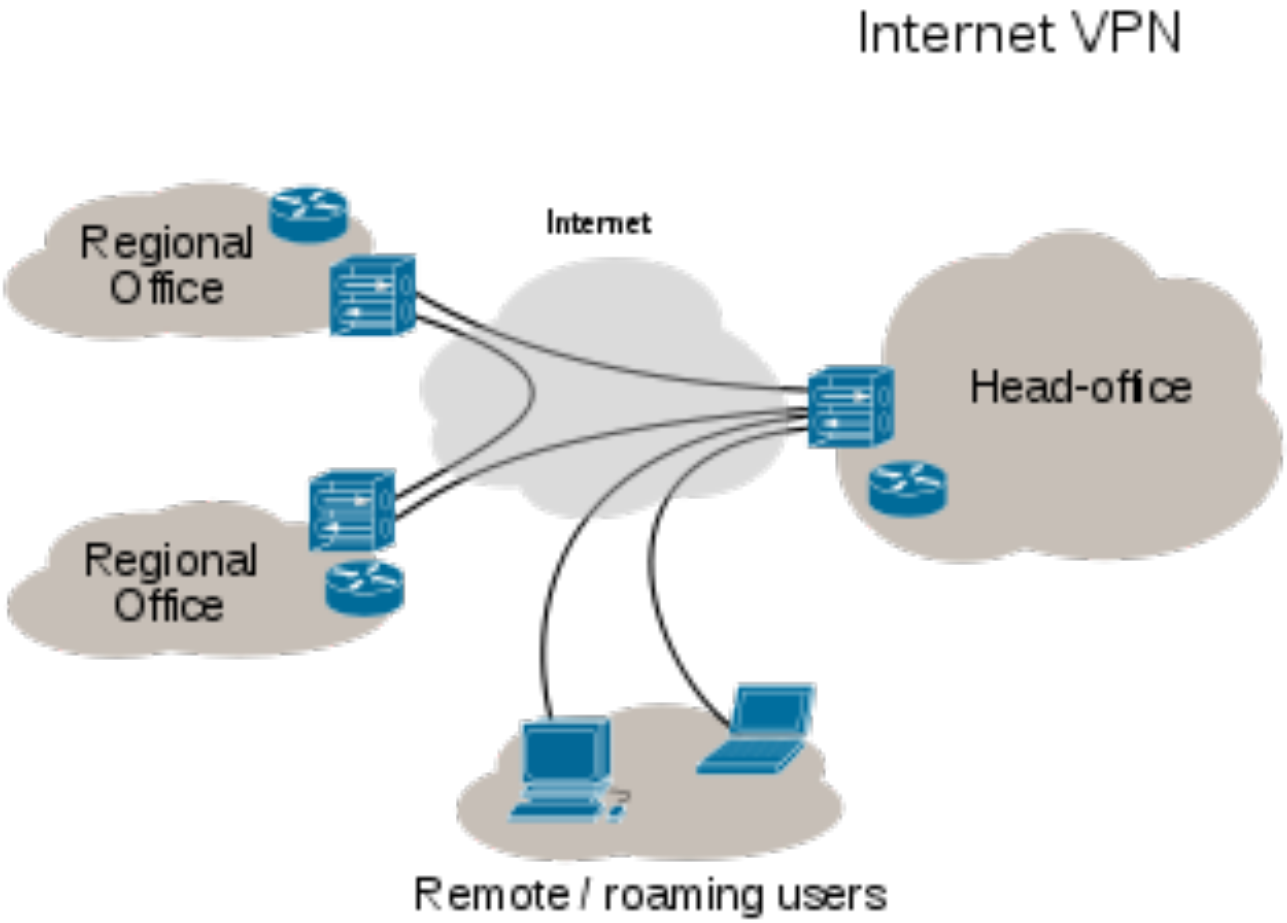
VIRTUAL PRIVATE NETWORK

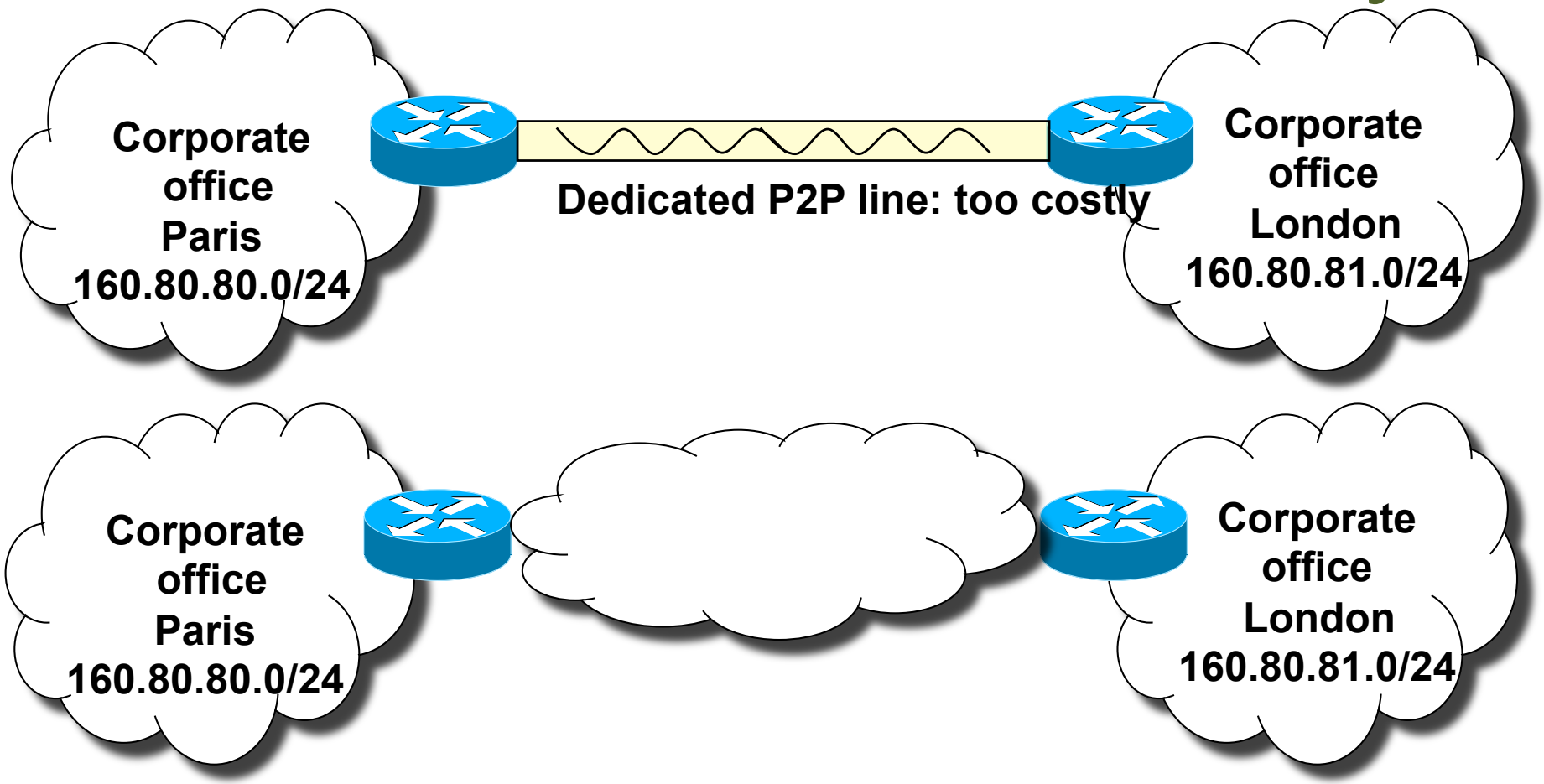
Virtual Private Networks

- A virtual private network (VPN) is a private network that interconnects remote (and often geographically separate) networks through primarily public communication infrastructures such as the Internet
- VPNs provide security through tunneling protocols and security procedures such as encryption
  - Tunneling provides also reachability of private subnetworks
- There are two main types of VPN:
  - remote-access VPNs allow individual users to connect to a remote network such as roaming salespeople connecting to their company's intranet
  - Site-to-site VPNsallow inter-connection of networks of multiple users for example, branch offices to the main company network

Virtual Private Networks

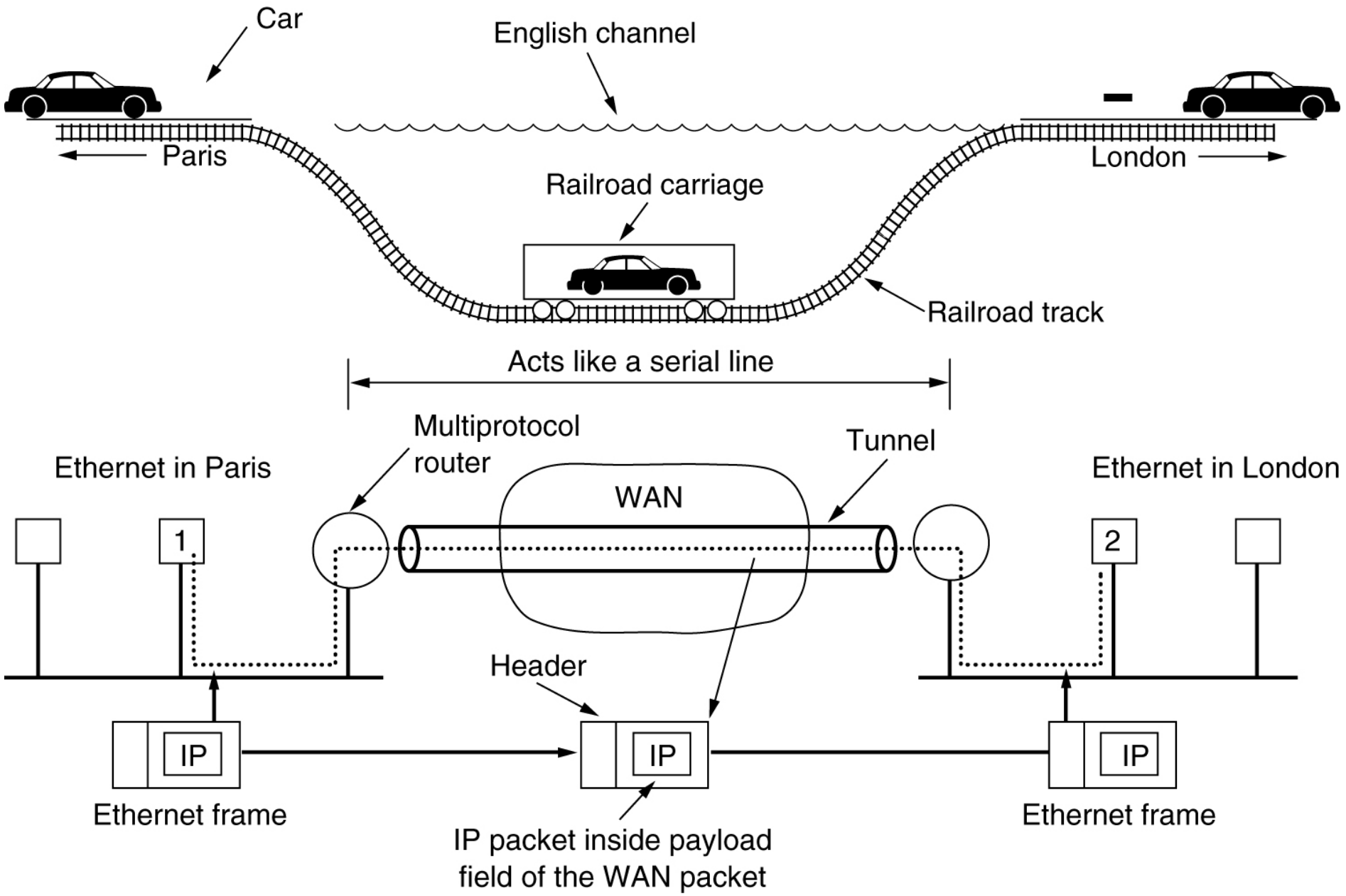


Virtual Private Networks: why?

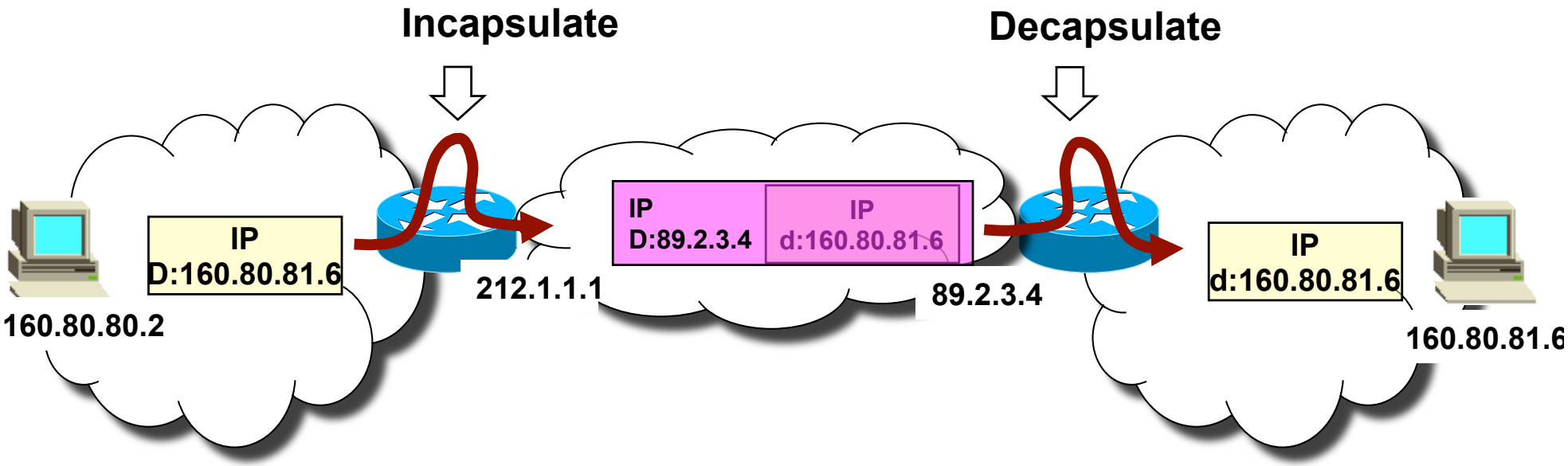


Public Internet or operator IP network:
Emerging issues:
- **How to manage private address space across distributed sites?**
- **How to protect data in transit (especially if public Internet)?**

Virtual Networks → tunnels

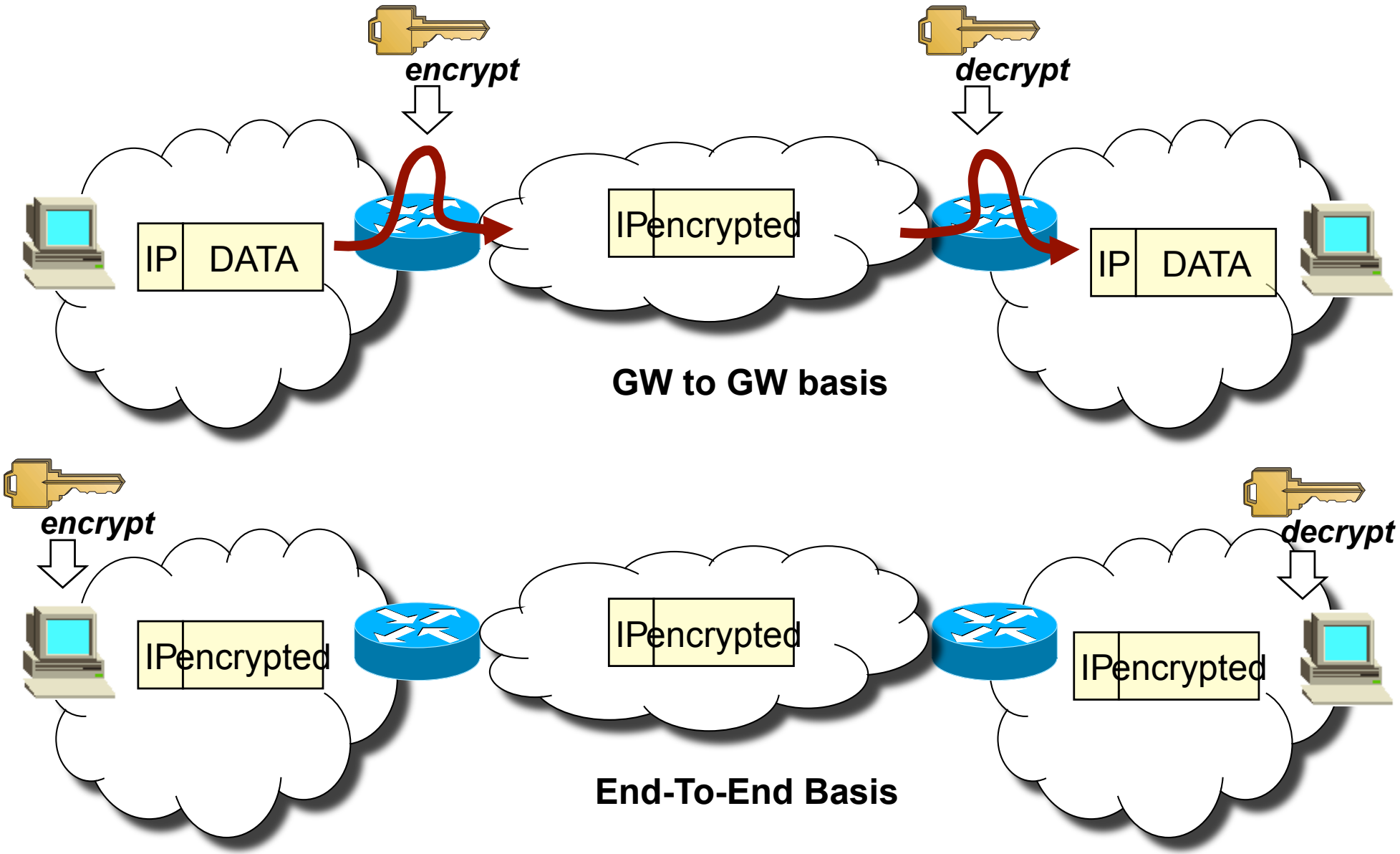


Virtual Networks over IP



- IP in IP tunnels
  - Not the most effective approach!
- MPLS tunnels by far more performance effective
  - Typical VPN offer from today operators
    - MPLS tunnels alone = VPN without the “P” ☺
    - However customer may trust operator (the only one with “hands on” the net)

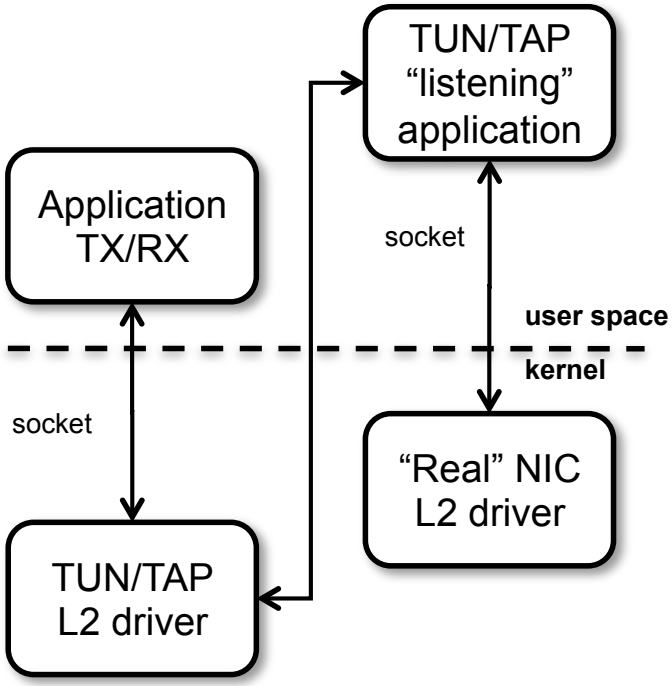
Private Networks: encryption



OpenVPN

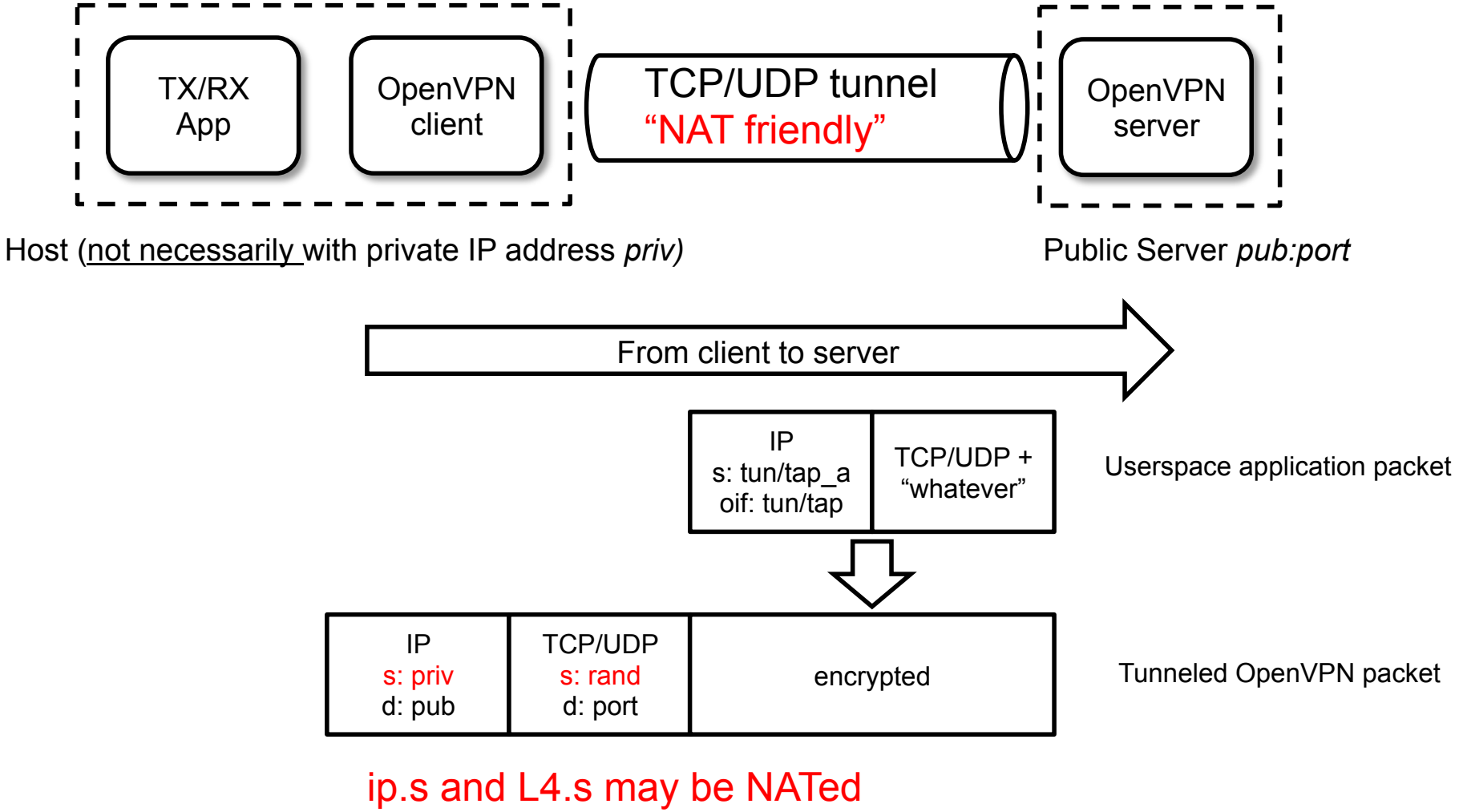
- tunnel any IP subnetwork or virtual ethernet adapter over a single UDP or TCP port
- configure a scalable, load-balanced VPN server farm using one or more machines which can handle thousands of dynamic connections from incoming VPN clients
- use all of the encryption, authentication, and certification features of the OpenSSL library to protect your private network traffic as it transits the internet
- use any cipher, key size, or HMAC digest (for datagram integrity checking) supported by the OpenSSL library
- choose between static-key based conventional encryption or certificate-based public key encryption
- use static, pre-shared keys or TLS-based dynamic key exchange
- use real-time adaptive link compression and traffic-shaping to manage link bandwidth utilization
- tunnel networks whose public endpoints are dynamic such as DHCP or dial-in clients
- tunnel networks through connection-oriented stateful firewalls without having to use explicit firewall rules
- tunnel networks over NAT
- create secure ethernet bridges using virtual tap devices

TUN/TAP drivers



- TUN is a virtual Point-to-Point network device for IP tunneling
- TAP is a virtual Ethernet network device for Ethernet tunneling
- Userland application can write {IP|Ethernet} frame to /dev/{tun|tap}X and kernel will receive this frame from {tun|tap}X interface
- In the same time every frame that kernel writes to {tun|tap}X interface can be read by userland application from {tun|tap}X device

OpenVPN architecture



OpenVPN PKI

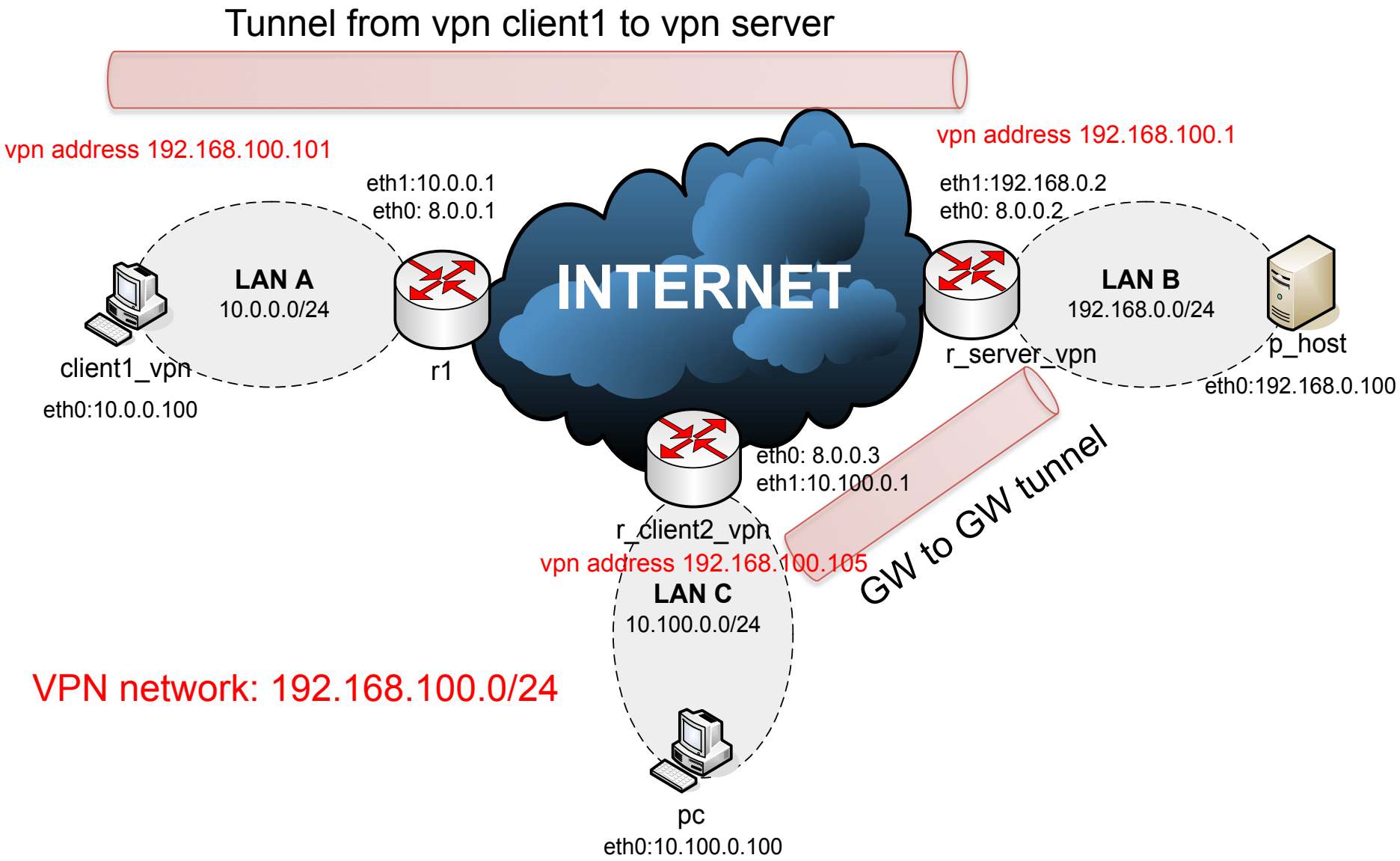
- The first step in building an OpenVPN 2.0 configuration is to establish a PKI which consists of:
  - a separate certificate (also known as a public key) and private key for the server and each client
  - a master Certificate Authority (CA) certificate and key which is used to sign each of the server and client certificates
- OpenVPN supports bidirectional authentication based on certificates, meaning that the client must authenticate the server certificate and the server must authenticate the client certificate before mutual trust is established
- Both server and client will authenticate the other by first verifying that the presented certificate was signed by the master certificate authority (CA), and then by testing information in the now-authenticated certificate header, such as the certificate common name or certificate type (client or server)

This tutorial is based on the following link:
http://openvpn.net/index.php/open-source/documentation/howto.html

OpenVPN security model

- This security model has a number of desirable features from the VPN perspective:
  - The server only needs its own certificate/key -- it doesn't need to know the individual certificates of every client which might possibly connect to it.
  - The server will only accept clients whose certificates were signed by the master CA certificate (which we will generate below). And because the server can perform this signature verification without needing access to the CA private key itself, it is possible for the CA key (the most sensitive key in the entire PKI) to reside on a completely different machine, even one without a network connection.
  - If a private key is compromised, it can be disabled by adding its certificate to a CRL (certificate revocation list). The CRL allows compromised certificates to be selectively rejected without requiring that the entire PKI be rebuilt.
  - The server can enforce client-specific access rights based on embedded certificate fields, such as the Common Name.

Lab10-vpn


Tunnel from vpn client1 to vpn server
vpn address 192.168.100.101
eth1:10.0.0.1
eth0: 8.0.0.1
vpn address 192.168.100.1
eth1:192.168.0.2
eth0: 8.0.0.2
LAN A
10.0.0.0/24
INTERNET
LAN B
192.168.0.0/24
client1_vpn
eth0:10.0.0.100
r1
r_server_vpn
p_host
eth0:192.168.0.100
eth0: 8.0.0.3
eth1:10.100.0.1
r_client2_vpn
GW to GW tunnel
vpn address 192.168.100.105
LAN C
10.100.0.0/24
VPN network: 192.168.100.0/24
pc
eth0:10.100.0.100

Generate the master Certificate Authority (CA) certificate & key

- We'll use set of scripts bundled with OpenVPN
  - In netkit, cd into the directory: `/usr/share/doc/openvpn/example/easy-rsa/2.0`
- Edit the `vars` file on r_server_vpn

Initialize the PKI

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

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
```

Key and certificate files

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

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`

Server configuration

- Important options
  - `port [port_number]`
  - `proto {udp|tcp}`
  - `dev {tun|tap}`
  - `ca [path]`
  - `cert [path]`
  - `key [path]`
  - `server [net_addr] [met_mask]`
  - `client_to_client`
  - `push "route net_addr net_mask"`
  - `route net_addr net_mask`
  - `client-config-dir [path]`
  - `tls-auth [path] 0`

client-config-dir

- 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 10.100.0.0 255.255.255.0`

`Allowed /30 pairs`

```
[  1,  2] [  5,  6] [  9, 10] [ 13, 14] [ 17, 18]
[ 21, 22] [ 25, 26] [ 29, 30] [ 33, 34] [ 37, 38]
[ 41, 42] [ 45, 46] [ 49, 50] [ 53, 54] [ 57, 58]
[ 61, 62] [ 65, 66] [ 69, 70] [ 73, 74] [ 77, 78]
[ 81, 82] [ 85, 86] [ 89, 90] [ 93, 94] [ 97, 98]
[101,102] [105,106] [109,110] [113,114] [117,118]
[121,122] [125,126] [129,130] [133,134] [137,138]
[141,142] [145,146] [149,150] [153,154] [157,158]
[161,162] [165,166] [169,170] [173,174] [177,178]
[181,182] [185,186] [189,190] [193,194] [197,198]
[201,202] [205,206] [209,210] [213,214] [217,218]
[221,222] [225,226] [229,230] [233,234] [237,238]
[241,242] [245,246] [249,250] [253,254]
```

Client configuration

- Important options
  - `port [port_number]`
  - `proto {udp|tcp}`
  - `dev {tun|tap}`
  - `remote [server_addr] [port]`
  - `ca [path]`
  - `cert [path]`
  - `key [path]`
  - `tls-auth [path] 1`

r_server_vpn configuration

```
port 1194
proto udp
dev tun
cert /root/server.crt
key /root/server.key
dh /root/dh.pem
server 192.168.100.0 255.255.255.0
ifconfig-pool-persist ipp.txt
push "route 192.168.0.0 255.255.255.0”
route 10.100.0.0 255.255.255.0
client-config-dir /root/ccd
client-to-client
keepalive 10 120
comp-lzo
persist-key
persist-tun
status openvpn-status.log
verb 3
```

```
r_server_vpn~# openvpn server.conf
```

Why 2 route to 10.100.0.0/24?

- Both route and iroute statements are necessary for network 10.100.0.0/24
- “route” controls the routing from the kernel to the OpenVPN server (via the TUN interface)
- “iroute” controls the routing from the OpenVPN server to the remote clients

Clients configuration

r_client2_vpn

```
client
dev tun
proto udp
remote 8.0.0.2 1194
resolv-retry infinite
nobind
persist-key
persist-tun
ca /root/ca.crt
cert /root/client2.crt
key /root/client2.key
ns-cert-type server
comp-lzo
verb 3
```

```
r_client2_vpn~# openvpn client.conf
```

client1_vpn

```
client
dev tun
proto udp
remote 8.0.0.2 1194
resolv-retry infinite
nobind
persist-key
persist-tun
ca /root/ca.crt
cert /root/client1.crt
key /root/client1.key
ns-cert-type server
comp-lzo
verb 3
```

```
client1_vpn~# openvpn client.conf
```