Upgrades to ISEFlow: Offloading ARP and supporting IPv6

by

Joel Aaron May

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Area of Specialization: Software Systems

Program of Study Committee:
Doug Jacobson, Major Professor

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2019

TABLE OF CONTENTS

Page

ACKNOWLEDGMENTS

I would like to thank my parents, Norm and Donna, for sparking my interest in computer engineering and supporting me on my journey. I would also like to thank my sister, Sarah, who dragged me along to a campus tour of ISU, which ended up being the most influential part of my decision to pursue higher education.

ABSTRACT

This paper describes my improvements to ISEFlow, which is a part of the ISERink software suite. ISERink is the virtualization of the Internet, allowing teaching and attack testing in an isolated environment. ISEFlow is the software router at the heart of the ISERink environment. I made two improvements to ISEFlow. First, I offloaded ARP to the operating system, which resolves several problems that ISERink users have experienced under heavy utilization load. Second, I made ISEFlow able to route IPv6 packets and have IPv6 interfaces. Making ISEFlow dual stack capable allows teaching and testing for the upcoming generation of the Internet with IPv6 coexisting with IPv4.

In this paper, I document some shortcomings of my implementation of IPv6 capability and suggestions on how to resolve the problems. I have also documented some preliminary testing which shows that the ARP offloading to the operating system is working well.

CHAPTER 1. BACKGROUND OF ISEFLOW

ISEFlow is a software router designed to be a core part of a simulation of the Internet. This simulated environment, called ISERink, is designed to allow an isolated environment for testing cyber security attacks and defense without putting any larger networks at significant risk. One of the primary advantages of ISEFlow over other software routing solutions that exist today is that multiple layer 2 network hops can be simulated within one OS instance.

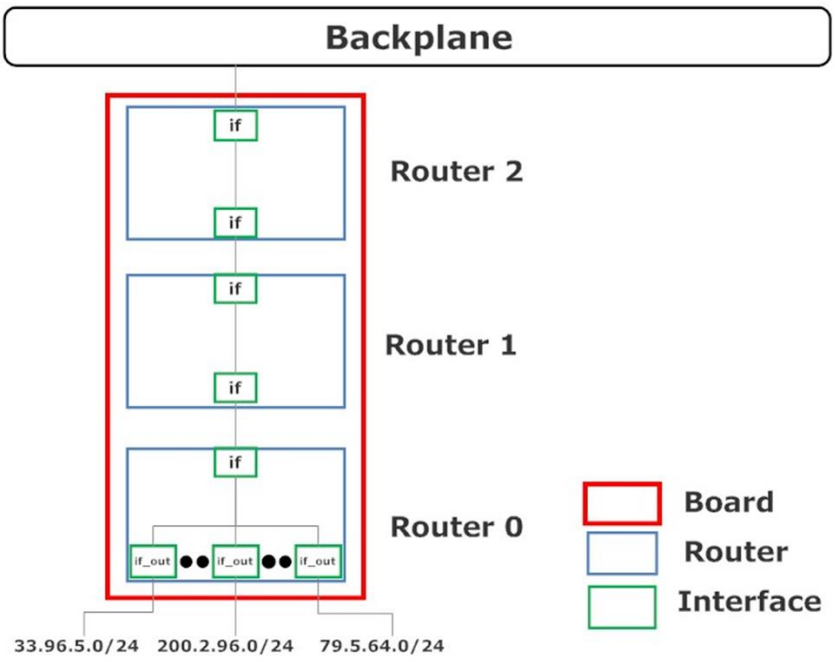

Figure 1. An ISEFlow with Multiple Internal Hops (Neel).

ISEFlow is used for the Iowa State University Cyber Defense Competitions, where students secure systems that industry professionals attempt to attack and breach on the competition day. ISEAGE currently hosts five competitions a year, many of which are open to students from other colleges and universities.

ISEFlow is also used for many computer engineering and information assurance classes. This gives students a hands-on experience that would be impossible to teach with the campus network alone due to security concerns and the limited scope of the campus network.

ISERink has a much wider reach than only the Iowa State Campus. ISERink is a core of an Iowa State University hosted playground for high schools in Iowa desiring such an environment, but who do not have the hardware and infrastructure to support it. ISERink is also distributed to other schools nationwide for running their own environments for classes and cyber defense competitions.

With how prolific ISERink and, hence, ISEFlow are, maintaining them is important. In the Iowa State University Cyber Defense Competitions, the ISEAGE lab which puts on the competition has discovered some bugs and limitations that occur during heavy load, which is difficult to replicate without the heavy competition load. The primary bug has been diagnosed as a problem in the address resolution protocol (ARP, RFC 826) implementation. While maintaining the ISEFlow code, it is a convenient time to also introduce compatibility with Internet Protocol version 6 (IPv6, RFC 8200) to modernize ISEFlow's capabilities.

CHAPTER 2. ARP OFFLOADING

Motivation

The ISEAGE lab found a couple problems with ISEFlow when being used in such a demanding and diverse situation as the Cyber Defense Competitions. These problems generally resulted in failed address resolution via ARP, leading to unreachable systems in the competition. This was especially prominent when competitors were using pfSense "virtual IPs". The use of virtual IPs has become more popular since ISEFlow was initially developed, so it wasn't as large of a concern. With the large number of IP addresses competitors have available to work with and the availability of creating as many virtual machines as they see fit, the desire for virtual IP addresses has increased greatly.

Another problem is that occasionally, under high load, ISEFlow stops resolving new addresses via ARP at all. There were attempts to fix the problem with a larger ARP table, but the problem still arose sometimes.

There was also a concern that performing ARP in userland performed worse than the optimized ARP implementation in modern operation systems. This could result as a bottleneck on packet bandwidth. There has been little effort to test this hypothesis, because the other two aforementioned reasons are enough to warrant a major change.

This left us with two choices, rewrite the ARP feature with a better implementation or offload ARP to the operating system. Since the ARP implementation was a mess according to Dr. Jacobson, we decided that offloading ARP to the operating system would be the optimal solution without requiring an overhaul.

Approach

Prior Implementation

In the existing ISEFlow implementation, the operating system had a network interface with no IP addresses assigned to it. ISEFlow would put that network interface into promiscuous mode, so ISEFlow could receive and inspect every network packet, especially the ones not destined for the network interface's MAC address as seen by the operating system. ISEFlow operated on alternative MAC addresses specified by the ISERink administrator in the configuration file. ISEFlow was able to send packets from those configured MAC addresses by crafting Ethernet frames, bypassing the operating system's OSI layer 3 to layer 2 encapsulation and MAC address.

Changes

To offload ARP to the operating system, the operating system must be using the MAC address it knows about. There was the possibility of instructing the operating system to spoof the configured MAC addresses, but for ease of implementation, that choice was declined. A simpler choice was to use the operating system's configured MAC address for all external IP addresses. With the advent of virtualization, adding more virtual network interfaces with distinct MAC addresses is free, up to a certain limit, so if an ISERink administrator desires this feature, it could be added back via supporting multiple "outside" interfaces with distinct MAC addresses.

For the operating system to send ARP requests and reply to ARP messages, the network interface must be configured in the operating system to have the desired IP addresses on the correct subnets. A side effect of this is that the operating system will also reply to some OSI layer 3 messages, such as ICMP echo request (colloquially, "ping"). Formerly, ISEFlow handled all OSI layer 3 messages, which would result in some duplicate messages being sent by both the operating system and ISEFlow. Since ISEFlow's responses were nothing significantly different

than the operating system was replying with, the decision was made to prevent ISEFlow from replying to any OSI layer 3 messages that would also be handled by the operating system.

The way ISEFlow communicated the configured IP addresses to the operating system is by executing the `ifconfig` command, instructing the operating system to add IP addresses to the network interface and update the routing table with the appropriate subnets.

The last change required for fully removing ARP from ISEFlow is the sending of packets. Formerly, ISEFlow would craft an Ethernet frame. Building Ethernet frames requires knowledge of the MAC source and destination addresses. The source MAC address is trivial, because it can be queried from the operating system. The destination address is much more complicated to resolve. It would be possible to query the operating system's ARP table to resolve the destination MAC address, if it is populated. The problem is that if the ARP table does not contain the entry, an ARP request must be sent, which is exactly what we wanted to remove from ISEFlow. The solution was to use a "raw socket" with the "IP header included" option. This allowed ISEFlow to have complete control over the OSI layer 3 (IP layer) headers and payload, without needing any knowledge of OSI layer 2 (Ethernet) information.

Layer	MAC destination	MAC source	Ethertype (Ethernet II) or length (IEEE 802.3)	Payload	Frame check sequence (32-bit CRC)
	6 octets	6 octets	2 octets	46-1500 octets	4 octets
Layer 2 Ethernet frame	← 64–1522 octets →				

Figure 2. 802.3 Ethernet Packet and Frame Structure (Wikipedia).

Analysis of Implemented Changes

The basic functionality of ISEFlow has been tested to ensure the ARP offloading to the operating system appears to be working correctly on a small scale. ICMP, TCP, and UDP traffic have been verified to fully work. Furthermore, basic TCP functionality has been confirmed to work as desired on pfSense virtual IPs and "proxy ARP" IPs.

A multiple-hour long session of sending a "ping flood" through ISEFlow showed stability with no dropped packets and a reasonable CPU load of less than 30% per board of a single core of an Intel i7-6900K CPU.

IP performance was tested with the `iperf` network performance tool. On UDP, testing showed that just over 1 Gbps throughput was able to be achieved on the i7-6900K CPU. ISEFlow was dropping some packets at that rate, and the CPU usage was 100% the single CPU core that was provisioned to each board. This CPU usage is comparable to the around 80% of a core that pfSense used to accomplish the same task. Removing routers from the testing resulted in a maximum possible UDP bandwidth of about 3.5 Gbps on the virtualization environment and hardware. A two-thirds performance hit is within the range I consider acceptable for inserting a router like this.

With `iperf` sending 104-byte UDP packets, the throughput is just under 100 Mbps, which is about one-third the speed, 255 Mbps, when running the test over a loopback interface with no external network involved, indicating reasonable performance degradation when compared to a highly optimized configuration.

TCP testing with `iperf` was inconclusive. My virtualization environment for development is Hyper-V, which is different than the officially supported ESXi environment. Under Hyper-V, TCP packet delivery is reliable with ISEFlow, but there is an incapability with

Hyper-V somewhere in ISEFlow that caused terrible TCP performance before I made any code changes. This results in TCP performance between 50 Kbps and 3 Mbps when running in Hyper-V. I hypothesize that this performance problem does not exist when running the new version of ISEFlow on the ESXi hypervisor, but I left resolving this problem outside of the scope of this project, since there's no indication that my ARP changes to ISEFlow introduced the problem or have any reason to reduce the TCP performance.

In early stages of testing, ICMP echo requests to external IP addresses resulted in duplicate ICMP echo reply packets. As discussed above, this was corrected, so only one ICMP echo reply is sent for each ICMP echo request. This behavior has been tested on "outside" addresses from the local subnet, on "outside" addresses originating from a system connected to a different ISEFlow board, and to "internal" addresses from both local and remote systems on the "outside" interface.

One drawback of the new ISEFlow configuration options is that currently only one MAC address is shared for all "outside" IP addresses. This doesn't pose a problem, but rather is a reduction of features that could be desired by ISERink administrators. If it is found that an ISERink administrator wants multiple MAC addresses, the feature may be reintroduced through adding support for multiple "outside" network interfaces on the operating system. Out of the ISERink administrators I know, none of them rely on having multiple MAC addresses, which is why this feature was counted as a reasonable loss.

CHAPTER 3. IPv6 SUPPORT

Motivation

I have a passion to spread the adoption and knowledge of IPv6. IPv6 has some distinct advantages over IPv4 that I believe are important for decent end-user network connections and more flexible enterprise-scale networks. One main advantage of IPv6 is that the need for network address translation (NAT) will effectively be removed, resulting in every device having a publicly routable address. As someone who started in technology by hosting small websites and personal services from a home and later dorm internet connection, I abhor the idea of carrier grade NAT and other effective firewalling imposed by NAT.

According to Google's web traffic, approximately twenty-five per cent of their users access their services through IPv6, which means IPv6 is a current technology, not a future speculation. Since Iowa State University is teaching future generations of developers and IT professionals, teaching IPv6 security is important for a complete education. From my experience at Iowa State University, students currently get no hands-on IPv6 experience, unless they take the initiative and choose to make their projects IPv6 capable. From talking to some professors, ISERink's support of IPv6 is a key component required for them to start teaching IPv6 with hands-on labs.

Approach

To support IPv6, ISEFlow must be able to parse the IPv6 packet header. Fortunately, this was fairly easy, because the IPv6 packet header has a fixed length. IPv6 packets are routed in a very similar way to IPv4 packets, so ISEFlow's functions for processing any OSI layer 3 packets essentially were duplicated and modified to work with the IPv6 header structure and addresses.

The correct functions were chosen by the EtherType in the Ethernet header or by the first four bits of the IP packet that indicate the IP version.

Other than the address size, the only major difference ISEFlow had to deal with is how the time to live is processed. IPv4 packets expire *immediately* when their time to live field reaches zero, but IPv6 packets expire one hop *after* their hop count field reaches zero.

Transmitting the IPv6 packets on the "backplane" network between the boards was no problem at all, because the packets are encapsulated in such a way that OSI layer 3 data is transmitted as raw data that is later reconstructed by the destination ISEFlow board.

ICMP was redesigned for IPv6, which means the ISEFlow function to reply to ICMP echo requests on IPv4 will not work on ICMP echo requests on IPv6. The ICMPv6 header structure and checksum are different than ICMP on IPv4. This required creating a function to parse ICMPv6 packets and craft a response packet, if appropriate.

Problem Sending Raw IPv6 Packets on *NIX

While implementing the function that sends out packets on the "outside" interface to machines on that network, I ran into a difference between the *NIX (Linux and Unix) packet sending APIs that prevented a straightforward solution. The difference is that IPv6 raw network sockets do not allow the "header included" option. In fact, the APIs for IPv6 raw sockets do not provide complete control over header data. For normal usage, that's not a problem, but for an accurate simulation of the Internet routers, the OSI layer 3 headers should generally be passed with no modifications outside of the hop count.

This led me to going down to crafting Ethernet frames (OSI layer 2) to regain complete control over the IPv6 header. Like with IPv4, a router must know the MAC address of the destination system, which brought up the need for the exact feature I removed for IPv4 earlier in the project, ARP. In IPv6, the ARP protocol has been superseded by the neighbor discovery

protocol, but the same functionality was needed of discovering the MAC address belonging to the destination IP address. Implementing neighbor discovery would likely result in the same problems we had with ARP.

The path I chose is suboptimal, but gets the job done. In future work on ISEFlow, I recommend this design be rearchitected in a more robust way. My solution was for ISEFlow to query the operating system's neighbor table when trying to route an IPv6 packet to a local network. With the information retrieved there, it is possible to craft an Ethernet frame containing the IPv6 packet with the ability to pass all IPv6 headers transparently. If the neighbor table does not contain the desired address, ISEFlow sends an empty IPv6 packet from its own IP address on the target network. This forces the operating system to initiate neighbor discovery and populate the neighbor table, if there is a reply. Since neighbor discovery is not instant, ISEFlow simply drops that packet, letting the sending device choose to retransmit later after the neighbor table hopefully contains the desired entry.

Analysis of Implemented Changes

The first test is the functionality test. Can it route ICMPv6, TCP, and UDP traffic without dropping excessive packets? A ping flood is routed through ISERink works with no dropped packets. An `iperf` UDP test results in about 8 Mbps bandwidth on the i7-6900K CPU. An `iperf` TCP test resulted in about 8 Mbps, outperforming the IPv4 TCP test. I also tested HTTP via IPv6 for a simple webpage and found that it worked.

While running the `iperf` tests, I did find that some packets were dropped as incompatible, when they should be able to be routed. During development, ICMPv6 and HTTP were used to test functionality, so these problems discovered while using `iperf` have not been pinpointed yet.

The conclusion is that most IPv6 traffic is passed through ISEFlow correctly, but there is a bug that leads to some specific types of IPv6 packets being dropped.

Further Work Suggested

More Efficient Neighbor Discovery

The benchmarks show that the IPv6 throughput is quite lacking. The bottleneck is presumed to be that to route each packet, ISEFlow runs an instance of the `ndp` command. Spawning a new process for every IPv6 outbound packet entails significant overhead. The other parts of the IPv6 routing are similar enough to the IPv4 routing, so it is highly unlikely that the bottleneck is elsewhere.

The most straightforward way to improve this bottleneck is by querying the operating system's neighbor table without spawning a process. Since the Unix `ndp` tool is open source, it should be possible to integrate querying the neighbor table directly into ISEFlow without an external tool.

Store and Forward Neighbor Discovery

When a packet must be routed to a destination that is not in the neighbor table, ISEFlow currently drops the packet on the assumption that the sender will retransmit the packet eventually. This gives the operating system time to populate the neighbor table by the packet ISEFlow sent out from its own address.

A better design would be to have a buffer to store packets that are waiting on neighbor discovery that will be held for a period of time and retried when the operating system is expected to have a neighbor table entry for the destination system.

Request a Neighbor Discovery Without an Empty Packet

I would suggest looking into the possibility of requesting neighbor discovery from the operating system without the need for sending out an empty packet. Perhaps it would be

possible for ISEFlow to send out the initial neighbor discovery packet, but allow the operating system to receive the neighbor advertisement, despite not asking for it.

ICMPv6 Replies

Some non-critical parts of the IPv6 specification were omitted in this first implementation of IPv6 capability. For full compliance with the IPv6 specification, ISEFlow must create ICMPv6 response packets on certain failures, including hop limit exceeded and destination unreachable.

CHAPTER 4. CHANGES TO USAGE

ARP Changes

An ISERink administrator must make a few changes to upgrade from a previous version of ISERink with inbuilt ARP functionality. The first change is removing all MAC address fields from the ISERink config file. The configuration file has a MAC address field in the `if_out` and `if` commands. The `if` command's MAC field may be empty in existing configuration files. The new configuration must be compiled with the updated `mk_map` tool included with ISEFlow.

```
…
if_out=0,00:00:0c:31:01:aa,64.39.3.254,64.39.3.0,24
if=0,64.39.3.254,
…
```

Figure 3. ISERink Configuration Changes for ARP Offloading.

The ISERink administrator may now choose to turn off the hypervisor option to allow MAC address spoofing and/or promiscuous mode, as it is no longer necessary.

IPv6 Changes

The IPv6-capable ISEFlow is completely backward compatible with unmodified IPv4-only ISERink configuration files. To allow use of the new IPv6 configuration options, an IPv6 command has been introduced for each command that takes an IPv4 address. These commands are `if_out`, `if`, and `r_table`. The equivalent commands are `if6_out`, `if6`, and `r_table6`. The syntax and usage are exactly the same as the IPv4 commands, other than accepting IPv6 addresses and IPv6 subnet sizes.

```
…
if_out=0,64.39.3.254,64.39.3.0,24
if6_out=0,1::ffff,1::0,64
if=0,64.39.3.254
if6=0,1::ffff
# Dest IP, mask, type, next IP, next board, next router, next int
r_table=64.39.3.0,24,outside,64.39.3.254,1,0,0
r_table=0.0.0.0,24,internal,130.170.1.254,1,1,0
r_table6=1::0,64,outside,1::ffff,1,0,0
r_table6=0::0,0,inside,2000::4,4,0,1
…
```

Figure 4. Example Usage of New IPv6 ISERink Configuration Commands.

REFERENCES

RFC 826, “An Ethernet Address Resolution Protocol”, https://tools.ietf.org/html/rfc826

RFC 8200, “Internet Protocol, Version 6 (IPv6) Specification”, https://tools.ietf.org/html/rfc8200

Neel, Jeffrey Lincoln, “ISEConf: A management tool for the ISEAGE Environment to allow users to create a custom mock internet for testbed research”, https://doi.org/10.31274/etd-180810-5206

Wikipedia, “Ethernet frame”, https://en.wikipedia.org/wiki/Ethernet_frame

APPENDIX. EXAMPLE CONFIG FILE COMPATIBLE WITH UPDATED ISEFLOW

```
#this is a test file to be used by ISEFlow
board=1
router=0
type=outside
name=Outside router 1
if_out=0,64.39.3.254,64.39.3.0,24
if_out=1,33.96.5.254,33.96.5.0,24
if6_out=0,1::ffff,1::0,64
if=0,64.39.3.254
if=1,33.96.5.254
if=2,130.170.1.100
if6=0,1::ffff
if6=1,2000::1
# Dest IP, mask, type, next IP, next board, next router, next int
r_table=64.39.3.0,24,outside,64.39.3.254,1,0,0
r_table=33.96.5.0,24,outside,33.96.5.254,1,0,1
r_table=0.0.0.0,24,internal,130.170.1.254,1,1,0
r_table6=1::0,64,outside,1::ffff,1,0,0
r_table6=0::0,0,inside,2000::4,4,0,1
router=1
type=internal
name=b1r1
if=0,130.170.1.254
if=1,131.10.1.100
#To outside
r_table=64.39.3.0,24,internal,64.39.3.254,1,0,0
r_table=33.96.5.0,24,internal,33.96.5.254,1,0,1
#default router 2
r_table=0.0.0.0,24,internal,131.10.1.254,1,2,0
router=2
type=internal
name=b1r2
if=0,131.10.1.254
if=1,130.175.1.1
#Toward outside
r_table=64.39.3.0,24,internal,131.10.1.100,1,1,1
r_table=33.96.5.0,24,internal,131.10.1.100,1,1,1
r_table=130.170.1.0,24,internal,131.10.1.100,1,1,1
#proxy board 4 router 1
r_table=199.100.16.0,24,inside,130.175.1.4,4,0,1
#default board 5 router 1
r_table=0.0.0.0,24,inside,130.175.1.5,5,0,1
```

```
board=2
router=0
type=outside
name=Private Network range
if_out=0,201.203.200.254,201.203.200.0,24
if_out=1,108.64.203.254,108.64.203.0,24
if=0,201.203.200.254
if=1,108.64.203.254
if=2,130.170.2.100
r_table=201.203.200.0,24,outside,201.203.200.254,2,0,0
r_table=108.64.203.0,24,outside,108.64.203.254,2,0,1
r_table=0.0.0.0,24,internal,130.170.2.254,2,1,0
router=1
type=internal
name=b2r1
if=0,130.170.2.254
if=1,131.10.2.100
#To outside
r_table=201.203.200.0,24,internal,201.203.200.254,2,0,0
r_table=108.64.203.0,24,internal,108.64.203.254,2,0,0
#default router 2
r_table=0.0.0.0,24,internal,131.10.2.254,2,2,0
router=2
type=internal
name=b2r2
if=0,131.10.2.254
if=1,130.175.1.2
#Toward outside
r_table=201.203.200.0,24,internal,131.10.2.100,2,1,1
r_table=108.64.203.0,24,internal,131.10.2.100,2,1,1
r_table=130.170.1.0,24,internal,131.10.2.100,2,1,1
#proxy board 4 router 1
r_table=199.100.16.0,24,inside,130.175.1.4,4,0,1
#default board 5 router 1
r_table=0.0.0.0,24,inside,130.175.1.5,5,0,1

board=3
router=0
type=outside
name=Private Network range
if_out=0,104.190.101.254,104.190.101.0,24
if=0,104.190.101.254
if=1,130.170.3.100
r_table=104.190.101.0,24,outside,104.190.101.254,3,0,0
```

```
r_table=0.0.0.0,24,internal,131.10.3.254,3,1,0
router=1
type=internal
name=b2r1
if=0,130.170.3.254
if=1,130.175.1.3
#Toward outside
r_table=104.190.101.0,24,internal,130.170.3.100,3,0,1
r_table=130.170.1.0,24,internal,130.170.3.100,3,0,1
#proxy board 4 router 1
r_table=199.100.16.0,24,inside,130.175.1.4,4,0,1
#default board 5 router 1
r_table=0.0.0.0,24,inside,130.175.1.5,5,0,1

board=4
router=0
type=outside
name=Proxy router
if_out=0,199.100.16.254,199.100.16.0,24
if6_out=0,4::ffff,4::0,64
if=0,199.100.16.254
if=1,130.175.1.4
if6=0,4::ffff
if6=1,2000::4
r_table=199.100.16.0,24,outside,199.100.16.254,4,0,0
r_table6=4::0,64,outside,4::ffff,4,0,0
#routes to board 1
r_table=64.39.3.0,24,inside,130.175.1.1,1,2,1
r_table=33.96.5.0,24,inside,130.175.1.1,1,2,1
r_table=130.170.1.0,24,inside,130.175.1.1,1,2,1
r_table=131.10.1.0,24,inside,130.175.1.1,1,2,1
r_table6=1::0,64,inside,2000::1,1,0,1
#route to team 2
r_table=201.203.200.0,24,inside,130.175.1.2,2,2,1
r_table=108.64.203.0,24,inside,130.175.1.2,2,2,1
r_table=130.170.2.0,24,inside,130.175.1.2,2,2,1
r_table=131.10.2.0,24,inside,130.175.1.2,2,2,1
#route to board 3
r_table=104.190.101.0,24,inside,130.175.1.3,3,1,1
r_table=131.10.3.0,24,inside,130.175.1.3,3,1,1
#default board 5 router 0
r_table=0.0.0.0,24,inside,130.175.1.5,5,0,1

board=5
tap=1
```

```
router=0
type=outside
name=Default router
if_out=0,10.0.0.254,10.0.0.0,24
if=0,10.0.0.254
if=1,130.175.1.5
#routes to board 1
r_table=64.39.3.0,24,inside,130.175.1.1,1,2,1
r_table=33.96.5.0,24,inside,130.175.1.1,1,2,1
r_table=130.170.1.0,24,inside,130.175.1.1,1,2,1
r_table=131.10.1.0,24,inside,130.175.1.1,1,2,1
#routes to board 2
r_table=201.203.200.0,24,inside,130.175.1.2,2,2,1
r_table=108.64.203.0,24,inside,130.175.1.2,2,2,1
r_table=130.170.2.0,24,inside,130.175.1.2,2,2,1
r_table=131.10.2.0,24,inside,130.175.1.2,2,2,1
#route to board 3
r_table=104.190.101.0,24,inside,130.175.1.3,3,1,1
r_table=131.10.3.0,24,inside,130.175.1.3,3,1,1
#proxy board 4 router 0
r_table=199.100.16.0,24,inside,130.175.1.4,4,0,1
#default board 5 router 0
#r_table=0.0.0.0,24,inside,130.175.1.5,5,0,1
r_table=0.0.0.0,24,inside,0.0.0.0,5,0,0
```