VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

ABHIJITH M – 17BCE1052

AKASH MENON – 17BCE1301

SAI ADITYA – 17BCE1310

SANATHAN NARAYANAN – 17BCE1094

SHYAM SUNDAR – 17BCE1309

SHYAM SURESH – 17BCE1303

S SRIHARI – 17BCE1308

SRINATH – 17BCE1217

Dr. Reshmi T R

Slot: F1 + TF1

CSE1004– Network and Communication

DIGITAL ASSIGNMENT – 3

6 to 4

WHAT IS 6 to 4?

6to4 is an Internet transition mechanism for migrating from Internet Protocol version 4 (IPv4) to version 6 (IPv6), a system that allows IPv6 packets to be transmitted over an IPv4 network (generally the IPv4 Internet) without the need to configure explicit tunnels. Special relay servers are also in place that allow 6to4 networks to communicate with native IPv6 networks.

6to4 is especially relevant during the initial phases of deployment to full, native IPv6 connectivity, since IPv6 is not required on nodes between the host and the destination. However, it is intended only as a transition mechanism and is not meant to be used permanently.

6to4 may be used by an individual host, or by a local IPv6 network. When used by a host, it must have a global IPv4 address connected, and the host is responsible for encapsulation of outgoing IPv6 packets and decapsulation of incoming 6to4 packets. If the host is configured to forward packets for other clients, often a local network, it is then a router.

Most IPv6 networks use autoconfiguration, which requires the last 64 bits of the address for the host. The first 64 bits are the IPv6 prefix. The first 16 bits of the prefix are always 2002:, the next 32 bits are the IPv4 address, and the last 16 bits of the prefix are available for addressing multiple IPv6 subnets behind the same 6to4 router. Since the IPv6 hosts using autoconfiguration already have determined the unique 64 bit host portion of their address, they must simply wait for a Router Advertisement indicating the first 64 bits of prefix to have a complete IPv6 address. A 6to4 router will know to send an encapsulated packet directly over IPv4 if the first 16 bits are 2002, using the next 32 as the destination, or otherwise send the packet to a well-known relay server, which has access to native IPv6.

6to4 does not facilitate interoperation between IPv4-only hosts and IPv6-only hosts. 6to4 is simply a transparent mechanism used as a transport layer between IPv6 nodes.

Due to the high levels of misconfigured hosts and poor performance observed, an advisory about how 6to4 should be deployed was published in August 2011.[2] Due to unsolvable operational problems using the 6to4 anycast prefix, that part of the standard was deprecated in 2015.

SECURITY CONSIDERATIONS:

According to RFC 3964, 6to4 routers and relays should ensure that:

- either or both the source and destination addresses of any encapsulated packet is within the 6to4 IPv6 prefix 2002::/16,
- if the source IPv6 address is a 6to4 IPv6 address, its corresponding 6to4 router IPv4 address matches the IPv4 source address in the IPv4 encapsulation header,
- similarly, if the destination IPv6 address is a 6to4 IPv6 address, its corresponding 6to4 router IPv4 address matches the IPv4 destination address in the IPv4 encapsulation header,
- any embedded 6to4 router IPv4 address is global unicast.

NETWORK INGRESS FILTERING:

Network ingress filtering is a packet filtering technique used by many Internet service providers to try to prevent source address spoofing of Internet traffic, and thus indirectly combat various types of net abuse by making Internet traffic traceable to its source.

Network ingress filtering is a "good neighbour" policy which relies on cooperation between ISPs for their mutual benefit.

The best current practices for network ingress filtering are documented by the Internet Engineering Task Force in **BCP 38** and **BCP 84**, which are defined by RFCs 2827 and 3704, respectively.

BCP 84 recommends that upstream providers of IP connectivity filter packets entering their networks from downstream customers, and discard any packets which have a source address which is not allocated to that customer.

There are many possible ways of implementing this policy; one common mechanism is to enable reverse path forwarding on links to customers, which will indirectly apply this policy based on the provider's route filtering of their customers' route announcements.

PROBLEMS:

1) Networks receive packets from other networks. Normally a packet will contain the IP address of the computer that originally sent it. This allows devices in the receiving network to know where it came from, allowing a reply to be routed back (amongst other things), except when IP addresses are used through a proxy or a spoofed IP address, which does not pinpoint a specific user within that pool of users.
2) A sender IP address can be faked ('spoofed'), characterising a spoofing attack. This disguises the origin of packets sent, for example in a denial-of-service attack. The same holds true for proxies, although in a different manner than "IP spoofing."
3) **Spoofing in IPv4 with 6 to 4:** In this type of attack 6 to 4 tunneling spoofed traffic can be injected from IPv4 into IPv6.The IPv4 spoofed address acts like an IPv4 source, 6 to 4 relay any cast (192.88.99.1) acts like an IPv4 destination. The 2002::spoofed source address acts like an IPv4 destination.
4) **Theft of Service:** During the IPv6 transition period, many sites will use IPv6 tunnels over IPv4 infrastructure. Sometimes we will use static or automatic tunnels. The 6 to 4 relay administrators will often want to use some policy limit the use of the relay to specific 6 to 4 sites or specific IPv6 sites.However some users may be able to use the service regardless of these controls by configuring the address of the relay using its IPv4 address instead of 192.88.99.1 or using the router header to route the IPv6 packets to reach specific 6 to 4 relays.
5) **Attack with IPv4 broadcast address:** In the 6 to 4 mechanisms, some packets with the destination addresses spoofed and mapped to their broadcast addresses of the 6to4 or relay routers are sent to the target routers by the attackers in the IPv6 network. In this case also 6 to 4 or relay routers are attacked by the broadcast addresses.

POTENTIAL SOLUTIONS:

One potential solution involves implementing the use of intermediate Internet gateways (i.e., those servers connecting disparate networks along the path followed by any given packet) filtering or denying any packet deemed to be illegitimate. The gateway processing the packet might simply ignore the packet completely, or where possible, it might send a packet back to the sender relaying a message that the illegitimate packet has been denied. Host intrusion prevention systems (HIPS) are one example of technical engineering applications that help to identify, prevent and/or deter unwanted, unsuspected and/or suspicious events and intrusions.

Any router that implements ingress filtering checks the source IP field of IP packets it receives, and drops packets if the packets don't have an IP address in the IP address block, to which the interface is connected. This may not be possible if the end host is multi-homed and sends transit network traffic.

In ingress filtering, packets coming into the network are filtered if the network sending it should not send packets from the originating IP address(es). If the end host is a stub network or host, the router needs to filter all IP packets that have, as the source IP, private addresses (RFC 1918), bogon addresses or addresses that do not have the same network address as the interface.

The security issues in tunneling mechanisms can generally limited by investigating the validness of the source/destination address at each tunnel end point. Usually in tunneling techniques it is easier to avoid ingress filtering checks. Sometimes it is possible to send packets having linklocal addresses and hop-limit=255, which can be used to attack subnet hosts from the remote node, but it is very difficult to deal with attacks with legal IP addresses now. Since the tunnel end points of configuration tunnels are fixed, so IPSec can be used to avoid spoofed attacks. IPv6 Security issues even though it has provided lot off features but its known that automatic tunneling are dangerous as other end points are unspecified because it's very difficult to prevent automatic tunneling mechanisms DoS/reflect-DoS attacks by the attackers in IPv4 network.

SOME MORE SECURITY FLAWS & HOW THEY CAN BE CORRECTED:

Unless properly managed, tunneling mechanisms like 6 to 4 might result in negative security implications. For example, they might increase host exposure, might be leveraged to evade security controls, might contain protocol-based vulnerabilities, and/or the corresponding code might contain bugs with security implications.

6to4 and other tunneling mechanisms should be a concern not only to network administrators that have consciously deployed them, but also to those who have not deployed them, as these mechanisms might be leveraged to bypass their security policies.

The aforementioned issues could be mitigated by applying the common security practice of only allowing traffic deemed as "necessary" (i.e., the so-called "default deny" policy). Thus, when such policy is enforced, IPv6 transition/coexistence traffic would be blocked by default and would only be allowed as a result of an explicit decision.

In those scenarios in which transition/coexistence traffic is meant to be blocked, it is highly recommended that, in addition to the enforcement of filtering policies at the organizational perimeter, the corresponding transition/coexistence mechanisms be disabled on each node connected to the organizational network. This would not only prevent security breaches resulting from accidental use of these mechanisms, but would also disable this functionality altogether, possibly mitigating vulnerabilities that might be present in the host implementation of these transition/coexistence mechanisms.

IPv6-in-IPv4 tunneling mechanisms (such as 6to4 or configured tunnels) can generally be blocked by dropping IPv4 packets that contain a Protocol field set to 41. Security devices such as NIDS might also include signatures that detect such transition/coexistence traffic.

Administrators considering the filtering of transition/coexistence traffic are urged to pay attention to the general considerations for IPv6 traffic filtering discussed in Section 4

6to4 is an address assignment and router-to-router, host-to-router, and router-to-host automatic tunneling mechanism that is meant to provide IPv6 connectivity between IPv6 sites and hosts across the IPv4 Internet.

All IPv6-in-IPv4 traffic, including 6to4,could be easily blocked by filtering IPv4 packets that contain their Protocol field set to 41. This is the most effective way of filtering such traffic.

If 6to4 traffic is meant to be filtered while other IPv6-in-IPv4 traffic is allowed, then more finer-grained filtering rules could be applied. For example, 6to4 traffic could be filtered by applying filtering rules such as:

o Filter outgoing IPv4 packets that have the Destination Address set to an address that belongs to the prefix 192.88.99.0/24.

o Filter incoming IPv4 packets that have the Source Address set to an address that belongs to the prefix 192.88.99.0/24.

NOTE: These rules assume that the corresponding nodes employ
the "Anycast Prefix for 6to4 Relay Routers".It has
been suggested that 6to4 relays send their packets with their
IPv4 Source Address set to 192.88.99.1.

o Filter outgoing IPv4 packets that have the Destination Address set to the IPv4 address of well-known 6to4 relays.

o Filter incoming IPv4 packets that have the Source Address set to the IPv4 address of well-known 6to4 relays.

These last two filtering policies will generally be unnecessary, and possibly infeasible to enforce (given the number of potential 6to4 relays, and the fact that many relays might remain unknown to the network administrator). If anything, they should be applied with the additional requirement that such IPv4 packets have their Protocol field set to 41 to avoid the case where other services available at the same IPv4 address as a 6to4 relay are mistakenly made inaccessible.

If the filtering device has capabilities to inspect the payload of IPv4 packets, then the following filtering rules could be enforced:

o Filter outgoing IPv4 packets that have their Protocol field set to 41, and that have an IPv6 Source Address (embedded in the IPv4 payload) that belongs to the prefix 2002::/16.

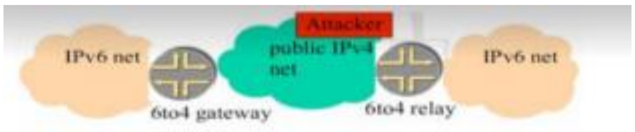
o Filter incoming IPv4 packets that have their Protocol field set to 41, and that have an IPv6 Destination address (embedded in the IPv4 payload) that belongs to the prefix 2002::/16.

Few Types of DOS attacks which can be done:

Potential reflect- DoS attack on Destination Host The hackers in the IPv4 networks can make a reflect–DoS attack to a normal IPv6 network (node) through the 6-to-4 router (tunnel) end point by sending the encapsulated packets with the spoofed IPv6 source address as the specific IPv6 node.

Cheat by a Hacker with the IPv6 Neighbour Discovery (ND) message: Whenever IPv4 network is treated as the link layer in tunneling technology, the hackers in the IPv4 networks can cheat and DoS attack the tunnel end point by sending encapsulated IPv6 neighbour discovery (ND)messages with a spoofed IPv6 link local address. The automatic tunneling techniques like 6-to 4 and Teredo get the information of remote tunnel end point from the certain IPv6 packets.

Distributed Reflection DoS: This type of attack can be performed if the very large number of nodes whenever involved in the sending spoofed traffic with same source IPv6 addresses.



Spoofing using tunnelling

GENERAL SPOOFING DEMONSTRATION:

- Ping a destination with packets
- Log on to another IP using telnet
- Using hping3, send packets to the destination using the logged on IP
- One can clearly view through Wireshark the scenario before and after spoofing
- This spoofing can be blocked dropping the respective packets

Idea to filter out the nodes to only which valid messages are sent:

Consider a chat application between a node(server here) and other nodes(valid clients).The nodes with which the server node wants to have communication with is given with unique password and the client side program to transmit messaegs to them.Encrypted messages are sent from server side to client side after valid password has been entered.A unique decryption mechanism is embedded in the client side program which were distributed to the clients which allows them to get the original message which was sent by the server.

Consider suppose if a new client tries to act as a old one[spoofing node] and tries to communicate with the server.The new client would not be able to get the message from server as unique password is not provided to it.Even if the new client cracks the password he would not be able to get the correct message as the unique client side program contaning decryption mechanism is not sent to him.Hence a two level security mechanism is provided through this mechanism

The same method can be applied to the given problem where the unique password and client side program are brodcasted to the valid clients only[clients except ones using 6to4 mechanism] hence disallowing the 6to4 using nodes from getting packets/messages from the main server node[or the node from which message is requested]

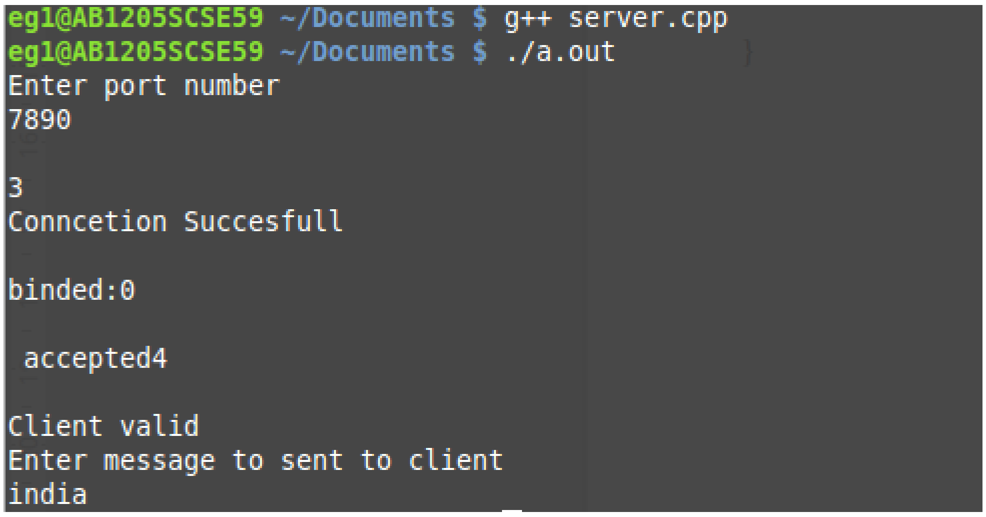
The mechanism is illustrated below.

Server Side program

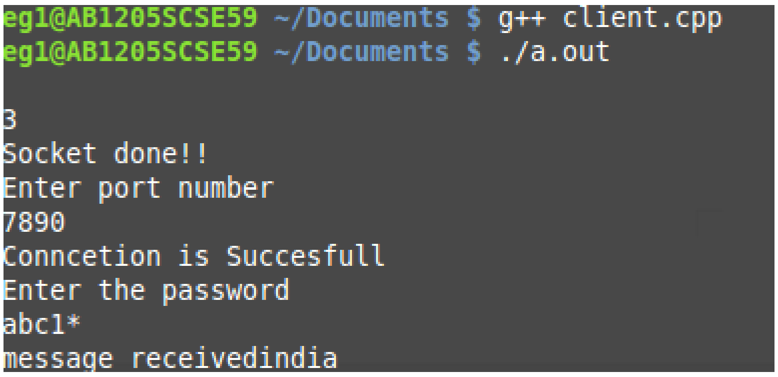
```
#include<netinet/in.h>
#include<iostream>
#include<string.h>
#include<stdlib.h>
#include<sys/types.h>
using namespace std;

main()
{
            int i;
        int portno;
            char msg[20],buff[30];
            struct sockaddr_in servaddr,cliaddr;
            cout<<"Enter port number\n";
            cin>>portno;
            int socketfd=socket(AF_INET,SOCK_STREAM,0);
    cout<<"\n"<<socketfd<<endl;
    if(socketfd>0)
            cout<<"Conncetion Succesfull"<<endl;
    servaddr.sin_family=AF_INET;
    servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
    servaddr.sin_port=htons(portno);

    int b=bind(socketfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
            if(b>=0)
                        cout<<"\nbinded:"<<b<<endl;


    int l=listen(socketfd,10);
    {
            if(l>0)
                        cout<<"listening...."<<endl;
    }

    socklen_t clilen=sizeof(cliaddr);

    int nsd=accept(socketfd,(struct sockaddr*) &cliaddr,&clilen);
    if(nsd>=0)
    {
            cout<<"\n accepted"<<nsd<<endl;
    }



//  if(ss>=0)
            //cout<<"message sent"<<endl;
int rs=recv(nsd,buff,15,0);
if(rs>0)
   {
            if(strcmp(buff,"abc1*")==0)
        cout<<"\nClient valid\n";
        else
        {
        cout<<"\nInvalid Client";
        exit(0);
        }
   }
cout<<"Enter message to sent to client"<<endl;
cin>>msg;
    for(i=0;msg[i]!='\0';i++)
 {
   if((msg[i])-96>=15)
     {
      int l=(msg[i])-96+12-26+64;
      msg[i]=(l);
     }
   else
     {
      int l=(msg[i])-96+12+64;
      msg[i]=(l);
     }
 }
int ss=send(nsd,msg,sizeof(msg),0);
}
```

Valid Client side program

```
#include<iostream>
#include<netinet/in.h>
#include<string.h>
using namespace std;
 int main()
{
            int i;
        char msg[20],buff[25];
            struct sockaddr_in servaddr,cliaddr;

            int socketfd=socket(AF_INET,SOCK_STREAM,0);
    cout<<"\n"<<socketfd<<endl;
    if(socketfd>0)
            cout<<"Socket done!!"<<endl;

int cport;
            cout<<"Enter port number"<<endl;
            cin>>cport;

   servaddr.sin_family=AF_INET;
   servaddr.sin_addr.s_addr=htonl(INADDR_ANY);

   servaddr.sin_port=htons(cport);

   int c=connect(socketfd,(struct sockaddr*) &servaddr,sizeof(servaddr));
   if(c>=0)
   {
            cout<<"Conncetion is Succesfull"<<endl;
   }
  //strcpy(msg,"hello");
   cout<<"Enter the password"<<endl;
   cin>>msg;
 int cs=send(socketfd,msg,15,0);
  int cr=recv(socketfd,buff,sizeof(buff),0);
      for(i=0;buff[i]!='\0';i++)
 {
   if(buff[i]<=76)
     {
       int l=(buff[i])+46;
       buff[i]=(l);
     }
   else
     {
       int l=(buff[i])+20;
       buff[i]=(l);
     }
 }

  if(cr>0)
  {
            cout<<"message received"<<buff<<endl;
  }

}
```



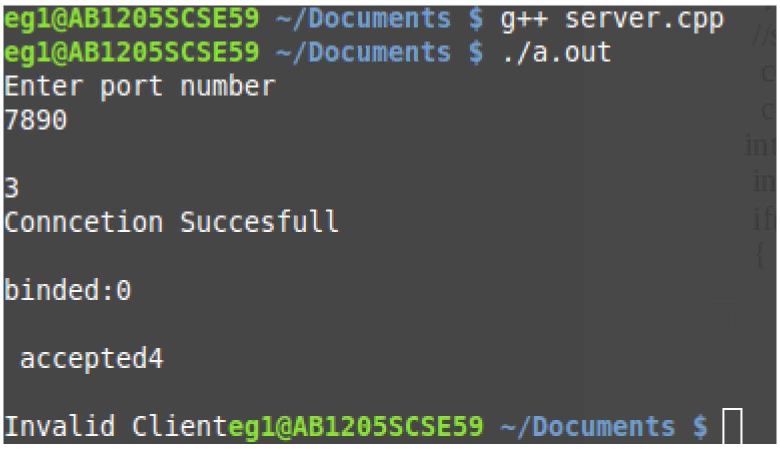
```
eg1@AB1205SCSE59 ~/Documents $ g++ server.cpp
eg1@AB1205SCSE59 ~/Documents $ ./a.out
Enter port number
7890

3
Conncetion Succesfull

binded:0

 accepted4

Client valid
Enter message to sent to client
india
```



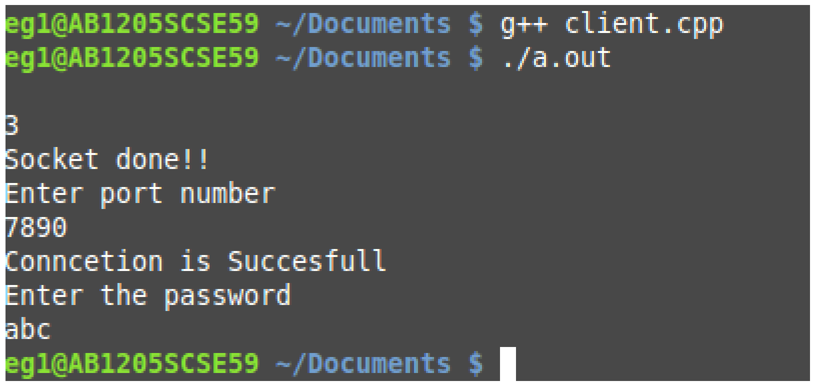
```
eg1@AB1205SCSE59 ~/Documents $ g++ client.cpp
eg1@AB1205SCSE59 ~/Documents $ ./a.out

3
Socket done!!
Enter port number
7890
Conncetion is Succesfull
Enter the password
abc1*
message receivedindia
```

Invalid client code

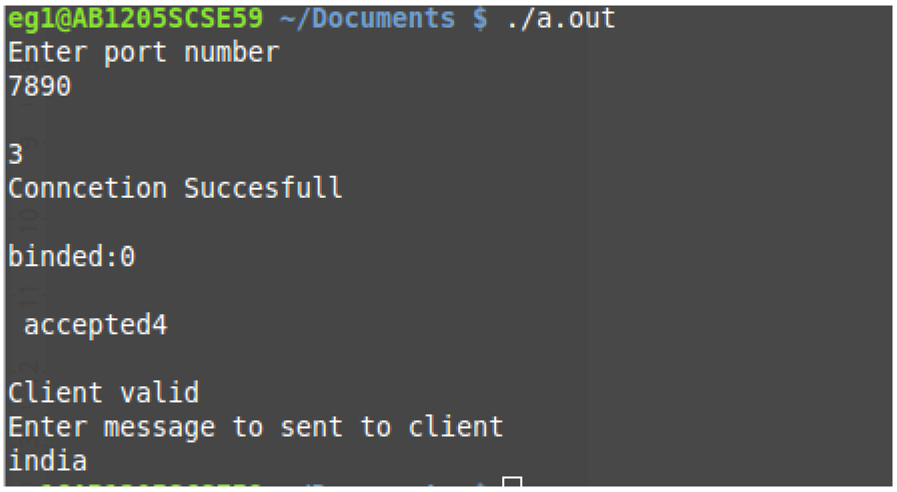
```
#include<iostream>
#include<netinet/in.h>
#include<string.h>
using namespace std;
 int main()
{
            int i;
        char msg[20],buff[25];
            struct sockaddr_in servaddr,cliaddr;

            int socketfd=socket(AF_INET,SOCK_STREAM,0);
    cout<<"\n"<<socketfd<<endl;
    if(socketfd>0)
            cout<<"Socket done!!"<<endl;
int cport;
            cout<<"Enter port number"<<endl;
            cin>>cport;

   servaddr.sin_family=AF_INET;
   servaddr.sin_addr.s_addr=htonl(INADDR_ANY);

   servaddr.sin_port=htons(cport);

   int c=connect(socketfd,(struct sockaddr*) &servaddr,sizeof(servaddr));
   if(c>=0)
   {
            cout<<"Conncetion is Succesfull"<<endl;
   }
  //strcpy(msg,"hello");
   cout<<"Enter the password"<<endl;
   cin>>msg;
 int cs=send(socketfd,msg,15,0);
  int cr=recv(socketfd,buff,sizeof(buff),0);
  if(cr>0)
  {
            cout<<"message received"<<buff<<endl;
  }

}
```



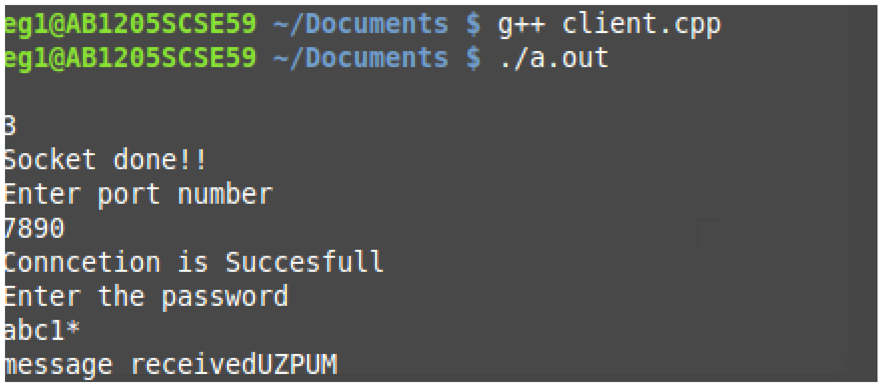
```
eg1@AB1205SCSE59 ~/Documents $ g++ server.cpp
eg1@AB1205SCSE59 ~/Documents $ ./a.out
Enter port number
7890

3
Conncetion Succesfull

binded:0

 accepted4

Invalid Clienteg1@AB1205SCSE59 ~/Documents $
```



```
eg1@AB1205SCSE59 ~/Documents $ g++ client.cpp
eg1@AB1205SCSE59 ~/Documents $ ./a.out

3
Socket done!!
Enter port number
7890
Conncetion is Succesfull
Enter the password
abc
eg1@AB1205SCSE59 ~/Documents $
```



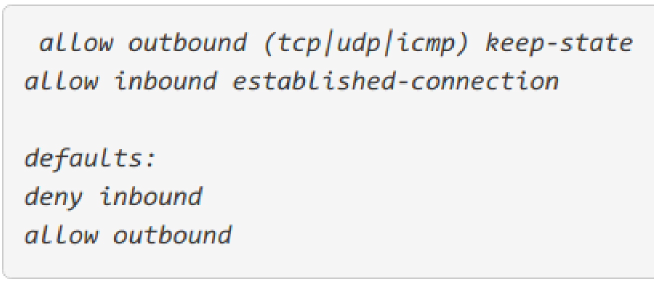
```
eg1@AB1205SCSE59 ~/Documents $ ./a.out
Enter port number
7890

3
Conncetion Succesfull

binded:0

 accepted4

Client valid
Enter message to sent to client
india
```



```
eg1@AB1205SCSE59 ~/Documents $ g++ client.cpp
eg1@AB1205SCSE59 ~/Documents $ ./a.out

3
Socket done!!
Enter port number
7890
Conncetion is Succesfull
Enter the password
abc1*
message receivedUZPUM
```

SOME MORE PROBLEMS FACED BY 6to4 MECHANISM:

1. PROBABILITY OF FAILURE: Probability of failure is really high. When 6to4 fails, it has no timeout before hosts try the IPv4 connection. This results in poor user experience.
2. FIREWALL RULES: Earlier, it was speculated that the very high rates of failing connections were caused due to filtering. But, there existed firewall rules which were responsible for the 6to4 connection failure.



```
 allow outbound (tcp|udp|icmp) keep-state
allow inbound established-connection

defaults:
deny inbound
allow outbound
```

Illustration 1: Firewall rules for 6to4 connection failure.

3. LAYER-3-ONLY FILTERING: A problem with layer-3-only filtering is that the source address of the return packets from a 6to4 connection is not well defined. The 6to4 relay that receives IPv6 packets and encapsulates these in IPv4 packets can be configured to use the anycast 192.88.99.1 address, or can use one of it's own IPv4 unicast addresses for the return traffic. In the latter (unicast) case, stateful layer-3-only filtering will break, so the amount of extra breakage caused by switching a 6to4 relay from anycast to unicast is an indication of how much of this layer-3-only stateful filtering is going on.

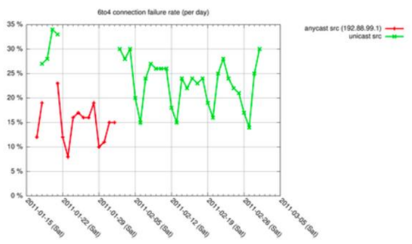


Illustration 2: 6to4 Fail Rate Graph

4. TCP RESETS: TCP-resets seem to be the fastest way that can make TCP connections over 6to4 fail. The prototype code could work as a stand-alone daemon somewhere near the inside interface of a firewall and would currently kill all TCP-over-6to4 traffic it detects.

6to4 TUNNELS:

Manual tunnels are easy to configure but they do not scale well when a large number of tunnels is necessary. IETF has defined a mechanism called 6to4 to automatically connect to multiple IPv6 networks over one configured tunnel. 6to4 tunnels are defined in RFC 3056 [10], connection of IPv6 Domains via IPv4 Clouds. This 6to4 is point-to-multipoint connection. The difference between 6to4 and a manual tunnel is that the manual tunnel has to configure statically the other end of the tunnel, the tunnel IPv4 destination address while the IPv4 destination address is automatically derived from the IPv6 destination of the packet.

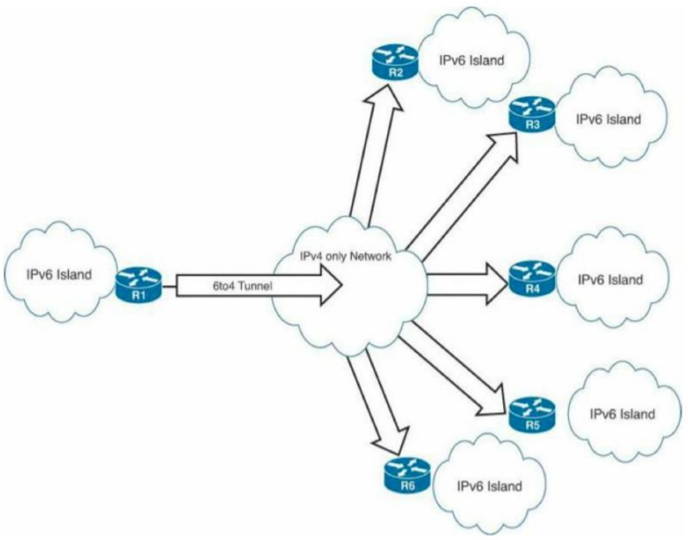
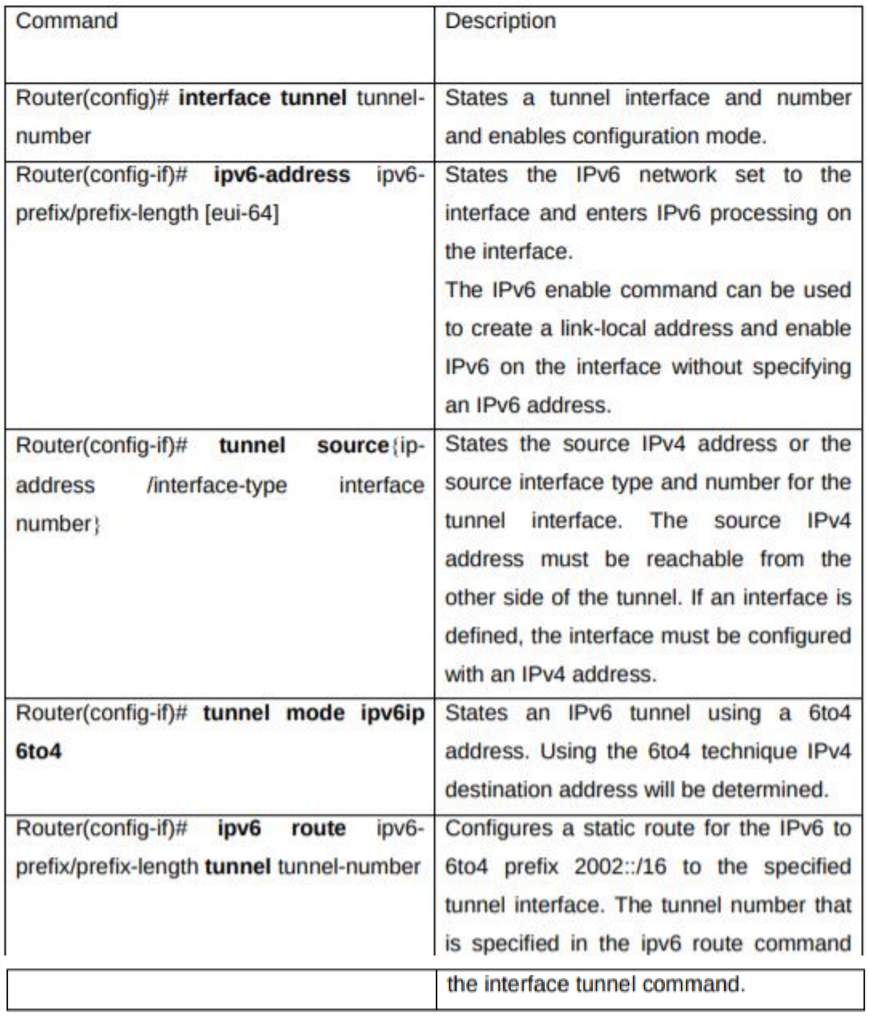


Illustration 3: 6to4 Tunnel

CONFIGURATION COMMANDS FOR 6to4 TUNNEL:



Command	Description
Router(config)# **interface tunnel** tunnel-number	States a tunnel interface and number and enables configuration mode.
Router(config-if)# **ipv6-address** ipv6-prefix/prefix-length [eui-64]	States the IPv6 network set to the interface and enters IPv6 processing on the interface. The IPv6 enable command can be used to create a link-local address and enable IPv6 on the interface without specifying an IPv6 address.
Router(config-if)# **tunnel source**{ip-address /interface-type interface number}	States the source IPv4 address or the source interface type and number for the tunnel interface. The source IPv4 address must be reachable from the other side of the tunnel. If an interface is defined, the interface must be configured with an IPv4 address.
Router(config-if)# **tunnel mode ipv6ip 6to4**	States an IPv6 tunnel using a 6to4 address. Using the 6to4 technique IPv4 destination address will be determined.
Router(config-if)# **ipv6 route** ipv6-prefix/prefix-length **tunnel** tunnel-number	Configures a static route for the IPv6 to 6to4 prefix 2002::/16 to the specified tunnel interface. The tunnel number that is specified in the ipv6 route command the interface tunnel command.

Encapsulating IPv6 into IPv4:

The checks described in this section are to be performed when encapsulating IPv6 into IPv4.

The encapsulation rules are mainly designed to keep implementors from "shooting themselves in the foot." For example, the source address check would verify that the packet will be acceptable to the decapsulator, or the sanity checks would ensure that addresses derived from private addresses are not used (which would be equally unacceptable).

```
src_v6 and dst_v6 MUST pass ipv6-sanity checks else drop
if prefix (src_v6) == 2002::/16
```

```
    ipv4 address embedded in src_v6 MUST match src_v4
else if prefix (dst_v6) == 2002::/16
        dst_v4 SHOULD NOT be assigned to the router
else
    drop
        /* we somehow got a native-native ipv6 packet */
fi
accept
```

Decapsulating IPv4 into IPv6:

The checks described in this section are to be performed when decapsulating IPv4 into IPv6. They will be performed in both the 6to4 router and relay.

```
src_v4 and dst_v4 MUST pass ipv4-sanity checks, else drop
src_v6 and dst_v6 MUST pass ipv6-sanity checks, else drop
if prefix (dst_v6) ==
    ipv4 address embedded in dst_v6 MUST match dst_v4
        if prefix (src_v6) == 2002::/16
            ipv4 address embedded in src_v6 MUST match src_v4
            dst_v4 SHOULD be assigned to the router
        fi
elif prefix (src_v6) == 2002::/16
    ipv4 address embedded in src_v6 MUST match src_v4
    dst_v4 SHOULD be assigned to the router
else
    drop
        /* the we somehow got a native-native ipv6 packet */
fi
accept
```

IPv4 and IPv6 Sanity Checks:

The encapsulation and decapsulation checks include certain sanity

checks for both IPv4 and IPv6. These are described here in detail.

IPv4:

IPv4 address MUST be a global unicast address, as required by the

6to4 specification. The disallowed addresses include those defined in, and others widely used and known not to be global. These are

o 0.0.0.0/8 (the system has no address assigned yet)

o 10.0.0.0/8 (private)

o 127.0.0.0/8 (loopback)

o 172.16.0.0/12 (private)

o 192.168.0.0/16 (private)

o 169.254.0.0/16 (IANA Assigned DHCP link-local)

o 224.0.0.0/4 (multicast)

o 240.0.0.0/4 (reserved and broadcast)

In addition, the address MUST NOT be any of the system's broadcast

addresses. This is especially important if the implementation is

made so that it can

o receive and process encapsulated IPv4 packets arriving at it broadcast addresses, or

o send encapsulated IPv4 packets to one of its broadcast addresses.

IPv6:

IPv6 address MUST NOT be

o 0::/16 (compatible, mapped addresses, loopback, unspecified, ...)

o fe80::/10 (link-local)

o fec0::/10 (site-local)

o ff00::/8 (any multicast)

Note: Only link-local multicast would be strictly required, but it is believed that multicast with 6to4 will not be feasible, so it has been disallowed as well.

In addition, it MUST be checked that equivalent 2002:V4ADDR::/48 checks, where V4ADDR is any of the above IPv4 addresses, will not be passed.

Optional Ingress Filtering:

In addition, the implementation in the 6to4 router may perform some form of ingress filtering (e.g., Unicast Reverse Path Forwarding checks). For example, if the 6to4 router has multiple interfaces, of which some are "internal", receiving either IPv4 or IPv6 packets with source address belonging to any of these internal networks from the Internet might be disallowed.

If these checks are implemented and enabled by default, it's recommended that there be a toggle to disable them if needed.

Checks:

The rule "dst_v4 SHOULD be assigned to the router" is not needed if the 6to4 router implementation only accepts and processes encapsulated IPv4 packets arriving to its unicast IPv4 addresses, and when the destination address is known to be a local broadcast address, it does not try to encapsulate and send packets to it.

Some checks, especially the IPv4/IPv6 Sanity Checks, could be at least partially implementable with system-level access lists, if one would like to avoid placing too many restrictions in the 6to4 implementation itself. This depends on how many hooks are in place for the access lists. In practice, it seems that this could not be done effectively enough unless the access list mechanism is able to parse the encapsulated packets.

Issues in 6to4 Implementation and Use:

This section tries to give an overview of some of the problems 6to4 implementations face, and the kind of generic problems the 6to4 users could come up with.

Implementation Considerations with Automatic Tunnels:

There is a problem with multiple transition mechanisms if strict security checks are implemented. This may vary a bit from implementation to implementation.

Consider three mechanisms using automatic tunneling: 6to4, ISATAP, and Automatic Tunneling using Compatible Addresses. All of these use IP-IP (protocol 41) IPv4 encapsulation with, more or less, pseudo-interface.

When a router, which has any two of these enabled, receives an IPv4

encapsulated IPv6 packet

src_v6 = 2001:db8::1

dst_v6 = 2002:1010:1010::2

src_v4 = 10.0.0.1

dst_v4 = 20.20.20.20

What can it do? How should it decide which transition mechanism this belongs to; there is no "transition mechanism number" in the IPv6 or IPv4 header to signify this. (This can also be viewed as a flexibility benefit.)

Without any kind of security checks (in any of the implemented methods), these often just "work", as the mechanisms aren't differentiated but handled in "one big lump".

Configured tunneling [4] does not suffer from this, as it is point-to-point and based on src_v6/dst_v6 pairs of both IPv4 and IPv6 addresses, so the tunnel interface can be logically deduced.

A Different Model for 6to4 Deployment:

Even though this was already discussed in Section 4.1.2, it bears some additional elaboration, as it was the only problem that cannot be even partially solved using the current deployment model. There are some mitigation methods.

6to4 routers receive traffic from non-6to4 ("native") sources via 6to4 relays. 6to4 routers have no way of matching the IPv4 source address of the relay with the non-6to4 IPv6 address of the source. Consequently, anyone can spoof any non-6to4 IPv6 address by sending traffic, encapsulated, directly to 6to4 routers.

It could be possible to turn the deployment assumptions of 6to4 around a bit to eliminate some threats caused by untrusted 6to4 relays:

o Every dual-stack site (or even ISP) would be required to have its own 6to4 relay. (This assumes that IPv6-only is so far away that 6to4 would be retired by that point.) That is, there would not be third-party relays, and 2002::/16 and 192.88.99.0/24 routes would not need to be advertised globally.

o The security implications of 6to4 use could be pushed back to the level of trust inside the site or ISP (or their acceptable use policies). This is something that the sites and ISPs should already be familiar with already.

However, this presents a number of problems:

This model would shift most of the burden of supporting 6to4 to IPv6 sites that don't employ or use 6to4 at all, i.e., "those who deploy proper native dual-stack." It could be argued that the deployment pain should be borne by 6to4 users, not by the others.

The main advantage of 6to4 is easy deployment and free relays. This would require that everyone the 6to4 sites wish to communicate with implement these measures.

The model would not fix the "relay spoofing problem", unless everybody also deployed 6to4 addresses on the nodes (alongside with native addresses, if necessary), which would in turn change 6to4 to operate without relays completely.

Security Considerations:

Even if proper checks are implemented, there are a large number of different security threats

There are mainly four classes of potential problem sources:

1. 6to4 routers not being able to identify whether relays are legitimate

2. Wrong or impartially implemented 6to4 router or relay security checks

3. 6to4 architecture used to participate in DoS or reflected DoS attacks or made to participate in "packet laundering", i.e.,

making another attack harder to trace

4. 6to4 relays being subject to "administrative abuse" e.g., theft of service or being seen as a source of abuse.

The first is the toughest problem, still under research. The second can be fixed by ensuring the correctness of implementations; this is important. The third is also a very difficult problem, impossible to

solve completely; therefore it is important to be able to analyze whether this results in a significant increase of threats. The fourth problem seems to have feasible solutions.

HASH ALGORITHMS:

Hash algorithms are one-way mathematical algorithms that take an arbitrary length input and produce a fixed length output string and fixed number of bits of data is generated.
This enables security for the transmission of packets from one node to another.
Some of the most used hashing techniques are md5 and sha.

md5 hashing technique :

It accepts a message of any length as input and returns digest value which is to be used for authentication of messages.

Basic idea:

We pre-define a set of ip addresses as valid addresses which are given permission to connect to the server. Now, when we get a packet, we check for the source IP address and obtain the hashed value for it and validate it with the given ip address hashed values. If the validation is successful, the packet is allowed to transmit to other clients from the server, or else it's rejected and sent back to the source.

Implementation in python:

Using a package in Python called 'hashlib' we can generate the hashed value.
For implementing md5 method, we use the function hashlib.md5() which takes byte encoded data of ip address as string and generates the hash value for the string.
Using hexdigest() method, we obtain the hashed value from the object in which it is stored, in the form of string.

IMPLEMENTATION OF 6to4:

The benefits to the enterprise of using 6to4 tunnels are as follows:

- ✓ The end-user host configuration is simple—it requires minimal management overhead.
- ✓ The tunnel is automatic; no enterprise-specific configuration is required at the 6to4 relay site. 6to4 tunnels scale well.
- ✓ The tunnel exists only for the duration of the session.

Limitations of 6to4 Tunnels:

- ❖ The 6to4 tunnel mechanism provides a /48 address block; no more addresses are available.
- ❖ Because 6to4 tunnels are configured many-to-one and tunnel traffic can originate from multiple endpoints, 6to4 tunnels can provide only overall traffic information to the ISP.

__Router 1:__

```
Would you like to enter the initial configuration dialog? [yes/no]: no


Press RETURN to get started!



Router>enable
Router#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#ipv6 unicast-routing
Router(config)#interface fa
% Incomplete command.
Router(config)#interface fastEthernet0/0
Router(config-if)#ipv6 enable
Router(config-if)#ipv6 address 1000:1:1:1:1:1:1:1158/112
Router(config-if)#ipv6 rip 6bone enable
Router(config-if)#no shutdown

Router(config-if)#
%LINK-5-CHANGED: Interface FastEthernet0/0, changed state to up

Router(config-if)#exit
Router(config)#exit
Router#
%SYS-5-CONFIG_I: Configured from console by console
```

<u>Router 2:</u>

```
Would you like to enter the initial configuration dialog? [yes/no]: no


Press RETURN to get started!



Router>enable
Router#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#ipv6 unicast-routing
Router(config)#interface tunnel0

Router(config-if)#
%LINK-5-CHANGED: Interface Tunnel0, changed state to up

Router(config-if)#ipv6 address 2000::1/64
Router(config-if)#ipv6 rip 6bone enable
Router(config-if)#tunnel source fastEthernet0/1
Router(config-if)#tunnel destination 192.168.20.1
Router(config-if)#tunnel mode ipv6ip
Router(config-if)#no shut
Router(config-if)#exit
Router(config)#interface fastEthernet0/1
Router(config-if)#ip address 192.168.10.1 255.255.255.0
Router(config-if)#no shut

Router(config-if)#
%LINK-5-CHANGED: Interface FastEthernet0/1, changed state to up

Router(config-if)#exit
Router(config)#interface fastEthernet0/0
Router(config-if)#ipv6 enable
Router(config-if)#ipv6 address 1000::2/64
Router(config-if)#ipv6 rip 6bone enable
Router(config-if)#no shut

Router(config-if)#
%LINK-5-CHANGED: Interface FastEthernet0/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to
up

Router(config-if)#exit
Router(config)#router ospf 1
Router(config-router)#network 192.168.10.0 0.0.0.255 area 0
Router(config-router)#exit
Router(config)#exit
Router#
%SYS-5-CONFIG_I: Configured from console by console
```

Router 3:

```
Would you like to enter the initial configuration dialog? [yes/no]: no


Press RETURN to get started!



Router>enable
Router#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#interface fastEthernet0/1
Router(config-if)#ip address 192.168.10.2 255.255.255.0
Router(config-if)#no shut

Router(config-if)#
%LINK-5-CHANGED: Interface FastEthernet0/1, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/1, changed state to
up

Router(config-if)#exit
Router(config)#interface fastEthernet0/0
Router(config-if)#ip address 192.168.20.2 255.255.255.0
Router(config-if)#no shut

Router(config-if)#
%LINK-5-CHANGED: Interface FastEthernet0/0, changed state to up

Router(config-if)#exit
Router(config)#router ospf 1
Router(config-router)#network 192.168.10.0 0.0.0.255 area 0
Router(config-router)#network 192.168.10.0 0.0.0.255 area 0
00:59:01: %OSPF-5-ADJCHG: Process 1, Nbr 192.168.10.1 on FastEthernet0/
Router(config-router)#network 192.168.20.0 0.0.0.255 area 0
Router(config-router)#exit
Router(config)#exit
Router#
%SYS-5-CONFIG_I: Configured from console by console
```

Router 4:

```
Would you like to enter the initial configuration dialog? [yes/no]: n
```

```
Press RETURN to get started!



Router>enable
Router#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#ipv6 unicast-routing
Router(config)#interface tunnel0

Router(config-if)#
%LINK-5-CHANGED: Interface Tunnel0, changed state to up

Router(config-if)#ipv6 address 2000::2/64
Router(config-if)#ipv6 rip 6bone enable
Router(config-if)#tunnel source fastEthernet0/0
Router(config-if)#tunnel destination 192.168.10.1
Router(config-if)#tunnel mode ipv6ip
Router(config-if)#no shut
Router(config-if)#exit
Router(config)#interface fastEthernet0/0
Router(config-if)#ip address 192.168.20.1 255.255.255.0
Router(config-if)#no shut

Router(config-if)#
%LINK-5-CHANGED: Interface FastEthernet0/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to
up

%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to up

Router(config-if)#exit
Router(config)#interface fastEthernet0/1
Router(config-if)#ipv6 enable
Router(config-if)#ipv6 address 3000::2/64
Router(config-if)#ipv6 rip 6bone enable
Router(config-if)#no shut

Router(config-if)#
%LINK-5-CHANGED: Interface FastEthernet0/1, changed state to up

Router(config-if)#exi
Router(config)#router ospf 1
Router(config-router)#network 192.168.20.0 0.0.0.255 area 0
Router(config-router)#
01:08:51: %OSPF-5-ADJCHG: Process 1, Nbr 192.168.20.2 on FastEthernet0/0 from
LOADING to FULL, Loading Done
```

```
Router(config-router)#exit
Router(config)#exit
Router#
%SYS-5-CONFIG_I: Configured from console by console
```

Router 5:

```
Would you like to enter the initial configuration dialog? [yes/no]: n


Press RETURN to get started!



Router>enable
Router#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#ipv6 unicast-routing
Router(config)#interface fastEthernet0/1
Router(config-if)#ipv6 enable
Router(config-if)#ipv6 address 3000:1:1:1:1:1:1:1158/112
Router(config-if)#ipv6 rip 6bone enable
Router(config-if)#no shutdown

Router(config-if)#
%LINK-5-CHANGED: Interface FastEthernet0/1, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/1, changed state to
up

Router(config-if)#exit
Router(config)#exit
Router#
%SYS-5-CONFIG_I: Configured from console by console
```

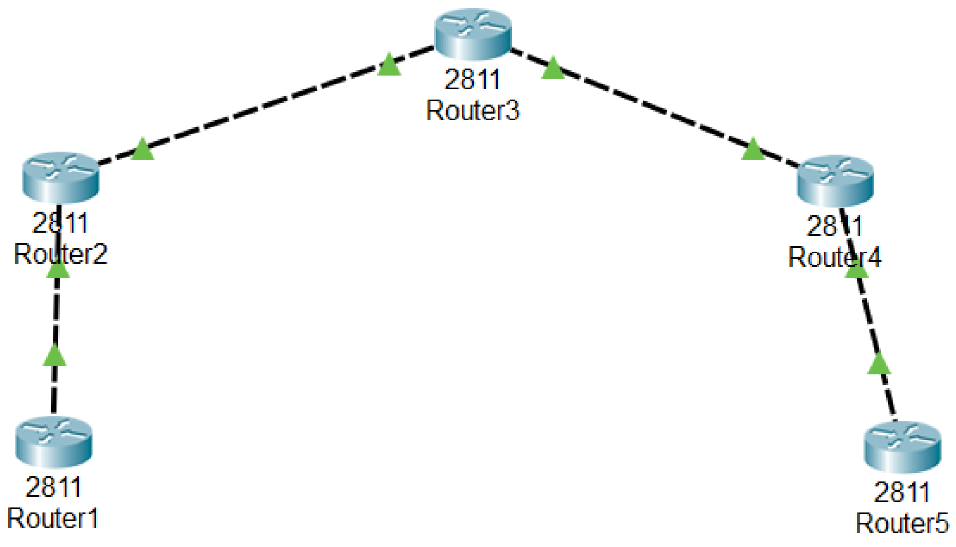
Router 1:

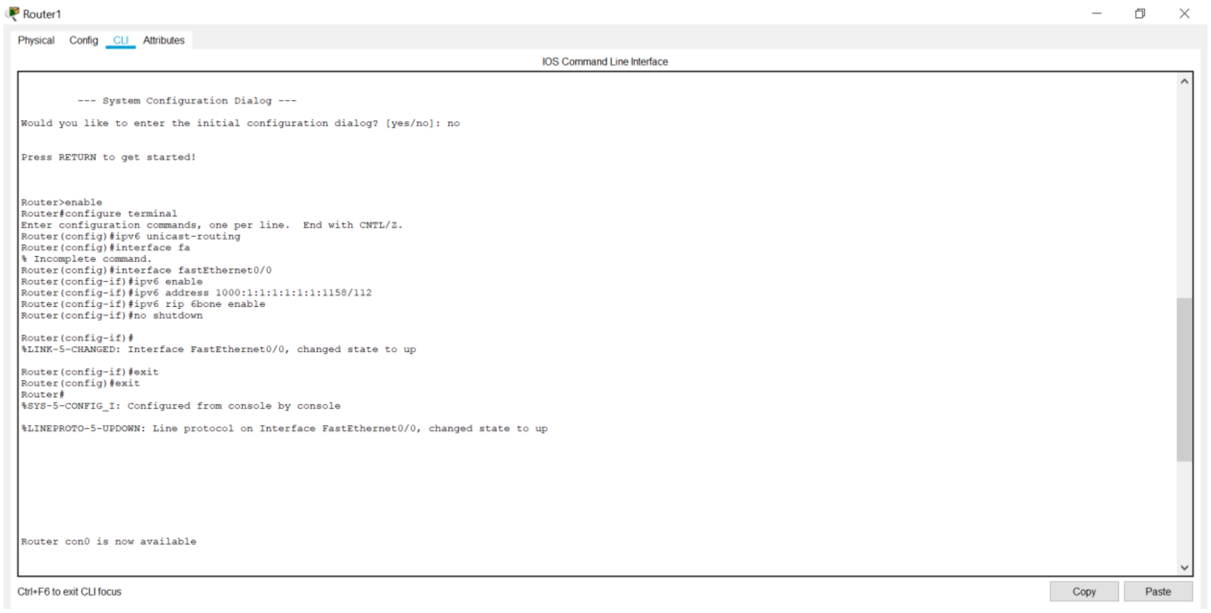
```
Router>ping ipv6 3000:1:1:1:1:1:1:1158

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 3000:1:1:1:1:1:1:1158, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 0/8/28 ms

Router>
```

Router 5:

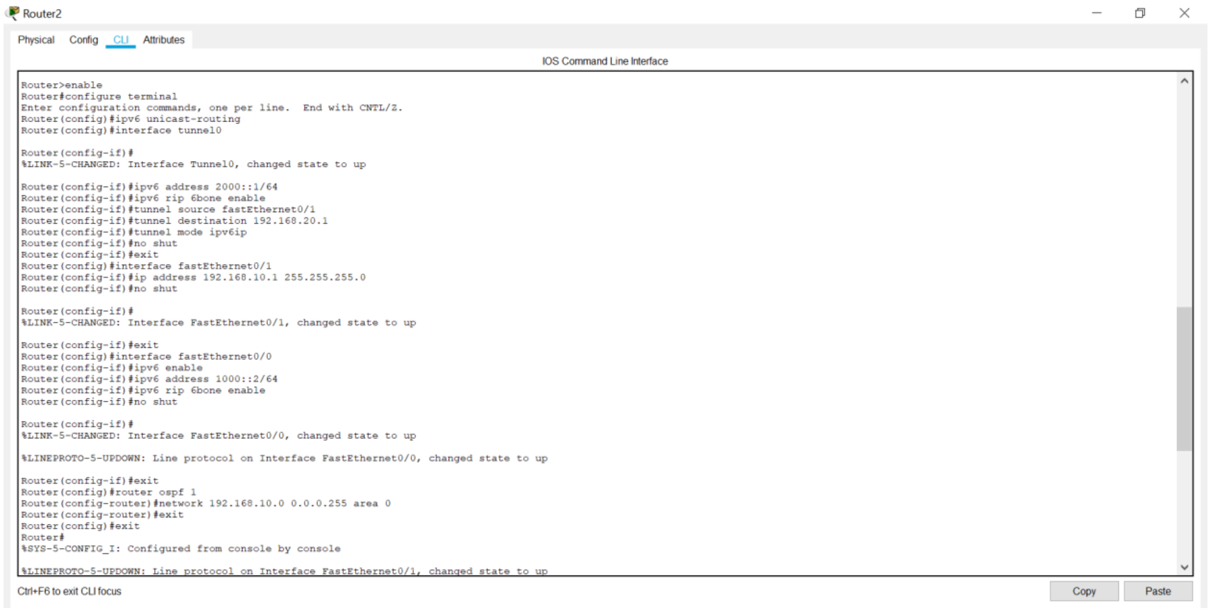
```
Router#ping ipv6 1000:1:1:1:1:1:1:1158

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 1000:1:1:1:1:1:1:1158, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/10/16 ms

Router#
```

Screenshots:

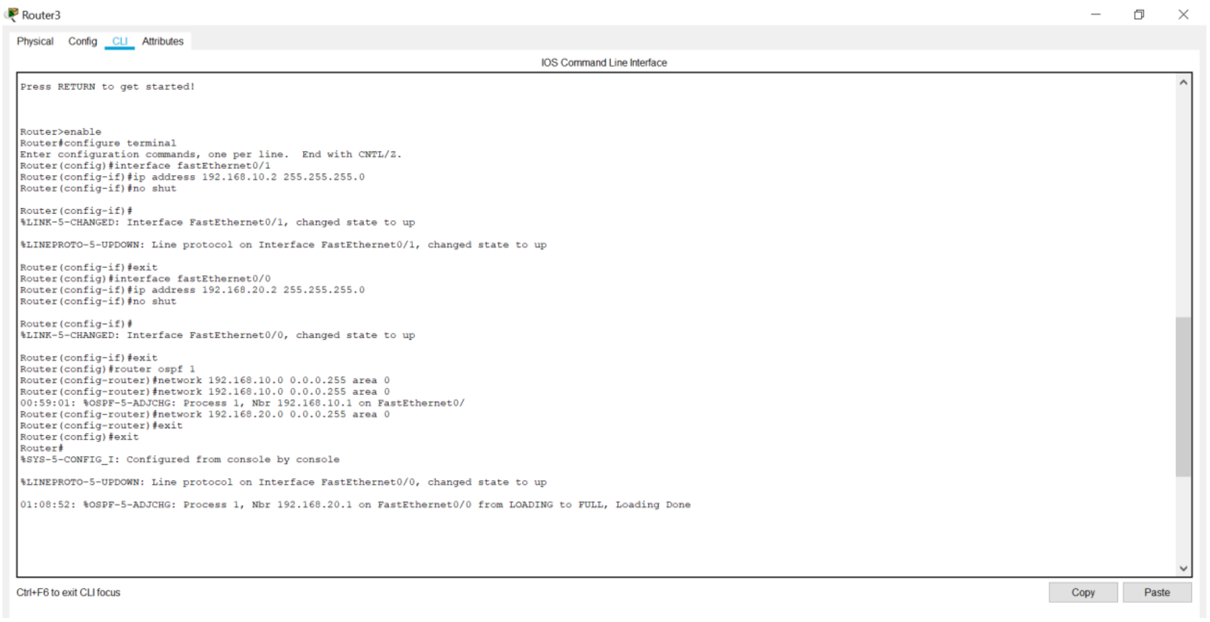




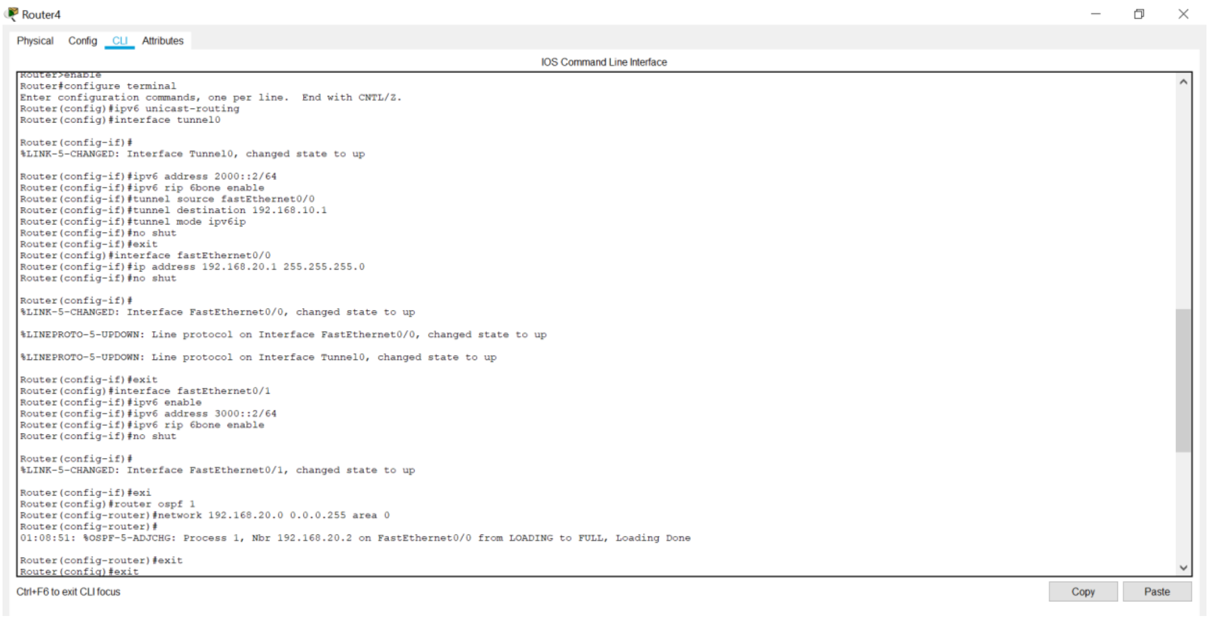
```
        --- System Configuration Dialog ---

Would you like to enter the initial configuration dialog? [yes/no]: no


Press RETURN to get started!



Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ipv6 unicast-routing
Router(config)#interface fa
% Incomplete command.
Router(config)#interface fastEthernet0/0
Router(config-if)#ipv6 enable
Router(config-if)#ipv6 address 1000:1:1:1:1:1:1:1158/112
Router(config-if)#ipv6 rip 6bone enable
Router(config-if)#no shutdown

Router(config-if)#
%LINK-5-CHANGED: Interface FastEthernet0/0, changed state to up

Router(config-if)#exit
Router(config)#exit
Router#
%SYS-5-CONFIG_I: Configured from console by console

%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to up








Router con0 is now available
```



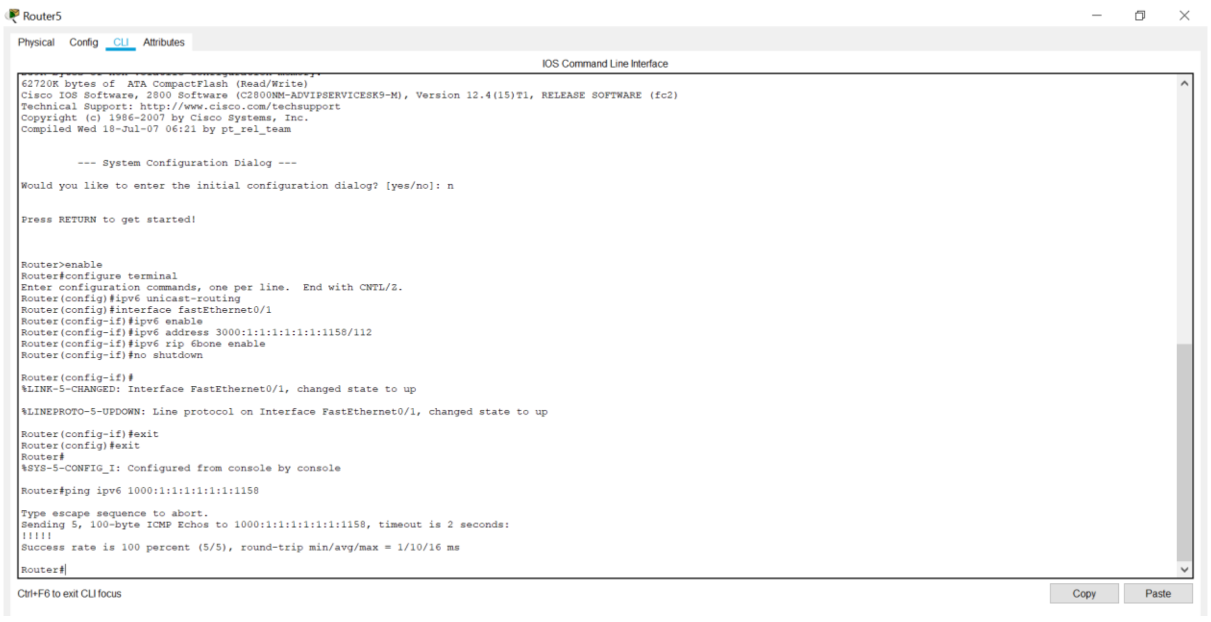
```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ipv6 unicast-routing
Router(config)#interface tunnel0

Router(config-if)#
%LINK-5-CHANGED: Interface Tunnel0, changed state to up

Router(config-if)#ipv6 address 2000::1/64
Router(config-if)#ipv6 rip 6bone enable
Router(config-if)#tunnel source fastEthernet0/1
Router(config-if)#tunnel destination 192.168.20.1
Router(config-if)#tunnel mode ipv6ip
Router(config-if)#no shut
Router(config-if)#exit
Router(config)#interface fastEthernet0/1
Router(config-if)#ip address 192.168.10.1 255.255.255.0
Router(config-if)#no shut

Router(config-if)#
%LINK-5-CHANGED: Interface FastEthernet0/1, changed state to up

Router(config-if)#exit
Router(config)#interface fastEthernet0/0
Router(config-if)#ipv6 enable
Router(config-if)#ipv6 address 1000::2/64
Router(config-if)#ipv6 rip 6bone enable
Router(config-if)#no shut

Router(config-if)#
%LINK-5-CHANGED: Interface FastEthernet0/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to up

Router(config-if)#exit
Router(config)#router ospf 1
Router(config-router)#network 192.168.10.0 0.0.0.255 area 0
Router(config-router)#exit
Router(config)#exit
Router#
%SYS-5-CONFIG_I: Configured from console by console

%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/1, changed state to up
```



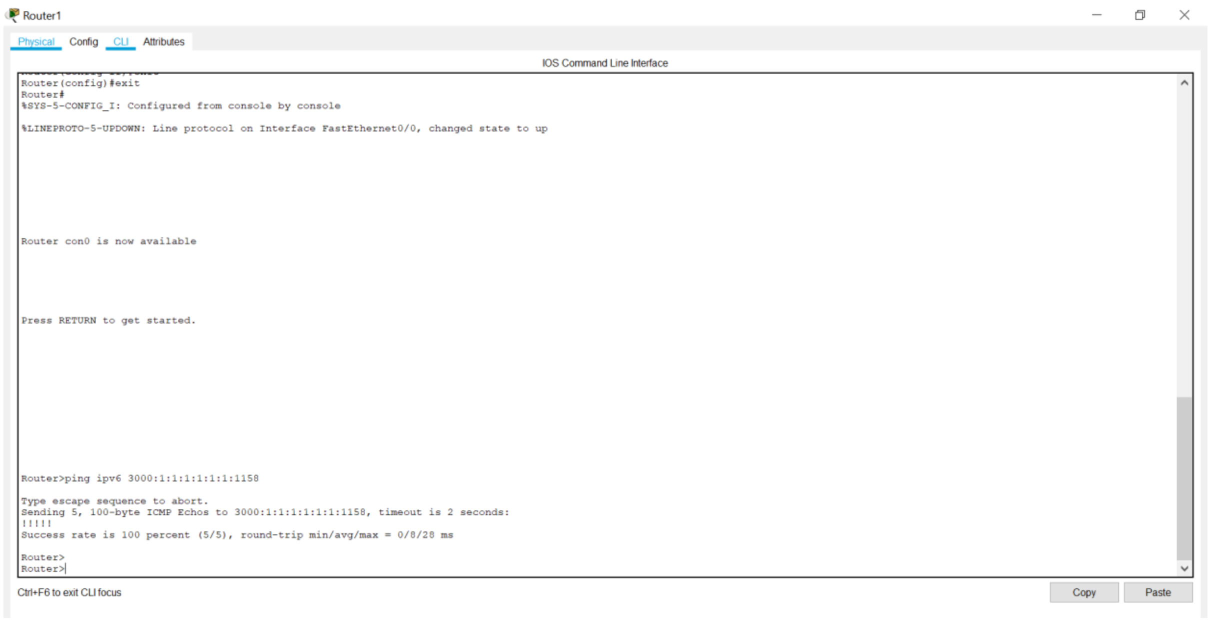
```
Press RETURN to get started!


Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface fastEthernet0/1
Router(config-if)#ip address 192.168.10.2 255.255.255.0
Router(config-if)#no shut

Router(config-if)#
%LINK-5-CHANGED: Interface FastEthernet0/1, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/1, changed state to up

Router(config-if)#exit
Router(config)#interface fastEthernet0/0
Router(config-if)#ip address 192.168.20.2 255.255.255.0
Router(config-if)#no shut

Router(config-if)#
%LINK-5-CHANGED: Interface FastEthernet0/0, changed state to up

Router(config-if)#exit
Router(config)#router ospf 1
Router(config-router)#network 192.168.10.0 0.0.0.255 area 0
Router(config-router)#network 192.168.10.0 0.0.0.255 area 0
00:59:01: %OSPF-5-ADJCHG: Process 1, Nbr 192.168.10.1 on FastEthernet0/
Router(config-router)#network 192.168.20.0 0.0.0.255 area 0
Router(config-router)#exit
Router(config)#exit
Router#
%SYS-5-CONFIG_I: Configured from console by console

%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to up

01:08:52: %OSPF-5-ADJCHG: Process 1, Nbr 192.168.20.1 on FastEthernet0/0 from LOADING to FULL, Loading Done
```



```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ipv6 unicast-routing
Router(config)#interface tunnel0

Router(config-if)#
%LINK-5-CHANGED: Interface Tunnel0, changed state to up

Router(config-if)#ipv6 address 2000::2/64
Router(config-if)#ipv6 rip 6bone enable
Router(config-if)#tunnel source fastEthernet0/0
Router(config-if)#tunnel destination 192.168.10.1
Router(config-if)#tunnel mode ipv6ip
Router(config-if)#no shut
Router(config-if)#exit
Router(config)#interface fastEthernet0/0
Router(config-if)#ip address 192.168.20.1 255.255.255.0
Router(config-if)#no shut

Router(config-if)#
%LINK-5-CHANGED: Interface FastEthernet0/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to up

Router(config-if)#exit
Router(config)#interface fastEthernet0/1
Router(config-if)#ipv6 enable
Router(config-if)#ipv6 address 3000::2/64
Router(config-if)#ipv6 rip 6bone enable
Router(config-if)#no shut

Router(config-if)#
%LINK-5-CHANGED: Interface FastEthernet0/1, changed state to up

Router(config-if)#exi
Router(config)#router ospf 1
Router(config-router)#network 192.168.20.0 0.0.0.255 area 0
Router(config-router)#
01:08:51: %OSPF-5-ADJCHG: Process 1, Nbr 192.168.20.2 on FastEthernet0/0 from LOADING to FULL, Loading Done

Router(config-router)#exit
Router(config)#exit
```



```
62720K bytes of  ATA CompactFlash (Read/Write)
Cisco IOS Software, 2800 Software (C2800NM-ADVIPSERVICESK9-M), Version 12.4(15)T1, RELEASE SOFTWARE (fc2)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2007 by Cisco Systems, Inc.
Compiled Wed 18-Jul-07 06:21 by pt_rel_team


         --- System Configuration Dialog ---

Would you like to enter the initial configuration dialog? [yes/no]: n


Press RETURN to get started!



Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ipv6 unicast-routing
Router(config)#interface fastEthernet0/1
Router(config-if)#ipv6 enable
Router(config-if)#ipv6 address 3000:1:1:1:1:1:1:1158/112
Router(config-if)#ipv6 rip 6bone enable
Router(config-if)#no shutdown

Router(config-if)#
%LINK-5-CHANGED: Interface FastEthernet0/1, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/1, changed state to up

Router(config-if)#exit
Router(config)#exit
Router#
%SYS-5-CONFIG_I: Configured from console by console

Router#ping ipv6 1000:1:1:1:1:1:1:1158

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 1000:1:1:1:1:1:1:1158, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/10/16 ms

Router#
```



```
Router(config)#exit
Router#
%SYS-5-CONFIG_I: Configured from console by console

%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to up









Router con0 is now available






Press RETURN to get started.













Router>ping ipv6 3000:1:1:1:1:1:1:1158

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 3000:1:1:1:1:1:1:1158, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 0/8/28 ms

Router>
Router>
```

REFERENCES:

https://en.wikipedia.org/wiki/6to4

https://en.wikipedia.org/wiki/Ingress_filtering

https://en.wikipedia.org/wiki/6in4#Security_issues

https://labs.ripe.net/Members/emileaben/6to4-why-is-it-so-bad

https://www.theseus.fi/bitstream/handle/10024/98016/Transition%20from%20IPv4%20to%20IPv6.pdf?sequence=1

https://deepitp5.com/2017/06/06/simple-steps-to-tunnel-ipv6-over-ipv4-in-cisco-packet-tracer/

https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=11&cad=rja&uact=8&ved=2ahUKEwjux4_6k6LeAhUEaI8KHTg1DowQFjAKegQIBxAB&url=https%3A%2F%2Fdeepitp5.com%2F2017%2F06%2F06%2Fsimple-steps-to-tunnel-ipv6-over-ipv4-in-cisco-packet-tracer%2F&usg=AOvVaw28XIhgMh_qlzaCh759fBMa