RFC: 814

NAME, ADDRESSES, PORTS, AND ROUTES

David D. Clark
MIT Laboratory for Computer Science
Computer Systems and Communications Group
July, 1982

1. Introduction

It has been said that the principal function of an operating system is to define a number of different names for the same object, so that it can busy itself keeping track of the relationship between all of the different names. Network protocols seem to have somewhat the same characteristic. In TCP/IP, there are several ways of referring to things. At the human visible interface, there are character string "names" to identify networks, hosts, and services. Host names are translated into network "addresses", 32-bit values that identify the network to which a host is attached, and the location of the host on that net. Service names are translated into a "port identifier", which in TCP is a 16-bit value. Finally, addresses are translated into "routes", which are the sequence of steps a packet must take to reach the specified addresses. Routes show up explicitly in the form of the internet routing options, and also implicitly in the address to route translation tables which all hosts and gateways maintain.

This RFC gives suggestions and guidance for the design of the tables and algorithms necessary to keep track of these various sorts of identifiers inside a host implementation of TCP/IP.

2. The Scope of the Problem

One of the first questions one can ask about a naming mechanism is how many names one can expect to encounter. In order to answer this, it is necessary to know something about the expected maximum size of the internet. Currently, the internet is fairly small. It contains no more than 25 active networks, and no more than a few hundred hosts. This makes it possible to install tables which exhaustively list all of these elements. However, any implementation undertaken now should be based on an assumption of a much larger internet. The guidelines currently recommended are an upper limit of about 1,000 networks. If we imagine an average number of 25 hosts per net, this would suggest a maximum number of 25,000 hosts. It is quite unclear whether this host estimate is high or low, but even if it is off by several factors of two, the resulting number is still large enough to suggest that current table management strategies are unacceptable. Some fresh techniques will be required to deal with the internet of the future.

3. Names

As the previous section suggests, the internet will eventually have a sufficient number of names that a host cannot have a static table which provides a translation from every name to its associated address. There are several reasons other than sheer size why a host would not wish to have such a table. First, with that many names, we can expect names to be added and deleted at such a rate that an installer might spend all his time just revising the table. Second, most of the names will refer to addresses of machines with which nothing will ever be

exchanged. In fact, there may be whole networks with which a particular host will never have any traffic.

To cope with this large and somewhat dynamic environment, the internet is moving from its current position in which a single name table is maintained by the NIC and distributed to all hosts, to a distributed approach in which each network (or group of networks) is responsible for maintaining its own names and providing a "name server" to translate between the names and the addresses in that network. Each host is assumed to store not a complete set of name-address translations, but only a cache of recently used names. When a name is provided by a user for translation to an address, the host will first examine its local cache, and if the name is not found there, will communicate with an appropriate name server to obtain the information, which it may then insert into its cache for future reference.

Unfortunately, the name server mechanism is not totally in place in the internet yet, so for the moment, it is necessary to continue to use the old strategy of maintaining a complete table of all names in every host. Implementors, however, should structure this table in such a way that it is easy to convert later to a name server approach. In particular, a reasonable programming strategy would be to make the name table accessible only through a subroutine interface, rather than by scattering direct references to the table all through the code. In this way, it will be possible, at a later date, to replace the subroutine with one capable of making calls on remote name servers.

A problem which occasionally arises in the ARPANET today is that

the information in a local host table is out of date, because a host has moved, and a revision of the host table has not yet been installed from the NIC. In this case, one attempts to connect to a particular host and discovers an unexpected machine at the address obtained from the local table. If a human is directly observing the connection attempt, the error is usually detected immediately. However, for unattended operations such as the sending of queued mail, this sort of problem can lead to a great deal of confusion.

The nameserver scheme will only make this problem worse, if hosts cache locally the address associated with names that have been looked up, because the host has no way of knowing when the address has changed and the cache entry should be removed. To solve this problem, plans are currently under way to define a simple facility by which a host can query a foreign address to determine what name is actually associated with it. SMTP already defines a verification technique based on this approach.

4. Addresses

The IP layer must know something about addresses. In particular, when a datagram is being sent out from a host, the IP layer must decide where to send it on the immediately connected network, based on the internet address. Mechanically, the IP first tests the internet address to see whether the network number of the recipient is the same as the network number of the sender. If so, the packet can be sent directly to the final recipient. If not, the datagram must be sent to a gateway for further forwarding. In this latter case, a second decision must be

made, as there may be more than one gateway available on the immediately attached network.

When the internet address format was first specified, 8 bits were reserved to identify the network. Early implementations thus implemented the above algorithm by means of a table with 256 entries, one for each possible net, that specified the gateway of choice for that net, with a special case entry for those nets to which the host was immediately connected. Such tables were sometimes statically filled in, which caused confusion and malfunctions when gateways and networks moved (or crashed).

The current definition of the internet address provides three different options for network numbering, with the goal of allowing a very large number of networks to be part of the internet. Thus, it is no longer possible to imagine having an exhaustive table to select a gateway for any foreign net. Again, current implementations must use a strategy based on a local cache of routing information for addresses currently being used.

The recommended strategy for address to route translation is as follows. When the IP layer receives an outbound datagram for transmission, it extracts the network number from the destination address, and queries its local table to determine whether it knows a suitable gateway to which to send the datagram. If it does, the job is done. (But see RFC 816 on Fault Isolation and Recovery, for recommendations on how to deal with the possible failure of the gateway.) If there is no such entry in the local table, then select any

accessible gateway at random, insert that as an entry in the table, and use it to send the packet. Either the guess will be right or wrong. If it is wrong, the gateway to which the packet was sent will return an ICMP redirect message to report that there is a better gateway to reach the net in question. The arrival of this redirect should cause an update of the local table.

The number of entries in the local table should be determined by the maximum number of active connections which this particular host can support at any one time. For a large time sharing system, one might imagine a table with 100 or more entries. For a personal computer being used to support a single user telnet connection, only one address to gateway association need be maintained at once.

The above strategy actually does not completely solve the problem, but only pushes it down one level, where the problem then arises of how a new host, freshly arriving on the internet, finds all of its accessible gateways. Intentionally, this problem is not solved within the internetwork architecture. The reason is that different networks have drastically different strategies for allowing a host to find out about other hosts on its immediate network. Some nets permit a broadcast mechanism. In this case, a host can send out a message and expect an answer back from all of the attached gateways. In other cases, where a particular network is richly provided with tools to support the internet, there may be a special network mechanism which a host can invoke to determine where the gateways are. In other cases, it may be necessary for an installer to manually provide the name of at

least one accessible gateway. Once a host has discovered the name of one gateway, it can build up a table of all other available gateways, by keeping track of every gateway that has been reported back to it in an ICMP message.

5. Advanced Topics in Addressing and Routing

The preceding discussion describes the mechanism required in a minimal implementation, an implementation intended only to provide operational service access today to the various networks that make up the internet. For any host which will participate in future research, as contrasted with service, some additional features are required. These features will also be helpful for service hosts if they wish to obtain access to some of the more exotic networks which will become part of the internet over the next few years. All implementors are urged to at least provide a structure into which these features could be later integrated.

There are several features, either already a part of the architecture or now under development, which are used to modify or expand the relationships between addresses and routes. The IP source route options allow a host to explicitly direct a datagram through a series of gateways to its foreign host. An alternative form of the ICMP redirect packet has been proposed, which would return information specific to a particular destination host, not a destination net. Finally, additional IP options have been proposed to identify particular routes within the internet that are unacceptable. The difficulty with implementing these new features is that the mechanisms do not lie

entirely within the bounds of IP. All the mechanisms above are designed to apply to a particular connection, so that their use must be specified at the TCP level. Thus, the interface between IP and the layers above it must include mechanisms to allow passing this information back and forth, and TCP (or any other protocol at this level, such as UDP), must be prepared to store this information. The passing of information between IP and TCP is made more complicated by the fact that some of the information, in particular ICMP packets, may arrive at any time. The normal interface envisioned between TCP and IP is one across which packets can be sent or received. The existence of asynchronous ICMP messages implies that there must be an additional channel between the two, unrelated to the actual sending and receiving of data. (In fact, there are many other ICMP messages which arrive asynchronously and which must be passed from IP up to higher layers. See RFC 816, Fault Isolation and Recovery.)

Source routes are already in use in the internet, and many implementations will wish to be able to take advantage of them. The following sorts of usages should be permitted. First, a user, when initiating a TCP connection, should be able to hand a source route into TCP, which in turn must hand the source route to IP with every outgoing datagram. The user might initially obtain the source route by querying a different sort of name server, which would return a source route instead of an address, or the user may have fabricated the source route manually. A TCP which is listening for a connection, rather than attempting to open one, must be prepared to receive a datagram which contains a IP return route, in which case it must remember this return route, and use it as a source route on all returning datagrams.

6. Ports and Service Identifiers

The IP layer of the architecture contains the address information which specifies the destination host to which the datagram is being sent. In fact, datagrams are not intended just for particular hosts, but for particular agents within a host, processes or other entities that are the actual source and sink of the data. IP performs only a very simple dispatching once the datagram has arrived at the target host, it dispatches it to a particular protocol. It is the responsibility of that protocol handler, for example TCP, to finish dispatching the datagram to the particular connection for which it is destined. This next layer of dispatching is done using "port identifiers", which are a part of the header of the higher level protocol, and not the IP layer.

This two-layer dispatching architecture has caused a problem for certain implementations. In particular, some implementations have wished to put the IP layer within the kernel of the operating system, and the TCP layer as a user domain application program. Strict adherence to this partitioning can lead to grave performance problems, for the datagram must first be dispatched from the kernel to a TCP process, which then dispatches the datagram to its final destination process. The overhead of scheduling this dispatch process can severely limit the achievable throughput of the implementation.

As is discussed in RFC 817, Modularity and Efficiency in Protocol Implementations, this particular separation between kernel and user leads to other performance problems, even ignoring the issue of port

level dispatching. However, there is an acceptable shortcut which can be taken to move the higher level dispatching function into the IP layer, if this makes the implementation substantially easier.

In principle, every higher level protocol could have a different dispatching algorithm. The reason for this is discussed below. However, for the protocols involved in the service offering being implemented today, TCP and UDP, the dispatching algorithm is exactly the same, and the port field is located in precisely the same place in the header. Therefore, unless one is interested in participating in further protocol research, there is only one higher level dispatch algorithm. This algorithm takes into account the internet level foreign address, the protocol number, and the local port and foreign port from the higher level protocol header. This algorithm can be implemented as a sort of adjunct to the IP layer implementation, as long as no other higher level protocols are to be implemented. (Actually, the above statement is only partially true, in that the UDP dispatch function is subset of the TCP dispatch function. UDP dispatch depends only protocol number and local port. However, there is an occasion within TCP when this exact same subset comes into play, when a process wishes to listen for a connection from any foreign host. Thus, the range of mechanisms necessary to support TCP dispatch are also sufficient to support precisely the UDP requirement.)

The decision to remove port level dispatching from IP to the higher level protocol has been questioned by some implementors. It has been argued that if all of the address structure were part of the IP layer,

then IP could do all of the packet dispatching function within the host, which would lead to a simpler modularity. Three problems were identified with this. First, not all protocol implementors could agree on the size of the port identifier. TCP selected a fairly short port identifier, 16 bits, to reduce header size. Other protocols being designed, however, wanted a larger port identifier, perhaps 32 bits, so that the port identifier, if properly selected, could be considered probabilistically unique. Thus, constraining the port id to one particular IP level mechanism would prevent certain fruitful lines of research. Second, ports serve a special function in addition to datagram delivery: certain port numbers are reserved to identify particular services. Thus, TCP port 23 is the remote login service. If ports were implemented at the IP level, then the assignment of well known ports could not be done on a protocol basis, but would have to be done in a centralized manner for all of the IP architecture. Third, IP was designed with a very simple layering role: IP contained exactly those functions that the gateways must understand. If the port idea had been made a part of the IP layer, it would have suggested that gateways needed to know about ports, which is not the case.

There are, of course, other ways to avoid these problems. In particular, the "well-known port" problem can be solved by devising a second mechanism, distinct from port dispatching, to name well-known ports. Several protocols have settled on the idea of including, in the packet which sets up a connection to a particular service, a more general service descriptor, such as a character string field. These special packets, which are requesting connection to a particular

service, are routed on arrival to a special server, sometimes called a "rendezvous server", which examines the service request, selects a random port which is to be used for this instance of the service, and then passes the packet along to the service itself to commence the interaction.

For the internet architecture, this strategy had the serious flaw that it presumed all protocols would fit into the same service paradigm: an initial setup phase, which might contain a certain overhead such as indirect routing through a rendezvous server, followed by the packets of the interaction itself, which would flow directly to the process providing the service. Unfortunately, not all high level protocols in internet were expected to fit this model. The best example of this is isolated datagram exchange using UDP. The simplest exchange in UDP is one process sending a single datagram to another. Especially on a local net, where the net related overhead is very low, this kind of simple single datagram interchange can be extremely efficient, with very low overhead in the hosts. However, since these individual packets would not be part of an established connection, if IP supported a strategy based on a rendezvous server and service descriptors, every isolated datagram would have to be routed indirectly in the receiving host through the rendezvous server, which would substantially increase the overhead of processing, and every datagram would have to carry the full service request field, which would increase the size of the packet header.

In general, if a network is intended for "virtual circuit service",

or things similar to that, then using a special high overhead mechanism for circuit setup makes sense. However, current directions in research are leading away from this class of protocol, so once again the architecture was designed not to preclude alternative protocol structures. The only rational position was that the particular dispatching strategy used should be part of the higher level protocol design, not the IP layer.

This same argument about circuit setup mechanisms also applies to the design of the IP address structure. Many protocols do not transmit a full address field as part of every packet, but rather transmit a short identifier which is created as part of a circuit setup from source to destination. If the full address needs to be carried in only the first packet of a long exchange, then the overhead of carrying a very long address field can easily be justified. Under these circumstances, one can create truly extravagant address fields, which are capable of extending to address almost any conceivable entity. However, this strategy is useable only in a virtual circuit net, where the packets being transmitted are part of a established sequence, otherwise this large extravagant address must be transported on every packet. Since Internet explicitly rejected this restriction on the architecture, it was necessary to come up with an address field that was compact enough to be sent in every datagram, but general enough to correctly route the datagram through the catanet without a previous setup phase. The IP address of 32 bits is the compromise that results. Clearly it requires a substantial amount of shoehorning to address all of the interesting places in the universe with only 32 bits. On the other hand, had the

address field become much bigger, IP would have been susceptible to another criticism, which is that the header had grown unworkably large. Again, the fundamental design decision was that the protocol be designed in such a way that it supported research in new and different sorts of protocol architectures.

There are some limited restrictions imposed by the IP design on the port mechanism selected by the higher level process. In particular, when a packet goes awry somewhere on the internet, the offending packet is returned, along with an error indication, as part of an ICMP packet. An ICMP packet returns only the IP layer, and the next 64 bits of the original datagram. Thus, any higher level protocol which wishes to sort out from which port a particular offending datagram came must make sure that the port information is contained within the first 64 bits of the next level header. This also means, in most cases, that it is possible to imagine, as part of the IP layer, a port dispatch mechanism which works by masking and matching on the first 64 bits of the incoming higher level header.