Part of your issue is a lack of a true ipencap tunnel, and a misconfig with the encap.txt file. Keep in mind, with ip-encapsulation you're building point to multipoint IP routes on the internet tunnelled through your ISP. This is why things such as port blocking gets mooted, because the path tunnels right through the ISP and they never see any port to filter... so if they block port 80 (web server services) and you wish to run a web server, ipencap allows this to occur thus bypassing your ISP's filters.

To test if ipencap is working a simple traceroute and route lookup will suffice. Here I did a traceroute to myself from Jerry's box:

```
w9bbs:~# traceroute n1uro.ampr.org
traceroute to n1uro.ampr.org (44.88.0.9), 30 hops max, 60 byte packets
 1  gw.ct.ampr.org (44.88.0.1)  70.118 ms  74.338 ms  74.309 ms
 2  n1uro.ampr.org (44.88.0.9)  81.957 ms  82.369 ms  85.186 ms
```

and you see the first "hop" is actually a tunnelled hop to my Pi (gw.ct.ampr.org) which is what faces the global amprnet.

Thusly a route lookup shows a direct "tunl0" ipencap route from Jerry to me as such:

```
w9bbs:~# ip ro get 44.88.0.9
44.88.0.9 via 76.23.214.38 dev tunl0
```

Now let's try Demetre in Greece:

```
w9bbs:~# traceroute sv1uy.ampr.org
traceroute to sv1uy.ampr.org (44.154.0.1), 30 hops max, 60 byte packets
 1  sv1uy.ampr.org (44.154.0.1)  225.618 ms  228.335 ms  229.175 ms
```

Notice: 1 hop.

With that said, the same should be true for Mike and Eddie however, Mike first:

```
w9bbs:~# traceroute n9pmo.ampr.org
traceroute to n9pmo.ampr.org (44.92.35.11), 30 hops max, 60 byte packets
 1  * * *
 2  * * *
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * * *
10  * * *
11  * * *
12  * * *
13  * * *
14  * * *
15  * * *
16  * * *
17  * * *
18  * * *
19  * * *
20  * * *
21  * * *
22  * * *
23  * * *
24  * * *
25  * * *
26  * * *
27  * * *
28  * * *
29  * * *
30  * * *
```

So the trace fails. Now lets look up the route:

```
w9bbs:~# ip ro get 44.92.35.11
44.92.35.11 via 121.99.232.227 dev tunl0
```

So there is a commercial tunneled route. Let's see where it is as I don't believe 121-net is part of the North America allocations:

```
w9bbs:~# host 121.99.232.227
227.232.99.121.in-addr.arpa domain name pointer olson.net.nz.
```

So it appears, Eddie has ownership of Mike's IP/Block! Not ideal when you should not have to leave the country to hit Mike! Needless to say, there's no ip-encapsulation here on this route.

Now let's look at Eddie's IP:

```
w9bbs:~# ip ro get 44.147.38.42
44.147.38.42 via 121.99.232.227 dev tunl0
```

We already established 121-net goes out to new zealand, so this (as far as amprnet rip/encap.txt is concerned) looks to be satisfied... now let's try a trace. Keep in mind being point to point it should only be 1 or 2 hops:

```
traceroute 44.147.38.42
traceroute to 44.147.38.42 (44.147.38.42), 30 hops max, 60 byte packets
 1  * * *
 2  * * *
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * * *
10  * * *
11  * * *
12  * * *
13  * * *
14  * * *
15  * * *
16  * * *
17  * * *
18  * * *
19  * * *
20  * * *
21  * * *
22  * * *
23  * * *
24  * * *
25  * * *
26  * * *
27  * * *
28  * * *
29  * * *
30  * * *
```

Again, a failure. Now this tells me one or both of the following is true:

1) windows (which is incapable of ip-encapsulation) is in use within the chain.

2) Eddie isn't running a true ip-encapsulation tunnel, thus this is also prohibiting the global amprnet from connectivity.

Yesterday in nos-bbs I posted a very simple script to start up and take down an amprnet ip-encap tunnel for linux, complete with rip routing. As one of the coders on MFNOS in it's final days with K2MF (MF, Maiko (current jnos maintainer), and myself share code and ideas). I still run MFNOS as I demo it at friends' locations to try and get them interested in packet... and ham radio. I ip-encap tunnel from the internet to my linux kernel, and then my tun0 interfaces to xNOS.

A very simple solution may be to set up a PI with Wheezy and have it act as an amprnet encapsulation router, or migrate any existing services from a Windows environment to a Linux environment. Linux has ip-encapsulatioin as part of it's kernel these days. Years ago you had to compile everything from scratch! (the good ol' days)

Keep in mind also, your router must have a DMZ set to your main amprnet gateway so that ip-encapsulation will pass. This will have zero effect on any commercial IP port forwarding statements you may have to other services. (another nice feature of ip-encap!)

Now... are you "stuck" in a tunnel? Somewhat... However, frames from the global internet to you still require to come into your router encapsulated! This is where UCSD.EDU's 44.0.0.1 comes into play. Frames from the global internet come into 44.0.0.1's commercial IP which is 169.228.34.84 and it sends the frames to you encapsulated as 44.0.0.1 so it's at that point the frames are encapsulated. This is also automatically announced via BGP and/or OSPF by UCSD to the major internet backhaul.

You can also pass amprnet point to point encapsulated under AX25 just as you do NetRom... or you can encapsulate IP under both ax25 AND netrom -not ideal- because ax25 has a frame size limitation of 256 bytes, and each protocol header takes up more bytes from the data portion of the frames. This is how BPQ handles IP routing, and thus it does so using an ARP table to tell itself which ax25 callsign-ssid to route the IP frames through. If this wasn't true BPQ would have 2 things:

1) an ip route table to manage (not BPQARP).
2) an ip protocol stack of it's own.

Now that I've clued you in as to how ip-encapsulation works in somewhat layman's terms let me show you how it works via an ascii diagram:

```
station 1<------>internet<------>station 2
tunnel<---------------------------->tunnel

station 1<------>internet<------>station 3
tunnel<---------------------------->tunnel
```

and so on. These routes are populated via rip and/or encap.txt By having point to multipoint routes, traffic is sent direct from station 1 to station 2 or station 1 to station 3 bypassing UCSD and keeping it's overhead down. Note: traffic and routing is bidirectional.

Now, from the global internet to your amprnet ip-encap tunnel and back looks like this:

```
commercial user<----->ucsd.edu<------->station 1
                        tunnel<------->tunnel
```

Using this method, the size of the frames are 1480 bytes, vs 192 bytes using ax25 to encapsulate IP, thus making data throughput MUCH faster :)

I hope this clears things up on how the tunnel routing works.
You're probably pinging Eddie via an ax25 arp