP 邻居关系的。

3. 配置 OSPF 单播更新：

OSPF 的 neighbor 命令使用上又与 EIGRP 不太一样，经测试，在以太网环境下，直接互指 neighbor，仍然会发组播 hello

6.3 调整路由协议的管理距离

1. 配置如下：

- 修改 OSPF 的 AD 值

```
Router(config)# router ospf 1
```

```
Router(config-router)# distance AD ip-src wildcard acls
```

或者

```
Router(config-router)# distance ospf external ad1 inter-area ad2 intra-area ad3
```

上述两条命令，都能起到调整路由协议管理距离的作用，第一条命令 `distance ad ip-src`，可以针对特定的路由更新源及特定的路由前缀调整管理距离，例如，我将某个 OSPF 邻居发给我的某些路由，AD 值调整为 130。

第二条命令，是针对外部路由、区域间或者区域内的路由进行 AD 值的调整。

- 修改 EIGRP 的 AD 值

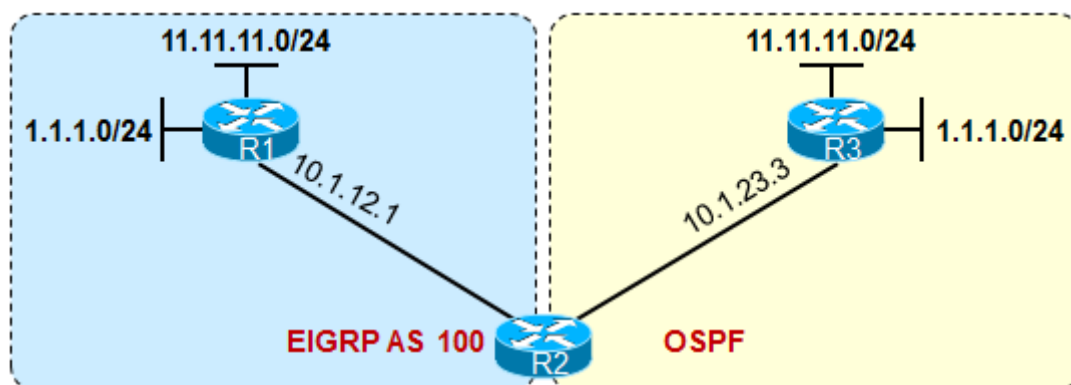
```
Router(config)# router eigrp 100
```

```
Router(config-router)# distance AD ip-src wildcard acls
```

或者

```
Router(config-router)# distance eigrp internal-distance external-distance
```

2. 配置示例



实现需求（R2的路由表）：

```
O 11.11.11.0
D 1.1.1.0
```

实现方式（R2的配置）：

```
access-list 1 permit 11.11.11.0
!
router eigrp 100
distance 130 10.1.12.1 0.0.0.0 1
```

上图中，R1、R3 都会更新路由 1.1.1.0 及 11.11.11.0 给 R2，当然，默认情况下，R2 肯定是优选 EIGRP

的路由。因此去往这两个网段，都走 R1。那么如果我们希望，去往 11.11.11.0 走 R3，从而使得这两个目的地能够起到分流的效果呢？我们就可以选择在 R2 的 EIGRP 进程中，将来源于 10.1.12.1 这个更新源的路由 11.11.11.0/24 的 AD 值调整为比 OSPF 大，例如 130。因此配置如下：

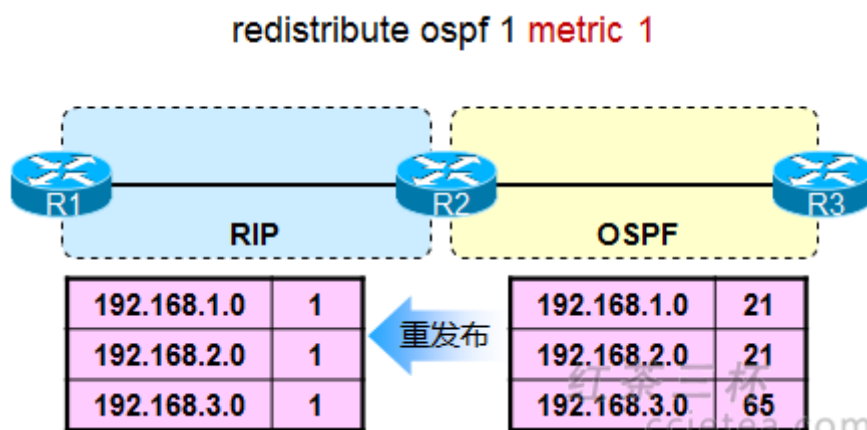
```
access-list 1 permit 11.11.11.0
router eigrp 100
  distance 130 10.1.12.1 0.0.0.0 1
```

更详细的关于各种路由协议调整 AD 值的内容，请见红茶三杯的 OSPF、EIGRP、BGP 等技术文档。

6.4 Route-map

6.4.1 Route-map 概述

1. 技术背景



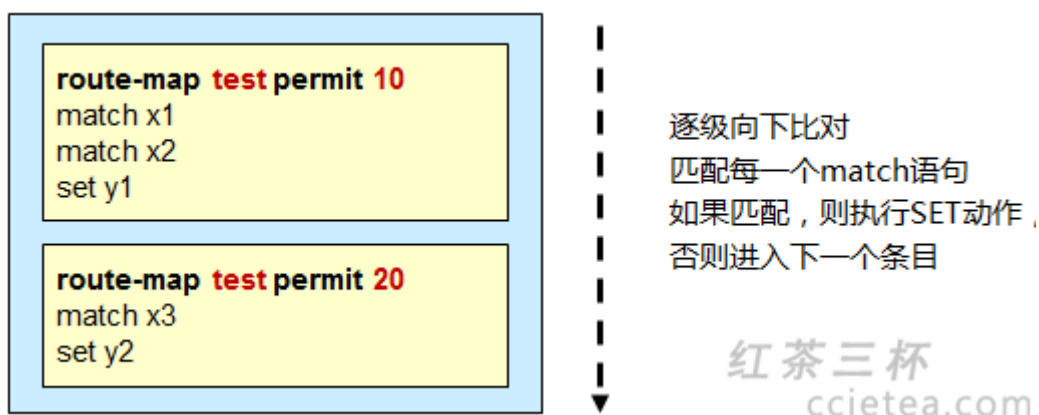
首先来初步认识一下 route-map。看上图，我们在 R2 上，将 OSPF 路由重发布进 RIP，前面已经说过了，在重发布时，可以使用 metric 关键字来设置路由被重发布进 RIP 后的 metric，这里设置为 1，那么直接的结果是，所有被注入到 RIP 的 OSPF 路由，metric 都是 1。那么如果我希望做些灵活性的调整呢？例如我希望在路由被注入 RIP 后，192.168.1.0 路由的 metric 为 1，2.0 的 metric 为 2 如此这般呢？传统的重发布是没办法做到的。

那么就可以使用 route-map 这个工具，也就是说，我们可以在执行重发布的时候，关联一个 route-map，来实现我刚才所说的这个功能。

2. Route-map 的使用场景

- 重分发期间进行路由过滤或执行策略
- PBR (策略路由)
- NAT (网络地址转换)
- BGP 中的策略部署
- 其他用途

3. Route-map 初相识



首先明确一下，route-map 是一个非常重要的工具，使用的范围非常广泛。在定义 route-map 的时候，我们采用 route-map 关键字，关联一个自定义的参数，例如 test 来创建。一个 route-map 列表，由这个 test 字符串统一表示，你可以在一个 route-map 下定义多个序列，用十进制的序列号来表示，例如上图中的，10、20。

那么在每一个序列中，我们就可以来定义供策略部署的两个元素：**匹配条件**、**执行动作**。你可以定义多个条件，当条件被匹配时，就会去执行 set 指定的相关动作。这里稍微提一下，set 语句并不是必须，例如如果该 route-map 仅仅为了匹配感兴趣流量，那么可能就只有 match 语句而没有 set 语句。

在 route-map 被调用后，匹配动作将会从最小的序列号开始执行，如果该序列号中的条件都被匹配了则执行 set 命令，如果条件不匹配，则切换到下一个序列号继续进行匹配动作。

4. Route-map 的特点

- 使用 match 命令匹配特定的分组或路由，set 修改该分组或路由相关属性。
- Route-map 中的每个序列号语句相当于于访问控制列表中的各行。按照序列号的顺序自上而下的处理，一旦找到匹配的序列则不会继续往下继续查找。
- Route-map 默认为 permit，默认序列号为 10，序列号不会自动递增，需要指定序列号
- 末尾隐含 deny any

- 单条 match 语句包括多个条件时，使用逻辑 or 运算；多条 match 语句时，使用逻辑 and 运算。

6.4.2 配置命令

1. 创建 route-map

route-map

- 这个全局配置命令创建一个 route-map，使用自定义的字符串来表示这个 route-map，你可以在一个 route-map 下定义多个序列号。序列号在进行匹配动作时具有优先顺序。
- Permit/deny 关键字在不同的部署场合中作用有所不同

route-map test permit/deny 10

```
match x1
match x2 , x3
set Y
```

route-map test permit/deny 20

```
match x4
set Y
```

2. 定义匹配条件

```
match ip address 匹配访问列表或前缀列表
match length 根据分组的第三层长度进行匹配
match interface 匹配下一跳出接口为指定接口之一的路由
match ip next-hop 匹配下一跳地址为特定访问列表中被允许的那些路由
match metric 匹配具有指定度量值的路由
match route-type 匹配指定类型的路由
match community 匹配 BGP 共同体
match tag 根据路由的标记进行匹配
```

3. 定义 set 动作

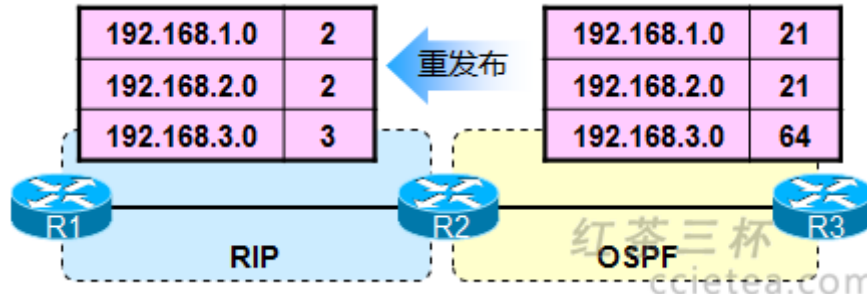
```
set metric 设置路由协议的度量值
set metric-type 设置目标路由协议的度量值类型
set default interface 指定如何发送这样的分组
```

```

set interface 指定如何发送这样的分组
set ip default next-hop 指定转发的下一跳
set ip next-hop 指定转发的下一跳
set next-hop 指定下一跳的地址，指定 BGP 的下一跳
set as-path
set community
set local-preference
set weight
set origin
set tag
default 关键字优先级低于明细路由
    
```

6.4.3 配置示例

1. 路由重发布时关联 route-map



在上图中，我们将 OSPF 路由注入到 RIP，传统的做法，你只能够对所有注入进来的路由统一设置 metric，但是有了 route-map，我们可以在配置重发布命令时，关联一个已经定义好的 route-map，在 route-map 中，我们可以通过创建多个序列号语句，进而对不同的路由，设置不同的属性或动作。

例如这个例子，我们希望注入进来后，192.168.1.0 和 192.168.2.0 这两条路由的 metric 变为 2 跳，3.0 变为 3 跳。

```

access-list 1 permit 192.168.1.0
access-list 1 permit 192.168.2.0
access-list 2 permit 192.168.3.0
    
```

!

!! 上面创建了两个 ACL，分别匹配需要差分对待的路由

route-map test permit 10

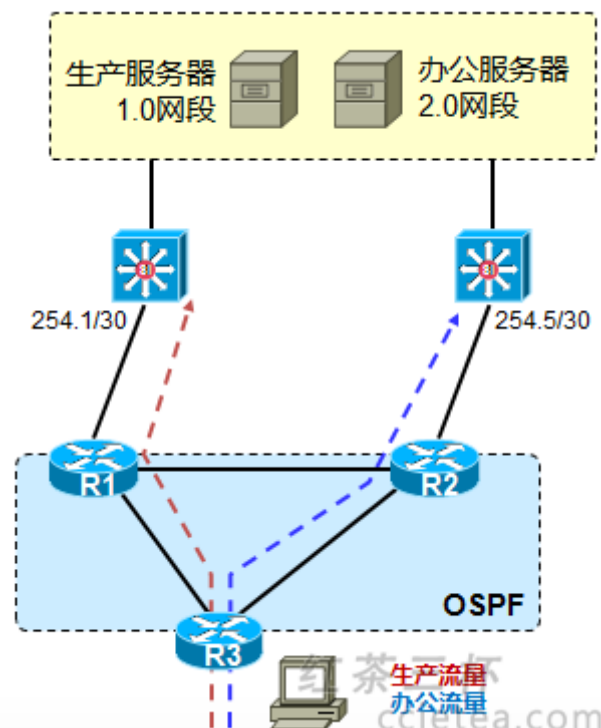
match ip address 1

!! 当路由匹配 ACL1 时

```

set metric 2                                !! 将 metric 修改为 2
route-map test permit 20
match ip address 2
set metric 3
router rip
redistribute ospf 1 route-map test
    
```

2. 路由重发布时关联 route-map （典型案例）



这是一个非常典型的案例，上图中，网络环境是这样的，假设我们有 R1、R2 两台路由器，连接到了服务器群，服务器群使用两台三层交换机下挂着网络的服务器，服务器中我们规划了两个子网分别是生产的 10.1.1.0/24，以及办公 10.1.2.0/24。R3 是接入路由器。R1、R2、R3 跑 OSPF。

R1、R2 与三层交换机之间，假设是静态路由环境。那么现在，我们希望 R3 下的用户，在访问生产服务器到时候，流量往红色虚线箭头所指示的方向流动，访问办公服务器的时候往蓝色箭头方向流动。

那么首先 R1 及 R2 上，为了让他们自己能够到达服务器 10.1.1.0 及 2.0 网段，需要配置两条静态路由：

```

ip route 10.1.1.0 255.255.255.0 10.1.254.1
ip route 10.1.2.0 255.255.255.0 10.1.254.1
    
```

接着为了让 R3 能够动态学习到生产及办公服务器的路由，现在需要将这两条静态路由重发布进 OSPF，当然，在重发布的时候就有技巧了。

R1 的配置如下：

```

access-list 1 permit 10.1.1.0
    
```

```
access-list 2 permit 10.1.2.0
route-map cisco permit 10
    match ip address 1
    set metric 10
route-map cisco permit 20
    match ip address 2
    set metric 20
router ospf 100
    redis static route-map cisco
```

R2 的配置如下：

```
access-list 1 permit 10.1.1.0
access-list 2 permit 10.1.2.0
route-map cisco permit 10
    match ip address 1
    set metric 20
route-map cisco permit 20
    match ip address 2
    set metric 10
router ospf 100
    redis static route-map cisco
```

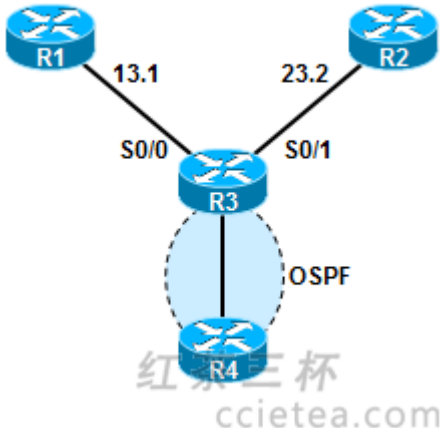
这样就实现了需求。

6.4.4 难点案例

1. 验证 match interface 的作用 1

一个 route-map 语句中，如果没有 match 语句，则匹配所有

Match interface : To distribute any routes that have their next hop out one of the interfaces specified, use the match interface command in route-map configuration mode 中文上的理解是，match interface 匹配的是下一跳出接口是这个接口的路由条目



在 R3 上：

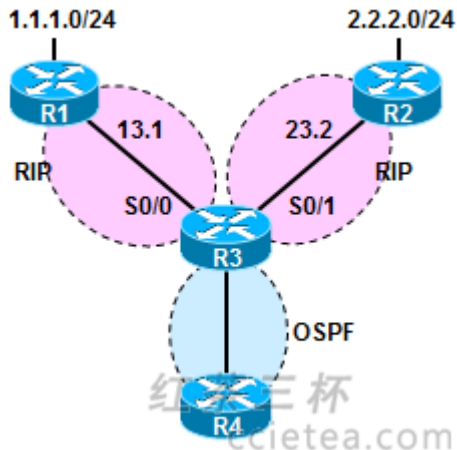
```

ip route 1.1.1.0 255.255.255.0 serial 0/0
ip route 2.2.2.0 255.255.255.0 192.168.13.1
ip route 3.3.3.0 255.255.255.0 serial 0/1
!
Route-map test permit 10
  Match interface s 0/0
Router os 1
  Redis static route-map test
        
```

那么在 R4 上，只能学习到 1.1.1.0。
2.2.2.0 路由虽然下一跳是指向 R1，但是并没有用出接口的方式创建路由条目，从实验现象看没有被 match 住。3.3.3.0 就不用说了，关联的接口是 s0/1。

所以 match interface，是 match 具有出接口属性的路由，并且这条路由的下一跳出接口是 match 的这个端口。

2. 验证 match interface 的作用 2



R1、R2、R3 之间跑 RIP，R1、R2 分别注入各自的 loopback 口
R3、R4 跑 OSPF，在 R3 上能分别学习到来自 R1 和 R2 的 loopback 路由，接下去：

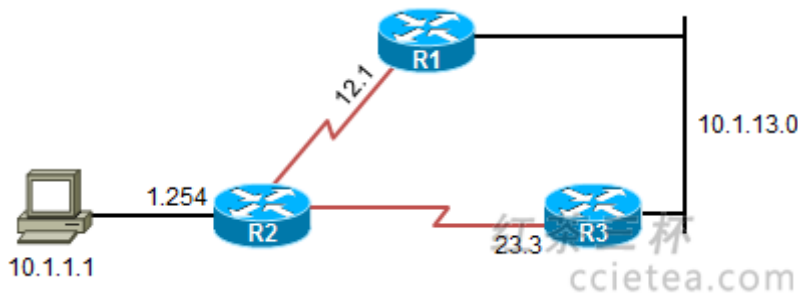
R3 上配置如下：

```

route-map test permit 10
  match interface Serial0/0
router ospf 1
  redistribute rip subnets route-map test
        
```

此时在 R4 上，只能学习到 1.1.1.0/24 以及 192.168.13.0/24
此时 R3 上看到 1.1.1.0 这条路由，是包含下一跳属性的，而前一个例子中，关联下一跳的静态路由是没有这个属性，所以不被匹配

3. 验证 set ip default next-hop



先保证 R1、R3 到 10.1.1.0 是有路由的，在 R2 上做测试：

- 如果 R2 上没有任何的动、静态路由，且配置如下：

```

access-list 1 permit 10.1.1.0 0.0.0.255
route-map test permit 10
  match ip address 1
  set ip default next-hop 10.1.12.1
  
```

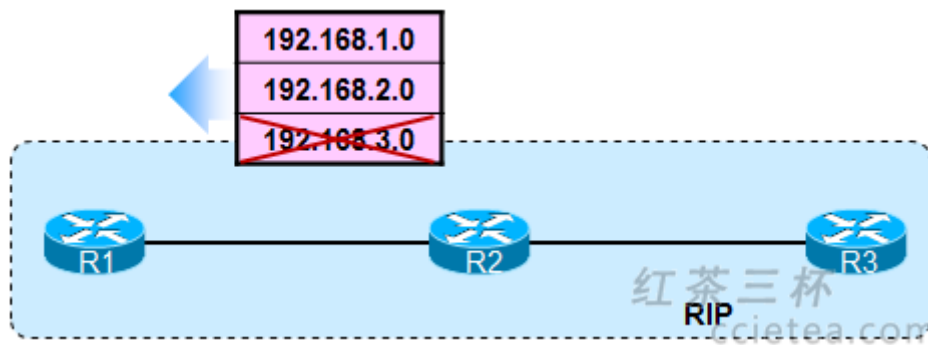
则 PC ping 10.1.13.0，数据走 R1；

- 如果在上述基础上，R2 增加到 R3 的默认路由，则 PC 到 10.1.13.0 网络的数据仍被丢给 R1，也就是说 ip default-next-hop 的优先级高于默认路由。
- No 掉上面配置的默认路由，再配一条去往 13.0 网络的路由，下一跳为 R3，则 PC 到 13.0 网络的数据切换到 R3 证明 ip default next-hop 的优先级低于明细路由，高于默认路由。
- 再次验证，不用明细路由，而是用一条 ip route 10.0.0.0 255.0.0.0 的汇总路由，下一跳为 R3，效果同上，也走 R3。因此只要不是默认路由，只要路由表中存在这么一条匹配的路由，则优先走路由，没有路由的情况下走 route-map。

6.5 distribute-list

6.5.1 工具概述

用于控制路由更新的一个工具，只能过滤路由信息，不能过滤 LSA。



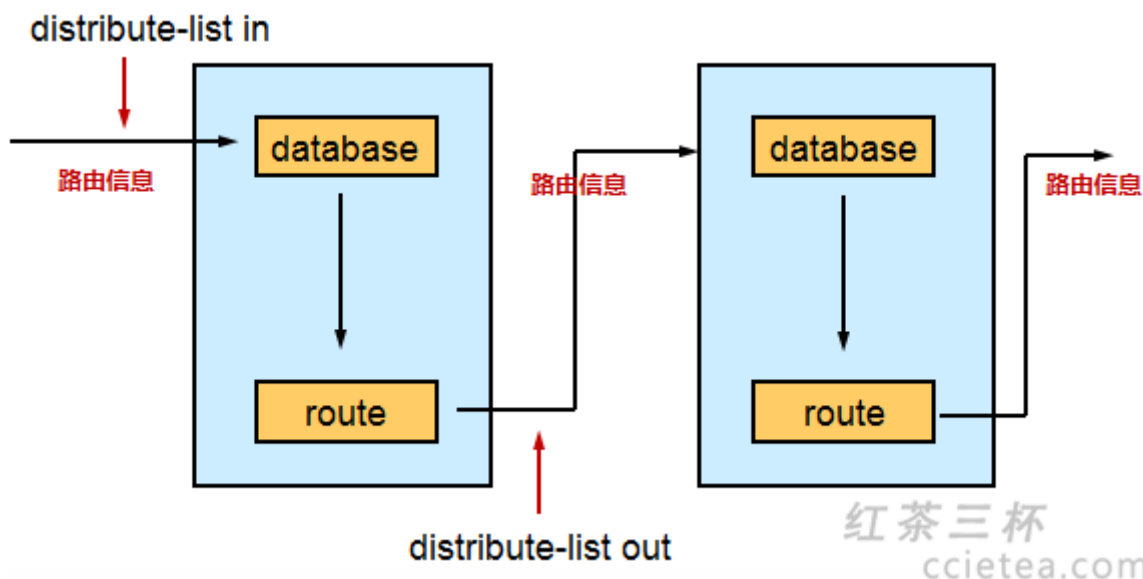
如上图，R1、R2、R3 运行 RIP。R2 在初始情况下，会将自己的路由表更新给 R1，其中假设包含三条路由 1.0、2.0 及 3.0。现在我们可以通过在 R2 上部署分发列表 distribute-list，使得 R2 在更新给 R1 的路由信息中过滤掉 3.0 这条路由。这就是分发列表的一个使用示例。当然，它还有更加广泛的应用。

6.5.2 部署要点

分发列表是**用于控制路由更新的一个工具，只能过滤路由信息，不能过滤 LSA。因此：**分发列表在距离矢量路由协议中使用，无论是 in 或者是 out 方向，都能正常的过滤路由。但是在链路状态路由协议中的工作就有点问题了。

The command distribute-list out works only on the routes being redistributed by the Autonomous System Boundary Routers (ASBRs) into OSPF. It can be applied to external type 2 and external type 1 routes, but not to intra-area and interarea routes.

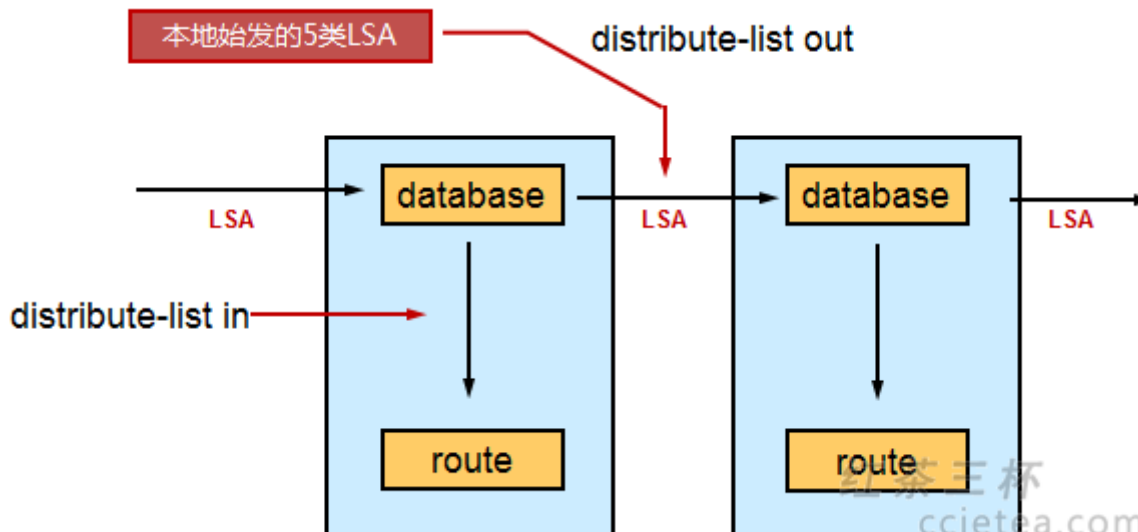
1. 对于距离矢量路由协议



路由器之间，传递的是路由信息，分发列表对路由信息是有绝对的控制权的。因此如果是 in 方向，那么通过部署分发列表，可以过滤特定的路由，使得执行分发列表的本地路由路由表发生变化，同时，本地路由器在更新路由信息给下游路由器的时候，实际上更新的内容是受分发列表影响之后的条目。

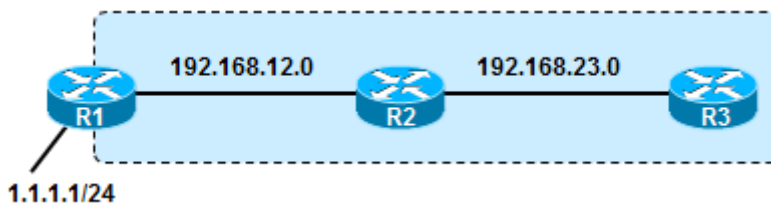
同时在 out 方向，也是没有问题的。

2. 对于链路状态路由协议，如 OSPF



值得注意的是，对于 OSPF 这样的链路状态路由协议，路由器之间传递的消息不再是路由信息了，而是 LSA，而分发列表是无法对 LSA 进行过滤的。因此，在链路状态协议中部署分发列表，就需要留意了：

- in 方向，分发列表只能在本地收到 LSA 后，生成路由的那一刹那进行路由的过滤，执行分发列表的路由器自己路由表会被分发列表影响（但是本地 LSDB 仍然是有 LSA 的），而且该路由器仍会将 LSADB 中的 LSA 发送给邻居，因此本地被过滤的路由，邻居还有（因为邻居已经收到 LSA 了）。
- out 方向，分发列表只能工作在执行路由重发布动作的那个 ASBR 上，且只能针对外部引入的路由起作用。因为 OSPF 执行重发布时，其实这些外部路由是以路由的形式引入进来的，因此分发列表在这个场合下能够正常工作，但是如果不是本地始发的外部路由，或者是内部的 OSPF 路由，out 方向的分发列表均束手无策。



例如在 R1 上重发布直连进 OSPF，用 out 方向的分发列表可过滤掉 1.1.1.0 这条外部路由。但 R1 重发布进来的路由，如果在 R2 上用 out 方向的分发列表试图阻挡 R3 接受路由或 LSA，则无法，因为这不是本地始发的外部路由。

6.5.3 配置命令

1. In 方向

R1(config-router)#distribute-list 1 in ? // 都是接口

| | |
|------------------|--------------------------------|
| Async | Async interface |
| BVI | Bridge-Group Virtual Interface |
| CDMA-lx | CDMA lx interface |
| Dialer | Dialer interface |
| FastEthernet | FastEthernet IEEE 802.3 |
| Multilink | Multilink-group interface |
| Port-channel | Ethernet Channel of interfaces |
| Tunnel | Tunnel interface |
| Vif | PGM Multicast Host interface |
| Virtual-PPP | Virtual PPP interface |
| Virtual-Template | Virtual Template interface |
| | |

2. OUT 方向

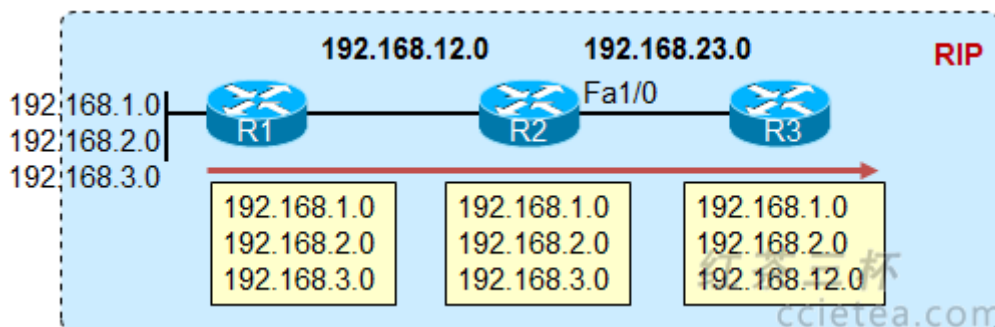
R1(config-router)#distribute-list 1 out ? // 接口或协议

| | |
|-------------------|--------------------------------|
| Async | Async interface |
| BVI | Bridge-Group Virtual Interface |
| Dialer | Dialer interface |
| FastEthernet | FastEthernet IEEE 802.3 |
| Loopback | Loopback interface |
| Multilink | Multilink-group interface |
| Port-channel | Ethernet Channel of interfaces |
| Tunnel | Tunnel interface |
| Virtual-PPP | Virtual PPP interface |
| Virtual-Template | Virtual Template interface |
| Virtual-TokenRing | Virtual TokenRing |

| | |
|-----------|--|
| bgp | Border Gateway Protocol (BGP) |
| connected | Connected |
| eigrp | Enhanced Interior Gateway Routing Protocol (EIGRP) |
| ospf | Open Shortest Path First (OSPF) |
| rip | Routing Information Protocol (RIP) |
| static | Static routes |
| | |
| <cr> | |

6.5.4 应用场合

1. 配置示例 1 (单一路由协议环境下-RIP)

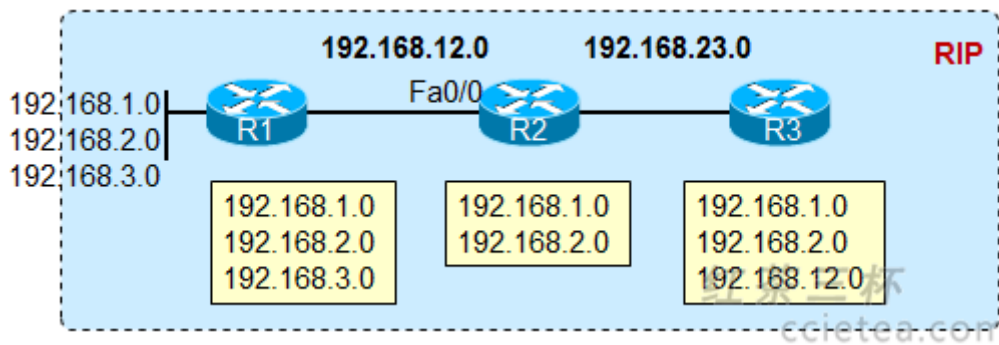


初始情况下，R3 能够学习到 R1 的三条 loopback 路由，以及 192.168.12.0/24 路由。现在不希望 R3 学习到 192.168.3.0/24 的路由，那么可以在 R2 上如下配置：

```
R2(config)# access-list 1 deny 192.168.3.0
R2(config)# access-list 1 permit any
R2(config)# router rip
R2(config-router)# distribute-list 1 out fa 1/0
```

当然，在 R3 上，用 in 方向的分发列表也可以达到同样的效果。

2. 配置示例 2 (单一路由协议环境下-RIP)



在 R2 上如果做如下配置：

```
R2(config)# access-list 1 deny 192.168.3.0
```

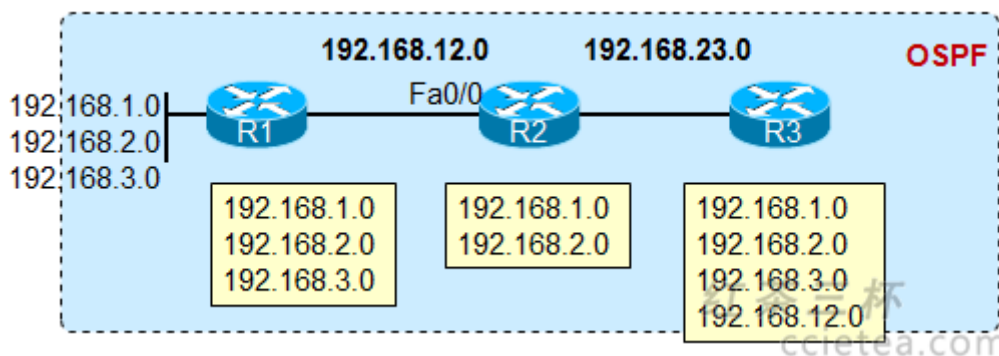
```
R2(config)# access-list 1 permit any
```

```
R2(config)# router rip
```

```
R2(config-router)# distribute-list 1 in fa0/0
```

那么，首先 R2 自己的路由表会发生改变，3.0 的路由被过滤掉了，同时 R3 也就是下游 RIP 路由器，3.0 也学不到。

3. 配置示例 3（单一路由协议环境下-OSPF）



R2 的配置如下：

```
R2(config)# access-list 1 deny 192.168.3.0
```

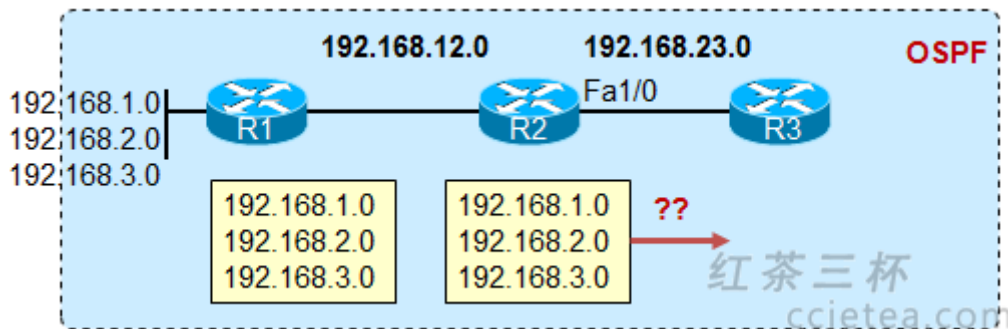
```
R2(config)# access-list 1 permit any
```

```
R2(config)# router ospf 1
```

```
R2(config-router)# distribute-list 1 in fa0/0
```

注意这时候，首先在 R2 的路由表里，3.0 的路由就被干掉了。注意，这时候实际上，area 内 OSPF 路由器产生的 LSA 已经是装载到了 R2 的 OSPF database 之中，而在 R2 从 OSPF database 中计算路由，并准备将路由条目装载进路由表之前，in 方向的分发列表发生作用了，将 3.0 的路由过滤掉了，因此 R2 的路由表中，是没有 3.0 的 OSPF 路由的。但是，虽然 R2 自己路由表里没 3.0 路由，这不妨碍 R2 将相关 LSA 泛洪给 R3，因此，R3 仍然是有 1.0、2.0、3.0 以及 12.0 的 OSPF 路由的。

4. 配置示例 4 (单一路由协议环境下-OSPF)

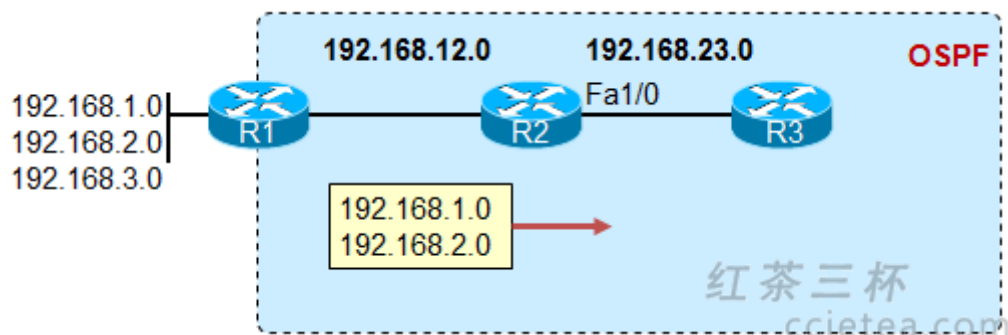


现在我们在 R2 上做如下配置：

```
R2(config)# access-list 1 deny 192.168.3.0
R2(config)# access-list 1 permit any
R2(config)# router ospf 1
R2(config-router)# distribute-list 1 out
```

R3 的路由表会是什么情况？实际上，没有任何影响，R3 能学习到全网的路由。至于为什么，我相信前面已经解释的非常清楚了。

5. 配置示例 5 (单一路由协议环境下-OSPF out 方向分发列表)



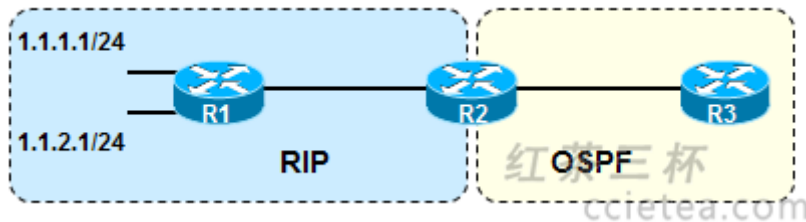
分发列表，部署在 OSPF 这样的链路状态路由协议中，如果要用 out 方向，则只能用在这样的场合。

如上图，在 R1 上部署，R1 使用重发布直连的方式引入这三条外部路由，那么 out 方向的分发列表，只能在 R1 上部署，且对这三条路由产生作用。

```
R1(config)# access-list 1 deny 192.168.3.0
R1(config)# access-list 1 permit any
R1(config)# router ospf 1
R1(config-router)# redistribute connected subnets
R1(config-router)# network 192.168.12.1 0.0.0.0 area 0
R1(config-router)# distribute-list 1 out
```

上述配置实现后，R1 将过滤掉 3.0 路由。

6. 配置示例 6 协议间重发布时部署分发列表



RIP 重发布进 OSPF

• 情况 1

R2 的配置如下：

```
access-list 1 permit 1.1.1.0
router ospf 1
 redistribute rip metric 10 subnets
 distribute-list 1 out rip
```

这里这条命令的意思是，从 RIP 路由协议重分发过来的路由中，只允许 1.1.1.0 出去（到 OSPF 协议，没有方向，只要是运行了 OSPF 的接口）

R3 的路由表里，只有 1.1.1.0 的路由

• 情况 2

在 R2 上开设 loopback 接口 2.2.2.0/24，R2 既重发布 RIP 进 OSPF，又重发布直连进 OSPF

```
access-list 1 permit 1.1.1.0
router ospf 1
 redistribute connected subnets
 redistribute rip metric 10 subnets
 network 192.168.23.0 0.0.0.255 area 0
 distribute-list 1 out
```

// 在 R3 上只有 1.1.1.0 的路由，也就是说 distribute-list 1 out 此处这条命令，对所有从外部注入进 OSPF 的路由都生效，最终只有 1.1.1.0 路由存活下来。而不断路由的来源是直连路由，还是 RIP。

• 情况 3

在 R2 上开设 loopback 接口 2.2.2.0/24，R2 既重发布 RIP 进 OSPF，又重发布直连进 OSPF

```
access-list 1 permit 1.1.1.0
router ospf 1
```

```
redistribute connected subnets
redistribute rip metric 10 subnets
distribute-list 1 out rip
```

// R3 的路由表中有路由：1.1.1.0 、 2.2.2.0 、 192.168.12.0

// 也就是屏蔽掉了从 RIP 重发布进来的除了 1.1.1.0 以外的路由，并重发布本地直连接口

6.6 路由匹配工具

6.6.1 ACL

1. 标准 ACL

标准 ACL 只能匹配路由前缀，无法匹配路由的前缀长度，例如，如果想抓取 192.168.1.0/24 这条路由，用 access-list 1 permit 192.168.1.0，则该条路由被匹配，但是同时，192.168.1.0/25、/26.....也都被匹配了，因为 ACL 无法匹配掩码，或者说，前缀长度。再者，在使用标准 ACL 抓取路由的时候，建议不加反掩码，否则被匹配的路由条目范围将更大更不精确。

2. 扩展 ACL

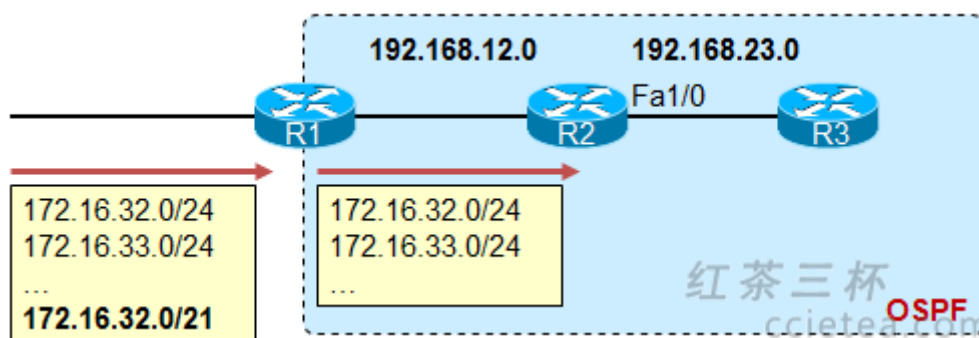
标准 ACL 有源部分、目的部分，使用源匹配路由前缀，使用目的部分匹配路由掩码。

例如，要抓取 192.168.1.0/24 这条路由，则 access-list 100 permit ip **192.168.1.0** **0.0.0.0** **255.255.255.0** **0.0.0.0**。

所以其实很简单，只要把路由的前缀和掩码部分，分别使用 ACL 的源和目的部分进行匹配即可。

6.6.2 Prefix-list

1. 技术背景



我们看上图，外部明细路由 172.16.32.0 – 39.0/24，以及汇总路由 32.0/21 被 R1 引入 OSPF，现在需在 R1 上，仅将汇总路由 32.0/21 过滤，而所有明细路由放行，如果使用标准 ACL 匹配路由，该如何写？

```
R1(config)# access-list 1 deny 172.16.32.0
```

```
R1(config)# access-list 1 permit any
```

仔细思考一下会有什么问题，实际上 access-list 1 deny 172.16.32.0 是一并把汇总路由 172.16. 32.0/21 和明细路由 172.16.32.0/24 给匹配上了，因此最终这两条路由都会被过滤掉，这就与我们的需求不符了。

其实这就是用标准的 ACL 去匹配路由的弊端，你只能匹配路由的网络号，而无法进一步匹配路由的前缀长度，或者说掩码长度。在配上上面这条 ACL 的时候，有些同学甚至会写成 access-list 1 deny 172.16.32.0 0.0.0.255 实际上，这就更有问题了，因为这种写法，实际上最后一个 8 位组不管是什么，都会被匹配住，他就更不精确了。

2. 关于前缀列表

- 可匹配路由前缀中的网络号及前缀长度，增强了匹配的精确度
- 前缀列表的可控性比访问列表高得多，支持增量修改，更为灵活
- 前缀列表包含序列号，从最小的开始匹配
- 如果前缀不与前缀列表中的任何条目匹配，将被拒绝

3. 前缀列表的配置

```
router(config)# ip prefix-list {list-name [seq number] {deny | permit} network/length [ge ge-value] [le le-value]}
```

| 参数 | 描述 |
|-------------|--------------------------------|
| ge ge-value | 要匹配的前缀范围，范围为 ge-value 到 32 |
| le le-value | 要匹配的前缀范围，范围为 length 到 le-value |

输入条件：length < ge-value < le-value <= 32

4. 配置示例

```
ip prefix-list ABC seq 5 permit 172.0.0.0/8
```

路由前 8bits 必须与 172.0.0.0 的前 8bits 完全匹配，其他位不关心，并且掩码必须是/8 的

```
ip prefix-list ABC seq 5 permit 172.0.0.0/8 le 24
```

路由前 8bits 必须与 172.0.0.0 的前 8bits 完全匹配，其他位不关心，并且掩码必须是/8，/9，/10，.....，/24

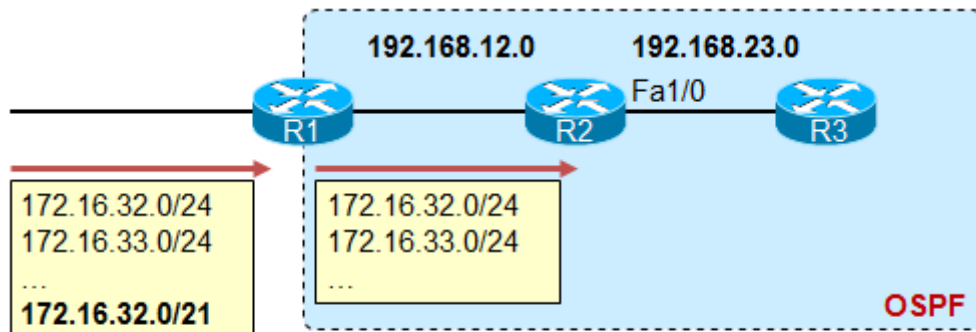
```
ip prefix-list ABC seq 5 permit 172.0.0.0/8 ge 24
```

路由前 8bits 必须与 172.0.0.0 的前 8bits 完全匹配，其他位不关心，并且掩码长度必须大于 24，注意，这里 le 关键字没写，那么默认是 大于 24 小于 32

```
ip prefix-list ABC seq 5 permit 0.0.0.0/0 le 32
```

路由的前 0bits 必须与 0.0.0.0 的前 0 个 bit 匹配，实际上就是所有 bits 都无所谓了，而且对于掩码，以为没有写 ge 关键字，所以隐含的是 ge 0 le 32，也就是掩码长度大于 0 小于等于 32。所以这条前缀列表就是 permit any

5. 应用示例



外部路由 172.16.32.0 – 39.0/24，以及汇总路由 32.0/21 被 R1 引入 OSPF

现在需在注入过程中，仅将汇总路由 32.0/21 过滤，所有明细放行。

```
R1(config)# ip prefix-list list1 deny 172.16.32.0/21
```

```
R1(config)# ip prefix-list list1 permit 0.0.0.0/0 le 32
```

```
R1(config)# route-map test permit 10
```

```
R1(config-route-map)# match ip address prefix-list list1
```


6.6.3 路由标记 Tag

1. OSPF TAG 概述

TAG 字段 (32bits) 只在外部 LSA 中存在。

在 ASBR 重发布时可以使用多种方式修改外部 LSA 的 TAG 值。

- 修改 ASBR 产生的所有外部 LSA 的 TAG

```
r1(config-router)#redistribute rip subnets tag ?
<0-4294967295> 32-bit tag value
```

当然，也可以在重发布命令后关联 route-map，在 route-map 中设置 tag

- 修改 ASBR 通告的汇总 LSA 的 TAG

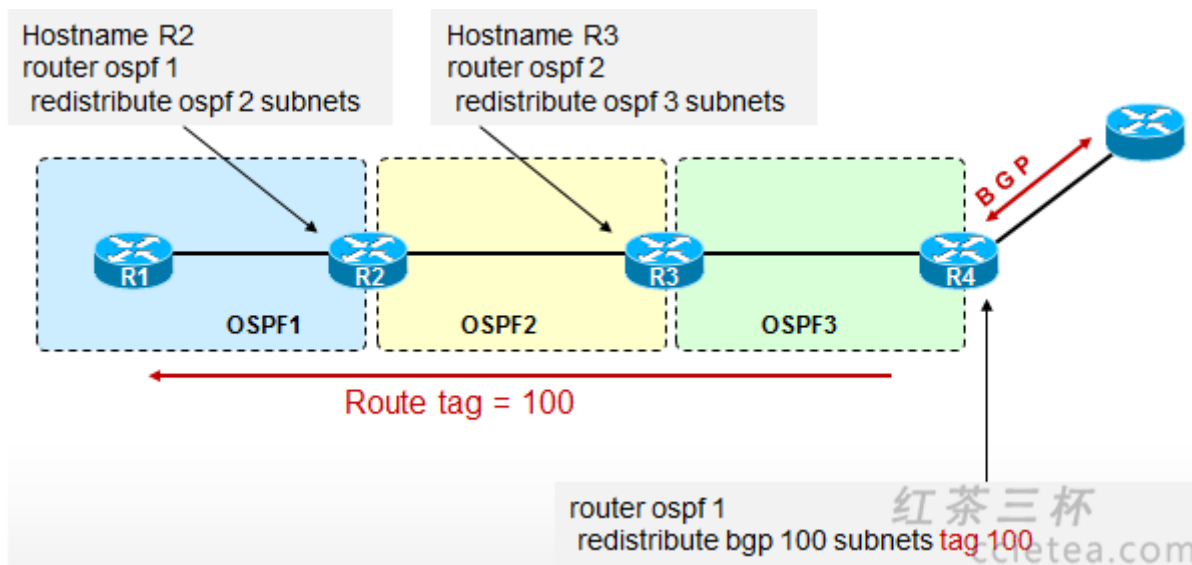
```
r1(config-router)#summary-address 10.0.0.0 255.0.0.0 tag ?
<0-4294967295> 32-bit tag value
```

下面是 5 类 LSA 报文中，TAG 字段的位置：

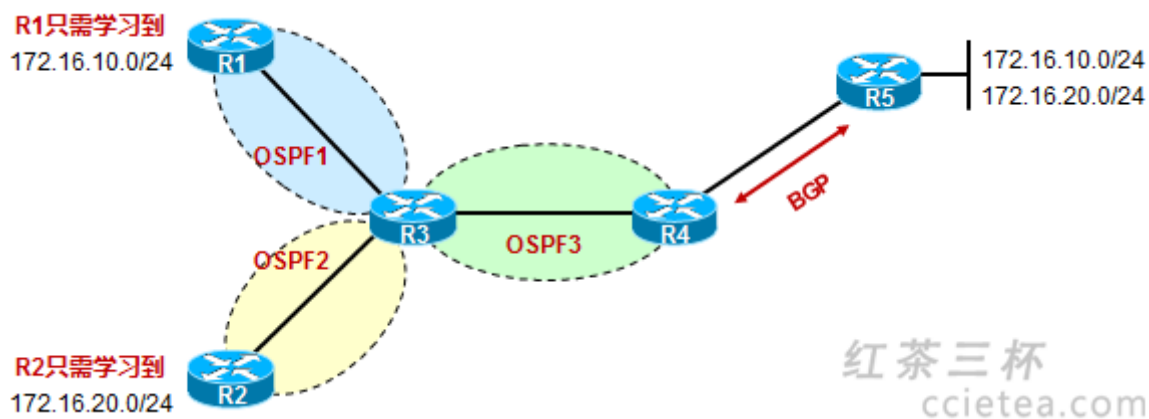
```
⊕ OSPF Header
⊖ LS Update Packet
  Number of LSAs: 1
  ⊖ LS Type: AS-External-LSA (ASBR)
    LS Age: 2 seconds
    Do Not Age: False
    ⊕ Options: 0x20 (DC)
      Link-State Advertisement Type: AS-External-LSA (ASBR) (5)
      Link State ID: 3.3.3.0
      Advertising Router: 3.3.3.3 (3.3.3.3)
      LS Sequence Number: 0x80000001
      LS Checksum: 0x216a
      Length: 36
      Netmask: 255.255.255.0
      External Type: Type 2(metric is larger than any other ltr
      Metric: 20
      Forwarding Address: 0.0.0.0
      External Route Tag: 0
```

2. 外部 LSA 中的 TAG 值的传递范围

在始发的 ASBR 产生外部 LSA 时设置了 TAG 值，如果该外部 LSA 重发布进其他 OSPF 自治系统，TAG 值为默认携带。



3. 实验示例：重发布时调用



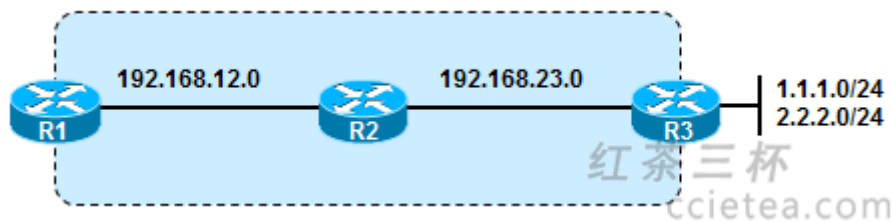
R4 的配置如下：

```
access-list 1 permit 172.16.10.0
access-list 2 permit 172.16.20.0
route-map BGP2OSPF permit 10
  match ip address 1
  set tag 10
route-map BGP2OSPF permit 20
  match ip address 2
  set tag 20
router ospf 3
  redistribute BGP 1 subnets route-map BGP2OSPF1
```

R3 的配置如下：

```
route-map OSPF3to1 permit 10
  match tag 10
route-map OSPF3to2 permit 20
  match tag 20
router ospf 1
  redistribute ospf 3 subnets route-map OSPF3to1
router ospf 2
  redistribute ospf 3 subnets route-map OSPF3to2
```

4. 实验示例：在分发列表中调用



R3 在注入路由的时候，设置上 tag

R3 的配置如下

```
access-list 1 permit 1.1.1.0
access-list 2 permit 2.2.2.0
router ospf 1
  redistribute connected metric 10 subnets route-map test
  network 192.168.23.3 0.0.0.0 area 0
route-map test permit 10
  match ip address 1
  set tag 10
route-map test permit 20
  match ip address 2
  set tag 20
```

R2 的配置如下：

```
router ospf 1
  network 192.168.12.2 0.0.0.0 area 0
```

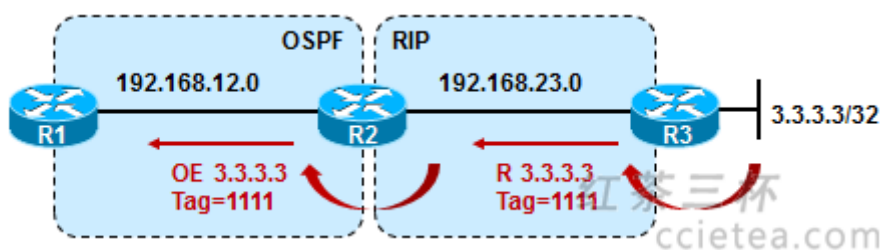
```
network 192.168.23.2 0.0.0.0 area 0
distribute-list route-map test in
route-map test permit 10
match tag 10
```

【结果】R2 上路由表中只有 1.1.1.0 的路由。

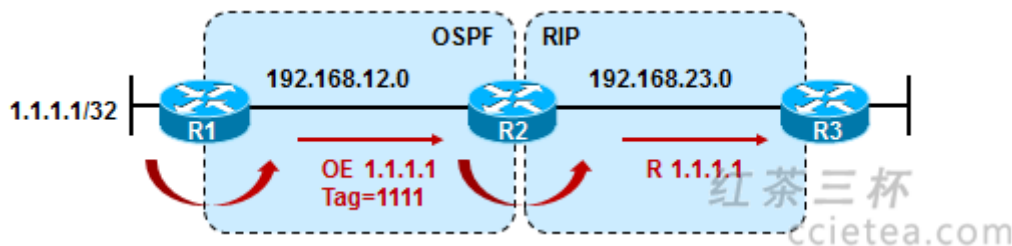
R1 上有 1.1.1.0、2.2.2.0 的路由

5. Tag 值的传递

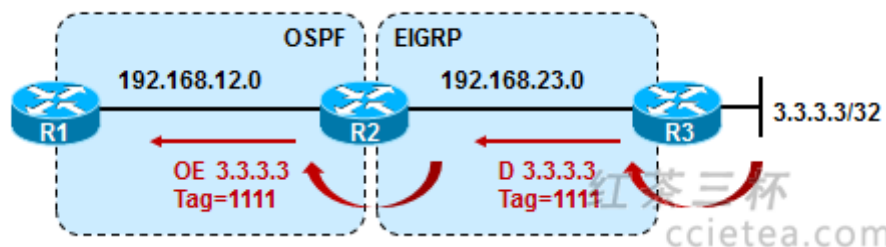
RIP 也是支持 tag 的，但是必须是 version 2。



R3 在重发布直连路由 3.3.3.3/32 的时候关联 route-map，打上 tag1111，这条路由传递到了 R2。在 R2 上，部署 RIP 到 OSPF 的重发布，那么 3.3.3.3 的外部路由注入 OSPF 后，tag 值是默认携带的。然而如果在 R1 上发布一条携带 tag 的外部路由，并且在 R2 上重发布 OSPF 进 RIP，tag 丢失：

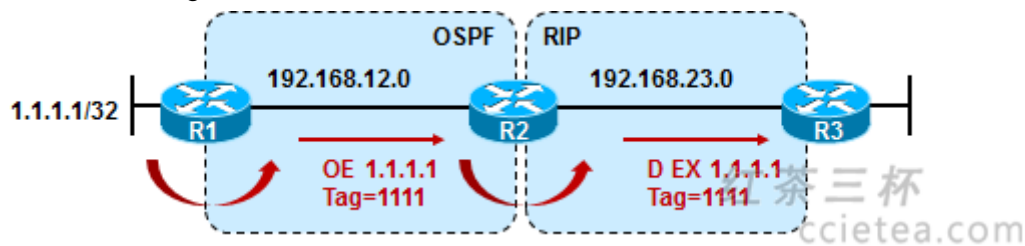


EIGRP 也是支持 tag 的。



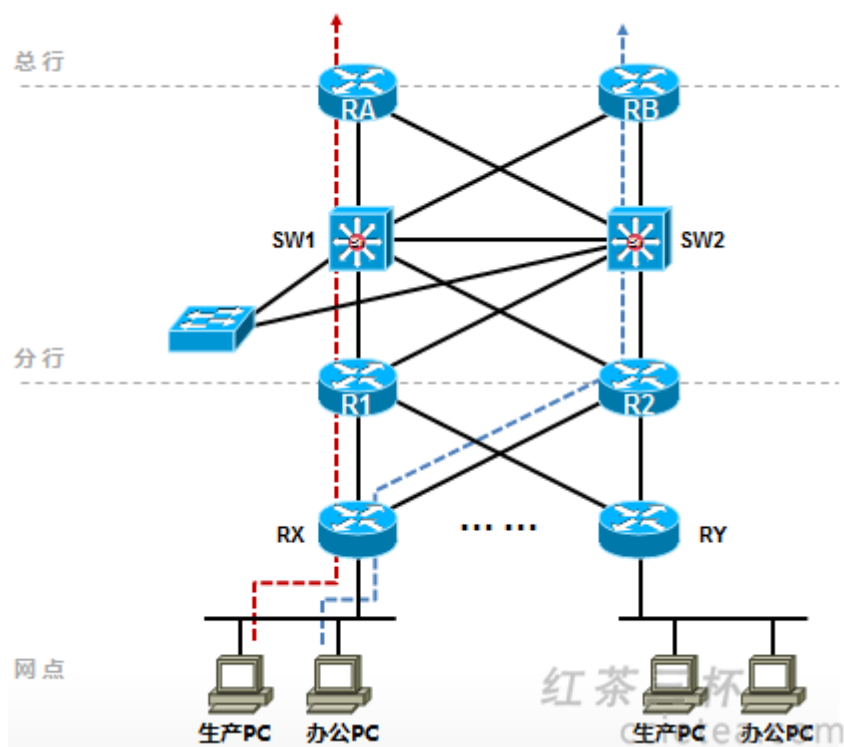
R3 在重发布直连路由 3.3.3.3/32 的时候关联 route-map，打上 tag1111，这条路由传递到了 R2。在 R2 上，部署 RIP 到 OSPF 的重发布，那么 3.3.3.3 的外部路由注入 OSPF 后，tag 值是默认携带的。而如果在 R1 上发布一条携带 tag 的外部路由，并且在 R2 上部署 OSPF 到 EIGRP 的重发布，那么这条路由

在注入到 EIGRP 后，tag 也依然携带。



7 路径控制

7.1 路径控制概述



严格的说，路径控制是一个非常大的课题，在一个大型网络的部署中，往往需要费尽心思考虑对数据流量访问路径的控制，为的是更加合理的、科学的利用和分配网络资源，同时增强网络的可靠性、冗余性和健壮性。

而实现数据访问路径控制的需求，方法和工具往往有非常多，如果一个网络前期的 IP、VLAN 等基本元素规划的非常科学和合理，那么在策略部署这块就更加的轻松和选择多样。本章不会详细讨论各种路径控制的工具，仅仅列举几个常见的工具和方法做个讨论。其实利用前面所学的工具，诸如 route-map 等，通过在路由控制

层面执行多样化的策略，已经能够起到非常不错路径控制的效果，而且，通过控制路由来控制数据流走向，是一个非常科学也非常建议的方法。除此之外，还有许多，如：

- 妥善的编址方案：VLSM和CIDR
- 重分发和路由协议的特征
- passive-interface
- distribute-list
- prefix-list
- AD的把控

- route-map
- 路由标记
- offset-list
- Cisco IOS IP SLAs
- PBR
-

红茶三杯
ccietea.com

再次强调一下，这里只是简单的各种工具的罗列，在实际网络的部署中，需要根据实际的情况，灵活的挑选最经济、最科学、最可靠的工具和方法来部署。

7.2 Offset-list 偏移列表

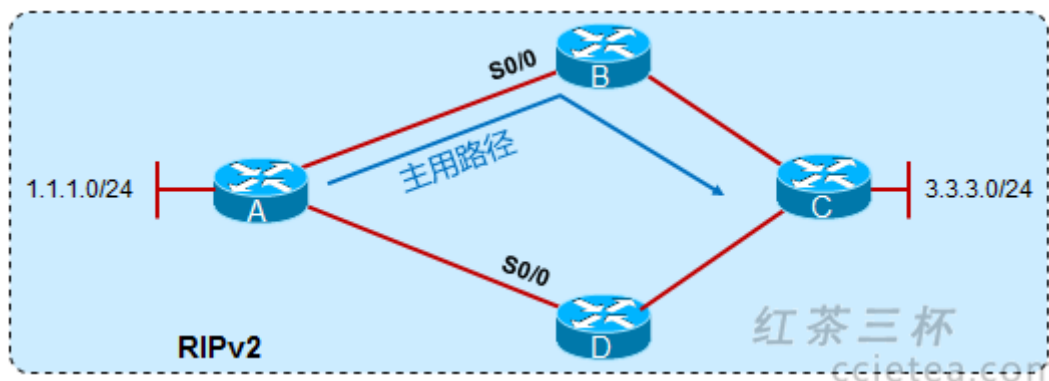
1. 技术概述

用于在入站或出站时增大通过 EIGRP 或 RIP 获悉的路由度量值。

2. 配置命令

```
router(config-router)#  
offset-list {access-list-number | name} {in|out} offset [interface-type interface-number]
```

3. 配置示例 RIP 环境



1.1.1.0 到 3.3.3.0 实际上有两条路径可走,如果网络中运行 RIP 协议,实际上对于 A 而言去往 3.3.3.0/24 是可以通过 B 和 D 负载分担的。但是在某些情况下,我们更希望数据的走向是可控的,例如,我们希望 1.1.1.0 访问 3.3.3.0 的流量主走 B,当 B 挂掉了,则切换到 D 上。那么我们只要简单的在 D 上部署 offset-list,在其向 A 通告 3.3.3.0 路由时,增加 1 跳,那么这条路由就会相比 B 通告给 A 的路由 metric 大 1 跳,A 自然会优选 B。

```
access-list 1 permit 3.3.3.0
```

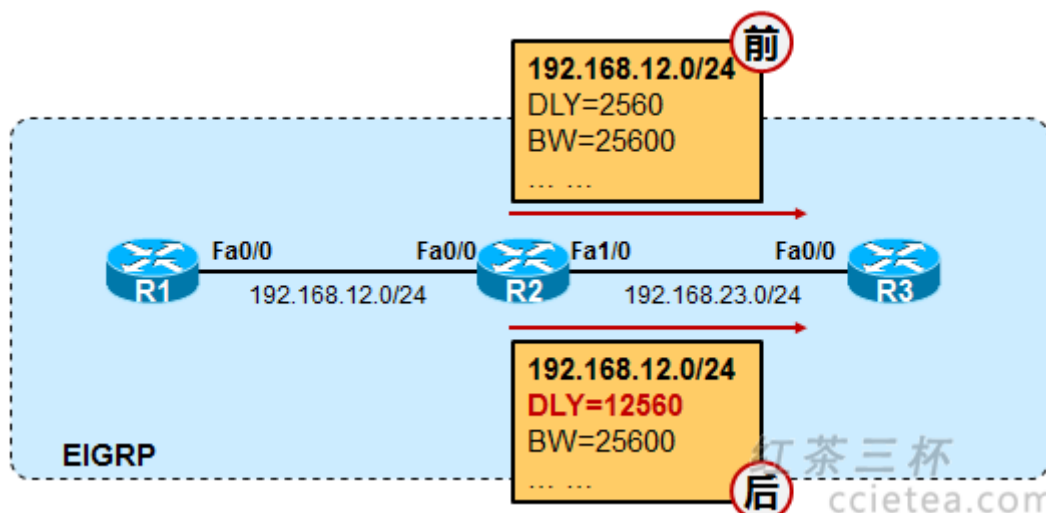
```
router rip
```

```
offset-list 1 out 1 serial 0/0
```

!! 红色字体部分为要增加的跳数

这里补充一句,既然关注了流量,就要注意流量的往返,一般情况下,我们希望 1.1.1.0 访问 3.3.3.0 的流量走 ADC,那么往返流量一般都是需要路径一致的。也就是说 3.3.3.0 到 1.1.1.0 的数据走向是 CDA,所以要进一步在 D 上,将更新给 C 的 1.1.1.0 的路由的 metric 加 1 跳。

4. 配置示例 EIGRP 环境



初始情况下 R2 更新 192.168.12.0/24 路由给 R3 的时候,我们看一下,update 报文中,该路由前缀携带的各项属性 DLY=2560,因为 R2 Fa0/0 接口的 DLY 是 100 微妙,而报文中的单位是 10us,因此,10*256=2560,这个值就是报文中 DLY 字段携带的值。带宽的值也类似。这都是可以算出来的,没啥问题。

那么现在，我们在 R2 的配置上增加如下：

```
access-list 1 permit 192.168.12.0
```

```
router rip
```

```
offset-list 1 out 10000 fastEthernet 1/0
```

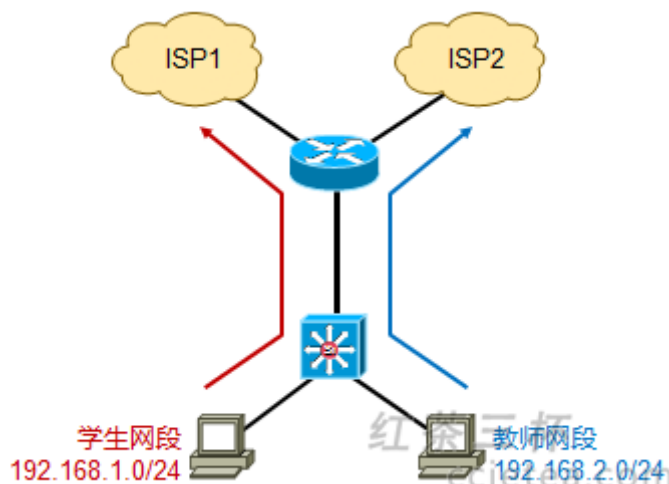
这条命令的直接结果是，R3 原本路由表中 192.168.12.0 的 metric 为 30720，而现在则加了 10000，变成 40720。我们看到，配置了 offset 命令后，实际上变化的是更新给邻居的 DLY。因为确实 DLY 最好算，默认的 $\text{metric} = \text{BW} + \text{DLY}$ ，而其中 DLY 是路由沿途入接口的 DLY 的累加，而 BW 则是最小接口带宽。显然通过 DLY 来控制，最简单方便。

7.3 Policy-based Routing (PBR) 策略路由

7.3.1 关于 PBR

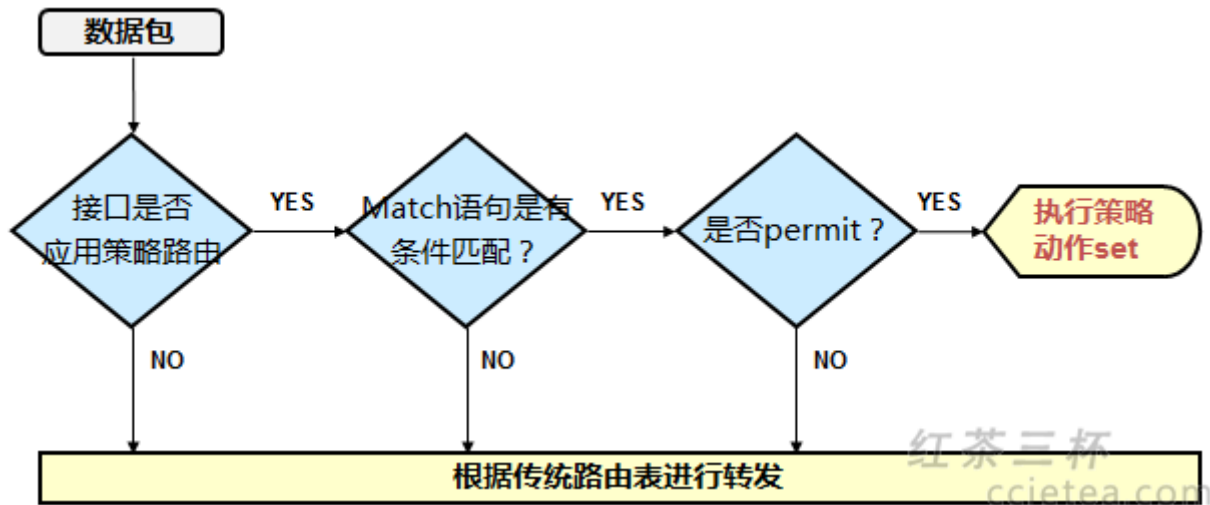
传统的 IP 路由的概念我们再回顾一下：当路由器收到一个 IP 数据包的时候，查看数据包的 IP 包头，将包头的目的 IP 拿到路由表中按最长匹配原则进行比对，最后，将数据包转发出去。因此，传统的 IP 路由只能根据数据的目的进行选路控制。

基于策略的路由比传统路由能力更强，使用更灵活，它使网络管理者不仅能够根据目的地址而且能够根据协议类型、报文大小、应用或 IP 源地址来选择转发路径。



例如上图中，网络有两条出口线路，分别连接到 ISP1 及 ISP2，如果希望让内网学生用户，访问外网的时候走 ISP1 的线路，而教师网段访问外网的时候走 ISP2 线路，这是传统路由无法实现的，因为传统的 IP 路由选择不会去关心数据的源地址。这就必须借助 PBR 了。

当我们在一个接口上部署了 PBR 后，如果这个接口收到一个数据包，它将：



7.3.2 命令汇总

1. Route-map xx permit 10

这个就不多说了吧？对于 PBR 而言，在创建 route-map 的时候，都是 permit 的。

2. match ip address

后面跟上 ACL，用于匹配流量。

3. set interface

设置数据的出接口

4. set ip next-hop { ip-address [...ip-address] | recursive ip-address }

允许写多个下一跳 IP，但这些 IP 必须是直连路由器的接口 IP。

如果定义了多个下一跳 IP，则当第一个下一跳关联的本地出接口 DOWN 掉，则自动切换到下一个 next-hop。

- **recursive next-hop (递归下一跳)** 特性突破了传统下一跳必须是直连路由器下一跳接口 IP 的限制。Recursive next-hop 可以不是直连网络，只要路由表中有相关的路由可达即可。一般 recursive next-hop 不可达，数据将交由路由处理（一般就被默认路由匹配走了）
- 如果在一个 route-map 列表的同一个序列中同时使用 ip next-hop 及 ip next-hop recursive，则 ip next-hop 有效。如果 ip next-hop 挂了，则启用 ip next-hop recursive，如果 ip next-hop recursive 和 ip next-hop 都挂了，则丢给路由表处理。注意：一个 route-map 序列（如 route-map test permit 10），只允许配置一个

ip next-hop recursive

5. set ip next-hop verify-availability [next-hop-address sequence track object]

检测下一跳的可达性，默认是关闭的

Sequence of next hops. The acceptable range is from 1 to 65535.

此条命令可以下列方式使用：

- **在 PBR 环境下使用 CDP 检测下一跳 IP 可达性（不加后面的可选参数）**

使用该特性可能会一定程度上降低设备性能，另外必须保证自己以及邻居路由器接口 CDP 都是开启的，过程交换及 CEF 都支持该特性，但 dCEF 不支持。

该特性借助设备的 CDP 表来判断下一跳的可达性，

如果本端开启了该特性，next-hop 设备不支持 CDP，则切换至下一个 next-hop，如果没 next-hop 了，则跳过 PBR

如果本端没开启该特性，那么数据包要么被成功策略路由，要么永远无法正常路由出去（被丢弃）

如果仅仅想检测部分 next-hop 设备的可达性，则可以配置不同的 route-map 序列号，来选择性的使用该特性（在同一个 route-map 中）。

- **结合 object tracking 来检测一个远端设备（或 IP）的可达性**

使用 object tracking, PBR 可以做的更加灵活，可依据 ICMP、HTTP、路由表中某条路由的存在与否、接口的 up/DOWN 等来进行决策。

注意：如若基于 CDP 的检测及基于 object tracking 的检测都应用了，则后者优先

6. set ip next-hop 与 set ip default next-hop 的区别比较简单，这里就不解析了

在本文档的 route-map 章节有介绍

7. ip policy route-map x

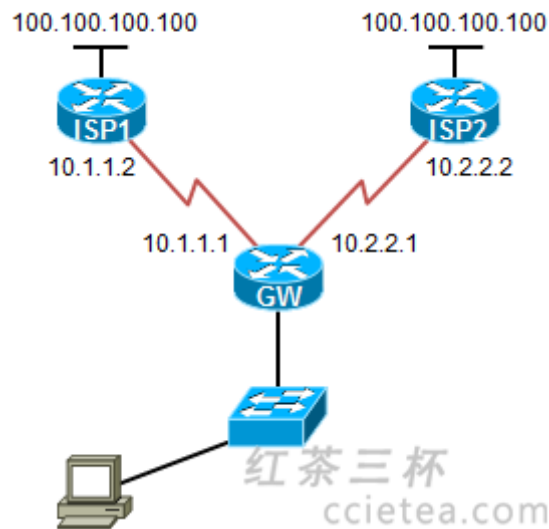
在接口上定义好的 route-map。这种方式，将只会对进入该接口的流量执行 PBR 动作。而对本设备始发的流量无效。

8. ip local policy route-map x

这条全局配置命令，可以使得 PBR 对本地始发的流量生效。

7.3.3 实验验证

7.3.3.1 set ip next-hop



GW 的配置如下：

```
access-list 1 permit any
route-map PBR permit 10
  match ip address 1
  set ip next-hop 10.1.1.2 10.2.2.2
interface fast 1/0
  ip policy route-map PBR
!! GW 并无其他关于路由的配置
```

实验现象：

1. 当网络正常时，数据强制走 ISP1，ping 100 的远程网络数据到 ISP1
2. 当 ISP1 宕机时，GW 连接 ISP1 的接口 DOWN 掉，则 PC 访问 100 的流量自动切换至 ISP2
3. 当 ISP1 宕机时，且 GW 检测不到时（也就是 GW 连接 ISP1 的接口没 DOWN），PC 访问 100 的流量仍然被扔给 ISP1，这就断网了

补充：

Set ip next-hop ip1 ip2 ip3，这个知识点已经没问题了吧？match 住相关条件后，数据包首先被送到第一个 next-hop ip address，如果这个 ip 地址所关联的直连接口 DOWN 了，则切换至下一个 next-hop ip address，如此反复，可以配置多个 next-hop。但是，如果直连的 next-hop（对端路由器或其接口）自己挂了，而本地直连接口没感知到（如中间串了台 switch），则无法自动切换，路由器仍然会一股脑的把数据丢给这个 next-hop。

另外，如果配置的时候命令这么写的话：

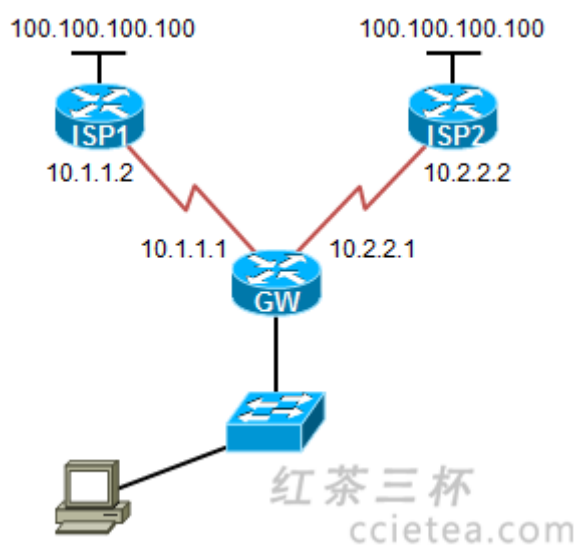
Set ip next-hop ip1

Set ip next-hop ip2

.....

则 IOS 会自动将命令变成 Set ip next-hop ip1 ip2

7.3.3.2 set ip next-hop verify-availability



GW 的配置如下：

```
access-list 1 permit any
```

```
route-map PBR permit 10
```

```
match ip address 1
```

```
set ip next-hop 10.1.1.2 10.2.2.2
```

```
set ip next-hop verify-availability
```

```
interface fast 1/0
```

```
ip policy route-map PBR
```

!! GW 并无其他关于路由的配置

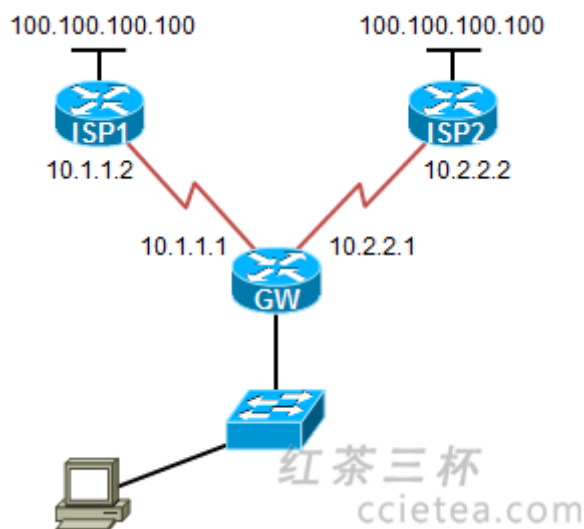
!! ISP1、ISP2 及 GW 都需开启 CDP

实验现象：

1. 当网络正常时，数据强制走 ISP1，ping 100 的远程网络数据到 ISP1
2. 当 ISP1 宕机时，且 GW 连接 ISP1 的接口 DOWN 掉，则 PC 访问 100 网络的数据切换至 ISP2

- 当 ISP1 宕机时, 且 GW 连接 ISP1 的接口没 DOWN (如关闭 ISP1 的 CDP), 由于 GW 丢失了 ISP1 的 CDP 信息, 因此认为 ISP1 挂了, 于是 PC 访问 100 网络的数据切换至 ISP2, 网络不断

7.3.3.3 set ip next-hop verify-availability 基于 object tracking



```
ip sla monitor responder
```

ip sla monitor 1

```
type echo protocol iplcmpEcho 10.1.1.2 source-ipaddr 10.1.1.1
```

```
frequency 10
```

```
exit
```

```
ip sla monitor schedule 1 life forever start-time now
```

track 1 rtr 1 reachability

ip sla monitor 2

```
type echo protocol iplcmpEcho 10.2.2.2 source-ipaddr 10.2.2.1
```

```
frequency 10
```

```
exit
```

```
ip sla monitor schedule 2 life forever start-time now
```

track 2 rtr 2 reachability

```
access-list 1 permit any
route-map PBR permit 10
  match ip address 1
  set ip next-hop verify-availability 10.1.1.2 10 track 1
  set ip next-hop verify-availability 10.2.2.2 20 track 2
```

实验现象：

1. 当网络正常时，数据强制走 ISP1，ping 100 的远程网络数据到 ISP1
2. 当 ISP1 故障，GW 通过 tracking 感知到，于是数据切换至 ISP2
3. 回复 ISP1，GW 通过 tracking 感知到，数据又切换回 ISP1

技术解析：

1. 首先定义 track object，关联到一个 ip sla monitor

使用 ICMP 协议去探测 10.1.1.2 的可达性（使用源地址 10.1.1.1 去 ping 10.1.1.2）

Track object 的 ID 为 1，关联到 ip sla monitor 1

当 10.1.1.2 可达，则 track 对象为 true

ip sla monitor 1

```
type echo protocol icmpEcho 10.1.1.2 source-ipaddr 10.1.1.1
frequency 10
exit
ip sla monitor schedule 1 life forever start-time now
```

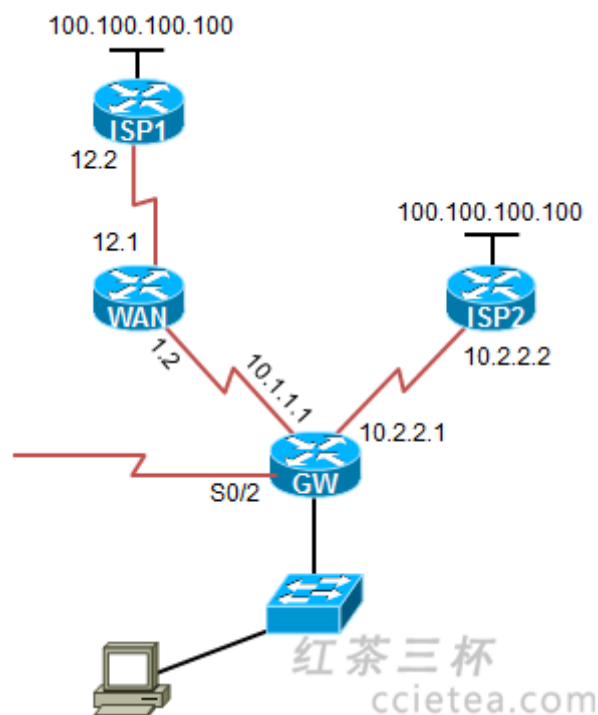
track 1 rtr 1 reachability

2. 然后在 route-map 中调用该 track object

```
route-map PBR permit 10
  match ip address 1
  set ip next-hop verify-availability 10.1.1.2 10 track 1
```

当 track1 为 true，也就是 10.1.1.2 可达，则 PC 访问 100 网络的数据被丢给 10.1.1.2，如果 track 1 挂了，则切换至下一个 next-hop

7.3.3.4 set ip next-hop recursive



GW 的配置如下：

```
access-list 1 permit any
route-map PBR permit 10
 match ip address 1
 set ip next-hop 10.2.2.2
 set ip next-hop recursive 10.1.12.2

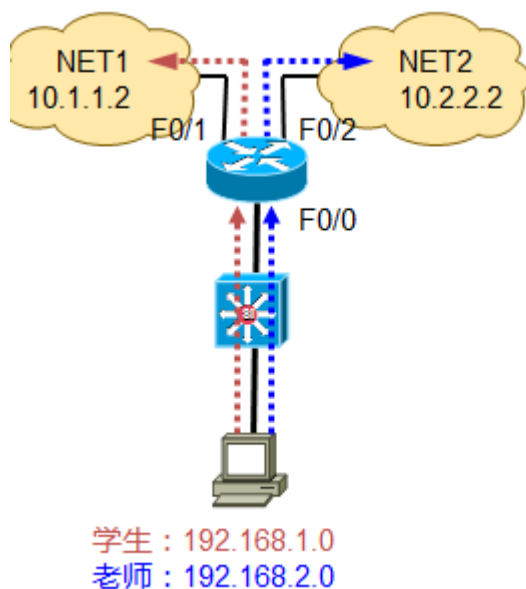
ip route 10.1.12.0 255.255.255.0 10.1.1.2
ip route 0.0.0.0 0.0.0.0 serial s0/2
```

实验现象：

1. 正常情况下，数据优先走 ip next-hop，也就是走 ISP2
2. 当 GW 连接 ISP2 的出接口 DOWN 掉（也就是 ISP2 挂了），则切换至 ip next-hop recursive,也就是 ISP2
3. 注意，这个时候是才用路由表递归找到去往 10.1.12.2 的路由的，因此路由表里必须有可达路由
4. 当 GW 丢失了去往 10.1.12.2 的路由，并且连接 ISP2 的连接也丢失了，则走默认路由

7.3.4 PBR 案例

1. 案例：通过 PBR 实现内网出口数据分流



```
access-list 1 permit 192.168.1.0 0.0.0.255
access-list 2 permit 192.168.2.0 0.0.0.255
```

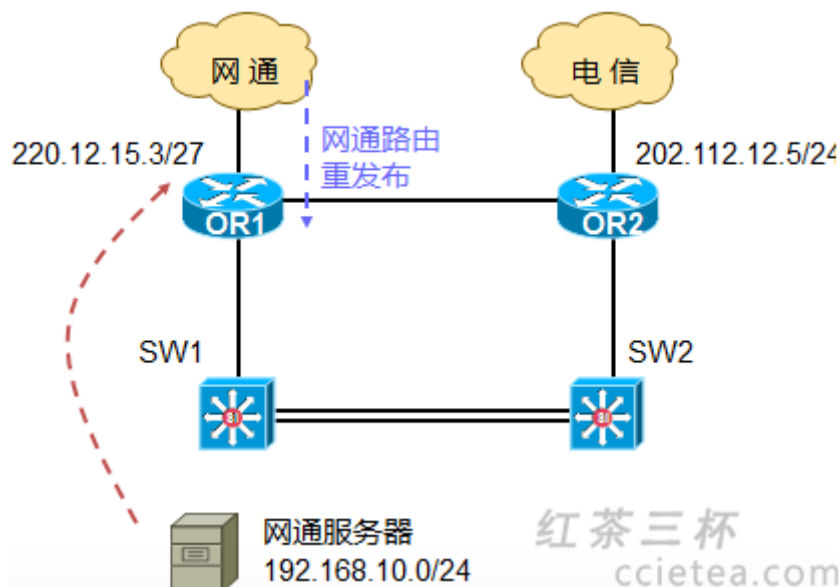
```
route-map test permit 10
match ip address 1
set ip next-hop 10.1.1.2
```

```
route-map test permit 40
match ip address 2
set ip next-hop 10.2.2.2
```

```
int f0/0
ip policy route-map test
```

```
ip route 0.0.0.0 0.0.0.0 10.1.1.2
ip route 0.0.0.0 0.0.0.0 10.2.2.2
```

2. 案例：通过 PBR 规避教育、电信双出口 NAT 网络环境中存在的问题



哥们先来介绍下上面的网络环境，这尼玛是一个相当经典的案例。OR1、OR2 是网络的出口路由器，分别连接到网通及电信的出口线路，两台出口路由器上都做了 PAT，使得内网用户能够访问外网。另外 OR1、OR2、SW1、SW2 跑 OSPF，OR2 作为电信的出口路由器，向 OSPF 域注入一条默认路由。OR1 则将本地

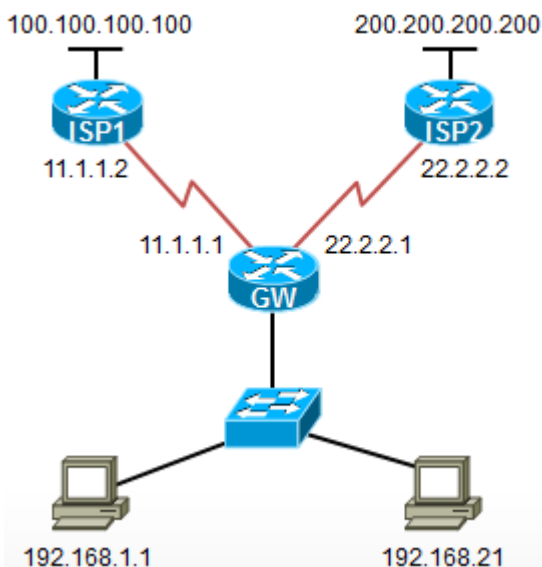
配置的静态网通路由重发布进 OSPF。这样一来，内网用户需访问网通资源时，由于有网通路由的明细，则走 OR1 出去。而访问其他资源，则走默认路由到 OR2 出去。

内网有一个网段，192.168.10.0/24，这是服务器网段，在 OR1 上部署了静态 NAT 映射，将特定的服务器映射到了 OR1 的出口上，这样一来，网通的用户能够从外网访问到这些服务器资源。但是问题来了，电信的外网用户，却无法通过映射出来的外网地址访问这些服务器资源，为什么？

仔细分析一下数据流的走向。当外网电信用户访问这些服务器资源的时候，数据的源地址是，电信公网 IP，目的地址是服务器映射出来的公网地址，于是数据从电信 WAN，绕到网通的 WAN，然后到达 OR1 的外网口，由于 OR1 上做了静态 NAT 映射，因此，数据包的目的 IP 被转换成服务器的内网 IP，数据包最终被送到了服务器。然后，服务器要回包吧？服务器的回包，源地址是服务器的内网 IP，目的地址是电信公网 IP，数据包被送到了三层交换机，三层交换机现在要做路由了，查路由表发现，只能走默认了，因此通过默认路由转发到 OR2，到了 OR2，它做了 PAT，因此将数据包的源地址，也就是服务器的内网 IP 转换成 OR2 的出口 IP，也就是一个电信的公网 IP，然后将数据包发回给那个电信的访问者，这哥们一收傻逼了，因为数据的源地址它压根不知道啊，它是访问的 OR1 上的地址啊。

这就是问题所在，怎么解决呢？很简单，在 OR2 上，部署 PBR，只要是来自网通的那些服务器数据，全部强制丢给 OR1，就解决了。

3. 案例：双出口 NAT



- 192.168.1.0 用户访问外网强制走 ISP1 的线路，当 ISP1 线路 DOWN 掉，则自动切换至 ISP2
- 192.168.2.0 用户访问外网强制走 ISP2 的线路，当 ISP2 线路 DOWN 掉，则自动切换至 ISP1
- 内网为私有 IP，访问公网需使用公网地址

重点看 GW 的配置：

```
access-list 1 permit 192.168.1.0 0.0.0.255
access-list 2 permit 192.168.2.0 0.0.0.255
route-map PBR permit 10
  match ip address 1
```

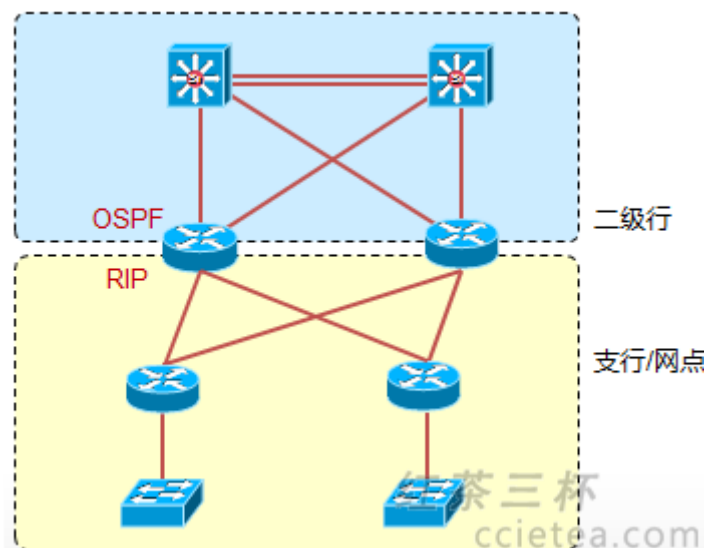
```
set ip next-hop 11.1.1.2
exit
route-map PBR permit 20
  match ip address 2
  set ip next-hop 22.2.2.2
exit
```

上面的配置只是解决数据的走向问题，但是 NAT 的问题呢：

```
route-map nat1 permit 10
  match ip address 1
  match interface serial0/0          !! 匹配数据包的出口
route-map nat2 permit 10
  match ip address 1
route-map nat3 permit 10
  match ip address 2
  match interface serial0/1
route-map nat4 permit 10
  match ip address 2
ip nat inside source route-map nat1 interface serial0/0 overload
ip nat inside source route-map nat2 interface serial0/1 overload
ip nat inside source route-map nat3 interface serial0/1 overload
ip nat inside source route-map nat4 interface serial0/0 overload
```

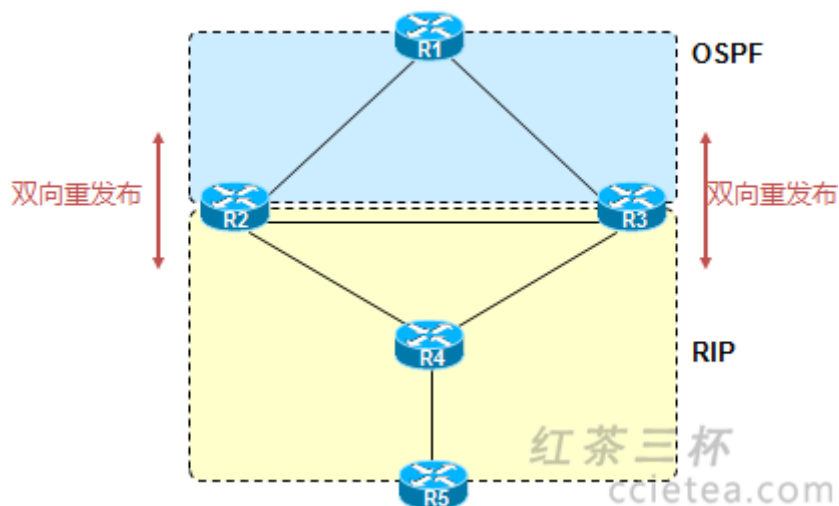
8 双点双向路由重发布

8.1 概述



“双点双向”路由重发布是一个在大型网络中时常能简单的模型。譬如金融网络中，二级行与网点路由器之间，如果运行动态路由协议，那么就有可能涉及到这个模型。所谓双点双向，指的是两个路由选择域的边界上有两个路由重发布的节点，并且重发布的方向是双向的。例如上图中，二级行有两台汇聚路由器，用于下面网点或支行路由器的接入。那么这两台路由器就是双点，另外，为了让全网的路由可达，因此在这两台路由器上部署双向路由重发布，也就是 OSPF 向 RIP 重发布，同时 RIP 也向 OSPF 来做重发布。

双点双向路由重发布的过程中，可能会发生许多问题，例如次优路径、潜在路由环路等等，本章对这个模型以及存在的问题将做深入探讨。



我们主要分析：

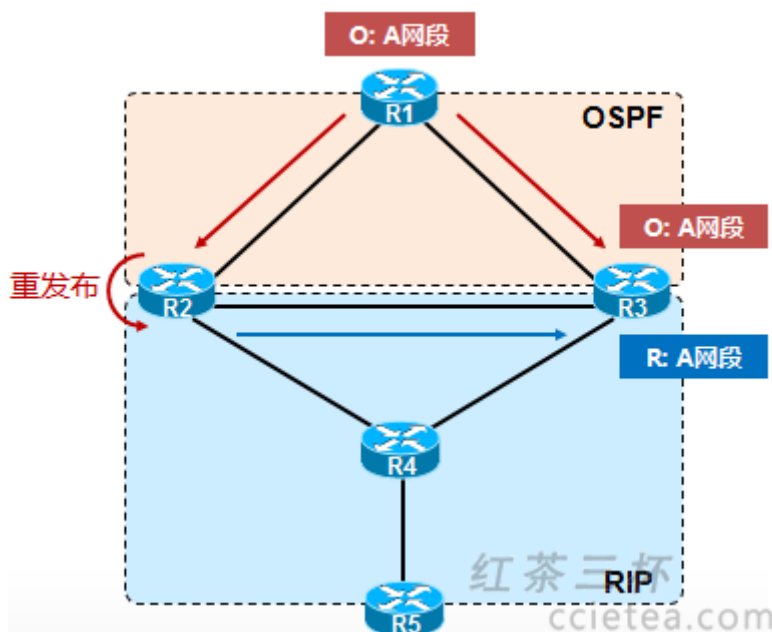
- RIP 与 OSPF
- OSPF 与 OSPF
- OSPF 与 BGP

这三个有代表性的路由重发布模型。

8.2 双点双向路由重发布存在的问题

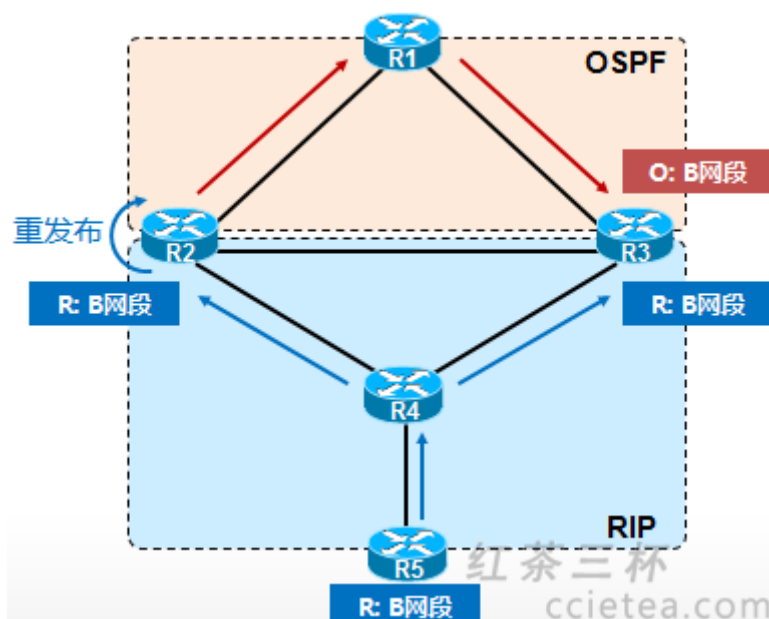
1. OSPF 与 RIP 的双向重发布

- OSPF 向 RIP 重发布路由：

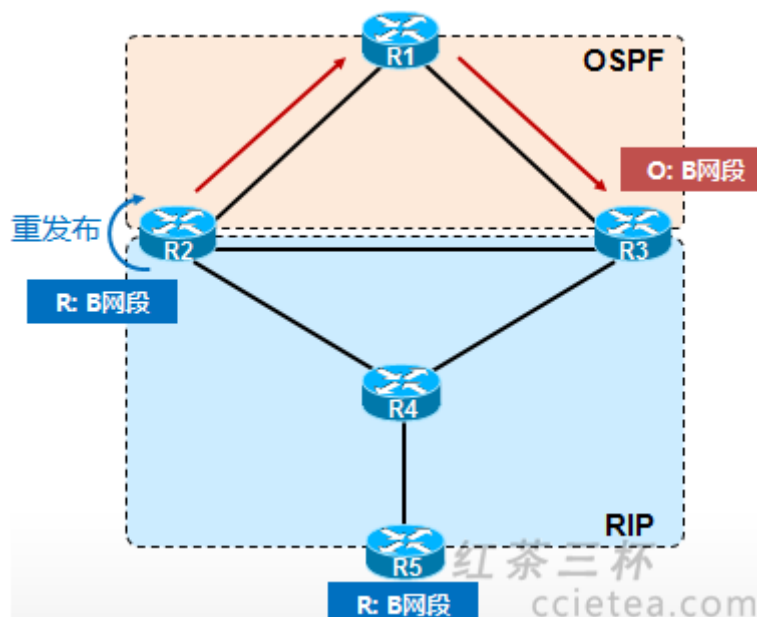


R1 发布 A 网段路由，R3 会学习到这条 OSPF 路由；另外，R2 也学到了，假设 R2 先部署 OSPF 到 RIP 的路由重发布，这条路由最终 R3 会通过 RIP 也学习到，那么 R3 将同时从 RIP 及 OSPF 学习到这条路由，R3 会优选 OSPF 路由，因为 OSPF 的 AD 小，所以 OSPF 向 RIP 重发布，不会造成次优路径问题。

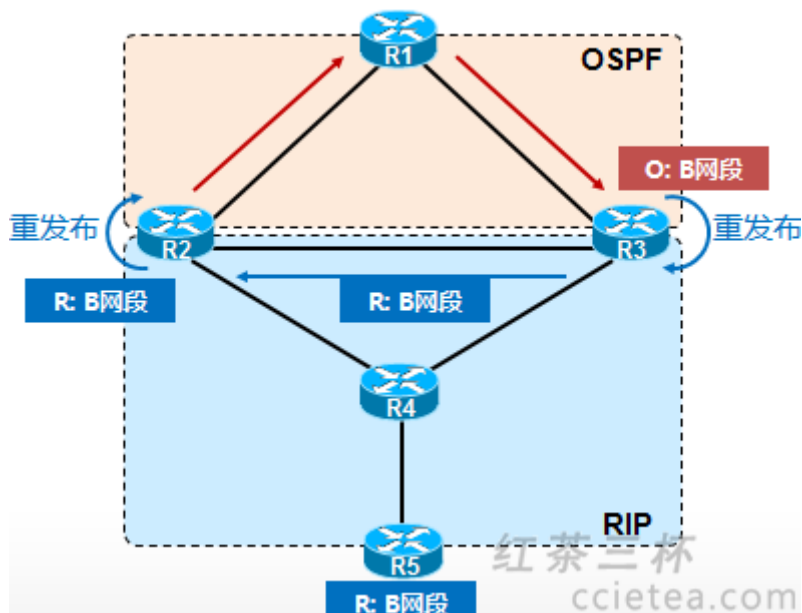
- RIP 向 OSPF 重发布路由：



R5 发布 B 网段的 RIP 路由，R2 及 R3 都能学习到，此刻我们在 R2 及 R3 上都向 OSPF 来注入 RIP 路由，假设 R2 先注入的，那么 B 路由将被 R1 更新给 R3，R3 这时候同时从 RIP 及 OSPF 都学习到了 B 路由，那么它会：

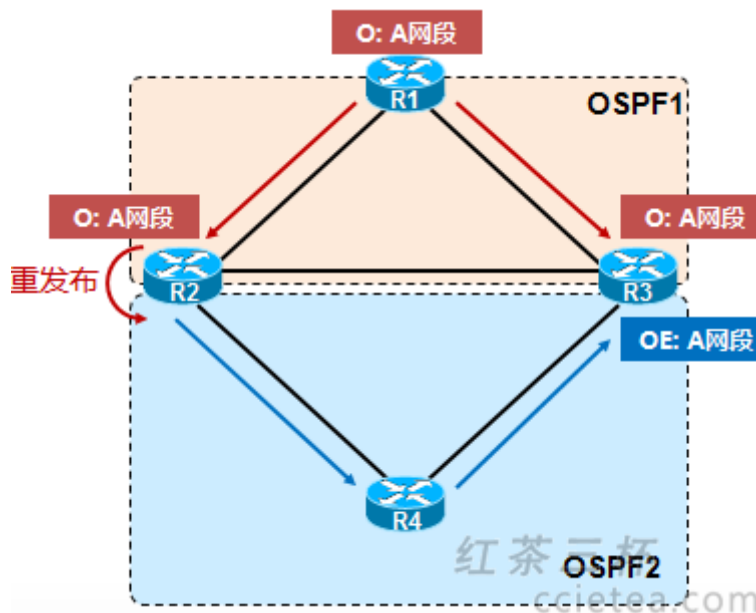


毫无疑问，R3 会优选 OSPF 的 B 路由，因为 OSPF AD 小，这样 R3 就形成了次优路径，它去往 B 网段，会选择 R1-R2-R4-R5 这条路径。再者，在 R3 的路由表中，关于 B 网段的路由是 O 的，那么即使 R3 做了从 RIP 到 OSPF 的重发布，重发布肯定是失败的，因为前面我们说过了，路由协议重发布只会注入路由表里有的路由，而此刻路由表关于 B 的路由是 OSPF 的，所以当然重发布失败。而且问题还没完，我们接下去看，如果此刻 R3 是双向重发布呢？也就是 R3 又部署了 OSPF 到 RIP 的重发布：



由 R3 路由表里是有 OSPF 的 B 网段路由，那么这条 B 网段路由会被重发布回 RIP 域，并且最终更新给 R4 和 R2，我们看 R2，R2 此刻是从 R4 已经学习到这条 RIP 路由，如果它又从 R3 也学到这条路由并且这条路由的 metric 更小的话，R2 就会优选来自 R3 的这条路由，这样一来，R1、R2、R3 就形成了一个路由环路。

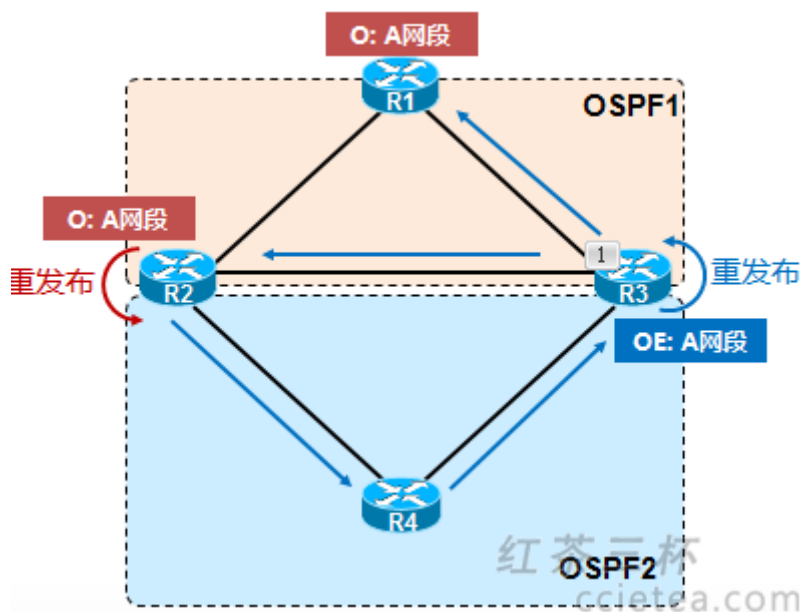
2. 不同的 OSPF 进程之间重发布路由



R2、R3 上，各有两个 OSPF 进程，如果所示，一个为 OSPF1，一个为 OSPF2，注意，其实这个双进程，主要是体现在 R2 和 R3 这两台 ASBR 上。

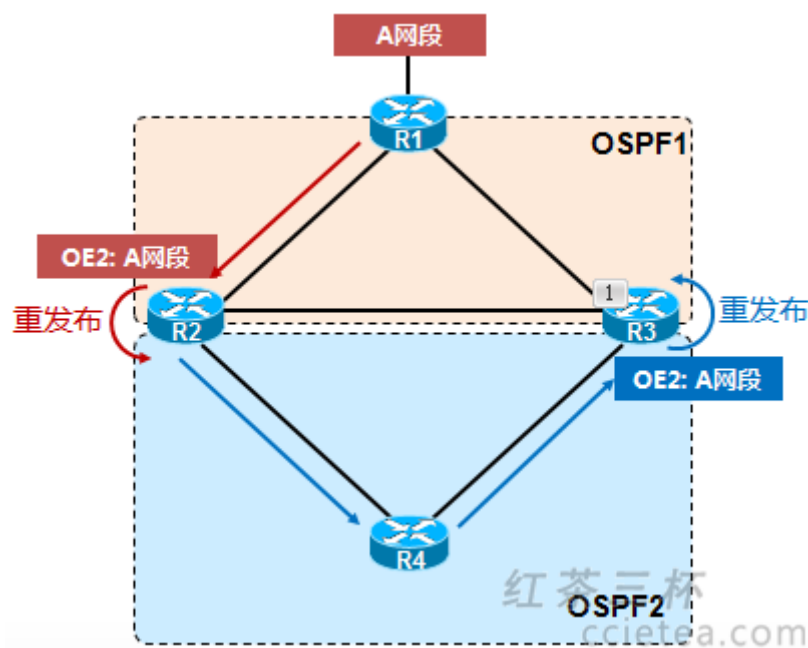
现在我们考虑当 R1 始发一条 OSPF 内部路由，R2、R3 都能学习到这条路由。那么在 R2 上，将路由从进程 1 重发布到进程 2，最终 R3 也会学习到这条从 R4 传递过来的路由。那么对于 R3 而言，双 OSPF 进程，

同时从两个进程都学习到同一条 OSPF 路由，采用先到先得的原则录用。所以，如果 R3 先学习到 R1 传来的路由，那么自然会忽略 R2 重发布进来的那条。但是，如果 R1、R3 之间的邻居关系是在 R3 已经获得了 R2 重发布进来的这条 A 网段路由之后才起来的呢？那么 R3 将忽略 R1 更新来的路由，对于 R3 而言，去往 A 网段的下一跳就变成了 R4，这样一来就产生次优路径了。并且，R3 上，进程 1 向进程 2 的重发布也就失败了。



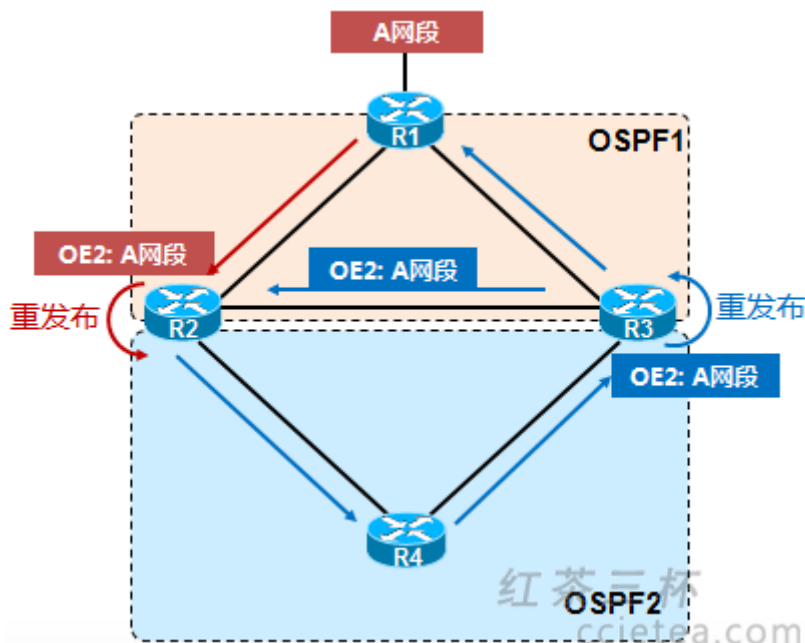
接着，R3 由于部署了双向重发布，因此会将进程 2 的路由注入进程 1，那么 A 网段的路由会被注入回进程 1，好在 O 的优先级大于 OE，因此 R1 及 R2 直接忽略这条路由。不会造成其他影响。

但是，如果 A 路由为 OSPF 外部路由呢？



假设现在环境是这样的，R1 重发布 A 路由进 OSPF，那么 R2 会在自己的 OSPF 进程 1 学习到这条 OE2 的

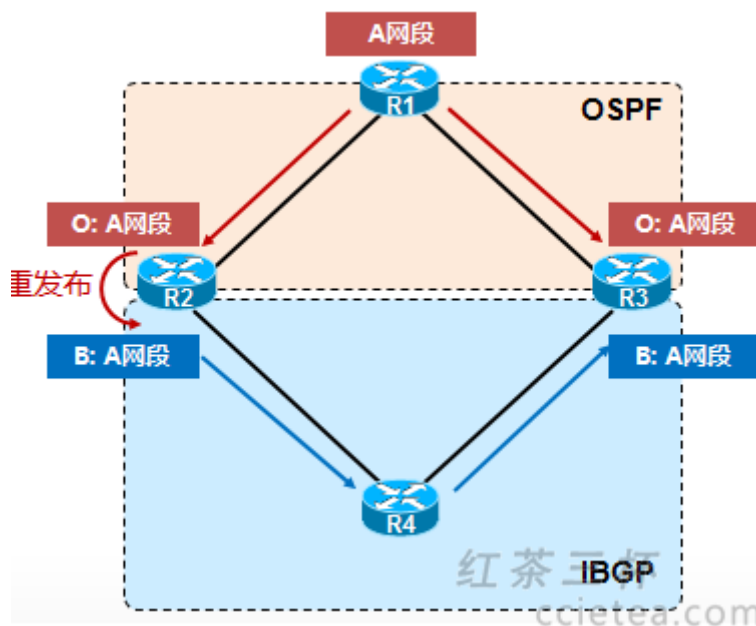
路由，并且将路由重发布进自己的 OSPF 进程 2，那么 R3 会从进程 2 学到这条路由，假设 R3 先学到这条 R2 重发布的 A 路由，那么根据先入为主的规则，R3 将忽略 R1 后来的关于 A 路由的更新。与此同时，由于部署了双向重发布，R3 又会进一步的将学习到的 A 路由重发布回进程 1。



对于 R2，它此刻是在自己的进程 1 中，同时从 R1 和 R3 学习到这条 OE2 的路由，它会如何选择？回顾一下 OE2 的比较原则，先比 OE2 的外部 metric，如果相等，进一步比较内部 metric 也就是到达 ASBR 的 metric（当然，是 FA=0.0.0.0 的情况下），那么，在这个图中，如果 R2 到 ASBR-R3 的开销更小，最终 R2 将优选 R3 重发布进来的 A 路由，于是乎，R2、R3、R4 就构成了路由环路。

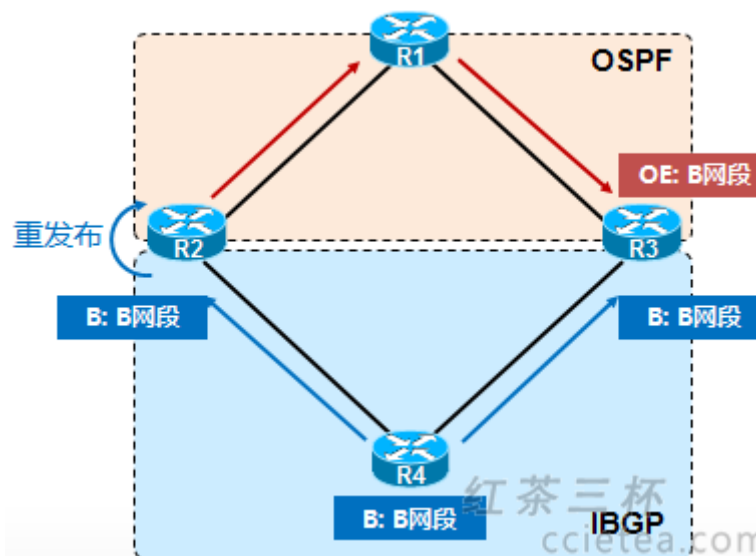
3. OSPF 与 IBGP 的路由重发布

- OSPF 向 IBGP 重发布路由



当 OSPF 向 IBGP 重发布路由时，由于 OSPF 的 AD 比 IBGP 的 AD 要小，因此不会出现次优路径或是路由环路的问题：

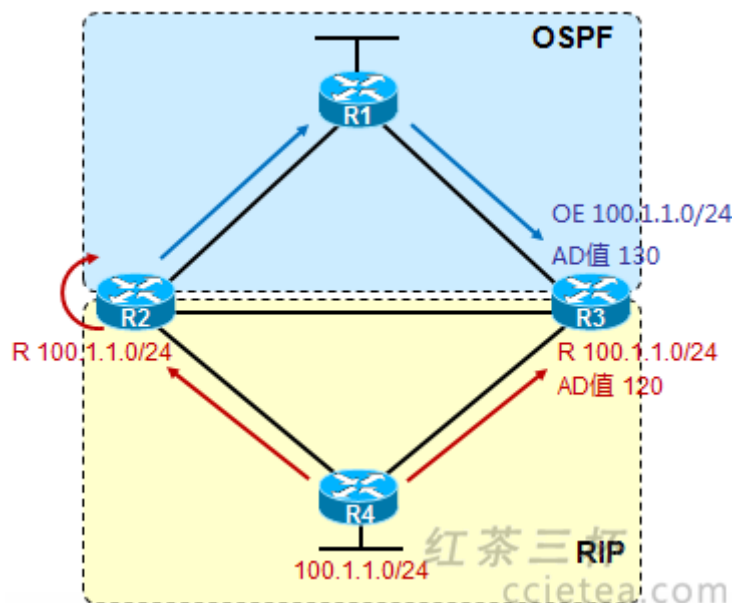
- IBGP 向 OSPF 重发布路由



问题和前面说到的类似，R2 及 R3 都能学习到更新自 R4 的 BGP B 网段路由。假设 R2 先做的重发布，那么 B 路由会注入到 OSPF 中，最终被 R3 学习到，那么 R3 上，同时从 OSPF 及 IBGP 学习到这条路由，会优选 OSPF，因为 OSPF 的 AD 要小。这样一来，R3 上，IBGP 向 OSPF 的重发布就失败了，因为 B 网段的路由在路由表中并不是 BGP 的。再者，R3 上做了双向重发布，因此 B 网段的路由，又会被重发布回 IBGP，自此，就有可能造成次优路径、环路等一系列问题。

8.3 双点双向路由重发布的实现

8.3.1 解决方案：修改路由协议管理距离



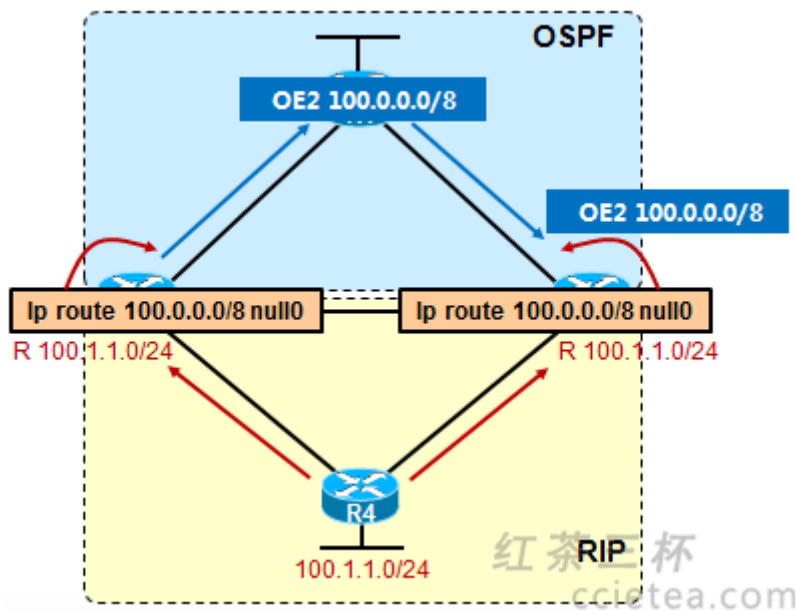
前面分析得很透彻了，当 RIP 向 OSPF 做双点重发布的时候，由于路由被注入到 OSPF 后，AD 就变成了 110，那么路由传递到 R2、R3 之后，由于 OSPF AD 比 RIP 要小，因此 OSPF 路由会覆盖 RIP 的由此造成次优路径。那么，我们可以在 R2、R3 上，用 ACL 等工具去抓取 RIP 域中的路由，然后在 OSPF 进程中，将这些路由的 AD 值调得比 RIP 要大，那么这样一来，这些路由就不会将 RIP 路由覆盖，次优路径问题也就可以规避。

举例 R3 的配置如下：

```
access-list 1 permit 100.1.1.0
router ospf 1
  distance 130 2.2.2.2 0.0.0.0 1    !!其中 2.2.2.2 为 R2 的 router-ID 注意由于 100.1.1.0 在 R3 的 OSPF
进程学习到的是外部路由，因此这里的更新源是 R2。
```

8.3.2 解决方案：采用重发布静态汇总路由的方式规避次优路由等问题

• RIP 与 OSPF 的重发布



R4 发布 100.1.1.0/24 的 RIP 路由，两台 ASBR 都能够学习到。

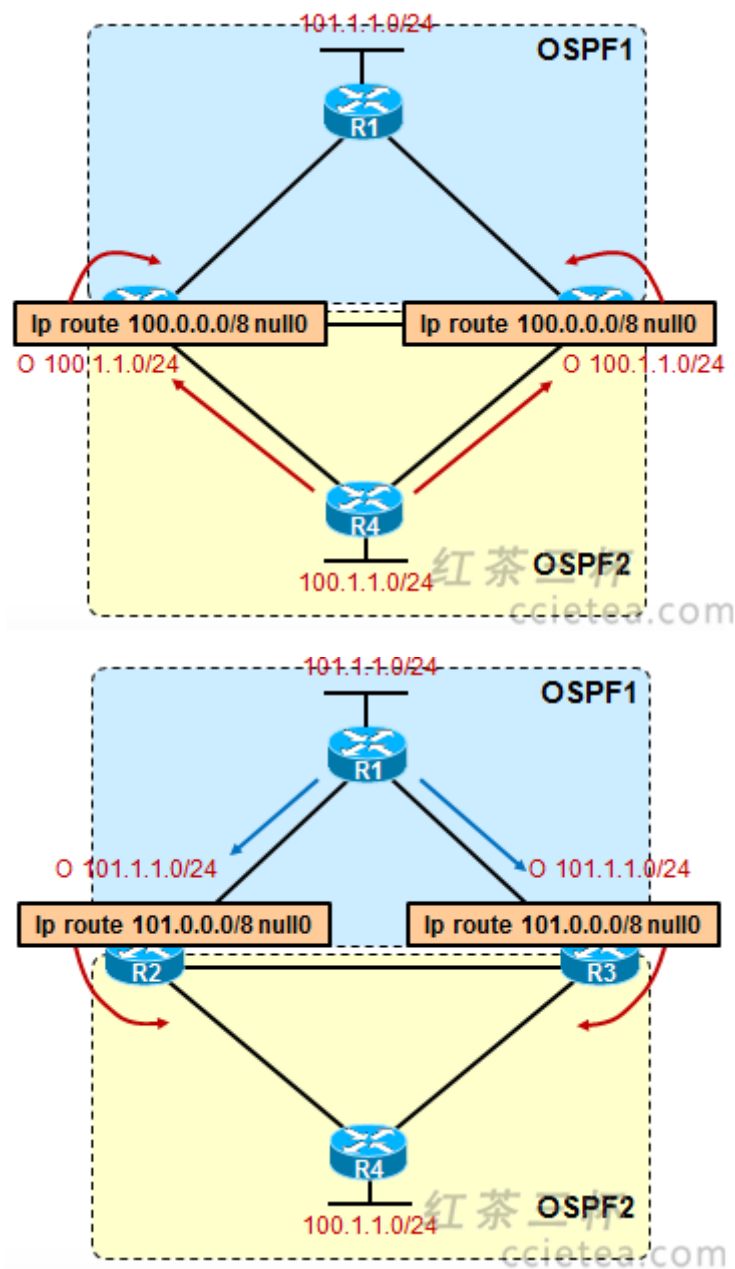
解决的办法是，在两台 ASBR 上配置静态汇总路由：

```
ip route 100.0.0.0 255.0.0.0 null0
```

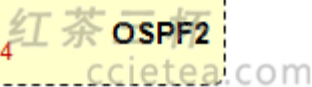
然后仅仅将这条静态汇总路由重发布进 OSPF。这样一来 OSPF 内的路由器都能通过这条汇总路由到达 100.1.1.0/24，同时，这条重发布后变成 OSPF 的汇总路由，就算传回了两台 ASBR，由于本地有静态的汇总路由，因此直接忽略 OSPF 的汇总路由，而不会产生次优路径，这条路由也不会被重发布回 RIP 而导致其他的什么问题。

• 不同的 OSPF 进程之间的重发布

不同的 OSPF 进程之间双点双向重发布，同样可以使用静态汇总路由重发布的方式，分别在两台 ASBR 上面创建不同的静态汇总路由，在相应的进程下进行重发布静态汇总路由。



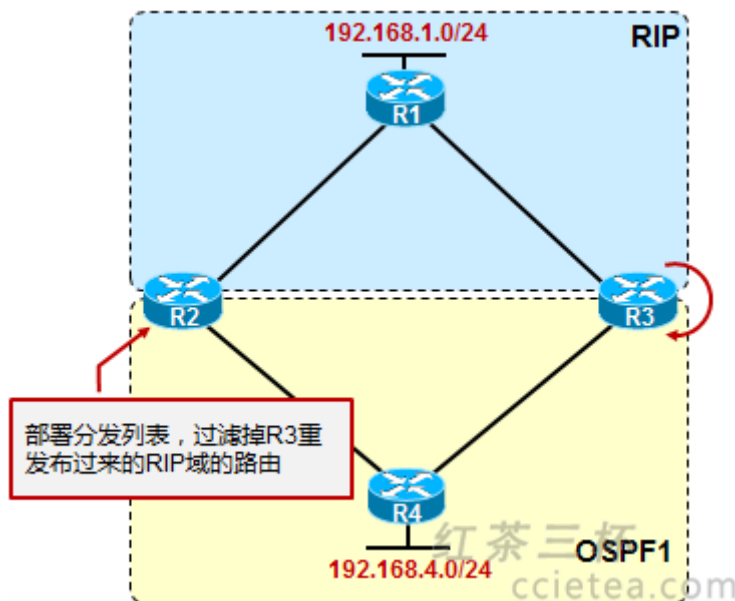
那么，如果某个 OSPF 进程中（这句话可能不大严谨，大家理解我的意思就成），路由无法汇总咋办，例如下图。



为 OSPF2 内的路由创建静态汇总路由 (Null0), 并将静态路由重发布进 OSPF1 中
修改 OSPF1 的管理距离, 使 OSPF1 的优先级高于 OSPF2 的管理距离

```
ip route 100.1.0.0 255.255.0.0 null0
!
access-list 1 permit 192.168.1.0
!
router ospf1
    redistribute static subnets
    distance 100 192.168.12.1 0.0.0.0 1
```

8.3.3 解决方案：使用分发列表规避次优路径问题

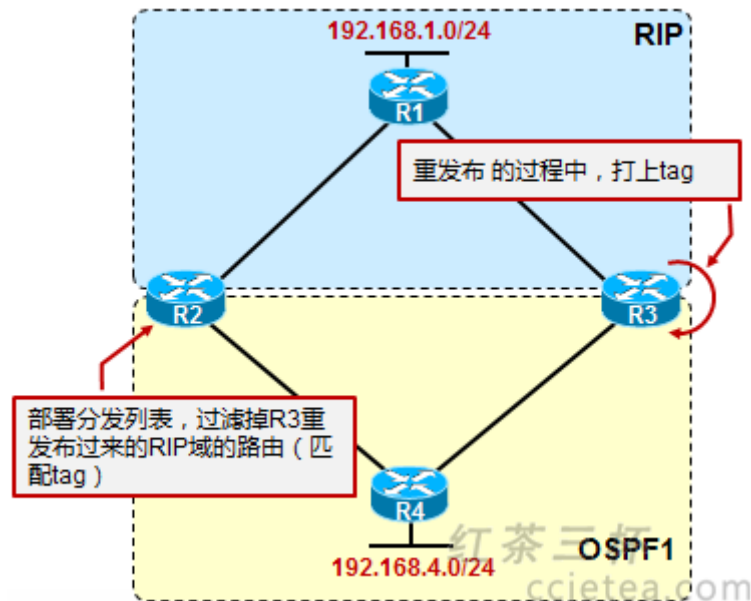


一个非常简单的方法，假设在 R3 上，先做的 RIP 到 OSPF 的路由重发布，前面已经说过了，一个直接的问题是在 R2 上，由于 192.168.1.0/24 这条路由，同时从 OSPF 及 RIP 学习到，而 OSPF 的 AD 值较小，从而导致次优路径的产生，因此为了规避这个问题，我们可以直接在 R2 上部署分发列表，使得 OSPF 路由不加载进路由表，从而规避次优路径问题。

在 R2 上配置如下：

```
access-list 1 deny 192.168.1.0
access-list 1 deny 192.168.13.0      !!这是 R1-R3 之间的直连链路
access-list 1 permit any
!
router ospf 1
  distribute-list 1 in
```

当然 R3 上也要做类似的配置。



上面的规避方案，我们是直接用 ACL 来匹配路由，再在分发列表里进行调用，这种方法可扩展性不高，如果重发布进 OSPF 的路由条目太多，ACL 条目就会写疯。因此我们尝试用一种更具有扩展性的方法，举例来说，在 R3 上，将 RIP 重发布进 OSPF 的时候，将注入的路由打上 tag，然后呢在 R2 上，部署分发列表的时候，就可以关联一个 route-map 来“抓取”这些带了 tag 的路由。

R3 的配置如下：

```
router ospf 1
```

```
  redistribute rip subnets tag 1111
```

完成这个配置后，我们可以 show 一下：

R4#sh ip ro 192.168.1.0

Routing entry for 192.168.1.0/24

Known via "ospf 1", distance 110, metric 20

Tag 1111, type extern 2, forward metric 64

Last update from 192.168.34.3 on Serial0/1, 00:09:32 ago

Routing Descriptor Blocks:

* 192.168.34.3, from 192.168.3.1, 00:09:32 ago, via Serial0/1

Route metric is 20, traffic share count is 1

Route tag 1111

由于我们在 R3 上，将 RIP 路由注入 OSPF 后打上了 tag 1111，那么接下去

在 R2 上：

```
route-map test deny 10
```

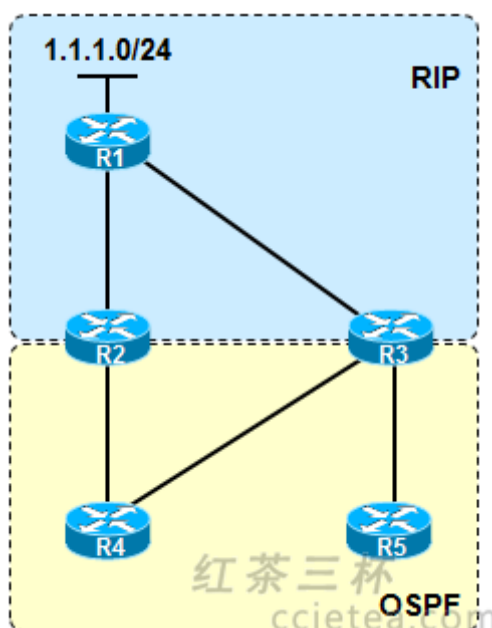
```
  match tag 1111
```

```
!
```

```
route-map test permit 20
router ospf 1
network 192.168.24.2 0.0.0.0 area 0
distribute-list route-map test in
```

8.3.4 思考题

这个综合思考题帮助大家梳理一下几种解决方案：

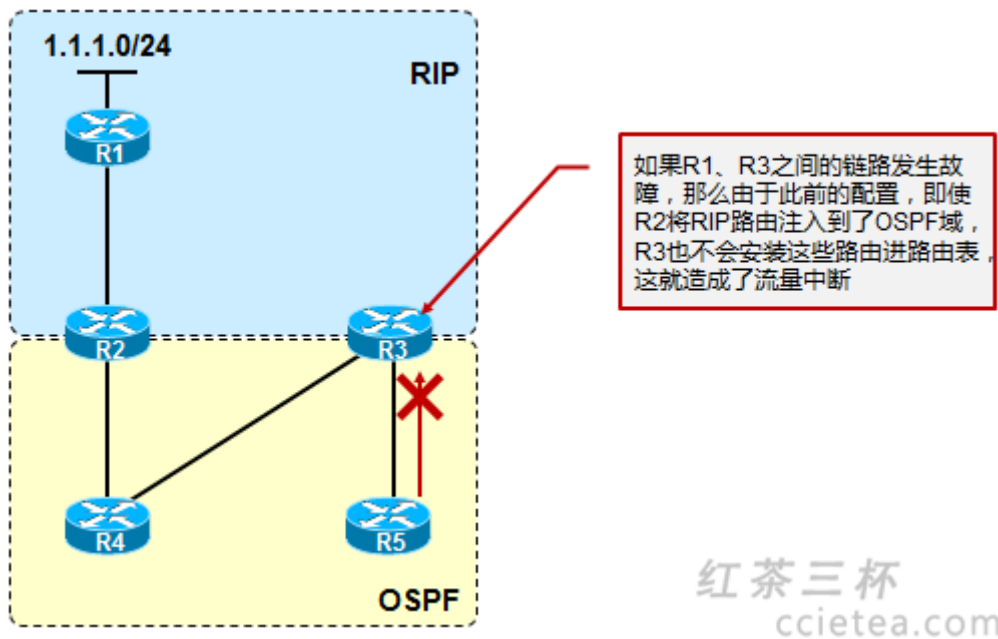


我们看上图，在 R2、R3 上部署了双向重发布。

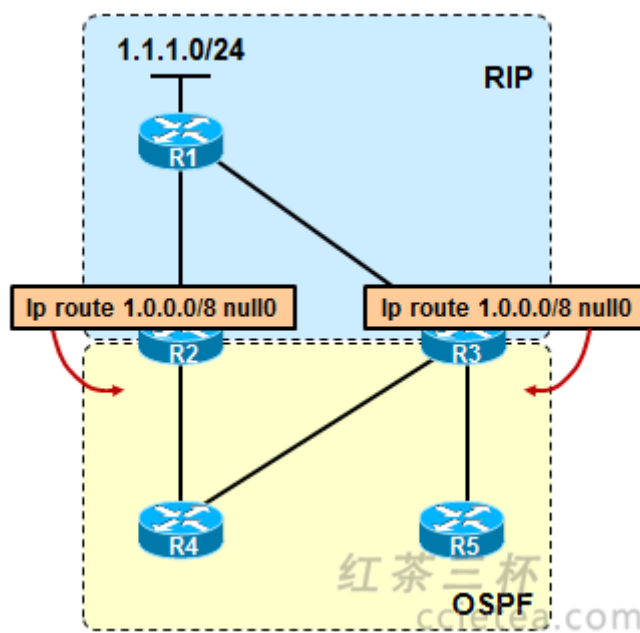
首先为了规避次优路径问题，我们在 R2 及 R3 上，使用 in 方向的分发列表，将 1.1.1.0 过滤掉。

```
access-list 1 permit 1.1.1.0
router ospf 1
distribute-list 1 in
```

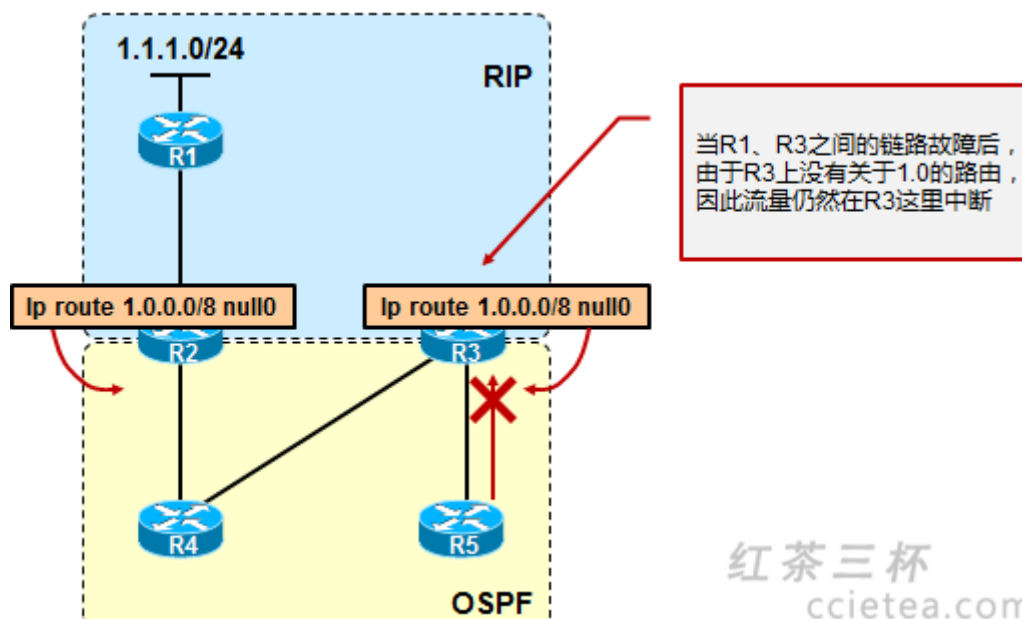
这样的确可以规避次优路径问题，但是，却留下了一个隐患：



那么如果我们用重发布静态汇总路由的解决方案呢：

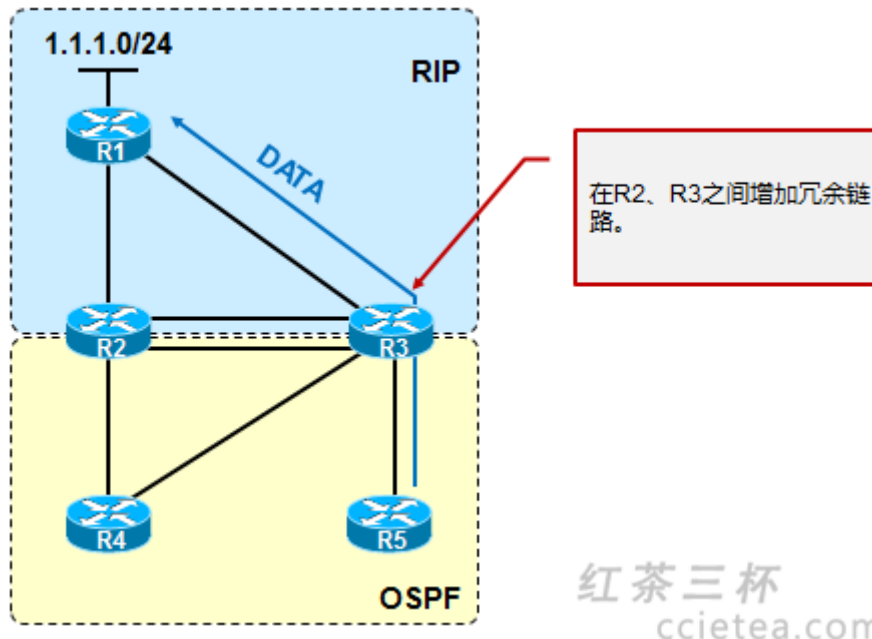


R2、R3 上，针对 RIP 域内的路由创建静态汇总路由，指向 null0，同时只将这些静态路由重发布进 OSPF，也确实可以起到规避次优路径的问题，但是：

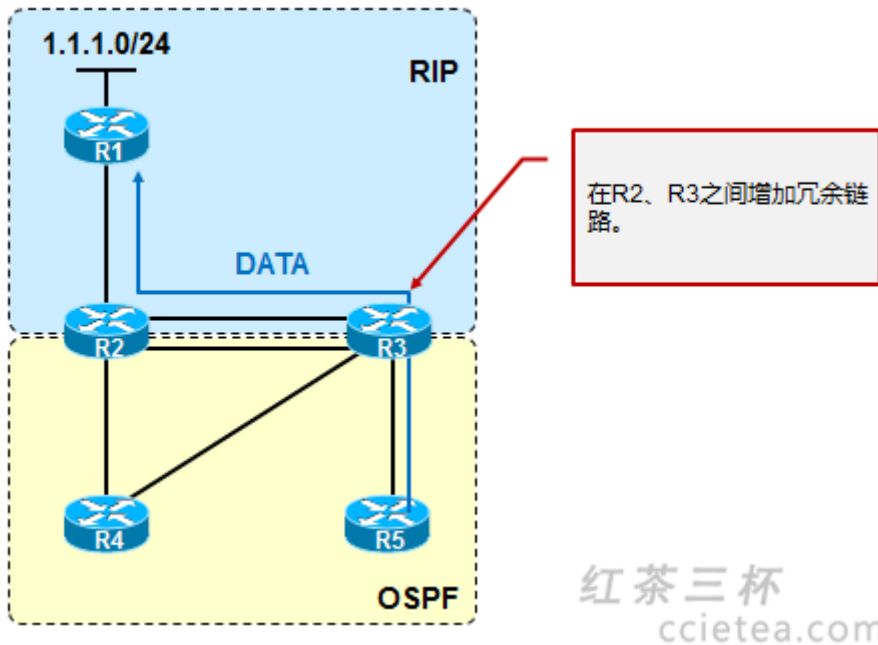


当 R1、R3 之间链路发生故障时，同样网络还是会出问题。因为虽然 R2 将本地配置的静态汇总路由重发布进了 OSPF，但是 R3 是忽略这条路由的（本地配置了静态的指向 null0 的汇总路由），虽然 LSA 还是被传递给了 R5，也就是说虽然 R5 还是有 1.0.0.0/8 的路由，但是数据包丢给 R3 后，最终在 R3 这里被丢弃，因此这里还是有问题。

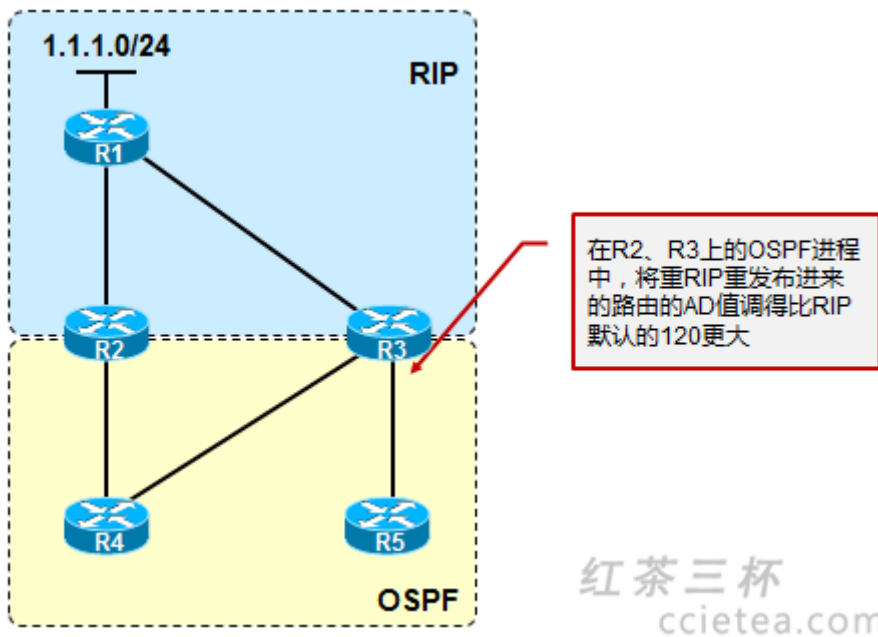
比较理想的解决方案之一是，改造网络拓扑：



这样一来，无论是采用重发布静态汇总路由的方式，还是分发列表过滤路由的方式，都可以在网络出现故障的情况下，保证网络不中断，如下：



另一个推荐的解决办法是，调整路由协议的管理距离：



9 缺省路由

9.1 ip default-gateway

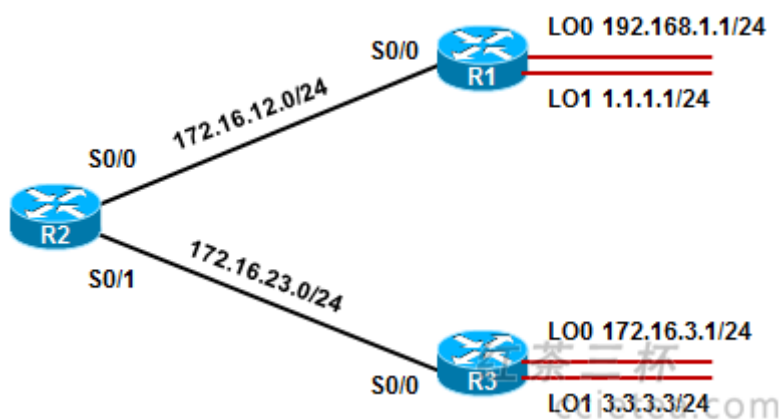
仅在路由器（或三层交换机）关闭路由功能的时候（no ip routing）有效；

如果开启了 ip routing，则忽略该命令（而是看默认路由 ip route 0.0.0.0 0.0.0.0 x）；

路由器在 boot 模式下，做软件升级的时候，由于 ip routing 也是关闭的，因此在必要时该命令也会用到。

9.2 ip default-network

当使用 ip default-network 在本地指一个网络时，这个网络号如果在路由表中存在，那么路由器会将该网络指定为缺省网关。



● 实验 1

完成基本的接口 IP 配置，R2 上增加如下配置：

```
ip route 192.168.1.0 255.255.255.0 172.16.12.1 // 路由表中有了标记为 S 条目
ip default-network 192.168.1.0 // 将路由表中 192.168.1.0 标记为缺省网络
```

完成配置后，R2 show ip route

```
Gateway of last resort is 172.16.12.1 to network 192.168.1.0
```

```
S* 192.168.1.0/24 [1/0] via 172.16.12.1
```

这时 R2 ping 1.1.1.1 就能通了（相当于将 172.16.12.1 作为最后一跳求助对象）

• 实验 2

完成基本的接口 IP 配置，R2 上增加如下配置：

```
ip route 172.16.3.0 255.255.255.0 172.16.12.3
ip default-network 172.16.3.0
```

完成配置后，R2 show ip route

```
S 172.16.0.0/16 [1/0] via 172.16.3.0          // 出来一条汇总路由，而不是缺省路由
S 172.16.3.0/24 [1/0] via 172.16.23.3
```

R2 show run 后发现：

```
ip default-network 172.16.3.0 变成了：ip route 172.16.0.0 255.255.0.0 172.16.3.0
```

这是因为 ip default-network 是有类的，因此如果使用该命令标记一个主类网络的某个子网，实际上路由器会安装该子网对应的主类网络路由进路由表而不会产生任何缺省路由。所以这时候就在上面的基础上，由于产生了 172.16.0.0 的路由，因此可以再使用一次 ip default-network 命令

```
ip default-network 172.16.0.0
```

这样一来路由表：

```
Gateway of last resort is 172.16.3.0 to network 172.16.0.0
* 172.16.0.0/16 is variably subnetted, 4 subnets, 2 masks
S* 172.16.0.0/16 [1/0] via 172.16.3.0
S 172.16.3.0/24 [1/0] via 172.16.23.3
```

如此 R2 就能 ping 通 3.3.3.3

• 实验 3

前面是使用静态路由，如果使用动态路由协议，那么情况就不太一样了，例如 IGRP 或 EIGRP，ip default-network 命令指定的网络就必须是通过 IGRP 或 EIGRP 获取到的（宣告、学习或重发布），该条缺省路由才能被动态的传递给其他协议路由器。

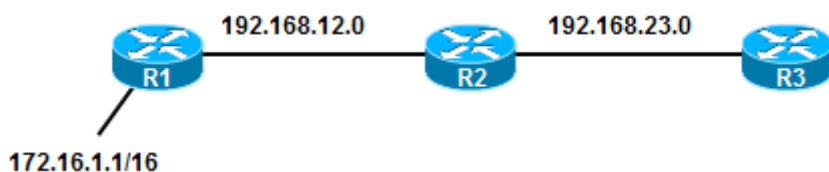
如果是 RIP，ip default-network 命令指定的网络则无需被显示宣告进 RIP，且一旦缺省网关被指定，RIP 会发布一条默认路由进 RIP，传递给其他路由器。例如 R1、R2、R3 运行 RIP，在 R1 上开 loopback 192.168.1.0/24，RIP 只宣告直连链路（而无需在进程中宣告这个 Loopback），在 R1 上 ip default-network 192.168.1.0，则会传递 0.0.0.0 的默认路由进 RIP，R2R3 都会学习到。

总结

如果使用 ip default-network 指定了多个候选缺省路由，那么拥有最低 AD 的将会成为缺省路由，并且设定为缺省网关（gateway of last resort），如果 AD 都相等，那么 show ip route 第一个显示的，就作为缺省网关。如果同时使用 ip default-network 及 ip route 0.0.0.0 0.0.0.0，且 ip default-network 指定的网络为静态路由配置的，那么 ip

default-network 的优先，并且成为缺省网关 gateway of last resort。但如果 ip default-network 指定的网络是学习自动态路由协议，则 ip route 0/0 的优先。

9.3 RIP 重发布默认路由



• 实验 1

R1、R2、R3 运行 RIP，只宣告三者之间的直连链路。在 R1 上 ip default-network 172.16.0.0 则 R2 的路由表如下：

```

Gateway of last resort is 192.168.12.1 to network 0.0.0.0
C    192.168.12.0/24 is directly connected, Serial0/0
C    192.168.23.0/24 is directly connected, Serial0/1
R*   0.0.0.0/0 [120/1] via 192.168.12.1, 00:00:03, Serial0/0
  
```

• 实验 2

在 R1 上 ip route 0.0.0.0 0.0.0.0 null 0 然后路由进程里重发布静态，也成

• 实验 3

在 R1 上，ip route 1.0.0.0 255.0.0.0 null 0 然后 ip default-network 1.0.0.0，则 R2R3 会学习到缺省路由

• 实验 4

在 R1 上，default information originate 也成，R2 及 R3 会学习到 0/0 的默认路由。这条命令对于 RIP 而言，不要求 R1 自己本地有默认路由。

10 参考书目

IP 路由疑难解析