**网络实验报告 exp6**

**2020K8009929032 刘耕印**

1. 实验内容

实现路由器的转发功能，使得节点能够通过路由器在网络层传递数据。以一个最简单的情况为例，即课件所给的星形拓扑：

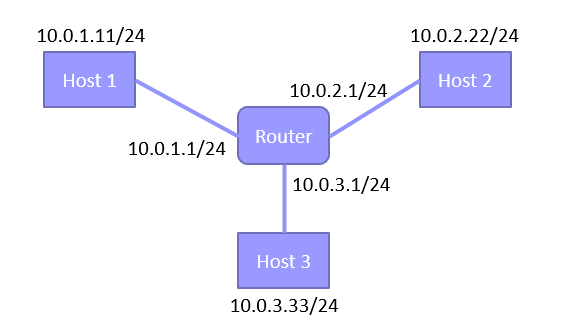


Figure 最简单的拓扑

初始时h1的默认路由网关为10.0.1.1。从主机h1向主机h3发包，该包将首先被r1的10.0.1.1端口收到。10.0.1.1检测此包不是ping自己的，并确认ttl不为0后，简单处理并根据包ip头的daddr在其本地路由表中进行最长前缀查找。不出意外的情况下，可以找到对应路由表条目，指示转发端口为10.0.3.33，网关为0即目标主机ip。但是，由于arp表中未缓存下一跳的mac地址，需要通过arp协议查询之，并将此包挂起。直到收到arp响应后才继续转发。

其中，需要实现的部分首先是arp协议，要实现arp请求和响应的发送、arp包的处理、arp条目缓存的维护，查找以及挂起未知目的的包等一系列功能。然后，在ip层面处理收到的数据包。在以上两个过程中都可能产生异常，需要通过icmp协议处理，所以最后要实现icmp包的组织和发送。

实现成功后，检测网络连通性，连通则合格。

1. 代码实现

首先概览框架，我重点阅读了arp.h, ip.h, arpcache.h, icmp.h和rtable.h几个头文件，其中声明的方法能够帮助理清思路，并对代码结构的构建有一定的指导作用。同时，其中的各种宏定义能够使代码更清晰。

1. ARP部分

我选择首先实现ARP协议相关的代码，因为这部分基本上不依赖于其他部分。首先，按照ARP协议的定义，实现ARP请求和回应功能。在这两个函数中，只要根据入参组装ARP头部并将其转发即可。其中要注意的是，入参中许多数据被我假设为网络字节序的，此时需要进行大量的字节序转换。同时，额外申请的空间要释放掉（此处其实多申请了一倍的空间，可以优化）。这两个函数比较简单，但代码比较多，就不放上来了。

然后，对于收到的ARP包，要进行处理。读取ARP头部，确定其种类后，即可对应处理。如果是一个给自己的请求，那么组织回复即可；如果是一个发给自己的回复，那么涉及到arpcache中挂起的数据包的转发，这部分马上会提到。代码如下：

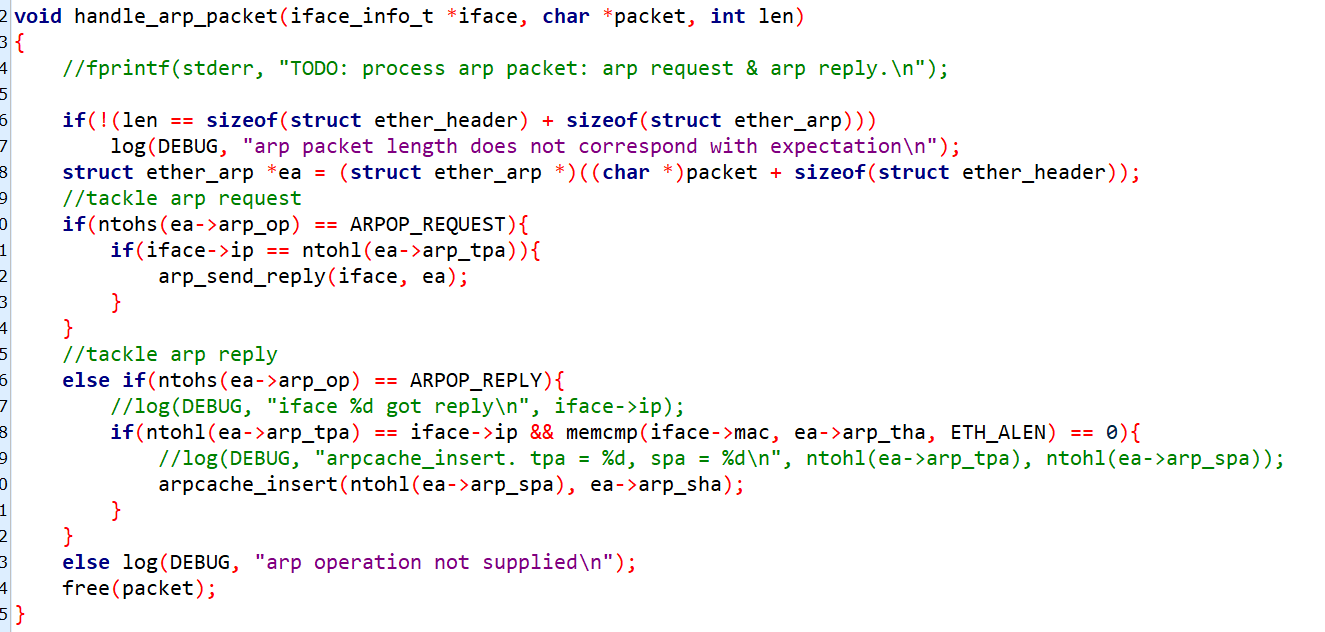


Figure 处理ARP请求和响应

接下来实现arp缓存相关操作。最基本的是查找缓存条目，只需遍历比较即可：

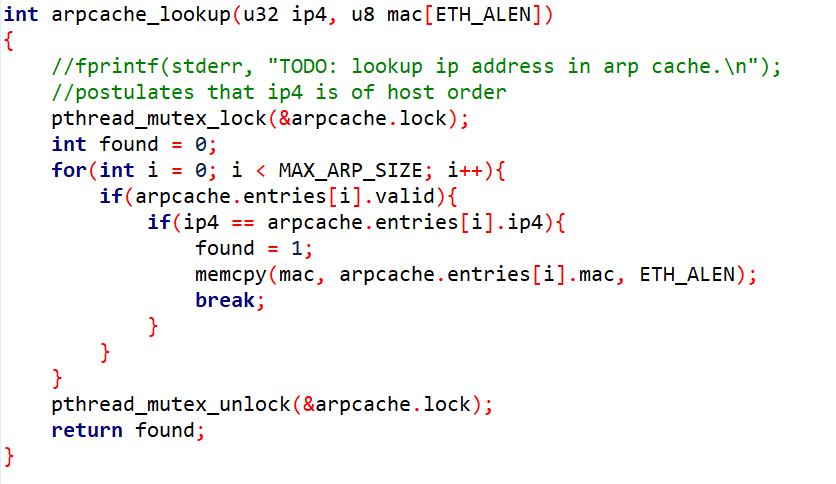


Figure arpcache查找

注意加锁。当查找不到条目时，即需要发送arp请求，此时实现上文提到的挂起操作。在arp缓存中有一个二级队列，存储被挂起的数据包。我们首先在第一级队列中查找，查找到的话就把数据包放进对应的队列中。如果没有查找到，那么还需要申请一个新的一级队列节点，并把数据包入队。在这里还涉及一个问题，即在什么时候发送arp请求。我的做法是，只有当某个一级队列节点（struct arp\_req）刚刚被建立时，才主动发送一次arp请求，并把重试次数设为1。在后边的清扫线程中，可能会继续发送请求，直到重试次数大于5为止。下边是代码：

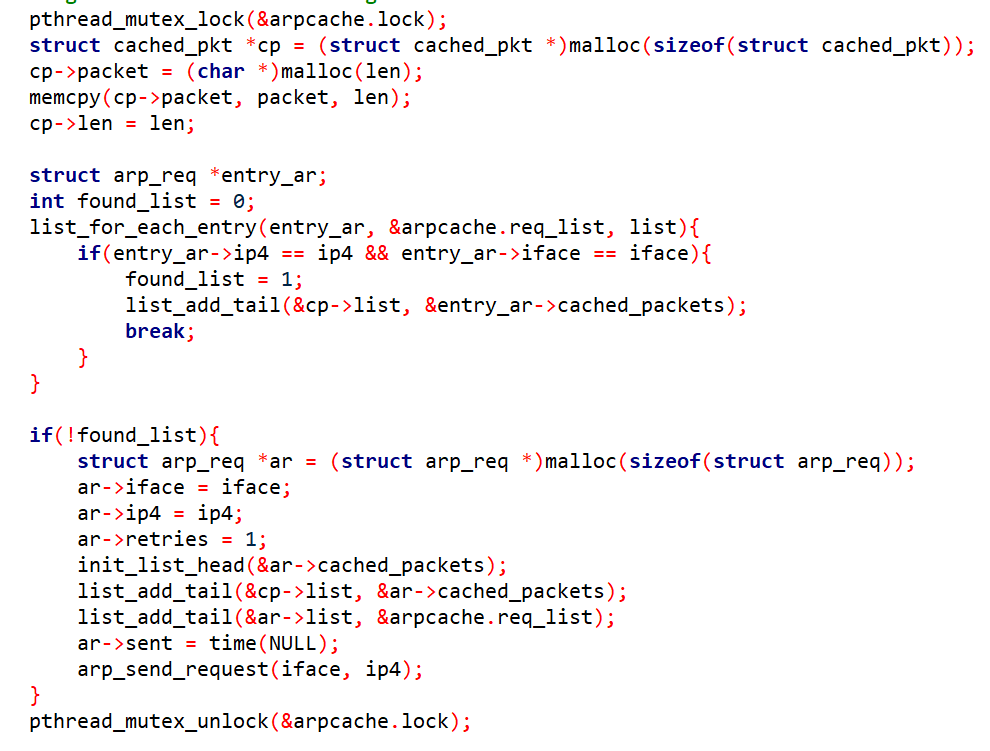


Figure 把数据包挂到arpcache中

这里其实还是犯了一个错误，多申请了一倍的空间进行了数据包的深拷贝，其实只要进行浅拷贝即可。接下来是清扫线程的实现，分为两部分，第一是缓存条目的老化，第二是重发arp请求。老化非常简单，对于超时的条目，将其valid置0即可。重发请求部分，为避免死锁，我采取了faq的建议，设立一个伪队列将需要回应的所有条目组织起来，解锁后遍历该队列进行依次回应。这里就涉及到icmp包的发送，后面会再提到。代码如下：

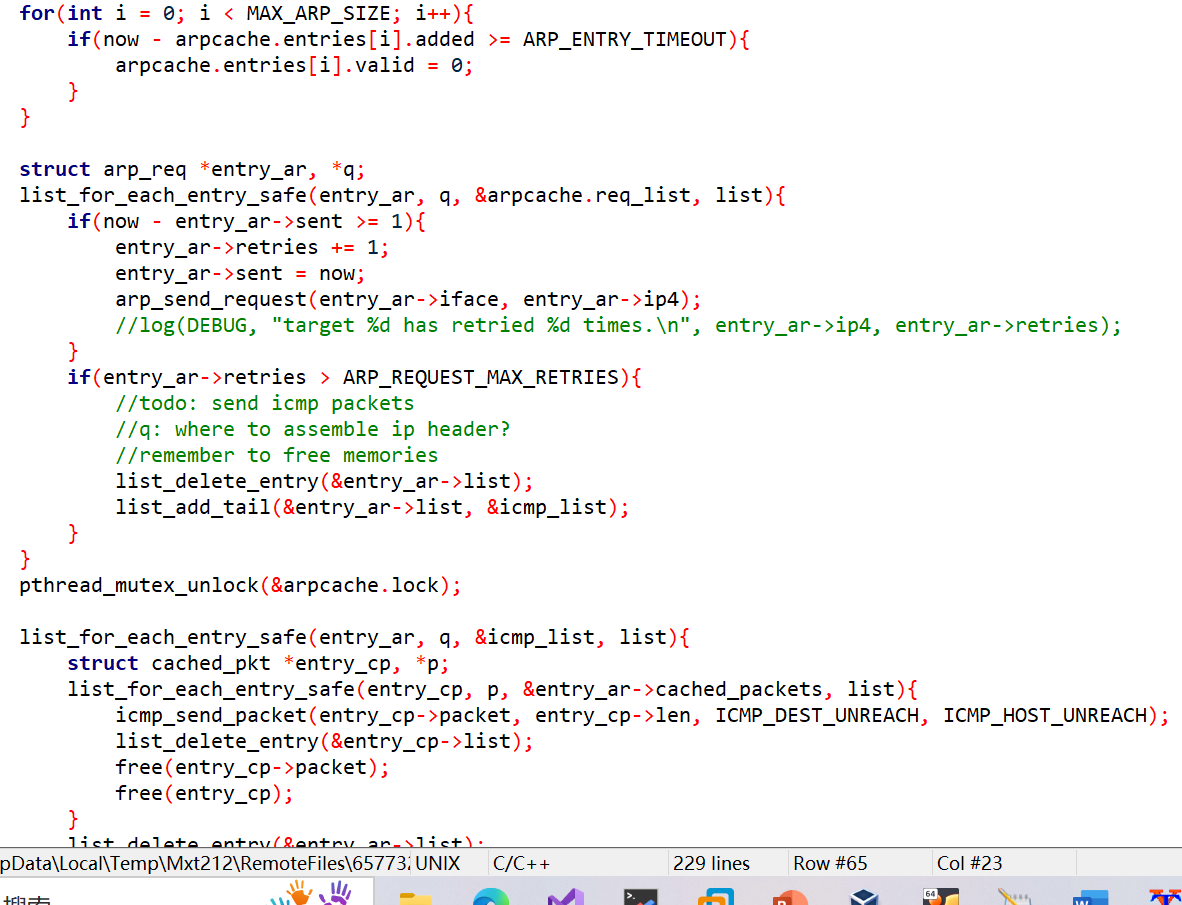


Figure 清扫线程

至此ARP协议相关的部分就处理完了。

1. IP部分

这部分也是在网络层进行考虑的，只需考虑ip协议即可。由于接下来的处理与转发过程需要多次用到最长前缀匹配，因此先对其进行实现：

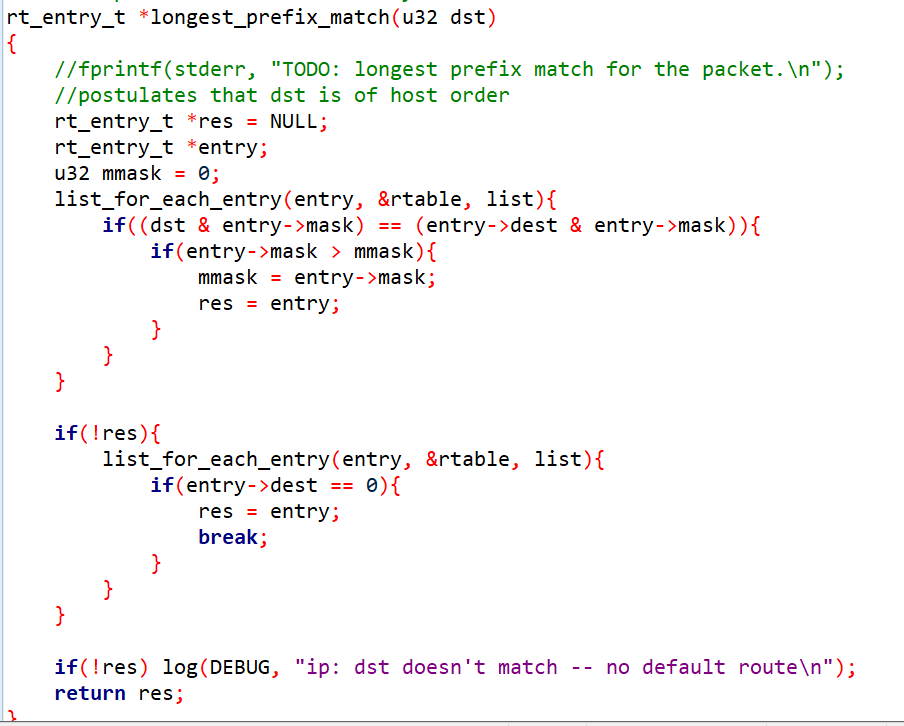


Figure 最长前缀匹配

匹配中包含了默认路由，在找不到更佳选项时就选择该项。正常地，该函数不会返回空指针。一般流程中若真的返回空指针，意味着目的网络不可达，要返回icmp通知。

接下来就实现ip部分主函数handle\_ip\_packet：

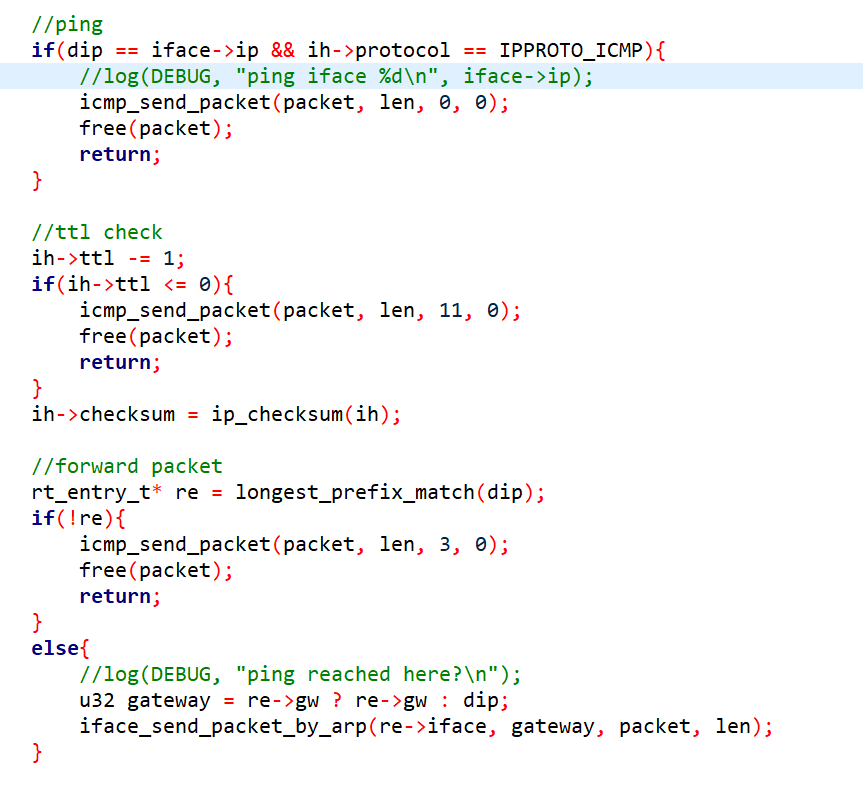


Figure 处理ip包

首先检测该包是不是ping自己的。若是，使用icmp发包进行回应。否则，对该包的ttl-1，并重新计算校验和。此处若ttl<=0也要发icmp包通知源主机，并丢弃此包。若到最后此包还不必丢弃，则按照查找到的路由表条目，用加入arp协议的发送过程进行转发。因为，这里目标mac地址尚不能填充。

1. ICMP部分

此次实验中icmp有4种情况，上文均有提及，分别是：目的主机不可达（ARP查询超时）、目的网络不可达（路由表查询返回空指针）、ttl归零（ttl<=0）以及ping响应。事实上，与icmp有关的内容基本上只体现在包的icmp部分中，那部分只要根据icmp包类型对应进行组装即可。依赖于icmp类型地，其过程大概是：计算icmp头部长度，组装icmp包的ip头，最后组装icmp头和icmp体，计算校验和并输出。代码仍然是比较冗长，此过程比较清晰，还是不放代码了。

还有一个问题需要指出：当icmp函数运行完毕后，组装完成的部分只有包的ip头和icmp头，mac头还是空的。我们在ip.c文件中的ip\_send\_packet里查路由表得到转出端口和下一跳ip后，还是使用arp发包，在arp的发包过程中会直接填充mac头源mac，目标mac通过arp查询和请求得到。

1. 实验结果

实验代码编写调试完成后，按照给定的拓扑，h1可以ping通其网关以及其余主机，在ping不存在的主机和网段时能正常返回icmp通知：

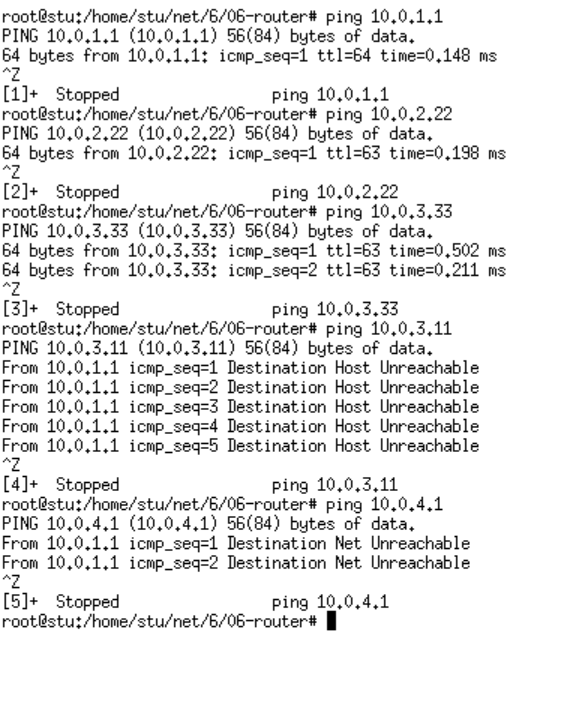


Figure 简单拓扑下的测试

然后，自建拓扑如下图：

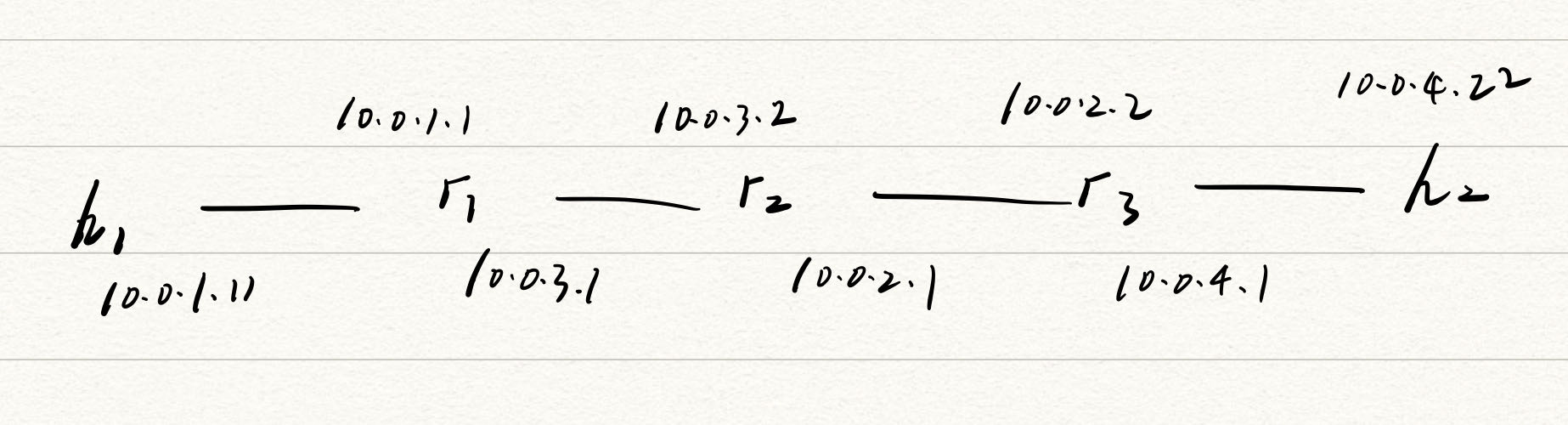


Figure 拓扑形状

修改python脚本后，h1可以ping通所有路由器入口，traceroute h2可以得到正确输出：

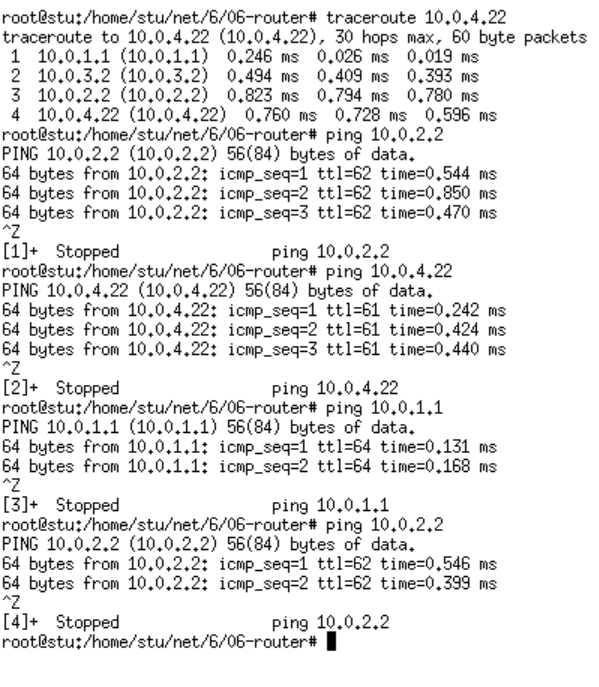


Figure 自建拓扑测试

将代码提交到oj后，测试通过。

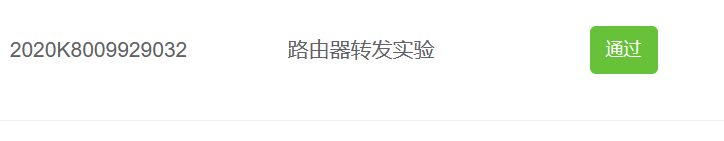


Figure 通过