Discovering IPv6-in-IPv4 Tunnels in the Internet

Lorenzo Colitti, Giuseppe Di Battista, and Maurizio Patrignani
Dipartimento di Informatica e Automazione, Università di Roma Tre*
Via della Vasca Navale 79, 00146 Roma, Italy
{*colitti,gdb,patrigna*}*@dia.uniroma3.it*

Abstract

Tunnels are widely used to improve security and to expand networks without having to deploy native infrastructure, and play an important role in the migration to IPv6. In this paper we introduce a number of techniques to detect, and collect information about, IPv6-in-IPv4 tunnels. We also show how a known tunnel can be used as a "vantage point" to launch third-party tunnel-discovery explorations, scaling up the discovery process. We describe our `Tunneltrace` tool, which implements the proposed techniques, and validate them by means of a wide experimentation on the 6bone tunneled network, on the GARR network, and through the test boxes deployed worldwide by the RIPE NCC as part of the Test Traffic Measurements Service. We assess to what extent 6bone registry information is coherent with the actual network topology, and we provide the first experimental results on the current distribution of IPv6-in-IPv4 tunnels in the Internet, showing that even "native" networks reach more than 60% of all IPv6 prefixes through tunnels.

Keywords

Tunnels, IPv6, Tunnel Discovery, IPv6 Topology Discovery, IPv4 to IPv6 Transition

1. Introduction

Tunnelling consists in the encapsulation of the packets of a network protocol within the packets of a second network protocol, such that the former regards the latter as its datalink layer. Because of the flexibility it provides, tunnelling is widely used both to expand networks without having to deploy native infrastructure [20, 11] and to improve security [14, 12]. Tunnels play an important role in the migration to IPv6, and several types of IPv6 tunnels are defined, including configured tunnels and automatic tunnels, 6to4, ISATAP, and Teredo; IPv6 may also use GRE tunnels over IPv4. Our results show that IPv6-in-IPv4 tunnels are very common in the Internet today; we expect this to be true for some time, as IPv4 network infrastructure will remain widely deployed for many years.

Tunnel discovery is the process of automatically detecting tunnels and determining their endpoints. Similarly to other network discovery problems, its importance derives from the need for up-to-date information about network topology, and from the impact

*Work partially supported by European Commission: 6NET (IST-2001-32603) and Fet Open project COSIN (IST-2001-33555); by "Progetto ALINWEB: Algoritmica per Internet e per il Web", MIUR Programmi di Ricerca Scientifica di Rilevante Interesse Nazionale. This work was completed while the first author was visiting the RIPE NCC.

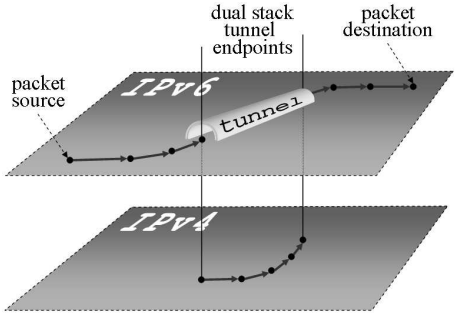

Figure 1 An IPv6 in IPv4 tunnel is seen as a single hop at the IPv6 layer, but IPv6 packets are encapsulated and sent as the payload of IPv4 packets between the tunnel endpoints.

that topology is known to have on crucial aspects of network behavior, such as the dynamics of routing protocols [15], the scalability of multicast [19], the efficacy of denial-of-service countermeasures [22, 18], and other aspects of protocol performance [21].

From a practical perspective, the ability to discover tunnels can be useful in several scenarios. One example is troubleshooting: if a link in the tunnel's path fails, the tunnel fails, and an IPv6 traceroute will not reveal the source of the problem. The ability to determine that the failed link is in a tunnel, and possibly perform an IPv4 traceroute between the tunnel endpoints, would greatly aid debugging. Secondly, tunnels are often used as an interim solution until native IPv6 infrastructure is in place. Tunnel detection techniques provide the means to follow the evolution of the IPv6 Internet from its origin as a completely tunneled network towards a completely native network, and determine how much has to be done to complete the migration to native IPv6. They can also provide insights into the structure of the network itself: for example, as the cost of a tunnel is much lower than that of a native link, predominantly tunneled regions may be more densely interconnected than native regions. The knowledge of these properties will aid the development of realistic IPv6 topology generators. Finally, tunnels offer lower performance than native links and are often used as backup paths in case of problems; the knowledge of whether a particular route contains a tunnel would allow routing protocols to prefer native routes. This is useful for Internet service providers and content delivery operators who wish to maximize the quality of service they provide.

Much has been written on the topic of IPv4 topology discovery, which is usually performed by interacting with the network using probing packets [4, 23] or through the observation of routing information, notably BGP tables [7], bridge forwarding tables [2], or IGP routing tables obtained via SNMP [1]. The combination of these approaches and the use of advanced techniques has led to the development of tools which achieve very good results in relatively little time [23, 1, 7]. However, tunnel discovery differs from other types of network discovery in that a tunneled network is made up of two distinct network layer topologies that interact, and the resulting network is thus a complex "overlay" of two forwarding planes (Figure 1), whose topology cannot be deduced simply by applying known methods to explore each plane separately. The impact of tunnel discovery is also

potentially more significant than that of other types of topology discovery because tunnels are more dynamic than physical links (and can be automatically generated [8]) and because they can undermine both performance, as all the links in the tunnel appear to be a single hop, and security, as the our techniques based on IP spoofing clearly illustrate.

A possible method is the use of SNMP queries to obtain information directly from the nodes involved. This is impractical: not only does it require administrative access to network equipment, and thus cannot be used to discover tunnels in the Internet at large, but the required MIBs are not yet finalized [9], and the specific tunnel MIB is very rarely implemented. Another method was outlined in [3], but it does not apply to existing infrastructure and it envisages authentication mechanisms, and thus has the same drawbacks as the use of SNMP. In this paper, we discuss methodologies for tunnel discovery that do not require administrative access to the network and thus may be applied to the Internet in general. A limited list of our contributions is as follows:

- We introduce techniques to infer the existence of IPv6-in-IPv4 tunnels, confirm the existence of inferred tunnels, and collect information about tunnel endpoints. We show how a tunnel, once discovered, can be used as a "vantage point" to launch third-party tunnel-discovery explorations.
- We describe `Tunneltrace`, a tool which uses our techniques to detect tunnels between a vantage point and a destination.
- We validate the techniques through wide experimentation, first on the 6bone tunneled network, then on native networks accessible through the GARR network and the test boxes deployed worldwide by the RIPE NCC as part of the Test Traffic Measurements Service. As a byproduct of our experimentation, we are able to assess to what extent information in the 6bone registry is coherent with the actual network topology.
- Finally, we provide the first experimental results on the current distribution of tunnels in the Internet, showing that tunnels are very common and that the percentage of native IPv6 connectivity is still very low.

The paper is organized as follows: Section 2 briefly provides the basic definitions and notations used both in Section 3, which introduces and formally describes our tunnel discovery techniques, and in Section 4, which describes `Tunneltrace`. Section 5 describes our experimentation and discusses our results. We conclude in Section 6.

2. Preliminary definitions

Our definitions of node, link and interface are consistent with the IPv6 specifications [6]: a *node* is a device implementing IPv6, a *link* is a communication medium, offered by an underlying link-layer (or, in the case of tunnels, network-layer) protocol, over which the IPv6 protocol may transmit packets, and an *interface* is a node's attachment to a link. A *point-to-point* link is a link to which exactly two interfaces are connected. A *dual stack* interface is an interface on which both IPv4 and IPv6 are enabled. We further (loosely) define a *routable* interface as an interface whose IPv6 address belongs to a prefix which exists in the global routing table and can thus be reached by any host on the network.

An IPv6-in-IPv4 *tunnel*, $T = \langle A, B \rangle$, is a point-to-point link between two dual stack interfaces A (the tunnel *source*) and B (the tunnel *destination*). We denote respectively

with A_4 and B_4 and with A_6 and B_6 the IPv4 and IPv6 addresses of A and B. An IPv6 packet sent through the tunnel is encapsulated in an IPv4 packet sent from A to B with source addresses A_4 and destination address B_4 and with the IPv4 Protocol field set to 41. We represent bidirectional tunnels as two tunnels with the same endpoints in inverted order; thus, if a tunnel $T = \langle A, B \rangle$ is bidirectional, then $T' = \langle B, A \rangle$ also exists. IPv6-in-IPv4 tunnels are modeled as "single-hop" [11], that is, they appear to the IPv6 network as a single point-to-point link which hides the complexity of the underlying IPv4 network.

In the rest of the paper, we shall denote a packet with a pair of square brackets enclosing a source address, a destination address, and other important features of the packet itself. For example, an ICMPv6 echo request message from address X_6 to address Y_6 is written $[X_6 Y_6$ echo-request$]$. Packet encapsulation is described by recursively using square brackets: if the aforementioned IPv6 packet were encapsulated in an IPv4 packet, it would be written $[A_4 B_4 [X_6 Y_6$ echo-request$]]$. To denote the interface that originates or receives a packet, we prepend the packet with the interface followed by a colon or append to the packet the interface a colon followed by the interface, e.g. $X{:}[X_6 Y_6$ echo-request$]{:}Y$ is a packet sent by interface X and received by interface Y. Finally, if the reception of a packet causes a node to emit another packet, we indicate this with the symbol $\triangleright$. For example, if an echo request packet causes a node to reply with an echo reply packet, we write $[X_6 Y_6$ echo-request$] \triangleright [Y_6 X_6$ echo-reply$]$.

3. Tunnel discovery methods

In this section, we present a number of techniques we developed to tackle the tunnel discovery problem. Depending on their objective, they may be divided into: (i) techniques to infer the existence of tunnels, (ii) techniques to confirm their existence, (iii) techniques to collect information about their endpoints, and (iv) techniques which allow a host to interact with the network as if it were located in a different place to the one in which it is actually located (*third party exploration* techniques). They may further be characterized according to their mode of operation: some query known sources of information, others interact with the network and observe the results, performing what we may refer to as "active probing". Each rule is a suitable blending of the following basic methods:

Path MTU discovery The *Maximum Transmit Unit* (MTU) of a link is the maximum size of a packet that may be transmitted through the link. The path MTU between two interfaces X and Y is the minimum MTU of the links composing the path between X and Y. Path MTU discovery [16] is a method that allows a node to determine the path MTU between one of its interfaces and another interface on the network, and thus obtain information on the MTUs of the intervening links. The presence on the path of certain MTU values may suggest the presence of a tunnel.

DNS lookups The Domain Name System is used to map IPv4 and IPv6 addresses to hostnames and vice versa. Often the IPv4 and IPv6 addresses of an interface have the same name; as tunnel interfaces are dual stack, DNS lookups can provide information about tunnel endpoints. DNS queries also help determine if an interface is dual stack.

IP spoofing Because IPv6-in-IPv4 tunnels do not use any form of authentication, a tunnel destination will accept an encapsulated packet sent by any host as long as the source

#	Rule	Infer existence	Confirm existence	Collect information	Third-party exploration
1	MTU	×			
2	DNS	×		×	
3	Packet injection		×		×
4	Fragment injection		×		×
5	Injected ping		×		
6	Dying packet		×	×	
7	Ping-pong packet			×	
8	Bouncing packet			×	

Table 1 Classification of tunnel discovery techniques.

IPv4 address of the packet is the IPv4 address of the tunnel source. This allows any host to cause the tunnel endpoint to emit arbitrary IPv6 packets by encapsulating them in IPv4 packets with spoofed source addresses.

Hop Limit manipulation The Hop limit field in the IPv6 header specifies the maximum number of routers a packet may pass through. When a router receives a packet with the Hop Limit field equal to 1 it discards it and sends the packet's source an ICMPv6 error message, thus revealing its IPv6 address.

IPv6 Routing header While source routing is prohibited in the majority of IPv4 networks, many IPv6 routers honor the IPv6 Routing header, which permits a host sending a packet to specify a list of nodes that the packet is to pass through. Combined with Hop Limit manipulation, the Routing header can be useful for determining the addresses of point-to-point interfaces and tunnel interfaces in particular.

The remainder of this section is devoted to a formal presentation of the main techniques we have devised. Each technique is expressed by means of a formal rule, which is identified by a number and by a short name. Table 1 classifies the rules according to their objective. Although each rule is expressed by means of an implication, the validity of the implication is not absolute, and in real-world conditions a rule may fail to apply due to nonstandard behavior, misconfiguration, or unexpected and uncommon network topologies. Data on the validity of the rules will be provided in section 5.

Rule 1 (MTU) Consider the sequence of links that make up the path between some interface X and some other interface Y. We may think of each link as a point-to-point link, because each packet that traverses a link is sent by exactly one of the interfaces on the link and is received by exactly one of the interfaces on the link*. Thus, if we number the links in the path progressively starting from 1, for each we may define a source interface $A(i)$ and a destination interface $B(i)$, which have IPv6 addresses $A_6(i)$ and $B_6(i)$ (Figure 2).

Let $MTU(i)$ be the MTU of link i. If we can send packets from X, then we may use Path MTU discovery [16] to determine, for each link, the value $\mathit{PMTU}(i) =$

*This is usually the case for all packets, but load-balancing mechanisms or policy routing may cause behavior that varies from packet to packet.

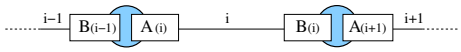

Figure 2 A generic link i in a path with its associated interfaces $A(i)$ and $B(i)$.

$\min\{\mathit{MTU}(i), \mathit{PMTU}(i-1)\}$, where $\mathit{PMTU}(1) = \mathit{MTU}(1)$. Because of encapsulation, the MTU of the tunnel is less than that of the underlying IPv4 network by a fixed amount depending on the tunnel type: 20 bytes for IPv6-in-IPv4 tunnels, 24 or 28 for GRE [13] tunnels. The most common MTU value on the IPv4 Internet today is 1500 bytes, so these tunnels will almost always have MTUs of 1480 and 1476 (or 1472) bytes respectively. Finally, many tunnel interfaces (notably on BSD systems) use a default MTU of 1280 bytes. Hence, if we consider two consecutive links $i-1$ and i on the path, we may write:

$$\mathit{PMTU}(i) < \mathit{PMTU}(i-1) \wedge \mathit{PMTU}(i) \in \{1480, 1476, 1472, 1280\} \Rightarrow Tunnel(A(i), B(i))$$

where $Tunnel(A(i), B(i))$ means that there is a tunnel between $A(i)$ and $B(i)$. Of course, if the tunnel is entirely contained in a portion of the IPv4 Internet where the MTU of all the links is higher than 1500, this rule may fail to detect a tunnel. It may also wrongly detect a link as a tunnel if an IPv6 link is manually configured to have an MTU equal to these values. This may be particularly common in the case of 1280 bytes, which is the minimum MTU permitted by the IPv6 specifications. The MTU rule is confirmed by experience in all the IPv6 networks on which we tested; note, however, that it will only find a tunnel if its MTU is lower than the MTU of all previous links in the path.

Rule 2 (DNS) We represent DNS lookups with a function, $Name()$, that accepts an IPv4 or IPv6 address and returns the corresponding DNS name, and two functions, $Addr_4()$ and $Addr_6()$, that accept a DNS name and provide the corresponding IPv4 or IPv6 address. If a name of an interface X has both an IPv6 and an IPv4 address, we may presume that it is dual stack and that the two addresses are its IPv4 and IPv6 address. Formally:

$$\exists \alpha | \alpha = Addr_4(Name(X_6)) \Rightarrow DualStack(X) \wedge X_4 = \alpha$$

$$\exists \beta | \beta = Addr_6(Name(X_4)) \Rightarrow DualStack(X) \wedge X_6 = \beta$$

Rule 3 (Packet injection) Given two IPv4 addresses A_4 and B_4, if there is a tunnel between A and B, it is possible to cause an arbitrary (though limited in size) IPv6 packet to enter the IPv6 network at interface B. This is done by sending, from any routable interface Z, an IPv6 packet encapsulated in an IPv4 packet with source and destination addresses A_4 and B_4. Because its source address is A_4, when the packet arrives at B it will be recognized as arriving from the tunnel and will be decapsulated and processed as if it had been sent by A (see Figure 3(a)). Formally, we may write:

$$Tunnel(A, B) \Rightarrow Z{:}[A_4 B_4 [X_6 Y_6 \text{ payload}]] \rhd [X_6 Y_6 \text{ payload}]{:}B$$

where the payload of the two IPv6 packets is the same. This technique may be used to "inject" an arbitrary IPv6 packet, up to the maximum size permitted by the MTU of the underlying IPv4 network minus the size of the IPv4 header, into the IPv6 network at interface B. We refer to this technique as *packet injection* and to Z as the *injecting interface*. Note that the packet, although sent by Z, actually enters the IPv6 network at interface B, and, as far as the IPv6 network is concerned, is simply a packet originated

by a node on the same link as B. Thanks to this technique, the injecting host may send packets as if it were physically located on the same link as B; if the node r to which B belongs is a router, the injected packet will be forwarded as normal towards X as if it had been sent by r itself. We then say that r is a *vantage point*. Thanks to this rule, a single host in a single location may interact with and explore the network as if it were located simultaneously in all the vantage points it is aware of. Note, however, that because it depends on IP spoofing, both this rule and Rules 4, 5, 6 and 7 which depend on it, will not work if the network in which B is located makes use of ingress filtering.

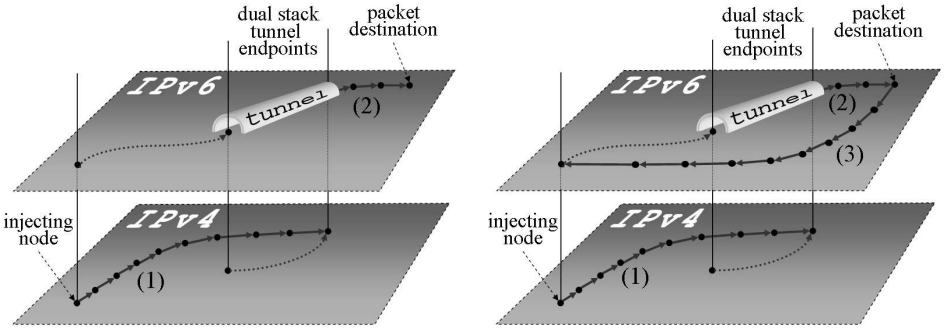

Figure 3 (a) Packet injection: a spoofed IPv4 packet is sent to a tunnel endpoint, is processed as if it had been sent by the other endpoint, and is forwarded to its destination. (b) An IPv6 echo-request packet is injected. The destination replies with an IPv6 echo-reply addressed to the source address of the encapsulated packet.

Rule 4 (Fragment injection) IPv6 packets injected using the packet injection technique described in Rule 3 are limited in size to the MTU of the underlying IPv4 network minus the size of the encapsulating headers. However, it is possible to inject a larger packet by fragmenting the IPv4 packet which encapsulates it. For example, suppose the packet $[A_4B_4[X_6Y_6 \text{ payload}]]$ is fragmented by the IPv4 network into two IPv4 packets f_1 and f_2. Upon arrival at B, the packet will be reassembled (resulting in an IPv4 packet larger than the MTU of the IPv4 network) and will be processed according to Rule 3:

$$Tunnel(A, B) \wedge f_1 \wr f_2 = [A_4B_4[X_6Y_6 \text{ payload}]] \Rightarrow Z{:}f_1 \wedge Z{:}f_2 \rhd [X_6Y_6 \text{ payload}]{:}B$$

This rule permits a host to use any vantage point r to inject IPv6 packets of arbitrary size from r as if it had a direct native connection to r. It is particularly useful in the search for tunnels: for example, by combining this rule with Rule 1, we may perform Path MTU discovery from the node having interface B.

Rule 5 (Injected ping) Given two IPv4 addresses A_4 and B_4, it is possible to determine whether there is a tunnel $T = \langle A, B \rangle$ by applying Rule 3, with $X_6 = Z_6$, to inject an echo request packet addressed to any routable interface Y. The packet will arrive at interface B; if there is no tunnel between A and B, it will be discarded. Otherwise, it will

be forwarded to its destination Y, which will reply with an echo reply message addressed to Z_6. If the injecting host receives a reply, there is a tunnel (Figure 3(b)). More formally,

$$Z{:}[A_4 B_4[Z_6 Y_6 \text{ echo-request}]] \rhd [Y_6 Z_6 \text{ echo-reply}]{:}Z \Rightarrow Tunnel(A, B)$$

Rule 6 (Dying packet) Given a tunnel $T = \langle A, B \rangle$, it is possible to determine the IPv6 address B_6 of the tunnel destination by injecting a packet with the IPv6 Hop Limit field set to 1. Because IPv6-in-IPv4 tunnels are modeled as "single-hop", the packet will appear at interface B without ever having been processed by an IPv6 router, and thus with the contents of the Hop Limit field intact. Upon arrival at interface B, however, the Hop Limit of the packet will be decremented to zero. The resulting "time exceeded" message will arrive at Z and the injecting host may determine B_6 by examining its source address [5, section 2.2]. Stating this in terms of a rule, we have:

$$Z{:}[A_4 B_4[Z_6 X_6 \text{ HL=1}]] \rhd [Y_6 Z_6 \text{ time exceeded}]{:}Z \Rightarrow B_6 = Y_6$$

If the tunnel is bidirectional, it is possible to determine the IPv6 address of the other endpoint simply by exchanging A_4 and B_4.

Rule 7 (Ping-pong packet) Rule 6 does not allow us to determine the IPv6 address A_6 of the tunnel source if the tunnel is not bidirectional. However, it is frequently possible to determine it by other means. Suppose the tunnel has an IPv6 prefix T associated with it. Any IPv6 address in T will be routed towards the tunnel, and each tunnel endpoint will route through the tunnel all addresses in T except its own. Thus, if we use Rule 5 to inject an echo request packet with a Hop Limit of 2 and destination address X_6 in T but not equal to B_6, the packet will reach B and be sent back through the tunnel to A. If $X_6 = A_6$, then the injecting host will receive an echo reply. Otherwise*, it will receive a "time exceeded" message with source address A_6. In both cases, it obtains A_6.

It is simple to determine a suitable value for X_6: the length of T must be at most 127, because otherwise A_6 and B_6 cannot both be in T. So, whatever the length of T, $B_6 \pm 1$ (where the sign depends on whether B_6 is even or odd) is always in T. So we may write:

$$\begin{array}{lcl} Z{:}[A_4 B_4[Z_6 X_6 \text{ echo-request,HL=2}]] \rhd [X_6 Z_6 \text{ echo-reply}]{:}Z & \Rightarrow & A_6 = X_6 \\ Z{:}[A_4 B_4[Z_6 X_6 \text{ echo-request,HL=2}]] \rhd [Y_6 Z_6 \text{ time exceeded}]{:}Z & \Rightarrow & A_6 = Y_6 \end{array}$$

where

$$X_6 = \begin{cases} B_6 + 1 & \text{if } B_6 \text{ is even} \\ B_6 - 1 & \text{if } B_6 \text{ is odd} \end{cases}$$

Note that if the tunnel interfaces are unnumbered, A_6 and B_6 are not on the same subnet and this rule does not apply.

Rule 8 (Bouncing packet) Consider the path from some interface Z to some other interface W. An IPv6 traceroute from Z to W allows us to determine the sequence $B_6(i)$, where $i = 1 \ldots n$ is the i-th link in the path, but it does not allow us to determine $A_6(i)$ for any i. However, Z may use the IPv6 Routing header to send a packet to $B(i)$ which

*If subnet-router anycast addresses are used, X_6 may belong to the same router as B_6. However, the injecting host may determine if that is the case simply by sending $[A_4 B_4[Z_6 X_6$ echo-request,HL=1$]$ and seeing if B responds with an echo reply or a time exceeded. If it responds with an echo reply, then subnet-router anycast is active, so the subnet is at least a /126. The injecting host can then choose X_6 as another address in the subnet.

is routed back towards itself; if the Hop Limit *HL* of this packet is set to the appropriate value, the packet will expire on interface $A(i)$ and Z will receive a "time exceeded" message with source address $A_6(i)$. The required value of *HL* is not necessarily $i+1$, as the path taken by a packet $[Z_6 B_6(i)]$ may not be a subpath of the path taken by a packet $[Z_6 W_6]$. However, *HL* may be determined by adding one to the distance x between Z and $B(i)$, which can be measured, for example, by running a traceroute from Z to $B_6(i)$.

In the presence of asymmetric routing, this may not provide $A_6(i)$, because the path from $B(i)$ to Z may not be the same as the path from Z to $B(i)$. This problem may be partially overcome by setting the packet's destination not to Z_6 but to a previous hop on the path, to reduce the effects of route asymmetry. As ICMPv6 specifies[5, section 2.2] that if the packet is sent to $B_6(i-1)$, the source address of the error message must be $B_6(i-1)$ and not $A_6(i)$, Z may set the destination address to $B_6(i-2)$ *. More formally,

$$Z{:}[Z_6 B_6(i) B_6(i-2)\ \text{HL}{=}x+1] \rhd [Y_6 Z_6\ \text{time-exceeded}]{:}Z \Rightarrow A_6(i) = Y_6$$

where $[Z_6 B_6(i) B_6(i-2)]$ indicates a packet source routed through $B_6(i)$ with destination $B_6(i-2)$. While this rule applies to any link, including tunnels, it is particularly useful when combined with Rule 2 to determine tunnel endpoints given path information; if the IPv4 addresses of the tunnel endpoints are known, then Rules 6 and 7 are more effective.

4. A tunnel discovery tool

In this section we describe `Tunneltrace`, a tunnel discovery tool we have developed to test the techniques introduced in Section 3. Although `Tunneltrace` is not intended to be the main contribution of our work, which we believe lies in the techniques themselves, we discuss it here as an example of their application. `Tunneltrace` attempts to detect and collect information about tunnels in the path between the exploring host and a user-specified destination. Of course, by applying Rule 4, in principle it is possible, given a sufficient number of vantage points, to find tunnels in the entire network.

The strategy followed by `Tunneltrace` is simple: perform a traceroute to the destination host, and for each link i attempt to discover if it is a tunnel. If it is, attempt to discover the IPv4 addresses of the endpoints, confirm the tunnel's presence, and use it as a vantage point to explore the rest of the path.

Specifically, for each hop in the traceroute $B_6(i)$, `Tunneltrace` first applies Rule 1 to determine whether link i is a tunnel. If so, it attempts to obtain information about its endpoints in the following way: first, it attempts to obtain the IPv6 address of the previous hop's sending interface, $A_6(i)$, using Rule 8; then, it uses Rule 2 to attempt to obtain $A_4(i)$ and $B_4(i)$, and if it succeeds, it attempts to confirm the presence of the tunnel using Rule 5; finally, it verifies the information collected by using Rules 6 and 7. If the tunnel is confirmed, it is used as a vantage point to explore the rest of the path.

If Rule 2 does not provide enough information to use the tunnel as a vantage point, `Tunneltrace` combines it with heuristics on DNS names, attempting to perform piece-

*This may still provide incorrect results, because the asymmetry may be located between $B(i)$ and $B(i-2)$. If greater accuracy is desired, the inferred value of $A_6(i)$ can be compared with $B_6(i)$ and accepted only if it is on the same subnet; however, this will cause false negatives for unnumbered links.

Date	Total Tunnels	Up	Down	One endpoint unknown to DNS	Both endpoints unknown to DNS
2003-06-13	4334	998	1479	1328	529
2003-06-23	4319	1058	1394	1333	534
2003-07-18	4202	998	1322	1342	540
2003-08-07	4197	1046	1345	1316	490

Table 2 Status of tunnels in the 6bone registry

wise matching as proposed in [17], and, if the name contains strings such as "v6-", "ip6.", or "ipv6.", repeating the DNS lookup after removing them.

`Tunneltrace` also examines names looking for strings that suggest the presence of tunnels (such as "tunnel" or "tu"), queries the 6bone registry to check whether the node is a known tunnel endpoint, and performs AS lookups: if the IPv4 address of a node is in a different AS as its IPv6 address, or if the hops before and after the node are in different ASs as the node itself, the node may be the endpoint of an interdomain tunnel. In all these cases, `Tunneltrace` reports that a tunnel might be present.

For each hop, `Tunneltrace` also outputs information such as the IPv6 address (and DNS name and AS number) of the answering interface $B_6(i)$ and whether the interface is dual stack. It also provides this information about the sending interfaces $A(i)$.

5. Experimental results

The 6bone provides a useful experimental testbed for our work, as data on tunnels is publicly available in the 6bone registry. Thus, applying our techniques to the 6Bone may both (i) allow us to verify the validity of our techniques, and (ii) use our techniques to check the accuracy of the information in the registry itself. We used the tunnel data available in the 6bone registry in various ways. Firstly, we checked it for consistency, using DNS lookups and packet injection to determine how many tunnels in the tunnel database actually exist. Secondly, we used it as a large list of tunnels against which to check the validity of our tunnel discovery techniques. Finally, we used the list of existing tunnels as vantage points from which to search for tunnels in the IPv6 Internet at large.

5.1 Status of the 6bone registry

We have analyzed the 6bone registry a number of times over a two-month period to determine to what degree the information on tunnels it contains is accurate and up-to-date. For every tunnel, the registry contains the DNS names or IPv4 addresses of the tunnel endpoints, along with other information. We process one tunnel at a time, and attempt to resolve the DNS hostnames of the tunnel endpoints to IP addresses. If both names can be translated to IPv4 addresses, we use Rule 5 to determine whether the tunnel is actually working. The results of our analysis are in Table 2.

Our results show that almost half of all tunnel records in the registry have invalid DNS names for one or both endpoints and therefore are either out of date or refer to tunnels that no longer exist. About a quarter of the records are working tunnels. A further third do not

permit packet injection. While some of these may be GRE tunnels and/or have endpoints located in networks that employ ingress filtering, we expect most of them to be inactive: our MTU survey results indicate that GRE tunnels are much less common than IPv6-in-IPv4 tunnels, and as the majority of tunnels in the 6bone registry are interdomain tunnels, it is unlikely that ingress filtering has any significant impact on the results. Further study of these undecided cases would allow the development of a tool which could monitor all aspects of the quality of a tunnel registry.

As regards variation over time, notwithstanding the short time window in which our observations were made, our results indicate that the 6bone registry is fairly static, and that the quality of tunnel data is marginally improving: the percentage of working tunnels increased from 23.4% on 2003-06-13 to 24.9% on 2003-08-07. This is due to the fact that the total number of tunnels is decreasing, possibly because of the 6bone phaseout [10].

5.2 Rule validity data

The large number of working tunnels provided by the 6bone registry allows us to validate our tunnel discovery techniques against known data: once a tunnel is confirmed using Rule 5, we may check the validity of Rules 2, 4, 6, and 7. Using the 2003-08-07 dataset, we checked whether these rules applied to the tunnels in the registry that we had confirmed to be working. Of a total sample of 1046 tunnels, we found that Rule 4 (Fragment Injection) applied to 999 tunnels (95.5%) and Rule 6 (Dying packet) applied to 1013 tunnels (96.8%). Rule 7 was tested only on tunnels that did not permit packet injection in both directions, because for these, Rule 6 is much more effective. Of 218 tunnels that were not bidirectional, Rule 7 applied to 151 (69.2%). Together, Rules 6 and 7 allowed us to determine both IPv6 endpoints for 963 tunnels (92.1%).

Rule 2 (DNS) was significantly less useful: of the 963 tunnels for which we knew the IPv6 addresses of both endpoints, it applied to one endpoint in 169 cases (17.5%), and to both endpoints in only 6 cases (0.6%). Though its utility is rather limited, we feel that as it is probably the most obvious of our techniques, our work would not be complete without discussing it and determining its degree of usefulness. Clearly, the techniques that make use of active probing produce significantly better results than can be obtained by querying online sources of information such as the 6bone registry or the DNS.

5.3 MTU survey

The large number of vantage points obtained from the 6bone registry allows us to use third-party exploration to evaluate the impact of tunnels in a sizable portion of the IPv6 Internet. Using a sample of 995 vantage points in 92 different Autonomous Systems (AS's; for comparison, the total number of AS's which announce IPv6 routes is approximately 450), we applied Rule 4 to perform Path MTU discovery from each vantage point to every prefix in the IPv6 routing table. By applying Rule 1, we may deduce which paths contain one or more tunnels and which are native. The results of our analysis, excluding connectivity errors, are in Table 3.

We note that the most common MTU is 1480 bytes, that of an IPv6-in-IPv4 tunnel, followed by 1280, the minimum IPv6 MTU, which indicates that at least one link in the path has a MTU of 1280 bytes (probably an IPv6-in-IPv4 tunnel on a BSD system). The

MTU value	Number of paths	Percentage
1480	150946	39.4%
1280	138358	36.1%
1476	44404	11.6%
1500	31525	8.2%
1428	13619	3.6%
Other	4104	1.1%
Total	382956	100.0 %

Table 3 Path MTU values from vantage points to the Internet

paths with an MTU of 1476 are probably due to GRE tunnels, while the paths with a MTU of 1428 may be due to encapsulation of IPv6 in a L2TP VPN. Native paths (those with a MTU of 1500) make up only 8.2% of all the paths we surveyed. A relatively low percentage of native paths is to be expected, given the fact that our vantage points are tunnel endpoints, and each probably reaches some percentage of probably reached through the tunnel itself; nevertheless these results allow us to affirm that the percentage of native paths in the IPv6 Internet is still quite low.

5.4 Survey of "native" IPv6 networks: how native is native?

So far we have observed the IPv6 Internet through vantage points that are tunnel endpoints. Since these may be located in portions of the network that are dense in tunnels, we conducted a survey from hosts inside two native IPv6 networks, one in the GARR native IPv6 network and one at RIPE NCC, to discover how tunneled "native" networks really are. From each host we measured the Path MTU to every prefix in the global IPv6 BGP table and applied Rule 1 to determine whether the path to each prefix contained at least one tunnel. We found that of 417 prefixes in the BGP table at the time of the analysis, the GARR network reached at least 262 (62.8%) through tunnels, while the RIPE NCC network reached at least 297 (71.2%) through tunnels.

In order to obtain a more complete picture of the effect of tunnels on global IPv6 connectivity, we repeated the experiment from the "test-boxes" deployed worldwide by RIPE NCC as part of the Test Traffic Measurements service. About 100 test-boxes are currently active, of which 16 have IPv6 connections. The same prefixes were used for all test-boxes; since not all test-boxes could reach all prefixes, prefixes that were unreachable from a particular test-box were removed from its list so as not to affect the results.

Our results are in Figure 4. As can be seen from the graph, three of the test-boxes reached almost 100% of the prefixes through tunnels, from which we deduce that they are located in networks which do not have a native IPv6 connection; the others reached between 63.7% and 92.3% of all IPv6 prefixes through tunnels (the column labeled "tt average A-P" provides an average for all 16 test-boxes, while the column labeled "tt average E-P" provides the average of all natively connected test-boxes). The results show that global IPv6 connectivity still relies largely on tunnels, even when observed from a native IPv6 network. However, we note that there are non-trivial differences between

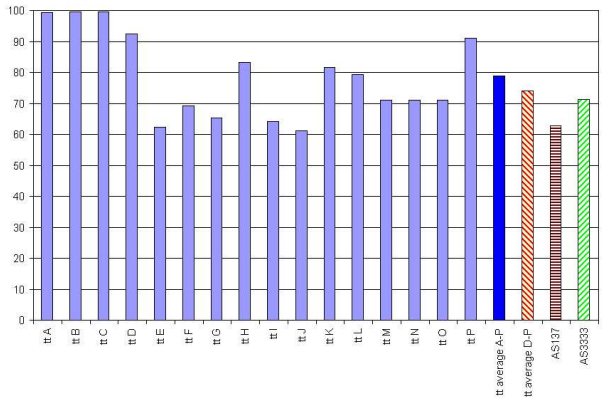

Figure 4 Percentage of tunneled prefixes seen by TTM test-boxes and two native sites.

the percentage of native destinations reached by the various test-boxes, indicating that measurements such as these offer a good indication of the quality of an IPv6 network.

6. Conclusions

We have introduced several techniques to infer the existence of IPv6-in-IPv4 tunnels, to confirm their existence, and to collect information about their endpoints, outlining a strategy for tunnel discovery along a path and showing how it is possible to use tunnels as ”vantage points” to inject packets into the network at multiple locations, performing third-party exploration and scaling up the discovery process.

By applying our techniques to the 6bone registry, we were able to assess to what degree it is coherent with the actual state of the network, showing that almost half of the information on tunnels is out of date, but that at least one quarter of tunnels present is still functioning. The information in the 6bone registry also allowed us to verify the validity of our techniques, showing that those which make use of active probing are very effective, providing results for over 90% of the tunnels in the registry we were able to make use of.

We used our techniques to provide the first experimental data on the presence of tunnels in the IPv6 Internet, by measuring the percentage of IPv6 prefixes reached through tunnels from the native GARR and RIPE NCC networks and from the 16 IPv6 test-boxes deployed worldwide by the RIPE NCC as part of the Test Traffic Measurements service. All the networks we tested reached less than 40% of IPv6 prefixes natively, showing that global IPv6 connectivity still relies largely on tunnels.

Note that our techniques expose, and make use of, security problems inherent in IPv6-in-IPv4 tunnels, which do not use any form of authentication except the source address of the IPv4 packet. This allows any host to inject packets into any IPv6 network in which it knows the IPv4 addresses of the endpoints of a tunnel, with security implications similar to those of source routing. If security is a factor, we recommend the use of native links, or, if this is not possible, of GRE tunnels [13] or keyed GRE tunnels.

References

[1] G. Barbagallo. Polyphemus, a system for discovering and visualizing OSPF networks. http://www.dia.uniroma3.it/~ polyph/.

[2] Yigal Bejerano, Yuri Breitbart, Minos Garofalakis, and Rajeev Rastogi. Physical topology discovery for large multi-subnet networks. In *Proc. IEEE/INFOCOM '03*, 2003.

[3] R. Bonica, K. Kompella, and D. Meyer. Tracing requirements for generic tunnels. Work in progress.

[4] H. Burch and B. Cheswick. Mapping the Internet. *IEEE Computer*, 32(4):97–98, April 1999.

[5] A. Conta and S. Deering. Internet control message protocol (ICMPv6) for the Internet Protocol version 6 (IPv6) specification. RFC 2463, December 1998.

[6] S. Deering and R. Hinden. Internet Protocol, version 6 (IPv6) specification. RFC 2460, December 1998.

[7] Giuseppe Di Battista, Federico Mariani, Maurizio Patrignani, and Maurizio Pizzonia. Archives of BGP updates: Integration and visualization. In *Proc. International Workshop on Inter-domain Performance and Simulation*, Salzburg, Austria, February 2003.

[8] A. Durand, P. Fasano, I. Guardini, and D. Lento. IPv6 tunnel broker. RFC 3053, January 2001.

[9] B. Fenner, B. Haberman, J. Schoenwalder, and D. Thaler. Management information base for the Internet Protocol (IP). Work in progress.

[10] R. Fink and R. Hinden. 6bone (IPv6 Testing Address Allocation) Phaseout. Work in progress.

[11] R. Gilligan and E. Nordmark. Transition mechanisms for IPv6 hosts and routers. RFC 2893, August 2000.

[12] B. Gleeson, A. Lin, J. Heinanen, G. Armitage, and A. Malis. A Framework for IP Based Virtual Private Networks. RFC 2764, February 2000.

[13] S. Hanks, T. Li, D. Farinacci, and P. Traina. RFC1701: Generic Routing Encapsulation (GRE), October 1994.

[14] S. Kent and R. Atkinson. Security architecture for the Internet Protocol. RFC 2401, November 1998.

[15] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed internet routing convergence. In *ACM SIGCOMM 2000*, September 2001.

[16] J. McCann, S. Deering, and J. Mogul. Path MTU discovery for IP version 6. RFC 1981, August 1996.

[17] V. N. Padmanabhan and L. Subramanian. An investigation of geographic mapping techniques for internet hosts. page 13, 2001.

[18] K. Park and H. Lee. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets. In *Proc. ACM/SIGCOMM '01*, August 2001.

[19] G. Philips, S. Shenker, and H. Tangmunarunkit. Scaling of multicast trees: Comments on the Chuang-Sirbu scaling law. In *ACM SIGCOMM 1999*, August 1999.

[20] D. Provan. RFC1234: Tunneling IPX traffic through IP networks, June 1991.

[21] P. Radoslavov, H. Tangmunarunkit, H. Yu, R. Govindan, S. Shenker, and D. Estrin. On characterizing network topologies and analyzing their impact on protocol design, September 2000.

[22] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for IP traceback. In *Proc. ACM/SIGCOMM '00*, August 2000.

[23] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring ISP topologies with rocketfuel. In *Proc. ACM/SIGCOMM '02*, August 2002.