IP Addresses Considered Harmful

Brian Carpenter
Draft 2013-11-18

This is a personal essay, also a work in progress, and is not a contribution to the IETF at the present time.

Abstract

This essay reviews problems in the Internet caused by, or related to, its addressing model and the way in which distributed applications use that addressing model. The roots of the problems go back to very early design and deployment choices, and affect scenarios such as referrals, multihoming, dual stack operation, multiple interfaces, roaming and renumbering. Numerous partial solutions have been proposed and in some cases partly adopted, but the problem persists and is complicated by the addition of IPv6 to the network. There are several possible future directions and no obvious choice.

Introduction

The alarming statement in the title above is not entirely a new idea. Shoch clearly described the benefits of late binding of names to addresses in 1978 [IEN19]. The point was repeated in 1983 in [RFC882]:

> The basic need is for a consistent name space which will be used for referring to resources. In order to avoid the problems caused by ad hoc encodings, names should not contain addresses, routes, or similar information as part of the name.

The same was implied by the DNS standard, whose first design goal was stated in 1987 to be “a consistent name space which will be used for referring to resources” [RFC1034]. The problems caused by using addresses have been stated in various forms ever since, for example [RFC1900], [RFC1958], [RFC2101], [RFC2775], [RFC4085], [RFC5887], [draft-irtf-nsrg-report], [draft-carpenter-referral-ps], [draft-ubillos-name-based-sockets], [RFC6879], [RFC6866], [draft-baker-happier-eyeballs]. It has also been argued that source addresses in datagrams are unnecessary [SNA].

The superficial message from these, and certainly other, references is this: Internet applications and configuration files should use names, not IP addresses, to initiate the binding of an application to a remote resource. Addresses should only be stored transiently, if at all. Specifically, the name should be a fully qualified domain name (FQDN) in the DNS; failing that, some other name space might apply (such as a Skype user identifier).

To argue from absurdity why this might be the right approach, let us consider the following assertion: “Computer applications should use numeric disk block addresses, instead of file names, to bind themselves to a particular mass storage file.” Binding to an IP address would be just like binding to a disk address. Yet people do it. Why?

What are people getting so worked up about?

A consequence of the original Berkeley socket API released in 1983 is that an application wishing to contact a remote resource on the Internet opens a socket by means of a layer 3

IP address. That means that in order to connect to, say, *host.example.net*, an application first needs to discover at least one IP address for *host.example.net*, store it locally, and use it as a parameter in the API call to open a socket. Moreover, a listener application in *host.example.net*, when it receives an unsolicited incoming packet, only learns the IP address of the host containing the caller; it is not told the name of the calling host. (It could in theory use reverse DNS to find a corresponding FQDN, but this cannot be relied on since by no means all hosts have valid reverse DNS entries, and in many cases the address may be that of a NAT box anyway.) Furthermore, it learns this by a layer violation – the transport layer peeks at layer 3 information in the incoming packet, and it's even used as part of the TCP checksum.

This address-based way of *requesting, starting* and *continuing* communication has consequences. Of course, using IP destination addresses to route the packets in an ongoing mono- or bi-directional communication is inevitable, but that is not our concern. Our concern is what happens when something other than simple packet flow occurs. Significant operational problems arise in several cases as follows.

1. Referrals: A referral is, in the general case, when an application in host A that is communicating with an application in host B passes a reference to B over to an application in a third host C. There is a degenerate case when B = C, and there are indirect referrals where C passes the reference on to D, etc. In traditional multiparty Internet protocol design, the reference is passed as an IP address. There are several reasons why this often fails, basically because there is no such thing as a single universal address space in today's Internet [draft-carpenter-referral-ps]. The address scope is typically chopped up into separate realms, separated by NATs, firewalls, and address families (IPv4 and IPv6). Sometimes, disjoint realms are linked by VPN connections. The Internet has no way to identify, label or take account of these disjoint addressing scopes, yet every address is only meaningful if we know which scope it belongs to. Thus, the traditional referral model of passing an IP address on to a third party is simply broken. Some (but not all) of this problem would be overcome if referral was based on an FQDN, but addresses are widely used because of ancient history, although their scopes are unknowable.

Because of this problem, applications that need referrals either have to invent their own identifier space, build their own rendezvous mechanism, or adopt mechanisms to traverse NATs and other middleboxes. Widespread examples include BitTorrent and Skype.

2. Multihoming failover: Even without referrals, any form of host or site multihoming that involves multiple IP addresses for the same host fails if IP addresses are used to bind transport sessions to applications. If the path using address **α** fails, we need the transport session to switch to address **β**. This problem [RFC3582] [RFC4177] has stymied most attempts to design scalable solutions for multihoming. As a result, multihomed sites prefer to have a single provider-independent address prefix that needs its own BGP4 routing entry. Therefore the Internet is at some long-term risk of a blow-out of the wide area routing table to at least several million entries [RFC4984].

3. Dual stack operation: In a way this is the same as the previous case, except that **α** is an IPv4 address and **β** is an IPv6 address. Difficulties will arise if a transport session or a multiparty application needs to switch to the other version of IP for some reason.

4. Multiple Interfaces: A host, especially but not exclusively a mobile host, may have at least two Internet connections via two different interfaces and probably two different providers – for example 3G/4G/LTE and WiFi – with different latency, throughput, battery impact, and cost [RFC6418]. Like the two previous cases, there will be multiple addresses, and in addition applications might want to choose one interface or another for reasons such

as price/performance tradeoff or battery conservation. Analysis in the IETF has shown that a key part of these scenarios is the concept of a *provisioning domain* – defined as "a consistent set of network configuration information" normally derived from the service provider behind a given interface [draft-anipko-mif-mpvd-arch]. Different provisioning domains may provide better, worse or non-existent routes to specific addresses, and may even provide different views of the DNS.

5. Roaming: If a host is roaming while connected, it may experience an unpredictable change of address or even of address family. It may even move from being behind a NAT to having a global IP address, or vice versa. Thus, the impact may be similar to either or both of the above, with or without NAT traversal.

6. Renumbering: If a site is renumbered, transport sessions in progress will fail – essentially the same situation as multhoming failover. Although rare, site renumbering is considered very important to ensure future scaling of the wide-area routing system [RFC5887, RFC6879].

Various approaches to tackling this set of problems partially or completely have been proposed, including:

- RSIP (realm-specific IP) [RFC3102, RFC3103]. Years old, unused.

- HIP (host identity payload) [RFC4423, RFC5201, etc.] Years old, hardly used.

- STUN/TURN/ICE [RFC3489, RFC5766, RFC5768]. Aimed at NAT traversal for SIP (but why was SIP designed with so much dependency on address transparency at a time when NATs were already prevalent?)

- PCP (port control protocol). Also aimed at NAT traversal [RFC 6887].

- SHIM6 [RFC5533, RFC5534, RFC5535, RFC6316, RFC6629]. In SHIM6, a shim on top of the network layer and below the transport layer swaps alternative IPv6 addresses between the two ends of a session, so that a broken session can be automatically restored by picking a new address pair. Host stacks are affected (but not TCP or the socket API), and traffic engineering is impacted. SHIM6 relies on IPv6 extension headers, which many firewalls discard.

- LISP [RFC6830 to RFC6837]. In LISP, packets sent between two prefixes that are not present in the global BGP4 table are encapsulated to traverse the wide-area network. This relies on a distributed mapping system to map between the true destination and the decapsulator, and on a transition mechanism to interface the LISP and non-LISP Internets. On the other hand, site infrastructure and host software are not affected.

- ILNP [RFC6740 to RFC6748]. In ILNP, there is a conceptual boundary between the routing part of an IPv6 address (64 bits) and the locally-significant identifying part (also 64 bits). The routing part is rewritten at the site boundary. Host code, including transport code, and on-site routers are affected.

- Just as network address and port translation (NAT) has become widespread for IPv4 enterprise networks wishing to be internally indpendent of ISP-based addressing, network prefix translation has been proposed, and implemented, for IPv6 networks [RFC6296]. It has some, but not all, the disadvantages of traditional NAT.

- Happy Eyeballs [RFC6555]. In this technique, applications use explicit probing to discover the “best” IP address to use when there is a choice, but only before establishing a transport connection. Application code using the socket API must be updated.

- Multipath TCP (MPTCP) [RFC6824]. Here, TCP itself is modified to operate several paths (i.e. several pairs of IP addresses) simultaneously, with automatic load sharing between the paths. In some ways this is a compromise between SHIM6 and Happy Eyeballs, but is only effective for TCP traffic, as well as requiring host stack modifications.

- Name Based Sockets (NBS) [draft-ubillos-name-based-sockets, NBS1, NBS2]. In NBS, applications open sockets by name and the transport session, initially opened by address, swaps DNS names between the two ends, so that a broken session can be automatically reopened by name. It relies on IPv6 extension headers or IPv4 options, which many firewalls discard. NBS requires retooling of the DNS, not just of the hosts using it, but should be transparent to routers and middleboxes. At this time it remains a prototype.

- Ongoing research projects such as [Signposts] and Polyversal TCP [PolyTCP]. A more ambitious project is Named Data Networking [NDN]. This attempts to change the conversation by viewing individual data resources as the source and destination of packets, demoting both IP addresses and DNS names from their current primacy.

- In a sense, the entire work of the IETF MIF WG is related to this, for example [draft-ietf-mif-api-extension], [draft-deng-mif-api-session-continuity-guide].

Why wasn’t this fixed years ago?

It’s easy to see that we have a problem, and why there have been numerous attempts to find a solution. Some of these solutions are compared in some detail in [Naderi], which shows that none of them is perfect. However, we should consider *why* the problem persists even after the series of exhortations to use names instead of addresses that were cited at the beginning of this article, and the numerous attempts at (partial) solutions. The answer is probably the simplicity and generality of the Berkeley socket API, which was released with 4.2BSD Unix in early 1983. Dynamic translation of names into addresses was proposed by Jon Postel in 1978 [IEN61] and the general concept of the DNS was described in 1982 [RFC819, RFC830], but it was still some months after the 4.2BSD release before the first stable specification of the DNS was published [RFC882,RFC883], and several years before the DNS was widely deployed, fully standardised [RFC1034, RFC1035], reasonably complete, and reliable.

It’s also worth noting that the DNS added a third party to every two-party communication, in order to automate name-to-address (or identifier-to-locator) translation. This third party needs to be reliable, trustworthy, fast, constantly updated, and remarkably scalable. These requirements apply to the DNS or to any other form of identifier-to-locator mapping system.

Back in 1983, there were fewer than 1000 hosts on the Internet, all with manually assigned and static addresses. There was only one address family and there was no NAT. All six of the above contingencies, even site renumbering, would have seemed highly unlikely. Thus, a three stage process of opening a socket – first, convert the remote host’s name to an address (originally by looking it up in `/etc/hosts`), second, store the resulting address, and third, use the stored address to bind a socket to the remote host, would have been the natural choice. If the address lookup failed or went slowly, the application failed or went slowly. It is not surprising that programmers chose to resolve DNS names only when they had to, and stored addresses whenever they could. Here we are thirty years later, and that’s still what C programmers do, with a POSIX twist and a little extra logic to cover the IPv4/IPv6 choice.

This is obviously wrong. At first sight, NBS seems right. The typical application programmer expects open/send/receive/close to "just work". To some extent, this is what Java already provides. The Java programmer opens a connection by DNS name, not by address, and with limited control over the details. Somebody else is supposed to take care of the hard stuff (multiple addresses and interfaces, failover, latency, throughput, battery life, bandwidth caps, cost per minute) but clearly Java, TCP (and still less UDP) do none of this. To a considerable extent, Java is an existence proof for NBS, and also shows that it is no panacea. If a Java connection fails to start, the user can't tell whether there is a DNS hangup or a TCP/IP hangup, and doesn't know how to recover.

By comparision, however, the alternatives are worse. LISP and ILNP require radical retooling of network elements, but still need a mapping system. (LISP adds a new mapping system, and ILNP re-uses DNS.) SHIM6 or MPTCP will only take care of some aspects, and Happy Eyeballs won't take care of anything that goes wrong after a TCP connection has been established. Translation hurts Internet transparency, MPTCP (or Polyversal TCP) only takes care of TCP cases, and Signposts is a tailored solution for certain scenarios.

The most ambitious approach is NDN. More so than NBS or Signposts, it sweeps the problems discussed above under the carpet by adding a name-based architecture on top of the existing Internet. It also calls into question an underlying assumption of the Internet addressing model: that the addressable entity (whether by locator or by identifier) is a "host." In reality, since the early days, the concept of an Internet (or ARPANET) host has become fuzzy. Firstly, addresses are generally taken to refer to a specific interface on some box, not to the box as a whole. Secondly, today millions of apparent hosts (or interfaces) are virtual, in fact being hosted by some other host. Thirdly, some apparent hosts are in fact load balancers standing in front of an array of actual hosts (whether virtual or real). Fourthly, many names that appear to be translated into host addresses by the DNS are in fact bogus: what they really translate into are the addresses of proxies or caches, and the translation varies according to the topological location of the resolver. Another form of bogus name is a name synthesised purely for the purposes of Reverse DNS, because some applications operate on the assumption that an address without a Reverse DNS entry is itself bogus. In this case the bogus name frequently refers to a NAT box rather than to a real host. NDN's premise seems to be that all of this is irrelevant to the real purpose of getting a packet from a named source object to a named destination object, and we should focus on those objects.

Meanwhile, there was some distraction...

Another reason this issue has not been fixed is that a great deal of effort (variously intellectual, engineering, and marketing) has gone into IPv6 since 1994 or thereabouts. IPv6 has proved immensely harder to deploy than its progenitors expected. Partly this was because of the amazing growth rate of the IPv4 Internet from 1995 onwards, which put the focus of attention elsewhere. Partly it was because IPv6, although not conceptually radical, is more than just IPv4 with bigger addresses – so it was perceived as a complex step for vendors and operators already dealing with the growth rate. Partly it was because there is an intrinsic difficulty caused by the need for IPv4-only nodes (which know nothing of IPv6 addresses) to interoperate with nodes that, even if they have their own IPv6 addresses, cannot get unique IPv4 addresses. In any case, the result was that for almost twenty years, thinking about changes in the network layer has concentrated on IPv6.

Where next?

Why hasn't NBS caught on? It turns out to be pretty complicated and not self-contained in the hosts concerned. As IPv6 has shown, that makes deployment a very major, earthquake-like, endeavour.

Will SHIM6, LISP or ILNP catch on? There are reasons for pessimism. SHIM6 is currently undeployable because many firewalls block the necessary extension headers. LISP requires retooling the Internet wide-area routing system, and has a significant bottleneck in its method of interworking with the non-LISP Internet. ILNP requires retooling of both host network stacks and site routers.

Will MPTCP catch on? It "only" requires both hosts to have updated stacks, and an MPTCP host can speak plain TCP if the other host is not MPTCP-aware. This has a chance of catching on – but it only fixes a subset of the above issues, because it assumes the host already knows a useful set of address pairs and it only works for TCP applications. Similar remarks apply to other TCP "retreads" such as Polyversal TCP.

Will Happy Eyeballs catch on? Well yes, it has done, for the specific case of choosing IPv6 vs IPv4, in specific browsers that happen to include the necessary code. This fixes an even smaller subset of the above issues, because it does not respond dynamically to changes in connectivity. Extensions and variants of this approach are already appearing.

Will solutions like Signpost catch on? This author's prediction: only for a niche market, where customers care enough about security and reliability to invest in such an approach. This is related to the way in which IPsec has only really caught on for corporate "dial home" VPNs and the like, rather than becoming ubiquitous.

Will translation catch on? Sadly, it has done for IPv4, and NPTv6 is attracting attention for IPv6.

What else? All major distributed applications, especially the multiparty ones such as BitTorrent and Skype, have been forced to solve some or all of these problems. Essentially they do so by inventing their own globally unique identifier space (Skype ID and the like) and their own rendez-vous mechanism that lives in the global part of the Internet ("outside" all proxies, firewalls and NATs). This indeed takes the harm out of IP addresses by not relying on them except in a transitory way – if something goes wrong, a multiparty application shrugs its shoulders and repeats its rendezvous process via unique identifiers again.

Another angle of attack is the notion of a connection manager. This is a piece of magic that is more pro-active than the Happy Eyeballs approach (which is entirely reactive). A connection manager decides how to maintain connectivity as the network environment changes around it. There are numerous proprietary connection managers, often bundled with corporate VPN solutions, where the motivation is to set up tunnels and host routes such that the user gets access to both the Internet and the corporate network while roaming arbitrarily. One step up from a connection manager is a congestion manager [RFC 3124] but that is double-ended whereas a connection manager is typically single-ended.

What does the Internet need? It goes without saying that so-called "clean slate" approaches, in which radical basic changes are made, are excluded from serious consideration, except perhaps as thinking aids. Today's infrastructure, operating systems and applications are not going to be replaced overnight. In this sense, the NDN approach is more hopeful, because although architecturally radical, it is deployed as an overlay on the existing network.

There's no point in doing anything that is not deployable. Only incremental change is physically possible. In this writer's opinion, that creates difficulty for solutions requiring *simultaneous* actions by independent parties, i.e. any kind of double-ended solution that doesn't automatically fall back to a single-ended mode, or any solution that requires a host to be aware of a specific middlebox. Also any solution that requires firewall transparency has a poor chance of deployment. Solutions that can be installed one host at a time, without changing firewall policies, and don't need new middleboxes, seem right.

There could be partial solutions that break these deployability guidelines, for cases where there is strong motivation – the proof of concept for that is the corporate VPN, where the enterprise has to deploy a VPN end-point and the users have to install a VPN client. This works because there is a pre-existing link between the two parties and a strong motivation to get connected. Signpost is similar, but this will not work for the general case of Joe Random Citizen connecting to some new server or service. NDN seems to have adequate generality but it does require new infrastructure (a name-based routing system, depending on a systematic naming mechanism) .

There are several possible approaches:

1. Do nothing. Applications will have to continue doing what they do today – use DNS and their own form of identifier (hopefully authenticated) to rendez-vous at a global IP address, but treat IP addresses obtained during the rendez-vous process as transitory. When a connection fails, repeat. The basic interface between apps and the network remains Berkley sockets, possibly lightly concealed by Java etc. Issues such as multihoming or interface choice will be resolved by proprietary connection managers via rules of thumb. Translation will not go away, and various point solutions will emerge.

2. Design a superSocket API, but don't specify the engine behind it. Let the implementors sort it out, and may the best solution win.

3. Specify requirements for a generic name-driven connection manager (to go with #2). The IETF HOMENET and MIF efforts may go in this direction, but that only covers a subset of the network.

4. Design a one-size-fits-all solution: more practical than NBS, more general than MPTCP or Happy Eyeballs, with knobs to set policy for interface choice etc. But if this needs a generic rendez-vous server of some kind (superICE), it hits the deployability barrier.

5. Overlay a new architecture; NDN is an example. As noted above, this could indeed hide the problems discussed above, but it radically affects host stacks and requires an elaborate new routing system.

Any solution has implementation aspects as well as external aspects. Where does the new intelligence sit in the host system? Kernel space or user space? In the VM or the hypervisor (on the assumption that in future, virtual hosts will greatly outnumber physical hosts)? What is the structure (library, process, thread)? What is the relationship to the DNS resolver, and how does DNSSEC change things? How will middleboxes such as proxies, caches, content distribution mechanisms, load balancers, and firewalls deal with the solution?

This essay doesn't attempt to answer these questions or predict the future.

Acknowledgments

Useful comments and hints have come from Jon Crowcroft, John Klensin, Anil Madhavapeddy, Vsevolod Stakhov, Lixia Zhang, and others, who share no responsibility for the resulting text.

References

RFCs at http://tools.ietf.org/html/rfcXXXX
Drafts at http://tools.ietf.org/html/draft-YYYY

[IEN19] J.F. Shoch, *A note on Inter-Network Naming, Addressing, and Routing*, IEN #19, Xerox PARC, January 1978, http://www.rfc-editor.org/ien/ien19.txt .

[IEN61] J. Postel, *Internet Name Server*, IEN #61, ISI, October 1978, http://www.rfc-editor.org/ien/ien61.txt .

[Naderi] H. Naderi, B.E. Carpenter, *A Review of IPv6 Multihoming Solutions*, Tenth International Conference on Networks (ICN 2011), St. Maarten, (January 2011) 145-150, http://www.thinkmind.org/index.php?view=article&articleid=icn_2011_7_30_10340 .

[NBS1] Christian Vogt, *Simplifying Internet Applications Development With A Name-Oriented Sockets Interface*, unpublished paper, 2009.

[NBS2] Zhongxing Ming, Javier Ubillos, Mingwei Xu, *Name-based Shim6: A name-based approach to host mobility*, AsiaFI 2011 Summer School

[NDN] Named Data Networking project, http://named-data.net/

[PolyTCP] Zubair Nabi, Toby Moncaster, Anil Madhavapeddy, Steven Hand, Jon Crowcroft, *Evolving TCP. How hard can it be?*, CoNEXT Student'12, Nice, France , December 10, 2012

[Signposts] Charalampos Rotsos, Heidi Howard, David Sheets, Richard Mortier, Anil Madhavapeddy, Amir Chaudhry, Jon Crowcroft*, Lost In the Edge: Finding Your Way With DNSSEC Signposts*, 3rd USENIX Workshop on Free and Open Communications on the Internet (FOCI '13), Washington D.C., 13 August 2013

[SNA] J. Crowcroft, *SNA: Sourceless Network Architecture*, private communication, 2008 (also see http://www.cl.cam.ac.uk/~jac22/talks/sna.ppt, presented at *Perspectives Workshop: End-to-End Protocols for the Future Internet*, Schloss Dagstuhl, 2008)