Lab4

Dynamic Tunnel Creation

Deadline: 2022/04/11 (MON) 23:59

Outline

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

Objective

- GRE Tunnel Configuration and Observation
  - Inner and outer headers of packets
- Dynamic Tunnel Creation
  - Write an Auto Tunnel Creation Program in C/C++/Golang to
    - Filter and parse incoming encapsulated packets
    - Create tunnel automatically with the parsed result

Outline

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

Lab Environment and Languages

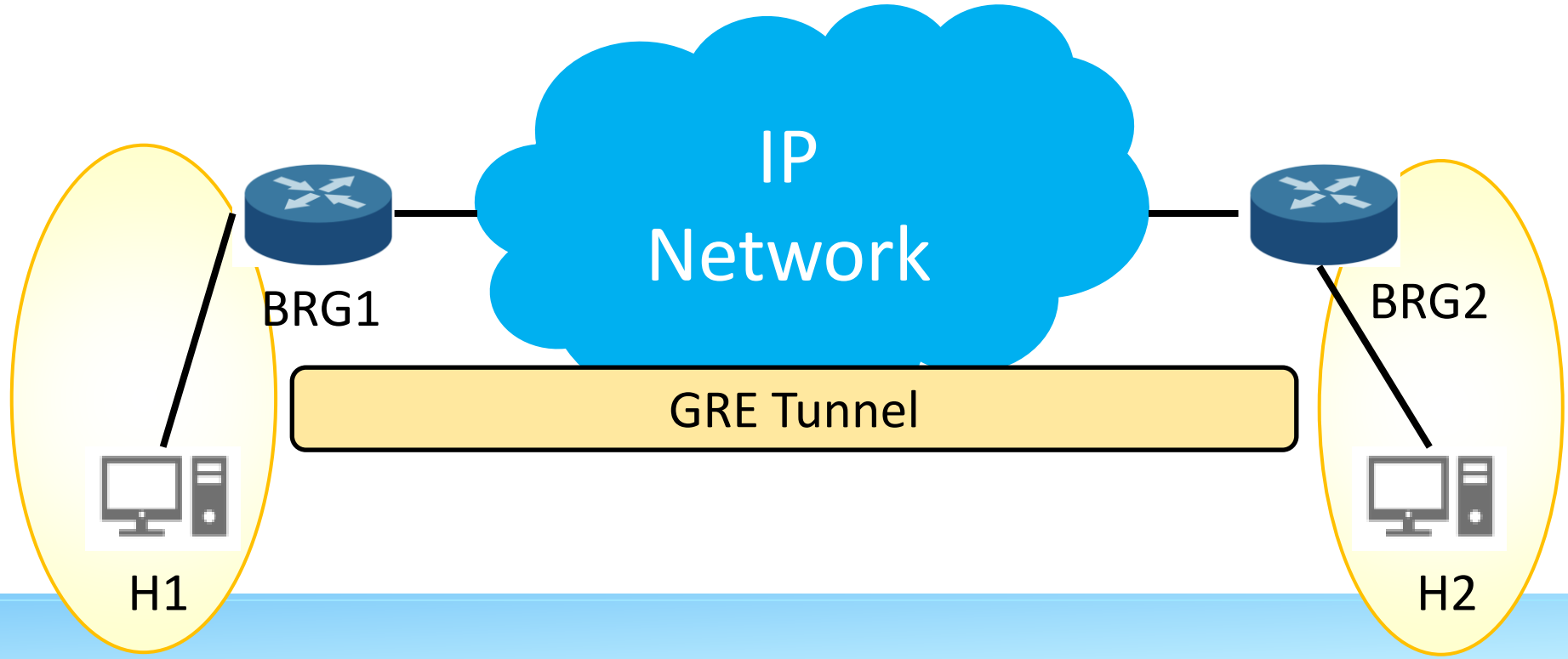
- **Environment**: same as Lab3
  - Ubuntu 16.04
  - Docker
- Language and Compiler
  - GCC/G++
  - Golang
    - Latest version
    - https://golang.org/doc/install

Outline

- Objective
- Lab Environment
- Generic Routing Encapsulation Tunnel (GRE Tunnel)
  - Overview
    - GRE Headers
    - Tunneling Workflows
  - Example Topology Configuration and Testing
- Lab Requirements
- Appendix

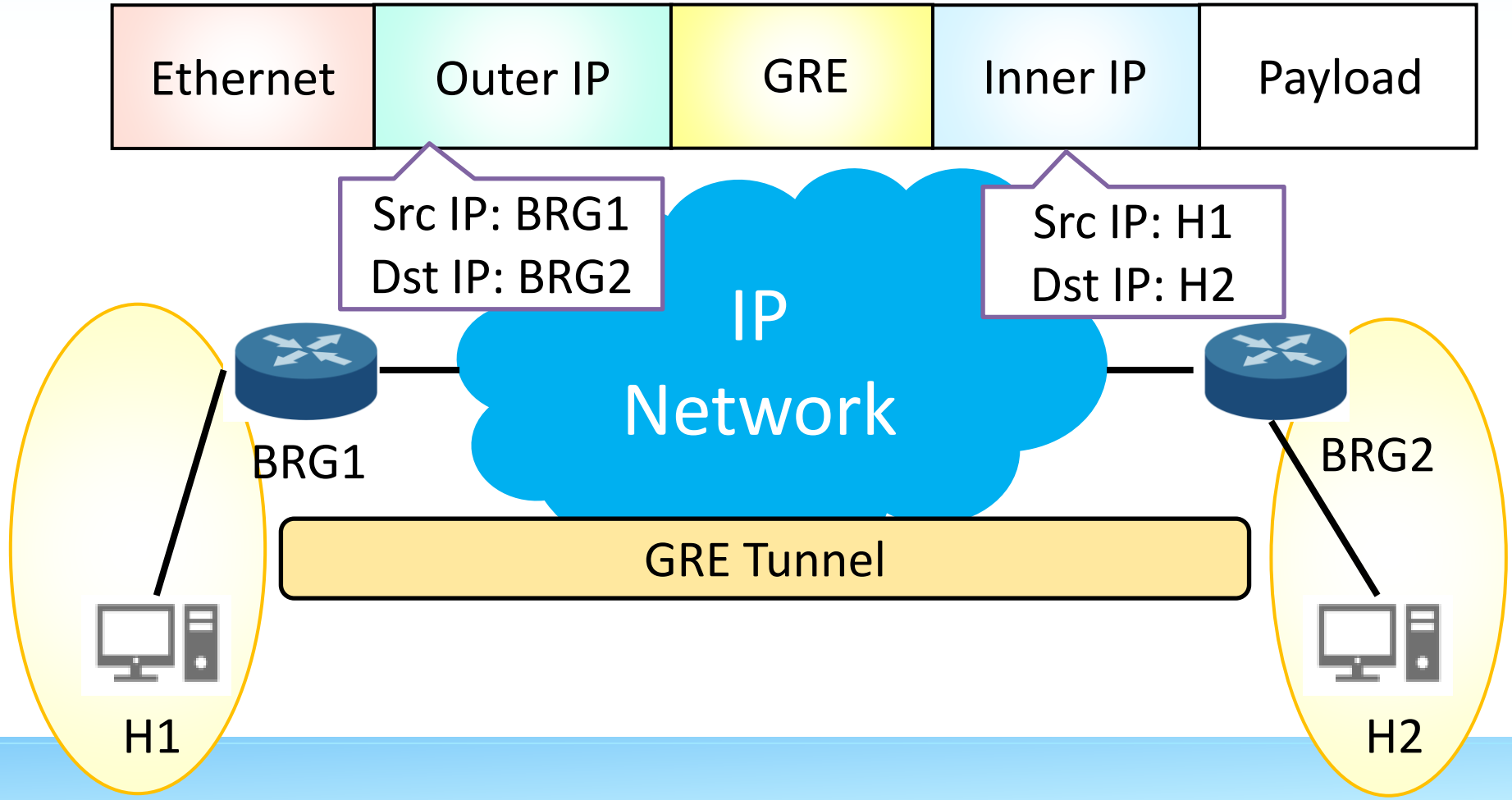
GRE Tunnel and Virtual LANs

- 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

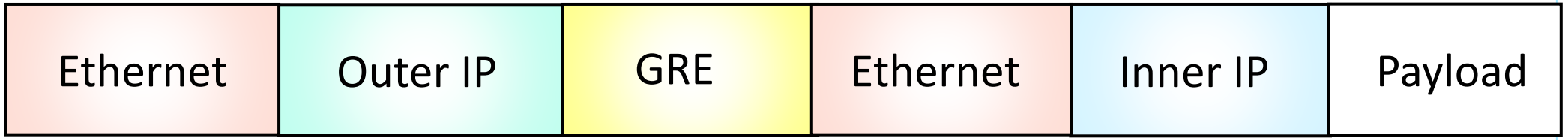
- An IP in IP tunneling protocol
  - Outer IPs help forward packets to remote LANs across Internet



Types of GRE Tunnels

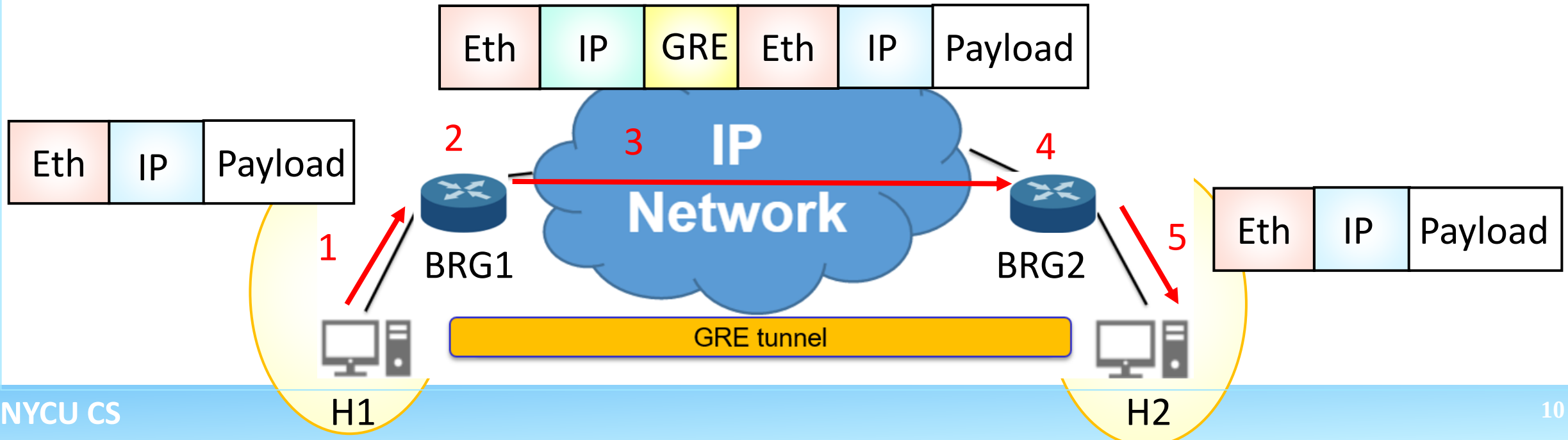
- GRE

- GRETAP



GRETAP Tunneling Workflows

1. H1 sends a packet to H2
2. BRG1 receives and encapsulates the packet with GRETAP
3. BRG1 forwards the GRETAP encapsulated packet to the remote BRG (BRG2)
4. BRG2 receives and decapsulates the packet
5. BRG2 forwards the original packet to H2, based on normal MAC forwarding

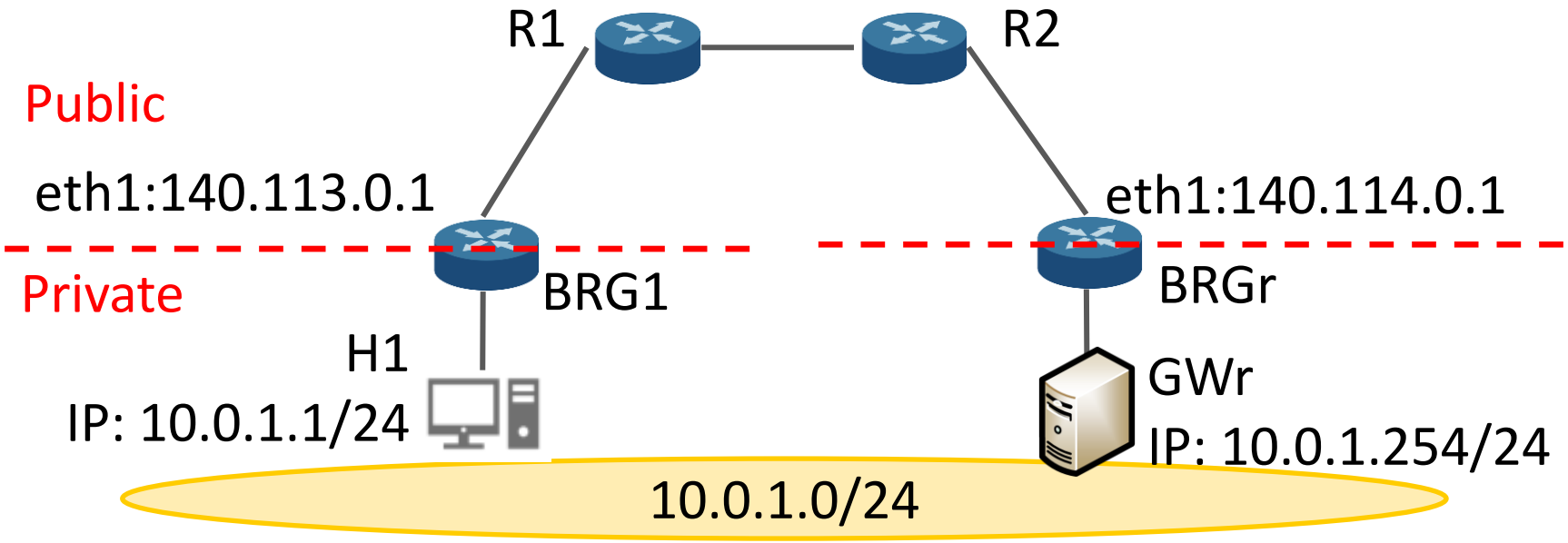


Outline

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

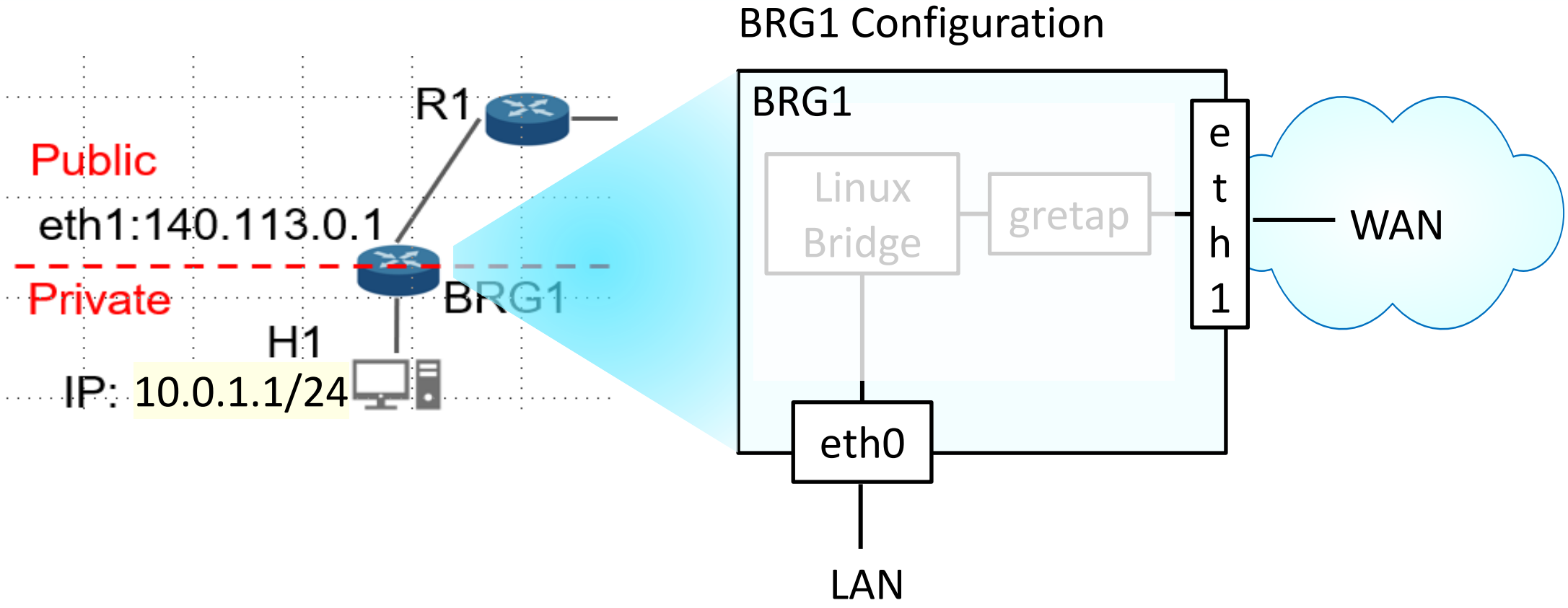
Example Topology

- 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 GRETAP Tunnel
- H1 uses GWr as the default gateway



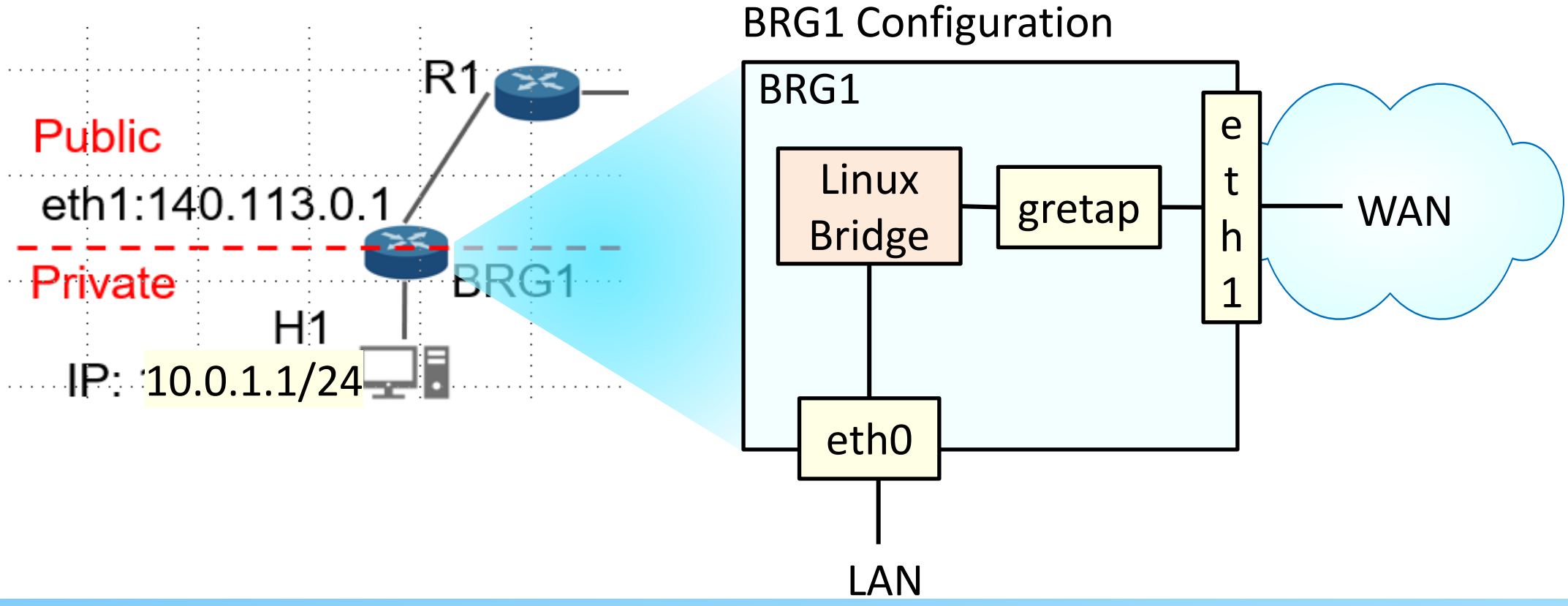
BRG1 Network Configuration

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



BRG1 GRETAP Configuration

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



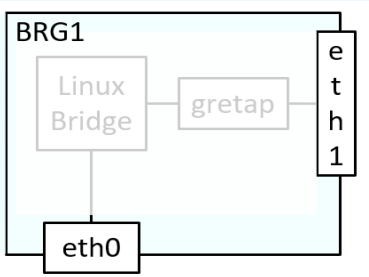
GRETAP Command

- GRETAP command syntax

```
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 parameters
  - []: Optional parameters

Step1: GRE Tunnel Interface Creation



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

```
lab4$ docker exec BRG1 ip link add GRETAP type gretap remote 140.114.0.1 local 140.113.0.1
lab4$ docker exec BRGr ip link add GRETAP type gretap remote 140.113.0.1 local 140.114.0.1
```

2. Bring up gretap devices in BRG1 and BRGr, respectively

```
andy78644@andy78644:~/ta/lab4$ docker exec BRG1 ip link set GRETAP up
andy78644@andy78644:~/ta/lab4$ docker exec BRG1 ip link show GRETAP
5: GRETAP@NONE: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1462 qdisc fq_codel
    state UNKNOWN mode DEFAULT group default qlen 1000
    link/ether a2:ec:46:e5:85:31 brd ff:ff:ff:ff:ff:ff
```

```
andy78644@andy78644:~/ta/lab4$ docker exec BRGr ip link set GRETAP up
andy78644@andy78644:~/ta/lab4$ docker exec BRGr ip link show GRETAP
5: GRETAP@NONE: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1462 qdisc fq_codel state UNKNOWN mode DEFAULT group default qlen 1000
    link/ether a6:b1:c6:6b:df:39 brd ff:ff:ff:ff:ff:ff
```

Step2: Interfaces Bridging

1. Create a Linux Bridge on BGR1

```
lab4$ docker exec BRG1 ip link add br0 type bridge
```

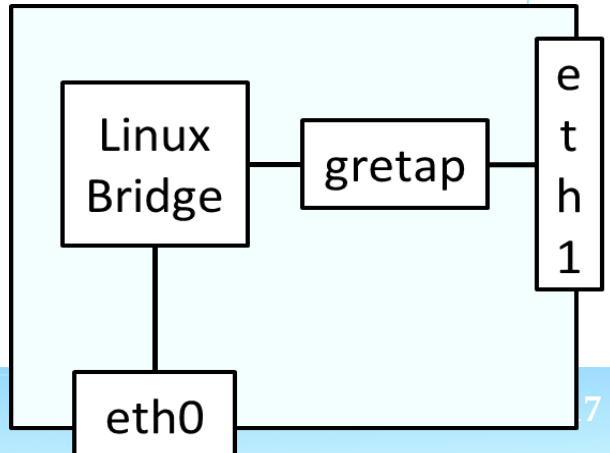
2. Bridge interface **gretap** and **eth0** with br0

```
lab4$ docker exec BRG1 ip link set eth0 master br0
lab4$ docker exec BRG1 ip link set GRETAP master br0
```

3. Bring up Linux Bridge

```
lab4$ docker exec BRG1 ip link set br0 up
```

- Repeat same steps for BRGr



Step3: Connectivity Test

- H1 sends ARP request to GWr (10.0.0.2)

```
root@2312c904fefd:/# arping 10.0.1.254 -c 1
ARPING 10.0.1.254
42 bytes from 32:b0:0f:37:a3:30 (10.0.1.254): index=0 time=8.519 msec

--- 10.0.1.254 statistics ---
1 packets transmitted, 1 packets received,   0% unanswered (0 extra)
rtt min/avg/max/std-dev = 8.519/8.519/8.519/0.000 ms
```

- H1 pings GWr

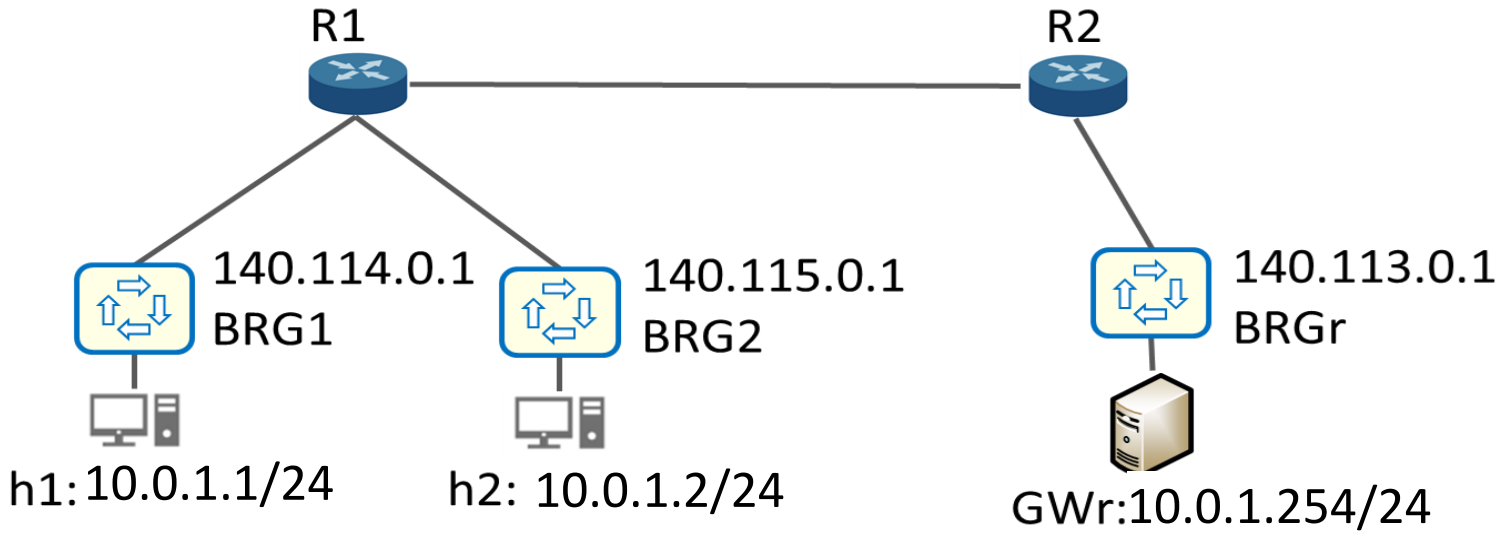
```
root@2312c904fefd:/# ping 10.0.1.254 -c 1
PING 10.0.1.254 (10.0.1.254) 56(84) bytes of data.
64 bytes from 10.0.1.254: icmp_seq=1 ttl=64 time=0.104 ms

--- 10.0.1.254 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.104/0.104/0.104/0.000 ms
```

Outline

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

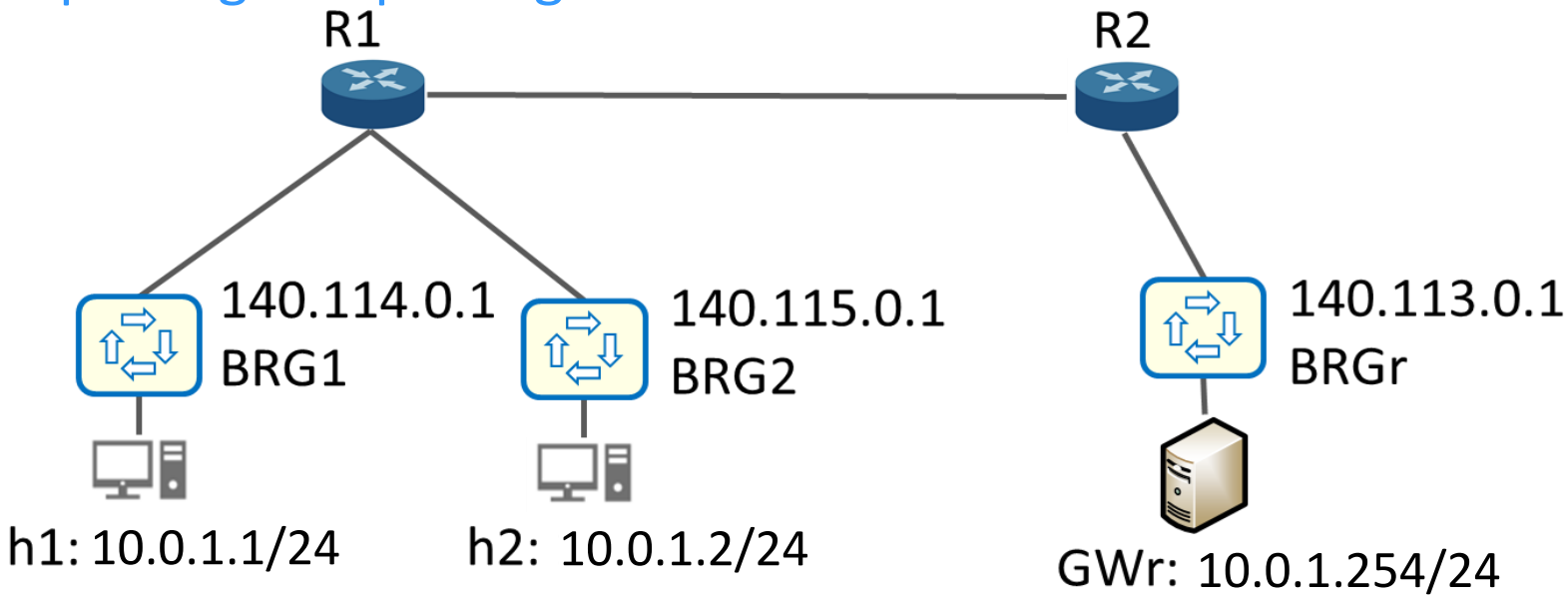
Lab Topology

- Part1: static GRETAP tunnel creation
  - Complete all routers/BRGs static routing rules
  - Complete all BRGs static GRETAP interfaces
- Part2: dynamic GRETAP tunnel creation
  - Create a static bridge on BRGr
  - Create BRGr GRETAP interfaces dynamically



Tunnel Auto Creation Workflows

- Write and run a tunnel creation program on BRGr
  - Set BPF filter rules to capture GRE packets
  - Parse out Outer Src/Dst IPs of incoming GRE packets
  - Create corresponding GRE interface with Outer Src/Dst IPs
  - Update BPF filter rules
  - Stops packet capturing and parsing for established GRE tunnels



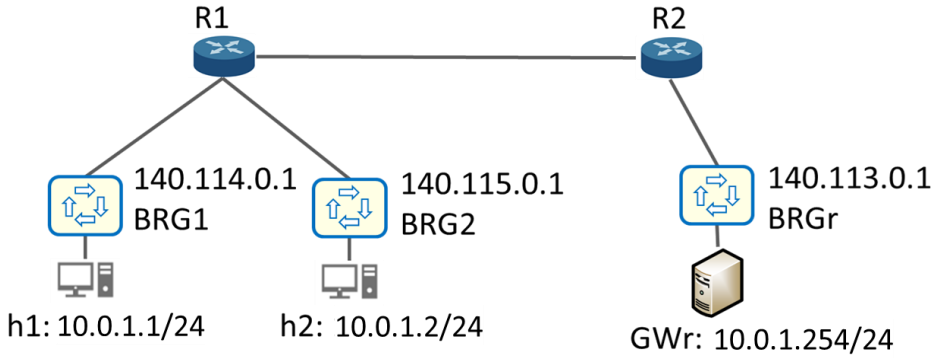
Tunnel Auto Creation Program

- Write a C/C++/Golang program with pcap library to create GRE Tunnel dynamically
  - C/C++: libpcap.c
    - Compile your program with “lpcap” option to use pcap library

      > gcc/g++ <code>.c –lpcap
  - 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

Report: Answer Question (1/2)

- Part1:
  1. Show all interfaces of node BRGr and draw the interconnection diagram of interfaces and Linux bridge on BRGr. Explain your diagram with the interface list of BRGr. (10%)
  2. Let h1 and h2 ping GWr and take screenshot of ping results. (5%)
  3. Can h1 ping h2? Take screenshots to explain why or why not (10%)
     - Briefly explain the route the packets go through
     - Run tcpdump on nodes to capture ICMP packets received or sent by each node
     - h1 pings h2



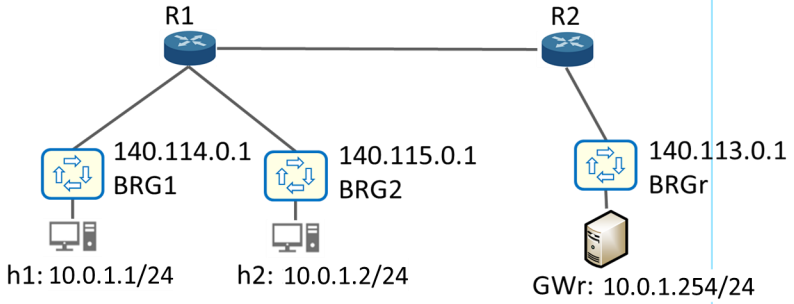
Report: Answer Question (2/2)

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 screenshots. (5%)

5. Is h1 aware of GRE tunneling? Take screenshot to explain why or why not (5%)
   - Run tcpdump on h1 and BRG nodes to capture ICMP packets
   - Let h1 ping GWr and show tcpdump results to justify your answer

- Part2:

  Run Tunnel Auto Creation Program on BRGr and show the parsed result of one captured packet. (5%)

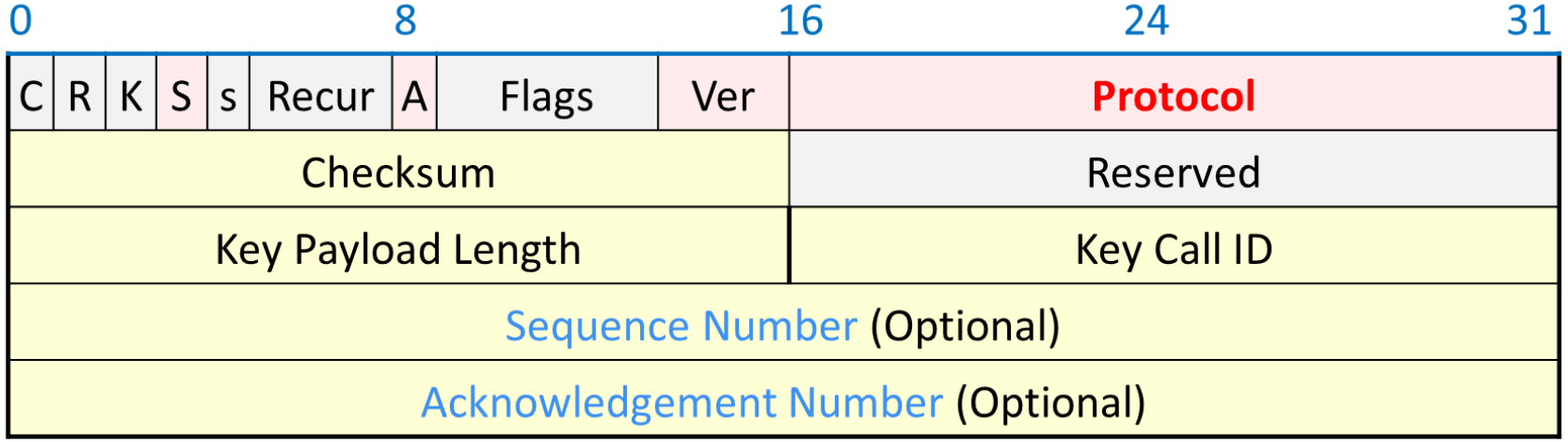


Demo: Program Function and Architecture (1/2)

1. Show Interfaces on BRGr before and after Tunnel Auto Creation program starts (5%)
2. Input the interface of BRGr on which we want to capture packets (5%)
3. Packet filtering functionality test
   - Input basic BPF Filtering expression (5%)
   - Print byte codes of all captured packets in Hexadecimal (5%)
   - Show the efficiency of packet filtering and processing. (20%)
     - You program must **update BPF filtering expression** dynamically
       - To minimize the number of packets captured by BPF and processed in userspace

Demo: Program Function and Architecture (2/2)

4. Show packet parsed result (10%)
   - Outer Ethernet Header: MAC address and Ether Type (Hexadecimal)
   - Outer IP Header: Source and Destination IP (Decimal)
   - GRE Header: **Protocol** (Compliant with Ethertype)
   - Inner Ethernet Header: MAC address and ether type (Hexadecimal)



5. Correctness of tunnel creation
   - Reachability among hosts (10%)

Submission

- Files
  - Code (60%, with DEMO)
    - <studentID>.c/cpp/go
  - Report (40%)
    - lab4_<studentID>.pdf
- Submission
  - Zip all file into a zip file
    - Name: lab4_<studentID>.zip
- Wrong filename or format subjects to 10 points deduction

Outline

- Objective
- Lab 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
- Gopacket
  - https://github.com/google/gopacket

Appendix

- Libpcap.c function
  - pcap_findalldevs
    - https://man7.org/linux/manpages/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/manpages/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

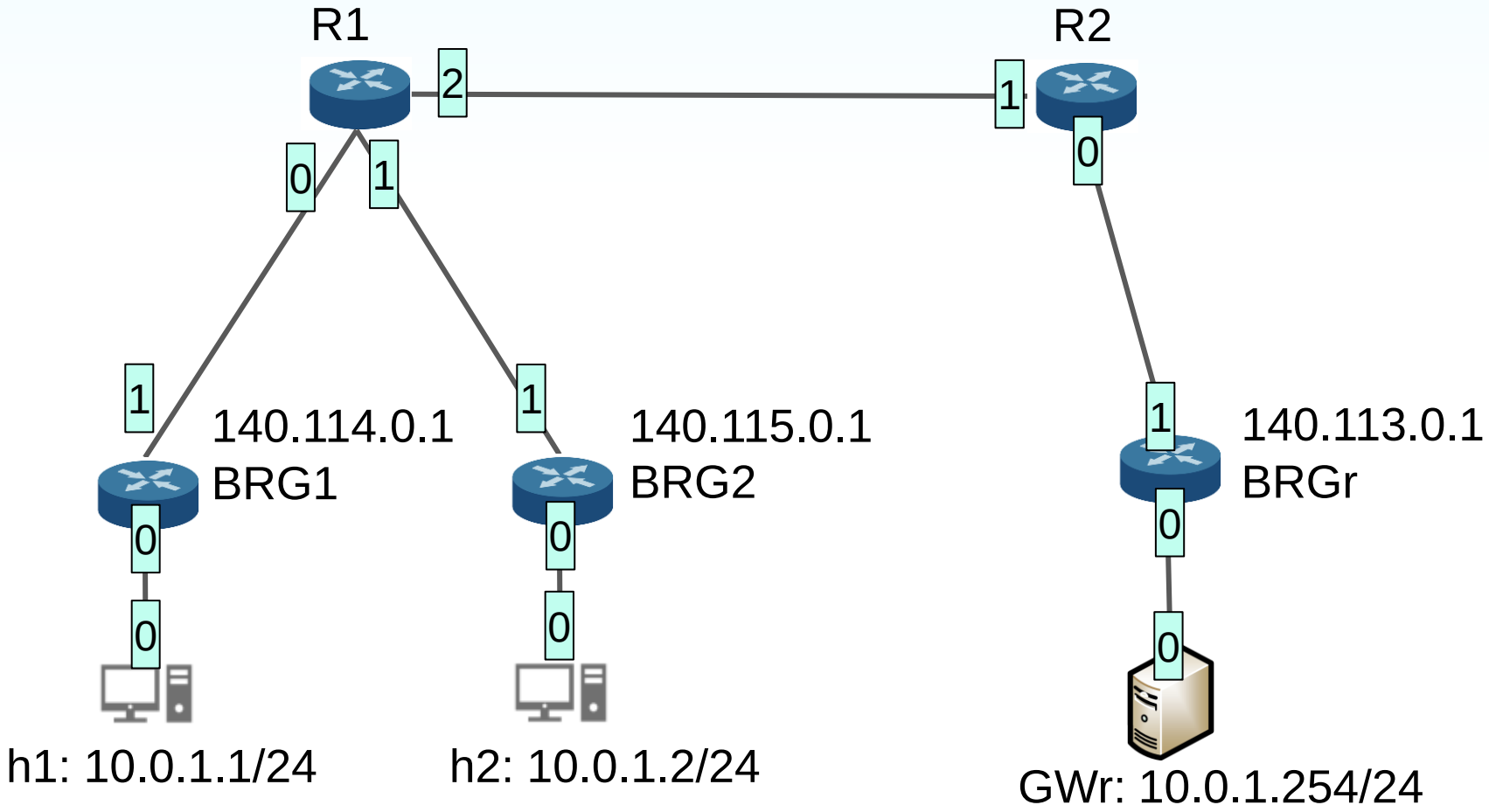
- RFC 2784: GRE protocol
  - https://tools.ietf.org/html/rfc2784
- RFC 1701: GRE protocol family type
  - https://tools.ietf.org/html/rfc1701

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)

- List interfaces
- Interactive inputs:
  - Interface selection
  - 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
```

- h1 and h2 ping GWr

```
root@a9462707fb86:/# ping 10.0.1.254
PING 10.0.1.254 (10.0.1.254) 56(84) bytes of data.
64 bytes from 10.0.1.254: icmp_seq=1 ttl=64 time=0.195 ms
64 bytes from 10.0.1.254: icmp_seq=2 ttl=64 time=0.079 ms
```

Program Example (2/2)

- Parse captured packets
- Create Tunnel
- Update BPF filter (for the existing tunnel)
  - To stop packet capturing

Number of packets sent by h1 to GWr

```
64 bytes from 10.0.1.254: icmp_seq=99 ttl=64 time=0.263 ms
64 bytes from 10.0.1.254: icmp_seq=100 ttl=64 time=0.257 ms
64 bytes from 10.0.1.254: icmp_seq=101 ttl=64 time=0.209 ms
```

Number of Packets received

```
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
```

Parsed 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
```

Q & A