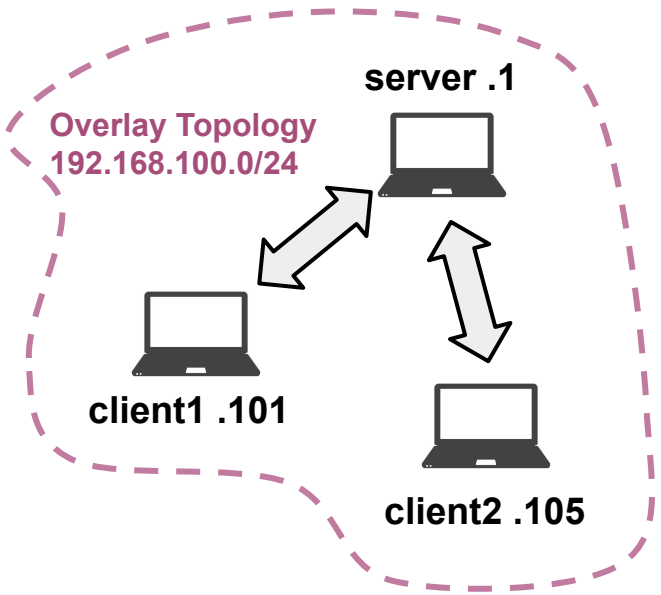
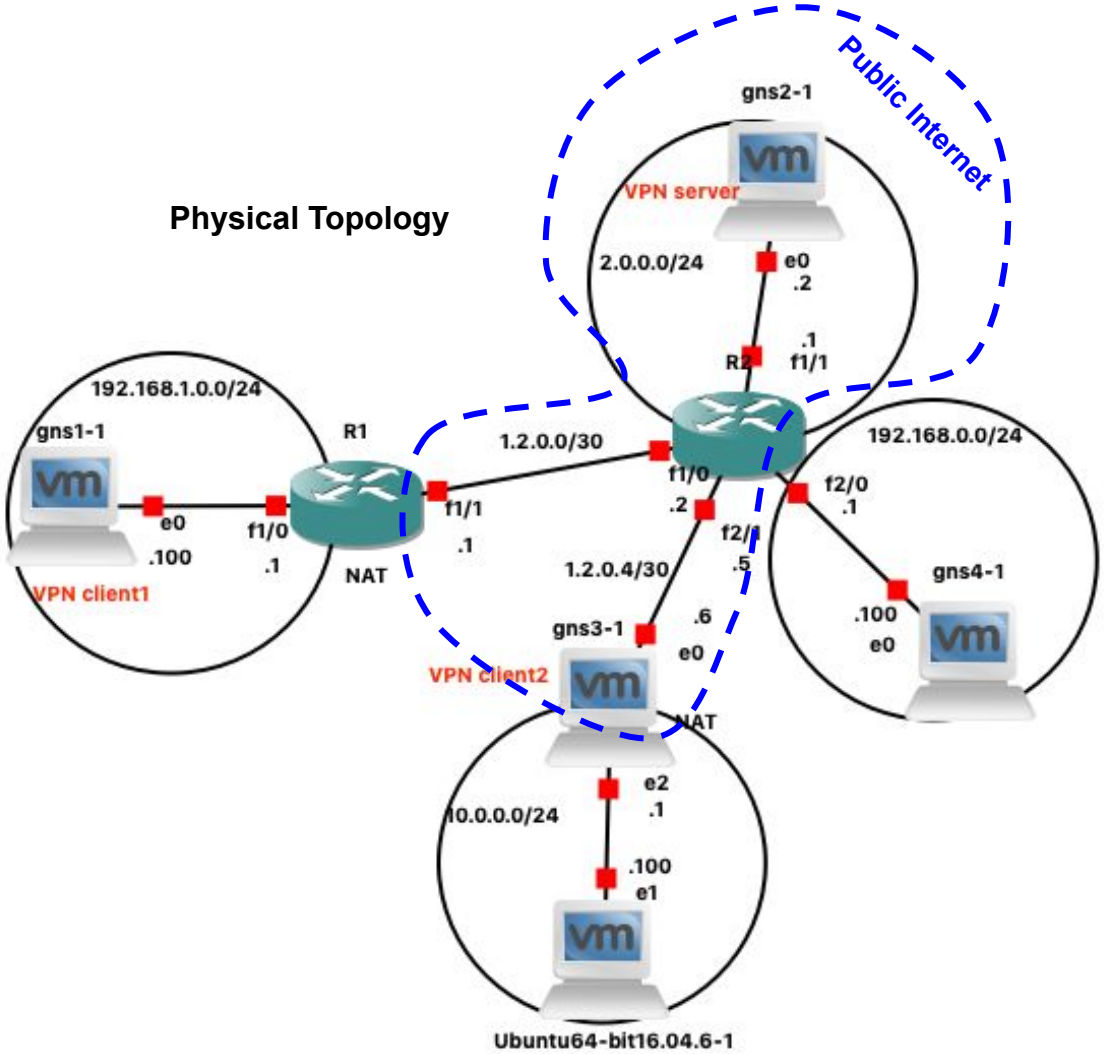
Lab 8: Overlay VPN with OpenVPN

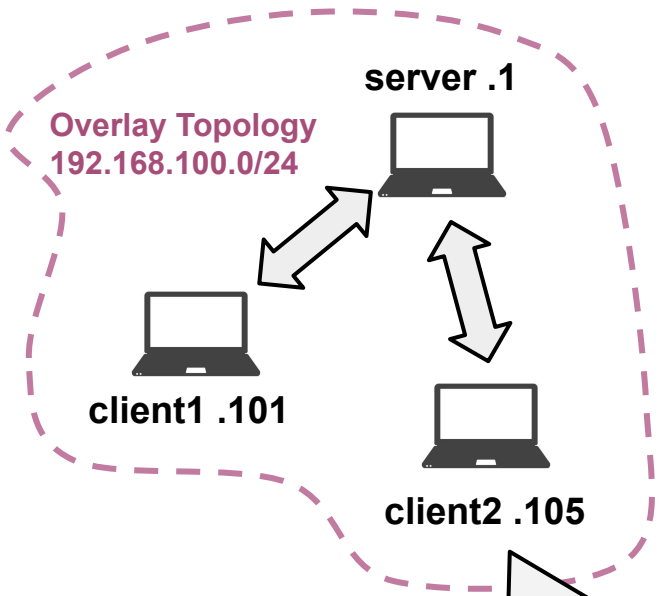
Lab Topology



Physical Topology

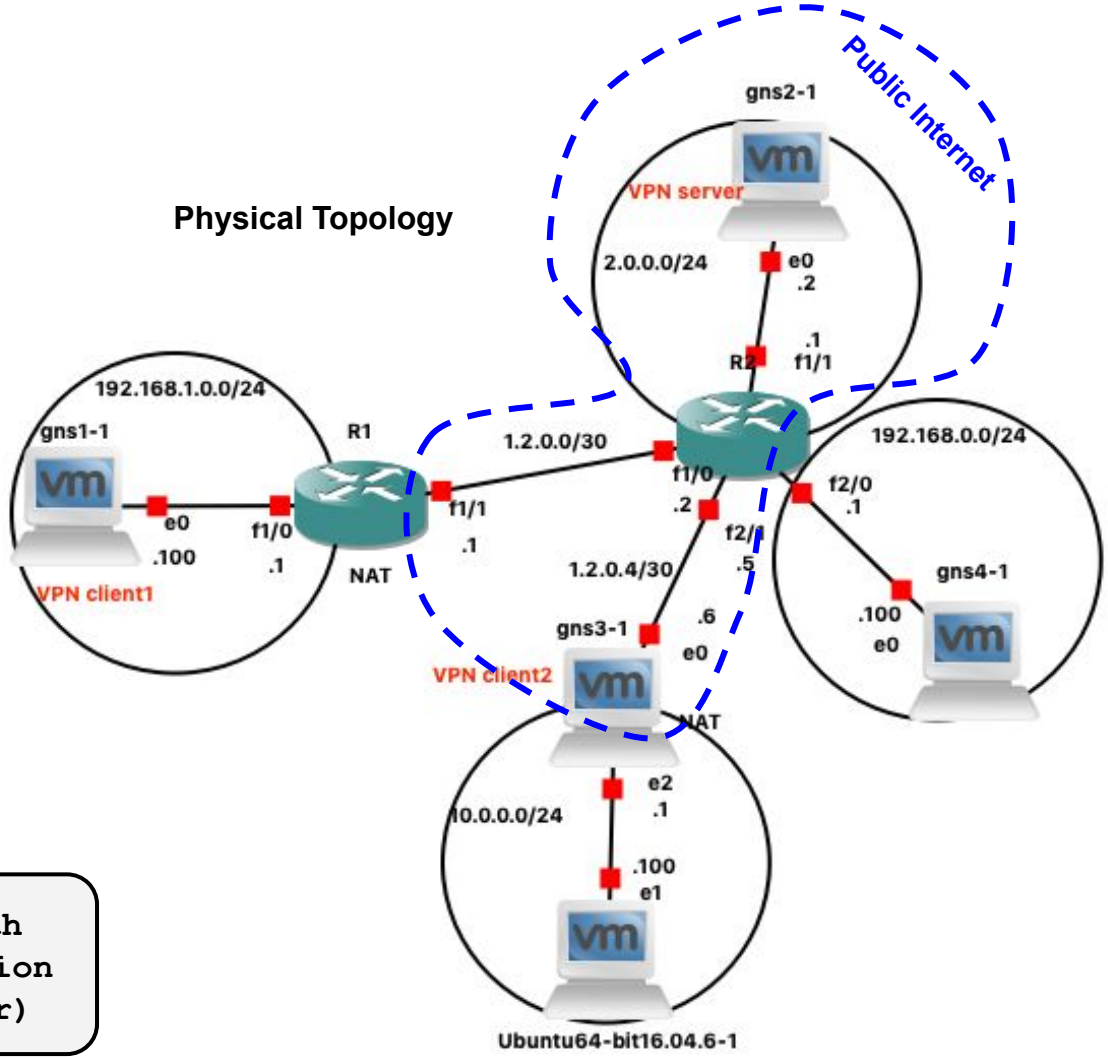


Lab Topology

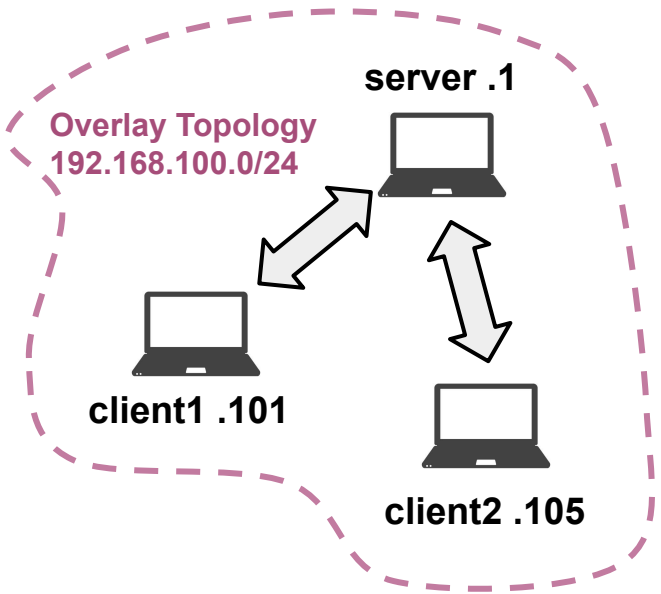


hub and spoke topology with client-to-client communication allowed (through the server)

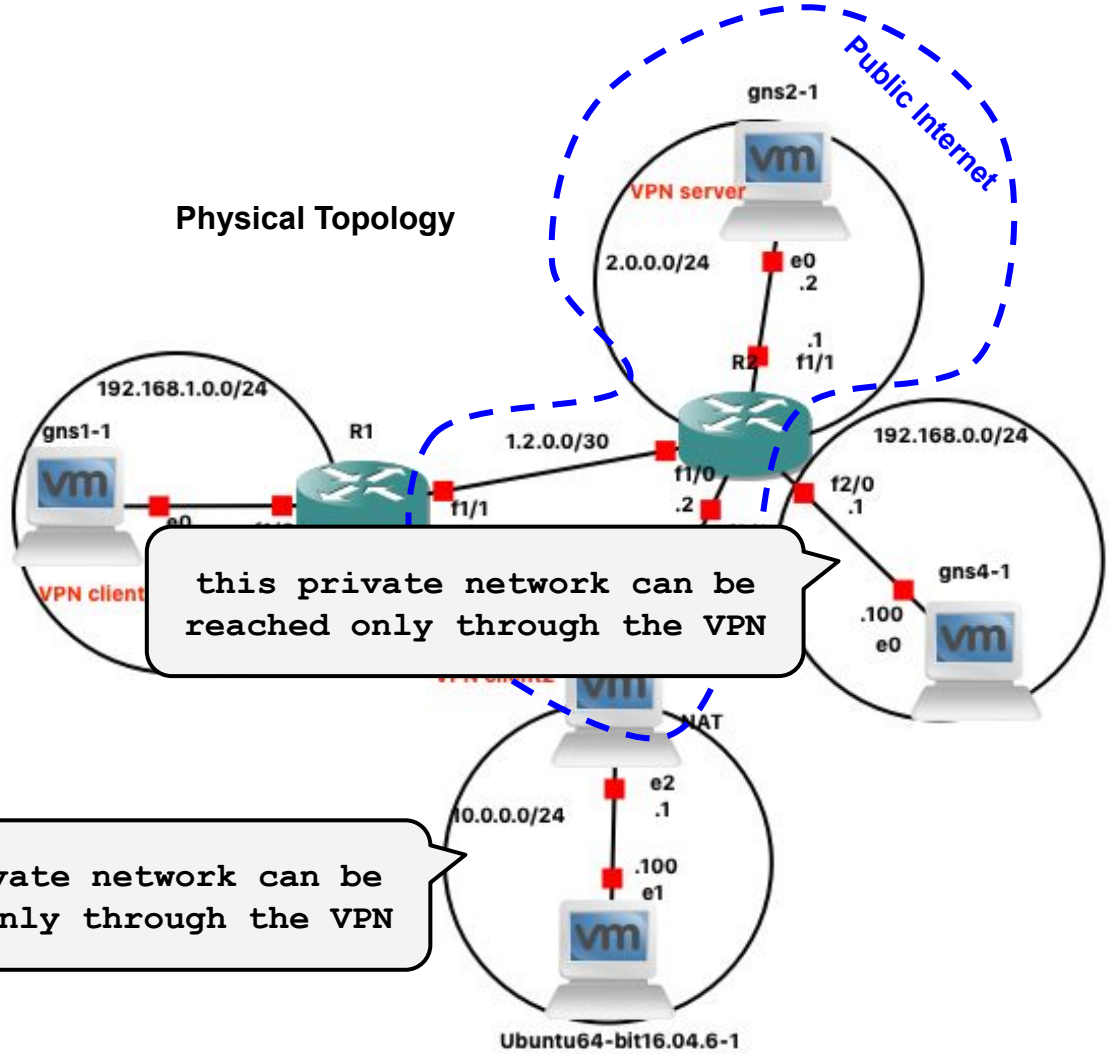
Physical Topology



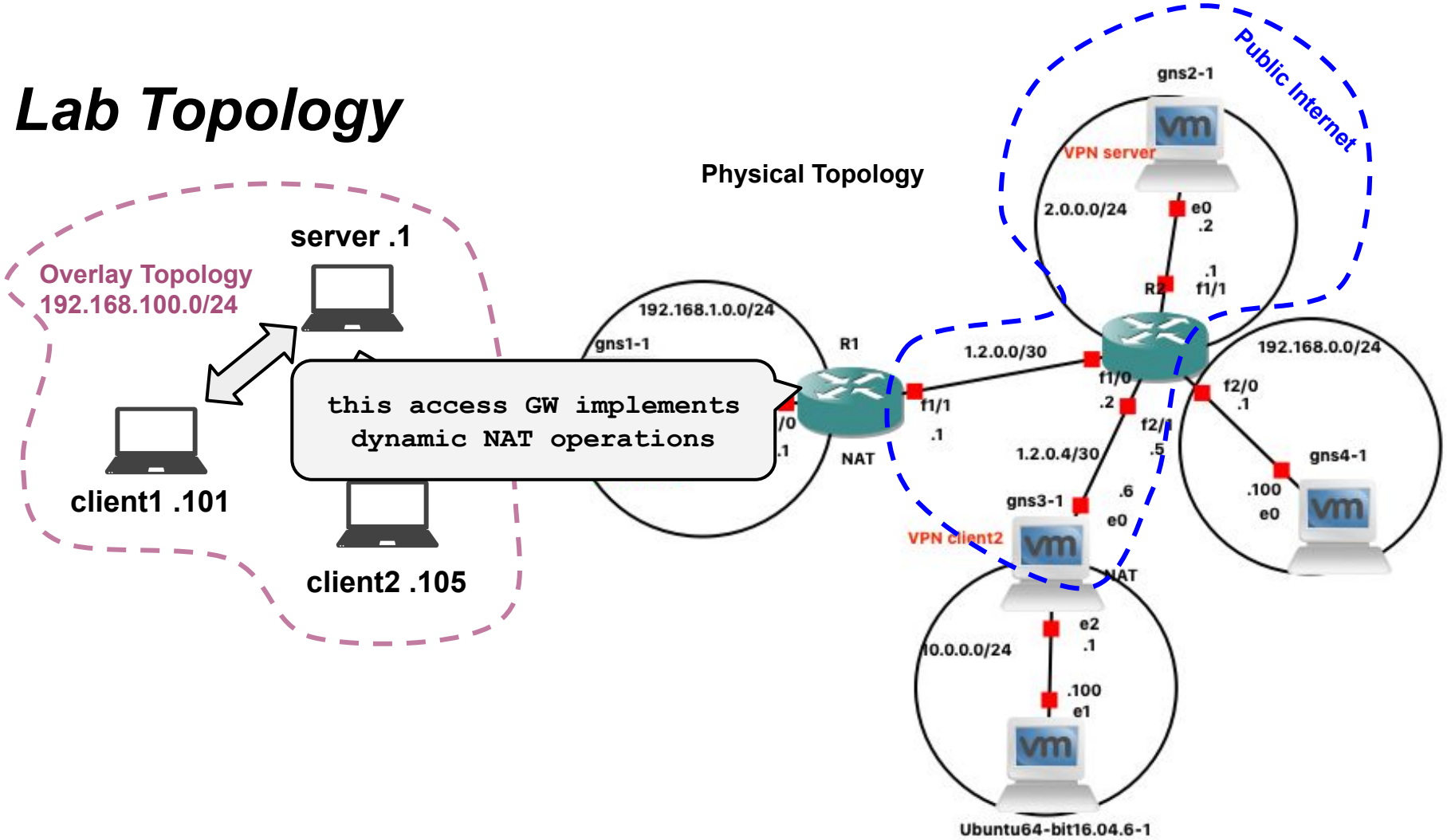
Lab Topology



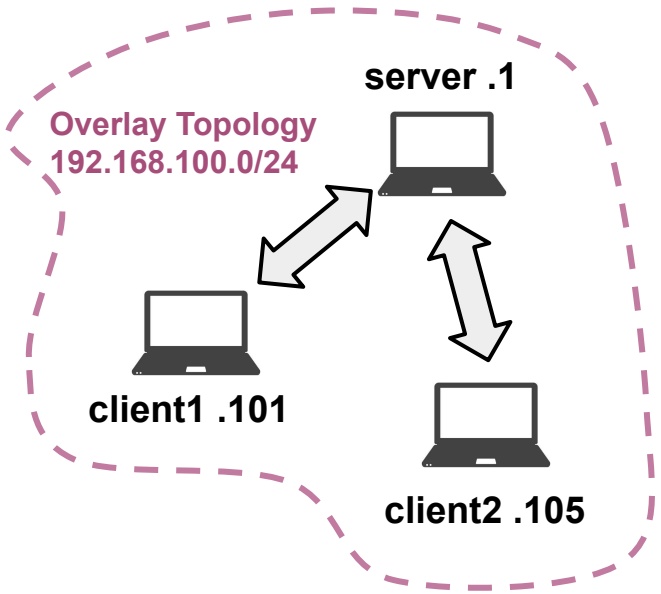
Physical Topology

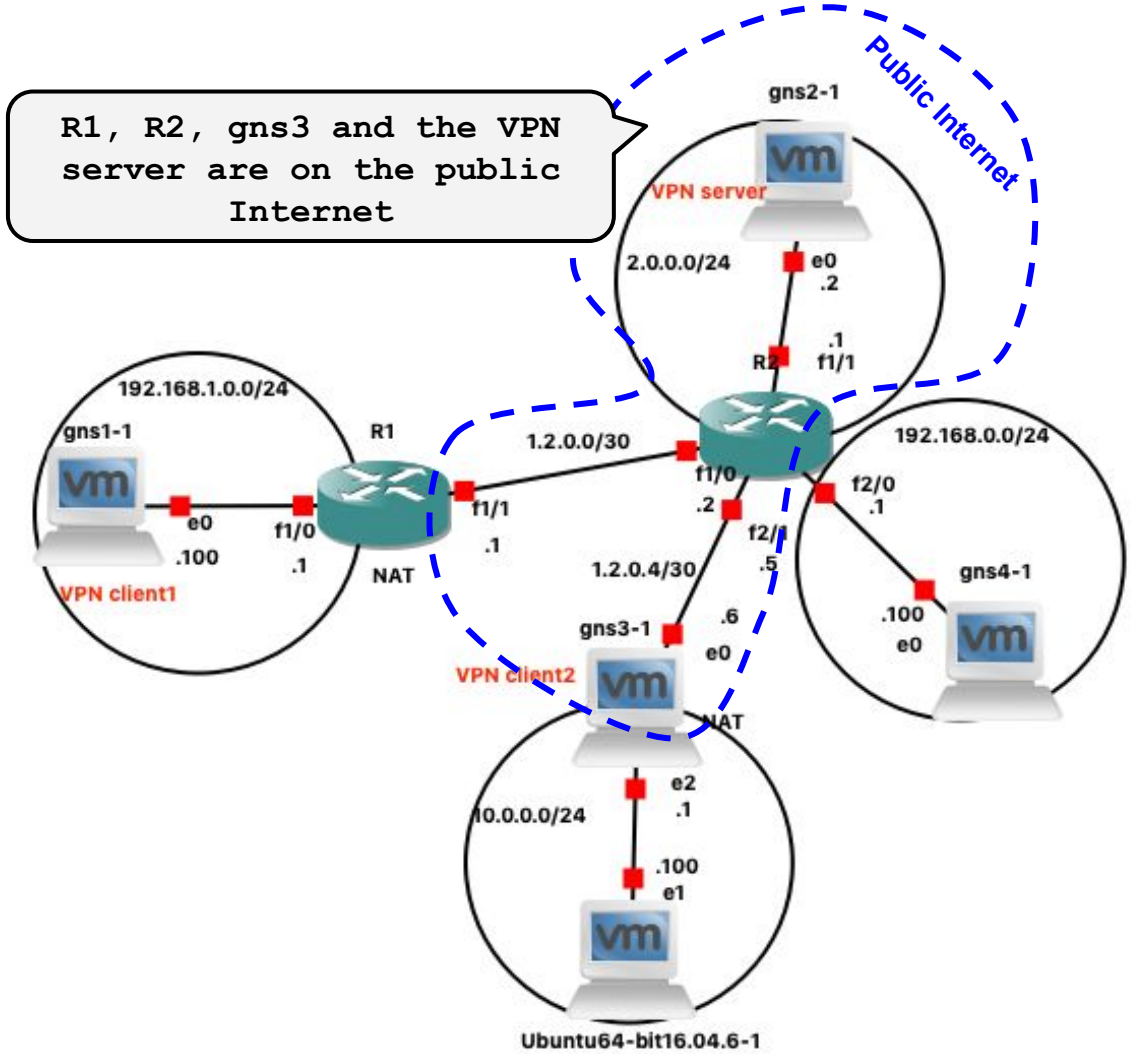


Lab Topology

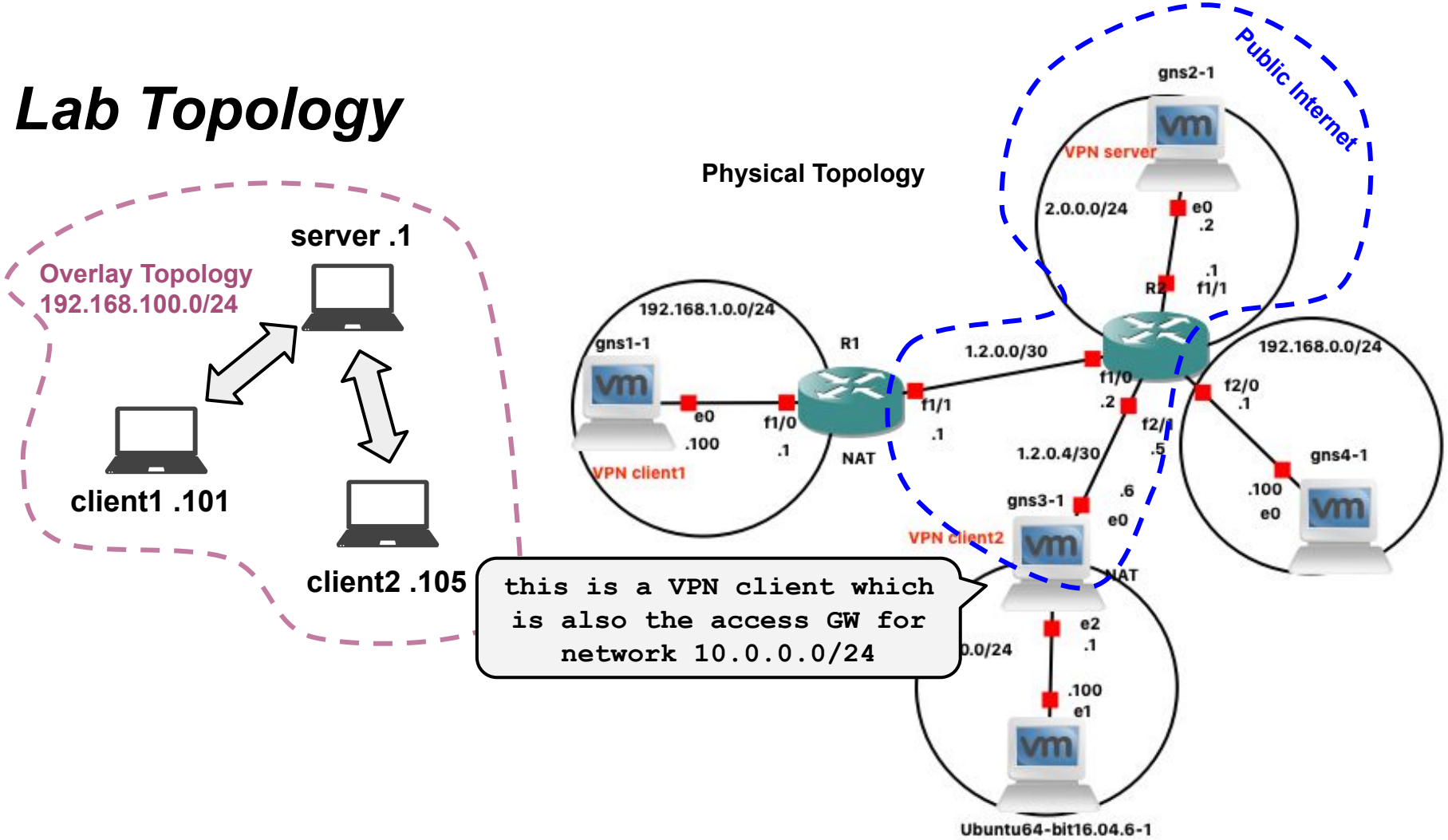


Lab Topology

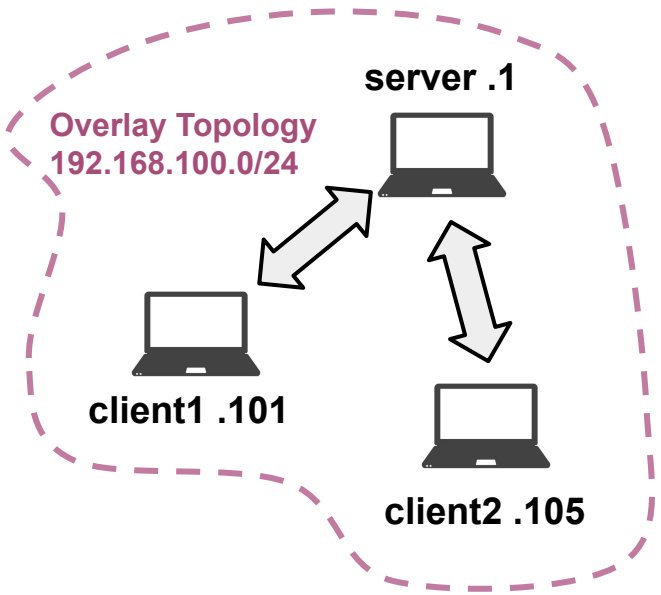




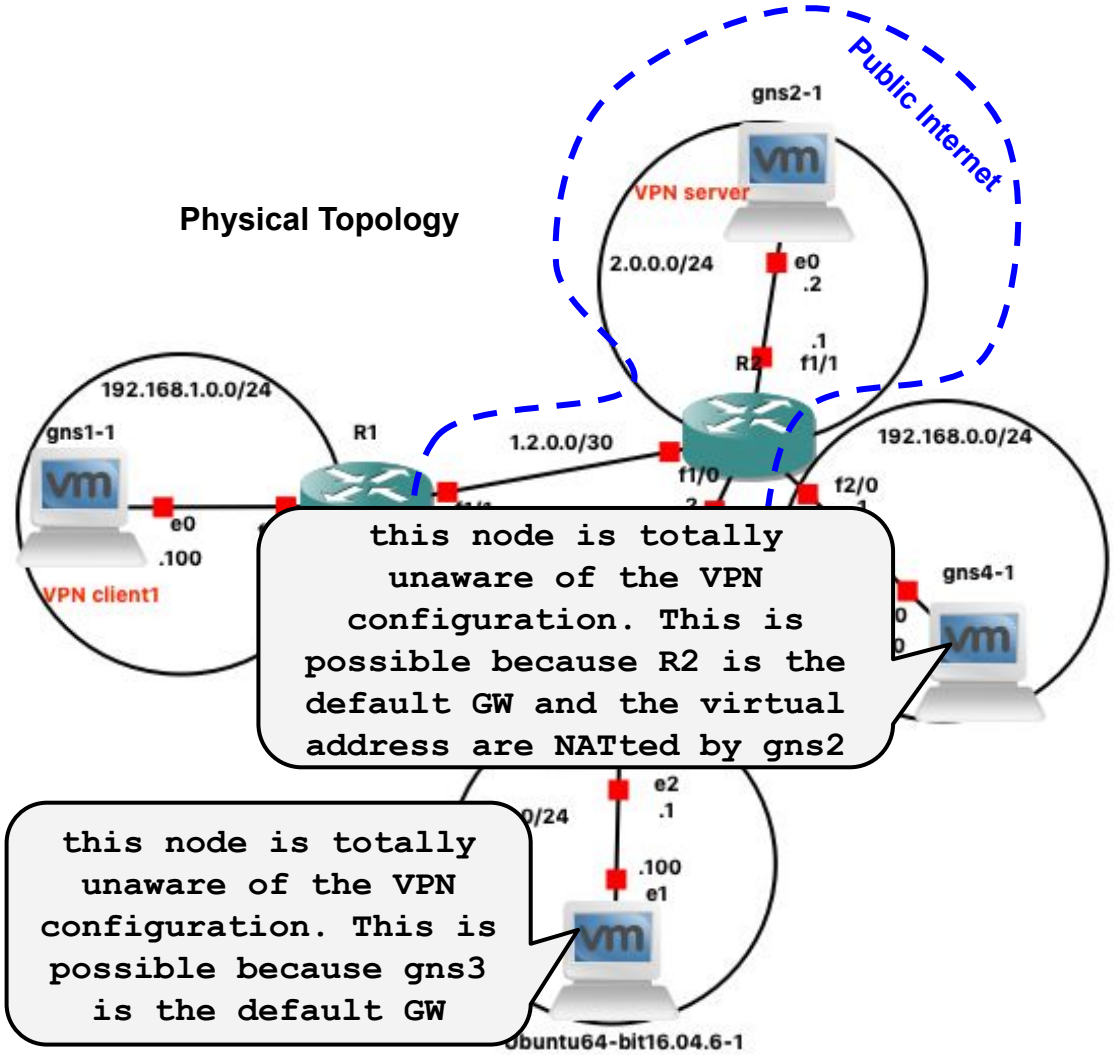
Lab Topology



Lab Topology



Physical Topology



Certificate management

Generate the master Certificate Authority (CA) certificate & key

To create and manage our VPN CA we use a tool named easy-rsa (which is basically a wrapper of opensll) which originally was bundeled with openVPN.

NOw it is a separate package (on -ubuntu: apt install easy-rsa)

```
r_server_vpn# . ./vars
r_server_vpn# ./clean-all
r_server_vpn# ./build-ca
```

Initialization

The final command (build-ca) will build the certificate authority (CA) certificate and key by invoking the interactive openssl command. Most queried parameters were defaulted to the values set in the vars file. The only parameter which must be explicitly entered is the Common Name.

Generate certificate & key for server and clients

Generate a certificate and private key for the server

```
r_server_vpn# ./build-key-server server
```

Generate client keys and certificates

```
r_server_vpn# ./build-key client1
r_server_vpn# ./build-key client2
```

Diffie Hellman parameters must be generated for the OpenVPN server

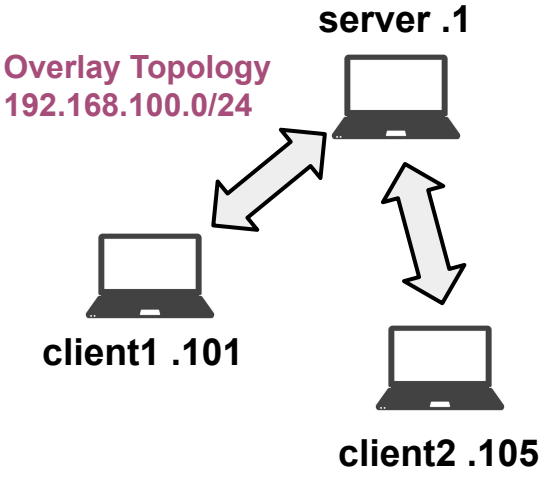
```
r_server_vpn# ./build-dh
```

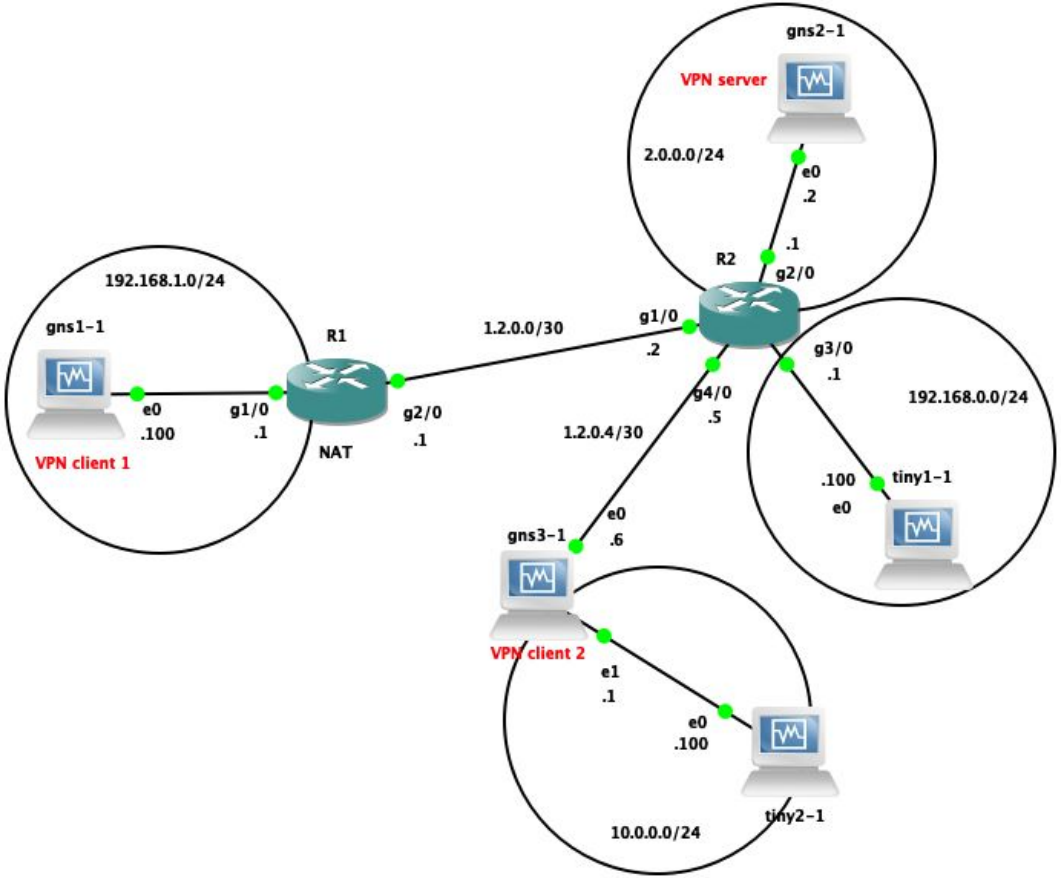
Where do I need the certificates and keys?

Filename	Needed By	Purpose	Secret
ca.crt	server + all clients	Root CA certificate	NO
ca.key	key signing machine only	Root CA key	YES
dh{n}.pem	server only	Diffie Hellman parameters	NO
server.crt	server only	Server Certificate	NO
server.key	server only	Server Key	YES
client1.crt	client1 only	Client1 Certificate	NO
client1.key	client1 only	Client1 Key	YES
client2.crt	client2 only	Client2 Certificate	NO
client2.key	client2 only	Client2 Key	YES

Network Configuration

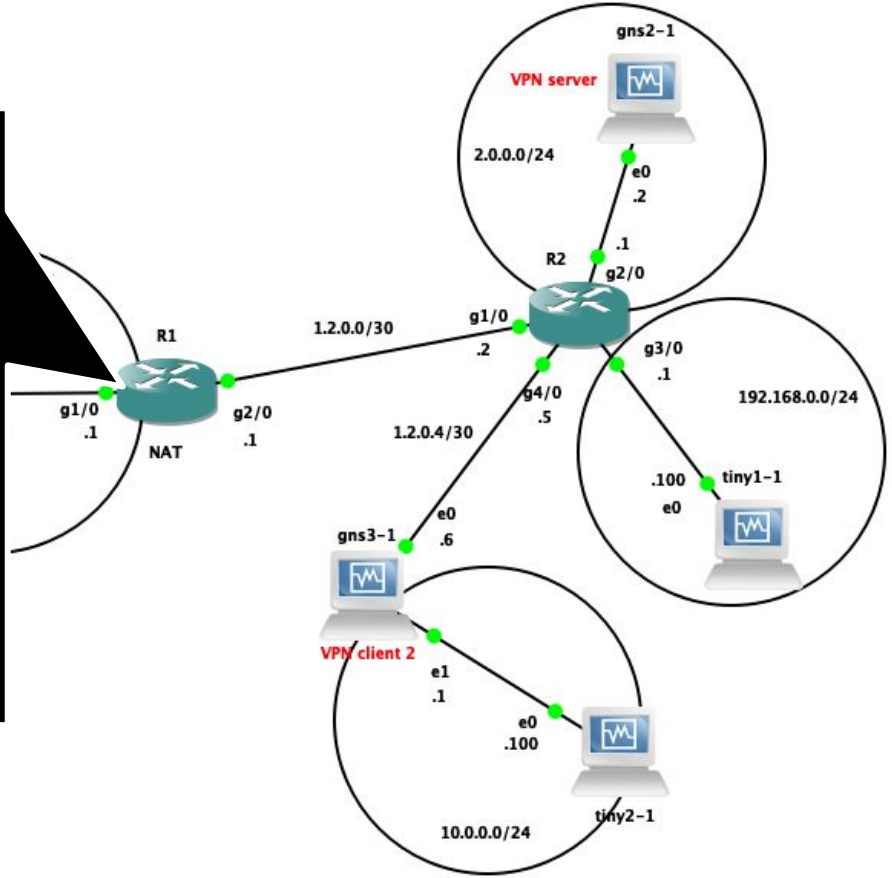
Configuration



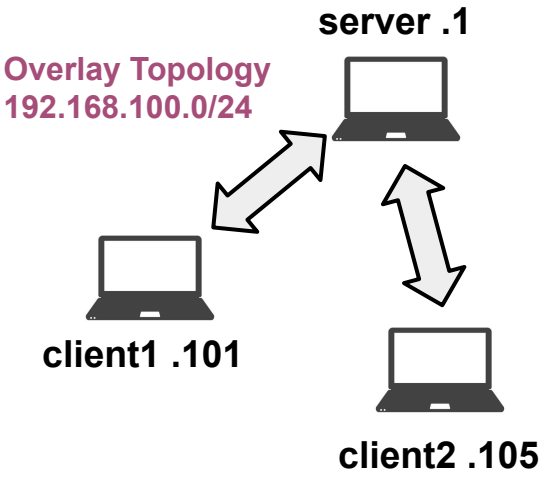


Configuration

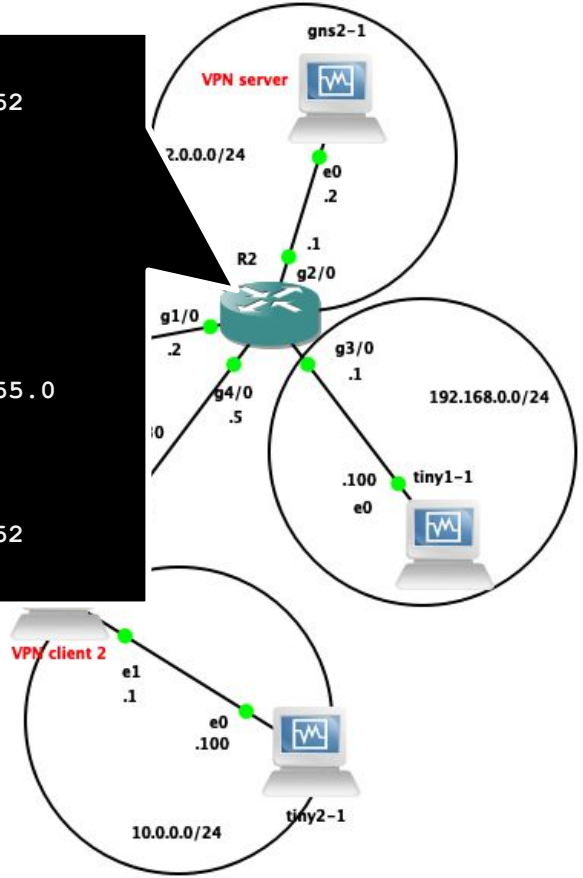
```
interface g1/0
ip address 192.168.1.1 255.255.255.0
ip nat inside
no shutdown
!
interface g2/0
ip address 1.2.0.1 255.255.255.252
ip nat outside
no shutdown
!
ip route 2.0.0.0 255.255.255.0 1.2.0.2
ip route 1.2.0.4 255.255.255.252 1.2.0.2
!
access-list 101 permit ip 192.168.1.0 0.0.0.255 any
!
ip nat inside source list 101 interface g2/0 overload
```



Configuration




```
interface g1/0
ip address 1.2.0.2 255.255.255.252
no shutdown
!
interface g2/0
ip address 2.0.0.1 255.255.255.0
no shutdown
!
interface g3/0
ip address 192.168.0.1 255.255.255.0
no shutdown
!
interface g4/0
ip address 1.2.0.5 255.255.255.252
no shutdown
```



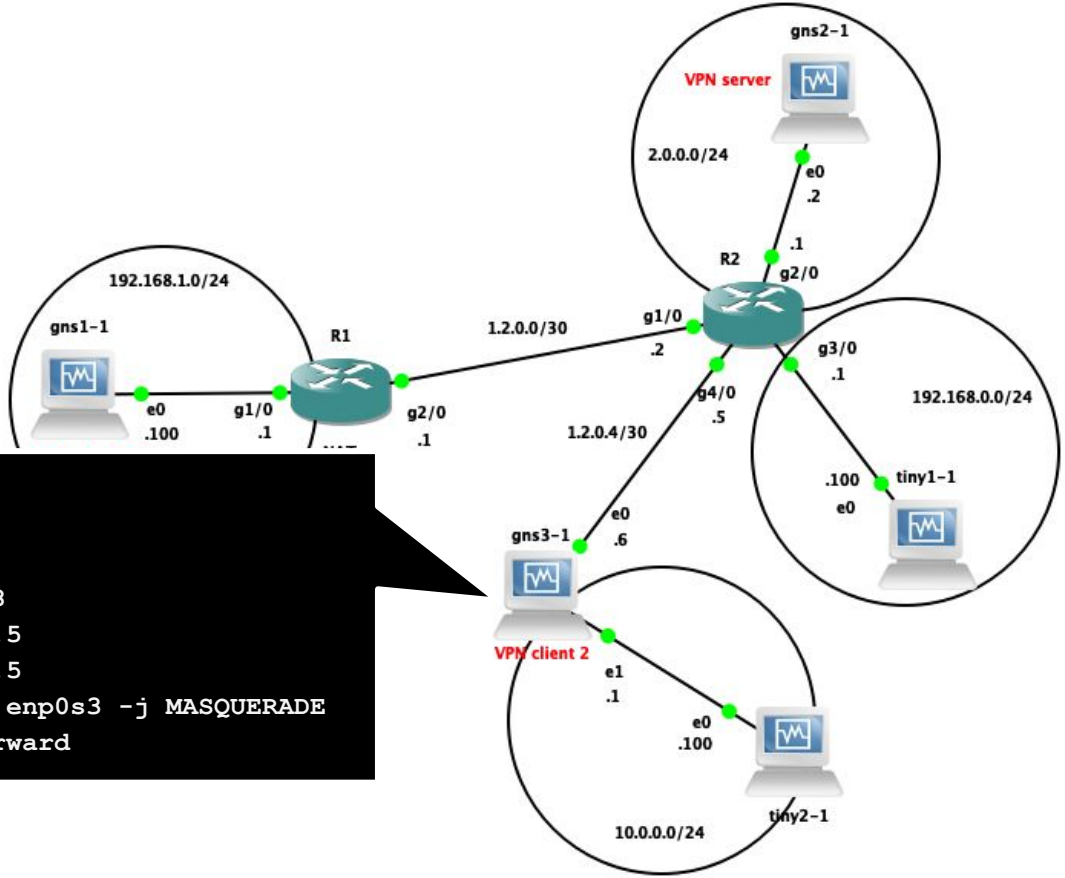
Configuration

server .1

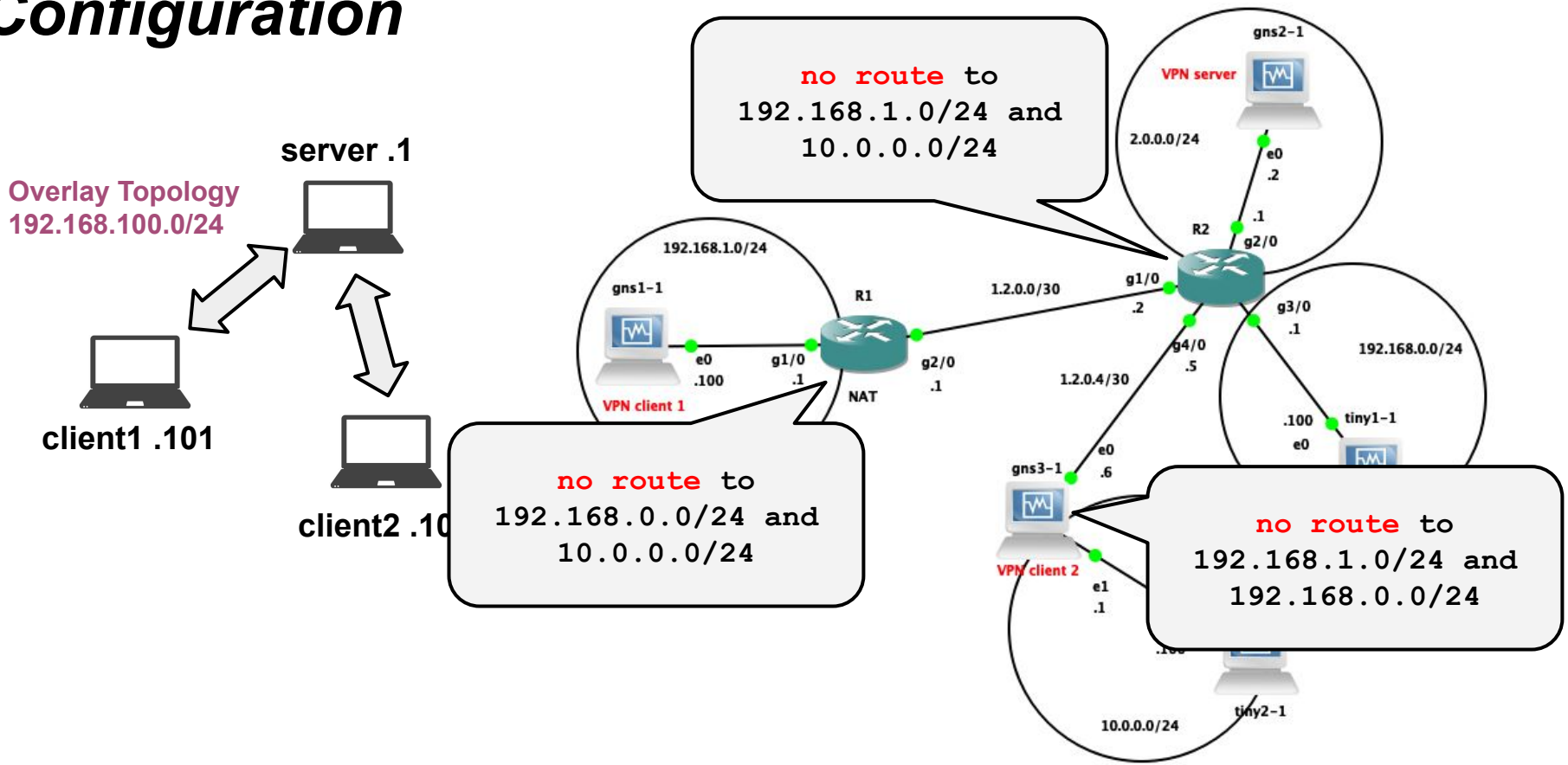
Overlay Topology
192.168.100.0/24

clie

```
ip link set enp0s3 up
ip link set enp0s8 up
ip addr add 1.2.0.6/30 dev enp0s3
ip addr add 10.0.0.1/24 dev enp0s8
ip route add 1.2.0.0/30 via 1.2.0.5
ip route add 2.0.0.0/24 via 1.2.0.5
iptables -t NAT -A POSTROUTING -o enp0s3 -j MASQUERADE
echo 1 > /proc/sys/net/ipv4/ip_forward
```



Configuration



Configur

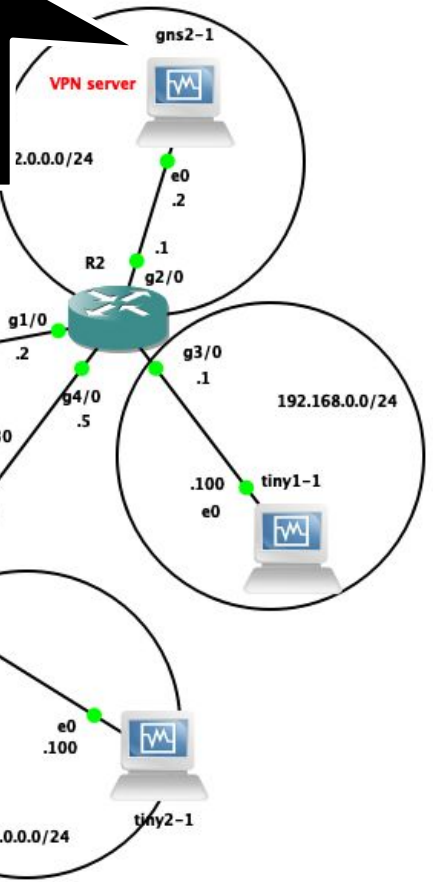
```
ip link set enp0s3 up
ip addr add 2.0.0.2/24 dev enp0s3
ip route add default via 2.0.0.1

echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -t NAT -A POSTROUTING -o enp0s3 -j MASQUERADE
```

Overlay Topology
192.168.100.0/24

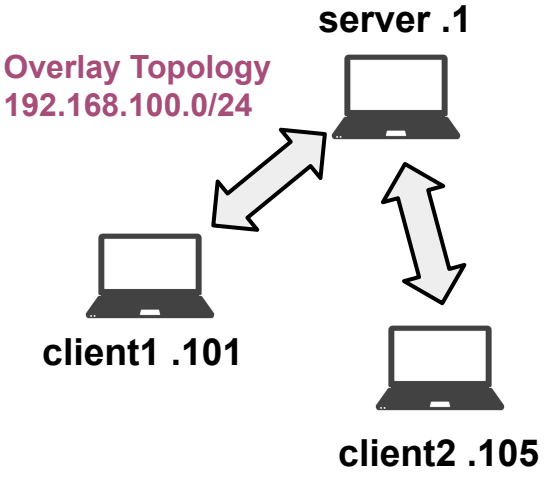
client1 .101

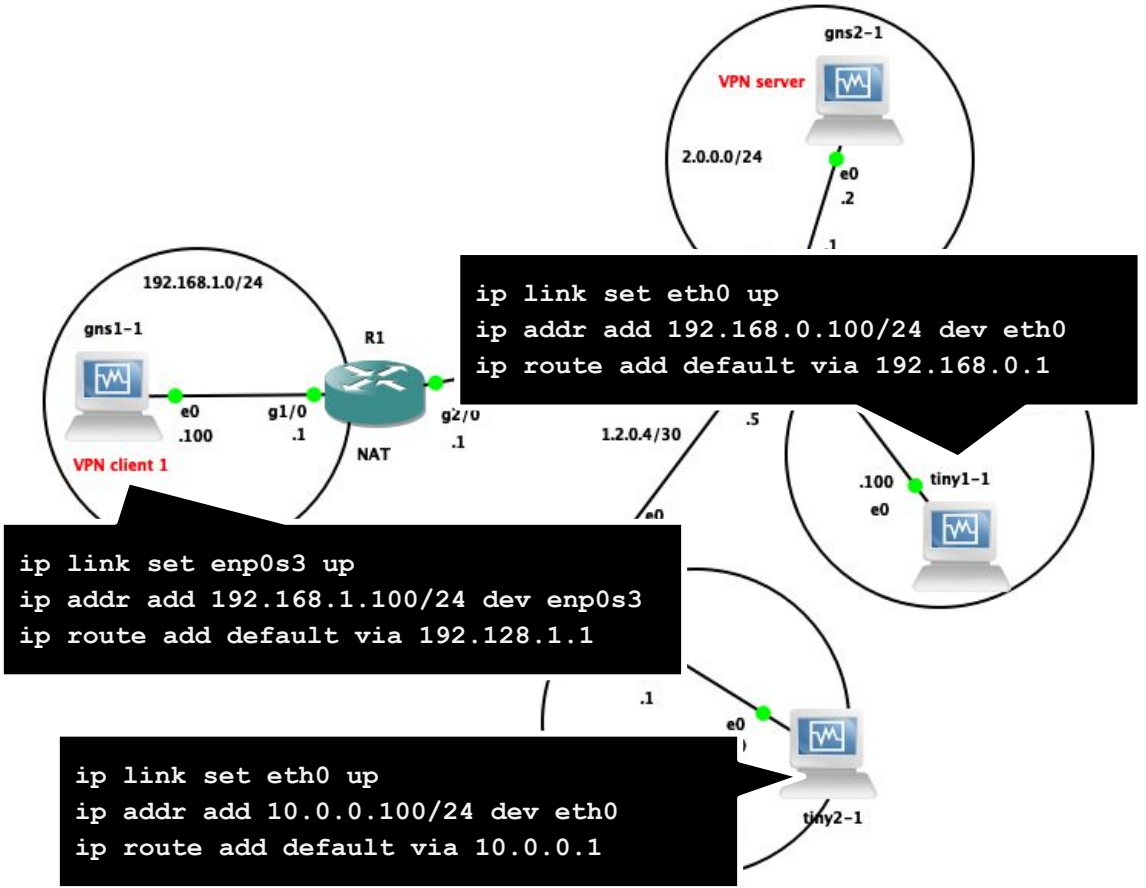
client2 .1

why the last two commands?
The first is required to forward packets received from the VPN
The second is for masquerading the virtual addresses of the VPN. By doing this we don't have to install routes to the VPN in R2 (which would work anyway…)



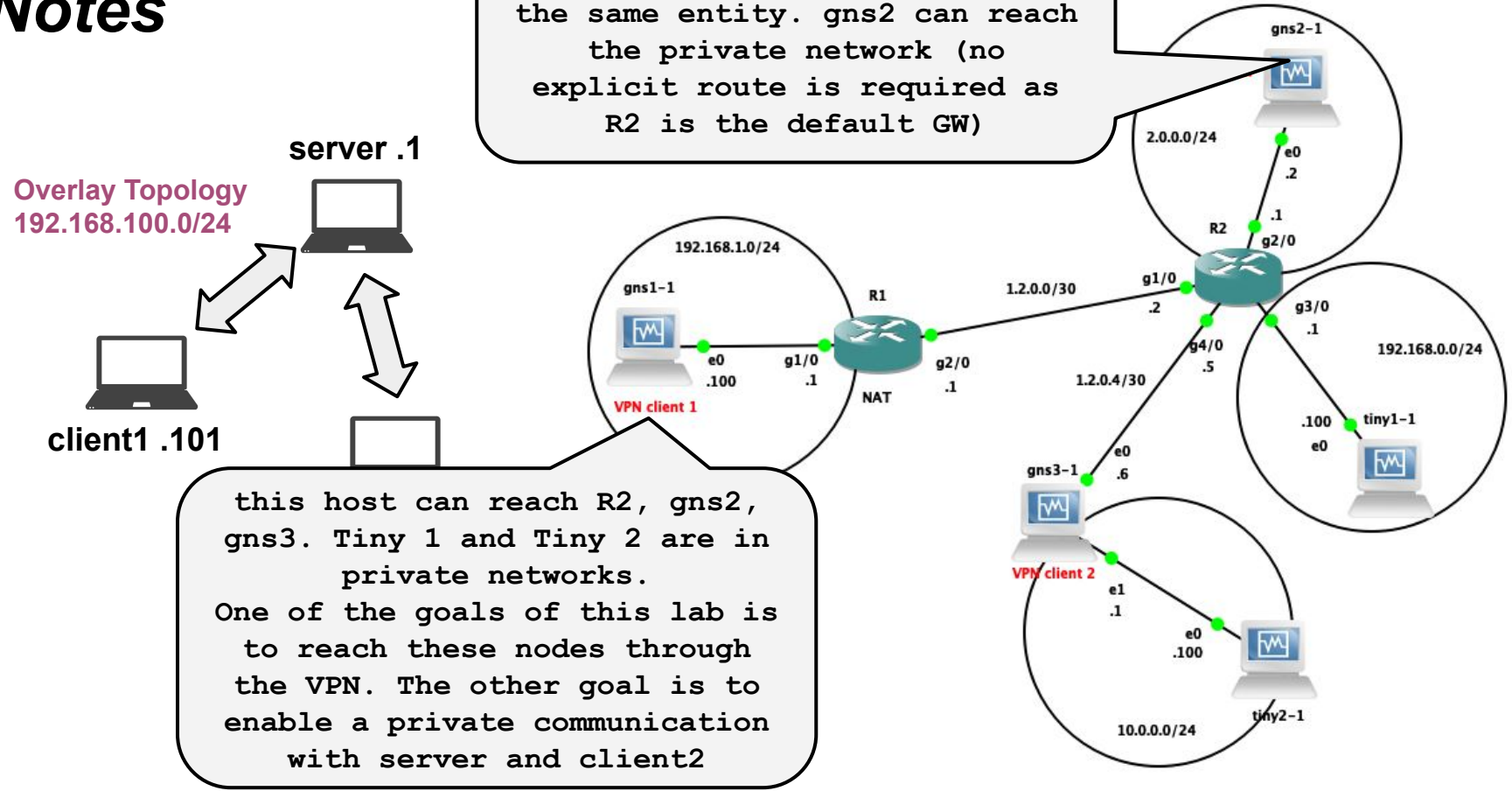
Configuration





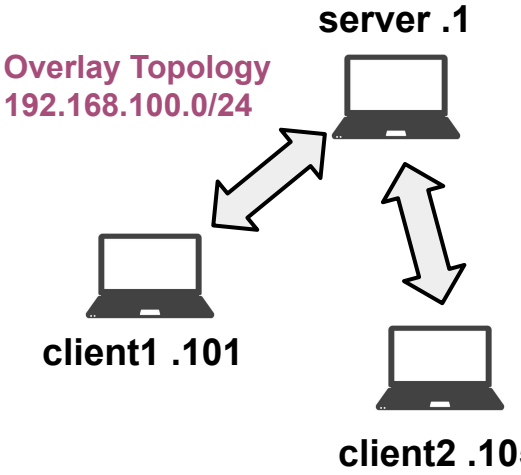
Notes

in this scenario 2.0.0.0/24 and 192.168.0.0/24 are handled by the same entity. gns2 can reach the private network (no explicit route is required as R2 is the default GW)



this host can reach R2, gns2, gns3. Tiny 1 and Tiny 2 are in private networks.
One of the goals of this lab is to reach these nodes through the VPN. The other goal is to enable a private communication with server and client2

Notes



Even though gns3 is on a public address, tiny 2 can be reached only through the VPN (by using the private virtual address)

this host can not reach R2, gns1 and tiny. The only way to do so is through the VPN (by using the private virtual addresses)

OpenVPN configuration

Creating configuration files for server and clients

The best way to configure the clients and server is to start from the example configuration files in:

/usr/share/doc/openvpn/example/sample-config-files/

client.conf

server.conf.gz

Important options

- port [port_number]
- proto {udp|tcp}
- dev {tun|tap}
- ca [path]
- cert [path]
- key [path]
- dh [path]
- client
- remote [server_addr] [port]
- server [net_addr] [met_mask]
- client_to_client
- push "route net_addr net_mask"
- route net_addr net_mask
- client-config-dir [path]

this specifies the listening port for the server. This port must match the one specified with the client configuration directive "remote". This port will also be the remote port for the external headers of packets sent from the clients to the server

Important options

- port [port_number]
- proto {udp|tcp}
- dev {tun|tap}
- ca [path]
- cert [path]
- key [path]
- dh [path]
- client
- remote [server_addr] [port]
- server [net_addr] [met_mask]
- client_to_client
- push "route net_addr net_mask"
- route net_addr net_mask
- client-config-dir [path]

this specifies the encapsulation format and must be the same in both clients and server configuration files

Important options

- port [port_number]
- proto {udp|tcp}
- dev {tun|tap}
- ca [path]
- cert [path]
- key [path]
- dh [path]
- client
- remote [server_addr] [port]
- server [net_addr] [met_mask]
- client_to_client
- push "route net_addr net_mask"
- route net_addr net_mask
- client-config-dir [path]

this specifies the virtual interface type and must be the same in both clients and server configuration files

Important options

- port [port_number]
- proto {udp|tcp}
- dev {tun|tap}
- ca [path]
- cert [path]
- key [path]
- dh [path]
- client
- remote [server_addr] [port]
- server [net_addr] [met_mask]
- client_to_client
- push "route net_addr net_mask"
- route net_addr net_mask
- client-config-dir [path]

path to:
1. CA certificate (server/client)
2. local certificate (server/client)
3. key pair file (server/client)
4. DH parameters (server)

Important options

- port [port_number]
- proto {udp|tcp}
- dev {tun|tap}
- ca [path]
- cert [path]
- key [path]
- dh [path]
- client
- remote [server_addr] [port]
- server [net_addr] [met_mask]
- client_to_client
- push "route net_addr net_mask"
- route net_addr net_mask
- client-config-dir [path]

this tells OpenVPN to run as a client instance. In this case the next directive specifies the IP and port of the server

Important options

- port [port_number]
- proto {udp|tcp}
- dev {tun|tap}
- ca [path]
- cert [path]
- key [path]
- dh [path]
- client
- remote [server_addr] [port]
- server [net_addr] [met_mask]
- client_to_client
- push "route net_addr net_mask"
- route net_addr net_mask
- client-config-dir [path]

this tells OpenVPN to run as a server instance and to allocate a given VPN address range. The server's virtual adapter will be configured with the first valid IP address of this address range

Important options

- port [port_number]
- proto {udp|tcp}
- dev {tun|tap}
- ca [path]
- cert [path]
- key [path]
- dh [path]
- client
- remote [server_addr]
- server [net_addr] [
- client_to_client
- push "route net_addr net_mask"
- route net_addr net_mask
- client-config-dir [path]

this enables overlay client to client communication through the VPN server. The overlay topology is a hub and spoke

Important options

- port [port_number]
- proto {udp|tcp}
- dev {tun|tap}
- ca [path]
- cert [path]
- key [path]
- dh [path]
- client
- remote [server_addr] [p
- server [net_addr] [met_mas
- client_to_client
- push "route net_addr net_mask"
- route net_addr net_mask
- client-config-dir [path]

These two option influence the real routing table of all clients (first directive) and servers (second directive). Multiple routes can be specified. Each directive will result in the automatic insertion of a routing entry:

net_addr/net_mast via virtual_next_hop dev viface

Important options

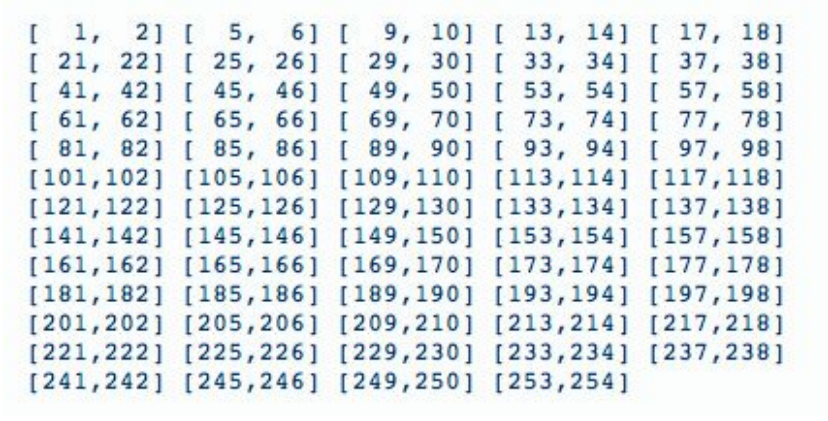
- port [port_number]
- proto {udp|tcp}
- dev {tun|tap}
- ca [path]
- cert [path]
- key [path]
- dh [path]
- client
- remote [server_addr] [p
- server [net_addr] [met_
- client_to_client
- push "route net_addr net_mas
- route net_addr net_mask
- client-config-dir [path]

This set the path to the per-client specific configuration directory. In this directory the server can have multiple files names as the CN of the client. Configuration directives in a file will affect only the relative client

Client-specific configuration

- A file for each OpenVPN client “CN”
  - In this lab: client1, client2
- In each file (+ other commands we’re not considering):
  - if-config-push [local_ptp] [remote_ptp]
  - iroute [net_addr] [net_mask]
- client1
  - `if-config-push 192.168.100.101 192.168.100.102`
- client2
  - `if-config-push 192.168.100.105 192.168.100.106`
  - `iroute 10.0.0.0 255.255.255.0`

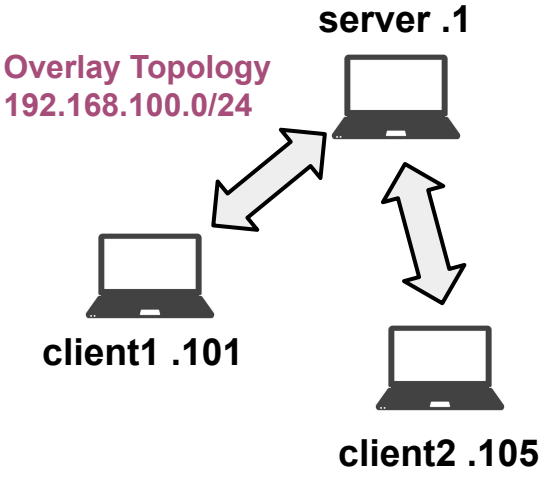
Allowed /30 pairs

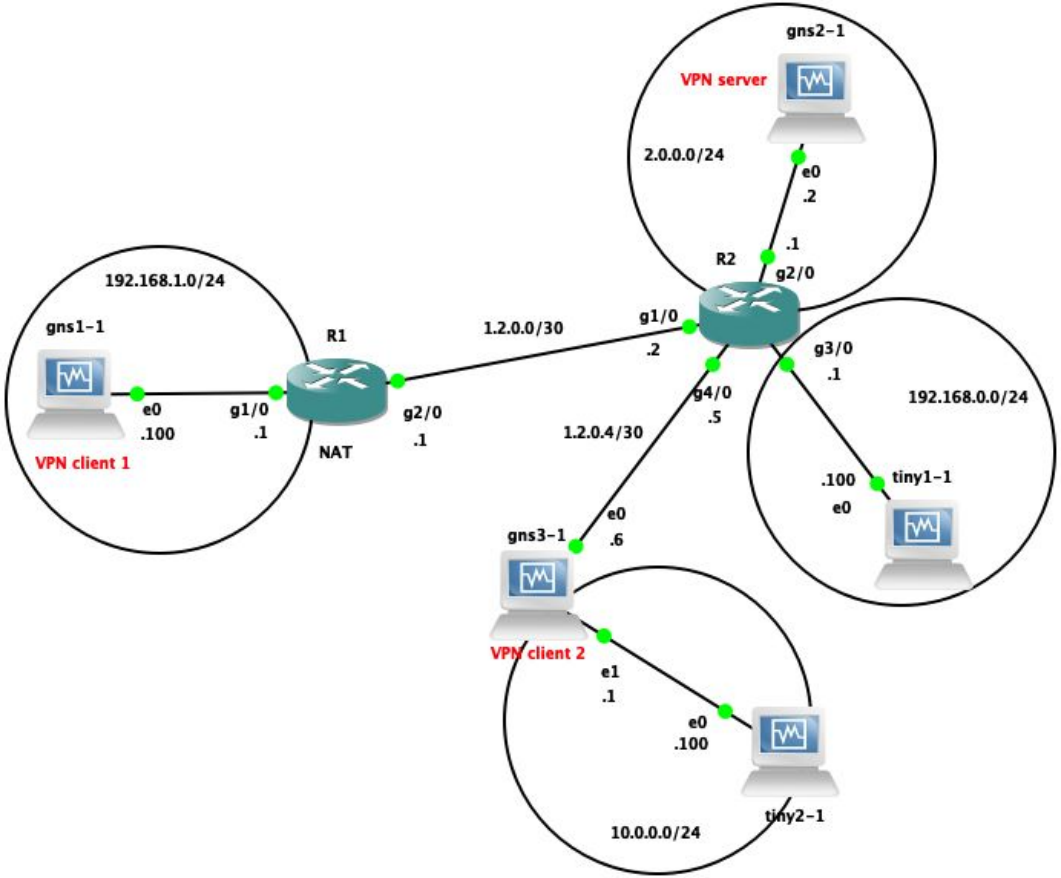


Why “push route”, “route” and “iroute”?

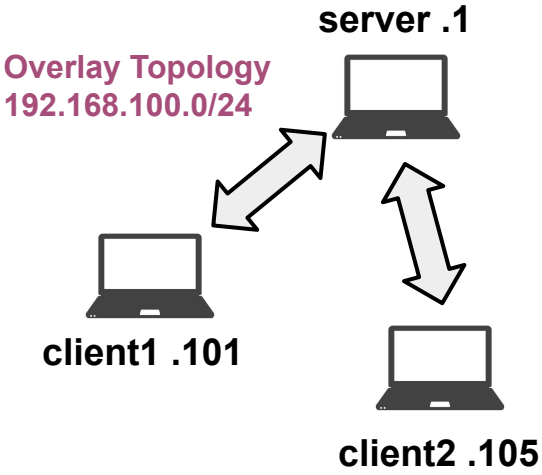
1. ***“push route”*** is pushing a given route to clients
   a. this influences the underlay routing in the clients
   b. after connection, the client will add a route via tun0 p2p peer
2. ***“route”*** controls the routing from the kernel to the OpenVPN server (via the TUN interface)
   a. this influences the underlay routing
   b. routes specified with this command are installed in the real IP routing table
3. ***“iroute”*** controls the routing from the OpenVPN server to the remote clients
   a. this influences the overlay routing
   b. routes specified with this command are installed in the overlay routing table (which is managed bny the OpenVPN server process)

Configuration

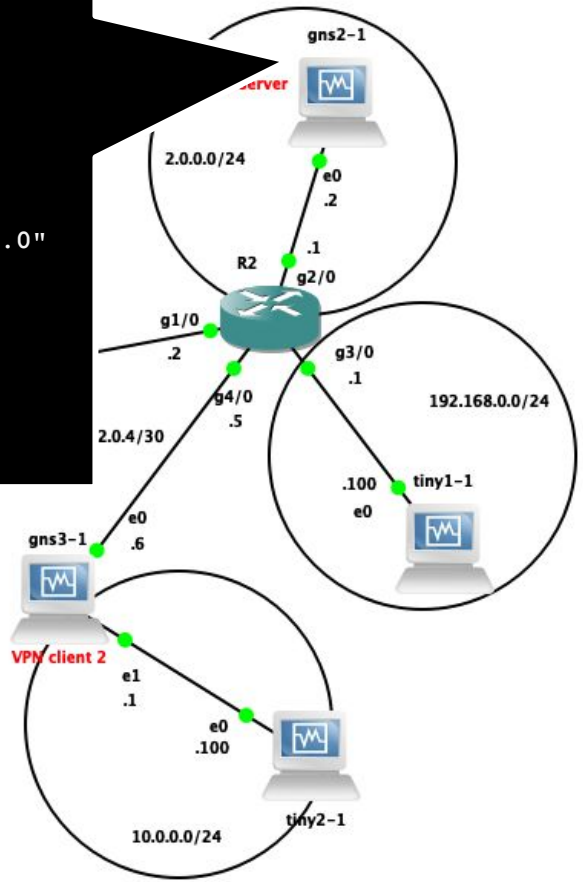




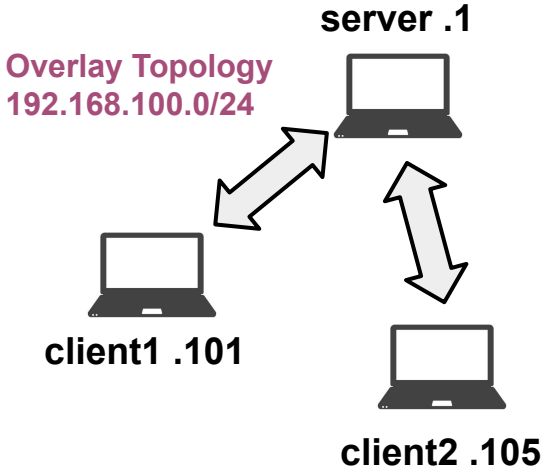
VPN Server



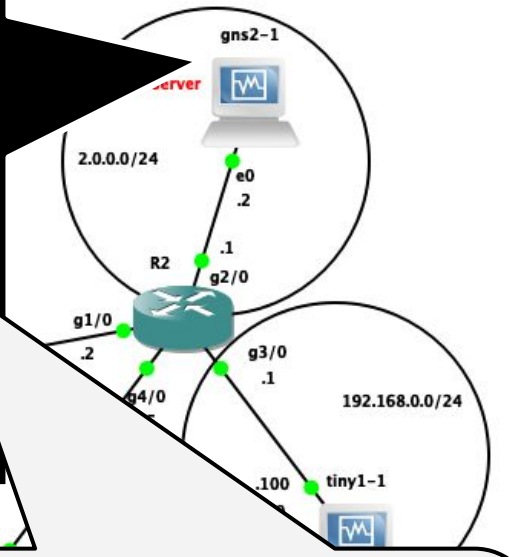

```
port 1194
proto udp
dev tun
ca ca.crt
cert server.crt
key server.key
dh dh2048.pem
server 192.168.100.0 255.255.255.0
push "route 192.168.0.0 255.255.255.0"
push "route 10.0.0.0 255.255.255.0"
route 10.0.0.0 255.255.255.0
client-config-dir ccd
client-to-client
keepalive 10 120
cipher AES-256-CBC
```



VPN Server




```
port 1194
proto udp
dev tun
ca ca.crt
cert server.crt
key server.key
dh dh2048.pem
server 192.168.100.0 255.255.255.0
push "route 192.168.0.0 255.255.255.0"
push "route 10.0.0.0 255.255.255.0"
route 10.0.0.0 255.255.255.0
client-config-dir ccd
client-to-client
keepalive 10 120
cipher AES-256-CBC
```

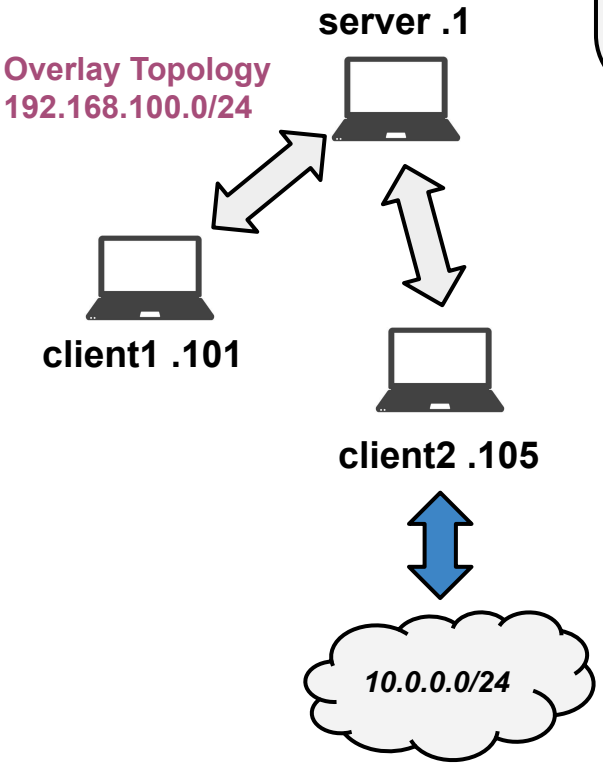


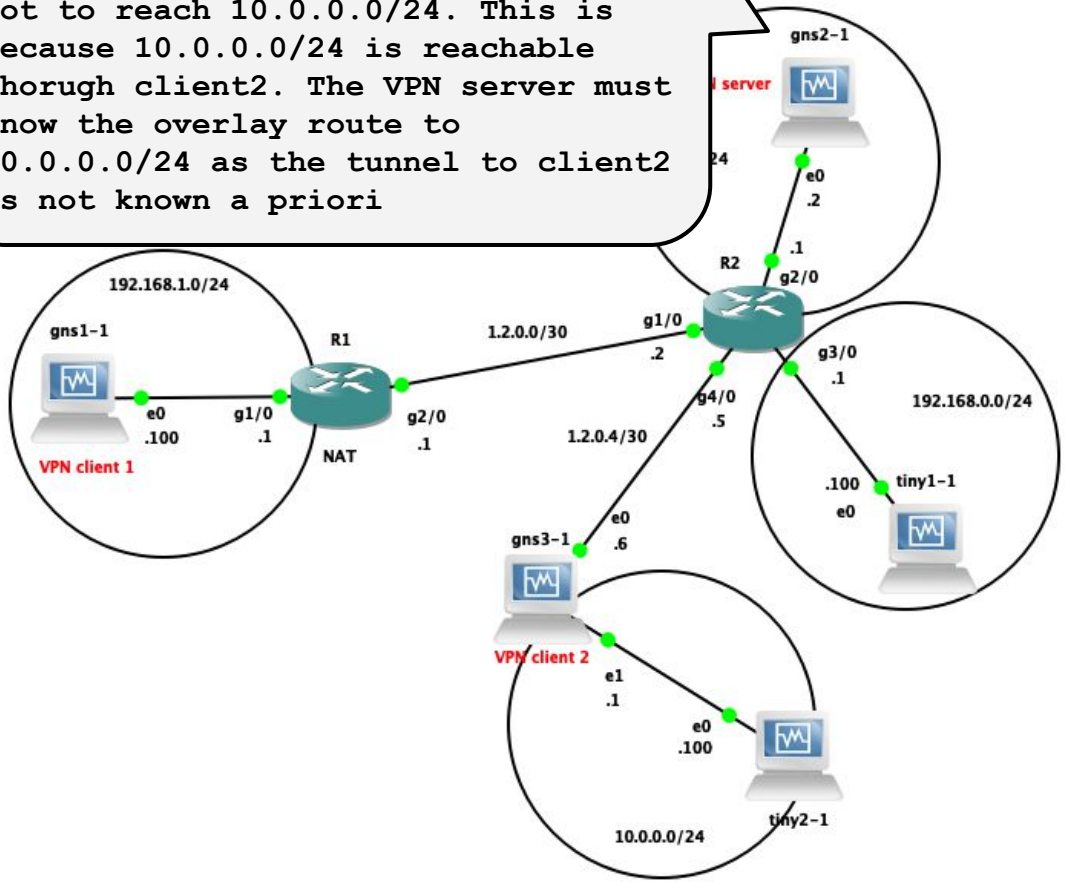
with this 2 directive the VPN server will push to every client a route to reach 192.168.0.0/24 and 10.0.0.0/24 "through the VPN".
This will result in the following 2 routing entries (for example in client 1):

1. destination 192.168.0.0/24, next hop 192.168.100.102, oiface tun0
2. destination 10.0.0.0/24, next hop 192.168.100.102, oiface tun0

VPN Server

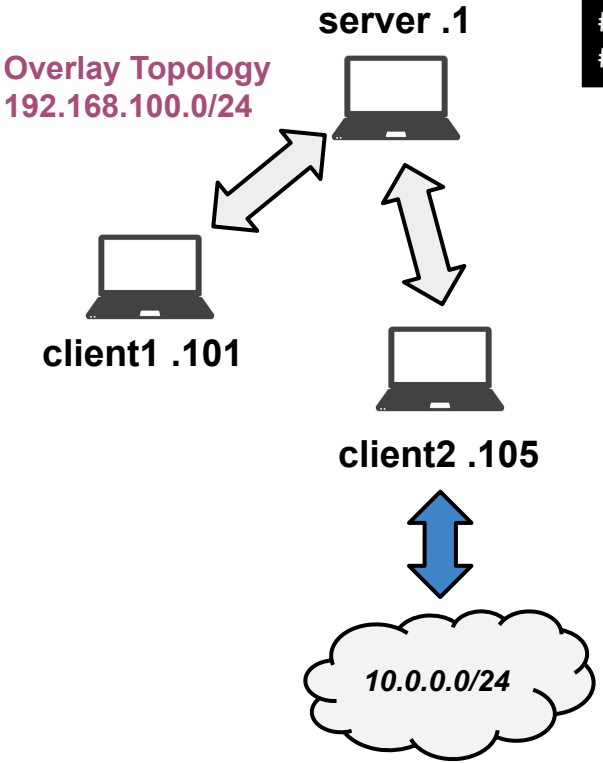
The 2 push route directives are enough to reeach 192.168.0.0/24, but not to reach 10.0.0.0/24. This is because 10.0.0.0/24 is reachable thorugh client2. The VPN server must know the overlay route to 10.0.0.0/24 as the tunnel to client2 is not known a priori

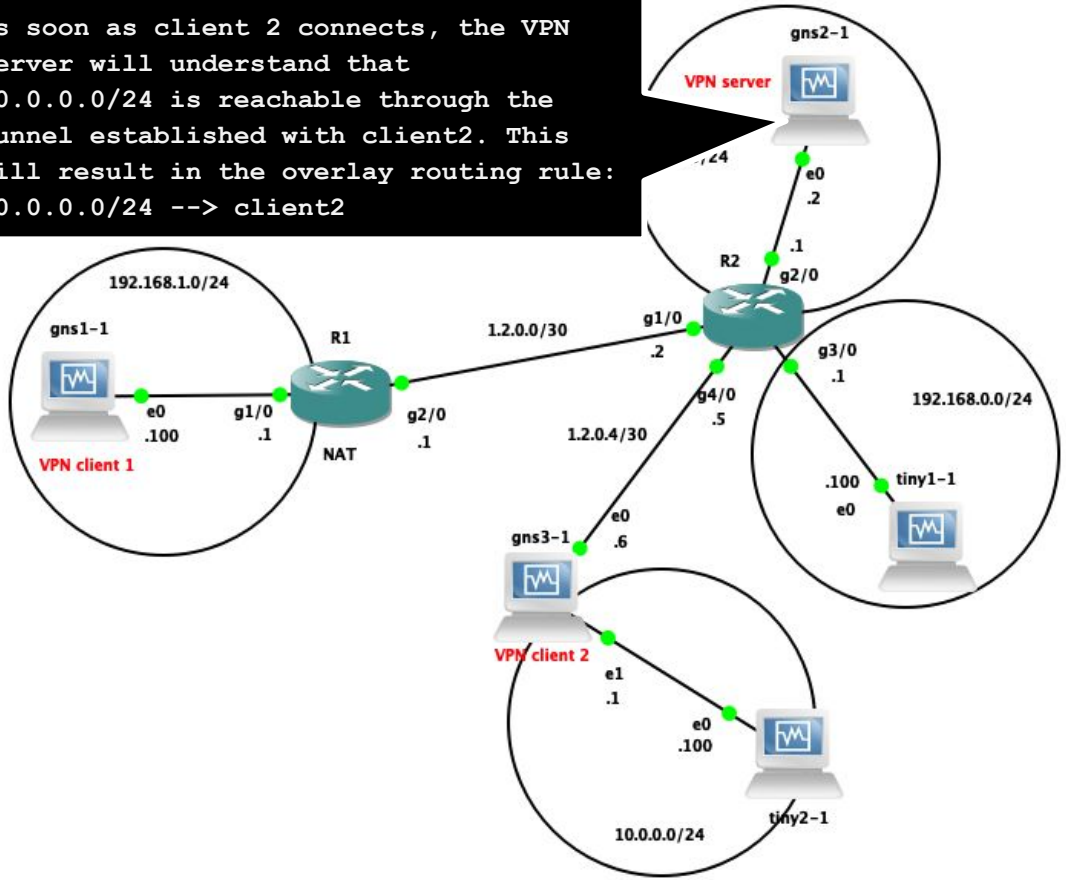




VPN Server

```
#in file ccd/client2
iroute 10.0.0.0 255.255.255.0

# as soon as client 2 connects, the VPN
# server will understand that
# 10.0.0.0/24 is reachable through the
# tunnel established with client2. This
# will result in the overlay routing rule:
# 10.0.0.0/24 --> client2
```

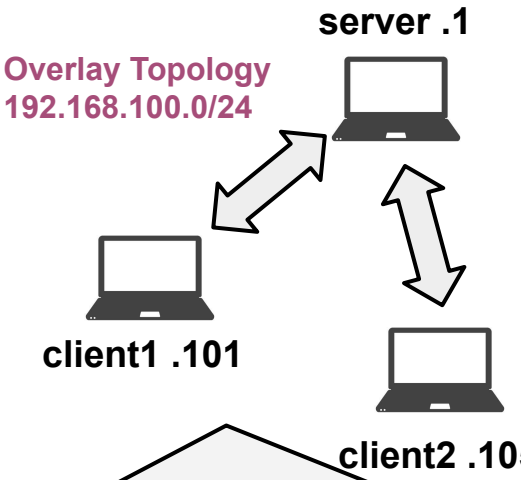


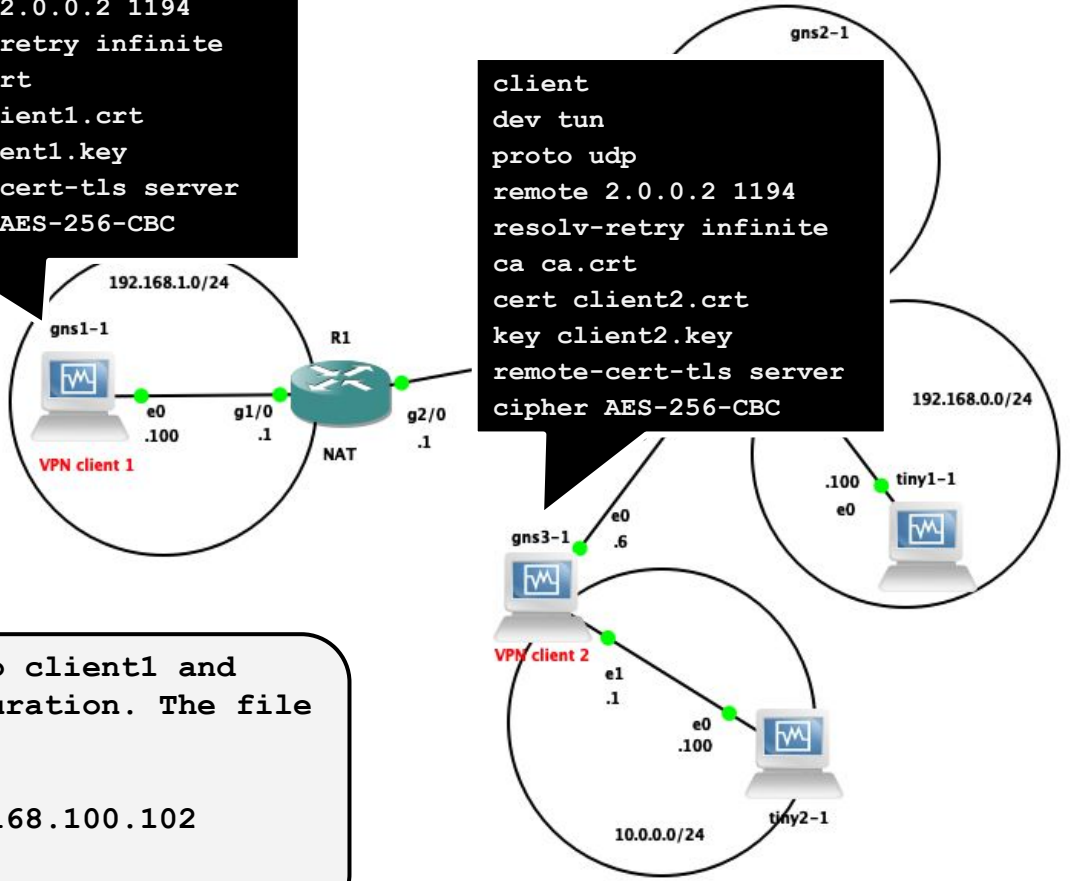


VPN Clients

```
client
dev tun
proto udp
remote 2.0.0.2 1194
resolv-retry infinite
ca ca.crt
cert client1.crt
key client1.key
remote-cert-tls server
cipher AES-256-CBC
```

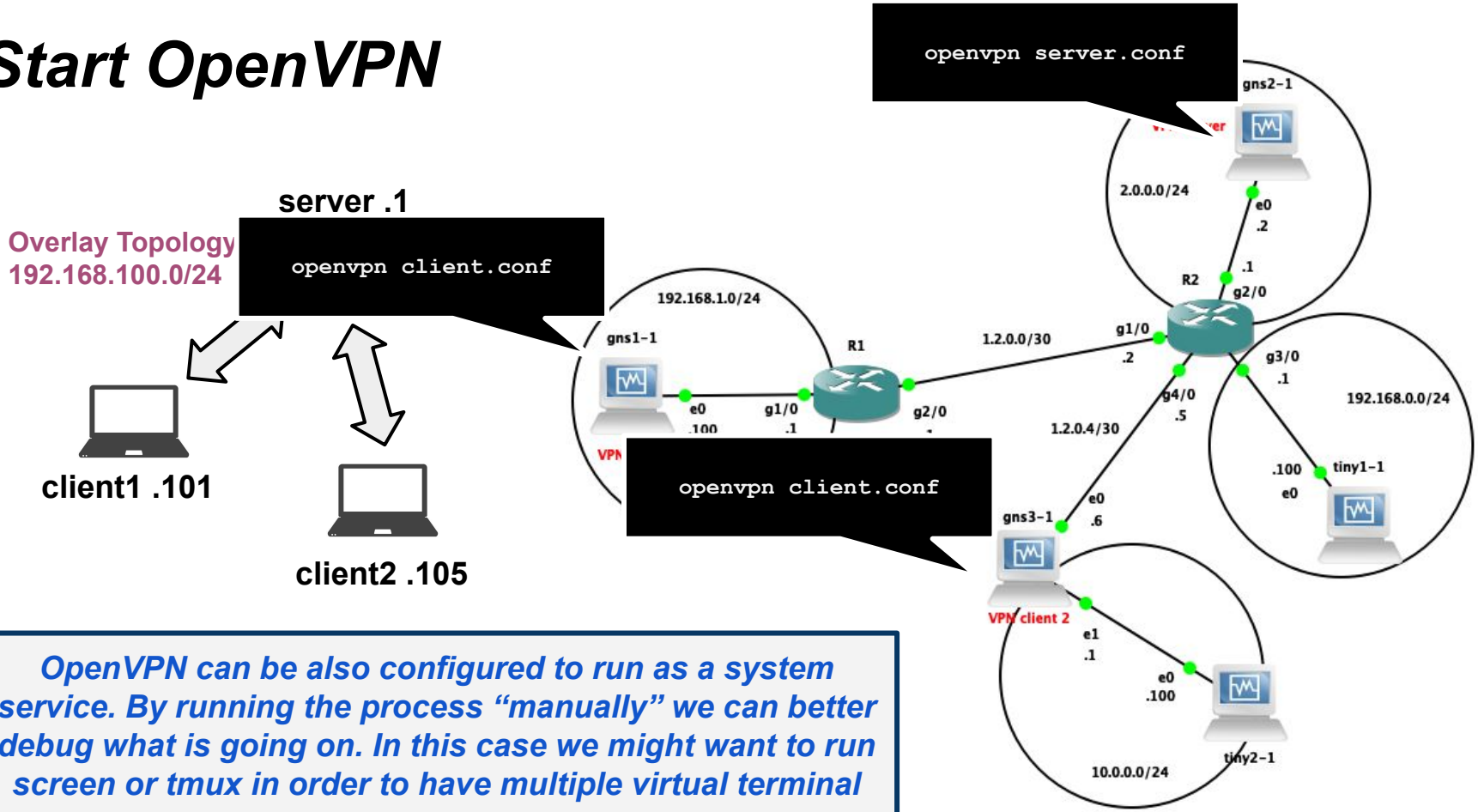
```
client
dev tun
proto udp
remote 2.0.0.2 1194
resolv-retry infinite
ca ca.crt
cert client2.crt
key client2.key
remote-cert-tls server
cipher AES-256-CBC
```





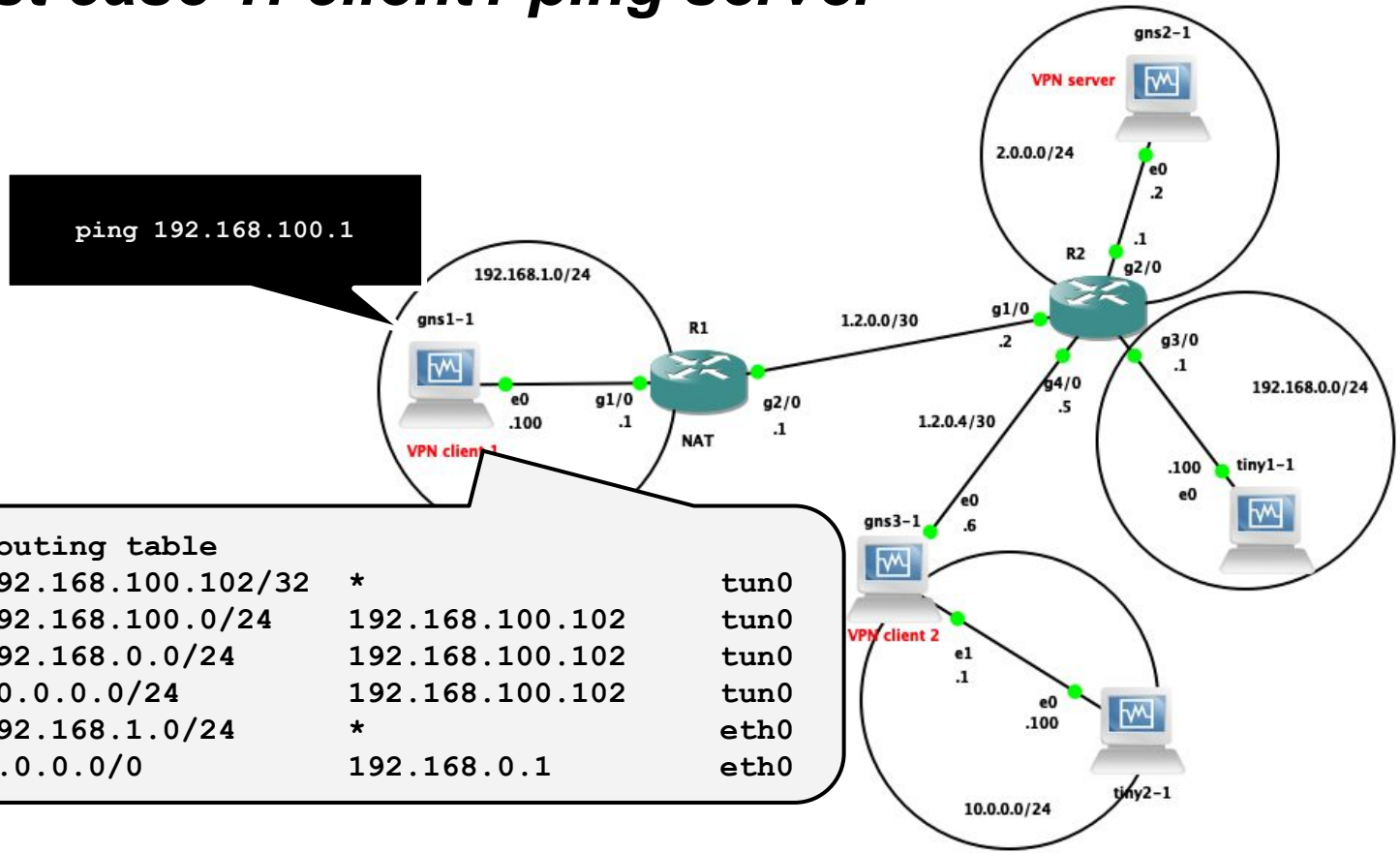

```
virtual IP addresses are assigned to client1 and
client2 according to the CCD configuration. The file
name must match the certificate CN.
#file ccd/client1
if-config-push 192.168.100.101 192.168.100.102
#file ccd/client2
if-config-push 192.168.100.105 192.168.100.106
```

Start OpenVPN



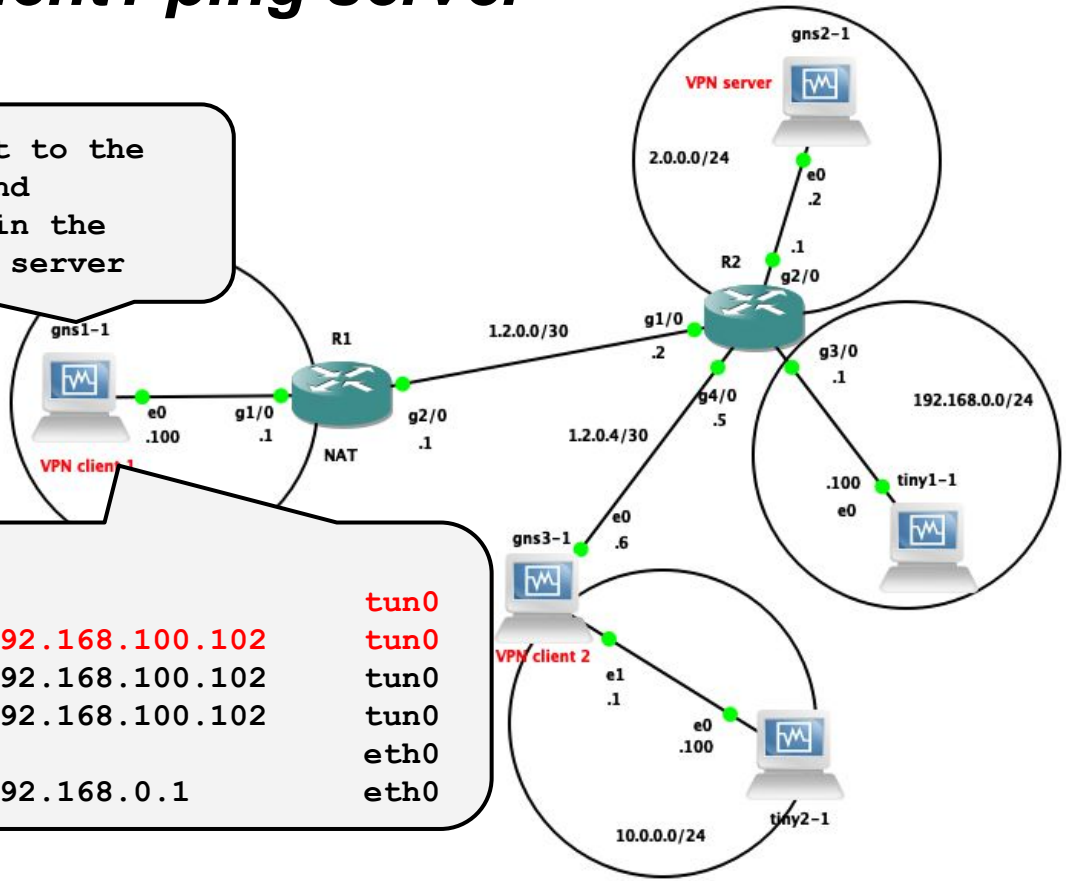
OpenVPN can be also configured to run as a system service. By running the process "manually" we can better debug what is going on. In this case we might want to run screen or tmux in order to have multiple virtual terminal

Test case 1: client1 ping server



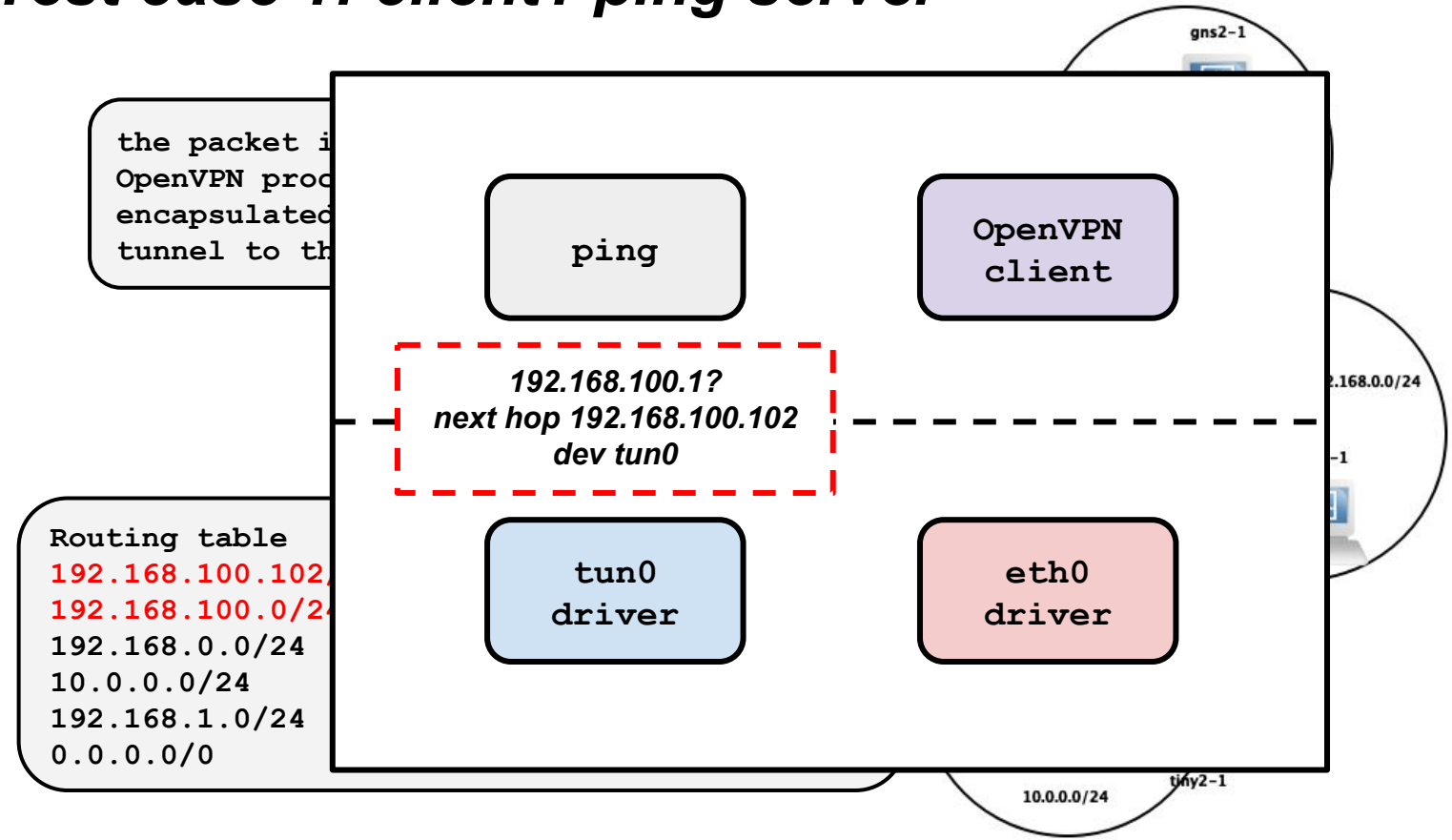
Test case 1: client1 ping server

the packet is sent to the OpenVPN process and encapsulated within the tunnel to the VPN server

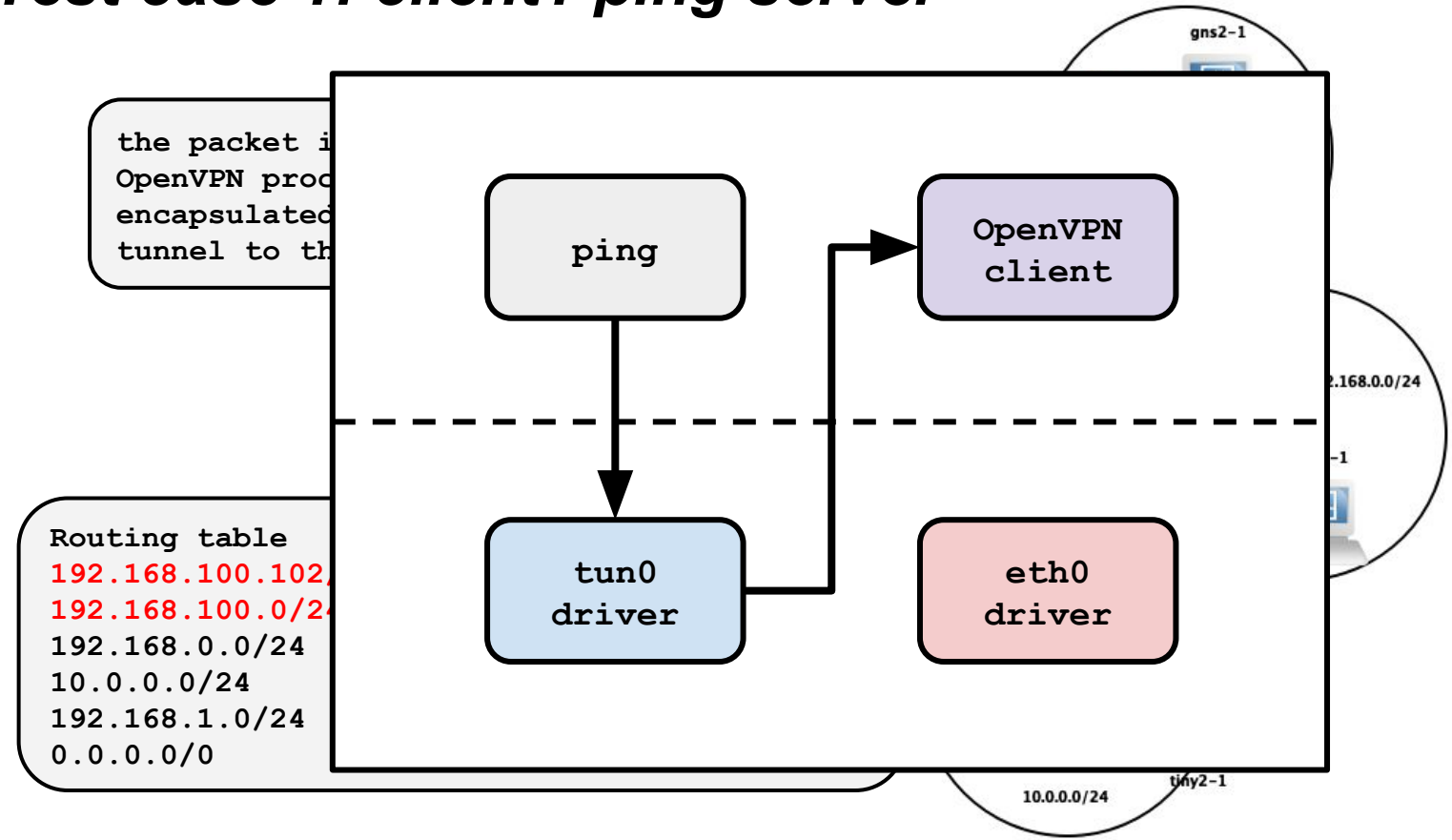



```
Routing table
192.168.100.102/32   *                 tun0
192.168.100.0/24     192.168.100.102   tun0
192.168.0.0/24       192.168.100.102   tun0
10.0.0.0/24          192.168.100.102   tun0
192.168.1.0/24       *                 eth0
0.0.0.0/0            192.168.0.1       eth0
```

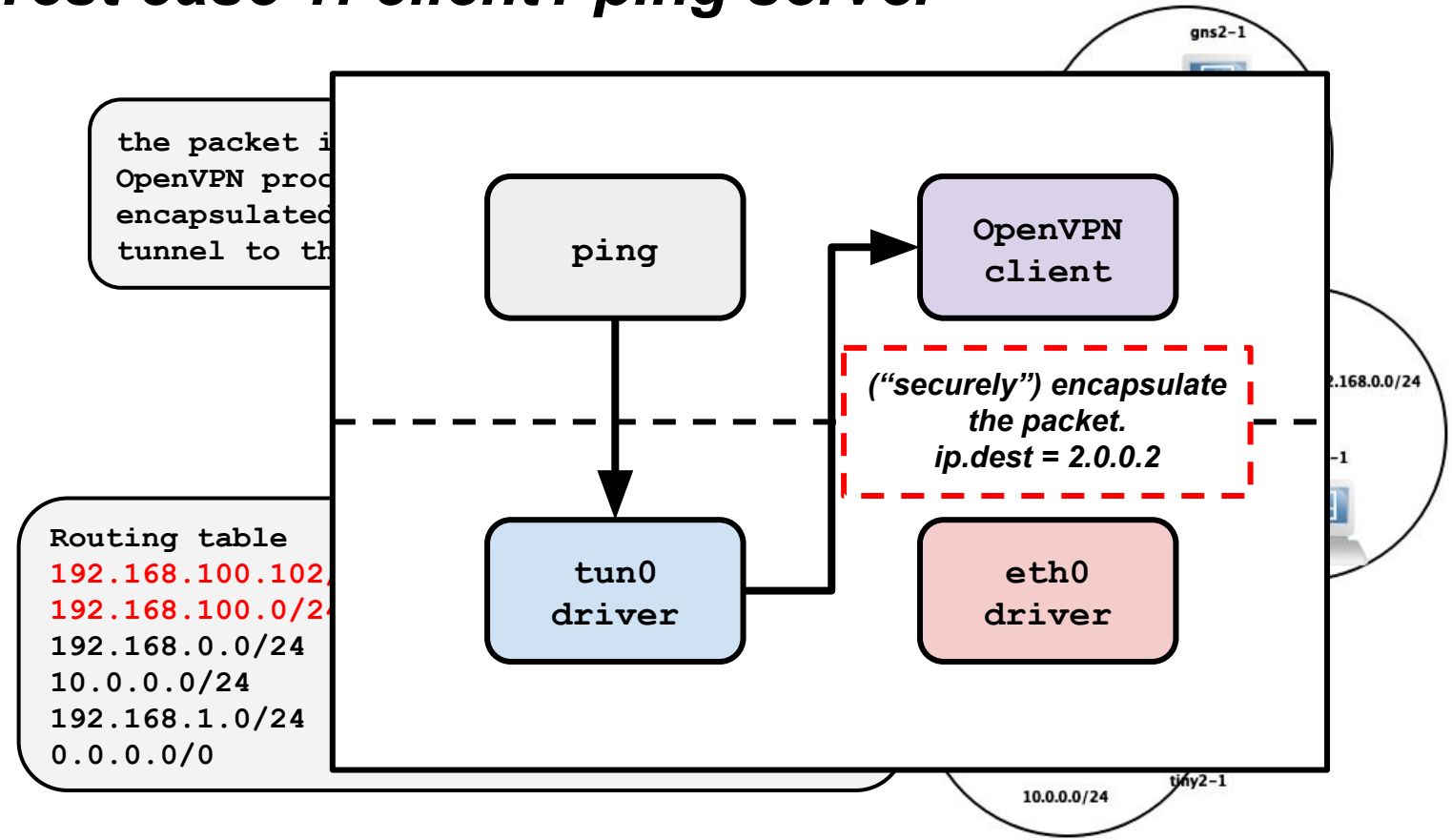
Test case 1: client1 ping server



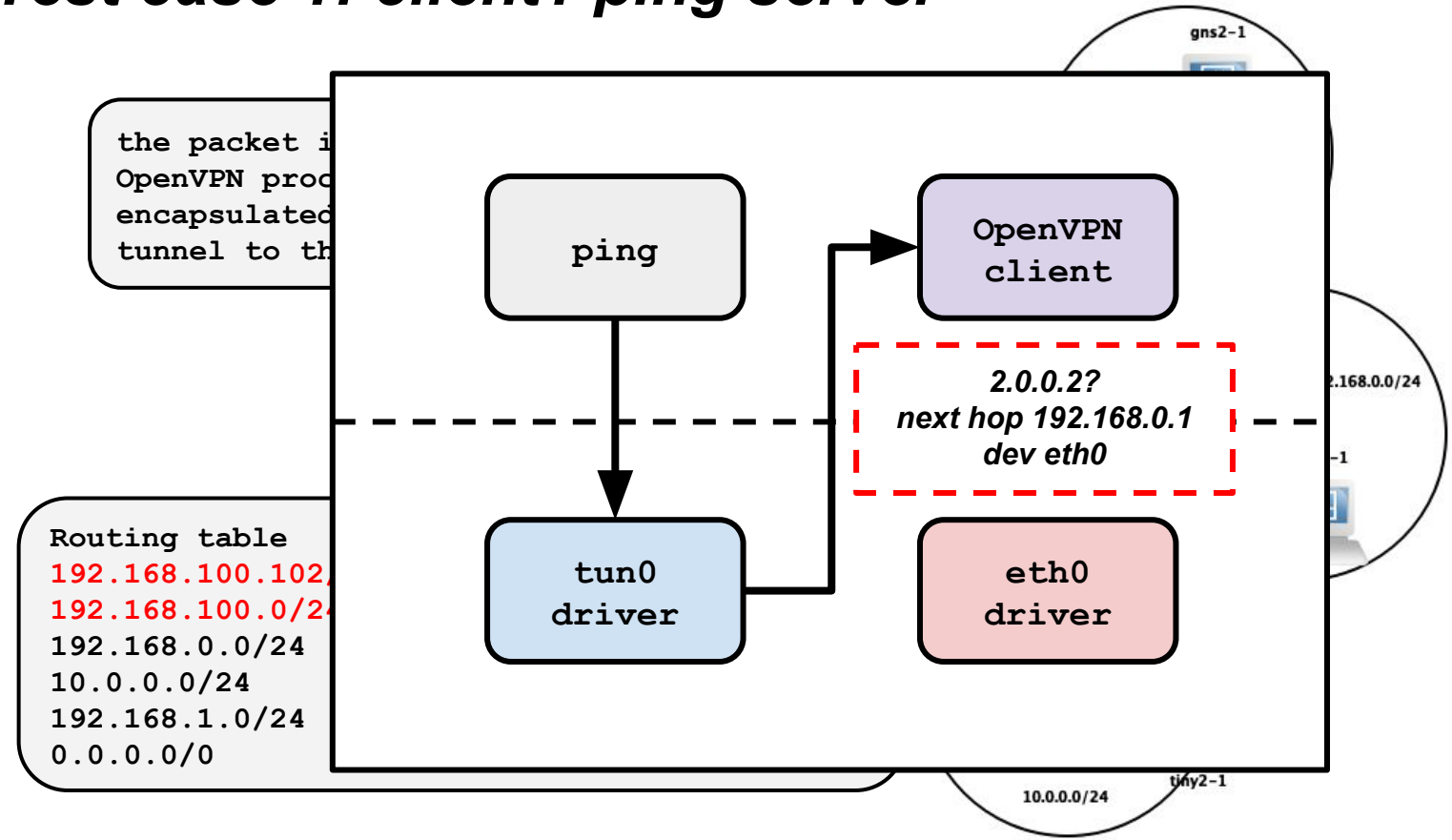
Test case 1: client1 ping server



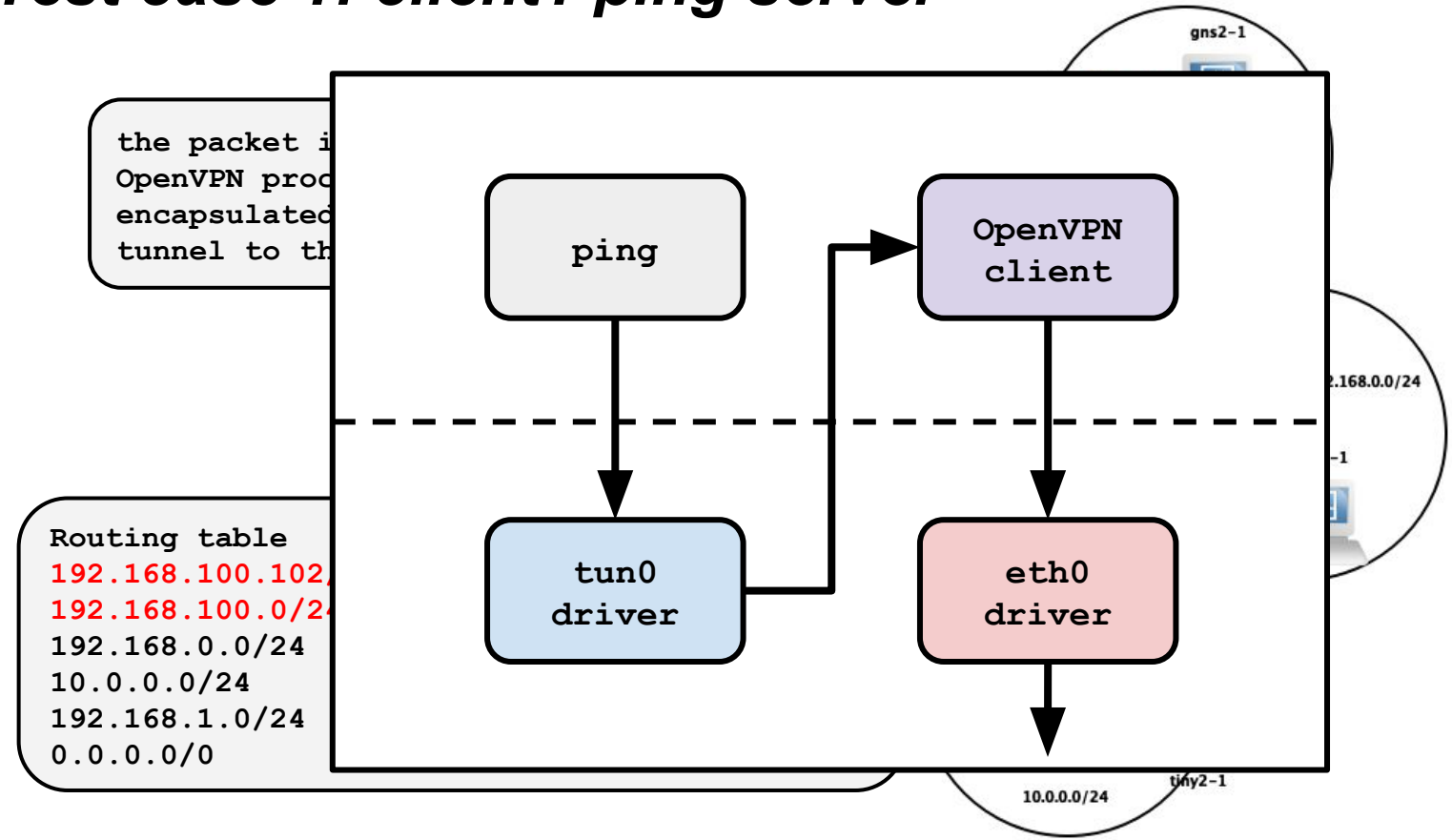
Test case 1: client1 ping server



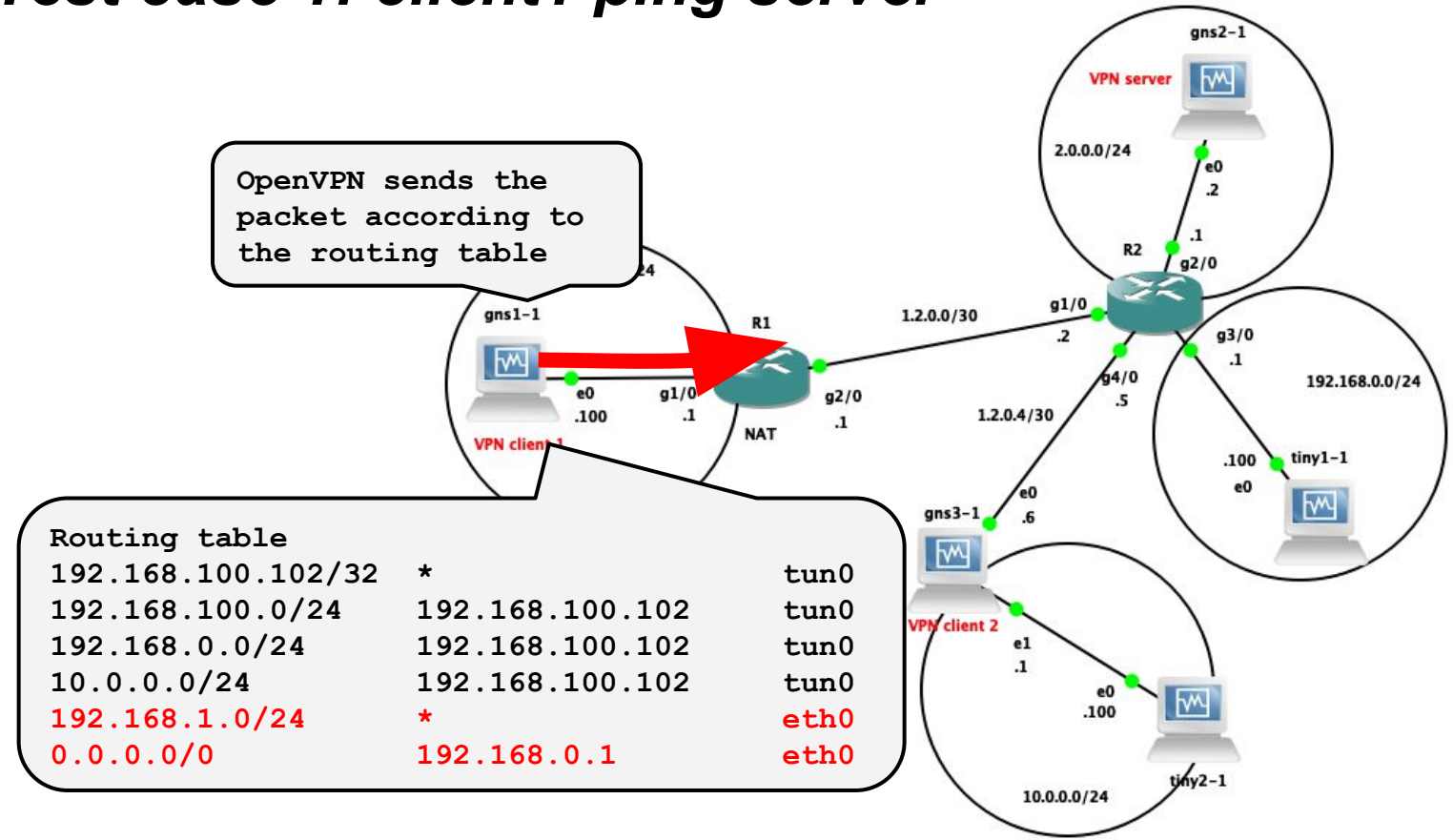
Test case 1: client1 ping server



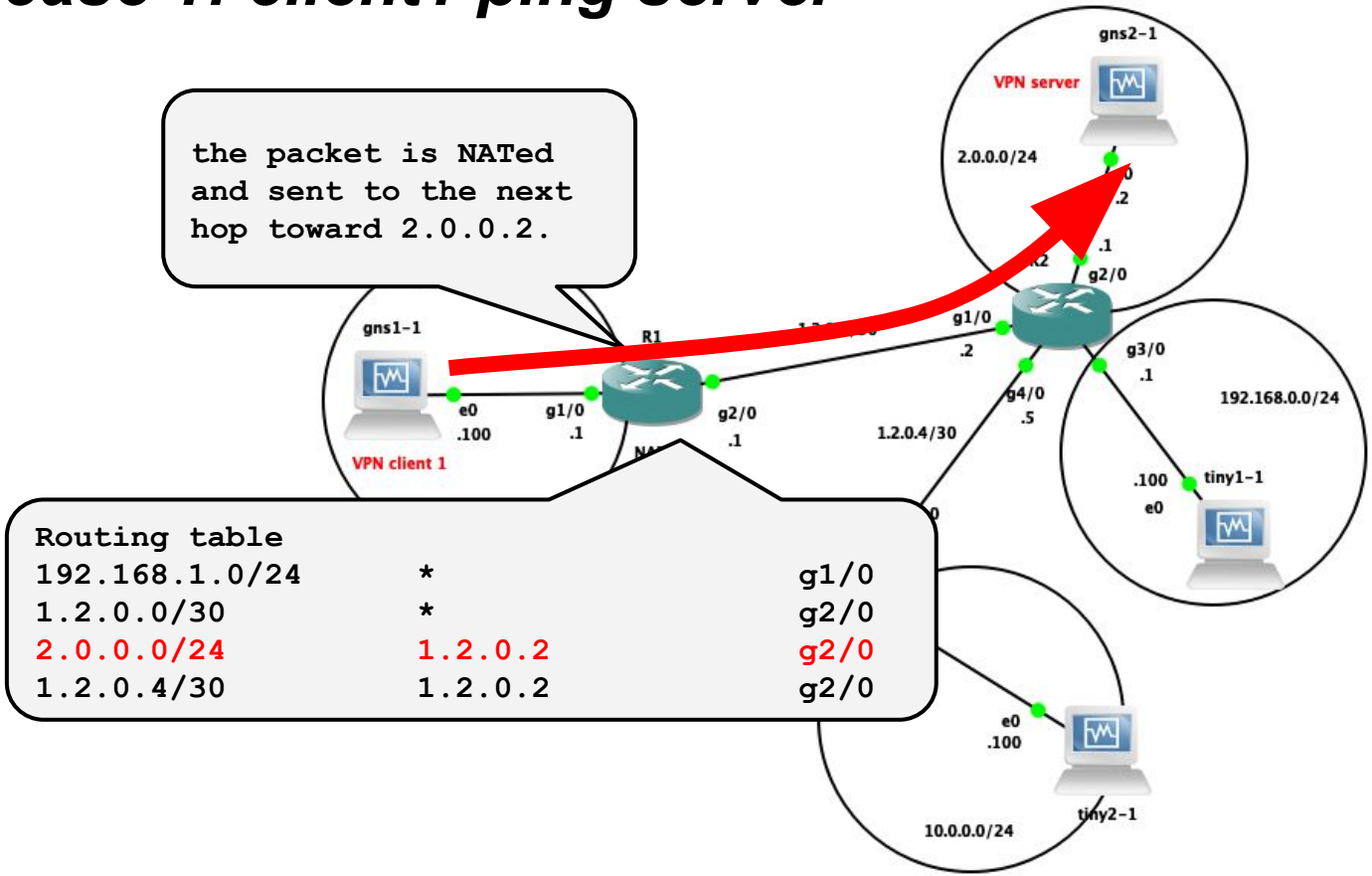
Test case 1: client1 ping server



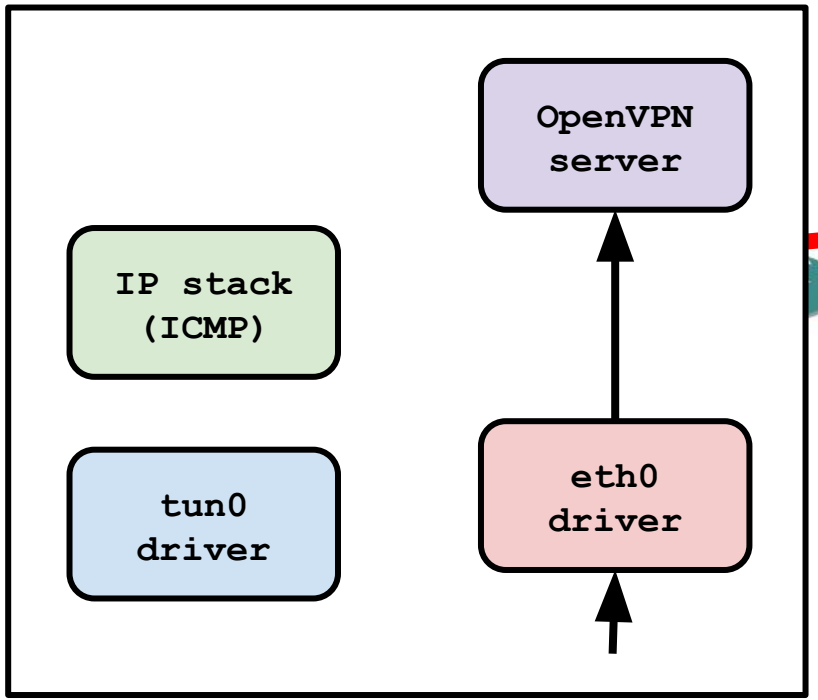
Test case 1: client1 ping server

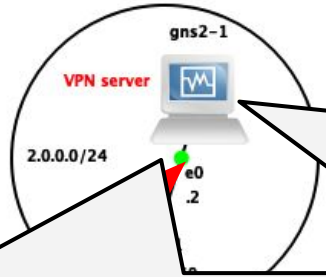


Test case 1: client1 ping server



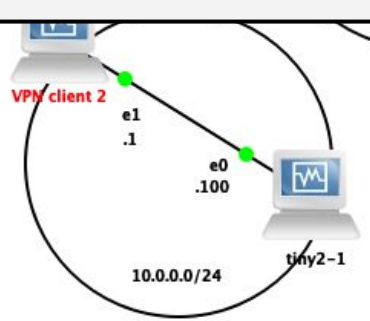
Test case 1: client1 ping server



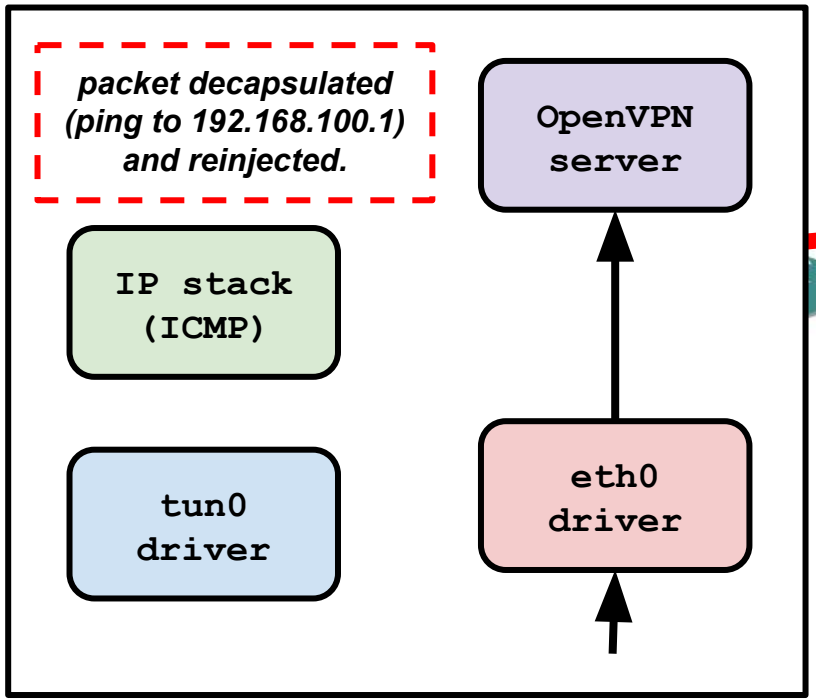


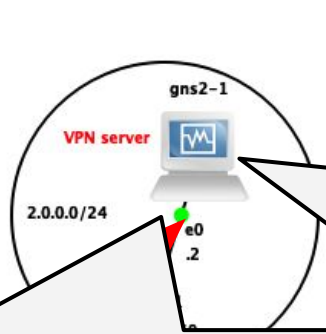
The encapsulated packet is received by OpenVPN. The RX process is the inverse of the TX process

```
Routing table
192.168.100.2/32     *                 tun0
192.168.100.0/24     192.168.100.102   tun0
10.0.0.0/24          192.168.100.102   tun0
2.0.0.0/24           *                 eth0
0.0.0.0/0            2.0.0.1           eth0
```



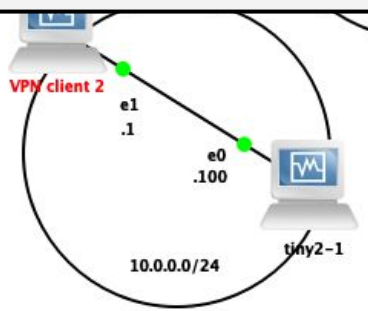
Test case 1: client1 ping server



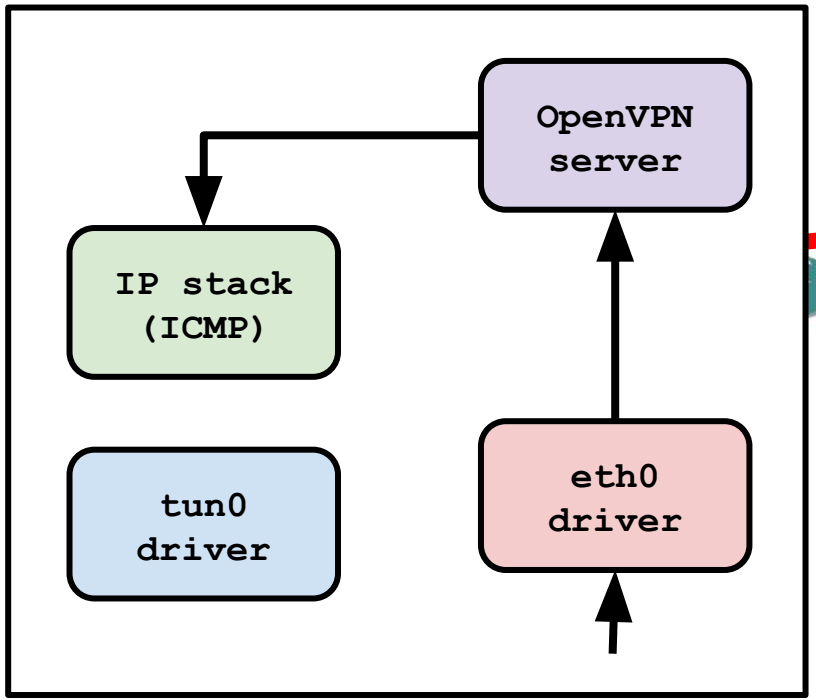


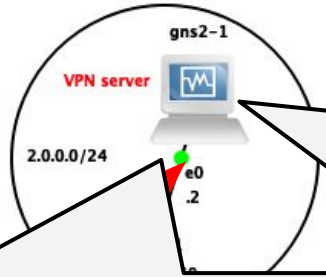
The encapsulated packet is received by OpenVPN. The RX process is the inverse of the TX process

```
Routing table
192.168.100.2/32     *                 tun0
192.168.100.0/24     192.168.100.102   tun0
10.0.0.0/24          192.168.100.102   tun0
2.0.0.0/24           *                 eth0
0.0.0.0/0            2.0.0.1           eth0
```



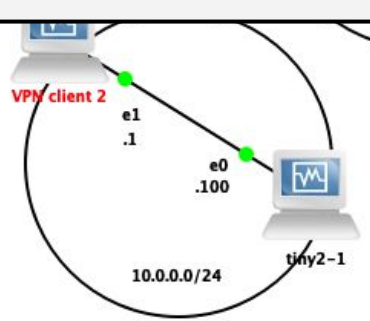
Test case 1: client1 ping server



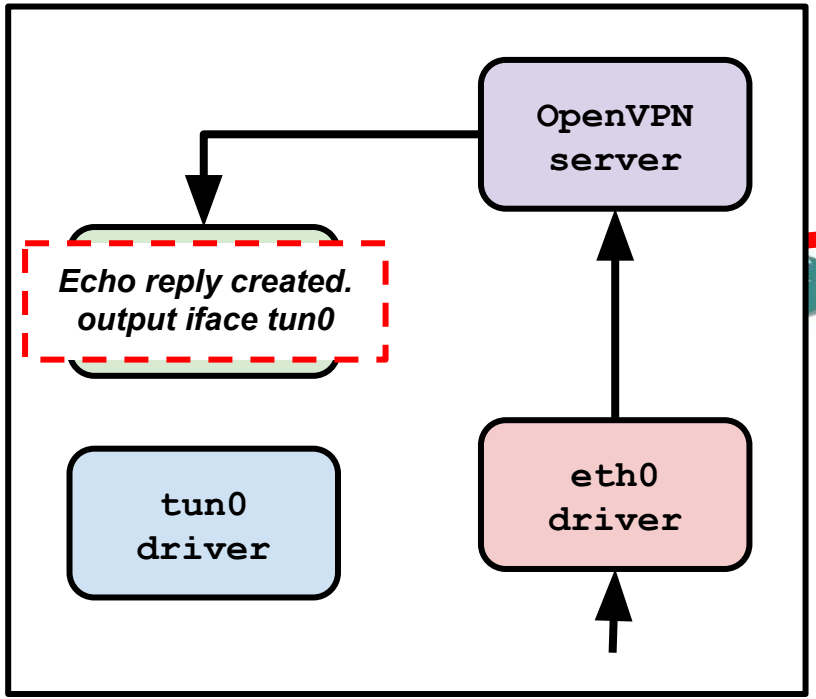


The encapsulated packet is received by OpenVPN. The RX process is the inverse of the TX process

```
Routing table
192.168.100.2/32     *                 tun0
192.168.100.0/24     192.168.100.102   tun0
10.0.0.0/24          192.168.100.102   tun0
2.0.0.0/24           *                 eth0
0.0.0.0/0            2.0.0.1           eth0
```



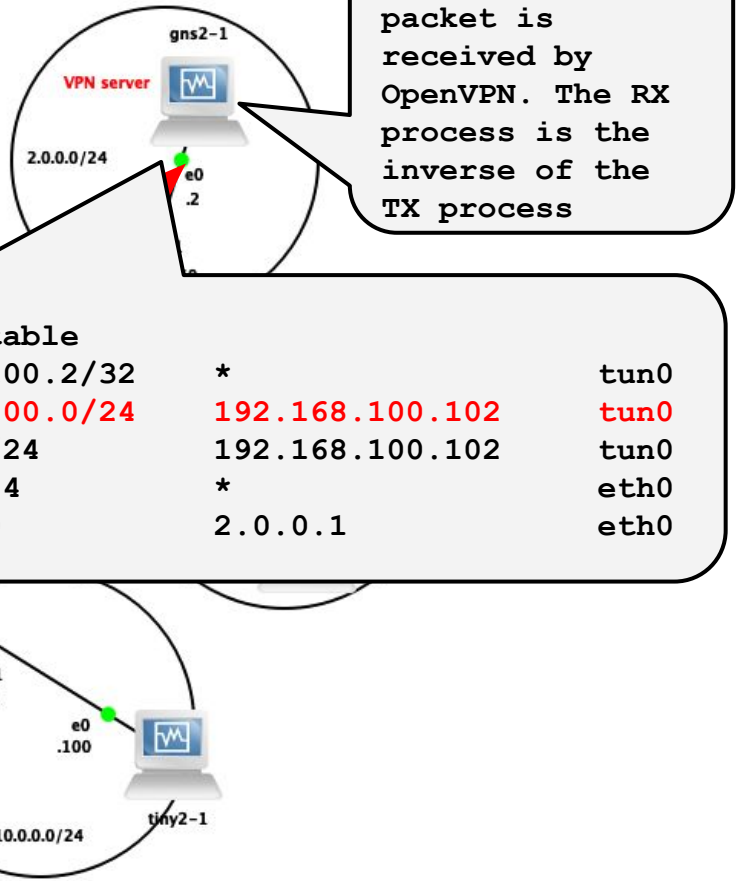
Test case 1: client1 ping server



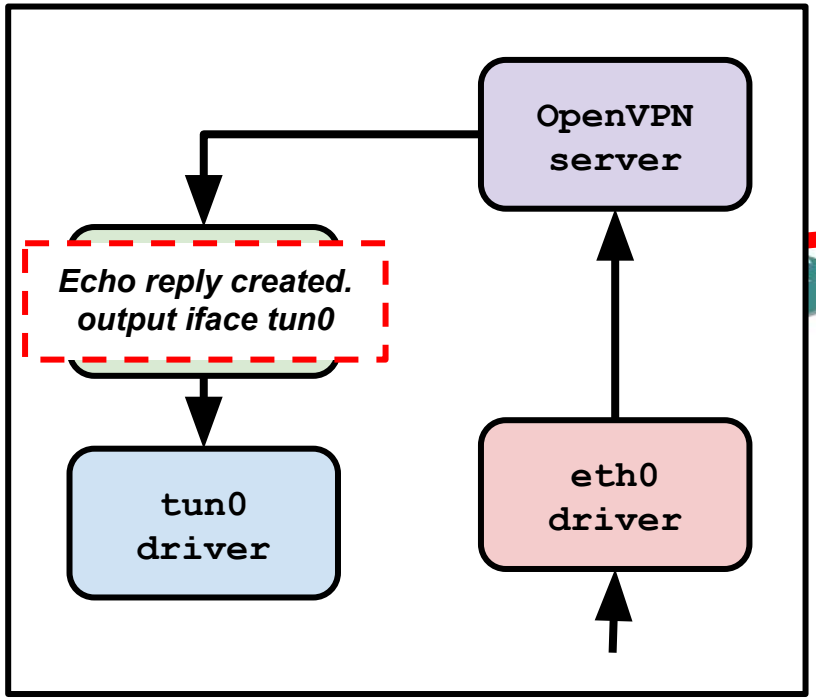
The encapsulated packet is received by OpenVPN. The RX process is the inverse of the TX process

Routing table

192.168.100.2/32	*	tun0
192.168.100.0/24	192.168.100.102	tun0
10.0.0.0/24	192.168.100.102	tun0
2.0.0.0/24	*	eth0
0.0.0.0/0	2.0.0.1	eth0



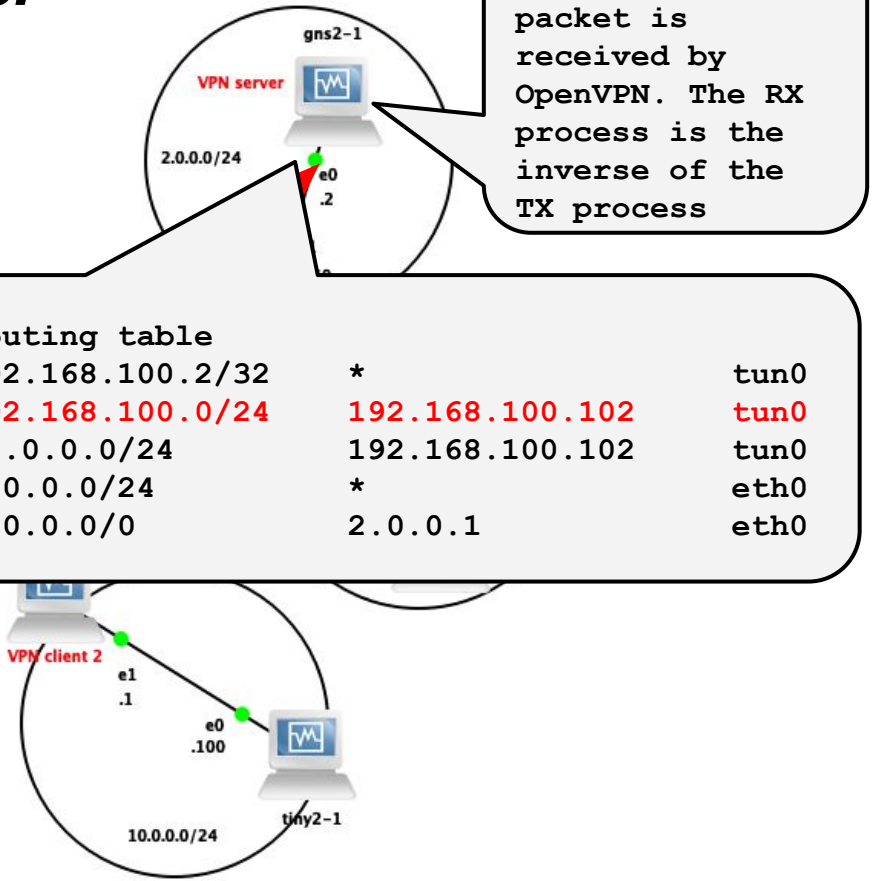
Test case 1: client1 ping server



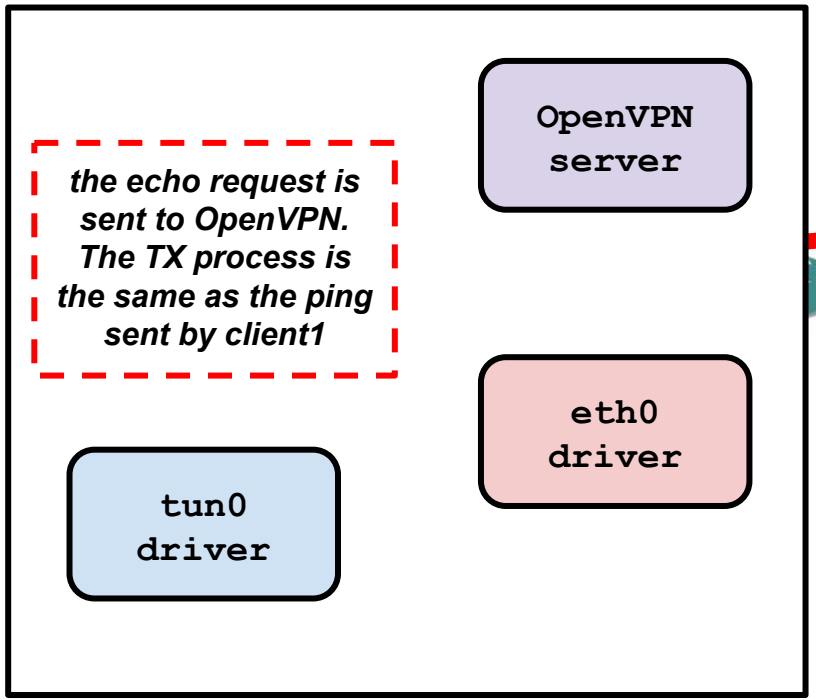
The encapsulated packet is received by OpenVPN. The RX process is the inverse of the TX process

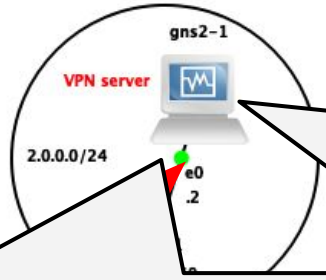
Routing table

192.168.100.2/32	*	tun0
192.168.100.0/24	192.168.100.102	tun0
10.0.0.0/24	192.168.100.102	tun0
2.0.0.0/24	*	eth0
0.0.0.0/0	2.0.0.1	eth0



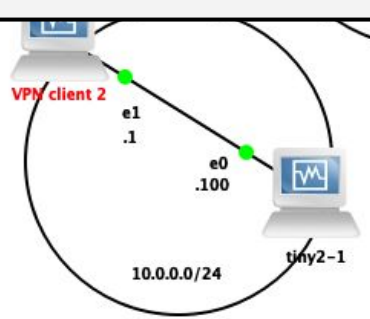
Test case 1: client1 ping server



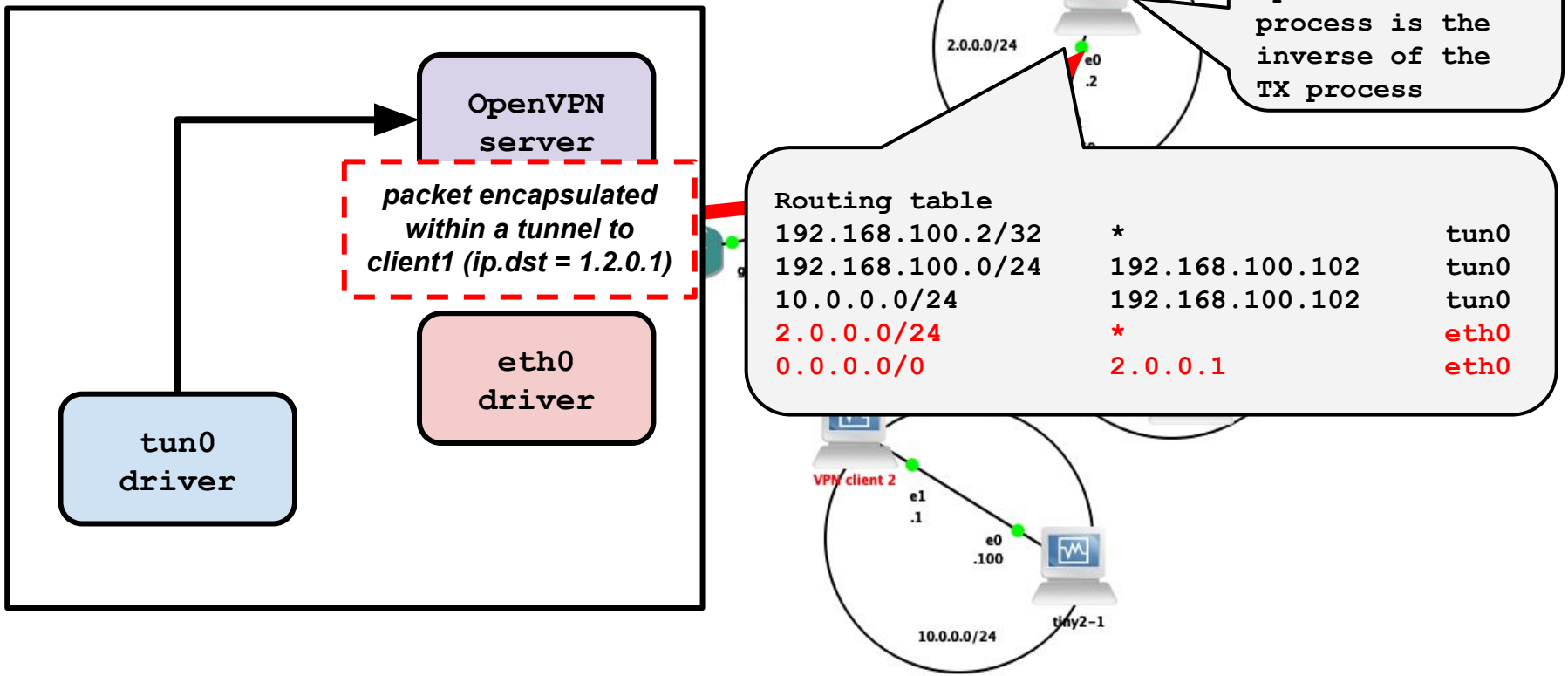


The encapsulated packet is received by OpenVPN. The RX process is the inverse of the TX process

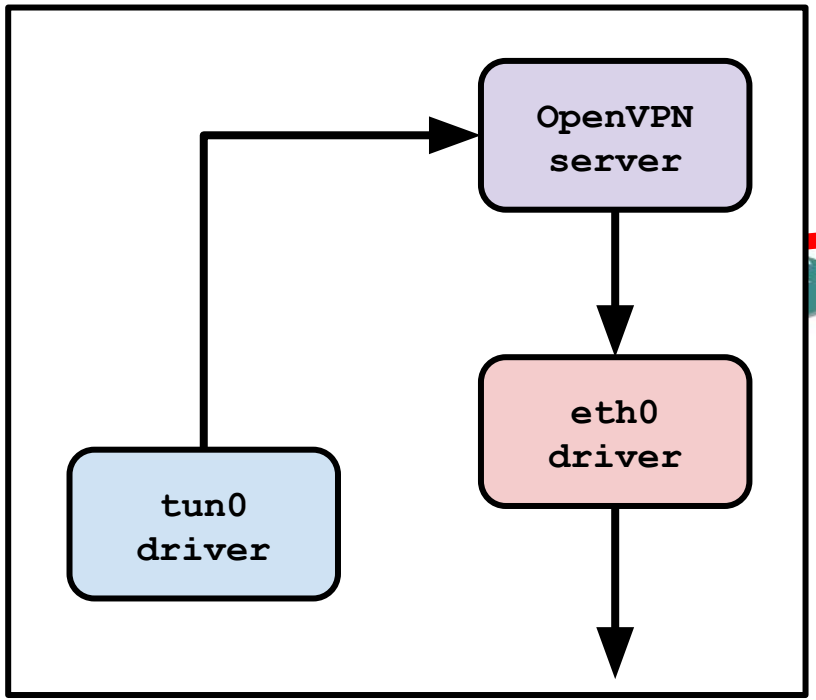
```
Routing table
192.168.100.2/32     *                  tun0
192.168.100.0/24     192.168.100.102    tun0
10.0.0.0/24          192.168.100.102    tun0
2.0.0.0/24           *                  eth0
0.0.0.0/0            2.0.0.1            eth0
```

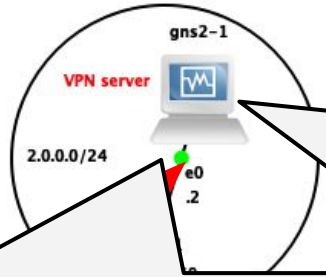


Test case 1: client1 ping server



Test case 1: client1 ping server

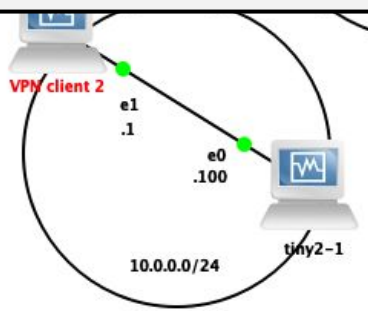




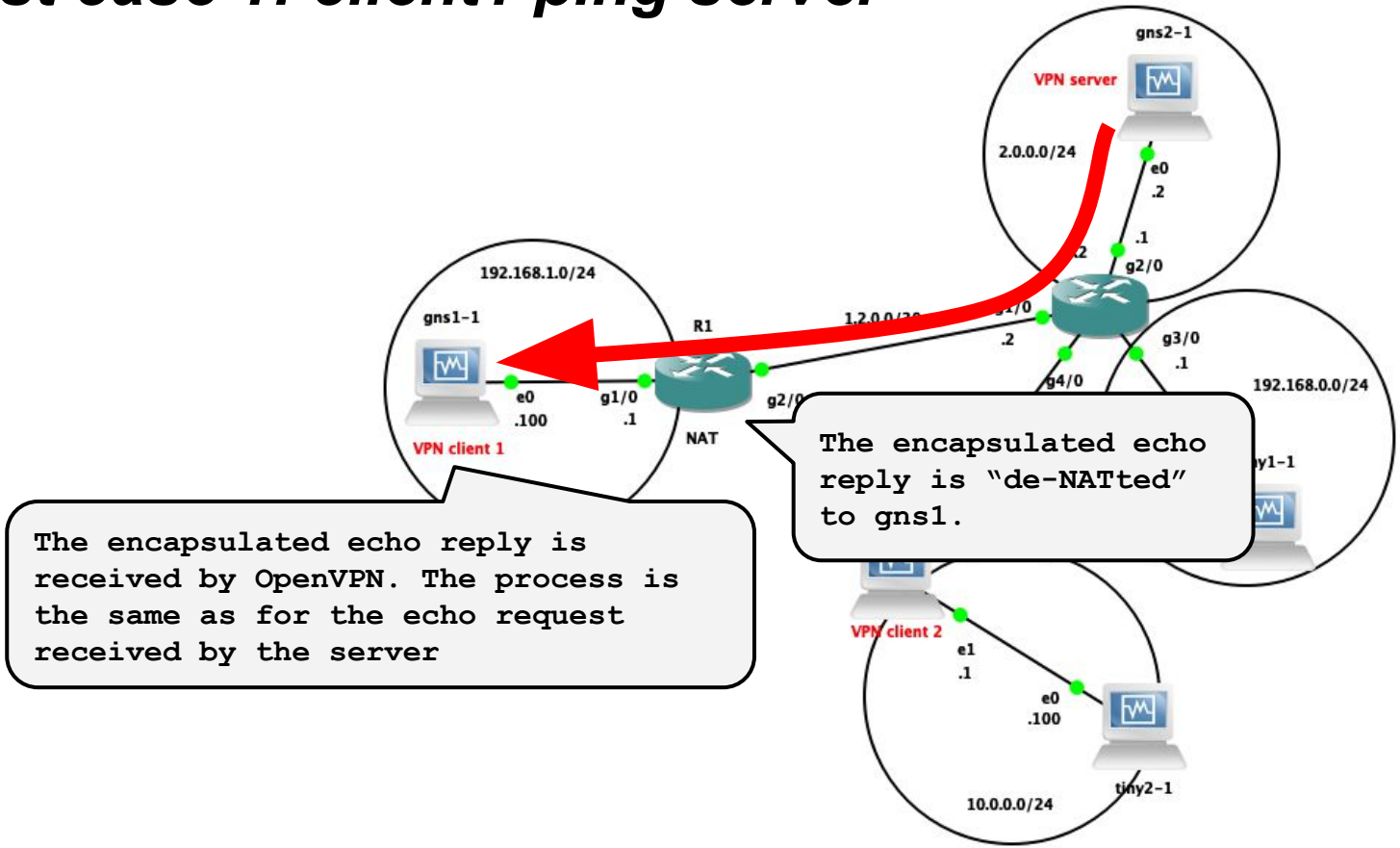
The encapsulated packet is received by OpenVPN. The RX process is the inverse of the TX process

Routing table

192.168.100.2/32	*	tun0
192.168.100.0/24	192.168.100.102	tun0
10.0.0.0/24	192.168.100.102	tun0
2.0.0.0/24	*	eth0
0.0.0.0/0	2.0.0.1	eth0



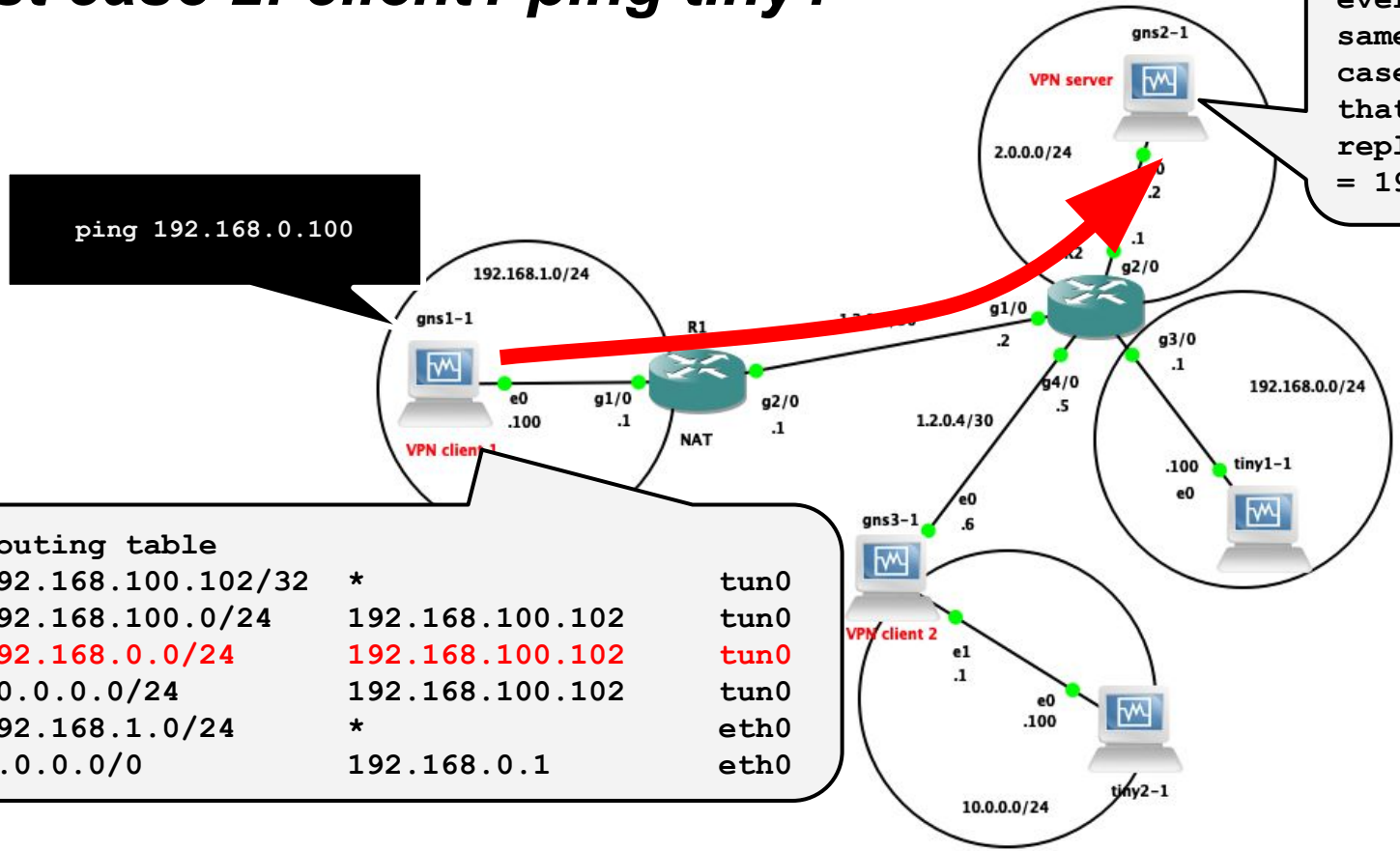
Test case 1: client1 ping server



Test case 2: client1 ping tiny1

everyting is the same as test case 1 except that the ehoc reply has ip.dst = 192.168.0.100

ping 192.168.0.100

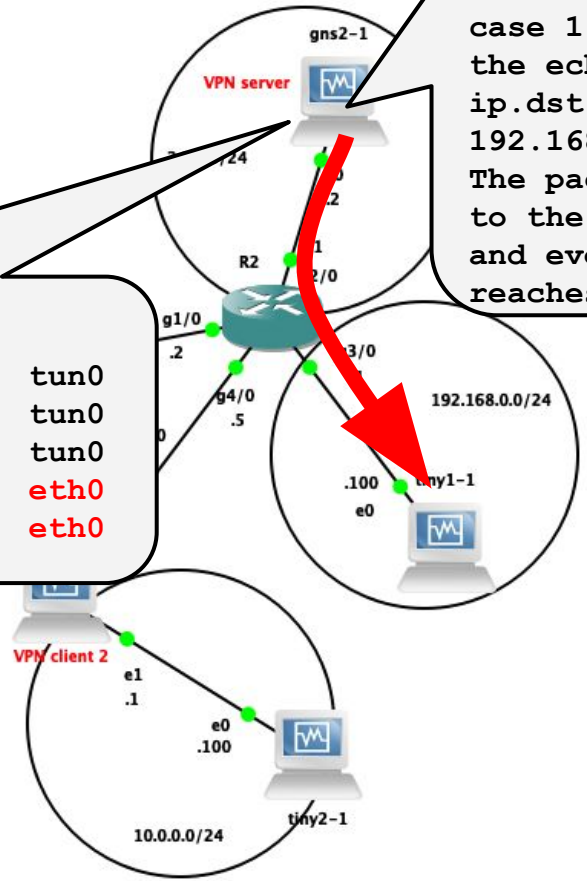



```
Routing table
192.168.100.102/32   *                 tun0
192.168.100.0/24     192.168.100.102   tun0
192.168.0.0/24       192.168.100.102   tun0
10.0.0.0/24          192.168.100.102   tun0
192.168.1.0/24       *                 eth0
0.0.0.0/0            192.168.0.1       eth0
```

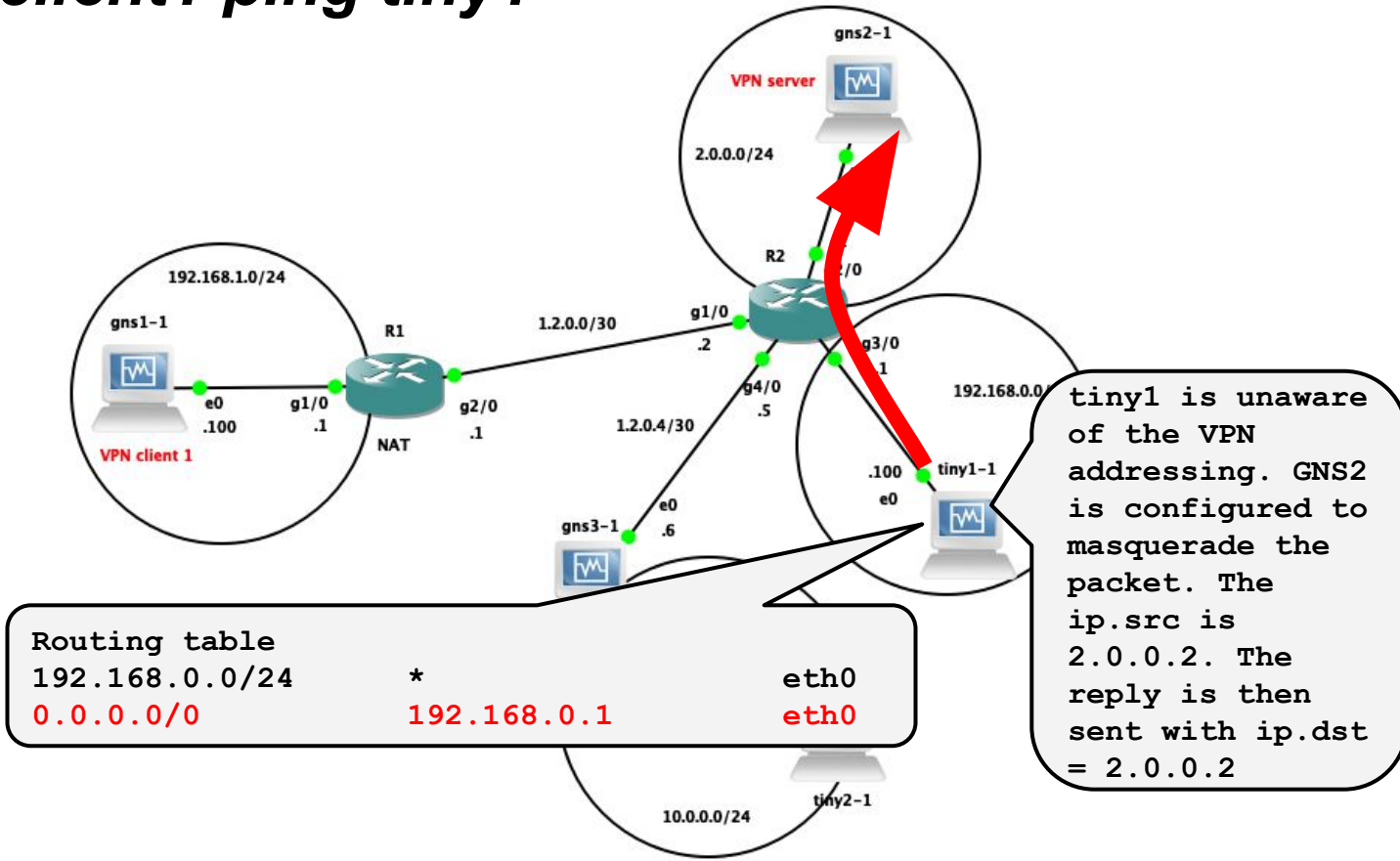
Test case 2: client1 ping tiny1

everyting is the same as in test case 1 except that the echo reply has ip.dst = 192.168.0.100
The packet is sent to the default GW and eventually reaches tiny1

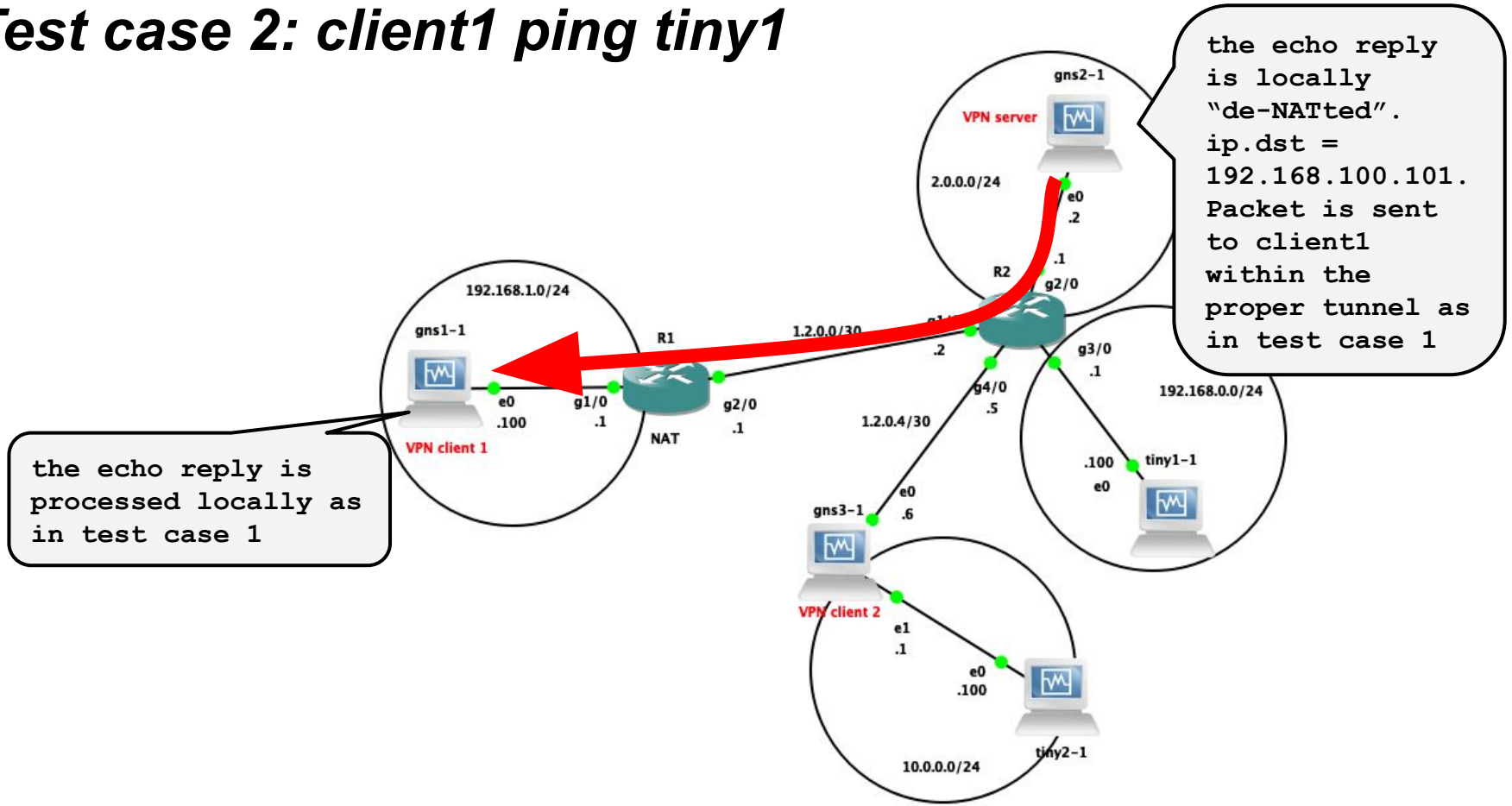
```
Routing table
192.168.100.2/32     *                  tun0
192.168.100.0/24     192.168.100.102    tun0
10.0.0.0/24          192.168.100.102    tun0
2.0.0.0/24           *                  eth0
0.0.0.0/0            2.0.0.1            eth0
```



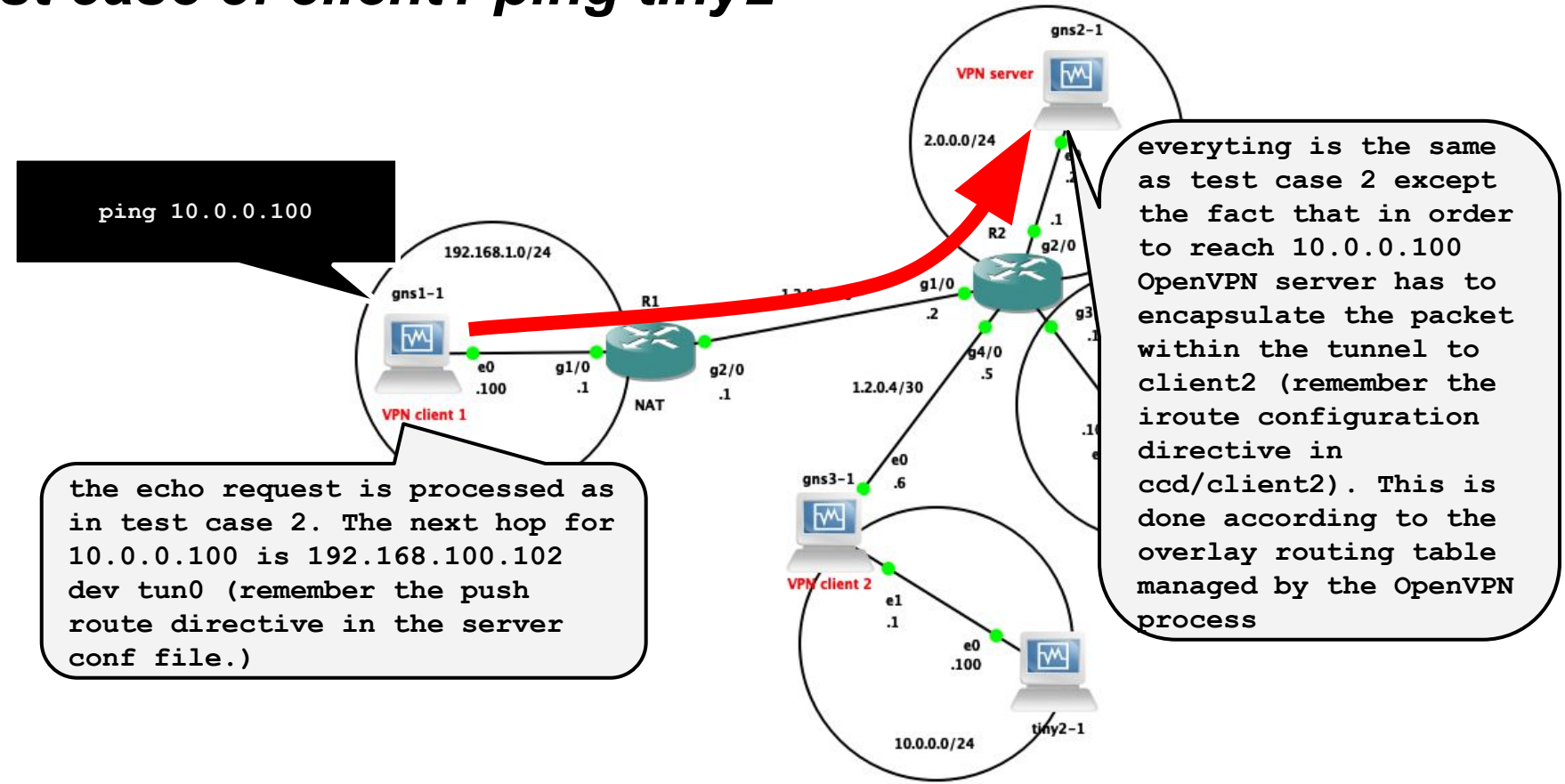
Test case 2: client1 ping tiny1



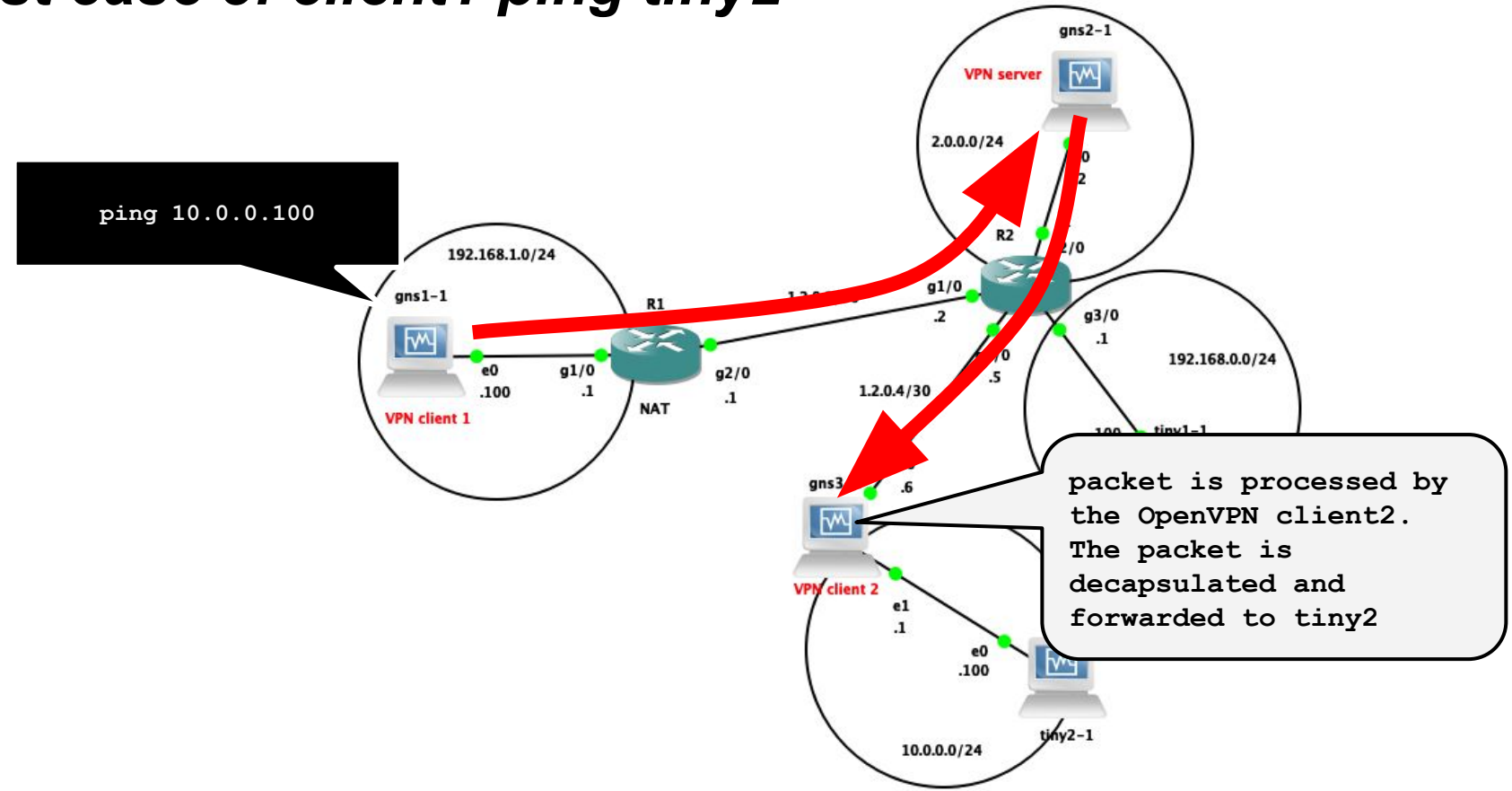
Test case 2: client1 ping tiny1



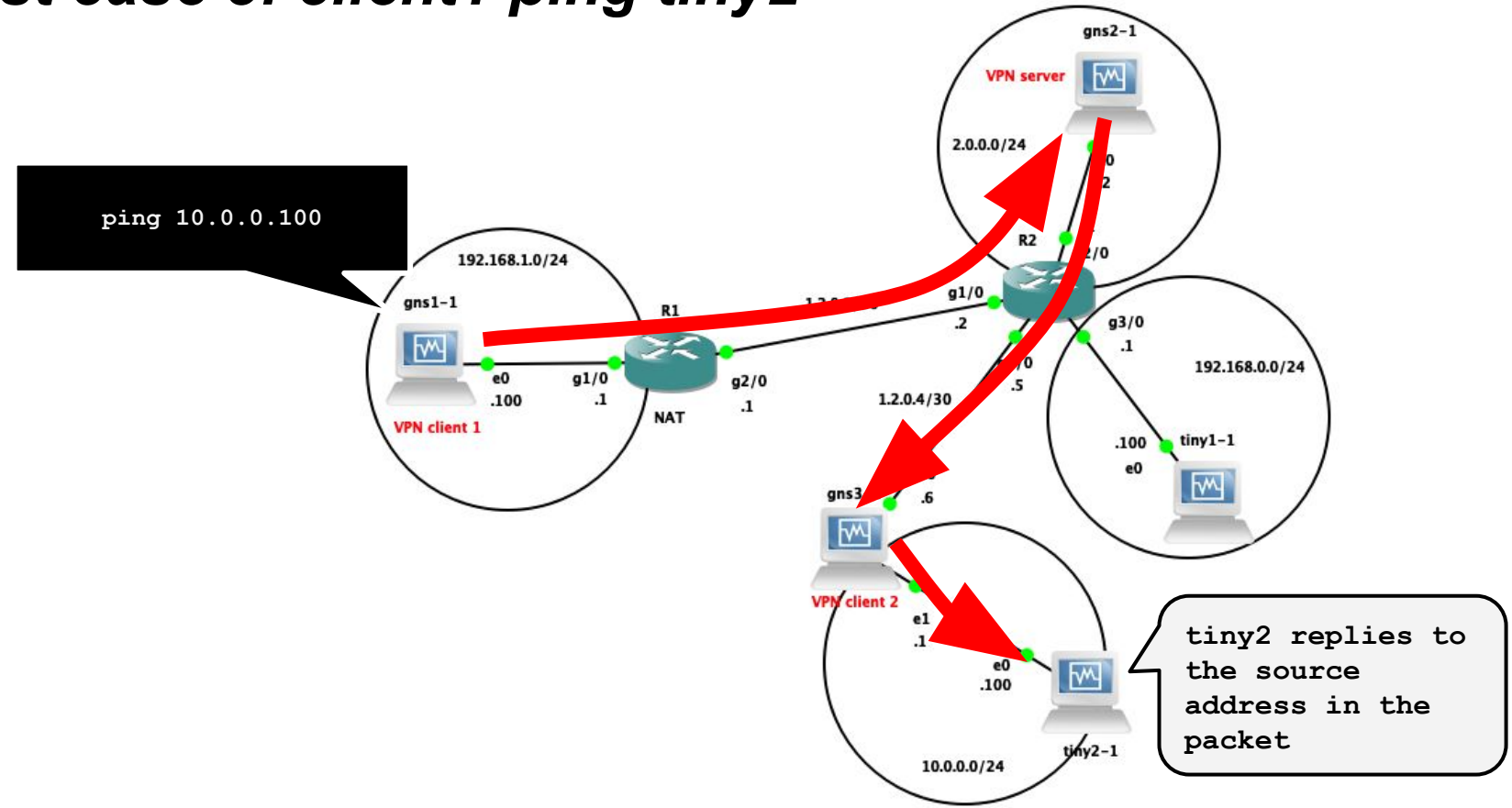
Test case 3: client1 ping tiny2



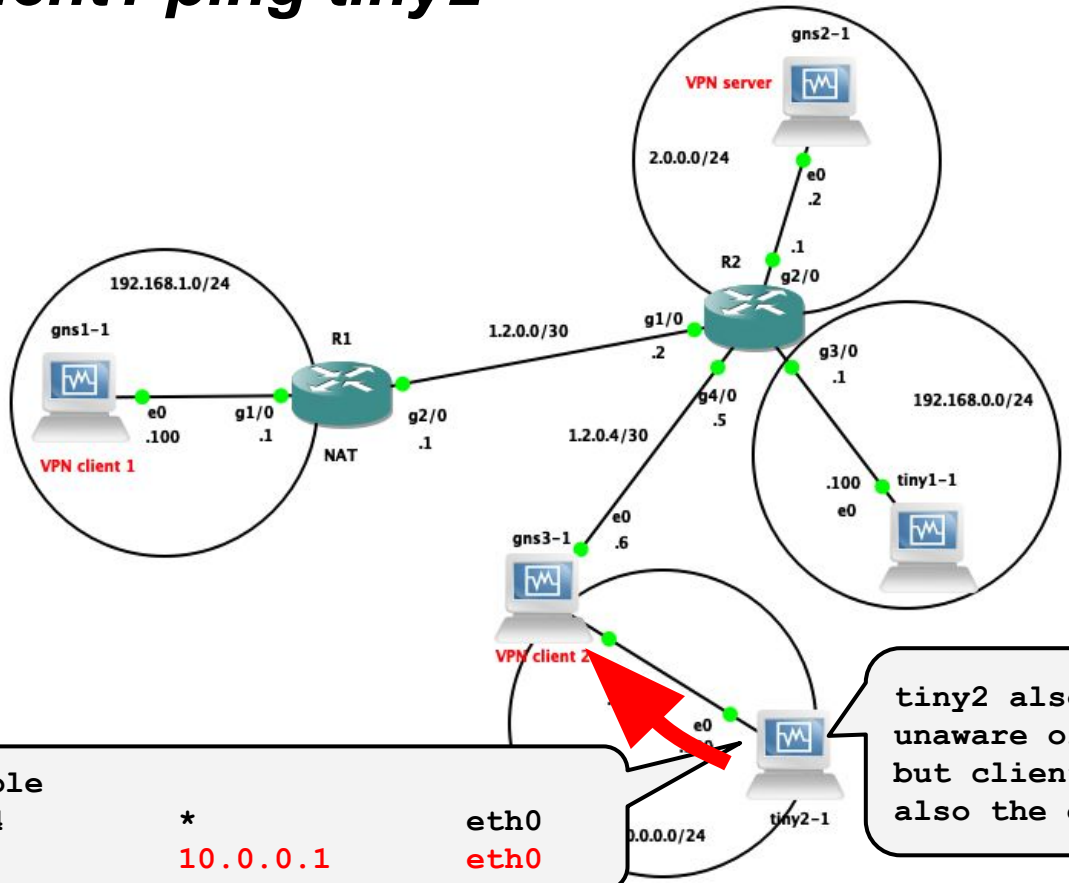
Test case 3: client1 ping tiny2



Test case 3: client1 ping tiny2



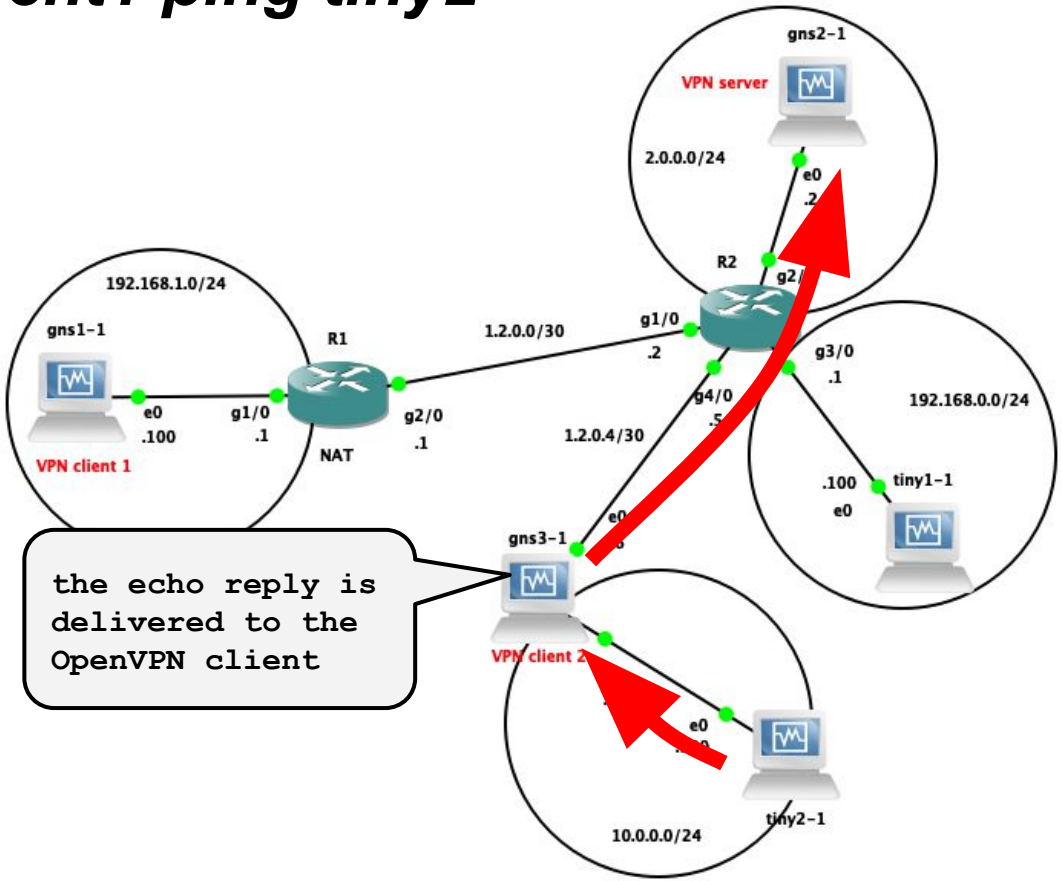
Test case 3: client1 ping tiny2



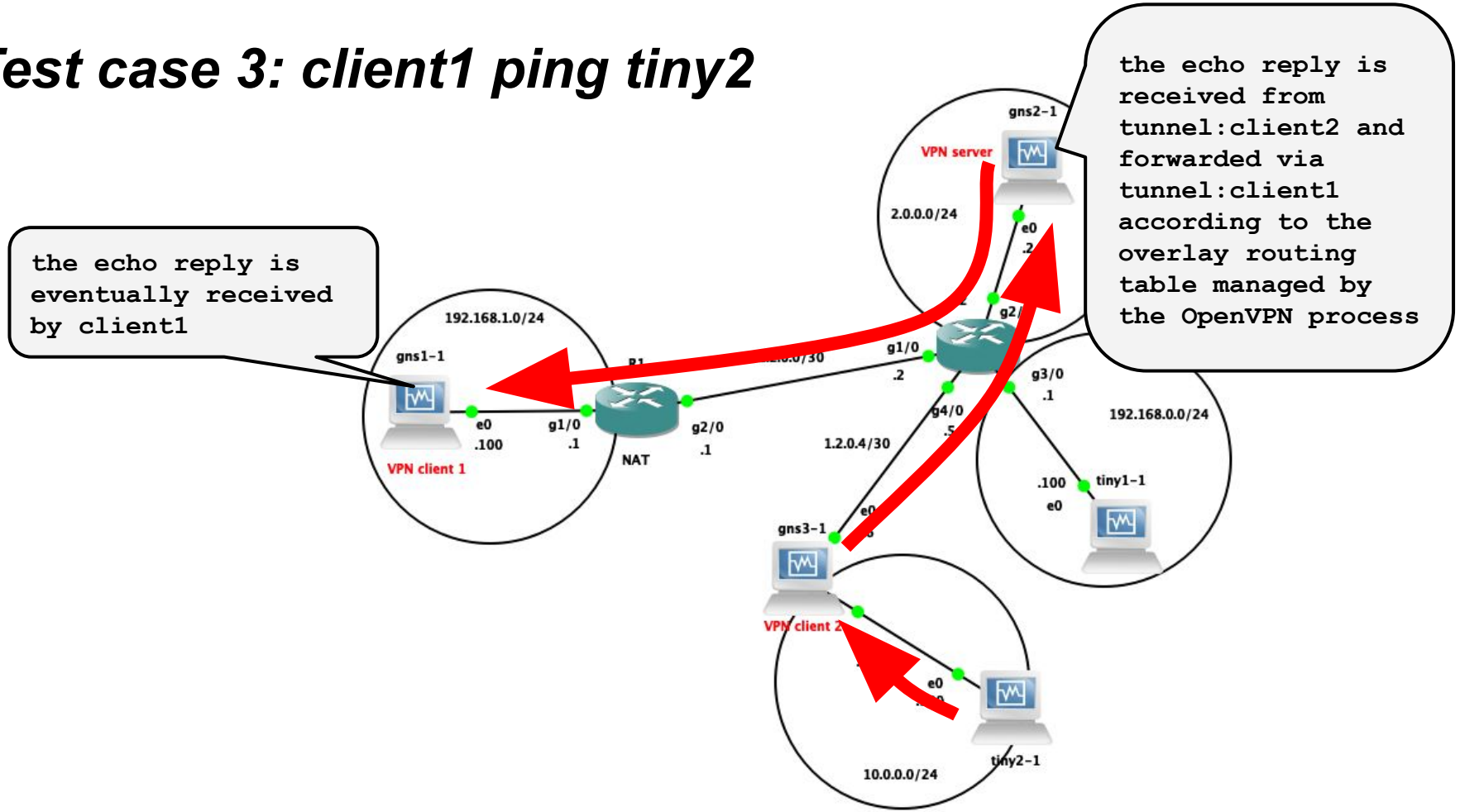

```
Routing table
10.0.0.0/24          *           eth0
0.0.0.0/0            10.0.0.1    eth0
```

tiny2 also is unaware of the VPN but client 2 is also the default GW

Test case 3: client1 ping tiny2



Test case 3: client1 ping tiny2



Interaction with NETFILTER (host firewall on the server)
(assuming everything in DROP)

- open openvpn listening destination port (udp or tcp) in INPUT
  - this is for the initial (D)TLS handshake and for all the subsequent encrypted packets
- allow also packets in OUTPUT (if needed). The same port can be used (but in this case this is the source port)
- allow packets to/from tun0
  - this may involve all chains: input, output, forward (in case of client-to-client)