# Internet Routing over Large Public Data Networks using Shortcuts

Paul F. Tsuchiya, Bellcore, tsuchiya@thumper.bellcore.com

## Abstract

With the emergence of large switched public data networks that are well-suited to connectionless internets, for instance SMDS, it is possible that larger and larger numbers of internet users will get their connectivity from large public data networks whose native protocols are not the same as the user's internet protocol. This results in a routing problem that has not yet been addressed. That is, large numbers of routers (potentially tens of thousands) must be able to find direct routes to each other in a robust and efficient way. This paper describes a solution to the problem, called shortcut routing, that incorporates 1) a sparse graph of logical connectivity between routers, 2) hierarchical addressing among the public data network subscribers, and 3) the use of "entry router" information in packets to allow routers to find one hop "shortcuts" across the multi-hop logical graph.

## 1.0 Introduction

The three network layer components of an internet are routers, hosts, and subnetworks. Hosts and routers connect to subnetworks. The subnetwork itself may be a point-to-point link, a multi-point link (such as a LAN), or a general topology network[1]. The subnetwork has its own native protocols and addressing scheme. Hosts and routers exchange internet protocol packets, which cross subnetworks by being encapsulated in the subnetwork's native protocol.

The addresses of the internet protocol are different from those of the subnetwork. While subnetwork addresses only have significance in the context of the single subnetwork, internet addresses have significance across the totality of hosts and routers that use the internet protocol. In certain circumstances the subnetwork address of a host or router may be related to its internet address (for instance, the subnetwork address may be embedded in the internet address). In the general case the two are unrelated.

1. The term subnetwork is different from and should not be confused with the IP term "subnet address".



COMM'92-8/92/MD,USA



When a system (a router or host) needs to send an internet packet, it must determine the destination subnetwork address to send the packet to. (IP systems traditionally do this as a two-step process. First the IP address of the receiving system is determined. Then the subnetwork address associated with the IP address is derived.) On broadcast LANs this has proven to be relatively simple. This is because 1) broadcast LANs have a small number of attached systems (hundreds), and 2) broadcast LANs have an inexpensive multicast, thus making "searching" for systems on a LAN inexpensive and easy.

On very large general topology subnetworks (called here public data networks, or PDNs[2]), however, determining "next hop" subnetwork (or PDN) addresses is not necessarily simple. There may be (eventually) tens of thousands of systems attached to a PDN, making it inefficient to distribute up-to-date information about all systems to all systems. And multicast on a PDN is not efficient and must therefore be limited. Moreover, although any given system should be capable of communicating with any other system, most systems are unlikely to ever communicate with the large majority of other systems, and so should not be expected to maintain information about all other systems.

This paper addresses the problem of routing over large PDNs. In section 2, we give some background to the problem, discussing past solutions such as those used for IP over ARPANET, and why those solutions are not appropriate for the current problem. In sections 3 and 4, variations on the past solutions that are applicable to the current problem are described. These variations do not solve the complete problem.

Section 5 comprises the majority of the paper. It specifies in detail a general solution to the problem whereby the internet addresses of PDN-attached networks are hierarchically clustered with those of other PDN-attached networks. This allows routers to store routes to a large number of networks with just one routing table entry. With hierarchical clustering, however, it may take several logical hops (router to router) across the PDN to reach an exit system. To allow exit systems to be reached in one hop, the PDN address of the entry system is carried along with every internet packet to the exit system. The exit system then uses the PDN address of the entry system combined with the source

2. A large general topology subnetwork may not in fact be "public" in the sense that anybody who can pay for it can use it. However, a public data network typifies the kind of subnetwork of concern in this paper. That is, a large network whose native protocol is not that of the internet.

internet address to deduce the one-hop reverse path. This method of finding one-hop paths is called shortcut routing, and is the major contribution of this paper.

Shortcut routing can be used as an adjunct to existing routing algorithms. Section 5 gives a complete specification of shortcut routing. It describes the characteristics of shortcut routing, including its potential for looping. It also briefly describes the use of hierarchical addressing to achieve scaling, discusses requirements for the logical router-to-router topology, and discusses advantageous ways to form the logical topology.

## 2.0 Background

It is interesting and instructive to consider the evolution of IP in the context of the problem of routing over PDNs. Originally, IP was used to glue together a very small number of subnetworks. The main subnetwork was the ARPANET, and the primary communications over the ARPANET were direct host to host. The number of routers was small enough that each host could be manually preconfigured with the ARPANET address of other routers, and the network numbers of the subnetworks behind them.[3]

The ARPANET address of each system attached to the ARPANET was embedded in the IP address of that system. Therefore, if one host wished to transmit an IP packet to another host, it could pull the ARPANET address out of the IP address and encapsulate and send the IP packet. (This is the first ARPANET solution.)

As the internet grew, particularly with the advent of LANs, the majority of systems attached to ARPANET were routers. Since few hosts were directly attached to the ARPANET (and therefore did not have an address that included the ARPANET address), it was rarely possible for an ARPANET-attached system to know which ARPANET address to use by simply pulling it from the destination IP address.

To solve this problem, the EGP protocol (Exterior Gateway Protocol) was used [Ro]. A small number of routers on the ARPANET were designated "core" routers. All non-core routers advertised the network numbers they could reach to the core routers. The core routers exchanged this information with each other, so that each would know all networks reachable behind all routers. The core routers then told the non-core routers about all other routers. (This is the second ARPANET solution.)

Note that this "route server" technique of a small number of core routers fanning out to a larger number of non-core routers did not reduce the amount of information that each router had to keep. Rather, the route servers reduced the amount of manual configuration required by each non-core router.

Note also that even though EGP advertized the IP network number of each router (along with the network numbers reachable by that router), the significant information being conveyed was the ARPANET address, since it was the ARPANET address, not the IP address, that each router needed to get IP packets across the ARPANET. If the ARPANET address was not embedded in the IP address, EGP would had to have advertized the ARPANET address explicitly. In other words, the fact that the ARPANET address was embedded in the IP address was no longer of any significant advantage (except in sending packets to the few remaining hosts).

The second ARPANET solution is not appropriate for the expected scale of PDNs. By today's standards, the ARPANET was relatively small. It was entirely feasible to advertise all few hundred routers connected to the ARPANET to all other routers (every few minutes!). With potentially tens of thousands of routers connected to a PDN, however, it is not feasible, or at least not desirable, to require all of them to know about all others.

## 3.0 A Useful Variant of the First ARPANET Solution

A variant of the first ARPANET solution, to embed the ARPANET address in the internetwork address, turns out to be applicable to the current problem in certain cases. These cases occur when 1) the internet address is large enough to contain the subnetwork address, such as is the case with the OSI internetwork protocol (ISO 8473, here called CLNP) [OSI3], and 2) the (private) network behind the router is a logical stub network, and is connected to the PDN by only one interface. By logical stub network, we mean a network (itself consisting of routers, hosts, and subnetworks) that cannot be used as a transit network to reach other networks (either for lack of physical connectivity or for policy reasons).

This can work for CLNP because the address in the CLNP header (the NSAP[4] address) is so large (20 octets) [OSI2]. There is therefore room for the NSAP addresses of all hosts and routers in the stub network to contain:

1. any fields needed for inter-domain routing (routing between backbones[5]),

3. IP addresses contain a network or subnetwork number that can only be assigned to systems attached to that subnetwork, plus a host number denoting the specific system on the subnetwork.

4. CLNP—Connection-less Network-layer Protocol. NSAP—Network Service Access Point.

5. A backbone is a transit network used by stub networks to reach other stub networks. A PDN is a type of backbone.

2. the PDN address of the router attached to the PDN, and
3. the information needed to identify and route to destinations inside the stub network.

If the NSAP addresses of all systems in the stub network contain the PDN address of the router attached to the PDN (item 2 above), then any PDN-attached router need only extract the PDN address from the NSAP address to properly route the CLNP packet across the PDN.

This technique is well-known. It is for instance included in the intra-domain routing standard for OSI [OSI1]. While it is extremely useful, it is not the primary technique of interest in this paper because its can't be used in a large number of cases. This embedded-address technique can't be used with IP because the IP address is too short [Pos]. This is also the case for other internet level protocols such as DECNET phase IV [Dec], Appletalk [Apl], and Novell's IPX [Ipx]. It can't be used with stub networks that have more than one interface to the PDN because then one of the interfaces won't be discovered by systems routing to the stub network. Even small stub networks might be expected to have two connections to the PDN for reliability.

Note that the stub network's PDN-attached router must still maintain routing information about all other PDN-attached networks that do not use the embedded-address technique. The following variant of the second ARPANET solution, however, can eliminate the need for this routing information in certain cases.

# 4.0 A Useful Variant of the Second ARPANET Solution

A variant of the "core router" model of the second ARPANET solution is useful in certain cases. This variant can be used with any internet protocol, but has the restriction that the network using it is a physical stub (that is, the network's only access to the internet is via the PDN).

In this variant, there are core routers, here called route servers. All routers advertize their reachable networks, in the form of address prefixes, to the route servers. Unlike the second ARPANET solution, however, the route servers do not advertize reachable address prefixes back to the routers. Instead, when a router X has an internet packet with destination address D to transmit over the PDN, it sends the packet to its route server. The route server forwards the packet back over the PDN to the appropriate exit router Y, and sends a redirect packet to router X. The redirect packet informs router X that the PDN address of router Y is the best next hop for internet packets destined for D. (Note that it is not necessary for X to know the internet address of Y.)

In order to avoid permanent black holes and loops[6], the router must implement certain precautions. It must flush out all redirects after a brief time. The time-out period can be relatively short without causing the router to go back to the route server too often, because the time-out can be refreshed every time a packet with source internet address D is received from the PDN address of router Y. This will prevent permanent black holes. Further, the router should drop all packets received from the PDN that are not destined for one of its own systems. This will prevent loops.

This route server solution has drawbacks. First, it puts a heavy load on the route servers in terms of handling routing updates, storing routes, and forwarding packets and redirecting routers. Thus, the route server is a potential bottleneck. Second, it cannot be used by stub networks that can access the internet via multiple PDNs (or other types of backbones). This is because these multi-homed stubs[7] cannot necessarily establish a simple default route to the route server. Multi-homed stubs may need to know which destinations are best reached via one backbone, and which are best reached via another. Therefore, they need complete routing information from each backbone so that they can compare routes.

# 5.0 A General Solution: Hierarchy and Shortcut Routing

In this section, a general solution to the problem is described. The solution has the following components:

1. Form an internet address hierarchy among subscribers of the PDN, so that PDN-attached routers can view large numbers of PDN-attached stub networks with one routing table entry.

2. Form a sparse logical mesh peer[8] router topology over the PDN, so that each router has a small number of neighbors, but several logical paths to any destination. This logical path results in several hops across the PDN.

3. To shrink the several-hop path into a one-hop path, put the PDN address of the entry system (the system that initially injects a packet into the PDN) in internet packets, so that exit systems can learn of the one hop reverse path "shortcut" across the PDN. This is called shortcut routing.

---

6. A black hole is where packets are repeatedly sent to systems that cannot route them, for instance because they have crashed. A loop is where a packet is repeatedly routed back to a system that has already forwarded it.

7. A multi-homed stub is a stub attached to more than one backbone.

8. By peer, we mean form a direct relationship in the context of a routing algorithm—that is, exchange hello packets, routing updates, and so on.

In section 5.1, the first two components are briefly discussed. Section 5.2 gives a detailed description of shortcut routing.

## 5.1 Internet Hierarchy over a PDN

To reduce the amount of routing information a multi-homed stub must keep, it is necessary to hierarchically cluster the internet addresses of stub networks. To do this, groups of stubs would be assigned internet addresses with the same prefix. For instance, a group of 256 stubs could be given address prefixes 26-0, 26-1, 26-2, ... , 26-255.[9] If all of these stubs exchanged routing information with the same route server (or servers, for redundancy), then the route server could advertize just one prefix, 26, to other route servers.

If more reduction is needed, then another level of hierarchy can be added (26-0-0, 26-0-1, ... , 26-0-255, 26-1-0, 26-1-1, ... , 26-1-255, ... , 26-255-0, 26-255-1, ... , 26-255-255). In this case, roughly 65,000 stubs could be represented by one prefix (for instance, when advertized to a route server with another prefix (27, 28, etc.), or when advertized to routers off of the PDN).

Depending on how many stub routers a given route server could peer with, it may require tens or even hundreds of route servers to handle the 65,000 stubs under prefix 26. These route servers must form peer routing relationships with each other. They do not necessarily each need to become peers with all other route servers, but they must form enough peer relationships such that there is a logical peer router path from any route server to any other route server. Some route servers with prefix 26 must also become peers with route servers with other prefixes such that there is a logical peer router path from any router attached to the PDN to any other router.

Since any two PDN-attached routers can become peers, there is a great deal of freedom in determining which stubs get which prefixes, and in which routers peer with which others. The topics of how to do the address clustering and where to form peer router relationships is not explored in this paper. Previous work ([KK] for clustering, and [Sch] for constructing topologies) covers these topics. In general, address clustering according to geographic proximity is a good strategy. And, the (sparse) logical peer router topology needs to be rich enough that there are multiple logical paths between any two systems, so as to prevent single points of failure. The logical topology also needs to be rich enough to prevent individual routers from becoming bottlenecks.

9. The dash notation (i.e., 26-4) indicates two fields of a hierarchical address. In this example, each field is one byte in length.

### 5.1.1 CLNP and IP

Assigning hierarchical addresses with CLNP is no problem because the NSAP address is so large, and because the OSI routing protocols [OSI1, OSI4] route based on general prefixes. Assigning hierarchical addresses with IP is possible, but only if the traditional class A/B/C structure of IP is ignored, thus allowing for more levels of hierarchy than IP traditionally has. For instance, by assigning class B's out of a contiguous block of class B's to stubs, it would be possible to advertize that block as one prefix. As discussed above, each stub with one of these class B's would need to peer with the same route servers.

Some IP routing protocols, such as OSPF, base routing entirely on general prefixes and therefore don't care about the class A/B/C structure [OSPF]. Other routing protocols, such as EGP [Ro] and BGP [LR], still encode addresses according to the class A/B/C structure. For IP to scale better (both inside and outside PDNs), routing protocols such as EGP and BGP must be modified or phased out.

## 5.2 Shortcut Routing

Consider Figures 1 and 2. For the scaling reasons discussed in section 5.1, the four routers form the sparse logical peer-router topology shown in Figure 1. For instance, Rb might be a route server for a cluster that Sa belongs to, and Rc might be the route server for a cluster that Sd belongs to.

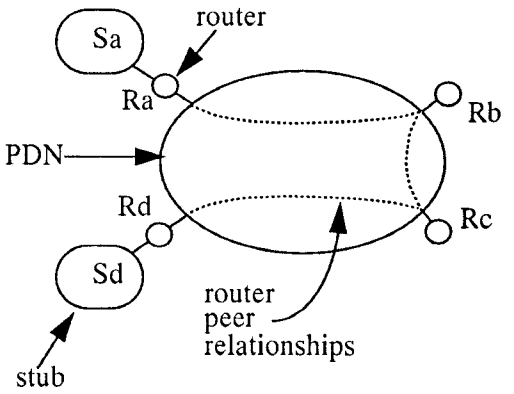



**Figure 1**
**Sparse Peer Router Relationships**

This results in the logical topology shown in Figure 2. Because of the logical topology, router Ra learns from the normal routing algorithm (called the "base" routing algorithm) that addresses in stub Sd are reachable through router Rb. Therefore, the path from stub Sa to stub Sd is Sa-Ra-Rb-Rc-Rd-Sd. This path enters and exits the PDN three times. Ideally, the path should be Sa-Ra-Rd-Sd.

Shortcut routing is used to discover the ideal one-hop path. The idea behind shortcut routing is as follows. An extra

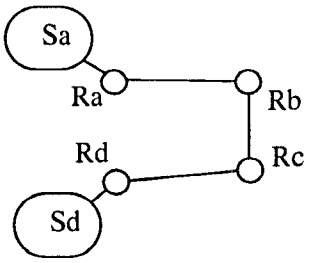



**Figure 2**
**Logical Topology of Figure 1**

layer of protocol is added below the internetwork protocol. This layer contains a single PDN address, called the shortcut address. When a system transmits an internet packet onto the PDN that was *not* received from the PDN, it puts its PDN address into the shortcut address field. When a router transmits an internet packet onto the PDN that *was* received from the PDN, it does not modify the shortcut address field. Therefore, when Rd receives the internet packet from Sa, it sees the reverse one-hop path to Ra, and caches this information. When return packets from Sd to Sa reach Ra via Rd, Ra likewise sees that a direct path is available via Rd. Subsequent packets between Sa and Sd go directly between Ra and Rd.

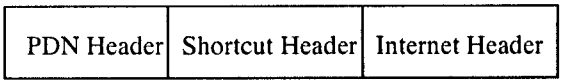
Shortcut routing is not a stand-alone routing algorithm. Rather, it is an enhancement to existing routing algorithms for use over a PDN. While it would be beneficial to modify existing base routing algorithms to account for shortcut routing, it is not necessary. Shortcut routing improves on the paths otherwise found by the base routing algorithm.

### 5.2.1 Detailed Specification of Shortcut Routing

The primary information passed between systems running the shortcut routing function is the shortcut address **sAddrShortcut**, which is the PDN address of the entry system into the PDN. This parameter is passed in a shortcut protocol header in every internet packet that crosses the PDN.[10] It is manipulated by the hosts and routers attached to the PDN, but does not have end-to-end significance as the internet address does. It is not manipulated by the PDN switches. The shortcut protocol header is therefore best positioned between the PDN protocol header and the internet protocol header. [11]

10. Alternatively, it is possible to only attach this field when an internet packet is "put-back" onto the PDN.

11. The shortcut header does not necessarily need to be a separate layer per se (with separate protocol identifiers, encapsulation, and so on). It could be incorporated into the PDN header or possibly even the internet header.




In cases where symmetric paths are not used (that is, the forward and reverse paths across the PDN are not the same), then it is necessary to transmit the sAddrShortcut in a separate packet, called the **shortcutMessage.** The shortcutMessage has parameters <dIAddr; sAddrShortcut>, where sAddrShortcut is the PDN address to which internet packets with destination address dIAddr would be sent.

### System Parameters and Functions

Each system has a **forwardingTable** that is used for forwarding packets along the path found by base routing. Each entry of the forwardingTable has the following information: <iAddrSet; outSubnet; sAddr>. The iAddrSet is a set of reachable *internet* addresses (often encoded as an address/mask pair). The outSubnet denotes the subnetwork that the internet packet should be forwarded over. The sAddr is the *subnetwork* address of the next hop system that was learned via the base routing algorithm. When the router performs the **lookup(iAddr)** function[12] on the forwardingTable, where iAddr is a destination internet address, the forwardingTable entry with the smallest iAddrSet such that iAddr is a member of iAddrSet is returned (or NULL if there is no such entry).

Each system has a **shortcutTable** that is used for forwarding packets along the path found by shortcut routing. Each entry of the shortcutTable has parameters <iAddr; sAddr; lastRxTime; txCount>. iAddr is an internet address, derived from the source internet address of incoming packets. sAddr is the PDN address to which internet packets with address iAddr should be sent. lastRxTime and txCount are used to flush the shortcutTable entry to prevent loops and black holes. When the system performs the **lookupShortcut(iAddr)** function on the shortcutTable, the shortcutTable entry with matching iAddr is returned (or NULL if there is no match).

The function **createShortcut(iAddr, sAddr)** adds an entry to the shortcutTable. When the entry is created, lastRxTime is set to the **currentSystemTime**, and txCount is set to 0. In addition, a **shortcutTimer** is started, which is scheduled to expire in time **HOLDSHORTCUTTIME**. This timer is used to remove out-of-date information that may no longer be optimal. The function **flushShortcut(iAddr)** removes the entry with matching iAddr from the shortcutTable.

Each system has a **reversePathTable** that is used for establishing shortcuts in the case of asymmetric paths. Each entry of the reversePathTable has parameters <iAddr;

12. Some routing algorithms may include other information in the lookup function, such as source address or quality-of-service parameters. This specification assumes only destination address based routing.

sAddr; rxCount; rxThreshold>. iAddr is an internet address (derived from the destination internet address of incoming packets). sAddr is the subnetwork address of the system from which the incoming internet packet was sent. rxCount and rxThreshold govern the sending of shortcutMessages. When the system performs the **lookupReverse(iAddr, sAddr)** function on the reversePathTable, the entry with matching iAddr and sAddr is returned (or NULL if there is no match).

The function **createReversePath(iAddr, sAddr)** adds an entry to the reversePathTable, and sets rxCount to 0 and rxThreshold to the constant **RXTHRESHOLD**. The function **flushReversePath(iAddr, sAddr)** removes the entry with matching iAddr and sAddr from the reversePathTable.[13]

Internet packets are received with parameters <iSAddr; iDAddr; inSubnet; sAddrRemote; sAddrLocal; sAddrShortcut>, where iSAddr and iDAddr are the source and destination internet addresses, inSubnet is the subnetwork over which the internet packet was received, sAddrRemote is the subnetwork address of the system that sent the internet packet, sAddrLocal is the subnetwork address of the interface over which the packet was received, and sAddrShortcut is the subnetwork address in the shortcut field. sAddrShortcut is NULL for internet packets not received over the PDN.

### Shortcut Algorithms

When a system receives an internet packet from the PDN (i.e., packet[inSubnet] = PDN), it executes the algorithm in Figure 4.

In lines 1 and 2, the lookup operation is executed and the packet is forwarded (this would be normal packet processing without shortcut routing). Note that the packet is sent to the next hop learned by base routing, not the one learned by shortcut routing. Doing this prevents any intra-PDN loops that might otherwise have formed as a result of old shortcut information. Note also that the packet is forwarded without modification to the shortcut header.

If the packet is to be routed off the PDN and the shortcut is not from a peer router (lines 3 and 4), then the reverse path is recorded in shortcutTable (line 7 or 11). This will be used to route return packets (assuming symmetric paths). In addition, the black hole detection parameters txCount and lastRxTime are updated (line 7 or lines 9 and 10). Finally, the checkAsymmetricRoutes(packet) operation (line 14), given in Figure 5, is executed to handle the case where asymmetric routes are being used and therefore a shortcutMessage may need to be sent.

13. No timer is specified for reversePathTable entries. It is assumed that the system has an appropriate garbage collection mechanism.

The checkAsymmetricRoutes(packet) operation is illustrated by Figure 3. When a host in Src sends an internet packet to a host in Dest, Rd records the reverse path to Src as being through Rf. However, the return path does not go through Rd, and so Rd's recorded information is of no use. In addition, Rf does not receive return packets, and so cannot learn via the shortcut header of the direct path to Rd.

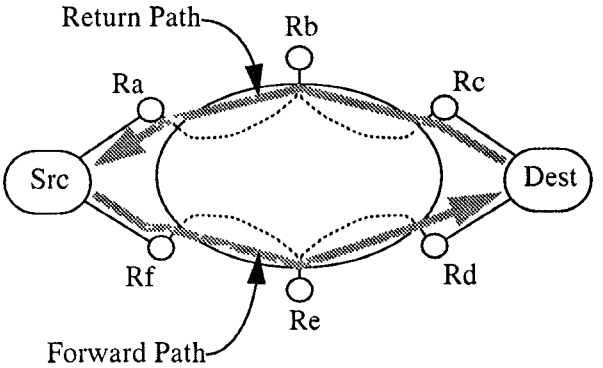



**Figure 3**
**Shortcut Routing with Asymmetric Paths**

Each time Rd receives a multi-hop (within the PDN) packet from Rf via Re, it (normally) increments the rxCount for the reversePathTable entry associated with the destination internet address and Rf's PDN address (line 10). If rxCount exceeds a threshold (rxThreshold, line 5), Rd transmits a shortcutMessage to Rf (line 6), telling it of the direct path. Rd also resets rxCount (line 7) and doubles the value of the threshold (line 8). Doubling the threshold prevents excessive sending of shortcutMessages in the case where the recipient is ignoring them. The threshold should be initially set to a small number, say 3 (RXTHRESHOLD, line 3 and line 14). This gives return packets a chance to convey the shortcut in the case where paths are symmetric and the shortcutMessage is not needed. Each time Rd receives an internet packet directly from Rf for the given destination, indicating that Rf has stored the shortcut, it resets the rxCount and rxThreshold (lines 13 and 14).

When a system receives an internet packet from a subnetwork other than the PDN (i.e., packet[inSubnet] ≠ PDN), it executes the algorithm in Figure 6.

First, the system searches the shortcutTable for a next hop PDN address (line 1). Note that this is different from the case where a packet is received from the PDN, where the forwardingTable is first searched.

If there is no shortcutTable entry for the destination (line 2), the packet is forwarded using the base routing information in the forwardingTable (lines 3 through 6). Note that the shortcut header is filled in with the system's PDN address if the packet is being forwarded over the PDN (lines 4 and 5). If there is a shortcutTable entry (line 7), then the packet is transmitted to the next hop PDN address in the

shortcutTable (again after filling in the shortcut header, line 8).

In lines 10 through 14, the router determines if the shortcut entry should be flushed. If return-path internet packets have not been received for some relatively short time (SHORTTIME, line 11), and a certain small number of packets have been transmitted in that time (TXTHRESHOLD, line 10), then the shortcut is flushed. This check allows shortcuts where the source is transmitting repeatedly, but there is no return traffic (presumably because packets are not reaching the destination, but possibly simply because the destination is not returning packets) to be flushed quickly. A shortcut will also be flushed if no return packets have been received for a relatively long time (LONGTIME, line 13), whether or not packets have been transmitted. This covers the case where a source is transmitting occasionally, but there is no return traffic. If the paths are asymmetric, however, this check will periodically flush the shortcut unnecessarily. Since the extra cost of re-discovering the shortcut is relatively cheap (one packet), this is acceptable behavior.

Typical parameters for the three settings are SHORTTIME = 5 seconds, TXTHRESHOLD = 10 packets, and LONGTIME = 1 minute. For the case where asymmetric paths are used, TXTHRESHOLD should be set to a larger value, perhaps several hundred, since no return packets will be received. The shortcutTimer will flush the shortcut in time HOLDSHORTCUTTIME whether or not packets are received over the shortcut. A typical value for HOLDSHORTCUTTIME is 15 minutes.

Note that this particular algorithm for flushing (using txCount and lastRxTime) is one of many possible ones. Experience may determine that a simpler but less responsive algorithm is adequate, for instance if it is found that loops or black holes occur rarely. Different routers may employ different algorithms. It is entirely a local matter.

In addition to the above shortcut flushing mechanisms, a router may optionally wish to flush a shortcut when base routing detects that the path to a destination that has a shortcut in effect has changed. Note that different routing algorithms have differing amounts of knowledge about the path to the destination. For instance, simple distance-vector routing algorithms know only the next hop to the destination, while link-state routing algorithms may know the entire path.

### 5.2.2 Looping in Shortcut Routing

Even though base routing may be loop-free, it is possible for loops to form when shortcut routing is used in conjunction with base routing. This is because part of a path from source to destination may be routed using information from base routing, and part from shortcut routing. If the base routing information from which the shortcut route was derived is no longer valid, a loop may form.

These loops, however, can only form if part of the looping path goes outside the PDN. In other words, purely intra-PDN loops cannot form. This is because all internet packets received from the PDN are routed using base routing. To prove that intra-PDN loops cannot form, assume that an intra-PDN loop does exist, where R1 is the entry router. Only R1 will route the packet using shortcut routing information. The looping path must eventually come back to R1, where it will be routed using base routing information (rather than shortcut routing information, as it did upon entering the PDN). The remainder of the path will follow base routing and will therefore not loop.

All looping can be eliminated with the following two restrictions:

1. If there exists a path[14] between two routers R1 and R2 that exits R1(R2) via a non-PDN interface and enters R2(R1) via a non-PDN interface, then R1 and R2 must be peer routers.
2. A system must never establish a shortcut to a routing peer.

The reason that these restrictions eliminate loops is as follows. Since we have eliminated the possibility of intra-PDN looping, one segment of the path (say R1->R2) must directly cross the PDN, and the R2->R1 segment must exit R2 on a non-PDN link, and enter R1 on a non-PDN link. (This segment may in fact enter and exit the PDN multiple times. This does not effect the argument.) By the first restriction, however, R1 and R2 must be routing peers if such a path exists. And by the second restriction, if R1 and R2 are routing peers, then R1 would not ever establish a shortcut to R2. If there is a loop, R1 would not send the packet to R2 because of base routing. Therefore, a loop cannot form if the above restrictions are followed.

The first restriction does not necessarily result in a large number of router peers for most routers. Most networks that connect to a PDN are stub networks. These do not need to setup peer router relationships for the sake of preventing loops. Backbones need to establish peer router relationships with the routers of other backbones, but backbones would normally want to do this anyway, due to the volume of traffic that passes between them (and the desire not to incur the overhead of shortcut routing on this traffic). And, the number of backbones is small relative to the number of stubs.

Some networks may be predominantly stub, but may transit traffic for a small number of other networks. Assuming that routing algorithms always prefer internal paths over external paths, then two such stubs A and B only need to establish router peers if they both transit traffic for the same third network C. Therefore, even most predominantly stub

14. This is a path that base routing can find. In other words, a physical path between R1 and R2 that base routing is configured to not use doesn't count.

networks do not need to establish router peers with each other.

It will not always be possible to insure that the above restriction holds, and therefore loops will occasionally occur. In addition, black holes may sometimes form, because a previously valid path used by a shortcut becomes invalid. In these cases, the shortcut will be flushed fairly quickly (more quickly if the paths are symmetric) through the reverse path refresh mechanisms, thus eliminating the loop or black hole.[15]

### 5.2.3 Discussion of Shortcut Routing

One of the negative attributes of shortcut routing is that it relies on caching per-source/destination address pair "flows". If there are many simultaneous flows, the amount of memory needed to cache the shortcuts is large. Recent measurements show that the number of simultaneous flows for a representative stub network is small (tens or low hundreds) [Pax]. While backbones will have more simultaneous flows, their flows with other backbones are not subject to shortcut routing. Therefore, backbones need only cache flows to stubs. While this may still be a large number of flows, we note that a shortcut can always be ignored (at the expense of a longer path) if there is not enough memory to store it (or, it can be stored in lieu of an idle shortcut). Therefore, a router does not necessarily need to keep enough cache entries for its peak load.

Another negative attribute of shortcut routing is that it can result in out-of-order internet packets (even in the case where the PDN preserves ordering). This is because at the time a shortcut is discovered, other internet packets may still be in transit via the longer path. When a packet is sent via the shortcut, it may arrive before the other packets already in transit.

A potential way to avoid out-of-sequence packets is to not activate a known shortcut until previously sent packets have exited the PDN with high probability. If the statistical delay characteristics of the PDN are known, then it suffices to monitor the time of the last packet sent, and to not use the shortcut until enough time has elapsed.

The shortcut routing algorithm described in this paper assumes a connectionless PDN. If the PDN is connection-oriented, different procedures may apply. For instance, mechanisms like the X.25 call redirect may be used to discover the shortcut upon call setup. In this case, it is only necessary to carry the shortcut information in the call setup packet. In some cases, it may be necessary to tear down an existing connection and setup a new one in order to take advantage of a shortcut.

The shortcut routing algorithm described in this paper assumes a decoupling between base routing and shortcut routing. This allows us to easily enhance existing routing algorithms with shortcut routing. Some aspects of shortcut routing would be improved, however, if the base routing algorithm could be modified to account for shortcut routing. For instance, base routing could avoid incrementing distance metrics in the case where routes across the PDN are advertized back over the PDN (for distance-vector type algorithms), or could avoid incrementing locally calculated path cost in a shortest path algorithm (for link-state type algorithms).[16] Base routing could also always find symmetric paths, thus eliminating the need for the checkAsymmetricRoutes() operation and the extra shortcutMessages.

The internet protocol itself could be enhanced to support shortcut routing. If the internet address of the entry system could be carried in the internet protocol even after the internet packet exits the PDN, then all loops resulting from shortcut routing could be detected (except those that expire because of hop count). Given that loops should be rare in any event, this feature is of questionable merit.

It may be impossible to add the shortcut header to existing PDNs, or it may be determined that the overhead of putting a shortcut header in each packet is excessive (although the author does not think so). An alternative to the shortcut header approach to shortcut routing is to have each router monitor when it routes an internet packet back over the PDN from which it was received. The router can then send a "initiate shortcut search" message to the PDN address from which it received the internet packet. The originator of the internet packet could then send a "shortcut search" message over its base routing path. The "shortcut search" message would include the originator's PDN address. When the "shortcut search" message reached the exit system, a "shortcut report" message could be sent back to the originator, informing the originator of the exit system's PDN address.

This type of solution was explored by the author, and found to be workable. But because this shortcut discovery method is so much more expensive (at least four extra packet transmissions, vs. usually none for the specified scheme), it is necessary to be much more selective about flushing shortcuts. This resulted in a significantly more complex protocol, and increased the delay in discovering loops.

The shortcut routing algorithm described in this paper has a privacy hole. Because a system will believe any shortcut information sent to it, the following scenario can occur.

15. It is theoretically possible that a black-hole or loop exists in one direction of a symmetric path, while the opposite direction is still good. Only the shortcutTimer will remove this black hole. This type of black hole or loop should be rare.

16. Even without this mechanism, it may be useful to keep the metrics for PDN crossings artificially low to more accurately reflect the ultimate (single-hop) cost of crossing the PDN.

Assume that two systems, X and Y, have established a shortcut to each other. A third system, Z, sends shortcut headers to X and Y, indicating itself as the next hop. X and Y will subsequently send their packets to Z, where Z can read the contents of the packet, and forward the packet appropriately. X and Y will not detect the privacy violation in this case.

The solution to this problem is to only record shortcut information that has come from base routing. This can be done by adding an "I have learned the shortcut" bit to the shortcut header. An entry system sends its packets via base routing unless the exit system indicates via the bit that it has learned the shortcut. On a typical packet exchange, this will cause two packets to take the longer base routing path instead of just one, as is the case with the algorithm described in this paper.

The version of shortcut routing being specified for the IP Internet in the Internet Engineering Task Force (IETF) uses this bit [TL].[17] The IETF version of shortcut routing has several other differences from the algorithm in this paper. It eliminates the shortcutMessage on the assumption that asymmetric paths are rare enough that the extra complexity is not justified. It has a simpler algorithm for flushing old shortcuts. And, it allows for operation with non-shortcut systems and for operation with routers attached to multiple PDNs.

## 6.0 Summary and Conclusions

With the emergence of large PDN data services that are designed with internet users in mind, the potential scope of the problem of running internet protocols over large PDNs has increased dramatically. As a result, previous approaches to the problem, while still useful, are not adequate.

The basic problem is that there may be thousands of internet systems attached to a single PDN. Every system cannot be expected to continuously maintain routing information for every other attached system. Embedding the PDN address in the internet address helps in the case of singly-attached stub networks, but many PDN-attached networks may not fall in this category, and most internet addresses are too small to hold the PDN address.

This paper suggests an approach to the problem whereby PDN-attached networks are hierarchically clustered with other PDN-attached networks. This allows routers to store routes to a large number of networks with just one routing table entry. With hierarchical clustering, however, it may take several logical hops (router to router) across the PDN to reach an exit system. To allow exit systems to be reached in one hop, the PDN address of the entry system is carried along with every internet packet to the exit system. The exit system then uses the PDN address of the entry system combined with the source internet address to deduce the one-hop reverse path. This method of finding one-hop paths is called shortcut routing.

Shortcut routing can be used as an adjunct to existing routing algorithms. This paper gives a complete specification of shortcut routing. It describes the characteristics of shortcut routing, including its potential for looping. It also briefly describes the use of hierarchical addressing to achieve scaling, discusses requirements for the logical router-to-router topology, and discusses advantageous ways to form the logical topology.

Shortcut routing elegantly solves the problem of scaling over large PDNs. The main advantages of shortcut routing are that it is simple and that it can be used with existing routing algorithms. The main disadvantages of shortcut routing are that it can loop, and that it requires caching on a per source/destination address pair basis. The looping, however, can be avoided in almost all cases, and is short-lived when it occurs. In most cases, the amount of caching will be small. And since not caching a shortcut results in a non-optimal path rather than no path at all, undersizing the cache is not fatal.

Shortcut routing has not yet been implemented, so a considerable amount of work remains, especially in terms of measuring its performance over real networks. A prototype implementation is currently under way. In addition, shortcut routing is being specified in the IETF for various internet protocols running over various PDNs.

Acknowledgments

I would like to thank John Garrett of AT&T, John Hagen of the University of Pennsylvania, and John Chang of US West for the many insightful discussions of internet routing over PDNs that led to this paper.

REFERENCES

[Apl] Sidhu, G.S. et. al., "Inside AppleTalk", Addison Wesley, Reading, Massachusetts, 1989.

[LR] Lougheed, K., Rekhter, Y., "Border Gateway Protocol (BGP)", RFC-1163, USC/Information Sciences Institute, June 1990.

[Dec] Malamud, C., "Analyzing DECnet OSI Phase V", Van Nostrand Reinhold, New York, 1992.

[Ro] Rosen, E.C., "Exterior Gateway Protocol (EGP)", RFC-827, USC/Information Sciences Institute, October 1982.

[Ipx] Malamud, C., "Analyzing Novell Networks", Van Nostrand Reinhold, New York, 1991.

17. This document is currently an Internet-Draft, but is expected to become an RFC and an Internet-Standard in the future.

[KK] Kamoun, F., Kleinrock, L., "Hierarchical Routing for Large Networks", Computer Networks 1 (1977), North-Holland Publishing Co.

[OSI1] International Organization for Standardization ISO10589, "Intermediate System to Intermediate System Intra-Domain routeing exchange protocol for use in Conjunction with the Protocol for providing the Connectionless-mode Network Service (ISO 8473)"

[OSI2] International Organization for Standardization ISO8348, Addendum 2, "Network Layer Addressing"

[OSI3] International Organization for Standardization ISO8473, "Protocol for providing the Connectionless-mode Network Service"

[OSI4] International Organization for Standardization CD10747, "Protocol for Exchange of Inter-domain Routeing Information among Intermediate Systems to Support Forwarding of ISO 8473 PDUs"

[OSPF] Moy, J., "OSPF Specification", RFC-1131, USC/ Information Sciences Institute, October 1989.

[Pax] Paxson, V., "Measurements and Models of Wide Area TCP Conversations", LBL-30840, Lawrence Berkeley Laboratory, Berkeley, CA, USA, May 1991

[Pos] Postel, J.B., "DoD Standard Internet Protocol", RFC-760, USC/Information Sciences Institute, January 1980.

[Sch] Schwartz, M., "Computer Communication Network Design and Analysis", Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1977.

[TL] Tsuchiya, P., Lawrence, J., "IP Routing and Discovery over SMDS", IETF Internet Draft, available via Anonymous FTP at nnsc.nsf.net, file internet-drafts/draft-ietf-iplpdn-shortcutrouting-00.txt.

```
1  forwardEntry ⇐ lookup(packet[iDAddr])
2  transmit internet packet to forwardEntry[sAddr]
3  IF forwardEntry[outSubnet] ≠ PDN
4      IF packet[sAddrRemote] ≠ packet[sAddrShortcut]
5          shortcutEntry ⇐ lookupShortcut(packet[iSAddr])
6          IF shortcutEntry = NULL
7              createShortcut(packet[iSAddr, sAddrRemote])
8          ELSE
9              shortcutEntry[txCount] ⇐ 0
10             shortcutEntry[lastRxTime] ⇐ currentSystemTime
11             shortcutEntry[sAddr] ⇐ packet[sAddrRemote]
12         FI
13     FI
14     checkAsymmetricRoutes(packet)
15 FI
```

**Figure 4: Receive Internet Packet from PDN**

```
1  reverseEntry ⇐ lookupReverse(packet[iDAddr, sAddrShortcut])
2  IF reverseEntry = NULL
3      createReversePath(packet[iDAddr, sAddrRemote])
4  IF packet[sAddrRemote] ≠ packet[sAddrShortcut]
5      IF reverseEntry[rxCount] > reverseEntry[rxThreshold]
6          transmit shortcutMessage(packet[iDAddr], localPDNAddr) to packet[sAddrShortcut]
7          reverseEntry[rxCount] ⇐ 0
8          reverseEntry[rxThreshold] ⇐ reverseEntry[rxThreshold] x 2
9      ELSE
10         reverseEntry[rxCount] ⇐ reverseEntry[rxCount] + 1
11     FI
12 ELSE
13     reverseEntry[rxCount] ⇐ 0
14     reverseEntry[rxThreshold] ⇐ RXTHRESHOLD
15 FI
```

**Figure 5: checkAsymmetricRoutes(packet) Operation**

```
1  shortcutEntry ⇐ lookupShortcut(packet[iDAddr])
2  IF shorcutEntry = NULL
3      forwardEntry ⇐ lookup(packet[iDAddr])
4      IF forwardEntry[outSubnet] = PDN
5          packet[sAddrShortcut] ⇐ localPDNAddr
6      transmit internet packet to forwardEntry[sAddr]
7  ELSE
8      packet[sAddrShortcut] ⇐ localPDNAddr
9      transmit internet packet to shortcutEntry[sAddr]
10     IF shortcutEntry[txCount] > TXTHRESHOLD
11     AND currentSystemTime - shortcutEntry[lastRxTime] > SHORTTIME
12         flushShortcut(iDAddr)
13     IF currentSystemTime - shortcutEntry[lastRxTime] > LONGTIME
14         flushShortcut(iDAddr)
15 FI
```

**Figure 6: Receive Internet Packet from other than PDN**