James Madison University

Integrated Science & Technology (ISAT)

ISAT/CS 465 Wireless Networking, Security, & Forensics

Implementation of Babel Protocol and Hybrid Network Architectures

Author(s):
Troy Gamboa
Isaac Sumner

Submitted to:
Dr. Emil Salib

May 3, 2017

Honor Pledge: I have neither given nor received help on his lab that violates the spirit of the JMU Honor Code.

Troy Gamboa

Signature

Date

Isaac Sumner

Signature

Date

Contents

1 Introduction . . . 6
2 Network Diagrams & Tables . . . 7
2.1 Exercise 1 - Network Diagrams & Tables . . . 7
2.2 Exercise 2 - Network Diagrams & Tables . . . 8
2.3 Exercise 3 - Network Diagrams & Tables . . . 9
2.4 Exercise 4 - Network Diagrams & Tables . . . 10
3 Lab Exercises : Results & Analysis . . . 11
3.1 Exercise 1: Setting up Babel with a basic topology . . . 11
3.1.1 Analysis & Evidence . . . 11
3.1.1.1 Step 0: Exercise 1 - Network Diagrams & Tables . . . 11
3.1.1.2 Step 0: Draw your network diagram and fill out your network information table . . . 11
3.1.1.3 Step 0: Preparation . . . 11
3.1.1.4 Step 1: Babel Configuration . . . 11
3.1.1.5 Step 2: Babel Redistribution . . . 13
3.1.2 Key Learning/Takeaways: . . . 13
3.2 Exercise 2: Setting Up a Mesh Network & Convergence Times . . . 14
3.2.1 Analysis & Evidence . . . 14
3.2.1.1 Step 0: Exercise 2 - Network Diagrams & Tables . . . 14
3.2.1.2 Step 0: Preparation . . . 14
3.2.1.3 Step 1: Convergence Time in Comparison to RIP . . . 14
3.2.1.4 Step 2: Routing Tables Affecting Traceroutes . . . 16
3.2.1.5 Step 3: Compare and Contrasting RIP and Babel . . . 17
3.2.2 Key Learning/Takeaways: . . . 18
3.3 Exercise 3: Configuring OpenWrt and Setting up a Babel Network on Physical routers . . . 19
3.3.1 Analysis & Evidence . . . 19

3.3.1.1 Step 0: Exercise 3 - Network Diagrams & Tables . . . 19
3.3.1.2 Step 1: Construct Your Network 19
3.3.1.3 Step 2: Configuring a Babel Network Topology on multiple physical routers . 25
3.3.2 Key Learning/Takeaways: . . . 28
3.4 Exercise 4: Babel/Rip Hybrid Mesh Architecture with CORE . . . 30
3.4.1 Analysis & Evidence . . . 30
3.4.1.1 Step 0: Preparation . . . 30
3.4.1.2 Step 1: Setting Up Babel Enabled Wired Mesh Network Using C.O.R.E. 30
3.4.1.3 Step 2: Setting up a Wireless Mesh Network Using C.O.R.E. . . . 34
3.4.1.4 Step 3: Adding A RIP network to the Babel Configuration . . . 37
3.4.1.5 Step 4: Wireless Mesh and Link Quality Routing . . . 42
3.4.2 Key Learning/Takeaways: . . . 45
3.5 Exercise 5: Security Issues with Babel . . . 46
3.5.1 Analysis & Evidence . . . 46
3.5.1.1 Step 0: Exercise 5 - Network Diagrams & Tables . . . 46
3.5.1.2 Step 1: . . . 46
3.5.2 Key Learning/Takeaways: . . . 46
4 Lab List of Attachments & Additional Questions . . . 47
4.1 List of Attachments . . . 47
4.2 Additional Questions . . . 47
5 Lab Observations, Suggestions & Best Practices . . . 47
5.1 Observations . . . 47
5.1.0.1 OpenWrt . . . 47
5.1.0.2 CORE . . . 48
5.2 Suggestions . . . 48
5.3 Best Practices . . . 48
6 Lab References . . . 48
7 Acknowledgments . . . 48
8 Lab Extra Credit Exercises . . . 49
8.1 Extra Credit 1: . . . 49
8.2 Extra Credit 2: . . . 49
9 Appendices . . . 50

List of Figures

1 Network Topology For Exercise 1 7
2 Final Network Topology For Exercise 2 8
3 Final Network Topology For Exercise 3 10
4 Final Network Diagram For Exercise 4 10
5 GNS3 Network for Exercise 1 11
6 Starting the Babel daemon 11
7 Capturing Babel packets 12
8 Breakdown of Babel Update 12
9 Successfully pinging babel2 13
10 Telling Babel to redistribute routes 14
11 Capture between babel1 and babel3 with link between babel1 and babel4 down 15
12 Capture between babel2 and babel4 with link between babel1 and babel4 down 15
13 Capture between babel1 and babel4 after bringing interface back up 16
14 Traceroute with link up 16
15 Traceroute with link down 16
16 Wireshark Captures 17
17 Configuring password / ssh 19
18 Configuring Static IP 20
19 Configuring wireless mesh interface with 802.11s 21
20 SSH proof to 192.168.11.1 OpenWrt Router 22
21 Installing and Enabling the Babel Daemon 22
22 Editing the firewall configuration 23
23 Editing the wireless configuration 23
24 Reloading the firewall and network configurations 24
25 Checking the inital routing table 24
26 Preliminary test of the babel daemon 25
27 Starting inital packet capture 26
28 Proof of neighbor messages in the babel daemon debug 26
29 Proof of connectivity between 192.168.11.1 and 192.168.12.1 nodes 26
30 Verifying the Associated Stations in the OpenWrt GUI. 27
31 Checking the routing table first as Babel is running, and then after the babel daemon is stopped. 27

32 Example of finalized wireshark capture 28
33 Wired Babel Mesh Network 30
34 main configuration page 31
35 Enable Service configuration 32
36 Babel daemon start up command 32
37 Babel configuration, redistribute metric 33
38 Babel startup command pointing to configuration file 34
39 Wireless Lan Configuration Tab 35
40 Wireless Lan EMANE Configuration Tab 35
41 Successful connectivity between wireless network and wired . . . 36
42 Added Wireless Messh Network 37
43 Shows the RIP configuration for n12 38
44 Shows the RIP configuration for n13 39
45 Shows the RIP configuration for n15 39
46 Show IP route on n12 40
47 Show IP route on n15 41
48 Succesful ping from host on RIP network to host on Babel Wired Mesh Network 42
49 Starting routing entry on N12 for network 10.0.5.0/24 43
50 Starting routing entry on N12 for network 10.0.5.0/24 43
51 After moving nodes around route entry on N12 for network 10.0.5.0/24 changed 44
52 Src IP switching during ICMP request 44

List of Tables

1 Network Information Table For Exercise 1 8
2 Final Network Information Table For Exercise 2 9

1 Introduction

The babel routing protocol is a distance-vector protocol in which was designed with the RIP protocol, with some additional features. It is widely known as "Speedy RIP", and is primarily a loop-avoiding protocol that was originally designed for wireless ad-hoc networks. This being said, babel is still viable and stable in wired networks, as well as hybrid networks consisting of wireless and wired nodes.

Babel limits the frequency and duration of routing paths when a path is lost or dropped. Based on this capability, Babel is very efficient at reconverging to another path in such a scenario. The is done using a technique called DSDV, or Destination Sequenced Distance-Vector" routing, based on the Bellman-Ford algorithm.

Being a "double-stack" routing protocol, Babel is supported in both IPv4 and IPv6 networks. In addition to this, Babel can automatically detect wireless and wired interfaces and adjust accordingly.

In this lab / Semester Project, We covered 5 main exercises. These being:

1. Preliminary Network Topology and Babel installation of Ubuntu Server VMs using GNS3
2. Settings up a Mesh Network Architecture and Convergence Time Investigation
3. Configuring OpenWRT and Setting up a Babel Network on Physical Routers
4. Creation of a Babel Enabled Network Hybrid Network Architecture.
5. Security Issues with Babel

2 Network Diagrams & Tables

2.1 Exercise 1 - Network Diagrams & Tables

Figure 1 shows the network configurations constructed and exercised in Exercise 1.

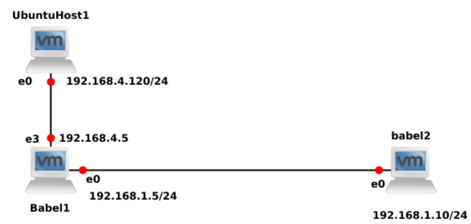


Figure 1: Network Topology For Exercise 1

Table 1 provides the network interfaces information for the hosts, switches shown in Figure 1

Name	interface	IP Address
UbuntuHost1	eth0	192.168.4.120
	eth0	192.168.1.5
Babel1	eth3	192.168.4.5
Babel2	eth0	192.168.1.10

Table 1: Network Information Table For Exercise 1

2.2 Exercise 2 - Network Diagrams & Tables

Figure 2 shows the network configurations constructed and exercised in Exercise 2.

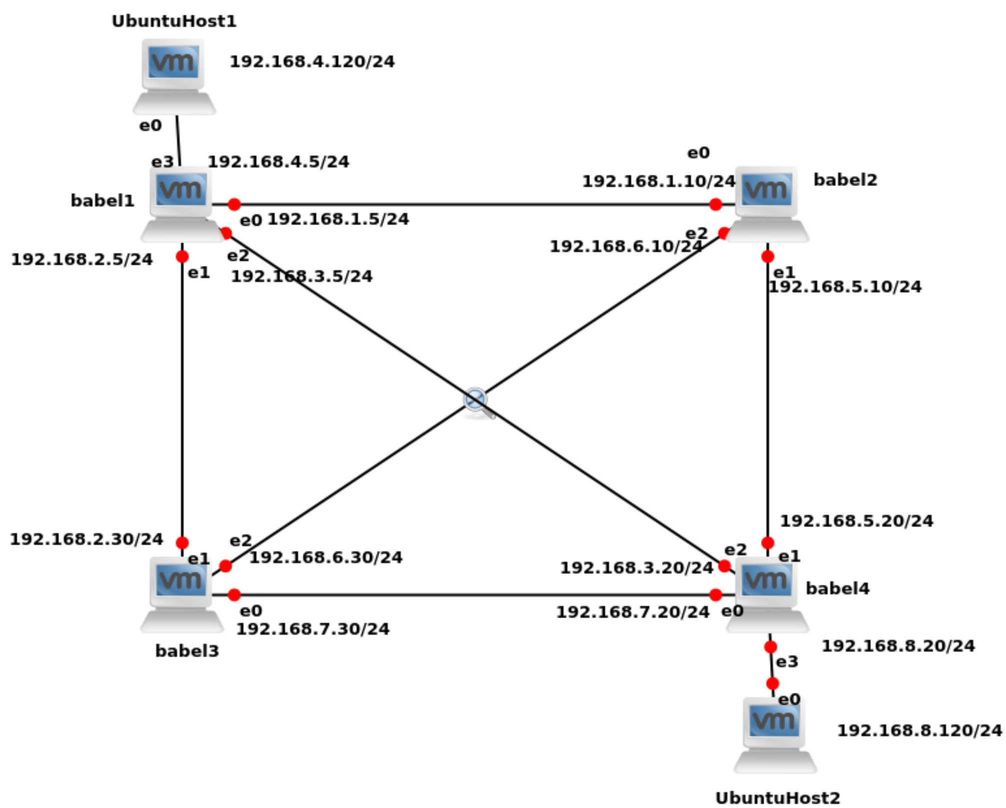


Figure 2: Final Network Topology For Exercise 2

Table 2 provides the network interfaces information for the hosts, switches shown in Figure 2

Name	interface	IP Address
UbuntuHost1	eth0	192.168.4.120
	eth0	192.168.1.5
Babel1	eth3	192.168.4.5
	eth0	192.168.1.10
	eth1	192.168.5.10
Babel2	eth2	192.168.6.10
	eth0	192.168.7.30
	eth1	192.168.2.30
Babel3	eth2	192.168.6.30
	eth0	192.168.7.20
	eth1	192.168.5.20
	eth2	192.168.3.20
Babel4	eth3	192.168.8.20
UbuntuHost2	eth0	192.168.8.120

Table 2: Final Network Information Table For Exercise 2

2.3 Exercise 3 - Network Diagrams & Tables

Figure 3 shows the network configurations constructed and exercised in Exercise 3.

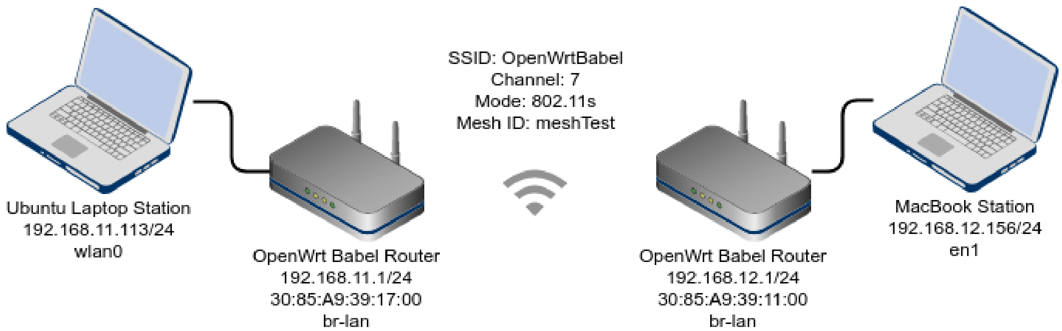


Figure 3: Final Network Topology For Exercise 3

2.4 Exercise 4 - Network Diagrams & Tables

Figure 4 shows the network configurations constructed and exercised in Exercise 4.

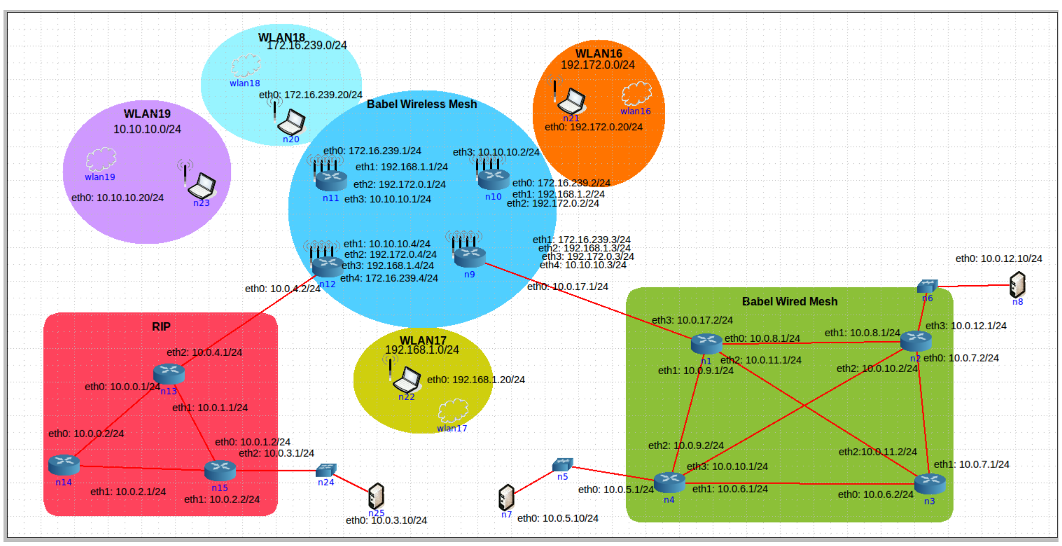


Figure 4: Final Network Diagram For Exercise 4

3 Lab Exercises : Results & Analysis

3.1 Exercise 1: Setting up Babel with a basic topology

3.1.1 Analysis & Evidence

3.1.1.1 Step 0: Exercise 1 - Network Diagrams & Tables

3.1.1.2 Step 0: Draw your network diagram and fill out your network information table

See Figure 1 and Table 1 in Section 2

3.1.1.3 Step 0: Preparation

To begin this exercise, we launched GNS3 and created the network topology shown in figure: 6

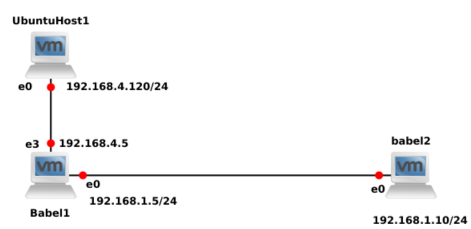


Figure 5: GNS3 Network for Exercise 1

We also created several VMnets in VMware to accommodate the interfaces.

Q1: What does the "-d1" option do?

A1: It sets the debug level

3.1.1.4 Step 1: Babel Configuration

We begin this step by starting the Babel daemon on babel1 and babel2.

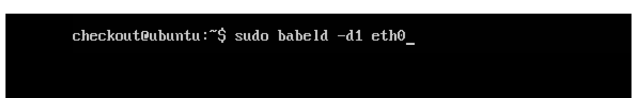


Figure 6: Starting the Babel daemon

Q2: What does the "-d1" option do?

A2: It sets the debug level

Next, we started capturing on the link between babel1 and babel2.

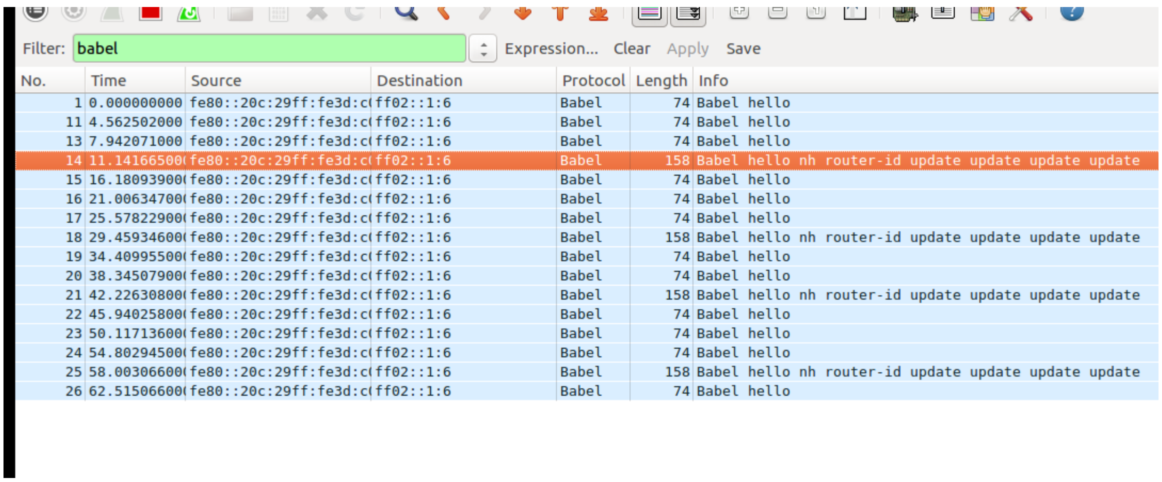


Figure 7: Capturing Babel packets

Q3: What is the destination address shown for the Babel packets? What is its significance?

A3: ff02::1:6, It is reserved for IPv6 multicasting.

Q4: What does the "ihu" shown in some of the packets stand for? What does "nh" stand for?

A4: "I hear you" and "next hop"

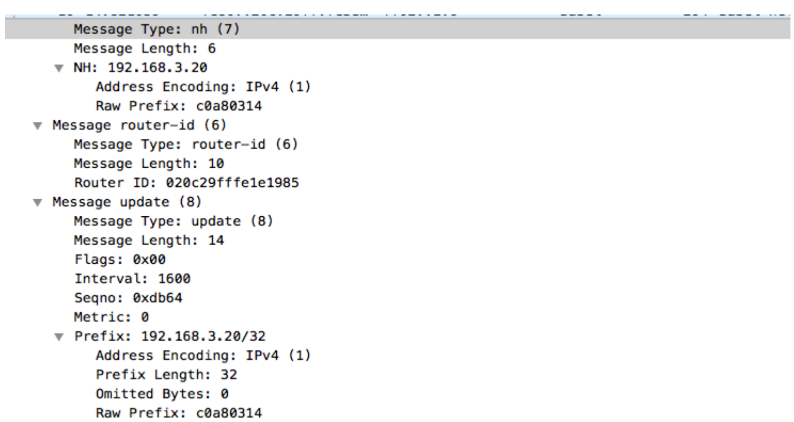


Figure 8: Breakdown of Babel Update

In Figure: 8 we see the structure of a Babel update. NH is the next hop address for all of the networks being advertised. Router-id is how Babel identifies it's neighbors, it is a portion of the IPv6 address. The update includes information like the update interval time and most importantly the network prefix.

3.1.1.5 Step 2: Babel Redistribution

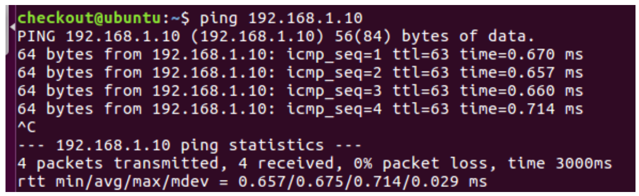
We begin by trying (unsuccessfully) to ping babel2 from the UbuntuDesktop1.

Q5: Why can't the Ubuntu Desktop ping babel2? (Hint: Wireshark)

A5: babel1 is not sending updates to babel2 with the 4.0 network

Next, we restarted the Babel daemon on babel1 with the following command
`sudo babeld -d1 -C "redistribute metric 128" eth0`

Then, we are able to successfully ping babel2 from the UbuntuDesktop!



```
checkout@ubuntu:~$ ping 192.168.1.10
PING 192.168.1.10 (192.168.1.10) 56(84) bytes of data.
64 bytes from 192.168.1.10: icmp_seq=1 ttl=63 time=0.670 ms
64 bytes from 192.168.1.10: icmp_seq=2 ttl=63 time=0.657 ms
64 bytes from 192.168.1.10: icmp_seq=3 ttl=63 time=0.660 ms
64 bytes from 192.168.1.10: icmp_seq=4 ttl=63 time=0.714 ms
^C
--- 192.168.1.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.657/0.675/0.714/0.029 ms
```

Figure 9: Successfully pinging babel2

Q6: Check Wireshark and find out what's new in the Babel packets.

A6: The updates from babel1 containing the 4.0 network

3.1.2 Key Learning/Takeaways:

In this exercise we learned how quick and easy it is to set up the babel routing protocol. It's easy to see how this would come in handy in an ad-hoc or mesh wireless setting. It takes a lot more effort to do the same thing with RIP and OSPF (as for as the number of commands necessary goes).

3.2 Exercise 2: Setting Up a Mesh Network & Convergence Times

3.2.1 Analysis & Evidence

3.2.1.1 Step 0: Exercise 2 - Network Diagrams & Tables

See Figure 2 and Table 2 in Section 2

3.2.1.2 Step 0: Preparation

We begin by setting up the topology shown in Figure: **??**. We did this by creating additional linked clones of the Ubuntu Server VM and 1 more clone of the Ubuntu Desktop VM. Next, we set up each of the VMs with the specified number of interfaces using their /etc/network/interfaces files. When our topology was fully configured, we verified that we had the Babel daemon installed on each of the routers using the `babeld --version` command. We also verified that the hosts could ping their gateways.

Q1: How long does it take for the network to converge to the new route after bringing eth2 down?

A1: 27 Seconds

3.2.1.3 Step 1: Convergence Time in Comparison to RIP

We began this step by executing the following command on babel1, shown in Figure: 10

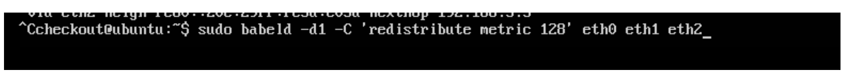


Figure 10: Telling Babel to redistribute routes

We then repeated this command on Babel4, and started Babel on babel2 and babel3 without the `-C 'redistribute metric 128'` configuration.

Next, we started pinging UbuntuHost2 from UbuntuHost1. Then, we brought down eth2 on babel4. We used the active ping to see how long it took Babel to converge.

Q2: How long does it take for the network to converge to the new route after bringing eth2 down?

A2: 27 Seconds

Q3: How does the convergence time compare to RIP from Lab 2?

A3: It is quicker

Then, we started capturing on the links between babel1 and babels 2, 3, and 4. We then brought eth2 on babel4 back up and observed the captures.

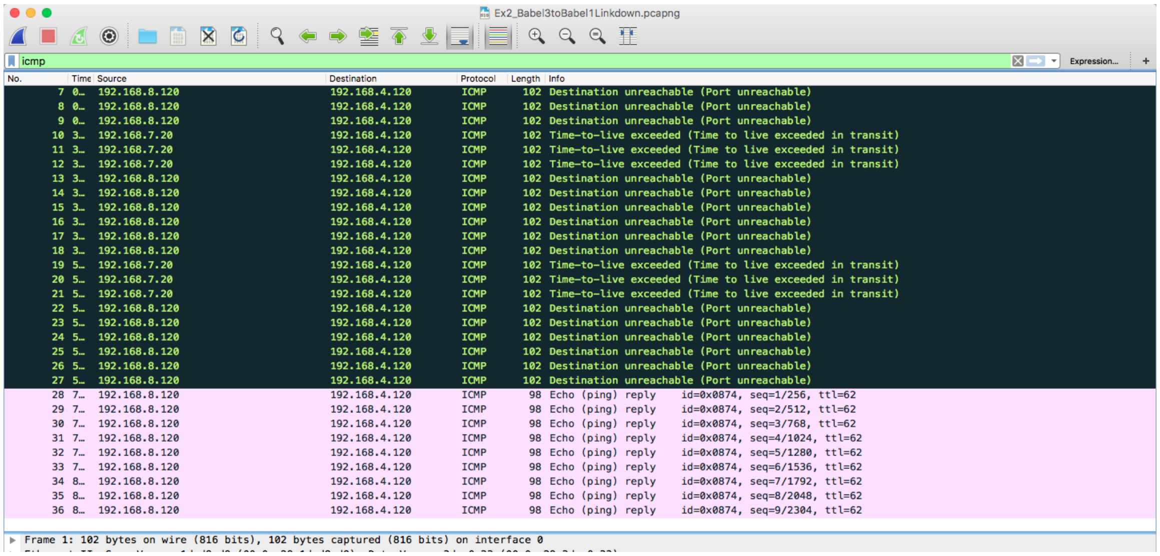


Figure 11: Capture between babel1 and babel3 with link between babel1 and babel4 down

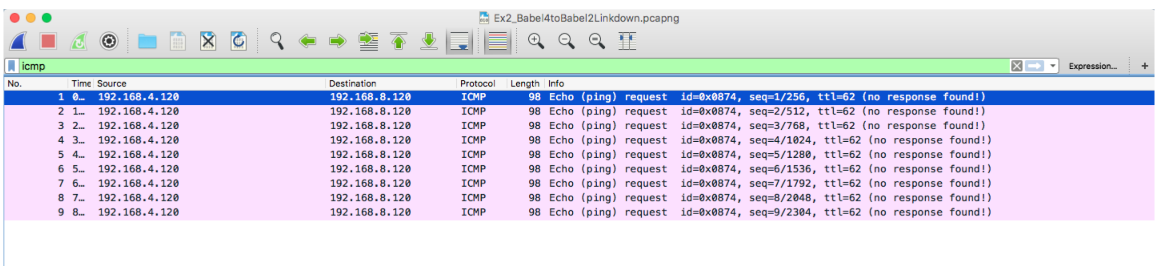
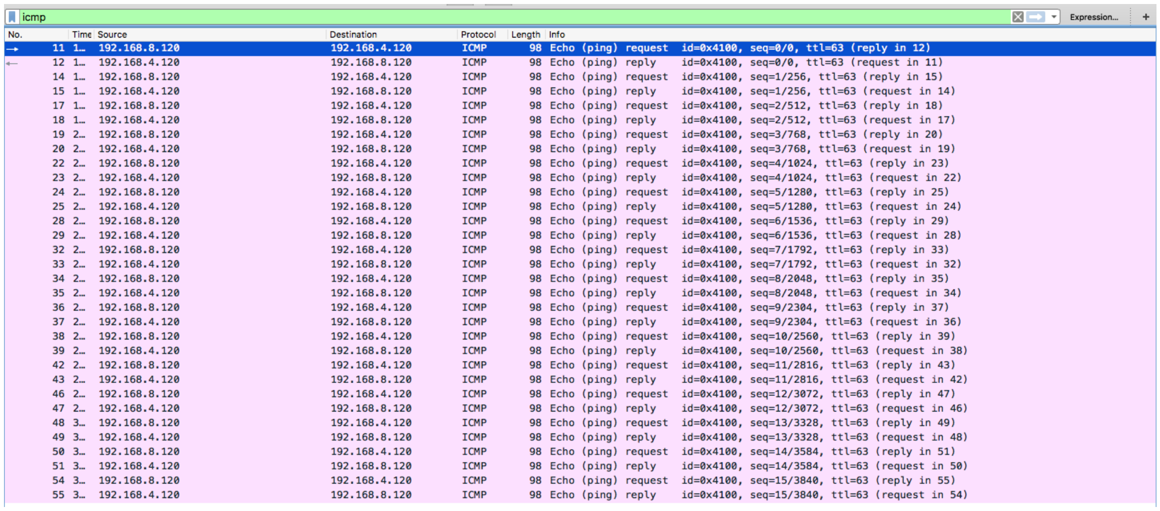


Figure 12: Capture between babel2 and babel4 with link between babel1 and babel4 down

At first all the requests are being sent to babel2 and all the replies are being sent to babel3, but after bringing eth2 on babel4 back up, we see both requests and replies in the 3.0 network between babel1 and babel4.



No.	Time	Source	Destination	Protocol	Length	Info
11	1...	192.168.8.120	192.168.4.120	ICMP	98	Echo (ping) request id=0x4100, seq=0/0, ttl=63 (reply in 12)
12	1...	192.168.4.120	192.168.8.120	ICMP	98	Echo (ping) reply id=0x4100, seq=0/0, ttl=63 (request in 11)
14	1...	192.168.8.120	192.168.4.120	ICMP	98	Echo (ping) request id=0x4100, seq=1/256, ttl=63 (reply in 15)
15	1...	192.168.4.120	192.168.8.120	ICMP	98	Echo (ping) reply id=0x4100, seq=1/256, ttl=63 (request in 14)
17	1...	192.168.8.120	192.168.4.120	ICMP	98	Echo (ping) request id=0x4100, seq=2/512, ttl=63 (reply in 18)
18	1...	192.168.4.120	192.168.8.120	ICMP	98	Echo (ping) reply id=0x4100, seq=2/512, ttl=63 (request in 17)
19	2...	192.168.8.120	192.168.4.120	ICMP	98	Echo (ping) request id=0x4100, seq=3/768, ttl=63 (reply in 20)
20	2...	192.168.4.120	192.168.8.120	ICMP	98	Echo (ping) reply id=0x4100, seq=3/768, ttl=63 (request in 19)
22	2...	192.168.8.120	192.168.4.120	ICMP	98	Echo (ping) request id=0x4100, seq=4/1024, ttl=63 (reply in 23)
23	2...	192.168.4.120	192.168.8.120	ICMP	98	Echo (ping) reply id=0x4100, seq=4/1024, ttl=63 (request in 22)
24	2...	192.168.8.120	192.168.4.120	ICMP	98	Echo (ping) request id=0x4100, seq=5/1280, ttl=63 (reply in 25)
25	2...	192.168.4.120	192.168.8.120	ICMP	98	Echo (ping) reply id=0x4100, seq=5/1280, ttl=63 (request in 24)
28	2...	192.168.8.120	192.168.4.120	ICMP	98	Echo (ping) request id=0x4100, seq=6/1536, ttl=63 (reply in 29)
29	2...	192.168.4.120	192.168.8.120	ICMP	98	Echo (ping) reply id=0x4100, seq=6/1536, ttl=63 (request in 28)
32	2...	192.168.8.120	192.168.4.120	ICMP	98	Echo (ping) request id=0x4100, seq=7/1792, ttl=63 (reply in 33)
33	2...	192.168.4.120	192.168.8.120	ICMP	98	Echo (ping) reply id=0x4100, seq=7/1792, ttl=63 (request in 32)
34	2...	192.168.8.120	192.168.4.120	ICMP	98	Echo (ping) request id=0x4100, seq=8/2048, ttl=63 (reply in 35)
35	2...	192.168.4.120	192.168.8.120	ICMP	98	Echo (ping) reply id=0x4100, seq=8/2048, ttl=63 (request in 34)
36	2...	192.168.8.120	192.168.4.120	ICMP	98	Echo (ping) request id=0x4100, seq=9/2304, ttl=63 (reply in 37)
37	2...	192.168.4.120	192.168.8.120	ICMP	98	Echo (ping) reply id=0x4100, seq=9/2304, ttl=63 (request in 36)
38	2...	192.168.8.120	192.168.4.120	ICMP	98	Echo (ping) request id=0x4100, seq=10/2560, ttl=63 (reply in 39)
39	2...	192.168.4.120	192.168.8.120	ICMP	98	Echo (ping) reply id=0x4100, seq=10/2560, ttl=63 (request in 38)
42	2...	192.168.8.120	192.168.4.120	ICMP	98	Echo (ping) request id=0x4100, seq=11/2816, ttl=63 (reply in 43)
43	2...	192.168.4.120	192.168.8.120	ICMP	98	Echo (ping) reply id=0x4100, seq=11/2816, ttl=63 (request in 42)
46	2...	192.168.8.120	192.168.4.120	ICMP	98	Echo (ping) request id=0x4100, seq=12/3072, ttl=63 (reply in 47)
47	2...	192.168.4.120	192.168.8.120	ICMP	98	Echo (ping) reply id=0x4100, seq=12/3072, ttl=63 (request in 46)
48	3...	192.168.8.120	192.168.4.120	ICMP	98	Echo (ping) request id=0x4100, seq=13/3328, ttl=63 (reply in 49)
49	3...	192.168.4.120	192.168.8.120	ICMP	98	Echo (ping) reply id=0x4100, seq=13/3328, ttl=63 (request in 48)
50	3...	192.168.8.120	192.168.4.120	ICMP	98	Echo (ping) request id=0x4100, seq=14/3584, ttl=63 (reply in 51)
51	3...	192.168.4.120	192.168.8.120	ICMP	98	Echo (ping) reply id=0x4100, seq=14/3584, ttl=63 (request in 50)
54	3...	192.168.8.120	192.168.4.120	ICMP	98	Echo (ping) request id=0x4100, seq=15/3840, ttl=63 (reply in 55)
55	3...	192.168.4.120	192.168.8.120	ICMP	98	Echo (ping) reply id=0x4100, seq=15/3840, ttl=63 (request in 54)

Figure 13: Capture between babel1 and babel4 after bringing interface back up

Q4: How long does it take for the icmp messages to return back to the originalroute? How does this time compare to that of Lab 2?

A4: 18 seconds and it is slower

3.2.1.4 Step 2: Routing Tables Affecting Traceroutes

We begin this step by verifying the installation of traceroute on Host1. We run traceroute to Host4 and get the results shown in Figure: 14

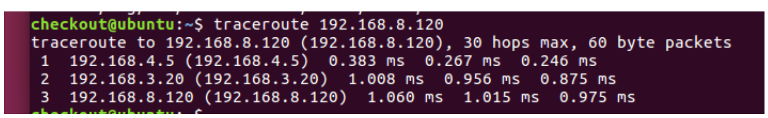


Figure 14: Traceroute with link up

Then, we brought down eth2 on babel4 once again and ran traceroute.

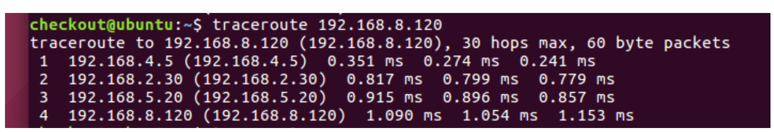


Figure 15: Traceroute with link down

Q5: Describe what you observe and provide a screenshot. Anything interesting that you noticed. Hint: The route should be somewhat puzzling

A5: It is showing the traceroute reaches babel3, but then it shows the interface on babel4 that is connected to babel2. See figure: 15

Q6: What do you observe in the Wireshark captures as the reason behind what you discovered in the traceroute? Provide evidence in the form of screenshots and Wireshark captures.

A6: The requests are going one way and the replies are going another. See Figure: 16

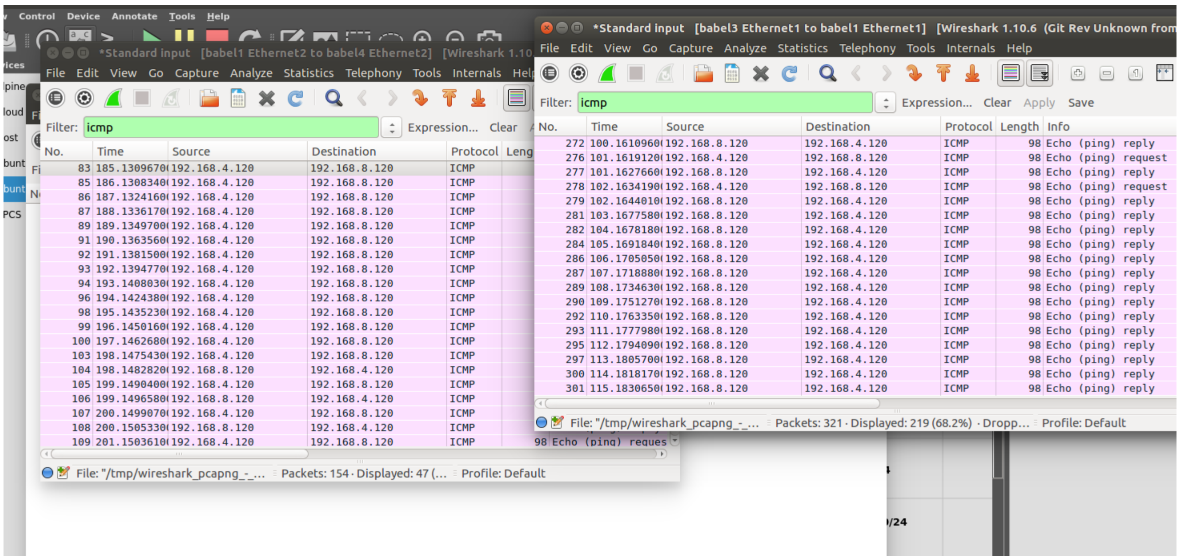


Figure 16: Wireshark Captures

3.2.1.5 Step 3: Compare and Contrasting RIP and Babel

In this step we reflected back to lab 2 in order to compar RIPv2 and Babel.

Q7: What commands did you use to redistribute static routes in RIP? How did we redistribute the static routes using Babel?

A7: After going into the `router rip` configuration we ran the `redistribute static` command. `-C 'redistribute metric 128'`

Q8: Referring to your answer to the previous question about Babel, does this command only redistribute static routes?

A8: No, it tells babel to redistribute all routes in the routing table. They can be from any source.

Q9: What protocol is used by Babel that is not used by RIPv2 by default? What protocol do they both use?

A9: Babel uses both IPv4 and IPv6 by default RIP only works with IPv4 by default. They both use UDP at the transport layer

3.2.2 Key Learning/Takeaways:

In this exercise we learned about how the network converges when a link goes down in Babel. We also compared these results to our results in Lab 2. From this exercise, we determined that Babel is just as effective as RIP in a wired network. While Babel and RIP have several similarities, Babel offers additional features like wireless routing and IPv6 support by default. In the following exercises we examine the Babel in wireless and hybrid networks

3.3 Exercise 3: Configuring OpenWrt and Setting up a Babel Network on Physical routers

3.3.1 Analysis & Evidence

3.3.1.1 Step 0: Exercise 3 - Network Diagrams & Tables

See Figure 3 and Table ?? in Section 2

3.3.1.2 Step 1: Construct Your Network

We begin by flashing the dd-wrt router with the open-wrt firmware that was provided to us in the lab instructions. Doing this, we opened a browser at `192.168.1.1`, configured a password of "admin", and allowed ssh access.

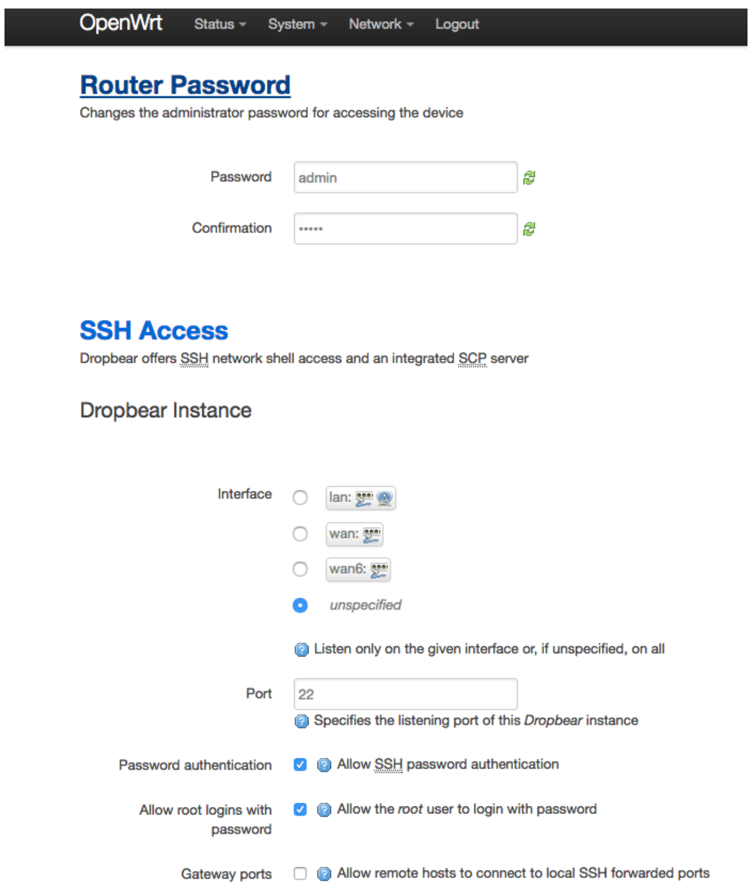


Figure 17: Configuring password / ssh

Next, we changed the default gateway of the AP to be 192.168.10.1. This was done in the interfaces tab, under network in the open-wrt browser gui.

Q1: What is the purpose of flashing the Access point with OpenWRT,rather than keeping DD-WRT?

A1: OpenWrt has direct support for the babel routing protocol, provided that you install the daemon

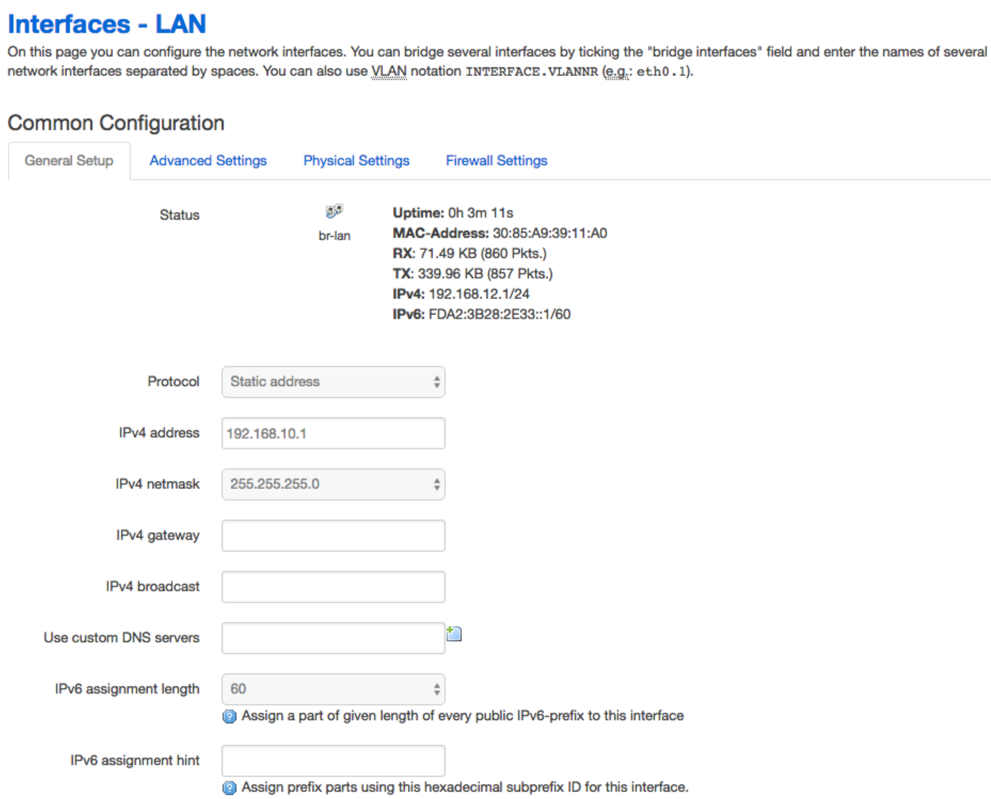


Figure 18: Configuring Static IP

Then, under the network/wifi tab, we edited the configurations of the wireless interface. In particular, we set the operating frequency to channel 7, the Mode to 802.11s, and the SSID to OpenWrtBabel.

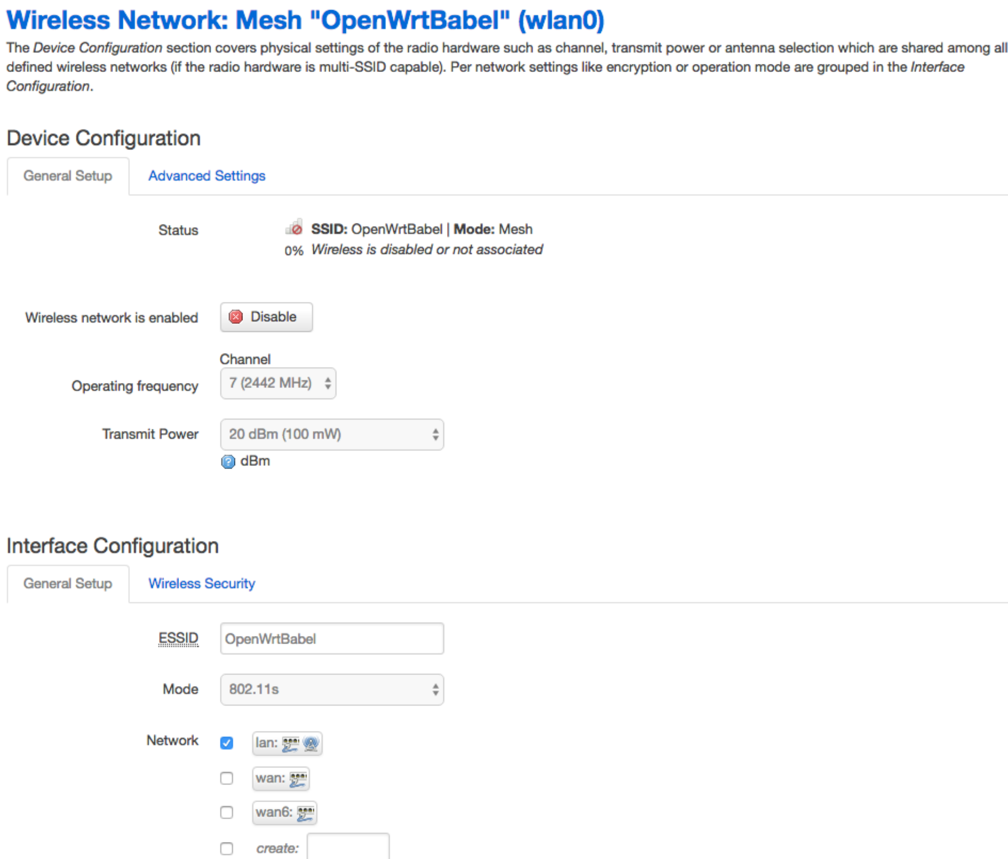


Figure 19: Configuring wireless mesh interface with 802.11s

Q2: What is 802.11s and why is it used?

A2: 802.11s is the standard used currently for Wireless Mesh Networks.

Then, we ssh'd to the new ip that we set on the AP. We were then presented with the open-wrt home page in the terminal as seen in the figure below.

Figure 20: SSH proof to 192.168.11.1 OpenWrt Router

Now, to install the requird package, it was first required to connect the AP's WAN port to the internet. Then we installed and enabled the babel daemon.

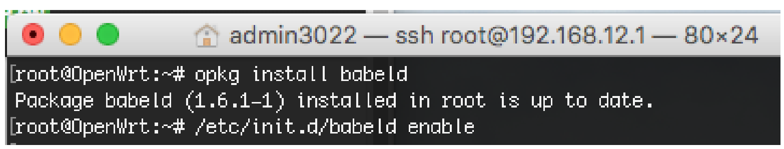


Figure 21: Installing and Enabling the Babel Daemon

We now configured the firewall at /etc/config/firewall as per the instructions in the figure 22 below. This allows the router to accept all babel traffic it sees.

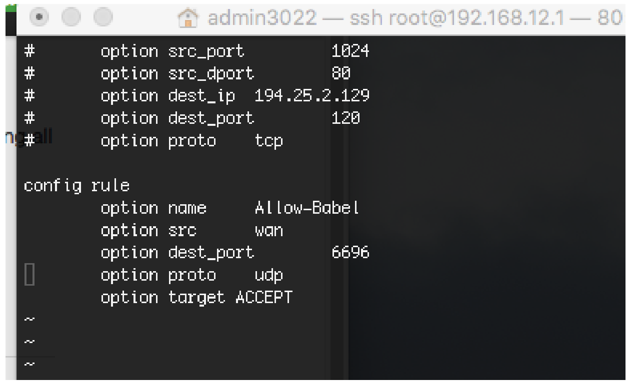


Figure 22: Editing the firewall configuration

We also configured the wireless interface in the command line as seen in the figure 23 below. When configuring the wireless interface for 802.11s in the GUI of OpenWrt, there is no option to specify a mesh id. This is imperative, as multiple routers on the same mesh network have no other way of communicating rather than verifying that the mesh id, ssid, and channel are all the same.

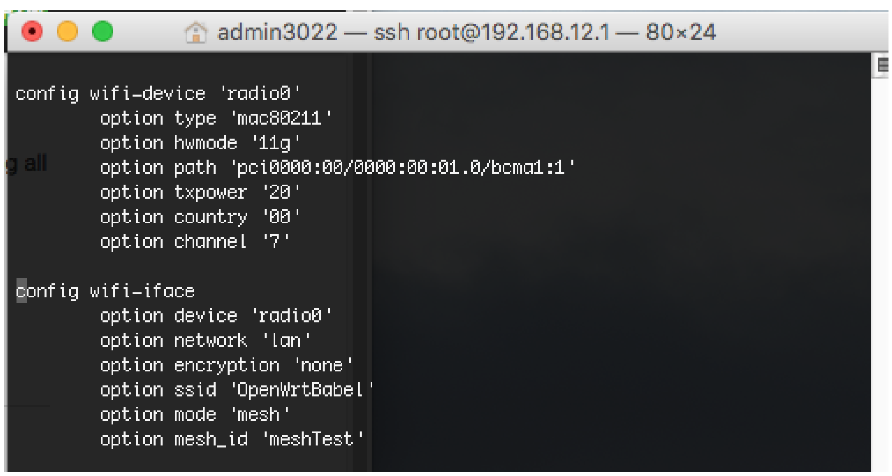


Figure 23: Editing the wireless configuration

As seen in the figure 24 below, we have reloaded both the network and firewall configuration files to ensure that our changes have gone through.

```
root@OpenWrt:~# /etc/init.d/network reload
root@OpenWrt:~# /etc/init.d/firewall reload
Warning: Unable to locate ipset utility, disabling ipset support
 * Clearing IPv4 filter table
 * Clearing IPv4 nat table
 * Clearing IPv4 mangle table
 * Clearing IPv4 raw table
 * Populating IPv4 filter table
   * Zone 'lan'
   * Zone 'wan'
   * Rule 'Allow-DHCP-Renew'
   * Rule 'Allow-Ping'
   * Rule 'Allow-IGMP'
   * Rule #7
   * Rule #8
   * Forward 'lan' -> 'wan'
 * Populating IPv4 nat table
   * Zone 'lan'
   * Zone 'wan'
 * Populating IPv4 mangle table
   * Zone 'lan'
   * Zone 'wan'
 * Populating IPv4 raw table
   * Zone 'lan'
   * Zone 'wan'
```

Figure 24: Reloading the firewall and network configurations

Next, we checked the routing table in 25.

```
root@OpenWrt:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.48.1     0.0.0.0         UG    0      0        0 eth0.2
172.16.48.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0.2
172.16.48.1     0.0.0.0         255.255.255.255 UH    0      0        0 eth0.2
192.168.11.0    0.0.0.0         255.255.255.0   U     0      0        0 br-lan
root@OpenWrt:~#
```

Figure 25: Checking the inital routing table

Finally, we started the babel daemon in the shell with the `babeld -d1 br-lan` command as seen in figure 26.

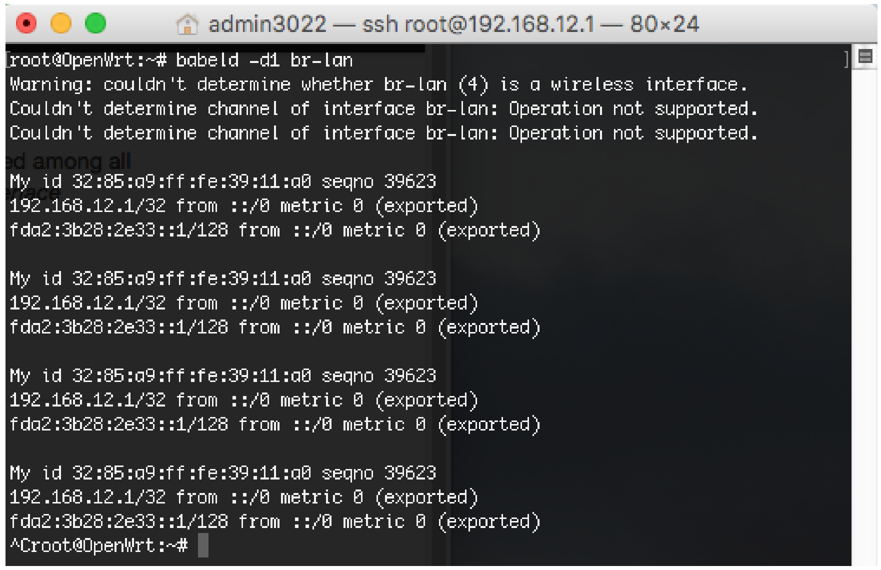


Figure 26: Preliminary test of the babel daemon

Q3: What is the purpose of flashing the Access point with OpenWRT,rather than keeping DD-WRT?

A3: OpenWrt has direct support for the babel routing protocol, provided that you install the daemon

3.3.1.3 Step 2: Configuring a Babel Network Topology on multiple physical routers

Now that one physical router was set up to be enabled for babel, we then sought out to create a topology that included multiple physical routers running the same babel daemon. In our case, we only configured one more additional router. In our case, the router setup in the previous step (192.168.11.1) had a MAC ID of `30:85:a9:39:17:00`, while the new router to be configured (192.168.12.1) had a MAC ID of `30:85:a9:39:11:a0`.

Next, on 192.168.11.1, we ssh'd in once more, and began a tcpdump packet capture on the `br-lan` interface. This capture was named: `babelDemo.pcap` and can be found in the list of attachments. See the figure 27 below for the capture command.

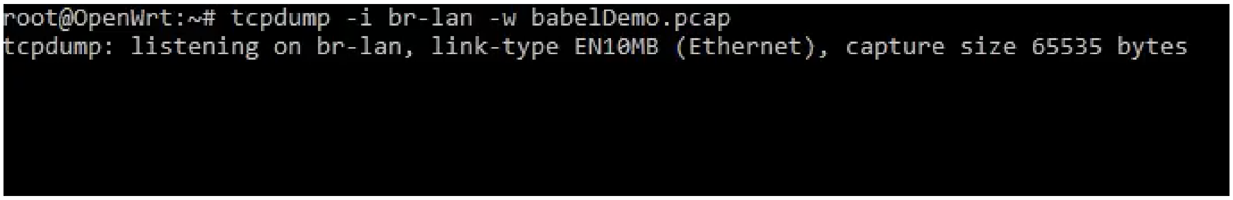


Figure 27: Starting inital packet capture

Next, we started the babel daemon once again, on another ssh terminal to 192.168.11. At the same time, we had the newly configured AP connected to another machine, already running the babel daemon. Very shortly after the babel daemon was active on both machines, we began to see similar neighbor messages in the debug terminal of the babel daemon. An example is in figure 28 below.

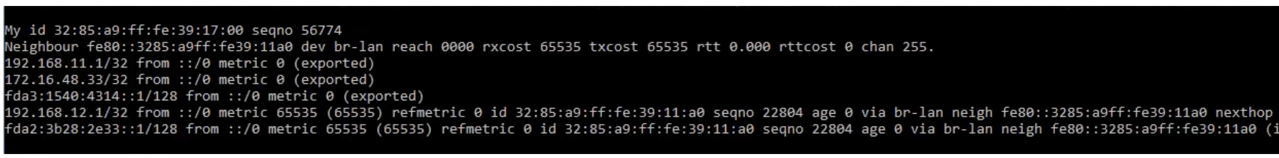


Figure 28: Proof of neighbor messages in the babel daemon debug

As seen in figure 28 above, the babel debug terminal shows the advertisement of the router's MAC ID, IPv6 address, and most importantly a route with the amount of cost it takes to get to the other router running the babel daemon, in both IPv4 and IPv6.

Shortly after seeing these neighbor messages, we initiated a ping from the 192.168.11.1 network to the 192.168.12.1, proving that we have connectivity in the mesh network from one node to another. See figure 29.

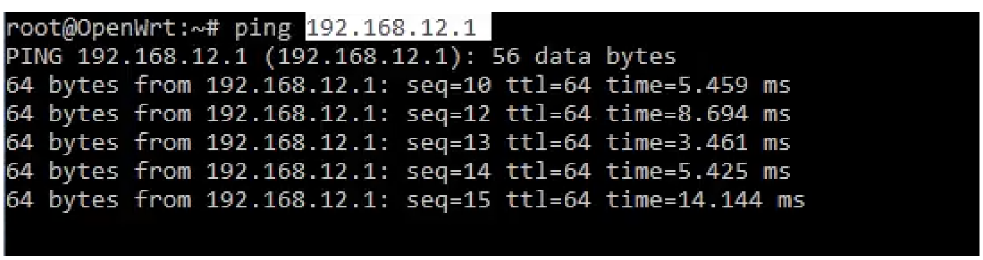


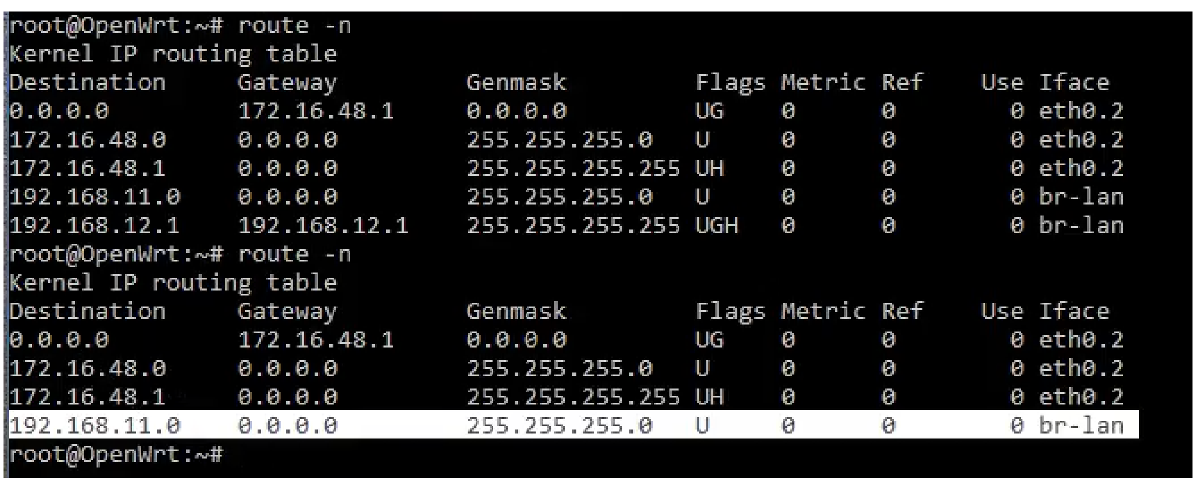
Figure 29: Proof of connectivity between 192.168.11.1 and 192.168.12.1 nodes

Next, we navigated to the browser on the machine connected to 192.168.11.1.

Figure 30: Verifying the Associated Stations in the OpenWrt GUI.

As seen in the figure 30 above, under Associated stations, the MAC ID, as well as the IPv4 address is shown of the other router running babel daemon (192.168.12.1). In this field, there are also various details that are shown, such as the signal, noise, rx rate, and tx rate.

Finally, we checked the routing table of the 192.168.11.1 router after we had stopped the babel daemon. As seen in figure 31 below, the first result is the routing table when the babel daemon is still running (showing the 192.168.12.1 node being populated), and the second result shows the routing table immediately after the babel daemon is stopped, geting rid of the 192.168.12.1 node.



```
root@OpenWrt:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.48.1     0.0.0.0         UG    0      0        0 eth0.2
172.16.48.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0.2
172.16.48.1     0.0.0.0         255.255.255.255 UH    0      0        0 eth0.2
192.168.11.0    0.0.0.0         255.255.255.0   U     0      0        0 br-lan
192.168.12.1    192.168.12.1    255.255.255.255 UGH   0      0        0 br-lan
root@OpenWrt:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.48.1     0.0.0.0         UG    0      0        0 eth0.2
172.16.48.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0.2
172.16.48.1     0.0.0.0         255.255.255.255 UH    0      0        0 eth0.2
192.168.11.0    0.0.0.0         255.255.255.0   U     0      0        0 br-lan
root@OpenWrt:~#
```

Figure 31: Checking the routing table first as Babel is running, and then after the babel daemon is stopped.

Finally, we stopped the terminal that had the tcpdump running, and exported the capture so that it was viewable in wireshark. See figure 32 for an example of this capture, after it was filtered for babel packets.

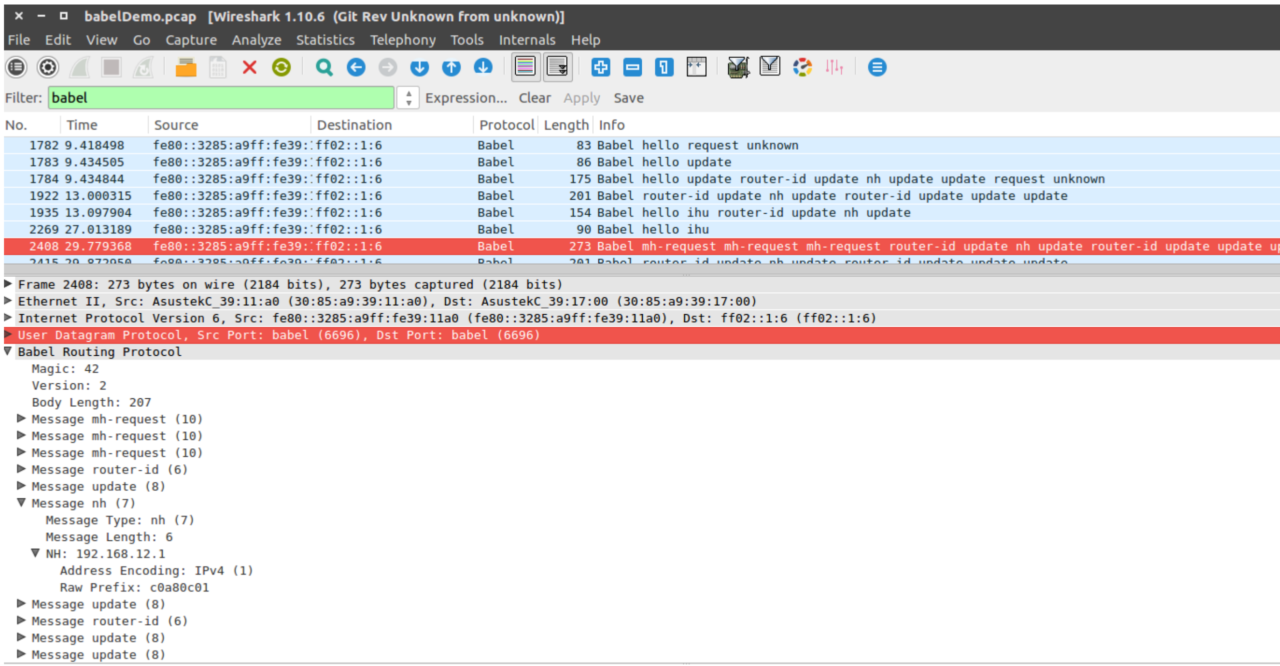


Figure 32: Example of finalized wireshark capture

This capture can be better viewed, as it is located in the list of attachments as `babelDemo.pcap`. The most important takeaway from figure 32 is that each of the babel packets belong to a particular ipv6 multicast group. Also, under the babel routing protocol layer, there are many other fields. These fields include important messages such as the nh (next hop), and all of the other information located in the routing table of the babel node advertising it.

A demo of this step can be found at the following link:
https://www.youtube.com/watch?v=f1m22gHIzjs

3.3.2 Key Learning/Takeaways:

In this exercise, we learned that OpenWrt can be difficult to deal with. For example, when attempting to assign static IPs for 3 of the access points, only 2 of them actually took the address, while the last one could not be accessed, even after multiple resets. Other downfalls of using the OpenWrt firmware will be assesed in the osbp section. Despite this, we were eventually able to create a fully operational Babel mesh network through use of 802.11s and analyze a convesation between two mesh nodes. Using wireshark, we observed the type and order that babel sends messages in (Babel hello, request, update, ihu (i hear u), etc.)

3.4 Exercise 4: Babel/Rip Hybrid Mesh Architecture with CORE

3.4.1 Analysis & Evidence

3.4.1.1 Step 0: Preparation

See Figure 4 in Section 2 for our final network diagram for Exercise 4.
We downloaded the CORE virtual machine and set it up according to step 0 in our lab instructions.

3.4.1.2 Step 1: Setting Up Babel Enabled Wired Mesh Network Using C.O.R.E.

After opening CORE, we made the canvas size wider by selecting canvas size from the View drop down menu in the top tool bar. Next, we clicked on the Router Icon in the left toolbar and placed 4 routers on our canvas. Next, we clicked on the link icon in the toolbar and began connecting all of the nodes in a mesh architecture, shown in Figure 33.

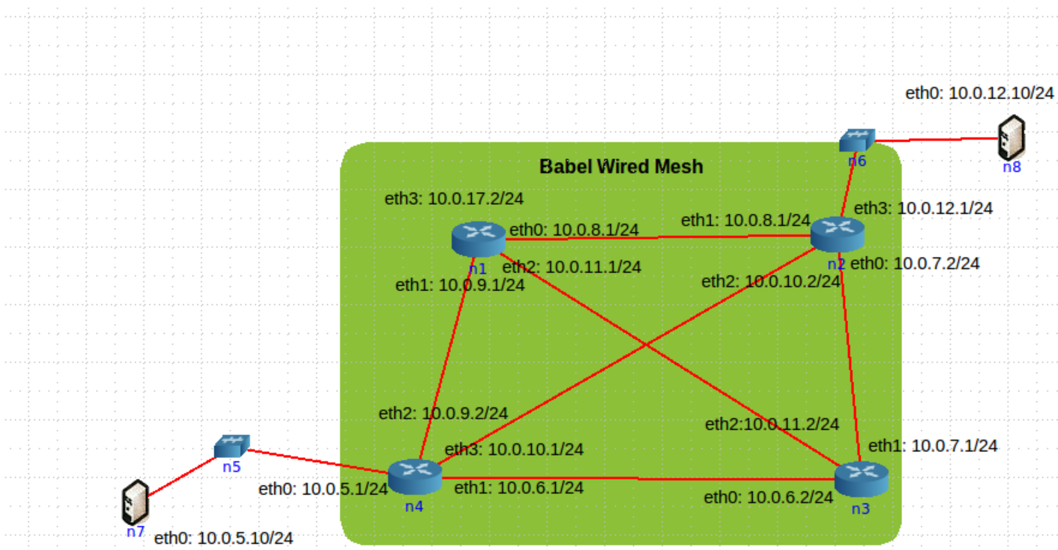


Figure 33: Wired Babel Mesh Network

We added two desktop host connected to switches and then connected the hubs to two separate networks.
Next we began configuring babel on the nodes. On n1, we right clicked on the icon and clicked configure to bring up the configuration menu shown in Figure 34. Next, we clicked on the services button 35and then clicked on Babel and disabled OSPF. We clicked the wrench Icon next to babel to bring up the start up

configuration. We added the babel daemon command to the startup/shutdown command tab shown in Figure 36.

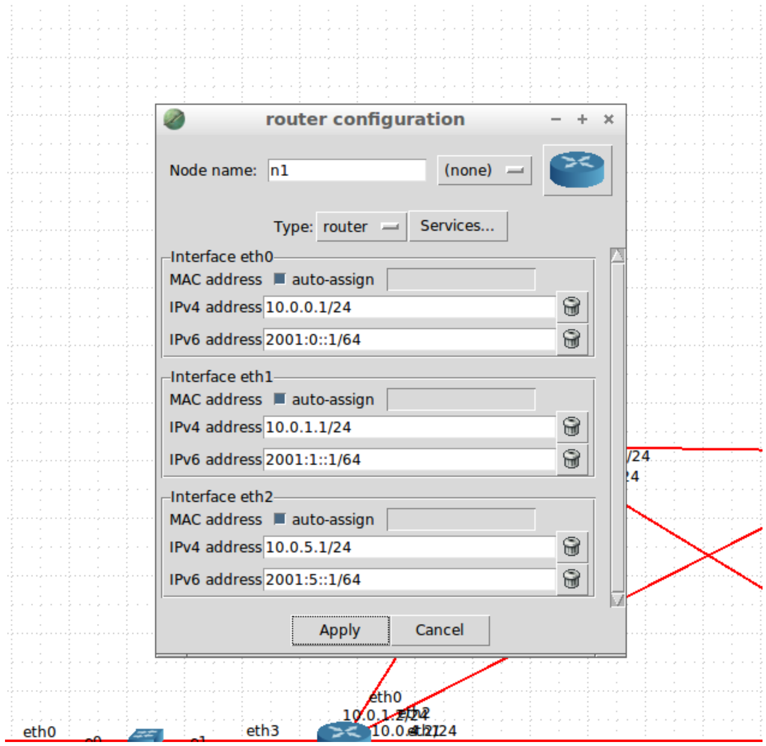


Figure 34: main configuration page

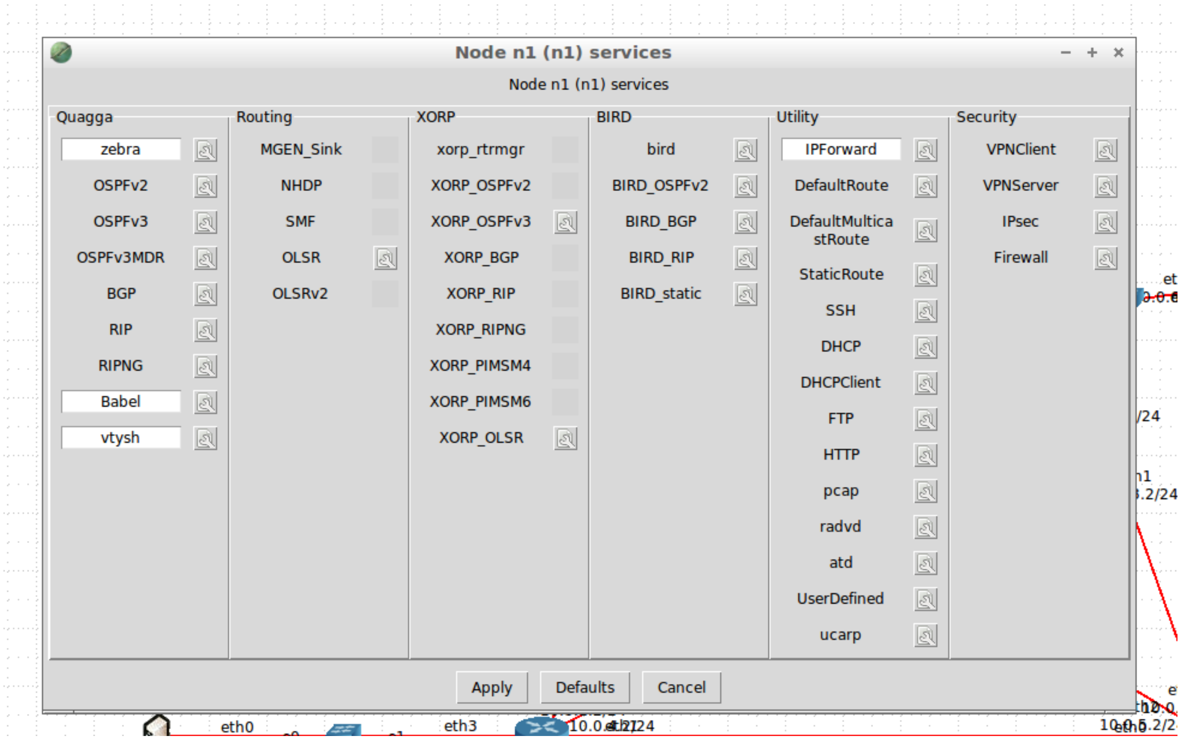


Figure 35: Enable Service configuration

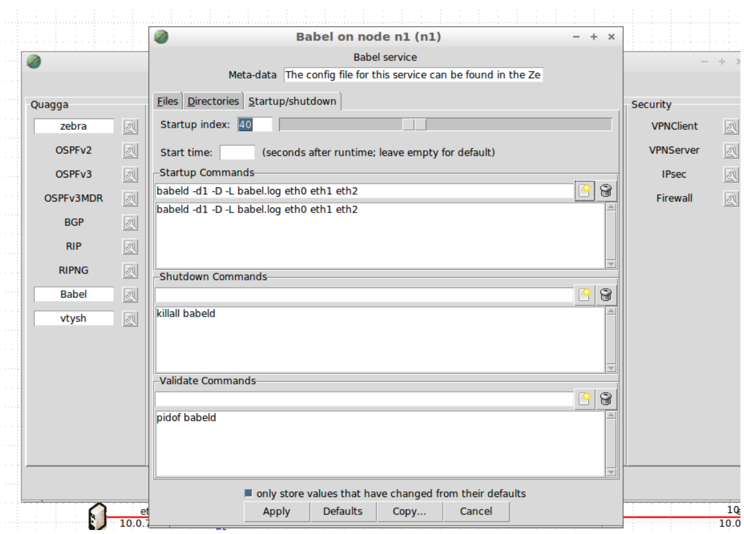


Figure 36: Babel daemon start up command

In the startup command section show in Figure 36, we entered the command **`babeld -d1 -L babel.log eth0 eth1 eth2`**. This allows babel to start automatically when we start the emulation.
We did the same thing on all of the other nodes. On the nodes that have host connected to them, we had to create a configuration file and point the startup command to it. Figure 37 shows we created a new file and added the command **`redistribute metric 128`**. This tells babel to redistribute connected basically, or you can specify a specific network to redistribute. Figure 38 shows the startup command we had to use on the two nodes connected to host.

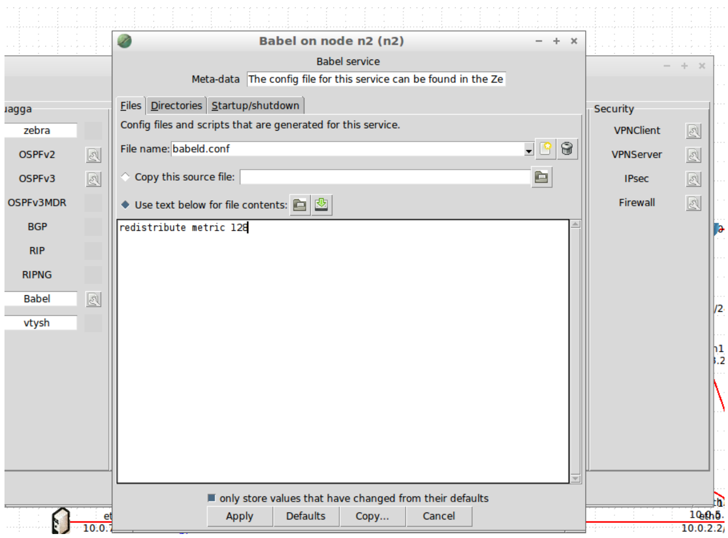


Figure 37: Babel configuration, redistribute metric

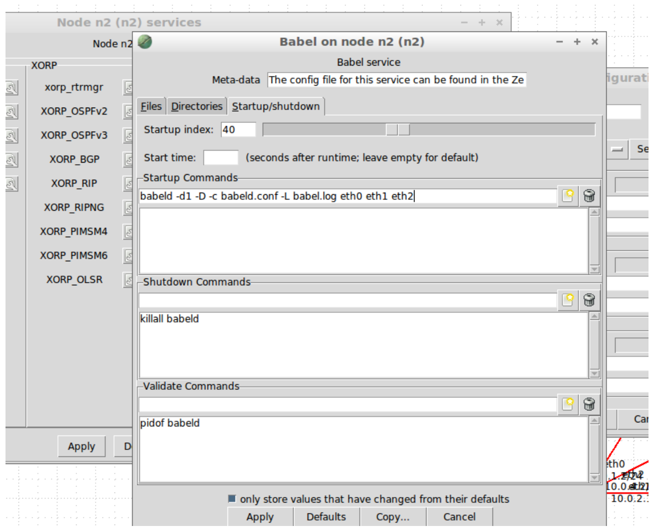


Figure 38: Babel startup command pointing to configuration file

After we had the network fully connected and configured, we selected IPv4 routes under the Widgets drop down menu. This allows us to simply hover over nodes and see their routing tables.
We clicked the green play button to start the emulation. We verified that the routing tables were getting populated by hovering over them and then pinged from one host to the other with success.
Next, we launched Wireshark on some of the nodes and observed babel packets. We did not spend much time analyze them, because they are the same from exercise 2.
We stopped the emulation and saved the file as new_sp.imn.

3.4.1.3 Step 2: Setting up a Wireless Mesh Network Using C.O.R.E.

In this exercise we setup the same sort of configuration, but with wireless routers. First we added four WLAN, by select the hub/switch button in the left toolbar and then selecting the cloud Icon. Next, we right clicked on one of the clouds and selected configure shown in Figure 39

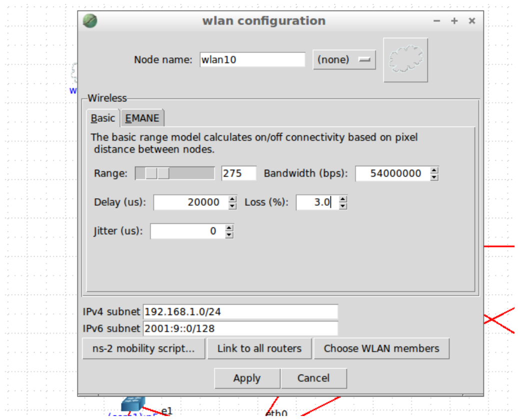


Figure 39: Wireless Lan Configuration Tab

We changed the network and subnet of the WLAN and then clicked the EMANE tab. Under the EMANE tab, We changed emane models to ieee80211abg shown in Figure 40 and clicked apply.

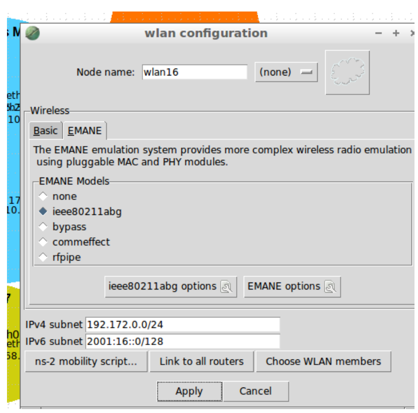


Figure 40: Wireless Lan EMANE Configuration Tab

We followed this same procedure for the other 3 WLANs, only changing their networks and subnets.
Next, we added another router and connected it to the wired network. Then we connected it to all four WLAN's, by dragging a link from it to the clouds. We right clicked on the router and did the same configuration as we did in on the Babel Wired Mesh network. We ran the babel protocol on all 5 interfaces(the wireline and the 4 WLAN).
We edited the startup command for our node in the Wired network that is connected to the new node, to include the new interface we just created.
We created 2 more Wireless routers and connected them in a mesh arrangement

(we connected each router to all the WLAN's) We configured these routers to run Babel in the same way.
Next, we added 4 Wireless clients, by selecting the laptop icon from the router toolbar and then we connected each of the laptops to separate WLAN's
Next we started up the emulation. We tested if we could ping between the wireless clients and we were successful. Next we tested connectivity between a wireless client and a host on the wired network. This was unsuccessful.

Q1: Was the ping successful? If not, then why (Hint: What did we forget to do?)

A1: No. We needed to tell babel to redistribute. The wireless clients are not running babel, so therefore babel needs to redistribute these connected devices so that they can talk to one another. The command redistribute metric 128, is basically like RIP's redistribute connected command. You can redistribute on certain networks if you wish, but we kept it simple by redistributing everything.

Q2: What Happens? Why?

A2: The time increases the further we get from the wireless network. Eventually the destination network becomes unreachable, because the wireless client is to far outside the range of the wireless network. This is a pretty cool simulation of wireless network.

We stopped the emulation and created babeld.conf files on all of the wireless routers and added the `redistribute metric 128` command to the file. Then we changed the startup command by adding the -c <Conf filename > to the command to point the startup command to the configuration. We restarted the emulation. We tested that we could ping from a client in the wireless mesh to another client in the wireless mesh. Next, we tested that a wireless client could ping one of the host in the Wired network. Figure 41 shows that we can ping the wired host from the wireless host.

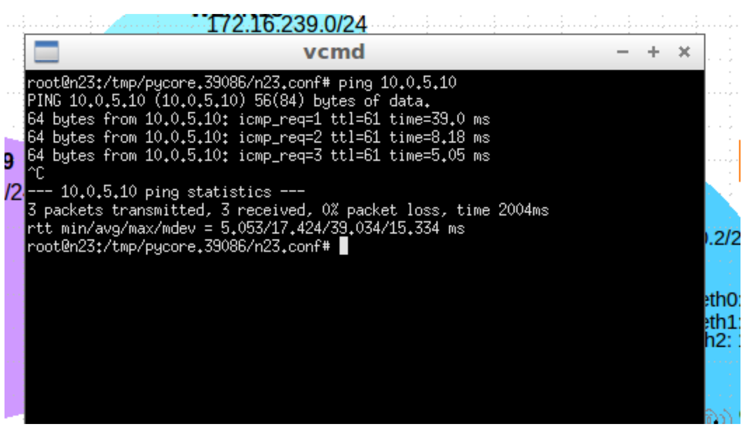


Figure 41: Successful connectivity between wireless network and wired

Figure 42 shows our final network setup after adding the wireless mesh network to the wired mesh network.

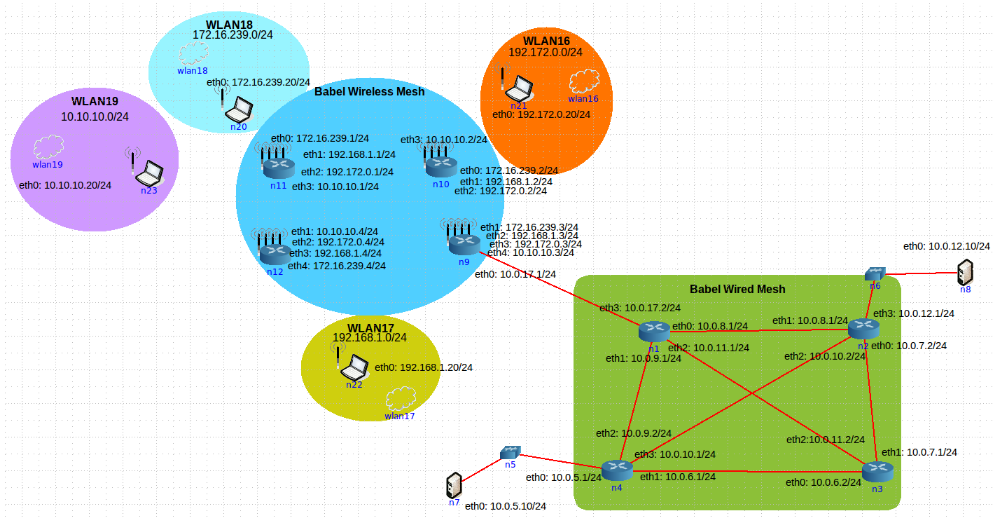


Figure 42: Added Wireless Messh Network

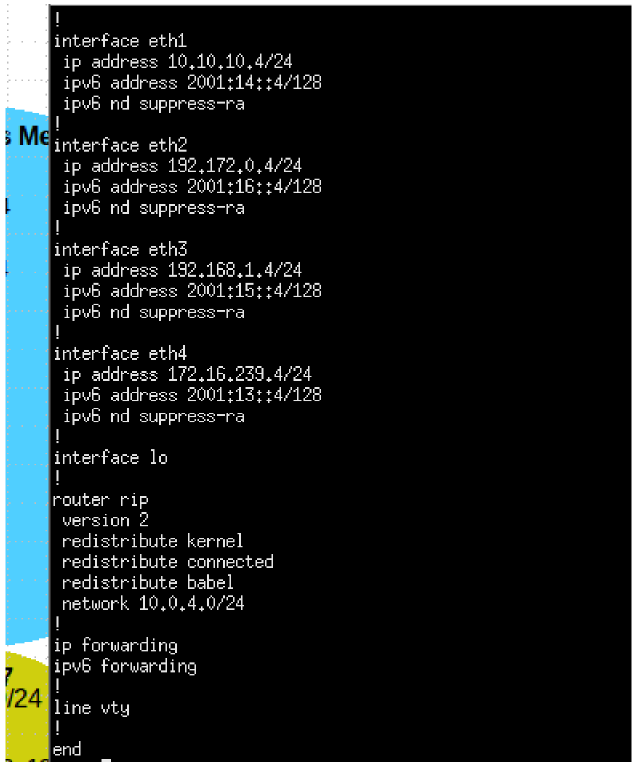
3.4.1.4 Step 3: Adding A RIP network to the Babel Configuration

Next, we added 4 new routers to the topology. On one router we enabled RIP and Babel. We connected the router to all of the WLAN's. Next, we created a configuration file with the `redistribute metric 128`, like we did on the previous babel routers. We created a startup command for all of the interfaces connected to the WLANs, like we did on previous routers pointing it to the configuration file.

On the other 3 routers we enabled RIP and disabled OSPF. We connected the 3 rip routers together and then connected the router running RIP and Babel via wireline to the RIP network.

We added a switch and a host and connected them to the RIP router(n15). And then we started the emulation.

On the Wireless Babel/Wireline RIP router, we opened a terminal and entered the quagga vty shell, by running the command `vtysh`. Then we entered the router config, by typing conf t. Next, we type no router rip and end, to clear out the default RIP settings. Next, we entered the configuration commands listed in the lab instructions to configure RIP on the wireline interface and redistribute the kernel,connected, and babel. Figure 43, shows our running configuration on node 12.



```
!
interface eth1
 ip address 10.10.10.4/24
 ipv6 address 2001:14::4/128
 ipv6 nd suppress-ra
!
interface eth2
 ip address 192.172.0.4/24
 ipv6 address 2001:16::4/128
 ipv6 nd suppress-ra
!
interface eth3
 ip address 192.168.1.4/24
 ipv6 address 2001:15::4/128
 ipv6 nd suppress-ra
!
interface eth4
 ip address 172.16.239.4/24
 ipv6 address 2001:13::4/128
 ipv6 nd suppress-ra
!
interface lo
!
router rip
 version 2
 redistribute kernel
 redistribute connected
 redistribute babel
 network 10.0.4.0/24
!
ip forwarding
ipv6 forwarding
!
line vty
!
end
```

Figure 43: Shows the RIP configuration for n12

I'm not sure we technically need to redistribute connected and babel. The babel daemon is populating the kernel routing tables on all the nodes running babel, so we can probably get by with just redistributing the kernel.
On the other 3 RIP routers we entered the RIP configuration and entered the networks we wanted to run RIP on. On one of the routers we entered redistribute connected, the one to be connected to a host. Figure 44 shows the RIP configuration on node 13, which is connected to the gateway router connected to the wireless mesh.

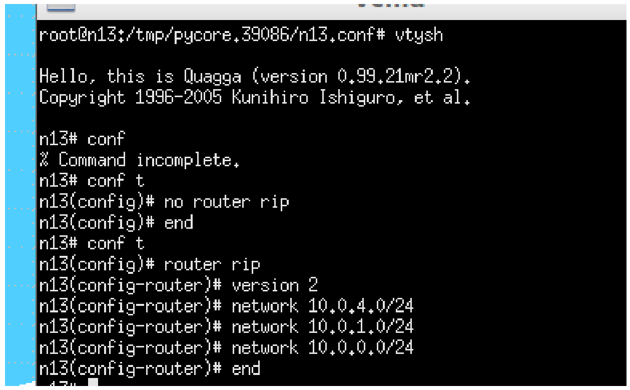


Figure 44: Shows the RIP configuration for n13

Figure 45 shows the RIP configuration used for n15, which is the node we connected to a host.

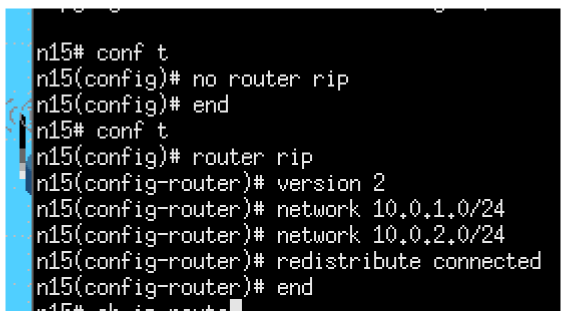


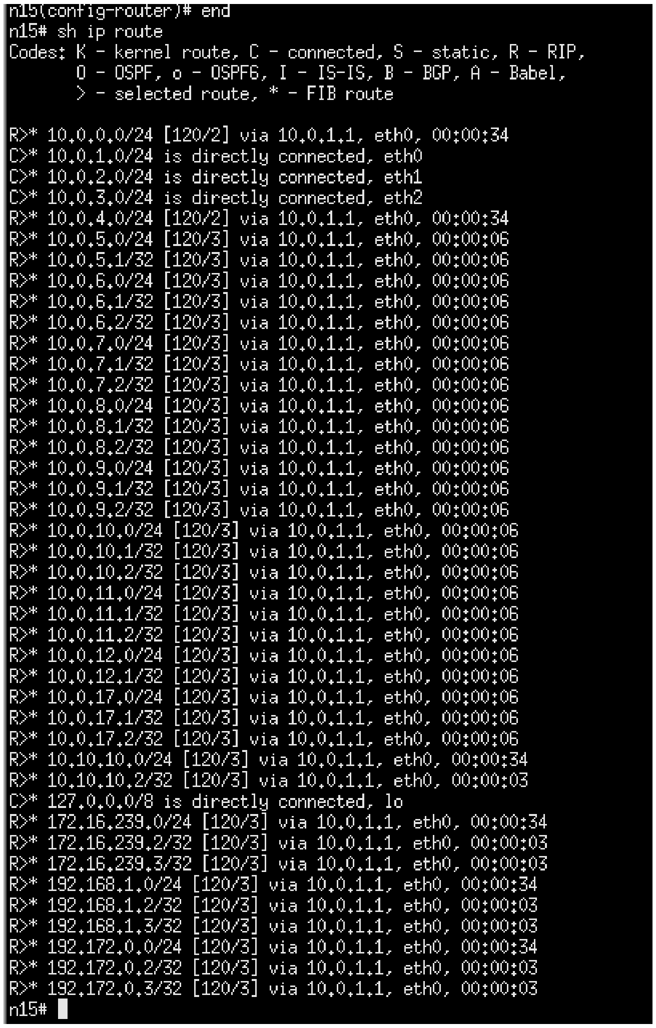
Figure 45: Shows the RIP configuration for n15

Figure 46 shows the routing table on n12. The node is redistributing the kernel routes shown in the Figure, into RIP. Evidence of this is shown in Figure 47, where n15's routing table shows RIP route entries for the routes that were listed as kernel entries on n12.

```
n12# sh ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, o - OSPF6, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

R>* 10.0.0.0/24 [120/2] via 10.0.4.1, eth0, 00:05:19
R>* 10.0.1.0/24 [120/2] via 10.0.4.1, eth0, 00:05:19
R>* 10.0.2.0/24 [120/3] via 10.0.4.1, eth0, 00:05:19
R>* 10.0.3.0/24 [120/3] via 10.0.4.1, eth0, 00:04:16
C>* 10.0.4.0/24 is directly connected, eth0
K * 10.0.5.0/24 via 192.172.0.3, eth3 inactive
K * 10.0.5.1/32 via 192.168.1.3, eth4 inactive
K * 10.0.6.0/24 via 192.168.1.3, eth3 inactive
K>* 10.0.6.1/32 via 10.10.10.3, eth2
K>* 10.0.6.2/32 via 10.10.10.3, eth3
K * 10.0.7.0/24 via 172.16.239.3, eth3 inactive
K * 10.0.7.1/32 via 172.16.239.3, eth4 inactive
K * 10.0.7.2/32 via 172.16.239.3, eth2 inactive
K * 10.0.8.0/24 via 192.168.1.3, eth1 inactive
K * 10.0.8.1/32 via 172.16.239.3, eth1 inactive
K * 10.0.8.2/32 via 172.16.239.3, eth3 inactive
K>* 10.0.9.0/24 via 10.10.10.3, eth2
K * 10.0.9.1/32 via 192.168.1.3, eth2 inactive
K * 10.0.9.2/32 via 192.172.0.3, eth1 inactive
K * 10.0.10.0/24 via 192.172.0.3, eth1 inactive
K * 10.0.10.1/32 via 172.16.239.3, eth4 inactive
K>* 10.0.10.2/32 via 10.10.10.3, eth3
K * 10.0.11.0/24 via 172.16.239.3, eth2 inactive
K * 10.0.11.1/32 via 192.168.1.3, eth3 inactive
K * 10.0.11.2/32 via 192.172.0.3, eth2 inactive
K * 10.0.12.0/24 via 192.168.1.3, eth2 inactive
K * 10.0.12.1/32 via 192.168.1.3, eth4 inactive
K>* 10.0.17.0/24 via 10.10.10.3, eth1
K>* 10.0.17.1/32 via 10.10.10.3, eth2
K>* 10.0.17.2/32 via 10.10.10.3, eth1
C>* 10.10.10.0/24 is directly connected, eth1
K * 10.10.10.1/32 via 192.168.1.1, eth3 inactive
K>* 10.10.10.2/32 via 172.16.239.2, eth3
K * 10.10.10.3/32 via 10.10.10.3, eth1 inactive
C>* 127.0.0.0/8 is directly connected, lo
C>* 172.16.239.0/24 is directly connected, eth4
K>* 172.16.239.1/32 via 192.172.0.1, eth3
K * 172.16.239.2/32 via 172.16.239.2, eth3 inactive
K>* 172.16.239.3/32 via 10.10.10.3, eth4
C>* 192.168.1.0/24 is directly connected, eth3
K>* 192.168.1.1/32 via 192.172.0.1, eth3
K>* 192.168.1.2/32 via 172.16.239.2, eth3
K>* 192.168.1.3/32 via 10.10.10.3, eth1
C>* 192.172.0.0/24 is directly connected, eth2
K * 192.172.0.1/32 via 192.172.0.1, eth3 inactive
K>* 192.172.0.2/32 via 172.16.239.2, eth3
K>* 192.172.0.3/32 via 10.10.10.3, eth1
```

Figure 46: Show IP route on n12



```
n15(config-router)# end
n15# sh ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, o - OSPF6, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

R>* 10.0.0.0/24 [120/2] via 10.0.1.1, eth0, 00:00:34
C>* 10.0.1.0/24 is directly connected, eth0
C>* 10.0.2.0/24 is directly connected, eth1
C>* 10.0.3.0/24 is directly connected, eth2
R>* 10.0.4.0/24 [120/2] via 10.0.1.1, eth0, 00:00:34
R>* 10.0.5.0/24 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.5.1/32 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.6.0/24 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.6.1/32 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.6.2/32 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.7.0/24 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.7.1/32 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.7.2/32 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.8.0/24 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.8.1/32 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.8.2/32 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.9.0/24 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.9.1/32 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.9.2/32 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.10.0/24 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.10.1/32 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.10.2/32 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.11.0/24 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.11.1/32 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.11.2/32 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.12.0/24 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.12.1/32 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.17.0/24 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.17.1/32 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.0.17.2/32 [120/3] via 10.0.1.1, eth0, 00:00:06
R>* 10.10.10.0/24 [120/3] via 10.0.1.1, eth0, 00:00:34
R>* 10.10.10.2/32 [120/3] via 10.0.1.1, eth0, 00:00:03
C>* 127.0.0.0/8 is directly connected, lo
R>* 172.16.239.0/24 [120/3] via 10.0.1.1, eth0, 00:00:34
R>* 172.16.239.2/32 [120/3] via 10.0.1.1, eth0, 00:00:03
R>* 172.16.239.3/32 [120/3] via 10.0.1.1, eth0, 00:00:03
R>* 192.168.1.0/24 [120/3] via 10.0.1.1, eth0, 00:00:34
R>* 192.168.1.2/32 [120/3] via 10.0.1.1, eth0, 00:00:03
R>* 192.168.1.3/32 [120/3] via 10.0.1.1, eth0, 00:00:03
R>* 192.172.0.0/24 [120/3] via 10.0.1.1, eth0, 00:00:34
R>* 192.172.0.2/32 [120/3] via 10.0.1.1, eth0, 00:00:03
R>* 192.172.0.3/32 [120/3] via 10.0.1.1, eth0, 00:00:03
n15#
```

Figure 47: Show IP route on n15

We pinged from the host in the RIP network to a host in the Wired Babel network. Figure 48 shows that we can successfully ping from the RIP network all the way to the Wired Babel Mesh network.

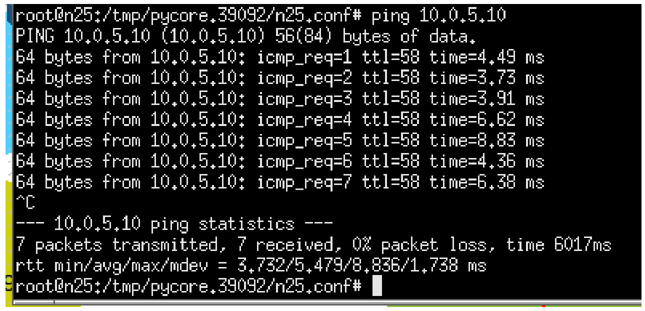


Figure 48: Succesful ping from host on RIP network to host on Babel Wired Mesh Network

We noticed sometimes it took about 30 seconds or more until we could ping, we guess because it was taking a little time to populate the routing tables. Also, there was some inconsistentcy with network 10.0.5.0/24. Sometimes the route would disappear from the routing tables on the nodes running RIP. It would return sometime later. We have not been able to figure out the cause of this, because all of the other nodes running babel have no trouble pinging them at anytime. When the route disappears from the RIP network, the babel networks still have connectivity.

3.4.1.5 Step 4: Wireless Mesh and Link Quality Routing

In this step we experimented with Babel's routing based on the quality of the wireless links. We setup the new configuration files shown in the lab instructions on the nodes running babel. Next, we started the emulation up with the Widget Observer IPv4 routes enabled. We hovered over n12 and watched the routing table, entry for 10.0.5.0/24, periodically removing and hovering over the node to refresh the routing table.

Q3: Explain What the diversity command is for.

A3: According to the babeld man, This option specifies the diversity algorithm to use; true is equivalent to kind 3. The default is false [1]

Q4: Explain what the smoothing-half-life command does.

A4: This specifies the half-life in seconds of the exponential decay used for smoothing metrics for performing route selection, and is equivalent to the command-line option -M.[1]

We took note of the routing table on the wireless node connected to the RIP network. Specifically we noted that the route to network 10.0.5.0/24 on N12 was 192.168.1.3 shown in Figure 49

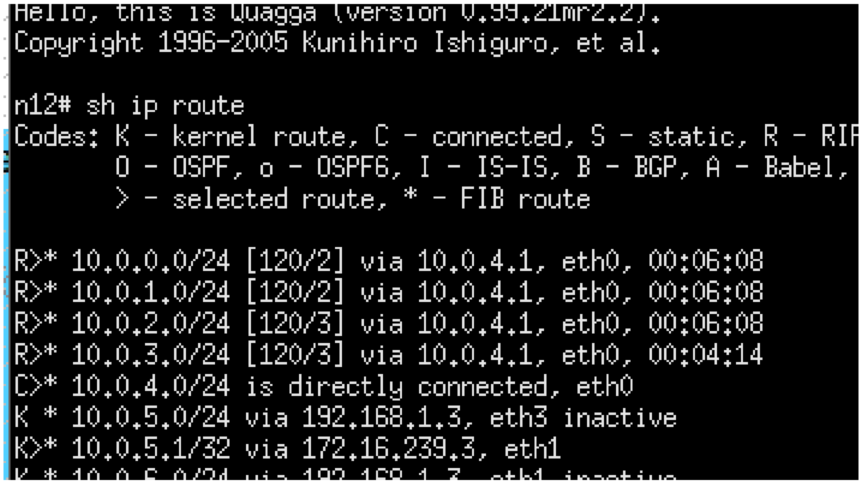


Figure 49: Starting routing entry on N12 for network 10.0.5.0/24

Next, we experimented with moving the wireless nodes around until they were spaced out, making sure the node with the interface that was listed in the routing table of N12 was the farthest away. We kept an eye on the routing table for N12 by enabling the Widget Observer IPv4 Routing and hovering over the node when ever we made location changes. Figure 50 shows how we had the wireless nodes spaced out to get the route to switch.

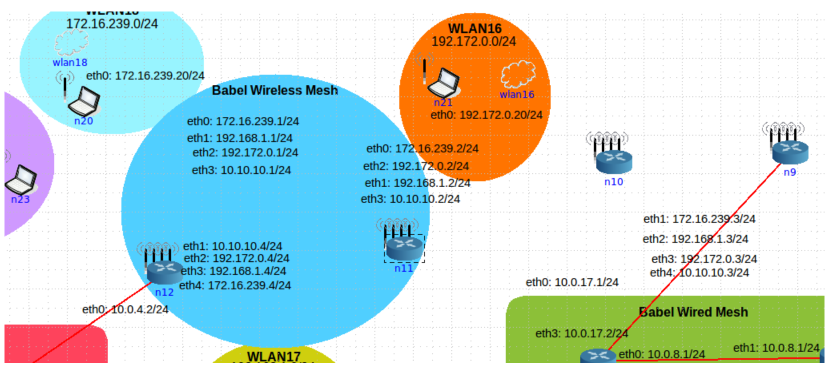


Figure 50: Starting routing entry on N12 for network 10.0.5.0/24

This took quite a bit of experimentation to get the nodes just right to where it switched in the routing table.

Q5: Describe what you see. What do you notice? Why?

A5: We observed the routing table change several times before settling on a route to network 10.0.5.0/24. It began as 192.172.0.1, switched to 192.168.1.3 and then again to 10.10.10.3. This is Babel detecting the best quality link to reach the network.

Figure 51 shows that the route entry for the network 10.0.5.0/24 switched from 192.168.1.3 to 10.10.10.1.

```
R>* 10.0.0.0/24 [120/2] via 10.0.4.1, eth0, 00:11:53
R>* 10.0.1.0/24 [120/2] via 10.0.4.1, eth0, 00:11:53
R>* 10.0.2.0/24 [120/3] via 10.0.4.1, eth0, 00:11:53
R>* 10.0.3.0/24 [120/3] via 10.0.4.1, eth0, 00:09:59
C>* 10.0.4.0/24 is directly connected, eth0
K * 10.0.5.0/24 via 10.10.10.1, eth3 inactive
K * 10.0.5.1/32 via 192.168.1.1, eth4 inactive
```

Figure 51: After moving nodes around route entry on N12 for network 10.0.5.0/24 changed

Because the connectivity between network 10.0.3.0/24 and 10.0.5.0/24 where a bit spotty, we created a video demo to show that we did have connectivity at some points. The video can be viewed at RIP Babel Demo.
Next, we pinged the host at 10.0.5.10 from n12 (the router running Babel and RIP) We started Wireshark on the router running Babel and RIP and identified an interface that we believed was on the router (this was a bit hard to determine, but it had more packets on it than some of the other ones).
We captured ICMP messages for a while and then stopped the capture and saved the pcap file as Ex4Step4-ICMPLinkChange.pcap included in 4 as a file attachment.

35	8.374195	10.10.10.4	10.0.5.10	ICMP	98	Echo (ping) request
36	8.374234	10.0.5.10	10.10.10.4	ICMP	98	Echo (ping) reply
37	9.419490	10.10.10.4	10.0.5.10	ICMP	98	Echo (ping) request
38	9.419530	10.0.5.10	10.10.10.4	ICMP	98	Echo (ping) reply
39	10.422105	10.10.10.4	10.0.5.10	ICMP	98	Echo (ping) request
40	10.422192	10.0.5.10	10.10.10.4	ICMP	98	Echo (ping) reply
44	11.427098	10.10.10.4	10.0.5.10	ICMP	98	Echo (ping) request
45	11.427136	10.0.5.10	10.10.10.4	ICMP	98	Echo (ping) reply
48	12.427427	10.10.10.4	10.0.5.10	ICMP	98	Echo (ping) request
49	12.427482	10.0.5.10	10.10.10.4	ICMP	98	Echo (ping) reply
51	13.431059	10.10.10.4	10.0.5.10	ICMP	98	Echo (ping) request
52	13.431104	10.0.5.10	10.10.10.4	ICMP	98	Echo (ping) reply
54	14.432995	10.10.10.4	10.0.5.10	ICMP	98	Echo (ping) request
55	14.433035	10.0.5.10	10.10.10.4	ICMP	98	Echo (ping) reply
59	15.478113	192.172.0.4	10.0.5.10	ICMP	98	Echo (ping) request
60	15.478173	10.0.5.10	192.172.0.4	ICMP	98	Echo (ping) reply
61	16.436941	192.172.0.4	10.0.5.10	ICMP	98	Echo (ping) request
62	16.437079	10.0.5.10	192.172.0.4	ICMP	98	Echo (ping) reply
64	17.448535	192.168.1.4	10.0.5.10	ICMP	98	Echo (ping) request

Figure 52: Src IP switching during ICMP request

Q6: Describe your observations from the ICMP messages on the Babel/RIP router.

A6: Figure 52 shows that the source IP is changing during the ping. All of the source IP's are on the n12 router. This is Babel detecting a better quality link to the route and adjusting. The other interesting point here is that there is no convergence down time when it selects a different route. It's instantaneous.

3.4.2 Key Learning/Takeaways:

The takeaways from this exercise are that Babel seems to be easier to setup than RIP. You simply just have to tell it to start and what interface to run on at a bare minimum. RIP requires manual configuration for IPv6, while Babel does both IPv4 and IPv6 by default. Babel is more efficient at routing in wireless networks than it is in wired networks.

Babel is capable of switching routes on the fly based on the quality of the wireless link, without any delays or convergence delays. It seems that the convergence times when a link goes down is not quicker than rip as it seemed to be in Lab 2, but this could be because we have a more complicated network than in Lab 2.

3.5 Exercise 5: Security Issues with Babel

3.5.1 Analysis & Evidence

3.5.1.1 Step 0: Exercise 5 - Network Diagrams & Tables

N/A

3.5.1.2 Step 1:

See Project Instructions for analysis of the security of Babel

3.5.2 Key Learning/Takeaways:

4 Lab List of Attachments & Additional Questions

4.1 List of Attachments

The following is a list of the attachments uploaded to the 01-atts directory:

1. 461SPBabelPackets5.10Network.pcapng
2. Babel4-E1toBabel2E1Convergence.pcapng
3. Babel4E0toBabel3E0Convergence.pcapng
4. Babel4E2tobabel1E2.pcapng
5. Ex1$_{B}ringingBabelUp.pcapng$
6. Ex2$_{B}abel3toBabel1Linkdown.pcapng$
7. Ex2$_{B}abel4toBabel2Linkdown.pcapng$
8. Ex4Step4-ICMPLinkChange.pcap
9. babelDemo.pcap
10. babelhelloneighbours.pcapng

4.2 Additional Questions

N/A

5 Lab Observations, Suggestions & Best Practices

5.1 Observations

5.1.0.1 OpenWrt

When configuring the access points with OpenWrt, there were many problems, especially with the GUI that OpenWrt provides. As stated earlier in the report, when attempting to assign 3 different access points with a static IP address, only two of the three actually maintained their address, while the other was not able to be accessed, even after multiple resets. This is where the ASUS firmware restoration tool was used. When editing the configurations through the command line, if there was even a small error, it would be frequent that we would not have access to the access point at all. When configuring the access points for 802.11s, there was no field in the GUI to specify a mesh id. This required an extra step in the lab to enter this manually in the command line. A big problem when completing this lab was reliability of the specific firmware installed on the ASUS RT-N66U Access points. It has been noted by many others in the OpenWrt community that these specific models do not maintain a 5Ghz connection, while maintaining a relatively weak connection when operating at 2.4GHz.

5.1.0.2 CORE

We noticed in Exercise 4 Step 3 that when we setup and enabled RIP on the RIP network sometimes the route to 10.0.5.0/24 would not show up. It seemed that the wireless RIP/BABEL node was not redistributing this route for some reason, even though we could ping from this node to the Babel Wired network. CORE is not very well documented, so I had to figure out a lot of things on my own.

5.2 Suggestions

5.3 Best Practices

Using Core to simulate wireless nodes.

6 Lab References

[1] *Babeld(8) system manager's manual*. [Online]. Available: https://www.irif.fr/~jch/software/babel/babeld.html.

7 Acknowledgments

Instruction - Do not include the question in your lab report.
Did you provide a well written acknowledgment? For example, we wish to acknowledge so and so (including TAs, Lab Manager, online communities, etc.) and for what? [No boiler plate kind of acknowledgment, be specific as for who and for what you are offering your acknowledgments]

8 Lab Extra Credit Exercises

8.1 Extra Credit 1:

N/A

8.2 Extra Credit 2:

N/A

9 Appendices