# Inter-Office Memorandum

| | | | |
|---|---|---|---|
| To | Communication Protocols | Date | May 29, 1979 |
| From | Ed Taft | Location | PARC/CSL |
| Subject | Caching Pup routing information | File | [Maxc1]<Pup>CacheRoute.bravo |

XEROX

Attributes: informal, technical, Alto, Distributed computing

Abstract: A technique is described by means of which non-gateway hosts may maintain a minimum amount of internetwork routing information, without loss of generality, through use of a cache.

While working on the Bcpl Pup package recently, I noticed that the size of the Pup internet (somewhere around 25 networks) was once again approaching the compiled-in size of the Pup routing table (32 entries this time). For implementation reasons, the number of entries in this table had to be a power of two. At four words per entry, increasing the size of the table to 64 entries would have caused it to consume an entire page of Alto memory.

Such an increase would be wasteful, since on any given Alto only a small number of routes are in use at one time. This seemed like a good opportunity to put into practice various ideas about caching routing information that have emerged from conversations among Dave Boggs, Hal Murray, Yogen Dalal, myself, and undoubtedly others.

I have implemented a routing information cache that will be included in the next release of the Bcpl Pup package. This memo describes the basic mechanisms, presents some relatively subtle issues that came to light during implementation, and touches on some possible variations to the strategies.

## Pup routing algorithm and routing update protocol

A general overview of the Pup routing procedures may be found in [1], and the detailed protocol specifications are contained in [2] and [3]. The routing protocols are divided into two parts: a *routing algorithm* that is applied to every outgoing Pup to direct it closer to its ultimate destination, and a *routing update protocol* by means of which dynamic routing information is maintained and distributed.

### *Internetwork routing*

Each host (gateways included) executes a routing procedure on every outgoing Pup. This procedure decides, as a function of the Pup destination port field, upon which *directly-connected network* the Pup is to be transmitted (if there is more than one choice), and it yields an *immediate destination host* which is the address on that network of either the ultimate destination or some gateway believed to be closer to the destination. As a Pup travels through the internet, each router employs the same algorithm based on local routing information, and each Pup is routed independently.

Under this organization, every host must maintain a local routing table containing an entry for each network to which the host might wish to direct Pups. For the purposes of routing Pups, each entry must either identify a network as being directly-connected or specify the address (network and host numbers) of a gateway on a directly-connected network. This information changes over time through the actions of the routing update protocol.

*Routing information update*

Routing information is maintained in a manner very similar to the Arpanet-style adaptive procedures [4]. Each gateway periodically broadcasts on every directly-connected network a Pup containing a copy of its own routing table. Other gateways that hear such a broadcast use the information in it to update their own local routing tables specifically, to incorporate new routes to networks or invalidate old routes to inaccessible networks as the internetwork topology changes.

With this arrangement, each gateway must maintain and distribute a routing table containing entries for *every* network known to be accessible. The routing information must include some measure of the quality of the route; the metric presently used is a fairly primitive one based on hop counts, but incorporating a better one based on delay or throughput would be a straightforward extension.

Since gateways broadcast their routing tables, non-gateway hosts on directly-connected networks can also hear these broadcasts and maintain their own routing tables in a similar manner. Indeed, existing implementations of the routing table maintenance procedures are identical for gateway and non-gateway hosts. (It is perhaps worth mentioning that at the basic Pup level, gateway and non-gateway hosts differ in only two respects: gateways forward Pups whereas non-gateway hosts simply discard Pups not addressed to them, and gateways both send and receive routing information Pups whereas non-gateways only receive them.)

A host may also broadcast a request for routing information and receive immediate responses from every gateway on the directly-connected network. This permits quick initialization of a host’s routing table when it comes up a frequent event for most Altos as opposed to waiting for the next periodic routing information broadcast.

## The routing information cache

Against this background, we desire to minimize the amount of routing information that individual non-gateway hosts must maintain locally. (We will consider extending this to gateway hosts later.) The obvious approach is for a host to maintain a cache of routing entries for those networks to which clients within the host are sending Pups; the number of such networks is quite small at any given time.

*Cache maintenance*

Since connections (or virtual circuits’’) are not maintained at the Pup internet level but rather are artifacts of higher-level, end-to-end protocols, the routing table maintenance algorithms must operate entirely on the basis of information gained during the Pup routing process itself. (This information can be augmented by hints provided by client programs, but the existence of such hints must never be depended upon.)

Thus, for each outgoing Pup, the router looks up the destination network in the cache. If an entry for that network is present, it is used to route the Pup. If no such entry exists, the Pup is discarded and activity is initiated to locate the network; specifically, a routing information request is broadcast. A dummy cache entry is created for the new network, possibly displacing the least recently used entry for some other network.

The routing information update procedure, upon receiving a routing table Pup from some gateway, updates the existing entries in its cache with the corresponding information from the Pup and discards the remainder of the routing table.

*Details and pitfalls*

The cache is arranged in order from most to least recently used to permit the replacement algorithm to work properly. It is acceptable to use a relatively simple structure to represent the cache, e.g., a queue, since in a machine with only a small number of simultaneous active connections, the relevant entries will almost always be near the front of the queue.

Use of an entry to route a Pup causes the entry to be promoted to the front of the queue (unless it is already at the front which it is usually is). Note, however, that the routing update procedure should *not* change the order of the entries.

Care must be taken not to permit entries for directly-connected networks to be displaced from the cache (though they may be demoted in the ordering due to lack of use), since a host must always know the identity of the networks to which it is connected in order to send Pups outside those networks.

The routing table update procedure may be improved further, as follows: after receiving a routing table Pup and updating existing entries in the cache, it then fills in any empty cache entries with information about other networks in the order they appear in the Pup. Assuming the entries in the Pup are ordered according to how recently they were used by the gateway that sent the Pup, this has the beneficial effect of initializing a host's cache with entries for the networks accessed most actively from within the local area.

Given this modification, a gateway may use precisely the same routing table management algorithm so long as its cache is large enough to hold entries for the entire accessible internet.

An externally-callable procedure called LocateNet is provided to permit clients to pass hints about networks they expect to send Pups to in the near future. LocateNet performs the same actions as the router (described previously) if no entry for the specified network is present in the cache. This facility is used, for example, within the Name Lookup Protocol module, since the address returned by a name lookup is almost always used to initiate a conversation.

*Degenerate implementations*

A much simpler mechanism is usable in specialized applications where it is known that there will be only one active conversation at a time. In this case, the cache may in effect consist of a single entry corresponding to the network of the other partner in the conversation. (Of course, the router must also keep information about the identity of the directly-connected networks.) This is precisely the technique used in the existing TeleSwat user implementation.

A further simplification is employed in certain very primitive servers that never send Pups spontaneously but only in response to request Pups. When generating a response to a Pup, such a server simply uses the immediate source network and host (i.e., the host that sent the Pup over its final hop) as the immediate destination network and host for the response. This takes advantage of the assumption that the reverse of the route taken by the request Pup is an acceptable route for the response; it does not require processing routing information Pups at all. This technique is used by the TeleSwat server, the EFTP receiver used in the Alto bootstrap process, the Spruce status server, and others.

## Future work

It is clear that a similar technique could be used to maintain routing caches (as opposed to complete routing tables) in gateway hosts. The routing update protocol would have to be changed so that a host could request information about a specific network rather than only obtain the full internet routing table as it does at present. A gateway receiving such a request for a network not in its own cache would in turn interrogate all its neighboring gateways, and the request would propagate through the internet until the desired network had been located. The resulting routing information would thereafter be retained in the caches of all gateways lying along the best route, so long as the route continued to be used.

Extending the routing cache strategy in this manner introduces several additional problems. The first is that of properly controlling what is in effect an internet broadcast, i.e., to prevent unbounded regeneration of the request. There is a known technique for accomplishing this, called *reverse-path forwarding* [5], but it depends on all nodes having complete and more-or-less consistent routing tables the very requirement we are trying to eliminate. Second, in a large internet such broadcasts would be extremely costly. Third, a route that had been discovered by such means would not thereafter be replaced automatically by better routes as it is at present, since information about reaching a particular network would tend to remain only in the caches of gateways along the route that was actually being used.

David Boggs is working on some techniques for finding and maintaining routes in a large internet. In the meantime, the Pup design requires that gateways maintain and distribute complete routing tables but permits non-gateway hosts to cache subsets of the routing information that they require.

## References

[1] D. Boggs, J. Shoch, E. Taft, and R. Metcalfe, Pup: "An Internetwork Architecture", file [Maxc1]<Pup>PupPaper.press.

[2] E. Taft and R. Metcalfe, "Pup Specifications", file [Maxc1]<Pup>PupSpec.press.

[3] E. Taft, "Gateway Information Protocol", file [Maxc1]<Pup>GatewayInformation.press.

[4] J. McQuillan, *Adaptive Routing Algorithms for Distributed Computer Networks*, Harvard Ph.D. thesis, Report no. 2831, Bolt Beranek and Newman, May 1974.

[5] Y. Dalal and R. Metcalfe, "Reverse Path Forwarding of Broadcast Packets", *CACM*, vol. 21 no. 12, December 1978.