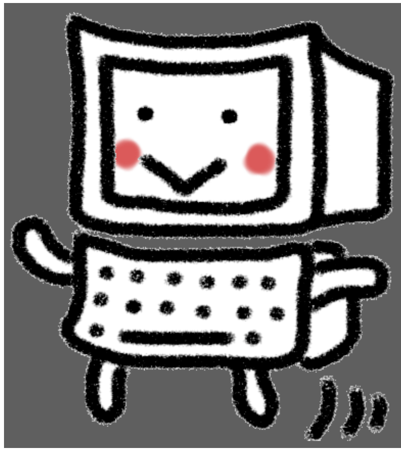


Lab4

GRE Tunnel

and

Auto Creation

Date:	2021/3/30
Deadline:	2021/4/13 00:00

Outline

- Objective
- Environment
- Generic Routing Encapsulation tunnel (GRE tunnel)
- Lab requirements
- Appendix

Outline

- Objective
- Environment
- Generic Routing Encapsulation tunnel (GRE tunnel)
- Lab requirements
- Appendix

Objective

- GRE tunnel configuration and observation
    - Inner and outer headers of packets
- Write a Auto Tunnel Creation Program in C/C++/Golang to
    - filter and parse incoming encapsulated packet
    - create tunnel automatically with parsing result

Outline

- Objective
- Environment
- Generic Routing Encapsulation tunnel (GRE tunnel)
- Lab requirements
- Appendix

Lab environment

- Previous Lab environment
  - Ubuntu 18.04
  - mininet 2.2.2
- C/C++ language compiler
  - Gcc/G++
- Golang
  - Latest version 1.16.2
  - https://golang.org/doc/install

Outline

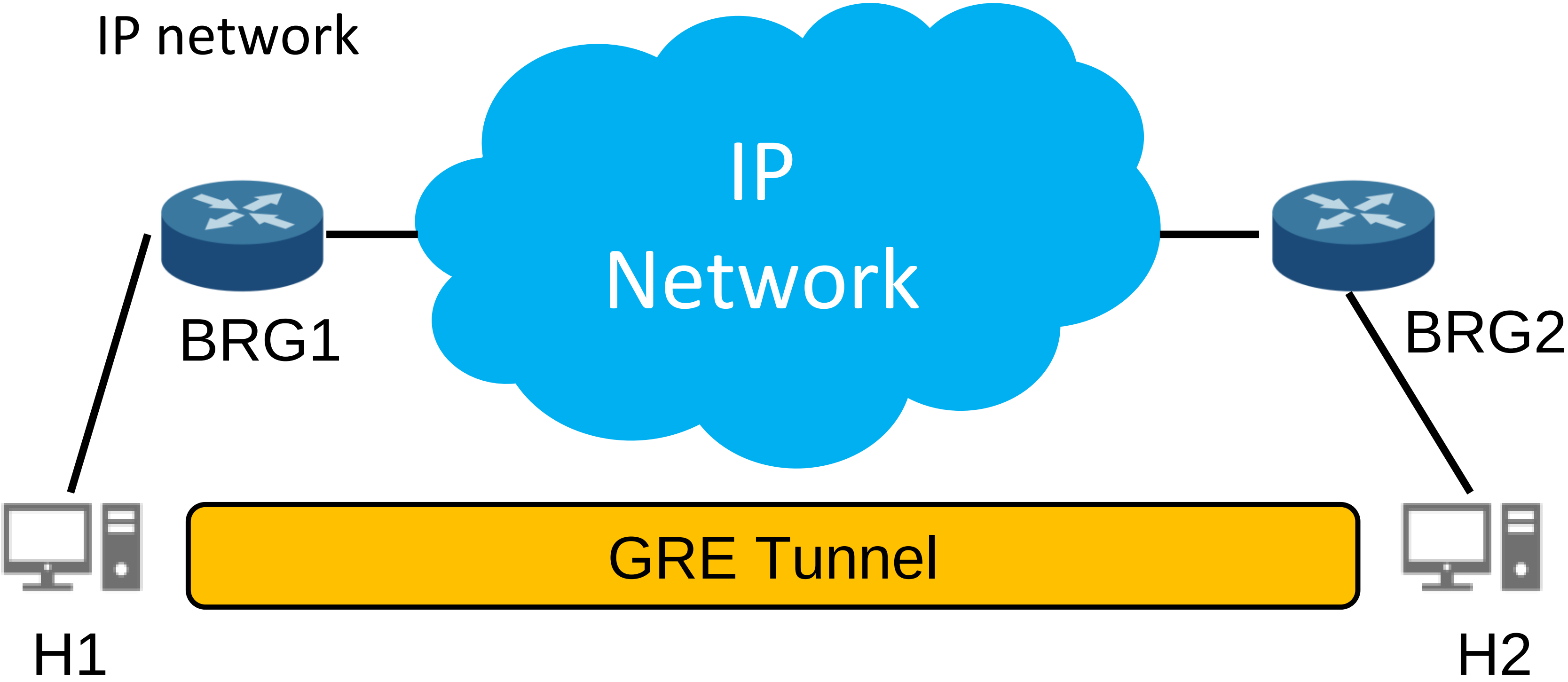
- Objective
- Environment
- Generic Routing Encapsulation tunnel (GRE tunnel)
  - Overview
    - GRE headers
    - Tunneling workflows
  - Example Topology
- Lab requirements
- Appendix

GRE Tunnel and Virtual LANs

- Generic Routing Encapsulation (GRE):

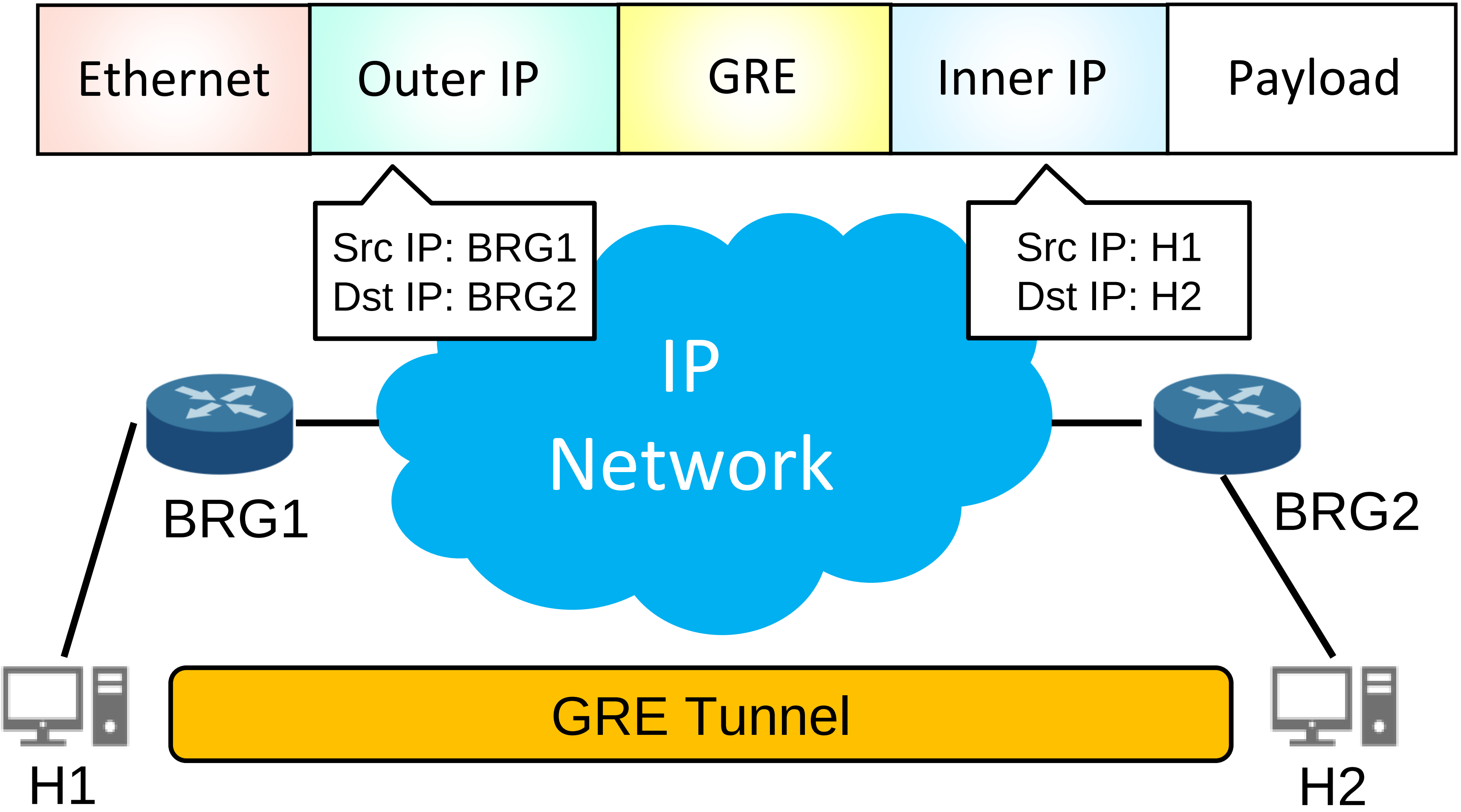
  a protocol for encapsulating data packet inside a virtual point-to-point connection across a network

- Usage in this Lab
  - To create a logically L2 LAN with multiple physical LANs cross IP network



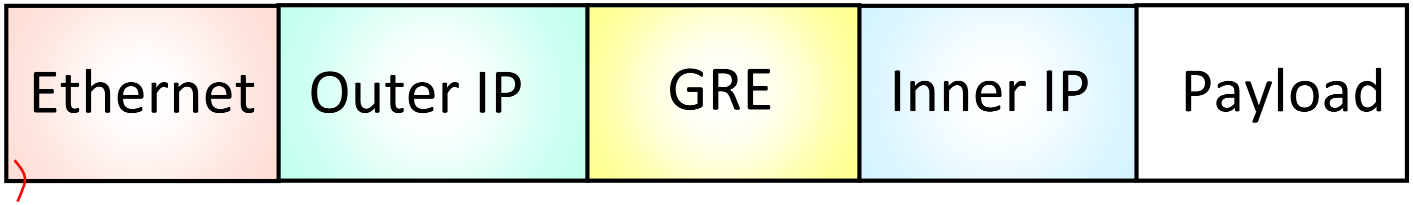
GRE Tunnel Headers

- An IP in IP tunneling protocol
    - Outer IP helps forward packets to remote LANs



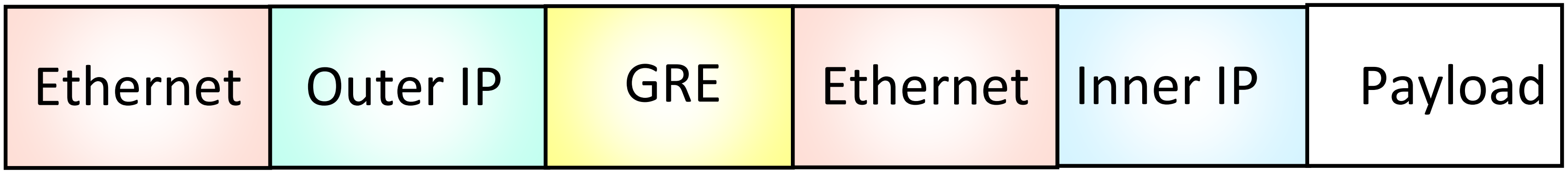
Types of GRE Tunnels

- GRE



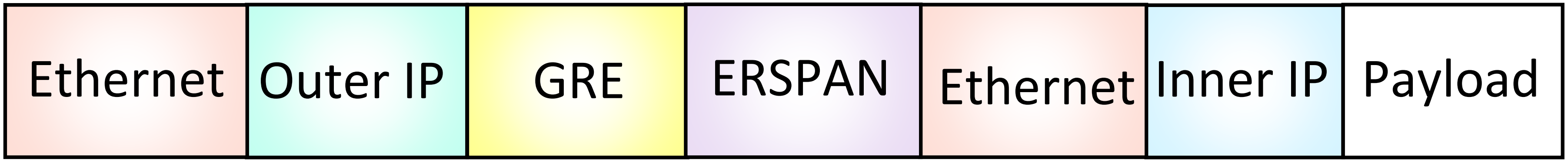
Ethernet	Outer IP	GRE	Inner IP	Payload

- GRETAP



Ethernet	Outer IP	GRE	Ethernet	Inner IP	Payload

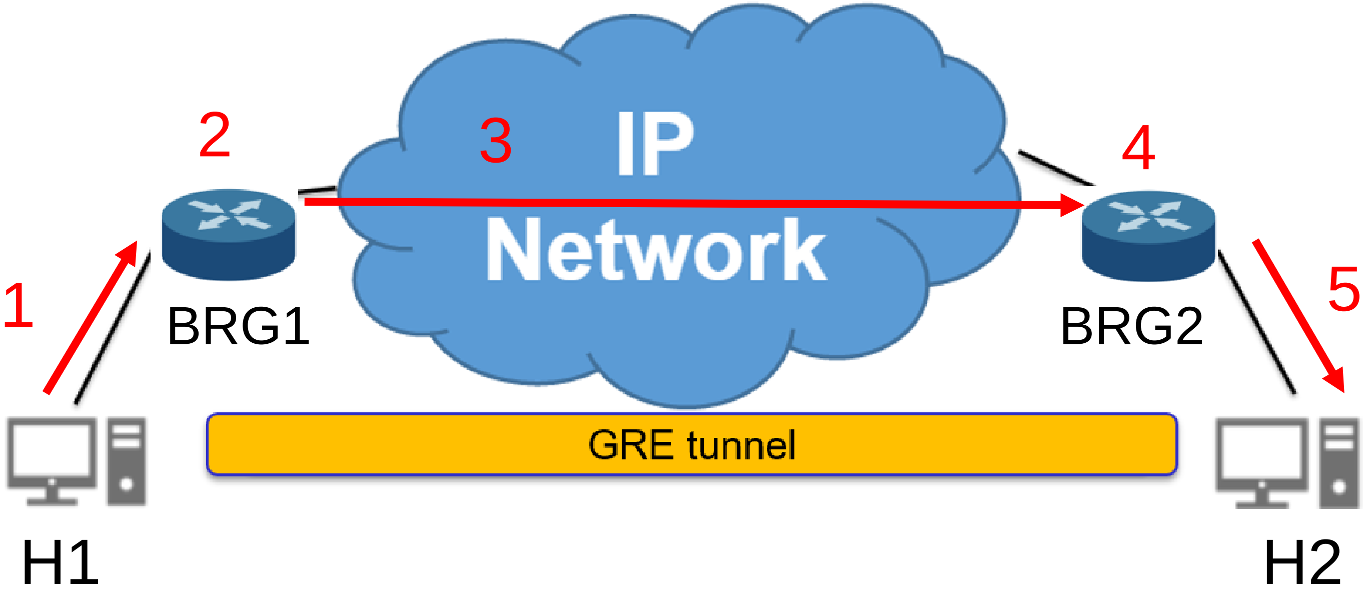
- ERSPAN (Encapsulated Remote Switch Port Analyzer)



Ethernet	Outer IP	GRE	ERSPAN	Ethernet	Inner IP	Payload

GRE Tunneling Workflows

1. H1 sends a packet to request H2
2. BRG1 receives and encapsulates the packet
3. BRG1 forwards the encapsulated packet to a remote BRG (BRG2)
4. BRG2 receives and decapsulates the packet
5. BRG2 forwards the origin packet to H2 base on normal MAC address look up

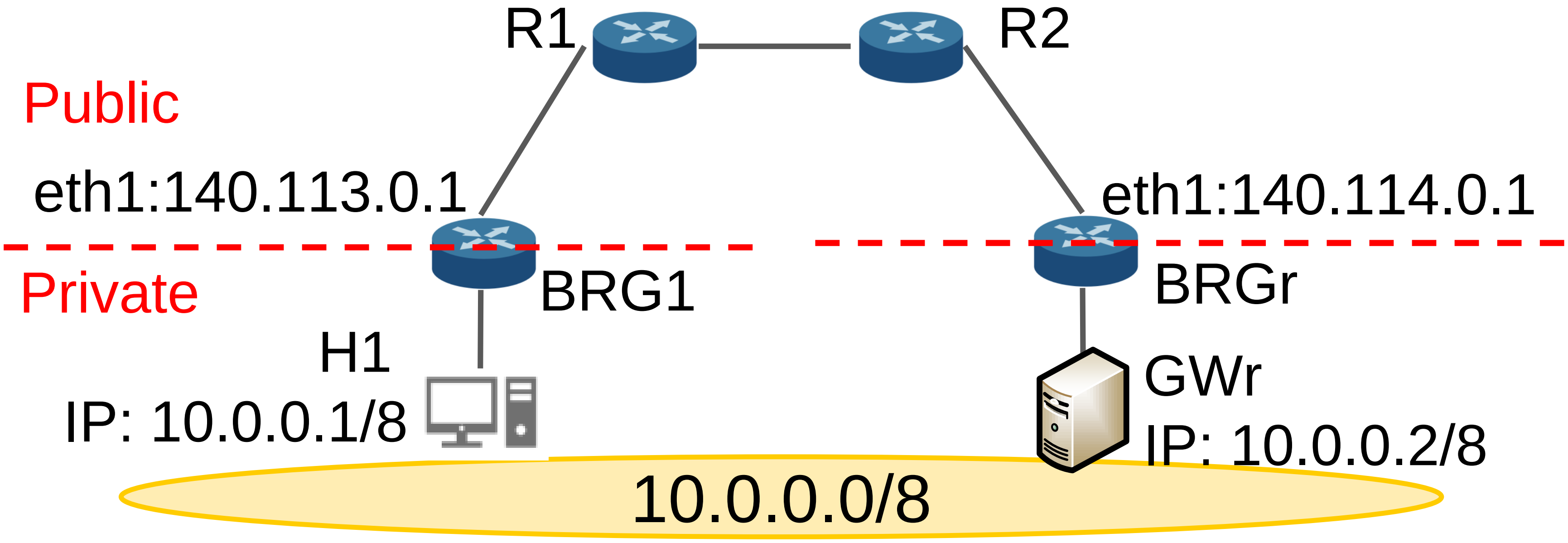


Outline

- Objective
- Environment
- Generic Routing Encapsulation Tunnel (GRE Tunnel)
  - Overview
    - GRE headers
    - Tunneling workflows
  - Example Scenario
- Lab requirements
- Appendix

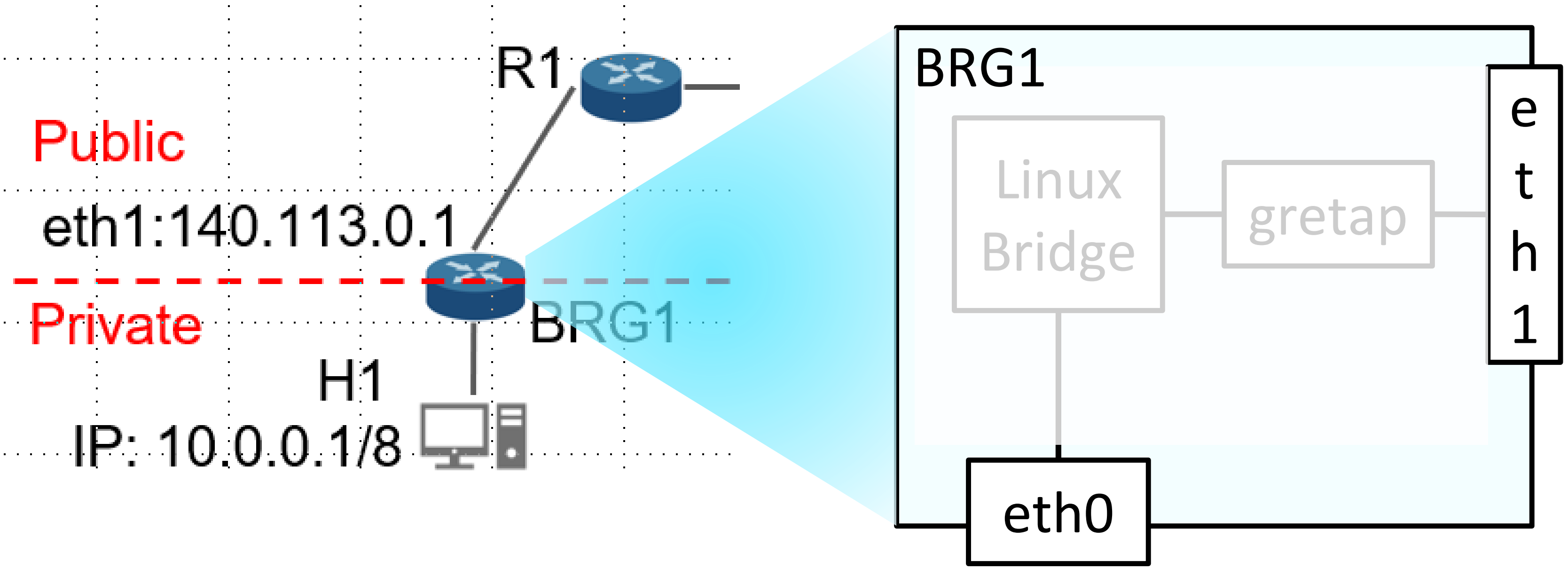
Example Topology

- A local host (H1) behinds a local bridge (BRG1) in a local network
- A remote gateway (GWr) behinds a remote bridge (BRGr) in a remote network
- Two BRGs establish a GRE Tunnel
- H1 uses GWr as the default gateway



BRG Bridge Configuration

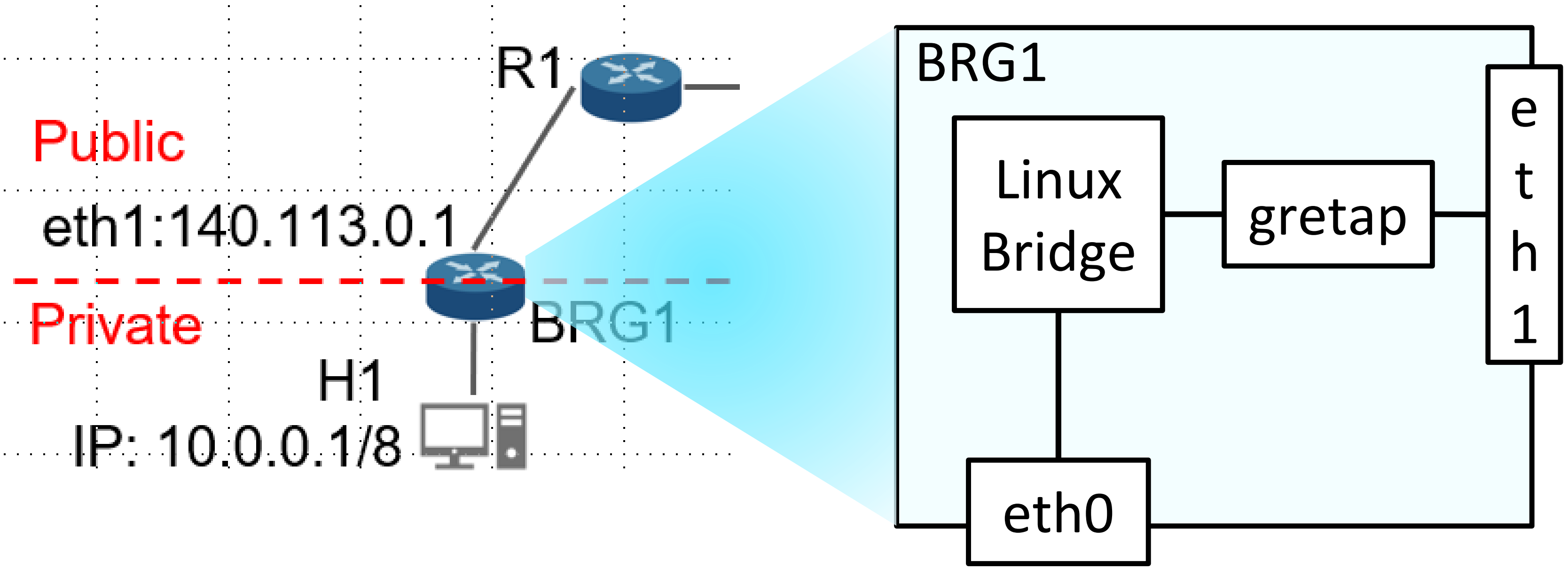
- BRG bridge network configuration
  - eth1 (WAN port) has a public IP address
  - eth0 (LAN bridge port) does not have an IP address



BRG interconnection diagram

GRETAP configuration

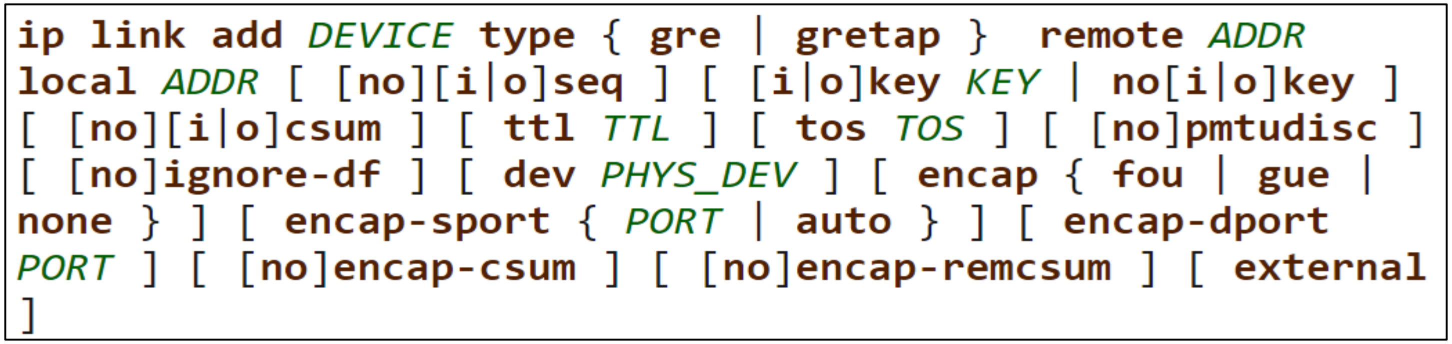
1. Create and bind a **gretap** interface to the physical interface **eth1**
2. Create a Linux Bridge and bridge **gretap** with the physical interface **eth0**



BRG interconnection diagram

GRETAP Command

- Gretap command format

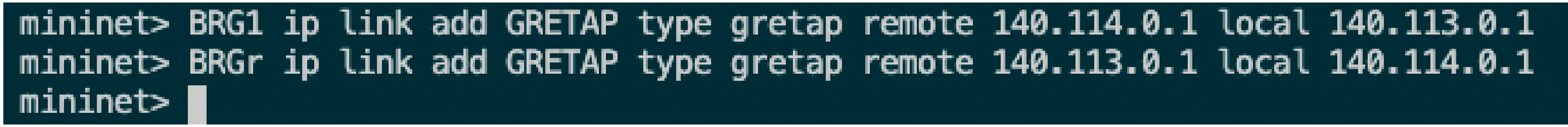


```
ip link add DEVICE type { gre | gretap }  remote ADDR
local ADDR [ [no][i|o]seq ] [ [i|o]key KEY | no[i|o]key ]
[ [no][i|o]csum ] [ ttl TTL ] [ tos TOS ] [ [no]pmtudisc ]
[ [no]ignore-df ] [ dev PHYS_DEV ] [ encap { fou | gue |
none } ] [ encap-sport { PORT | auto } ] [ encap-dport
PORT ] [ [no]encap-csum ] [ [no]encap-remcsum ] [ external
]
```

  - {}: Necessary parameter
  - []: Optional parameter

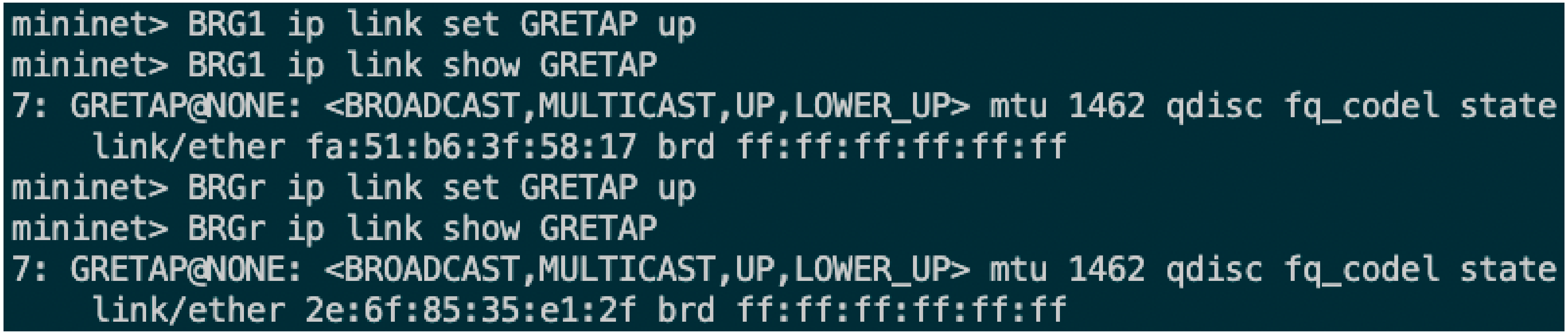
Step1: GRE Tunnel Interface Creation

1. Add a gretap interface on each of BRG1 and BRGr



```
mininet> BRG1 ip link add GRETAP type gretap remote 140.114.0.1 local 140.113.0.1
mininet> BRGr ip link add GRETAP type gretap remote 140.113.0.1 local 140.114.0.1
mininet> 
```

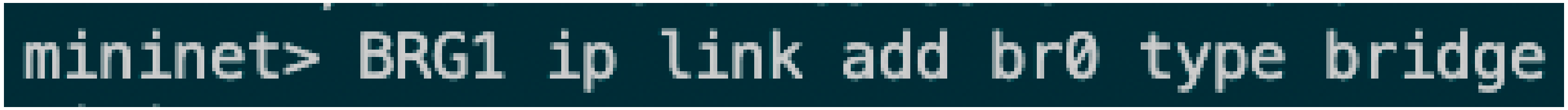
2. Bring up gretap devices



```
mininet> BRG1 ip link set GRETAP up
mininet> BRG1 ip link show GRETAP
7: GRETAP@NONE: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1462 qdisc fq_codel state
    link/ether fa:51:b6:3f:58:17 brd ff:ff:ff:ff:ff:ff
mininet> BRGr ip link set GRETAP up
mininet> BRGr ip link show GRETAP
7: GRETAP@NONE: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1462 qdisc fq_codel state
    link/ether 2e:6f:85:35:e1:2f brd ff:ff:ff:ff:ff:ff
```

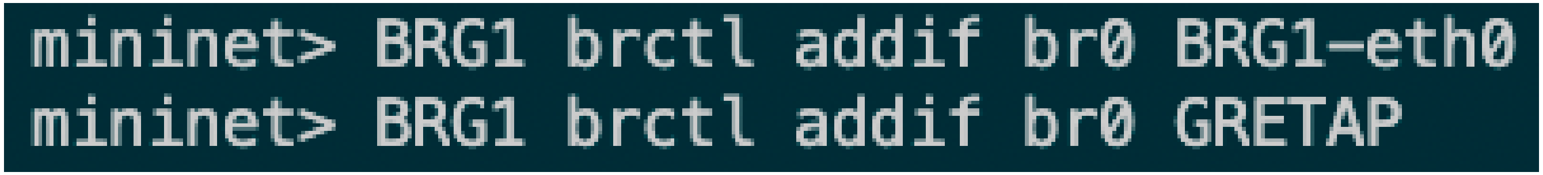
Step2: Interfaces Bridging

1. Create a Linux Bridge on BGR1



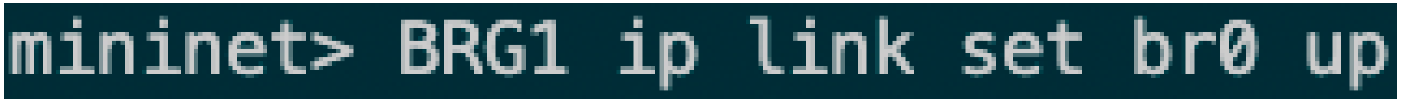
```
mininet> BRG1 ip link add br0 type bridge
```

2. Bridge interface **gretap** with **eth0**



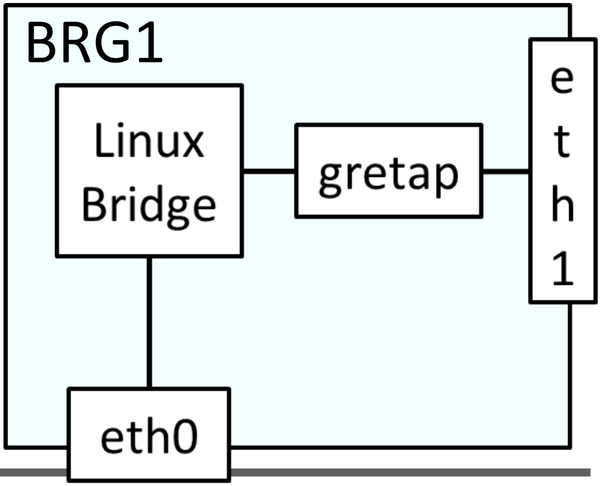
```
mininet> BRG1 brctl addif br0 BRG1-eth0
mininet> BRG1 brctl addif br0 GRETAP
```

3. Bring up Linux Bridge



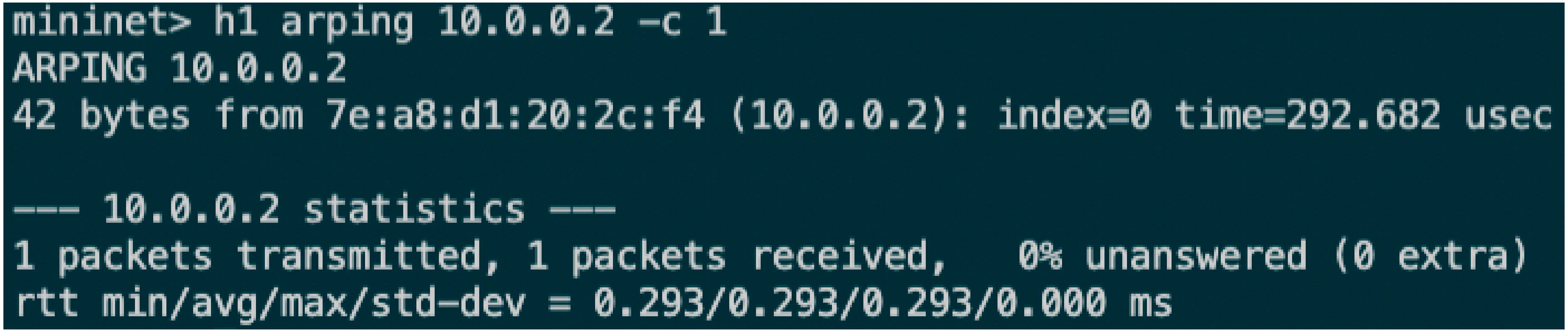
```
mininet> BRG1 ip link set br0 up
```

- Repeat same configuration on BRGr



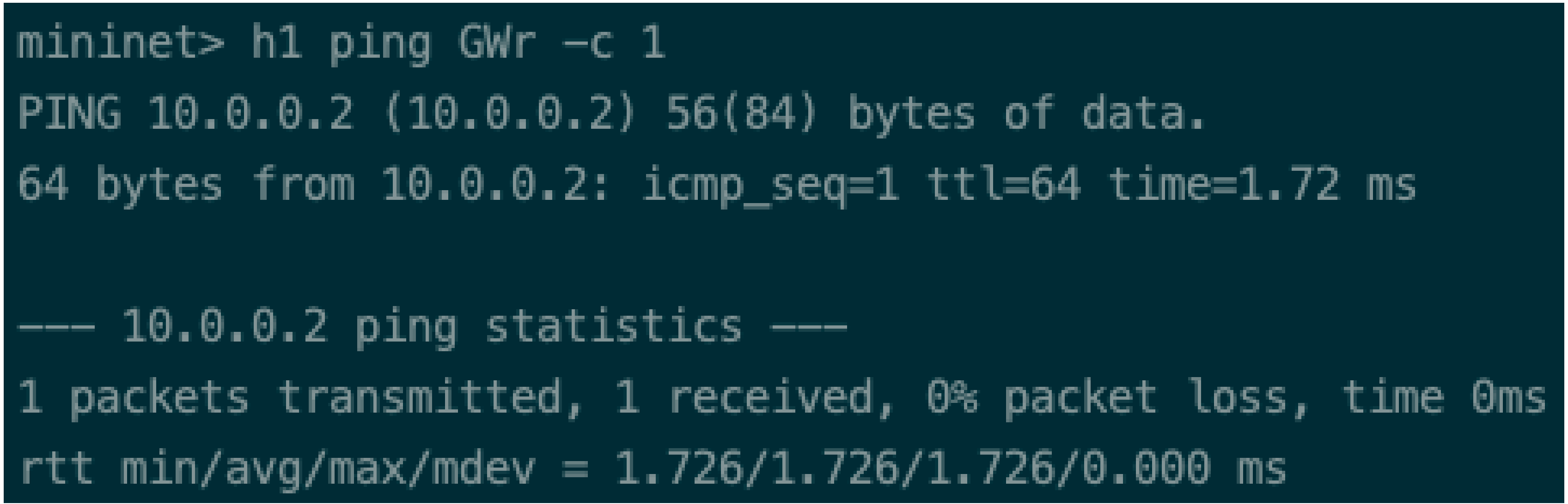
Step3: Sending Test

- H1 sends ARP request to GWr (10.0.0.2)



```
mininet> h1 arping 10.0.0.2 -c 1
ARPING 10.0.0.2
42 bytes from 7e:a8:d1:20:2c:f4 (10.0.0.2): index=0 time=292.682 usec

--- 10.0.0.2 statistics ---
1 packets transmitted, 1 packets received,   0% unanswered (0 extra)
rtt min/avg/max/std-dev = 0.293/0.293/0.293/0.000 ms
```

- H1 pings GWr



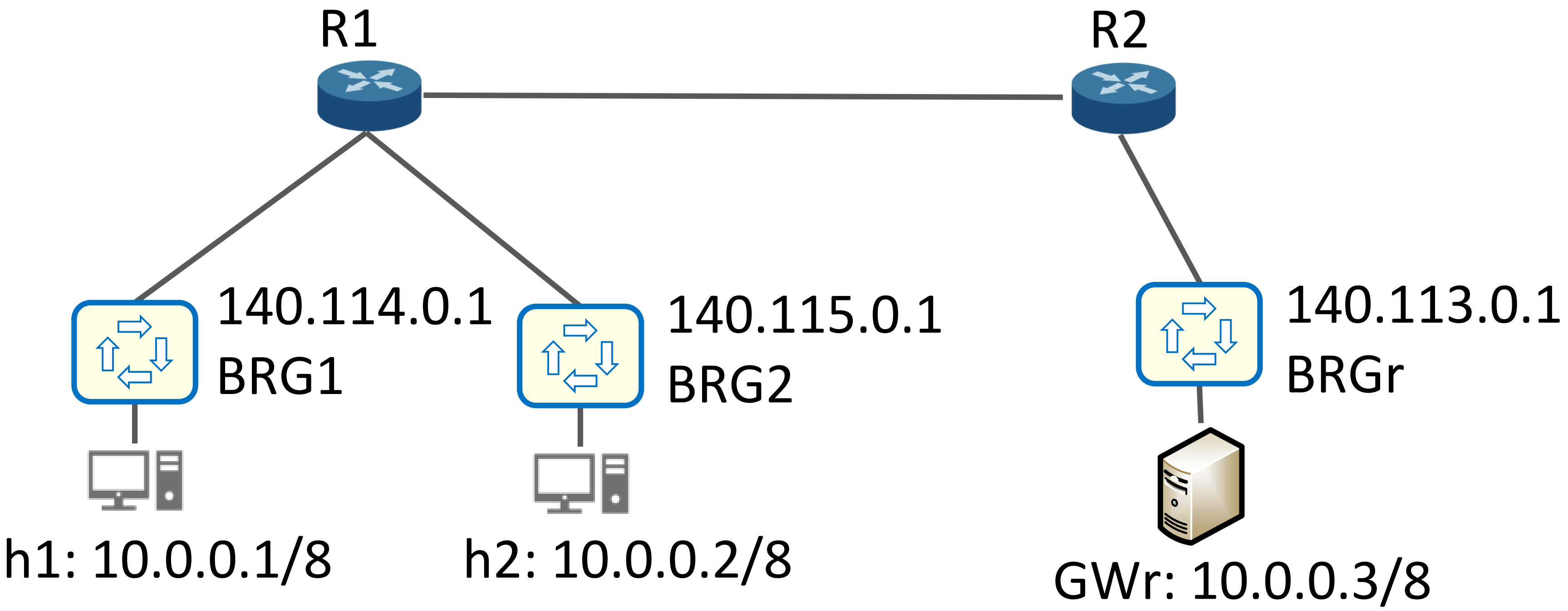
```
mininet> h1 ping GWr -c 1
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.72 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.726/1.726/1.726/0.000 ms
```

Outline

- Objective
- Environment
- Generic Routing Encapsulation tunnel (GRE tunnel)
- Lab
  - Topology
  - Tunnel Auto Creation Program
  - Requirement
- Appendix

Lab Topology

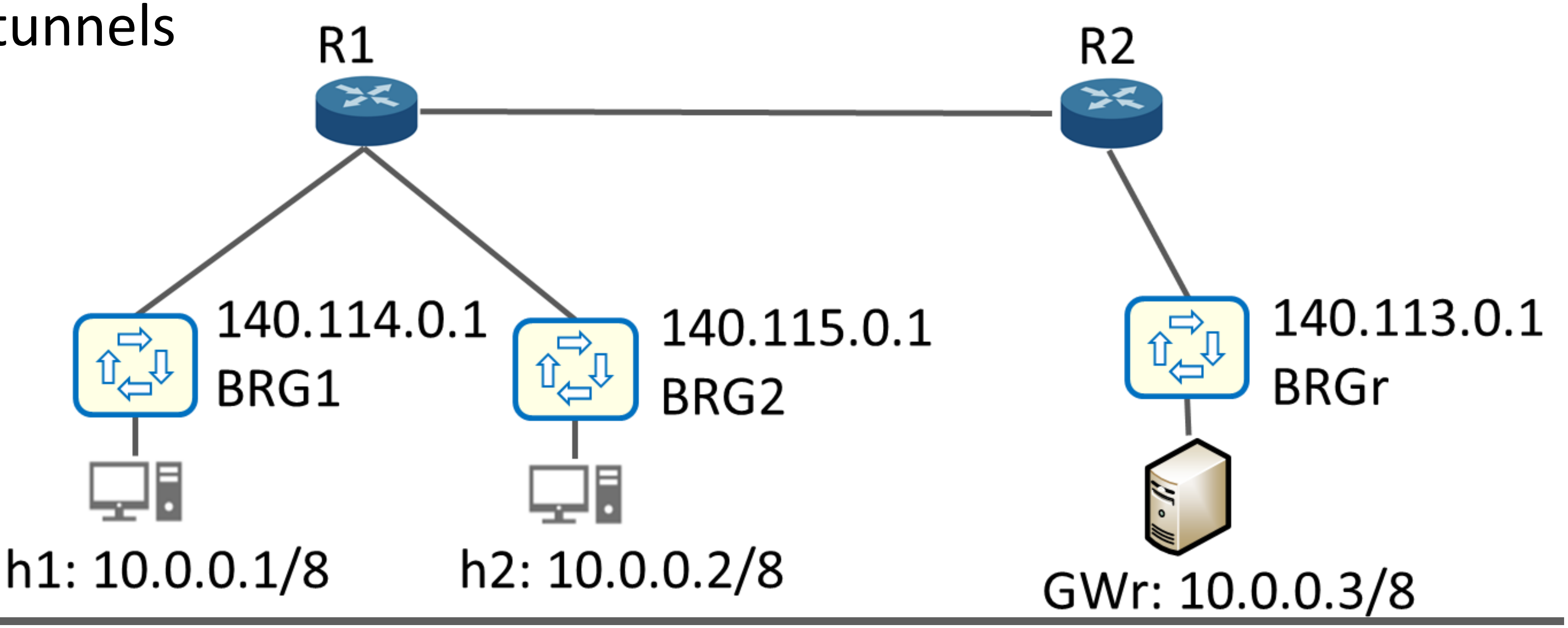
- Download topology.py from e3
- All routers/BRGs has pre-configured static routing rules
- BRG1 and BRG2 has pre-configured GRE interface
- BRGr does not have GRE interface yet



Tunnel Auto Creation Workflows

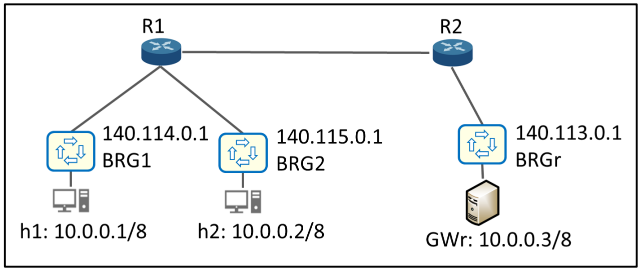
- A program running on BRGr
  1. Set BPF filter rules to capture GRE packet
  2. Parse out Outer Src/Dst IPs of incoming GRE packets to create corresponding GRE interface
  3. Update BPF filter rules

     Stops packet capturing and parsing for established GRE tunnels



Tunnel Auto Creation Program

- Write a **C/C++/Golang** program with **pcap** library to create GRE Tunnel dynamically
  - C/C++: libpcap.c
    - Compile with gcc/g++ <code>.c -lpcap to use pcap library
  - Golang: Gopacket
- Execute your program on node BRGr
  - This program should be able to
    - Show all Interfaces on BRGr
    - Select an interface on BRGr to capture packets
    - Parse packets and create corresponding GRE tunnel interface on BRGr



Demo: Program Function and Architecture (1/2)

1. Show Interface list after program starts (5%)
2. Select an interface to capture packet with a UI (5%)
3. Packet filtering
    - Input basic BPF Filtering expression with a UI (5%)
    - Print byte codes of all captured packets in Hexadecimal (5%)
    - Efficiency of packet filtering and processing (20%)
        - Use BPF to filter packets
        - Dynamic update BPF filtering expression
            - Minimize the number of packets captured by BPF and processed in user space

Demo: Program Function and Architecture (2/2)

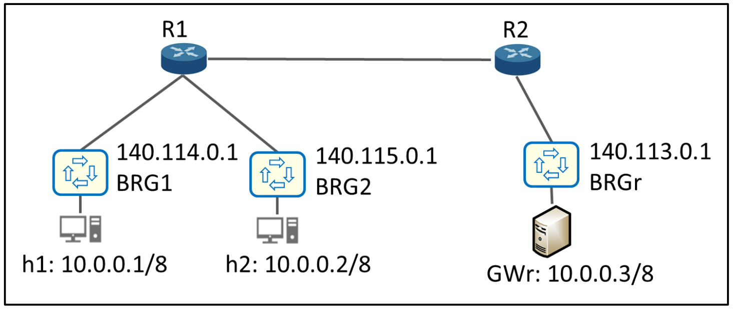
4. Show packet parsing result (10%)
    - Outer Ethernet Header: MAC address and ether type (Hexadecimal)
    - Outer IP Header: Source and Destination IP (Decimal)
    - GRE Header: Protocol types
    - Inner Ethernet Header: MAC address and ether type (Hexadecimal)
5. Correctness of tunnel creation
    - Reachability among hosts (10%)

Report: Answer Questions (1/2)

1. Show the ping results to test reachability (5%)
   a) h1 and h2 ping GWr
2. Show all interfaces of Node **BRGr** after h1 and h2 can ping GWr(5%)
3. Draw the **interconnection diagram** of interfaces and Linux bridge on **BRGr**. Explain your diagram with the screenshot of interface list of BRGr. (10%)
4. **Explain how** Linux kernel of BRGr determines which **gretap** interface to forward packets from GWr to hosts (h1 or h2)?

Describe your answer with appropriate screenshot. (10%)

Hint: MAC Learning



Report: Answer Questions (2/2)

5. Run tcpdump on h1 to capture packet and take screenshot to explain why or why not h1 is aware of GRE tunneling. (10%)
    - Run tcpdump on h1 to capture ICMP packet received by h1
    - h1 pings GWr

        -
          ```
          mininet> h1 ping GWr -c 1
          ```
    - Show screenshot and explain your answer.

Report Submission

- Files
  - <studentID>.c/cpp/go (60%, with Demo)
  - A Report: lab4_<studentID>.pdf (40%)
- Submission
  - Zip all files into a **zip** file
    - Name: lab4_<studentID>.zip
- Wrong filename or format subjects to score deduction (-5%)

Outline

- Objective
- Environment
- Generic Routing Encapsulation tunnel (GRE tunnel)
- Lab requirements
- Appendix

Appendix

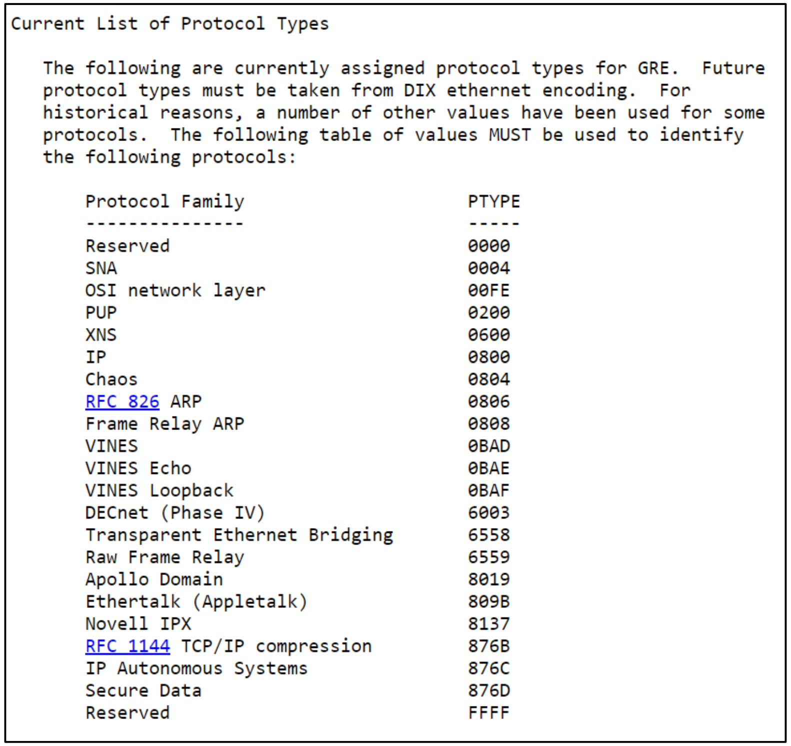
- ip link man page
  - https://man7.org/linux/man-pages/man8/ip-link.8.html
- Golang installation
  - https://golang.org/doc/install
- Golang Basic
  - https://www.openmymind.net/assets/go/go.pdf
- Gopcaket
  - https://github.com/google/gopacket

Appendix

- Libpcap.c function
  - pcap_findalldevs
    - https://man7.org/linux/man-pages/man3/pcap_findalldevs.3pcap.html
  - pcap_open_live
    - https://linux.die.net/man/3/pcap_open_live
  - pcap_compile
    - https://linux.die.net/man/3/pcap_compile
  - pcap_setfilter
    - https://man7.org/linux/man-pages/man3/pcap_setfilter.3pcap.html
  - pcap_loop
    - https://linux.die.net/man/3/pcap_loop
- BPF Filter expression
    - https://linux.die.net/man/7/pcap-filter

Appendix

- RFC 2784: GRE protocol
  - https://tools.ietf.org/html/rfc2784
- GRE protocol family type
  - https://tools.ietf.org/html/rfc1701

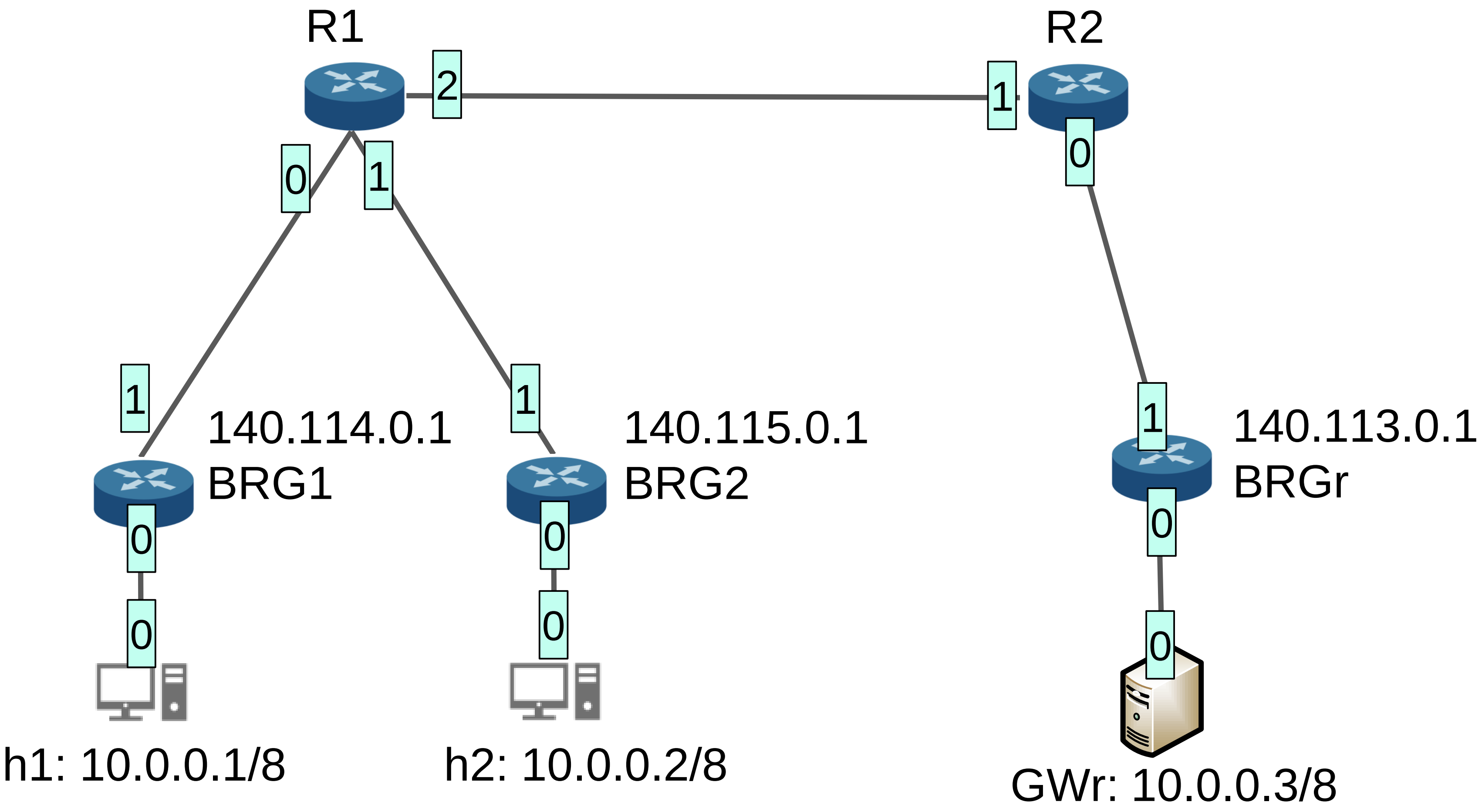


Current List of Protocol Types

The following are currently assigned protocol types for GRE. Future protocol types must be taken from DIX ethernet encoding. For historical reasons, a number of other values have been used for some protocols. The following table of values MUST be used to identify the following protocols:

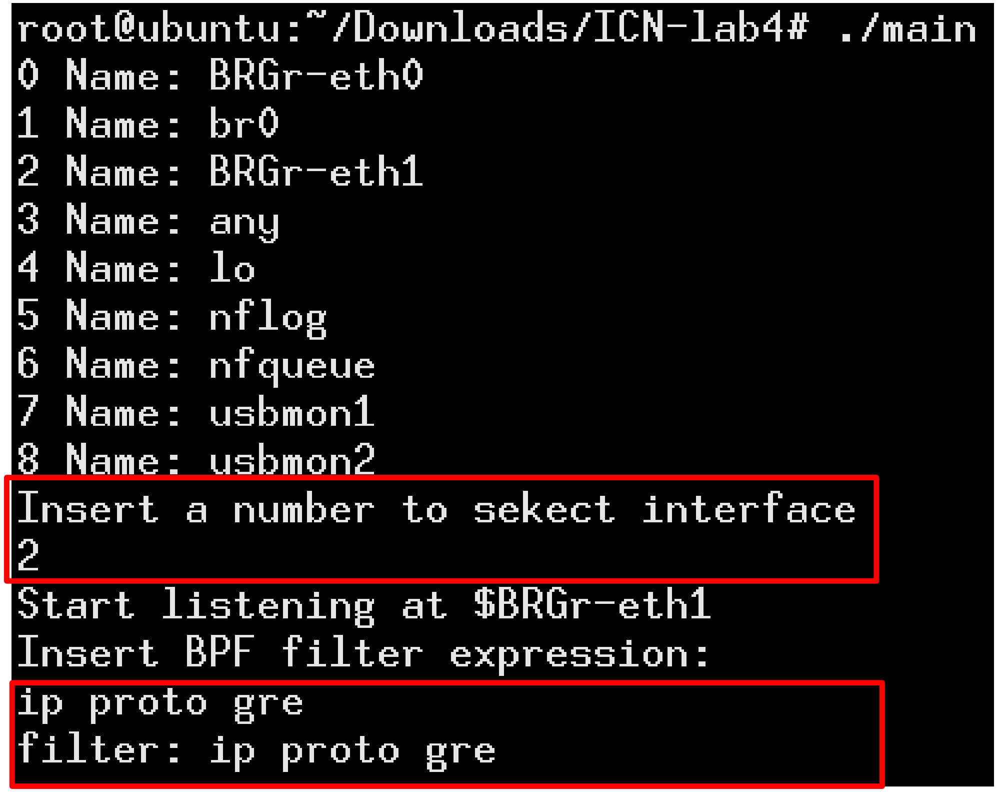
Protocol Family	PTYPE
Reserved	0000
SNA	0004
OSI network layer	00FE
PUP	0200
XNS	0600
IP	0800
Chaos	0804
RFC 826 ARP	0806
Frame Relay ARP	0808
VINES	0BAD
VINES Echo	0BAE
VINES Loopback	0BAF
DECnet (Phase IV)	6003
Transparent Ethernet Bridging	6558
Raw Frame Relay	6559
Apollo Domain	8019
Ethertalk (Appletalk)	809B
Novell IPX	8137
RFC 1144 TCP/IP compression	876B
IP Autonomous Systems	876C
Secure Data	876D
Reserved	FFFF

Lab Topology



Program Example (1/2)

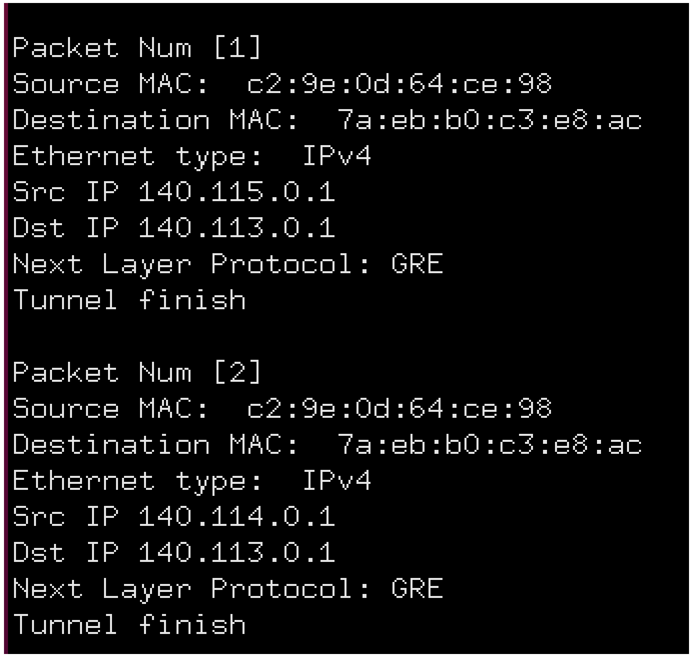
- Interface List and Selection
- Basic BPF filter expression



```
root@ubuntu:~/Downloads/ICN-lab4# ./main
0 Name: BRGr-eth0
1 Name: br0
2 Name: BRGr-eth1
3 Name: any
4 Name: lo
5 Name: nflog
6 Name: nfqueue
7 Name: usbmon1
8 Name: usbmon2
Insert a number to sekect interface
2
Start listening at $BRGr-eth1
Insert BPF filter expression:
ip proto gre
filter: ip proto gre
```

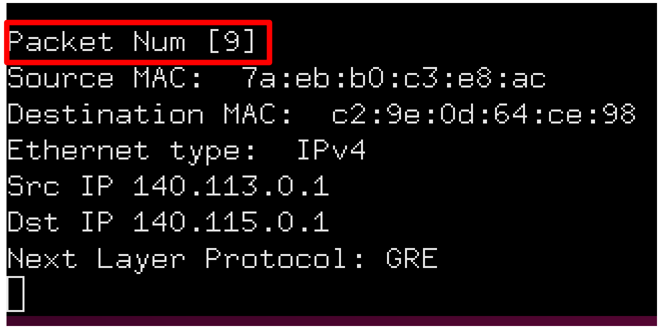
Program Example (2/2)

- Show parsing result

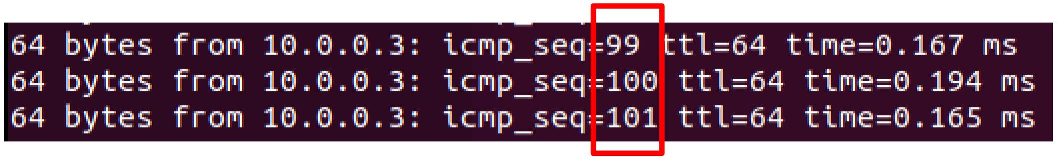


```
Packet Num [1]
Source MAC:  c2:9e:0d:64:ce:98
Destination MAC:  7a:eb:b0:c3:e8:ac
Ethernet type:  IPv4
Src IP 140.115.0.1
Dst IP 140.113.0.1
Next Layer Protocol: GRE
Tunnel finish

Packet Num [2]
Source MAC:  c2:9e:0d:64:ce:98
Destination MAC:  7a:eb:b0:c3:e8:ac
Ethernet type:  IPv4
Src IP 140.114.0.1
Dst IP 140.113.0.1
Next Layer Protocol: GRE
Tunnel finish
```

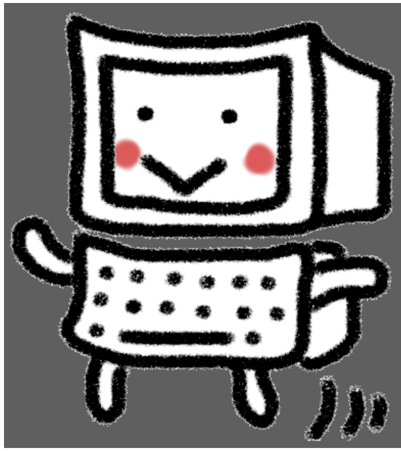
- Update BPF filter



```
Packet Num [9]
Source MAC:  7a:eb:b0:c3:e8:ac
Destination MAC:  c2:9e:0d:64:ce:98
Ethernet type:  IPv4
Src IP 140.113.0.1
Dst IP 140.115.0.1
Next Layer Protocol: GRE
```



```
64 bytes from 10.0.0.3: icmp_seq=99 ttl=64 time=0.167 ms
64 bytes from 10.0.0.3: icmp_seq=100 ttl=64 time=0.194 ms
64 bytes from 10.0.0.3: icmp_seq=101 ttl=64 time=0.165 ms
```



Q & A