RN/05/28
15 October 2005

Research
Note

Naming in the Internet Architecture: Provenance and Current Issues

R. Atkinson

Telephone: +44 (20)7679-7214
Fax: +44 (0)20 7679 1397
Electronic Mail: R.Atkinson@cs.ucl.ac.uk
URL: http://www.cs.ucl.ac.uk/

Abstract

This paper discusses the history and current issues with naming in the Internet. An overview of namespaces of the current Internet standards is provided. The history of those namespace is briefly outlined. Further, the existing namespaces are contrasted with other design choices, primarily the original ARPAnet and the ISO OSI model. Finally, current limitations and issues relating to Internet naming are identified.

Department of Computer Science
University College London
Gower Street
London WC1E 6BT, UK

1 ARPAnet Origins

In the late 1960's, the US Defense Department's Advanced Research Projects Agency (ARPA) sponsored research into packet networking. Up until that time, the focus in networking research and development had been almost exclusively on circuit-switched networking, such as the traditional telphone system uses. At the time, many doubted that packet networking would be very useful or have much success. Subsequently, those doubts have been put to rest; packet networking is now starting to displace traditional circuit switched networking even in the telephone system.

1.1 Early Namespaces

Probably the most fundamental namespace is the address used by the networking protocol. Two common examples of an address are an Ethernet MAC Address, at the link-layer, or an IP address, at the network-layer. In the early days of the ARPAnet, the Network Control Protocol (NCP) address would have been the network-layer address. Initially, applications and other protocols used the NCP address for nearly everything, mainly because it was essentially the only namespace extant at the time. In the early ARPAnet, there were three components to an address, the network, the host, and the socket. In this case, the socket distinguished one application from another on a given host. In more modern terminology, the network is equivalent to the IP routing prefix, the host is equivalent to the host portion of the IP address, and the socket is equivalent to the modern tuple of transport-layer protocol plus port number.

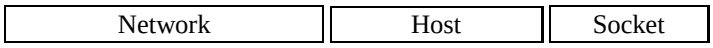

Figure 1: Early Addressing Example

Before too long, there was a strong desire for a more friendly form of name and the flat host namespace came into being.[Kar71] This initial host namespace lacked the heirarchy of the modern hostnaming system and name-to-address mappings were always performed locally, referencing a manually-updated copy of the HOSTS.TXT file.[Deu73] This did have the strong advantage of a much more user-friendly format than a numeric address. Most applications continued to use network-layer addresses internally and within the application-layer or transport-layer protocols, however the user interface could usually accept either a textual hostname or a numeric address.

Electronic mail was an early application of the ARPAnet. The mailbox name, consisting of *userhost* was well established by 1973.[BPTW73] Application programming in this era also relied extensively on the raw network-layer, at the time Network Control Protocol (NCP), address and the port number [1] identifier. The absence of a more rich set of namespaces left application programmers no other namespace choices in the early days of the ARPAnet. The practice of well-known port numbers for certain widely deployed services was developed during the early 1970s. [CP72]

In 1978, Shoch provided one of the earliest discussions of namespaces.[Sho78] In that paper, he defined the *name* as defining the resource that is sought, the *address* as defining the location of the resource, and the *route* as defining how to reach that resource. He noted that normally some mapping function was required to take a name and determine the address and to take an address and determine the route. He also noted that a single object might have more than one name, hence more than one address, and hence more than one route. He also raised the matter of how naming architecture impacts the ability to migrate processes among systems within a distributed computing system.

1.2 Application Impacts

In this early era, the File Transfer Protocol (FTP) [Bhu71] and the Telnet protocol [MW71] were already important applications. Those were probably the first two significant applications in packet networking. Because they were created very early on, each protocol was designed to primarily work with the address, specifically the combination of network, host, and socket, rather than some human-readable hostname or other identifier. Also, since other identifier options did not exist at this point, those addresses were embedded in the over-the-wire protocol and various other places in the system, in the application, or in the application's user interface. This also meant that the networking application programming interfaces (APIs) of this era primarily used the address as the identifier. Name to address translation was performed inside the application (or in a library called by the application) and then the address (not the hostname) was provided to the Networking API. To this day, FTP and Telnet still use raw network-layer addresses in numerous places both within the protocol and within the application itself.

[1] In the earliest years, what is now called a *port number* was apparently called a *socket number.* For clarity, this text uses the modern terms throughout.

2 Research Internet

Circa 1983, the NCP-based ARPAnet was replaced by the IP-based Internet.[Pos81] While the commonly used FTP and Telnet applications survived across that transition, there were major changes in the network architecture. The layering model used today in the global Internet was first used for the IP-based Internet. The transition from NCP to IP used a single globally applicable conversion date (”flag day”), where each site reconfigured their routers and hosts on the same date.

In the new architectural model, the transport-layer became distinct from the network-layer. However, the new architectural model did not immediately include significant additions to the namespaces that were supported. So the prior practice of designing application protocols to use, rely upon, and embed network-layer address information continued.

2.1 Internet Addresses

With the migration to IP, the network-layer address became a 32-bit unsigned integer. Larger sized addresses were considered as part of the design of the Internet Protocol (IP) and were rejected due to concerns about the size of the IP header.[Cla82] During this early period, IP addresses were classed. This meant that the value of the first octet in the address defined the network address mask in use. There were only 3 different network masks in use; addresses with a first octet of value 1 through 127 (decimal) were called Class A and had a network mask of 8 bits. Addresses with a first octet value of 128 through 191 were called Class B and had a network mask of 16 bits; finally addresses with a first octet value of 192 through 223 were called Class C and had a network mask of 24 bits. Classes D and E were defined, but not used, during this era. Class D was subsequently allocated for use by IP multicasting.[Dee89]. Class E remains reserved to this day.

Figure 2: Example of IPv4 address

2.2 Hostnames

During this era, the HOSTS.TXT system used in the ARPAnet continued in use, modified to handle IP addresses instead of NCP addresses. In 1981, a proposal was made to transform the flat hostname of the early ARPAnet into a heirarchical namespace.[Mil81, SP82]. In November 1983, the HOSTS.TXT system was enhanced to support such heirarchical naming.[Pos83] While there were several advantages to heirarchical textual names, one important early application was in distinguishing different electronic mail routing domains. In the initial deployment of heirarchical naming, the original ARPAnet hostname would be given ”.ARPA” as a suffix indicating that the host was connected to the ARPAnet. Alternatively, hosts connected via the UNIX to UNIX Copy (UUCP) program were given ”.UUCP” as a suffix and hosts connected via the IBM networking protocols of the BITNET network were given ”.BITNET” as a suffix. The addition of heirarchy to the HOSTS.TXT system was essential in paving the way for the later transition to the Domain Name System (DNS).

2.3 Service Names

In this era, services, such as Telnet, were distinguished by the combination of transport-layer protocol (e.g. TCP) and one or more application port numbers (e.g. 23). In effect, the combination of an address, a specific transport-layer protocol, and a transport-layer port number acted as a Service Identifier or Service Name. In modern parlance, this triplet defines a ”socket”, so called because the BSD Sockets API used the term ”socket”. The BSD Sockets API is by far the most widely used networking API, which has been true since shortly after the release of 4.2 BSD UNIX in August 1983. [LMKQ89]

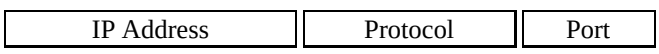

Figure 3: Example of a Modern Socket

2.4 Programming Considerations

Beginning shortly after the release of 4.2 BSD UNIX, the BSD Sockets Application Programming Interface (Sockets API) became the dominant interface for use in developing network-aware applications. While having a widely avail-

able interface greatly facilitated growth of the deployed Internet, the Sockets API arguably used the wrong objects in its interfaces.

The BSD Sockets API was first deployed circa 1981, prior to the invention and deployment of the Domain Name System (DNS) protocol. As a consequence, the Sockets API required all networked applications to be aware of, manipulate, and store lower-layer information, such as the network address itself, rather than using (at the time non-existent) more abstract higher-layer names. This design inadvertantly trained programmers to use the lower-layer objects within the application source code – and often also in the over-the-wire network protocol. The practice of embedding the lower-layer IP address within the application-layer protocol was quite common as one result of this. The BSD Sockets API has long included a library call that converts an IP service name, for example *telnet*, into the corresponding transport-layer protocol and port number, for example *TCP port 23*. Many network programmers do not use this library call, however. Further, the translation table for this service to protocol and port number function lives as a flat text file called */etc/services*.

Dunlap and Bloom had implemented the Berkeley Internet Naming Daemon (BIND) for 4.3 BSD by early 1986.[DB86] Mockapetris reports that 1985 was the first year that some hosts exclusively used the DNS protocol for name resolution and indicates that significant deployment had occurred by 1987.[MD88] As the DNS was implemented, additional library calls were created to permit the application to request translation of a domain-name into an IP address, which could then be provided to the BSD Sockets API. An alternative design, never implemented so far as the author knows, would have been to alter the BSD Sockets API to use a domain-name instead of the IP Address. Also, the initial deployment of the Domain Name System protocol on BSD Unix used the Berkeley Internet Naming Daemon (BIND), which was an application-layer service rather than a UNIX kernel service. While this did greatly facilitate deployment, it meant that the operating system kernel could not easily use the more generic fully-qualified domain name as an identifier and encouraged all parts of the kernel to use the IP address in its place.

In the late 1980s, AT&T led an effort to develop a more elegant and more appropriately abstracted networking API. This was known initially as the Transport-Layer Interface (TLI) or later, after adoption by the X/Open Consortium, as the X/Open Transport Interface (XTI). This interface was required by Issue 4 of the UNIX System V Interface Definition (SVID Issue 4), which corresponded to UNIX System V Release 4. Many who used it considered this programming interface to be easier to use and more elegant than the BSD Sockets API. However, the Sockets API was more widely available on deployed Internet hosts, because it had shipped earlier on. This meant that there were many programmers familiar only with the BSD Sockets API and that the arguably better TLI/XTI interface was never very widely used. As the BSD Sockets API required the programmer to directly manipulate and use the lower-layer information, most networked applications did not use higher-layer abstractions internally or externally.

As long as all IP addresses were globally routable, firewalls were not widely deployed, and network address translation was not in use, directly using the IP address within the transport-layer and within applications was an issue primarily for mobile hosts. Mobile hosts were uncommon in the 1980s and early 1990s, but are quite common now. Laptop computers with IEEE 802.11 wireless Ethernet interfaces are very commonly used today. Since firewalls became more widely deployed and particularly as network address translation became more widely deployed, the embedding of lower-layer information, such as the IP address, in transport-layer and application-layer protocols has been much more widely understood to be a problem. Some few however, continue not to understand that proper layering is important and consider the root cause to be a shortage of IP addresses (sic) rather than embedded lower-layer information in upper-layer protocols and applications that ought not be using that identifier.

2.5 Resulting Issues

The result of all of this has been that the IP address is commonly used in applications, in application protocols, and in transport protocols as the generic identifier for a given host. This is a curious outcome since the formal definition of an IP address is that it names a particular interface on a host, rather than naming the host itself, and because the real purpose of an IP address to for packet routing, not for host identification.[Sho78] This paper will argue that such uses are largely inappropriate and that the Internet Architecture is deficient in not providing a sufficiently rich set of namespaces. If a sufficiently rich set of namespaces existed, then NAT and mobility might well be non-issues, because the upper-layers would have used appropriate abstractions isolating the upper-layers from the details of the Internet Protocol and the IP Address that is used for routing packets from place to place.

3 ISO OSI Networking

This section discusses namespaces and identifiers used in the ISO OSI protocol suite. Although this suite never achieved the large-scale deployment that the Internet protocol suite achieved, nonetheless it was the focus of a great

deal of research and development effort in the 1980s and early 1990s. In some respects, the OSI protocol suite had a more rich set of standard namespaces, benefiting from the experience of the original ARPAnet.

3.1 Addresses

The ISO work on the Open Standards for Interconnection (OSI) networking protocol suite started in the 1980s and ceased being relevant to commercial networking in the mid-1990s when the US and several other governments abandoned their unsuccessful attempts to mandate migration from the Internet networking specifications to the OSI specifications. OSI failed to provide advantages sufficiently compelling to justify the economic costs of abandoning the larger installed base of TCP/IP-based networks and hosts.

Another issue with the OSI standards work was that two unrelated networking standards were being developed under the same umbrella. The OSI Connection-Less Network Protocol (CLNP) specified a datagram service and was very similar to (and arguably derived directly from) the Internet Protocol. However, many in the OSI community had a strong telephony background and insisted that circuit-oriented networking was the proper direction for OSI. Rather than decide between the two, the OSI community standardised both CLNP and a Connection-Oriented Networking Protocol. The Connection-Oriented Network Protocol is better known as X.25, which was the name of the applicable ITU-T standard that was endorsed by OSI for circuit-switched networking.

In turn, the decision to define both CLNP and X.24 as OSI standards at the network-layer meant that network-layer address formats needed to support both packet-network usage and circuit-switched network usage. So there were numerous syntaxes used for OSI addresses. Further, OSI addresses were defined to be variable-length, rather than using a fixed-length address as the Internet Protocol does. Chiappa has argued that addresses should be variable-length in a global packet network.[Chi99]

An OSI NSAP address is variable-length and there are numerous different formats, unlike an IPv4 address, which has a single format and a fixed 32-bit length, or IPv6, which has a fixed 128-bit length but several different formats.

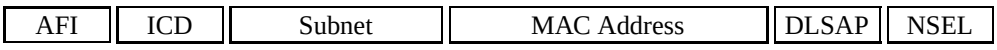

Figure 4: Example of OSI NSAP for use with CLNP

3.2 Service Names

The ISO OSI Reference Model defined 7 layers in the protocol architecture, rather than the 5 layers defined in the Internet reference model. While some viewed this as unwieldy, it did facilitate thinking within the OSI community about what sorts of names or identifiers ought to be used at which layer of the protocol model.

In particular, OSI defined transport-layer identifiers and also service names, whereas the Internet protocol architecture did not really define either of those, instead choosing to embed the IP address in various upper-layer protocols of the architecture. Some might argue that these additional namespaces represent additional complexity, but others might well observe that adding those additional namespaces provided additional decoupling and layers of indirection that provide better modularity to the networking architecture overall. In any event, the designers of the OSI architecture were able to take positive and negative experience with the Internet architecture into consideration as they designed their own. By contrast, the original architects of the Internet were pioneering packet networking and hence were not able to leverage much, if any, prior work in packet networking for their own designs.

3.3 Directory Service

As part of the OSI protocol specifications, a general-purpose directory service was defined in ITU-T X.500. This was deliberately designed to be an all-purpose general directory service, unlike the Internet's Domain Name System (DNS), which was really intended to be a limited service primarily used for hostname to address lookups.

A consequence of the choice that X.500 would have wide scope and very general applicability, the OSI directory service was much more complex than the DNS. While this impeded the deployment of X.500, a slightly simplified version of X.500 is in use today, but has been renamed Lightweight Directory Access Protocol (LDAP).[YHK93]

In the OSI concept, a user at a workstation might decide to use some network service, for example the Virtual Terminal service. He would then ask his workstation to open that service between the local workstation and some other named computing system. His workstation would then query the OSI directory for the correct transport-layer identifier for the remote end and open a session to that remote transport-layer and hence to the remote instance of the

Virtual Terminal service. In the course of this, the transport-layer protocol on his own workstation would translate the transport-layer identifier for the remote end into the corresponding network-layer address for the remote end and ask the network-layer protocol (most usually CLNP) to open a session with that remote end at the network-layer. At least in theory, the transport-layer identifier of a host could be decoupled from the network-layer identifier of the same host.

By contrast, the Internet does not permit such decoupling and the DNS does not have any concept of a transport-layer identifier for hosts listed in the DNS. For example, the Transmission Control Protocol (TCP) includes several elements of the IP header in the TCP checksum calculation. So the TCP state information of all existing TCP sessions is adversely affected whenever the IP address below might change. With OSI, the decoupling could, in theory, permit the CLNP address below to change without necessarily forcing the transport-layer protocol state information to change.

4 Link Layer Addressing

Link-layers having a broadcast or multicast capability require link-layer addresses to ensure that data reach their intended destination(s). Point to Point interfaces generally do not need a link-layer address. OSI CLNP and IPv6 each embed certain very common link-layer addresses into their network-layer addresses, at least for the most widely deployed address syntaxes of each network-layer protocol. So it is worthwhile to briefly describe the most common link-layer address types. that might be embedded in a network-layer address.

During the middle and late 1980s, several different LAN technologies vied for supremacy at the link-layer. Among these were Ethernet, defined by IEEE 802.3, Token Bus, defined by IEEE 802.4, and Token Ring, defined by IEEE 802.5. In the late 1980s, the American National Standards Institute approved the Fibre Distributed Data Interface (FDDI) standard. By the turn of the century, Ethernet was much more widely deployed than the sum of all of these other LAN technologies. Each of these LAN technologies used the 48-bit link-layer address originally defined by the IEEE 802 standard. An IEEE 802 address does not contain any topology information and normally is globally unique. An IEEE 802 address might be only locally unique, in which case the Local Use bit in the address is supposed to be set. All link-layer implementations that use the IEEE 802 address permanently store a globally-unique format link address into the interface hardware.

In the late 1990s, the IEEE 1394 standard defined the FireWire technology, which can be used as a link-layer. FireWire uses a 64-bit link-layer address, which is an extension of the 48-bit IEEE 802 MAC address space. Around the turn of the century, the IEEE 802.17 standards committee began work on a new LAN technology. The draft of this standard also specifies the use of IEEE 802 MAC addresses.

SONET has a 4-bit link-layer address; ATM uses a variant of the OSI NSAP Address, so is not easily embedded in a network-layer address; ISDN uses a telephony-format address. Neither SONET, ATM, nor ISDN addresses are commonly embedded in a network-layer address, in part because connectionless network protocols generally use those technologies to provide a reliable point-to-point circuit.

5 Commercial Internet

In the early 1990s, the Internet made the transition from being a small set of research and academic networks to the global commercial network of today. By 1995 it was clear that the Internet was not going to be replaced by OSI networking and that the Internet was going to grow wildly to become a major global communications system.

During this period it became clear to people in the Internet routing community that the original class-oriented IP address scheme was causing the inter-domain routing table size to grow more quickly than was desirable. To reduce the rate of inter-domain routing table growth, class-less inter-domain routing (CIDR) was deployed. With CIDR, each network prefix had an associated network mask – and that network mask could not be inferred from the first byte of the IP address. This succeeded in reducing the growth rate of the inter-domain routing table, at the modest cost of introducing the network mask as a new identifier within the Internet architecture.

Also during this period, a common misconception arose that there was or soon would be a shortage of IPv4 addresses. In order to simplify address management within an organisation, to reduce the consumption rate of IPv4 addresses, and to improve the perceived security of internal networks, many organisations started to deploy private addressing within the organisation. As part of this deployment, a Network Address Translation (NAT) device would be deployed at the border between the internal network and the global Internet. The NAT would then mechanically translate addresses in all packets crossing through the NAT. Also, in order to address the perceived shortage of IPv4 addresses, the Internet Engineering Task Force (IETF) began work on IP version 6 (IPv6), which used 128-bit addresses, but was otherwise nearly identical to IPv4.

5.1 Domain Name System

The Domain Name System (DNS) protocol was first widely deployed in the latter 1980s. Invented by Paul Mockapetris earlier in that decade, the DNS was primarily intended to provide domain-name to IP address translation.[Moc83a, Moc83b, MD88] Over time, other limited directory functions were added in to the DNS. Some would argue that those additions often more nearly resembled mutation of the original design, rather than proper evolution of the original design.

An unfortunate early policy decision was to create global top-level domains initially (e.g. .ARPA), later followed by the addition of country-code top-level domains (e.g. .UK, .US) in addition. In retrospect, the scalability of the DNS, and the ability of the DNS to accomodate the realities of nation-specific laws and regulations, would have been greatly enhanced by omitting the global top-level domains and instead only using country-code top-level domains. Such a change might also have obviated or significantly reduced the current international tension over management of top-level domains.

The DNS can provide both forward (e.g. name to address) and reverse (e.g. address to name) translation. It also can be used to locate hosts providing particular services to specified domains, for example the MX record can be used to locate a mail exchanger, the KX record can be used to locate a key exchanger, and the SRV record can be used for locating some other kinds of networked services.

The original DNS implementation was in a privileged application running in user-space on a UNIX operating system. [DB86] It was not implemented as a network service inside the UNIX kernel, as TCP and IP were. So the BSD Sockets API, which has always been a kernel interface, could not directly replace the IP address data element with the new domain-name in any straight forward manner. The designers of 4.3 BSD Unix claim that this was an intentional design choice.[LMKQ89]

5.2 Addresses

There are several kinds of addresses used in networking today. At the link layer one finds Ethernet MAC addresses, which are 48-bit opaque non-heirarchical bit strings. Token Ring, Token Bus, and FDDI link layers share this MAC address space, which makes it possible to build an inter-technology bridge at layer-2. From a packet networking perspective, technologies such as X.25 and Asynchronous Transfer Mode (ATM) provide link-layer services. So one might consider an X.25 or ATM address to be a link-layer address within the packet networking context.

At the network layer, the IPv4 address, which is a 32-bit heirarchical bit string, predominates, though one still finds some use of the OSI CLNP protocol and its variable length address. CLNP usually embeds the corresponding link-layer address in the low-order bits of the CLNP address. One also finds some deployment of the IPv6 address, which is a 128-bit heirarchical bit string. In a typical IPv6 address, the high-order 64-bits specifies a unique IP subnet, while the low-order 64 bits specify a unique host on that subnet. As with OSI CLNP addresses, it is common for the low-order portion of an IPv6 address to contain a link-layer address. In the case of IPv6, this embedded link-layer address would most commonly be formed from a 48-bit Ethernet MAC address, though it might be a 64-bit IEEE-1394 MAC address or an arbitrarily-created bit pattern instead.[ND01]

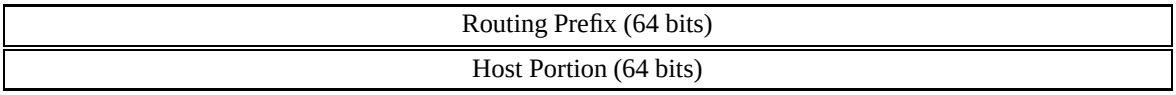

Figure 5: Example of IPv6 Global Unicast Address

IPv4 multicast addresses and IPv6 multicast addresses have a somewhat different format, since they are used for group communication. Also, IPv4 has the concept of a broadcast address, whilst IPv6 considers broadcast addresses to be multicast addresses.

5.3 Naming in Distributed Systems

From the middle 1980s through the late 1990s, a number of other naming systems were created and deployed. Many of these were proprietary to the inventing firm, though many were also relatively widely deployed. Most of these naming systems were invented primarily to facilitate distributed computing environments. Sun Microsystems invented the *Yellow Pages* system, later renamed *Network Information Service* due to a UK trademark. Similarly, NeXT Computer invented its NetInfo system, which has since been included by Apple Computer in its MacOS X

operating system. In academic environments, the Andrew System [MSC+86] from Carnegie-Mellon University was and is often deployed instead of Sun's NIS/NFS combination.

Sun's system provided both a naming system and a directory service. Their directory service included the ability to provide a service name and obtain the Remote Procedure Call (RPC) location of that service. One could also use NIS/YP for name to address resolution within a computing cluster, instead of using the DNS. Also, one could use NIS/YP to distribute certain files, most usually configuration files, to all the nodes in the computing cluster. Perhaps the most important use of NIS/YP was to provide transparent access to remote file systems accessed via Sun's Network File System (NFS). With NIS/YP, the actual location of a given portion of the logical local file system would be determined automatically, without the user having to be aware that a remote file system was being accessed rather than a local file system. With Sun RPC, there is a library call, *getrpcbyname()*, that can be used to take a well-known RPC Service name and determine the corresponding RPC Program Number. In practice, this is either implemented via a local flat file in */etc/rpc* or via the equivalent file made available to all nodes within a computing cluster through the NIS/YP service.

The Andrew File System was designed to address the same general problems that NFS was intended to solve, with some important enhancements. First, AFS uses Kerberos [KN93] for its underlying security mechanism, which enabled fine-grained user-to-host authentication. Kerberos provides authentication services for a distributed system. Kerberos has at least two namespaces, the first is the *User-ID* and the second is the *Realm*. A Kerberos Realm is an administrative domain, commonly, though not necessarily, the same as a DNS subdomain. Realm names are intended to be globally unique. The User-ID is intended to be unique within a given Realm. Second, AFS provides a single global filesystem naming structure, unlike NFS which provides a naming structure that varies depending on the configuration of one's local NIS group. The Open Software Foundation's Distributed Computing Environment was based upon AFS and Kerberos, but has never been widely deployed.

MIT also experimented with a set of extensions to the Domain Name System and also invented their own directory service known as Hesiod [Dye88] as part of their *Athena* research project. Unlike the ordinary Internet DNS, the Hesiod variant included sophisticated service location support. So one could perform a Hesiod lookup to determine a particular user's password file entry or the name of the default printer for some sub-domain. Hesiod was implemented by modifying the existing UC Berkeley BIND software package. This implementation choice meant that MIT did not need to license any commercial software. It also meant that the MIT implementation of Hesiod could be made freely available outside MIT.

5.4 URLs and URNs

In the early 1990s, students at the (US) University of Minnesota invented the *Gopher* application and application-protocol.[AML+93] This was a direct forerunner of the world wide web. It was much easier to use than prior Internet applications, such as ftp, but not quite so easy to use as early web browsers. Its key invention was having a single application that could use several different application protocols to access information across the global Internet.

Tim Berners-Lee's invention of the world wide web directly led to the Internet boom of the 1990s and a dramatic increase in the deployment of Internet services. An important component of his success was the creation of a new namespace for the Internet, the Universal Resource Locator (URL).[BLMM94] A URL specifies the access protocol (e.g. Hyper-Text Transfer Protocol or HTTP), a domain name (possibly a virtual domain name), and a pathname to the object relative to that domain name.

Ease of use considerations, and the absence of specific service location support in the DNS, led to widespread use of the DNS's CNAME record to provide domain names that acted as service names. For example, *www.cnn.com* is not the name of a host, but rather is the name of the web service of CNN. This web service is implemented as a set of web servers with a server load balancer appliance placed in front of those servers to hide the service implementation details. This overloading of the hostname function of the Domain Name Service would not have been necessary had the Internet Architecture included the concept of a service name or other integrated service location support.

Closely related to the URL is the URN. The URN provides a location-independent long-lived name for a data object, such as a particular document.[BL94, SM94, Moa97] For example, the Internet RFC-791 might have the URN value of *urn:ietf:rfc-791*. The designers of URNs intend that some object discovery service could be created that takes a URN as input and provides a copy of the object or a current pointer to the object. By contrast, URLs are often short-lived and always specify the location of the object in the URL name.

6 Tussle Era

The original set of namespaces provided by Internet standards were not adequate even during the late 1980s, as Hesiod and other naming services demonstrate. During the Tussle Era that started with the widespread deployment and use of the World Wide Web, these shortcomings became even more obvious. Unfortunately, the economic environment of the Internet bubble meant that good technical solutions usually lost out to heavily marketed solutions. One effect of the new economic environment is that many technical issues are resolved on the basis of marketing or economic strength (e.g. the desktop computing monopoly), rather than by selecting the best technical solution to the issue. This section will explore these issues in more detail.

6.1 Dynamic Host Configuration

Starting in the late 1980s, several vendors invented workstation boostrap protocols, for example BOOTP [CG85] to reduce the complexity and the cost of managing a set of workstations. Such protocols provided important system configuration parameters at boot time, for example the domain name, IP address, and network mask for the system.

In the earliest days of the Internet, hosts were large room-sized or desk-sized objects that were not mobile between rooms or between buildings. In later days, particularly with the advent of much more compact PCs and laptop computers, devices often moved from room to room or building to building. While the BOOTP protocol provided some of the desired capability, a more comprehensive solution was desired. As a result the BOOTP protocol evolved into the Dynamic Host Configuration Protocol (DHCP). [Dro93]

A limitation of DHCP is that it could not update the Domain Name System. So if a laptop was physically moved such that it could not retain the same IP address at its new location, the system would also have to acquire a new domain name. This is widely perceived as undesirable. People want to have a given device retain its domain name even if it moves from room to room or building to building. This led to the creation of the Dynamic DNS Update protocol [VTRB97] which enables a relocated host to retain its domain name and update the binding between that domain name and the IP address(es) of the device as the device relocated. While Dynamic DNS Update is now standardised by the IETF, it is not yet widely deployed. In order to prevent an adversary from forging a Dynamic DNS Update from a victim, security mechanisms were added to the proposed Dynamic DNS update protocol before final standardisation. In turn, this increased the operational and deployment complexity of Dynamic DNS, which has impeded its deployment.

In the event that any new namespace(s) were created for the Internet, it would be important to carefully consider how those new namespaces would interact with mobile networked devices and whether the new namespace directory, lookup, or mapping services would be able to appropriately support such mobility.

6.2 Load Balancers

As noted above, some domain names, such as *www.cnn.com*, are now being used as service names or cluster names, rather than being conventional host names. This practice is now quite common.

In a typical web deployment, there is a load-balancer providing application-layer proxy services. That load-balancer usually has a single public IP address and obscures the existence of multiple web servers that hide behind the load-balancer. In such a deployment, the end user sends a request to the service address, which is the load balancer's public address, and then the load balancer will select an internal web server to handle the request and provide bi-directional TCP header and IP header translation so that the user's request can be handled by that internal web server. Some such appliances can migrate a request from one internal server to another internal server in the event the first server should fail. To perform these functions, the load balancer appliance needs to have an intimate knowledge of the upper-layer protocols, such as the transport-protocol and also the application-layer protocol, in part because the IP address might be embedded in any or all of these.

Moreover, this use of the domain name as a service name indicates the perceived utility of having service names. In consequence, one should consider whether it would be sensible to enhance the Internet Architecture by adding formal support for service names or service identifiers. The experience of MIT Project Athena with Hesiod and the current abuse of domain names for service names each suggest that such support is strongly desirable.

6.3 Mobility

In the early Internet deployments, few if any hosts were truly mobile, moving from one location to another. In the modern era, there are numerous mobile devices with networking capability, for example laptop computers and

handheld personal data assistants. Mobility is an increasingly important capability for users, one that is not well supported by current Internet standards and technologies.

Because IP addresses are intended to be allocated topologically, as a device moves a sufficiently far distance then the natural event would be to change the IP address of its interface to reflect the device's new location in the topology. However, the existing Mobile IP devices do not really work this way – because a change in IP address would cause existing IP sessions to be dropped. IP sessions would be lost in such a case because the most widely deployed transport-layer protocols, UDP and TCP, include both local and remote IP addresses in the transport-layer state. [2] Additionally, some application-layer protocols, for example the File Transfer Protocol, use the raw IP addresses inside the application protocol. If the transport-layer and application-layer protocols used a topology-independent identifier, rather than the IP address, then mobility would be substantially easier to implement and deploy. At present, no such topology-idependent identifier exists. The domain-name is not a candidate because it has been so badly overloaded, sometimes referring to a host, other times referring to a cluster, and other times being used as a service name.

6.4 NAT and Firewalls

In the early days of the Internet, most users were in academic or research environments and most users were well behaved. As the network has grown, the number of malicious network users has grown significantly. Starting in the mid 1990s, many organisations deployed network firewall appliances between the global Internet and the organisation's internal network.

There are many kinds of firewalls, each providing different security properties. Perhaps the simplest firewall is a router with application-layer access lists that block external access to certain internal systems or internal services. The most severe firewall is probably an air gap – isolating one's network from the global Internet entirely. In between there are a wide range of implementations.

A common issue with firewalls and for firewall implementers is dealing with applications that embed the IP address inside the application or its application-layer protocol(s). Some firewalls try to inspect deep inside each packet to fully understand the payload before deciding whether or not to let that packet through the firewall.

Modern firewalls often include a network address translation (including its close cousin port address translation) capability. This lets the organisation's internal network be numbered in private IP address space, providing additional separation between the networks. NAT is incompatible with applications that embed the IP address inside the application or inside its application-layer protocol, unless the NAT has been specifically enhanced with knowledge of that application so that the NAT process can also modify the application-layer packets as they traverse the NAT. This increases the complexity and cost of the NAT device. Many users perceive that deploying a NAT will increase the security of their internal network, though this is not necessarily the case, particularly for simple NAT implementations.

Some in the Internet community view NAT devices as abominations. The author does not subscribe to that view. In the author's view, the NAT function is legitimate and would not be so problematic for end users if the protocols above the network-layer had used appropriate data hiding, appropriate identifiers, and appropriate networking abstractions.

6.5 IP Security

The IP Security protocols, the Encapsulating Security Payload (ESP and the Authentication Header (AH), are intended to provide network security, where the transmission is cryptographically protected and the communicating end-points have cryptographic confirmation of their respective identities, rather than transmission security, where the transmission would be protected on the wire but there might not be cryptographic confirmation of the respective endpoint identities.[Atk95c, Atk95a, Atk95b]

A fundamental concept in the IP Security protocols is the *Security Association*. A Security Association is the set of information about a given IPsec session. All IPsec Security Associations are unidirectional. The elements of an IPsec Security Association are:

When the author was first designing the IP Security protocols, ESP and AH, it quickly became apparent that the Internet Architecture did not have the type of network-layer identifier that IPsec truly requires for optimal operation. Ideally the IP Security Association does *not* bind to the network interface of the end systems, which is what an IP Address really denotes. Instead, the IP Security Association ought to bind to a topology-independent network-layer identifier for each end system. Since no such identifier exists, the engineering compromise was to use the IP

[2] In BSD Unix terms, in the *Transport Control Block*.

Data Element	Typical Data Type	Example
Source Identity	IP Address	1.2.3.4
Destination Identity	IP Address	2.3.4.5
Security Parameters Index	unsigned integer	6789
Security Protocol	Protocol-Number	ESP
Encryption Algorithm	Algorithm-Name	DES
Encryption Algorithm Mode	Mode-Name	CBC
Authentication Algorithm	Algorithm-Name	SHA-1
Authentication Algorithm Mode	Mode-Name	HMAC
Key LifeType	seconds or kilobytes	seconds
Key Lifetime	unsigned integer	600

Table 1: Typical IP Security Association

Address instead. This makes it difficult to use the IPsec protocols across a NAT device. If one tries to use a Security Association without any meaningful Source Identity or Destination Identity, then one becomes vulnerable to a range of different forgery attacks – which is entirely unacceptable, particularly for a security protocol.

7 Current Namespaces

In the current Internet, identifiers in common use today include the IP address, the fully qualified Domain Name, and the Socket, which defines the triple of IP address, transport-protocol type, and transport port number. Also, some applications use a mailbox name, which is the combination of userid and either hostname or domainname. In Kerberos environments, one also has the Kerberos realm and Kerberos userid, which might or might not map directly to the local DNS domain and local userid.

The IP address has heavily overloaded semantics; it is used for significantly different purposes in different applications, different protocols, and different protocol layers. In routing terms, an IP address is the name of a particular interface of a system, not a topology-independent name of the system. Similarly, the domain name has heavily overloaded semantics. In some contexts, a domain name represents a domain; in other contexts, it represents a single host or a set of hosts or even acts as a service name.

With CLNP and IPv6 on a local area network, the low-order bits of the address are normally the MAC address associated with the host's LAN interface(s). So some have proposed that this be used as a host identifier.

8 Conclusions

The overloaded semantics of the IP address and of the domain-name indicate that the Internet Architecture lacks a sufficiently rich set of namespaces. If a sufficiently rich and appropriately tailored set of namespaces existed, there would not be the need or desire to misuse the domain name as a service name, for example.

The lack of a distinct service address means that the semantics of a domain-name are overloaded. It can be difficult to know whether a given domain-name is a host, a set of hosts, or just a domain. It might be important for an application to know which set of semantics actually applies to a given domain-name. The lack of a distinct transport-layer identifier, one distinct from the IP address, adversely impacts the ability to use NAT devices, to support mobile hosts, and other desirable capabilities. Further, the lack of a suitable network-layer ID, one that is unrelated to the IP address, makes it much harder to deploy IP Security.

The historical accident of the BSD Sockets API being designed, implemented, and deployed before the widespread deployment and availability of the Domain Name System protocol has meant that inappropriately low-level objects are used in the primary application programming interface for networking and distributed systems. A replacement API with a more suitable set of high-level objects will help ensure that application designers and implementers do not inappropriately use network-layer addresses as system identifiers. In turn, this will help with deployment of mobile systems and the deployment of new applications through network address translation devices.

It appears strongly desirable to investigate the addition of one or more additional namespaces to the Internet Architecture. Such investigation should clearly indicate which existing issues any new namespace is intended to address. Also, any proposal for a new address space should also indicate which other namespaces need to be mappable with the new namespace. For example, if one defines a new host identifier namespace, ought that new host identifier be

mappable with an IP address ? If yes, should bi-directional mappings be supported or not ? and how would that mapping service or function be implemented ?

References

[AML+93] F. Anklesaria, M. McCahill, P. Lindner, D. B. Johnson, D. Torrey, and B. Albert. The Internet gopher protocol (a distributed document search and retrieval protocol). RFC 1436, Internet Engineering Task Force, March 1993.

[Atk95a] R. Atkinson. IP Authentication Header. RFC 1826, Internet Engineering Task Force, August 1995.

[Atk95b] R. Atkinson. IP Encapsulating Security Payload (ESP). RFC 1827, Internet Engineering Task Force, August 1995.

[Atk95c] R. Atkinson. Security Architecture for the Internet Protocol. RFC 1825, Internet Engineering Task Force, August 1995.

[Bhu71] A. K. Bhushan. File transfer protocol. RFC 114, Internet Engineering Task Force, April 1971.

[BL94] T. Berners-Lee. Universal resource identifiers in WWW: a unifying syntax for the expression of names and addresses of objects on the network as used in the world-wide web. RFC 1630, Internet Engineering Task Force, June 1994.

[BLMM94] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators (URL). RFC 1738, Internet Engineering Task Force, December 1994.

[BPTW73] A. K. Bhushan, K. T. Pogran, Raymond Tomlinson, and J. A. White. Standardizing network mail headers. RFC 561, Internet Engineering Task Force, September 1973.

[CG85] W. J. Croft and J. Gilmore. Bootstrap protocol. RFC 951, Internet Engineering Task Force, September 1985.

[Chi99] J.N. Chiappa. Why Topologically Sensitive Names (i.e. Addresses) Are Inherently Variable Length in a Global-Scale Communication Network. Unpublished draft, NameSpace Research Group (NSRG), Internet Research Task Force, 1999.

[Cla82] D. D. Clark. Name, addresses, ports, and routes. RFC 814, Internet Engineering Task Force, July 1982.

[CP72] V. G. Cerf and J. B. Postel. Well known socket numbers. RFC 322, Internet Engineering Task Force, March 1972.

[DB86] K.J. Dunlap and J. M. Bloom. Experiences Implementing BIND, A Distributed Name Server for the DARPA Internet. In *Proceedings of USENIX Summer Conference*, pages 172–181. USENIX, June 1986.

[Dee89] S. E. Deering. Host extensions for IP multicasting. RFC 1112, Internet Engineering Task Force, August 1989.

[Deu73] L. P. Deutsch. Host names on-line. RFC 606, Internet Engineering Task Force, December 1973.

[Dro93] R Droms. Dynamic host configuration protocol. RFC 1531, Internet Engineering Task Force, October 1993.

[Dye88] S. P. Dyer. The hesiod name server. In *Proceedings of the USENIX Winter 1988 Technical Conference*, pages 183–190, Berkeley, CA, 1988. USENIX Association.

[Kar71] P. M. Karp. Standardization of host mnemonics. RFC 226, Internet Engineering Task Force, September 1971.

[KN93] J. Kohl and C. Neuman. The kerberos network authentication service (V5). RFC 1510, Internet Engineering Task Force, September 1993.

[LMKQ89] Samuel J. Leffler, Marshall Kirk McKusick, Michael J. Karels, and John S. Quarterman. *"The Design and Implementation of the 4.3 BSD UNIX Operating System"*. Addison-Wesley, Menlo Park, CA, 1989.

[MD88] P.V. Mockapetris and K.J. Dunlap. Development of the Domain Name System. *ACM Computer Communication Review*, 18(4):123–133, August 1988.

[Mil81] D. L. Mills. Internet name domains. RFC 799, Internet Engineering Task Force, September 1981.

[Moa97] R. Moats. URN syntax. RFC 2141, Internet Engineering Task Force, May 1997.

[Moc83a] P. V. Mockapetris. Domain names: Concepts and facilities. RFC 882, Internet Engineering Task Force, November 1983.

[Moc83b] P. V. Mockapetris. Domain names: Implementation specification. RFC 883, Internet Engineering Task Force, November 1983.

[MSC+86] James H. Morris, Mahadev Satyanarayanan, Michael H. Conner, John H. Howard, David S. Rosenthal, and F. Donelson Smith. Andrew: a Distributed Personal Computing Environment. *Communications of the ACM*, 29(3):184–201, 1986.

[MW71] J. T. Melvin and R. W. Watson. First cut at a proposed telnet protocol. RFC 97, Internet Engineering Task Force, February 1971.

[ND01] T. Narten and R. Draves. Privacy Extensions for Stateless Address Autoconfiguration in IPv6. RFC 3041, Internet Engineering Task Force, January 2001.

[Pos81] J. B. Postel. NCP/TCP transition plan. RFC 801, Internet Engineering Task Force, November 1981.

[Pos83] J. B. Postel. Domain names plan and schedule. RFC 881, Internet Engineering Task Force, November 1983.

[Sho78] J.F. Shoch. Inter-Network Naming, Addressing, and Routing. Internet Experiment Note 19, ARPA Network Working Group, January 1978.

[SM94] K. Sollins and L. Masinter. Functional requirements for uniform resource names. RFC 1737, Internet Engineering Task Force, December 1994.

[SP82] Z. Su and J. B. Postel. Domain naming convention for Internet user applications. RFC 819, Internet Engineering Task Force, August 1982.

[VTRB97] Paul Vixie, Sue Thomson, Y. Rekhter, and Jim Bound. Dynamic updates in the domain name system (DNS UPDATE). RFC 2136, Internet Engineering Task Force, April 1997.

[YHK93] W. Yeong, T. Howes, and S. E. Kille. X.500 lightweight directory access protocol. RFC 1487, Internet Engineering Task Force, July 1993.