Inter-Office Memorandum

To	Communication Protocols	Date	July 1, 1978
From	Ed Taft	Location	Palo Alto
Subject	Naming and Addressing Conventions for Pup	Organization	PARC/CSL

XEROX

Filed on: [Maxc1]<Pup>PupName.press

This is a revision to a memo with the same title dated October 16, 1975. A small number of minor changes have been made, but the principal motivation for this revision is to re-issue the memo in Bravo and Press formats.

This memo still contains the original justifications for the design that led to the present implementation. Remarks based on our experience with this design are presented in a small font.

Introduction

The questions to be touched in this memo are:

1. How are networks, hosts, and processes to be *named*? That is, what form of identification is to be used by someone attempting to establish contact with some service or resource? It is desirable that the naming convention we adopt require little or no user knowledge of host or process numbering conventions or (inter)network topology.

2. How are hosts and processes to be *addressed*? At the lowest level, a *port address* is merely a <network, host, socket> triple (a 48-bit number). However, a single host may be connected to more than one network or to the same network more than once, and a single process (or multiple identical instances of a process) may be associated with multiple ports on the same or different networks, possibly even on different hosts.

3. How does a host discover what network it is on and what hosts are gateways? How do we handle the degenerate cases in which a host does not know what network it is on and/or there are no accessible gateways? (These questions are of particular importance in the PARC environment because it is impractical to attempt to store the necessary information permanently in each host).

This memo discusses these questions in some detail and describes the conventions that have been adopted.

Host Names

Most internetworking protocols, if they discuss naming conventions at all, specify a simple hierarchical scheme in which each network has its own name space, independent of the name spaces of other networks. In order to reference an internetwork process address, it is necessary to specify (1) the network name, (2) the target host name (whatever it is called on that network), and (3) the

name (or number) of the process, port, or whatever on that host.

In the case of the PARC networks, it seems desirable to adopt a somewhat different approach: that *a single name space encompasses all networks*. This really says two things:

1. A given host or process name refers to one and only one host or process (or possibly to a set of identical instances of a host or process).

2. For any given host or process that has more than one network address, all such addresses are associated with a *single name*.

Such a scheme has several benefits. First, the single name space allows users to specify a target host or process by means of a simple identifier rather than by a full pathname that assumes user knowledge of (inter)network topology. For example, to refer to the Ivy file server, it is necessary to specify only a single name (e.g. "Ivy") rather than a full pathname such as "Parc-Net3+Ivy+3".

Second, even if a host resides on more than one network, or a process listens on several ports on the same or different networks, it is unnecessary for the user to make a decision (specified though a full pathname) distinguishing between these choices. Since this is nothing more than a special case of message routing decisions that hosts and gateways must make anyway, it seems quite reasonable to leave such a choice completely in the hands of host and gateway software (unless there is a special reason the user wants to specify a full pathname).

Finally, in the case where there exist more than one identical instance of a particular host or process, it might be advantageous to give all such instances the same name. For example, if several hosts provide a "telephone directory" service, or several hosts are prepared to accept error statistics from Alto memory diagnostics, there is no reason not to give all instances of such a service the same name. One must take care in application of the "identicalness" criterion, however; e.g. it would be poor form if printer output for a resident of Building 35 were actually sent to a "printer" server in Building 34.

As our inter-network has grown, the obvious drawbacks of a universal non-hierarchical name space have made themselves felt. We now favor a hierarchical name space of at least two levels, but based on organizational or geographical considerations rather than on network topology.

With these considerations in mind, we have adopted the following naming conventions. Name to address translation is performed using a universal *network directory*. This directory is currently maintained as a file on Maxc (see the memo The *Pup Network Directory and the PUPNM JSYS*) and is distributed automatically to all hosts (principally Gateways) that provide Name Lookup servers (see *Miscellaneous Services*).

Each entry in the directory consists of a list of *names*, a list of *addresses*, and a list of *attributes*.

A *name* is a string consisting of alphanumerics plus the characters ’-’ and ’/’ (others may be admitted by popular demand). Both upper and lower case alphabetics may be used; two names differing only in the case of their letters are considered identical. If more than one name appears in an entry’s name list, all the names are synonyms and may be used interchangeably.

An *address* is a triple consisting of a *network number*, a *host number*, and a *socket number*, as defined in the Pup specification. In the case that all three of these numbers are nonzero, the address completely specifies a Pup *port*. However, if one or more of the numbers is zero (i.e. unspecified), the address represents a subset of all possible ports.

The classes of addresses that we expect to find useful in practice are the following:

1. *Network addresses*, in which only the network number is specified.

2. *Host addresses*, in which the network and host numbers are specified.

3. *Port addresses*, in which all three numbers are specified.

4. *Well-known sockets*, addresses in which only the socket number is specified.

The addresses in a directory entry's address list describe alternative ways of accessing the entity associated with the entry (or multiple, identical instances of that entity). Accordingly, we require that all addresses in an address list have corresponding patterns of specified and unspecified elements.

An *attribute* is merely a pair consisting of an attribute name and an attribute value. The attribute name is composed according to the same rules as the names in a directory entry's name list, while the attribute value may be an arbitrary string.

In addition to the attribute list associated with an entire directory entry, individual addresses in the entry's address list may have attribute lists associated with them.

In practice, we have not used attributes in any interesting way. The present network directory records Alto locations and owners as attributes, but even these are not always up-to-date. This is another unfavorable effect of utilizing a single, centrally-maintained data base.

A *port name expression* is composed of name strings and address constants joined by the operator '+'.

A *name* is one of the name strings defined in the network directory, as described above. Its value is the associated list of addresses.

An address constant is in the form

<network number> # <host number> # <socket number>

where the numbers are specified in octal. An element of this constant may be left unspecified by supplying zero or by leaving it out entirely. Leading '#'s may be omitted. For example, "0#0#3", "##3", and "3" all denote an address constant with network and host numbers unspecified and socket number 3, while "3##" denotes network number 3.

Names and address constants may be combined by means of the '+' operator, which is roughly speaking an intersection operator. Its effect is to make an expression whose value is more specific (i.e. contains fewer unspecified elements) than either of its operands. For example, the value of "3##+##123" is "3##123". If a particular element is specified in both operands but with conflicting values, the intersection is empty.

When either of the operands is a name whose value is a list of addresses, the resulting value is also potentially a list. For example, the value of "Maxc1" in the network directory is the list 1#1#, 2#1#, 3#200#, 4#40#. Hence the value of the expression "Maxc1+123" is the list 1#1#123, 2#1#123, 3#200#123, 4#40#123. However, the value of the expression "3##+Maxc1" (or "Parc-Net3+Maxc1") is a single address, 3#200#, since the intersections of the given address constant with the other addresses in the list are empty.

Port Addresses and Packet Destinations

Given a naming convention such as that just described, one might argue that a Pup should be allowed to have a *name* as a destination, or (an equivalent and more practical alternative) a *list* of addresses rather than a single address. Hosts and gateways would be free to route packets to any one of the listed destinations, with the choice being made dynamically as network load or topology changed over time.

Such a scheme is unfortunately too cumbersome to be practical. Among other things, it would require the already over-long Pup header to be further extended. Furthermore, anomalies could arise in the case where a name referred, not to multiple addresses for the same process, but rather to multiple instances of a process.

It is the intent of this proposal, therefore, that the choice between alternative addresses be made at the time of the *initial connection* or *rendezvous* and remain fixed for the life of a "connection" (whatever we mean by that; e.g. as characterized by the Byte Stream Protocol). All packets are marked as arising from a single, specified source port and are delivered to a single, specified destination port.

Of course, the foregoing statement does not prohibit hosts from implementing ports (process connections) that will accept Pups from any of several sources or directed to any of several destinations. All existing implementations support such a "wildcard" capability.

Nor do we restrict the manner in which Pups may be *routed* between hosts and gateways. For example, if two hosts are connected to each other by two networks, it is perfectly reasonable to transmit over one network a Pup whose destination port specifies the other network. This capability permits communications to continue without interruption should one of the networks go down.

In practice, the existence of hosts with multiple inter-network addresses has led to significant complications in Pup addressing and routing. We presently favor a scheme in which each host has a single, unique ID that is independent of the identity of the network(s) to which the host is connected. Pups directed to that host carry a destination network number as a *hint* as to how the host may be reached, but the hint does not participate in the identification of the host itself and may be changed at any time.

Degenerate Cases

An important consideration in the PARC environment is that no hosts besides gateways should need initially to know what network they are on. Given the portability of Alto disk packs and even of entire machines, there is no practical way to permanently build into each host the network number(s) of the connected network(s).

This consideration leads to two requirements:

1. Intra-network communication should be possible without any of the hosts involved knowing what network they are connected to (though they must somehow know that they are connected to the *same* network). Such a situation might arise if the network were in fact not connected to any other network, or if all available gateways were down.

2. For the purposes of initiating inter-network communication, it should be possible for a host to (a) discover what network it is on and (b) discover what hosts on that network are gateways, and to what other networks.

To meet these requirements, we first set down the convention that no network may have a network number of zero. A zero value in the source network number field of a Pup header is defined to mean that the sending host does not know what network it is connected to. A zero value in the destination network field indicates that the Pup's destination is on the same network as its source.

A user or user process desiring to initiate intra-network communication may transmit Pups to a specific host on that network, setting both source and destination network number fields to zero. In this case, the assumption is that somehow the initiator of this communication *knows* that the two hosts are on the same network. Such "knowledge" might be obtained in various ways. For example, the (human) user might simply direct his Alto user process to establish contact with Ethernet host 123, which is a specific machine which he *knows* is on the same Ethernet as his own machine.

Note that the network directory described earlier is (strictly speaking) not usable by a host that does not know what network it is on, even if it is able to access that data base. That is, having looked up "Maxc2" and discovered that its possible addresses are 3#5# and 4#240#, we cannot do anything with this information since we don't know whether we are connected to network 3 or network 4 (or neither), and hence have no way of deciding whether to transmit to host 5 or host 240 on our "own" network, or even whether we can get there from here at all.

This difficulty would be eliminated by the unique host ID scheme mentioned earlier.

Network Determination and Gateway Location

All the above problems go away if the host somehow discovers what network(s) it is connected to. Additionally, to perform *inter*-network communication, it is necessary for a host to know both what network it is on and what hosts on that network are gateways. Accordingly, all hosts should, at initialization time (e.g. when "booted"), make use of the protocol about to be outlined. The actual specification of the protocol is given in the memo *Gateway Information Protocol*.

We first observe that all gateways *must* know what networks they are connected to. We then provide each gateway host with an *gateway information service* on a standard socket.

A host desiring to discover its local network number and gateways broadcasts a *Gateway Information Request* Pup to this socket at all (or a selected subset of) the hosts on its *own* network (i.e., setting the source and destination network fields to zero). A destination host number of zero is specifically reserved to distinguish a Pup that is being broadcast. All the operating gateways on that network then reply with a *Gateway Information Reply* Pup that includes the actual source and destination network numbers in the Pup header (since the gateway host always knows the identity of the network over which the request Pup arrived). The body of the message includes routing information indicating what other networks are accessible via that gateway.

The connected network number should now be saved by the requesting host for inclusion in the source network field of all subsequent outgoing Pups, and the routing information should be used for determining where to send packets addressed to hosts on other networks.