

# IP Tunneling Generic Routing Encapsulation

David Davis

GRE (Generic Routing Encapsulation) or IP tunneling (IP encapsulation) is a technique that encapsulates IP datagrams within IP datagrams. GRE is a technique that allows datagrams to be encapsulated into IP packets and then redirected to an intermediate host. At this intermediate destination, the datagrams are decapsulated and then routed to the next leg. In doing so, the trip to the intermediate host appears to the inner datagrams as one hop. The general outline of GRE can be found in RFC 1701.

In the current stack, GRE is performed via GRE pseudo interfaces which simulate point-to-point connections. GRE interfaces are created with the `ifconfig` utility:

```
# ifconfig gre0 create
```

Each GRE interface supports 2 modes of operation:

- **GRE encapsulation (default)**  
Outgoing datagrams are encapsulated by an IP header of protocol type 47, and a GRE header specifying the type of the encapsulated datagram (currently only IP). This mode is described in RFC 1702. It's also the default mode on Cisco routers.
- **MOBILE encapsulation (IP protocol number 55)**  
Applicable for IP encapsulation only, outgoing IP datagrams are encapsulated by a smaller header and the original IP header is modified. This mode is described in RFC 2004.

## How do I use it?

Let's illustrate by use of an example:

Suppose you have the following two machines at your disposal on the ubiquitous LAN:

- machine A with IP address 10.0.0.25
- machine B with IP address 10.0.0.163

We'll assume that they're connected by a "real" interface, that they can communicate through generic methods. and that the stacks were started using the `ngre` option. Here's how you create a tunnel between "pseudo" addresses 12.0.0.1 and 12.0.0.2:

On machine A (10.0.0.25), enter:

```
# ifconfig gre0 12.1 12.2 link1      <- set the local (inner) addr
# ifconfig gre0 tunnel 10.25 10.163  <- set the encapsulating (outer) addr
# ifconfig gre0                    <- print the current state
```

The IP packet that's sent from machine A looks like this:

```
+-----+-----+-----+
| dst 10.163 | dst 12.2 | data |
+-----+-----+-----+
\_IP address_\_____data_____/
```

# IP Tunneling Generic Routing Encapsulation

David Davis

On machine B (10.0.0.163), do the reverse:

```
# ifconfig gre0 12.2 12.1 link1
# ifconfig gre0 tunnel 10.163 10.25
# ifconfig gre0
```

Since the IP packet is decapsulated on machine B, it now looks like this:

```
+-----+-----+
| dst 12.2 | data          |
+-----+-----+
\_IP address\_/\_____data_____/
```

From machine A, you should now be able to ping 12.0.0.2. What happens under the covers is an IP packet with source 12.0.0.1 and destination 12.0.0.2 is encapsulated with the "real" source 10.0.0.25 and destination 10.0.0.163.

That's interesting but not very useful since you would have accomplished the same thing if you went to 10.0.0.163 directly. Let's take a look at a more typical example.

## A More Typical Example

Let's say you usually work at the office on machine A that has an IP address 10.0.0.25 (we'll use the same IP addresses from the above example). You use the corporate LAN and you have access to all the machines on your subnet. You've just been sent on a business trip and you'll only be able to access the corporate gateway using the Internet. Let's call the gateway machine C, and assume it has forwarding enabled between the following two "real" interfaces:

- 10.0.0.1 -- attached to the corporate subnet.
- C.C.C.C -- some address on the Internet.

From your remote location, you'll set up the tunnel in similar way to the example above. Assuming that A.A.A.A is machine A's Internet address:

```
# ifconfig gre0 10.25 10.1 link1
# ifconfig gre0 tunnel A.A.A.A C.C.C.C
```

Then set up a route for all of network 10 to the other end of the tunnel:

```
# route add -net 10 10.1
```

Now on to machine C. Either before you leave for your trip, or from your remote location, call your colleague in the IS department and get set up with a proxy arp entry on the gateway. When this is done, the machines behind the gateway will think that you're still at the office. Assuming interface en0 is attached to the local subnet (network 10), they'd enter the following:

```
# arp -s 10.0.0.25 $(netstat -in | grep en0 | grep Link | cut -c 27-43) pub
```

# IP Tunneling Generic Routing Encapsulation

David Davis

The command in the brackets cuts out machine C's en0 MAC address and passes it on to the arp command. These commands set up the tunnel:

```
# ifconfig gre0 10.1 10.25 link1
# ifconfig gre0 tunnel C.C.C.C A.A.A.A
```

You should now have transparent remote access to your corporate LAN -- just as if you're sitting at your desk!

## Final Tidbits

What's that link1 stuff in the ifconfig command for?

It's a flag to the stack to suppress the creation of a nonspecific route that attempts to prevent loops when the inner destination address and real (tunnel) destination address are the same. The algorithm currently is to toggle the last bit in the tunnel destination.

This currently also applies to ipip tunneling, but there's no way to disable it as the inner and tunnel addresses are always the same for an ipip interface (ipip doesn't support the tunnel keyword to ifconfig).

For example, using an ipip interface to create a tunnel to from 10.1 to 10.2:

```
# ifconfig en0 10.1
# route add default 10.3
# ifconfig ipip0 11.1 11.2
```

The last command tries to create an implicit route to 11.3 (last bit of 11.2 is toggled) that resolves the default to 10.3, not the desired 10.2. The following shows how to reach the desired 10.2:

```
# ifconfig en0 10.1
# route add default 10.3
# route add -net 11 10.2
# ifconfig ipip0 11.1 11.2
```

The implicit route to 11.3 now resolves to the route to network 11 (gateway 10.2) instead of the default route.

If GRE encapsulation is the default, how do I enable MOBILE encapsulation?

Use the link2 flag to ifconfig. For example:

```
# ifconfig gre0 10.1 10.25 link1 link2
```