Discovering Internet Topology[④]

R. Siamwalla, R. Sharma, and S. Keshav
Cornell Network Research Group
Department of Computer Science
Cornell University, Ithaca, NY 14853
{rachit, sharma, skeshav}@cs.cornell.edu

Abstract
In large and constantly evolving networks, it is difficult to determine how the network is actually laid out. Yet this information is invaluable for network management, simulation, and server siting. Traditional topology discovery algorithms are based on SNMP, which is not universally deployed. We describe several heuristics and algorithms to discover both intra-domain and Internet backbone topology while making as few assumptions about the network as possible. We quantitatively evaluate their performance and also present a new technique for visualizing Internet backbone topology.

1. Introduction

Network topology is a representation of the interconnection between directly connected peers in a network. In a *physical* network topology, peers are ports on devices connected by a physical transmission link. A physical topology corresponds to many *logical* topologies, each at a different level of abstraction. For example, at the IP level, peers are hosts or routers one IP hop from each other, and at the workgroup level, the peers are workgroups connected by a logical link. In this paper, by network topology we refer exclusively to the logical IP topology, ignoring hubs and bridges, and link-level details such as FDDI token rotation times, ATM or Frame-relay links, and Ethernet segment lengths. At this level, a peer corresponds to one or more IP addresses, and a link corresponds to a channel with specific delay, capacity, and loss characteristics.

Network topology constantly changes as nodes and links join a network, personnel move offices, and network capacity is increased to deal with added traffic. Keeping track of network topology manually is a frustrating and often impossible job. Yet, accurate topology information is necessary for:

- *Simulation*: In order to simulate real networks, the topology of the network must be first obtained.
- *Network Management*: Network topology information is useful in deciding whether to add new routers and to figure out whether current hardware is configured correctly. It also allows network managers to find bottlenecks and failures in the network.
- *Siting:* A network map helps users determine where they are in the network so they can decide where to site servers, and which ISP to join to minimize latency and maximize available bandwidth.
- *Topology-aware algorithms*: Topology information enables a new class of protocols and algorithms that exploit knowledge of topology to improve performance. Examples include topology-sensitive policy and QoS routing, and group communication algorithms with topology-aware process group selection.

Thus, there is a considerable need for automatic discovery of network topology. Currently, the only effective way to do so is by exploiting SNMP (Simple Network Management Protocol). However, there are many situations where SNMP cannot be used. SNMP is not implemented in most older machines, and in newer machines, SNMP may be turned off or have restricted access. In a growing heterogeneous network, where decisions about the network and access are decentralized, it is naive to assume that SNMP has been installed, and is accessible, on every node in the network. For example, in the `cornell.edu` domain, SNMP can only discover 8% of the hosts. We think that this situation is the norm, rather than the exception.

[④] Submitted to IEEE INFOCOM'99

2. Goals

The goals of our work are to automatically discover network topology both within a single administrative domain, and in the Internet backbone, while making as few assumptions as possible about the network. In particular, we do not assume that SNMP is globally available, or that the discovery tool is allowed to participate in routing protocols such as OSPF or DVMRP. Moreover, we would like our algorithms to be:

- *Efficient*: impose the least possible overhead on the network,
- *Fast*: take the least possible time to complete the job,
- *Complete:* discover the entire topology, and
- *Accurate*: not make mistakes

Because every discovery algorithm represents a trade off between these competing goals, we present a suite of algorithms that make a range of tradeoffs, and is each suited to a different operating environment.

3. Background and tools

The Internet is divided into several thousand administrative *domains*. All hosts, routers, and links in a domain are administered by a single entity, and are addressed by IP addresses that share the same common prefix, also called the *network number*. Within each domain, IP addresses are further grouped by *subnets*, so that all IP addresses within a subnet share the same subnet prefix. The *subnet mask* identifies the number of bits in an IP address that correspond to the subnet number. For example, the Cornell domain owns, among others, the 128.84 network. This is further subdivided into subnets, including the 128.84.223 subnet and the 128.84.155 subnet. All IP addresses in each of these subnets starts with the subnet number, and has a subnet mask of 255.255.255, which indicates that the subnet number is 24 bits long.

Routing between subnets is accomplished by routers. A router is simultaneously present on multiple subnets, and therefore has multiple IP addresses. The standard convention is to give routers the first address in each subnet range. For example, a router that connects subnets 128.84.223 and 128.84.155 would have two IP addresses, 128.84.223.1 and 128.84.155.1.

The algorithms discussed in this paper are based on three widely available tools:

- Ping/Broadcast Ping

  Every IP host is required to echo an ICMP 'ping' packet back to its source. The ping tool therefore accurately indicates whether the pinged machine is on the Internet or not (actually, since ping packets can get lost, we always ping an address twice, deeming it unreachable only if both do not elicit a reply). With suitably small packets, ping also has a low overhead. Pings to live hosts succeed within a single round-trip time, which is a few tens of milliseconds, so the tool is fast. Pings to dead or non-existent hosts, however, timeout after a conservative interval of 20 seconds, so pings to such hosts are expensive.

  `Directed broadcast ping' refers to a ping packet addressed to an entire subnet rather than just one machine. This can be done by addressing either the '255' or the '0' node in the subnet (e.g. to broadcast to all nodes in the 128.84.155 subnet, ping 128.84.155.0 or ping 128.84.155.255—more generally, these two addresses corresponding to extending the subnet address either with all 0s or all 1s). A broadcast ping is received by all hosts in the subnet, each of which is supposed to reply to originator of the ping. This is useful in finding all the machines in a subnet. Ping broadcast however is not supported fully in all networks. In some networks, only the router responsible for that subnet responds to the broadcast ping (we refer to this as the *weak ping broadcast assumption*). In other networks, broadcast ping is not even responded to at all. These modifications prevent a denial-of-service attack called "smurfing" where a large subnet is broadcast with a ping packet whose return address is set to that of the victim. The victim gets swamped with ICMP ping replies and soon dies.

- Traceroute

  Traceroute discovers the route between a probe point and a destination host by sending packets with progressively increasing TTLs. Routers along the path, on seeing a packet with a zero TTL, send ICMP TTL-expired replies to the sender, which tallies these to discover the path. Traceroute is usually accurate because all Internet routers are required to send the TTL-expired ICMP message. However, some ISPs are known to hide their routers from traceroute by manipulating these replies to collapse their internal topology. This reduces both the accuracy and the completeness of topologies discovered using traceroute. Traceroute sends two probes to every router along the path, so it generates considerably more overhead than ping. Since probes to consecutive routers are spaced apart to minimize the instantaneous network load, the time to complete a traceroute is also much longer than a ping.

- Zone transfer from a DNS server

  A domain's DNS name server keeps a binding from every name in the domain to its IP address. Most DNS servers respond to an 'zone transfer' command by returning a list of every name in the domain. Thus DNS zone transfer is useful in finding all hosts and routers within a domain. This technique has low overhead, is fast and is accurate. It may not be complete, however, since hosts obtaining IP addresses using DHCP are not served by DNS. Moreover, some network managers disable DNS zone transfer due to security concerns.

The table below summarizes the performance characteristics of the three tools.

	Ping	**Traceroute**	**DNS zone transfer**
Applicability	All domains	All domains	Most domains
Overhead	Low	High	Low
Speed	Fast for live hosts; Slow otherwise	Slow	Fast
Accuracy	Accurate	Usually accurate	Usually accurate

4. Heuristics

We now describe three heuristics that we will use in the algorithms presented in Section 5.

4.1 Subnet guessing using broadcast pings

The first heuristic, when given an IP address, determines the length of the address mask associated with that address as follows:

```
for masklen = 31 to 7 do
        a. assume network mask is of length masklen
        b. construct the '0' and '255' directed broadcast addresses for that address and masklen
        c. ping these directed broadcast addresses
        d. if more than two hosts reply to both pings then return masklen  else continue
```

The idea is to test progressively decreasing mask lengths by pinging broadcast addresses corresponding to these mask lengths. A reply to a ping indicates that the guessed mask length is correct. The reason for sending a ping to both broadcast addresses is because otherwise we might make the following error. Suppose we are trying to guess the network mask for the address 171.64.255.16 with a ping to only the '255' broadcast address. Suppose also that the true mask length is 16 bits. When we probe a mask length of 24 bits we will ping 171.64.255.255, which happens to be the broadcast address for the 171.64 subnet. Since we get multiple replies for this address, we erroneously conclude that the mask length is 24 bits. We can avoid this by trying both broadcast addresses. Now, the broadcast to 171.64.255.0 will elicit no replies, so that we do not make this mistake. Notice that in step 1.d we look only for a non-zero number of broadcast ping replies. This allows us to deal with cases where a router sends a single ping reply on behalf of the subnet in response to a broadcast ping.

This heuristic can be used in a domain that supports pings to broadcast addresses. Although it is accurate, because it sends a series of pings, it is both slow and has a high per-address overhead.

4.2 Subnet guessing from a cluster of addresses

Suppose we know that hosts with addresses A1, A2, and A3 are all one hop away from a router whose closest interface to these hosts has IP address A. The goal of this heuristic is to determine the address of the subnet that A1, A2, and A3 belong to, and the corresponding subnet mask (see Figure 1). The idea is that the bitwise AND of A, A1, A2, and A3 approximates the subnet number because all four addresses must share this number as a common prefix, and changes in the remaining bit positions ought to, on average, cancel out. For example, assume that the router's interface address is 128.84.155.192 and the three host addresses are 128.84.155.195,

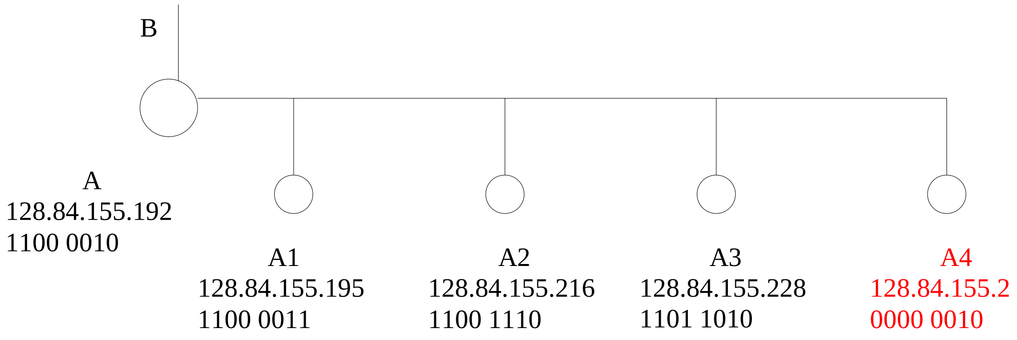

Figure 1

128.84.155.216, and 128.84.155.228. To clarify our presentation, we will omit the first three bytes of the address and represent the last byte in binary. With these modifications, the router's address is 1100 00010, and the three host addresses are 1100 0011, 1101 1110, and 1101 1010. The bitwise AND of these addresses is 1100 0010, which is our initial guess for the subnet number. We then compute the bitwise OR of the host and router addresses, because the subnet should cover at least these bits in the address space. Here, the bitwise OR is 1101 1111, which indicates that the subnet address space should include at least the last five bits of the address space (leading 1's can equally well be part of the subnet's network number). We can now refine our choice of the subnet number. Because subnet masks must be contiguous, the subnet number cannot be 1100 0010, so the last five bits of the mask should be of the form 0 0000. We now have four choices for the subnet number (assuming that the preceding three bytes are 128.84.155): 110, 11, 1, and null. Given only addresses A, A1, A2, and A3, we cannot make further progress. However, suppose that we discover that address A4 = 128.84.155.2 (0000 0010 in our notation) also can be reached by a one-hop path from the router. Then, the bitwise AND becomes 0000 0010, eliminating subnet choices of 110, 11, and 1. We can now confidently state that the subnet number must be null = 128.84.155, and the subnet mask must be 255.255.255.0.

This heuristic is applicable in all domains, is fast, and imposes no additional overhead. It is accurate if the hosts in a subnet are widely dispersed in the subnet's address space, so that the bitwise AND of the addresses results in outliers that can be eliminated by the contiguity rule. It is encouraging to note that almost all subnets have this property. If all hosts lie in the higher end of the address space, the heuristic cannot unambiguously decide on the subnet number and mask.

4.3 Guessing valid addresses in a domain

Our third heuristic is a way to choose 32-bit values that, with good probability, lie within a chosen subnet's address space. It is derived from the following observations in the `cornell.edu` and `cs.cornell.edu` domains (in decreasing order of generality). See the Appendix for a more detailed description of these observations.

- For each host with address a.b.c.d that responded to a ping, the subnet included a `corresponding' address of the form a.b.c.1, which usually was the gateway router. This corresponds to a subnet mask of 255.255.255.0 (24 bits long).
- Most hosts that respond to a ping are close to each other in the address space (for example, if 128.84.227.50 exists, there is a high chance that 128.84.227.51 exists)
- Some hosts that responded to ping had a corresponding address that ended in *.129, which was often a router. This was when the subnet mask is 25 bits long.
- Some hosts that responded to ping had a corresponding address that ended in *.65 and *.193, which was often a router. This is usually when the subnet mask is 26 bits long.

Based on these observations, we use the following heuristic to populate a 'temporary set' of addresses likely to be in domain's address space (we use 'ping' to discard invalid addresses from this set):

```
foreach address successfully pinged
        add the next N consecutive addresses to temporary set
        if (address ends in 1, 63, 129, or 193) // a router: may have other hosts in this space
                add N random addresses with the same prefix to the temporary set.
```

The choice of N determines how aggressively the heuristic populates the address space. If N is high, it finds all live hosts, but also many invalid addresses; if N is low, most guesses are valid, but not all hosts may be found. All results presented in this paper use an N value of 5.

5. Algorithms

5.1 Basic algorithm

The discovery algorithms described in this section are variations on the following algorithm:

```
1. Determine a 'temporary' set of IP addresses that may or may not correspond to actual hosts and routers.
2. For each element of the temporary set:
   a. Validate the address
   b. If the address is valid, find out how it relates to other addresses already in the permanent set, and add it
      to the permanent set.
   c. Use this address to generate more IP addresses and add them the temporary set.
```

When it completes, a discovery algorithm generates a topology representation consisting of lists of hosts, routers, and subnets. Each item may be associated with additional information such as the host name, or the number and type of interfaces present at each route. (At the moment, we do not discover link information such as capacity and delay: this is the subject of future work.) The information is stored as a simple directory tree in the Unix file system, where each directory node in the tree is a subnet or a router, and each file is either a host or a description of a subnet or router. This concisely represents topology information in an easily retrievable form. Better yet, simple shell scripts allow us to perform complex operations on the topology. Recently the IETF has proposed the PTOPO MIB information structure to store physical topology information [IETF 97]. Information in our directory structure can be easily converted to this format.

We now present a suite of discovery algorithms derived from the basic algorithm. We first discuss four discovery algorithms that operate within a single administrative domain, then discuss our algorithm for discovering the Internet backbone.

5.2 Algorithm 1: SNMP

This algorithm is the simplest because it assumes that SNMP is available everywhere in the domain. The first router added to the temporary set is the discovery node's gateway router (step 1). For each router in the temporary set, we find neighboring routers from that router's ipRouteTable MIB entry (step 2e). Hosts are obtained from the router's ARP table entries (step 2c).

```
1.  temporarySet = get_default_router()
2.  foreach router ∈ temporarySet do
        a.  ping(this_router)
        b.  if (this_router is alive) then permanentSet = permanentSet ∪ this_router
        c.  hostList = SNMP_GetArpTable(this_router)
        d.  permanentSet = permanentSet ∪ hostList
        e.  routerList = SNMP_GetIpRouteTable(router)
        f.  permanentSet = permanentSet ∪ routerList
        g.  temporarySet = temporarySet ∪ routerList
```

This algorithm is efficient, fast, complete, and accurate, and has the added benefit that additional SNMP information can be gathered for each node. However, it can only be used on networks where SNMP is enabled on all routers. Thus, it fails to meet one of our primary goals.

5.3 Algorithm 2: DNS zone transfer with broadcast ping

This algorithm assumes that the domain allows DNS zone transfer and pings to broadcast addresses. It first does a DNS zone transfer to get a list of all hosts in the domain (step 1). Assuming that this retrieves a list of all hosts in the domain, the only remaining problems are (a) to eliminate invalid names in the listing and (b) to determine the identity of each subnet in the network. We validate hosts by pinging them (step 2b). Subnets are discovered using the subnet-guessing algorithm (step 2c). Finally, in order to discover nodes not found in the DNS zone transfer, the algorithm directs a broadcast ping to newly discovered subnets (step 2e), and adds responding nodes to the temporary set (step 2f).

```
1.  temporarySet = DNS_domain_transfer(domain)
2.  foreach node ∈ temporarySet do
        a.  ping(this_node)
        b.  if (this_node is alive) then permanentSet = permanentSet ∪ this_node
        c.  this_subnet = SubnetGuessingAlgorithm(this_node)
        d.  permanentSet = permanentSet ∪ this_subnet
        e.  if unknown this_subnet then ping_broadcast(this_subnet)
        f.  foreach responding_host do
                add responding_host under this_subnet
                temporarySet = temporarySet ∪ responding_host
```

This algorithm is neither efficient nor fast, because subnet guessing is both slow and involves considerable overhead. Moreover, it heavily depends on DNS zone transfer and broadcast ping, which may both be unavailable for reasons of security. In this case, the algorithm would also be incomplete. Although running several versions in parallel can speed up the algorithm, we do not recommend its use. Its main purpose is to show that network topology can be correctly determined even in the absence of SNMP. Our subsequent algorithms eliminate many of the deficiencies in this algorithm.

5.4 Algorithm 3: DNS zone transfer with traceroute

This algorithm replaces the expensive subnet-guessing technique of the second algorithm with the more efficient computation of Heuristic 2. It still assumes that the domain a DNS zone transfer is allowed, and returns a list of all hosts and routers in the network. It also assumes that the DNS server returns all the IP addresses associated with a router, instead of picking one at random.

The basic idea here is to get a list of all routers and hosts in the domain with DNS zone transfer (step 1). We then initiate a ping and traceroute to each member of this list (steps 3a and 3c). Recall that Heuristic 2 requires us to provide the addresses of a cluster of hosts in a subnet and the 'closest' router interface. We therefore need to store the set of hosts associated with a particular router. A host's router is just the next-to-last value in a traceroute to that host, so the router is easily determined. However, traceroute may return the IP address of an interface that is not necessarily the interface used to route to that host (for example, in Figure 1, traceroute may return address B instead of address A). To find the right interface, we do an inverse name lookup on the next-to-last address and, for each returned address, XOR it with the hosts address. The best match between the router interface and the host has the fewest ‘1’ bits and is chosen as the correct interface. Instead of storing a list of host addresses in a hash table keyed on a router’s interface address, we maintain two hash tables called cumulativeAnds and cumulativeOrs keyed on this interface. The first contains the bitwise AND of all the hosts for which this interface is the next-to-last hop and the second contains the bitwise OR of all the hosts for which this interface is the next-to-last hop. As more and more hosts get processed, the cumulativeAnds value for a particular interface value gets closer and closer to the actual subnet. Given the subnet number and the cumulativeOrs value, we can quickly determine the subnet mask (step 3k). As explained earlier, this determination may be incorrect--we choose the smallest possible subnet mask that is consistent with the data. Here is the algorithm:

```
1. temporarySet = DNS_domain_transfer(domain)
2. initialize cumulativeAnds{} and cumulativeOrs{} hashtables to null
3. foreach node ∈ temporarySet do
       a. ping(this_node)
       b. if (this_node is alive) then permanentSet = permanentSet ∪ this_node else continue
       c. traceroute(node)
       d. this_router = next to last hop in traceroute
       e. find all IP addresses of this_router with a DNS lookup
       f. this_gateway = address such that number of ‘1’ bits in IP(this_router) XOR this_node is minimized
       g. oldSubnet = cumulativeAnds{this_gateway} //this notation means look up the hash table with this key
       h. cumulativeAnds{this_gateway} = this_node AND cumulativeAnds{this_gateway}
       i. cumulativeOrs{this_gateway} = this_node OR cumulativeOrs{this_gateway}
       j. newSubnet = cumulativeAnds{this_gateway}
       k. newSubnetMask = NOT (cumulativeAnds{this_gateway} XOR cumulativeOrs{this_gateway})
       l. Store node in newSubnet
       m. If necessarv. move hosts in permanent set's oldSubnet to newSubnet.
```

This algorithm is faster than the previous one and has fewer overheads because it replaces the expensive subnet guessing heuristic with a faster one. Moreover, it makes fewer assumptions than the previous algorithm, and thus is more widely applicable. However, because it depends exclusively on DNS zone transfer to retrieve the temporary set, it may not be as complete as the previous algorithm. There is also a problem with correctness: the subnet mask may be incorrectly determined when there all machines in a subnet use IP addresses at the high end of the subnet’s address space. If this were the case, then the discovered mask would be longer than it should be (because the common initial bit string is longer). A second correctness problem with the algorithm is that it assumes that an inverse DNS lookup returns all the IP addresses of a router. In some networks, all the routers’ IP addresses are not entered in the DNS. This results in incorrectly aggregating subnets that are served by routers one hop away from a specific router.

4.5 Algorithm 4: Probing with traceroute

Algorithm 3 uses DNS zone transfer to determine all the hosts in a network. This is disabled in many domains for reasons of security. Algorithm 4 replaces this by intelligent guesses about the IP addresses in the domain, as explained in Heuristic 3 (Section 3.3). Thus, it makes very few assumptions about the network. The algorithm assumes that traceroute is supported, and that a DNS lookup returns all the IP addresses associated with a router. The intuitive idea is to generate IP addresses at random in the address space belonging to the domain. Traceroute probes to these addresses expose the routers in the network and Heuristic 2 allows us to guess the associated subnet numbers. The differences between this algorithm and Algorithm 3 are shown in italics.

```
1. temporarySet = random addresses in domain that end in ".1"
2. initialize cumulativeAnds{} and cumulativeOrs{} hashtables to null
3. foreach node ∈ temporarySet do
      a. ping(node)
      b. if (this_node is alive) then permanentSet = permanentSet ∪ this_node else continue
      c. Use Heuristic 3 to add more addresses to temporary set
      d. traceroute(node)
      e. this_router = next to last hop in traceroute
      f. find all IP addresses of this_router with a DNS lookup
      g. this_gateway = address such that number of '1' bits in IP(this_router) XOR this_node is minimized
      h. oldSubnet = cumulativeAnds{this_gateway} //this notation means look up the hash table with this key
      i. cumulativeAnds{this_gateway} = this_node AND cumulativeAnds{this_gateway}
      j. cumulativeOrs{this_gateway} = this_node OR cumulativeOrs{this_gateway}
      k. newSubnet = cumulativeAnds{this_gateway}
      l. newSubnetMask = NOT (cumulativeAnds{this_gateway} XOR cumulativeOrs{this_gateway})
      m. Store node in newSubnet
      n. If necessary, move hosts in permanent set's oldSubnet to newSubnet.
```

This algorithm only uses traceroute and ping to determine topology. Thus, it can be used in practically any network. However, in order to find all hosts in the network, we need to choose a large N value in Heuristic 3, slowing down the algorithm. Specifically, to use ping to determine that node does not exist is expensive, because this requires waiting for a fairly long timeout. Thus, if Heuristic 3 adds many nonexistent nodes to the temporary set, the algorithm takes a long time to complete. We recommend that it should be used when discovering routers and subnets is more important than discovering all the hosts. A second problem is that we assume that inverse DNS lookups return all the IP addresses associated with a router, which may not be universally true.

4.6 Backbone topology discovery

Discovering the topology for the Internet backbone is quite different than discovering the topology within a domain. Within a domain, we can exploit addressing conventions and broadcast pings to quickly discover large clusters of hosts. In the backbone, the best, and perhaps the only way to find the topology is to find the all the routers and links using traceroute. Here is a naive algorithm that does this:

```
1. Generate a number of IP addresses well distributed in the IP space.
2. foreach address
      a. traceroute to the address
      b. store all the links and nodes the traceroute returns
3. Correlate the results
```

The problem with this algorithm is that, depending on the number of addresses probed, it is either very expensive, or discovers only a small fraction of the backbone. Even though the algorithm is easily parallelized, the number of addresses for effective discovery of backbone topology is still too large. Our initial attempt to use this algorithm to probe Internet backbone topology took about a month to complete, even when split between twelve different processes!

The solution to this problem is to reduce the number of IP addresses probed. This can be done by taking advantage of the aggregation of IP addresses used to help simplify routing in the backbone. Indeed, the publicly available BGP routing tables contain the IP addresses of a large percentage of the domains connected to the Internet backbone. By sending a probe to a randomly chosen IP address in each advertised domain, we are guaranteed to hit a large number of backbone routers and links. This leads to the following revised algorithm:

```
1.  Get BGP routing information, for instance from route-server.cerf.net
2.  Extract all domains from this information
3.  Choose an address in each domain, such as the a.0.1 or a.0.0.1, where a is the network number
4.  foreach address
        a.  traceroute to the address
        b.  store all the links and nodes the traceroute returns
5.  Correlate the results
```

This reduces the number of probes to the number of domains, which is less than 50,000 as of April 1998. The algorithm, when divided into twenty-five processes probing 2,000 addresses each, takes approximately 48 hours to complete (for more details, see Section 6).

While the revised algorithm is relatively efficient, it still suffers from a serious problem: it only discovers a tree of nodes leading away from the source of the probes and thus 'cross-links' will not be discovered. For example, in Figure 2, the intermediary node and links between Domain A and Domain B will not be found:

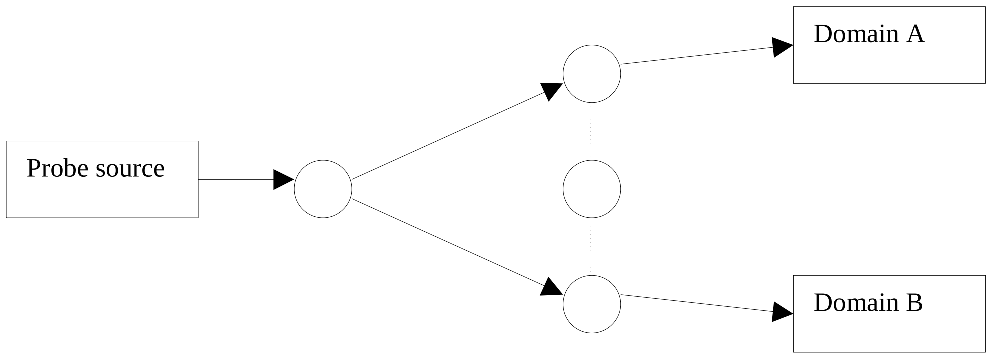

Figure 2

This problem is partially solved by choosing multiple probe sources. For example, suppose we had a probe point at Domain B, we would have found the link to Domain A. Although the algorithm can never guarantee to find all the nodes and links unless there is a probe source in every domain, if probe sources are evenly distributed throughout the network, we should be able to get a good approximation of Internet backbone topology.

6. Evaluation

6.1 Domain topology discovery

In this section we evaluate the performance of the four algorithms described in Sections 5.2-5.5, in the cs.cornell.edu (CUCS) and cornell.edu domains. The cs.cornell.edu domain has about 480 hosts and six routers, one of which is outside a firewall. The cornell.edu domain has about 8200 hosts and 140 routers in more than 500 subnets. We believe that these domains are representative of department and campus domains elsewhere in the Internet. The algorithms were implemented on a SPARCsystem-600 machine running SunOS 5.5.

We now compare the four algorithms in terms of their speed, overhead, completeness and accuracy. We measure *speed* by the time in minutes taken to discover the domain, as measured by the Unix `time` command. We measure *overhead* by the number of hops taken by ping packets and traceroutes in the domain. Recall that we ping each address twice. Thus, the overhead, in terms of packets forwarded in the network, to ping a node H hops away is 4H, because two request and two reply packets have to each traverse H hops. With traceroute, we choose to send an

ICMP packet to every router twice. The total cost to traceroute to a host H hops away is four times the arithmetic sum from 1 to H, i.e. 4H(H+1)/2, because we incur a cost of 4h, 1<= h <= H, for each router h hops from the source. Thus, the normalized cost of a traceroute to a host H hops away, relative to the cost of a ping, is (H+1)/2. If the mean distance between the probing source and the rest of the nodes in the domain is H*, the mean cost of a traceroute, compared to that of a ping is (H*+1)/2. The H* values for the CUCS domain and the cornell.edu domain are 2.8 and 7.9. We measure *completeness* by the fraction of the topology discovered by each algorithm. For simplicity, we assume that the domain size is the largest of those discovered using any of our algorithms. The only inaccuracy introduced in our algorithms is due to Heuristic 2, which may choose a subnet mask incorrectly if all hosts in a subnet are in the higher end of the subnet's address range. Thus, we measure inaccuracy as the ratio of the number of times the heuristic faced an ambiguity to the number of discovered subnets, and *accuracy* as its complement. Note that it is impossible to fairly compare the algorithms in the sense that some networks may not support DNS zone transfer or ping broadcast, ruling out the second and third algorithms. However, we do feel that relying entirely on SNMP to discover network topology is a bad idea: though other algorithms take more time and more overhead, for a large topology, they are significantly more complete.

CUCS Network	**Speed**	**Overhead**			**Completeness**[1]			**Accuracy**
	Time (minutes)	# pings	# traces	Normalized overhead	Hosts	Routers	Subnets	
SNMP	11	5	0	5	482 (99%)	5 (100%)	7 (100%)	100%
DNS zone transfer/ broadcast ping	148	1195	0	1195	485 (100%)	5 (100%)	6 (86%)	100%
DNS zone transfer/ traceroute	128	840	480	1752	480 (99%)	5 (100%)	7 (100%)	99%
Probing/traceroute	58	611	336	1249	317 (65%)	5 (100%)	7 (100%)	99%

Table 1: Comparison of intra-domain discovery algorithms in the cs.cornell.edu domain.

Table 1 compares the four algorithms in the CUCS domain. The SNMP algorithm runs the fastest by far, and, in this domain, discovers the entire network accurately. This is clearly the algorithm of choice. The other algorithms have nearly the same overhead, and all discover nearly the entire topology, except for the probing/traceroute algorithm, which takes significantly less time than the other two, but discovers only 65% of the hosts. Note that the time taken by any of the algorithms can be reduced by parallelizing it. Thus, for this domain, we recommend using SNMP, failing which, the second best choice is DNS zone transfer/ping broadcast, since it has less overhead than DNS zone transfer/traceroute, but still discovers nearly the entire topology.

Cornell Network	**Speed**	**Overhead**			**Completeness**[1]			**Accuracy**
	Time (minutes)	# pings	# traces	Normalized overhead	Hosts (%)	Routers (%)	Subnets (%)	
SNMP algorithm	193	139	0	139	602 (8%)	139 (90%)	93 (15%)	100%
DNS zone transfer/ broadcast ping *	-	-	-	-	-	-	-	-
DNS zone transfer/ traceroute	2880	10204	7367	42987	7367 (100%)	155 (100%)	622 (100%)	90%
Probing/traceroute	1080	8532	2735	20702	2734 (37%)	144 (93%)	512 (82%)	90%

*The cornell.edu domain does not allow pings to broadcast addresses

Table 2: Comparison of intra-domain discovery algorithms in the cornell.edu domain

Table 2 presents the results for the cornell.edu domain. Here, the performance of the four algorithms is rather different. In contrast to the excellent performance of SNMP in the CUCS domain, SNMP in this domain discovered

[1] Percentages are with respect to the largest number of hosts, routers, or subnets discovered by any of the four algorithms.

only 8% of the hosts and 15% of the subnets. We conjecture that this is because many hosts in the domain are idle and therefore do not show up in router ARP tables, and many of the routers do not accurately report subnet information. The cornell.edu domain does not allow pings to broadcast addresses, so this algorithm cannot be used. The DNS zone transfer/traceroute algorithm is complete, but has a high overhead and is rather slow. In contrast, the probing/traceroute algorithm takes about a third as much time, but discovers only a third as many hosts. Thus, for discovering the complete topology, we recommend the use of the DNS zone transfer/traceroute algorithm. However, if only router information is desired, the probing/traceroute algorithm may prove sufficient.

6.2 Backbone topology discovery

In this section, we report on the performance of the algorithm described in Section 4.6 that discovers the Internet backbone. We ran the algorithm for 48 hours from three probe points at Cornell, UC Berkeley, and Stanford. We discovered 76,127 nodes, where each node corresponds either to a backbone router or to an interface on a backbone router, and 98,538 links. While the volume of data is too large to display here, Figure 3 shows a fraction of the topology of the NYSERNET backbone, as discovered by our algorithm. (An interactive viewer for this topology can be found at http://www.cs.cornell.edu/cnrg/topology/nysernet/index.html.) Note that collating data from the three probe points allows us to discover more than just a tree rooted at a probe point, as is evident from the figure.

As is evident, displaying backbone topology is difficult because of the large number of nodes. Therefore, we have come up with a visualization technique called a *hop contour map* that summarizes topology information. Intuitively, a hop contour map shows the number of routers on a contour line corresponding to a certain number of hops from a source. The Y-axis of a hop contour map is a hop count, and the X-axis is the number of routers found at that hop count. For example, Figure 4, a hop contour map of the NYSERNET backbone, shows that we discovered three routers six hops away from Cornell. Figure 5 is a concatenation of hop contour maps for a number of ISP backbones. Here, the Y-axis is the number of hops from the Cornell probe point and the X-axis is a concatenation of contour maps for different ISPs. The distance between two columns is 500 routers. The plot shows, for example, that ISPs ‘alter’ and ‘MCI’ have 200 routers at some hops and an individual ISP can have more than 1000 routers. From this plot, we estimate that the number of backbone routers in the Internet is in the 20,000-router range.

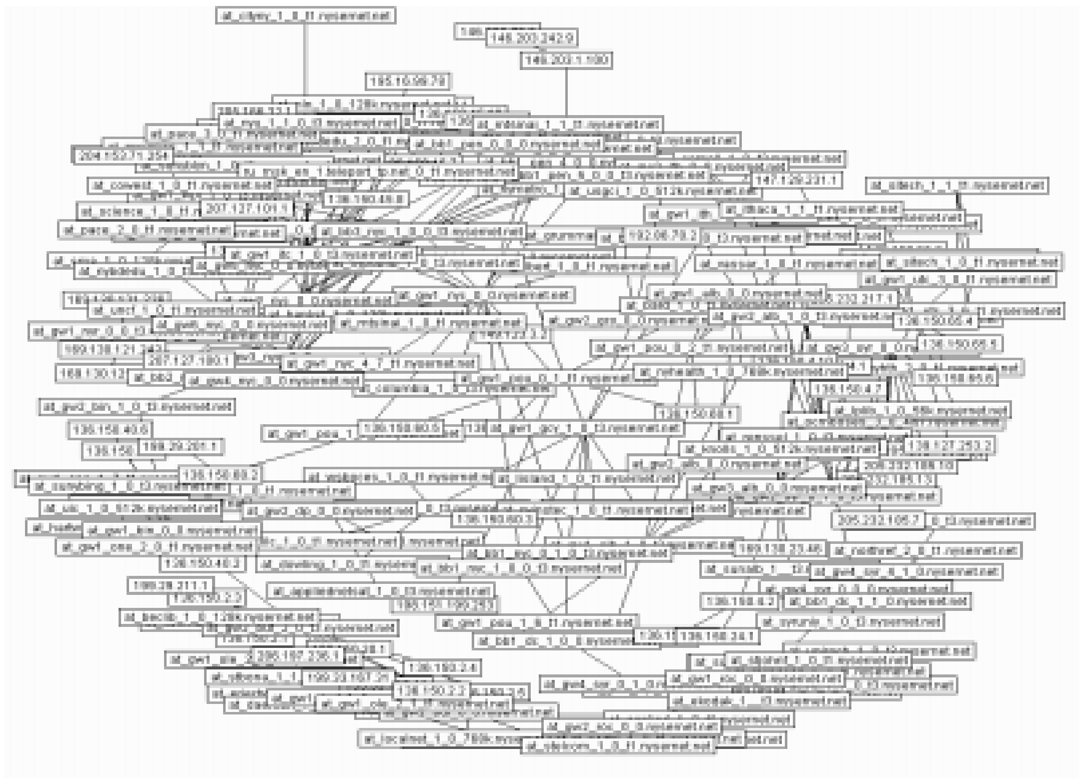

Figure 3: A portion of the NYSERNET topology discovered by our algorithm.

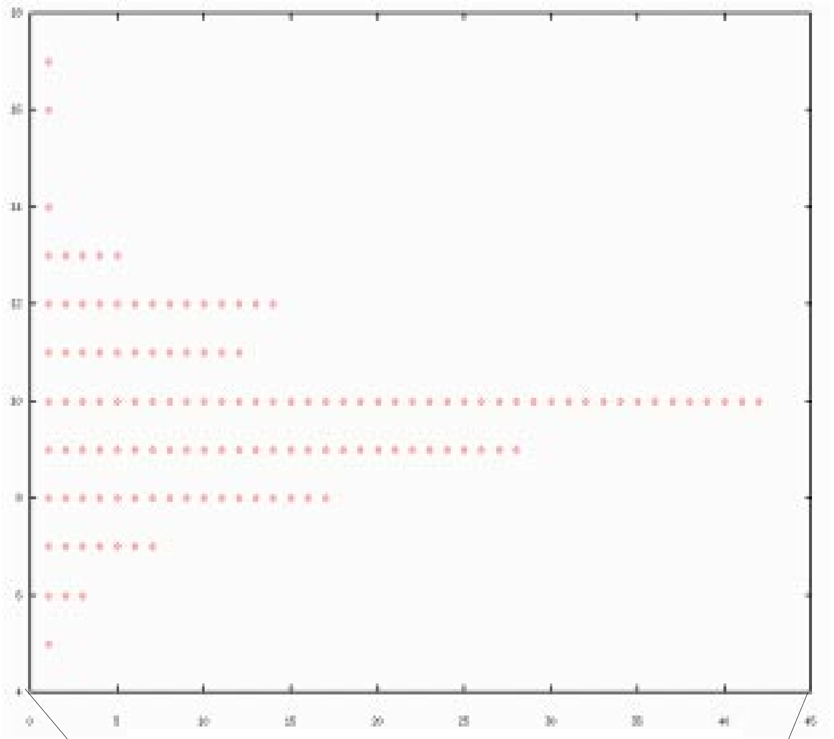

Figure 4: Hop contour graph for NYSERNET

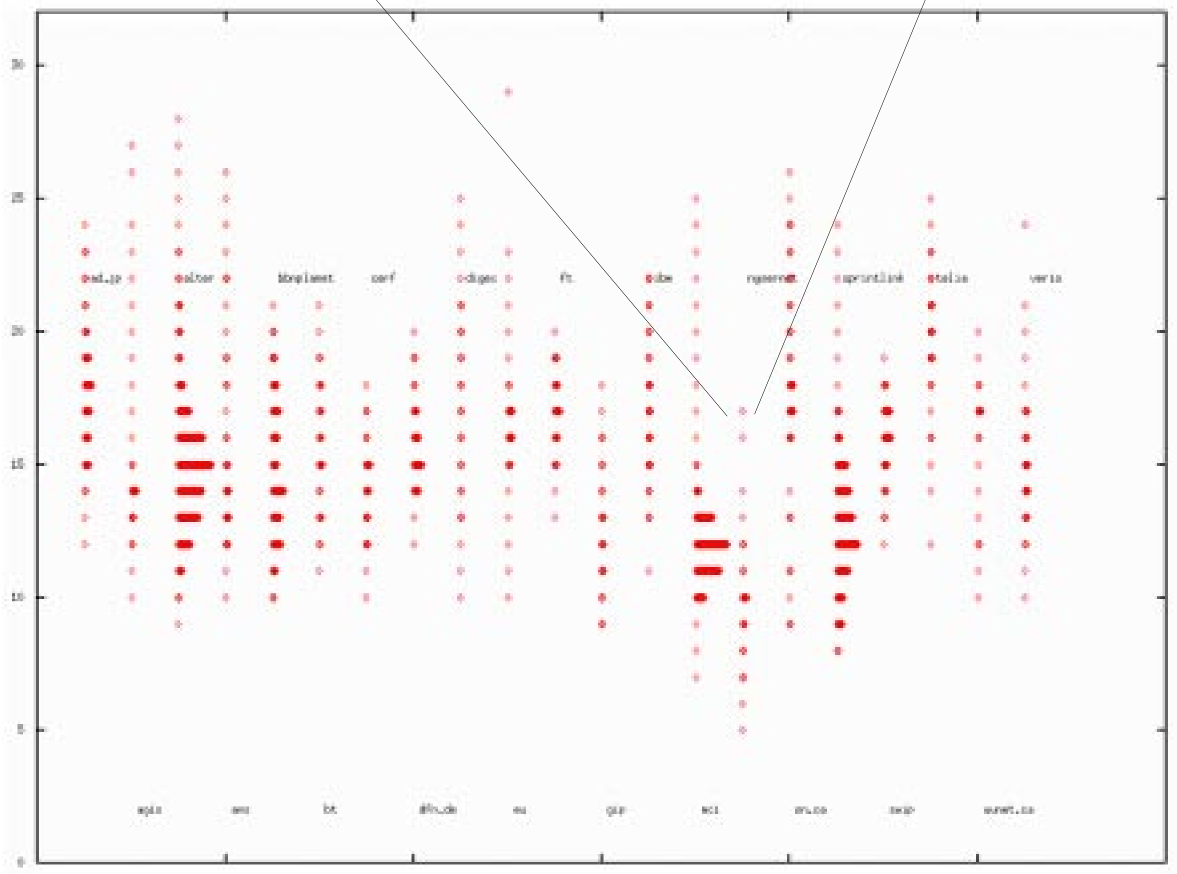

Figure 5: Hop contour maps for all large ISPs (labels are left aligned with columns)

Hop contour maps provide an interesting way to evaluate the 'goodness' of a particular ISP backbone. To see this, consider the hop contour map of a linear backbone (Figure 6a) It is clear that this corresponds to a 'tall and skinny' hop contour map, since each new hop only adds one more router to the set of discovered routers. In contrast, with a

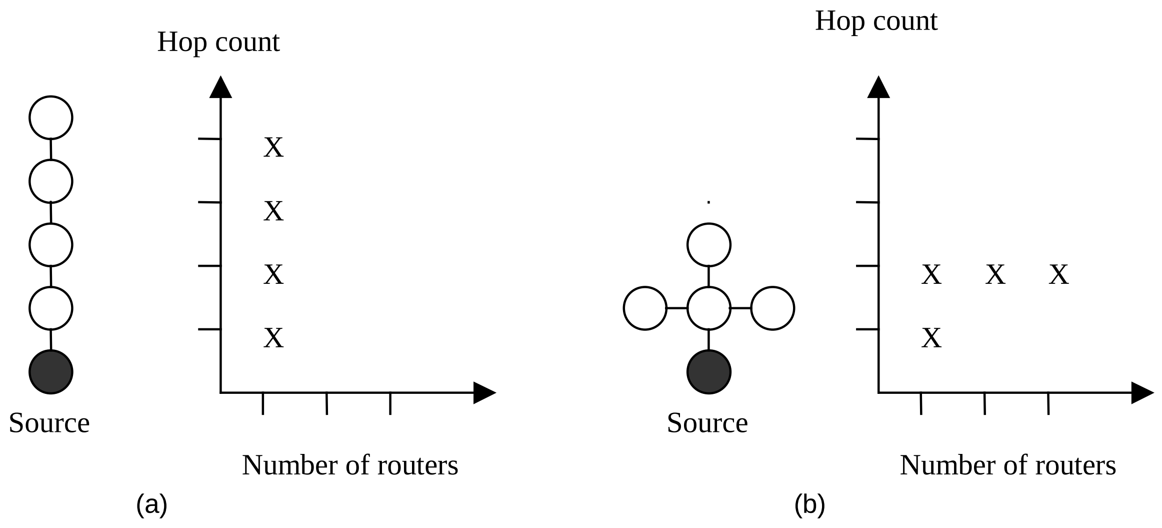

Figure 6: Hop contour map as a function of topology

densely connected topology such as a star or a clique, the hop contour map is 'short and fat': once we enter a backbone, all other routers are one or two hops away (Figure 6b). Since we would like to traverse backbones with as few hops as possible, we claim that a good measure of backbone design is the degree to which it is 'short and fat' as compared to 'tall and skinny'. Of course, the presence of ATM backbones distorts this measure, since ATM hops cannot be detected by traceroute. In future work, we intend to come up with appropriate metrics on hop contour maps that correlate with desirable backbone features such as high-connectivity and a diversity of alternate paths.

7. Related work

Network topology information is often disclosed by ISPs. Two notable collections of known topologies can be found at References [Atlas 98, CAIDA 98]. These topologies, however, are neither automatically discovered, nor updated.

Automatic network discovery is a feature of many common network management tools, such as HP's OpenView, and IBM's Tivoli. These tools, however, assume that SNMP is universally deployed. As we saw in Section 6.1, such a tool would fail to discover 10% of the routers and 92% of the hosts in the cornell.edu domain.

Other commercially available tools for network discovery use a variety of information sources to augment SNMP. These include:

- **netViz:** netViz exploits Microsoft Windows internal network calls to discover servers running Windows 95, 98 or NT, volumes and printers in a Microsoft network. The main focus of this product appears to be a user-friendly GUI to browse the topology [NetViz 98].

- **Optimal Surveyor:** Optimal Surveyor uses SNMP and Novell network queries to discover network topology [Actualit 98].

- **Dartmouth Intermapper:** Intermapper discovers topology using SNMP queries and Appletalk broadcast calls. Intermapper also discovers and displays link and server load [Intermapper 98].

To sum up, these tools exploit network-specific protocols, such as Windows Networking, Novell Networking, and Appletalk to augment the information provided by SNMP. From this perspective, our work can be viewed as augmenting SNMP in a *protocol-independent* fashion. Thus, it can be used to improve the functionality of any of these tools.

The idea of discovering network characteristics to help improve protocol performance is at the heart of several recent studies. For example, in the SPAND approach [SSK 97], network characteristics passively discovered by a cluster of hosts are shared through a performance server. This helps in decisions such as choosing the closest web site that contains a copy of the desired information, and choosing a data format that is consistent with the available capacity. Our work differs from this in that we perform active probing, and we do not discover network performance characteristics, only network topology. A detailed critique of several other algorithms that perform network probing for determining expected performance and server selection can be found in Reference [SSK 97].

The IDMaps project [FJPZ 98] seeks to provide client applications with approximate hop counts and expected performance between pairs of IP subnet numbers to allow them to suitably adapt their behavior. The focus of this work is primarily on creating a *virtual* topology that accurate represents performance characteristics between subnets. In contrast, we aim to discover the *logical* topology of the network at the IP level.

8. Future work

The algorithms described in this paper can be extended in a number of ways, including:

Exploiting history: We think that it would be interesting and insightful to periodically run intra-domain and backbone discovery algorithms and analyze how topologies change over time. This would allow us to track growth patterns, particularly in the backbone, where ISPs rarely reveal topology information.

Evaluating ISP goodness: Hop contour maps display backbone topologies in a way rather different from a normal graph layout. In future work, we would like to correlate metrics on hop contour maps, such as the ratio of the maximal width to the height, with the 'goodness' of an ISP's topology. This would allow customers to quickly and objectively evaluate the relative merits of competing ISPs.

Correlation: Currently, the backbone discovery algorithm discovers a tree of links and nodes rooted at a single source point. While it is possible to generate multiple rooted trees by choosing multiple probe points, correlating this information to form a single topology is a complex task. Our current approach to collating this information is semi-automatic and requires both skill and many hours of painstaking work. In future work, we would like to automate this process.

Analysis: While we have evaluated the performance of our algorithms in a couple of domains, we have yet to carry out a formal asymptotic analysis of their complexity. We would like to do so in the future, because we think that this will allow us to gain insight into their behavior, and perhaps suggest variations that have better performance.

Link characteristics: Currently, we do not discover link characteristics such as capacity and mean delay. Existing tools such as `pathchar` [Jacobson 97] and PBM [Paxson 97] can discover these automatically. In future work, we plan to integrate these tools with ours.

Exploit other sources of information: At the moment, our tools do not exploit routing information within a domain. If a topology discovery tool were allowed to extract information from a routing daemon running RIP or OSPF, it would be possible to quickly get subnet information within a domain. Similarly, we could extract network-specific information from a Microsoft WINS server, a Novell Directory Server, or an Appletalk network. In future work, we plan to integrate these sources of information with our tools.

9. Conclusions

Topology information is critical for simulation and network management. It can also be used effectively for siting decisions, and as an element in a new class of topology-aware distributed systems. We have presented several algorithms that discover intra-domain and backbone topology without relying on SNMP. We find that our algorithms, though slower than those using SNMP, are often able to discover far more nodes and subnets. This reflects the fact that SNMP is not universally deployed, particularly at end-systems, and indeed is the motivation for our work. We also evaluated a backbone discovery tool that was able to discover more than 70,000 nodes in the Internet backbone. This data, when visualized using hop contour maps, allows us to compare ISP backbone topology.

10. References

[Actualit 98] Actualit Corp., Optimal Surveyor home page `www.actualit.com/products/optimal/optimal_surveyor/optimal_surveyor.html`

[Atlas 98] An Atlas of Cyberspace, `http://www.geog.ucl.ac.uk/casa/martin/atlas/atlas.html`

[CAIDA 98] CAIDA Mapnet viewer, `http://www.caida.org/Tools/Mapnet/`

[FJPZ 98] P. Francis, S. Jamin, V. Paxson, L. Zhang,, Internet Distance Maps, Presentation available from `http://idmaps.eecs.umich.edu/`

[IETF 97] IETF Physical Topology MIB Working Group, PTOPO Discovery Protocol and MIB, Internet draft available from `http://www.ietf.org/internet-drafts/draft-ietf-ptopomib-pdp-02.txt`

[Intermapper 98] Intermapper home page, `http://intermapper.dartmouth.edu`

[Jacobson 97] V. Jacobson, “Pathchar” binary, `ftp://ftp.ee.lbl.gov/pathchar/`

[MJ 98] G.R. Malan and F. Jahanian. An extensible probe architecture for network protocol performance measurement, *Proceedings of SIGCOMM'98*, Sept. 1998, Online version available from `http://www.merit.edu/ipma/docs/paper.html`

[NetViz 98] NetViz home page, `http://www.netviz.com`

[Paxson 97] V. Paxson, Measurements and Analysis of End-to-End Internet Dynamics, *Ph.D. Thesis*, UC Berkeley, May 1997.

[SSK 97] S. Seshan, M. Stemm, and R.H. Katz, SPAND: Shared Passive Network Performance Discovery, *Proc 1st Usenix Symposium on Internet Technologies and Systems (USITS '97)*, Monterey, CA, December 1997.

Appendix: Evaluation of Heuristic 3--Guessing valid addresses in a domain

Heuristic 3 described some rules for guessing valid IP addresses in a domain. These are based on the following observations in the CUCS and cornell.edu domains.

	CUCS[2]	**cornell.edu**[2]
Total number of subnets	8	537
Subnets with a valid ".1" addresses	8 (100%)	421 (78%)
Subnets with a valid ".129" addresses	6 (75%)	120 (22%)
Subnets with a valid ".65" addresses	2 (25%)	104 (19%)
Subnets with a valid ".193" addresses	4 (50%)	48 (9%)

Table 3: Valid addresses in the CUCS and cornell.edu domain.

In Table 3, the first row is the number of subnets in the CUCS and cornell.edu domains, assuming that both domains use only the 255.255.255.0 netmask (other netmasks are accounted for by looking for *.65, *.129 and *.193 addresses, as described in Section 4.3). We see that in the CS domain, 100% of the subnets had a valid *.1 address, and 75% had a *.129 address. In the larger cornell.edu domain, nearly 80% of the subnets had a corresponding *.1 address, and 22% had a corresponding *.129 address. Thus, we believe that the heuristic of guessing that a subnet has a *.1 address is very effective, guessing the other 'typical' router addresses less so. (Incidentally, 95% of the subnets that did not have a *.1 address are in the Cornell medical school.)

	CUCS[3]	**cornell.edu**[3]
Total number of valid addresses	502	8189
Number of valid IP addresses adjacent to another valid address	384 (76.5%)	5924 (72.3%)
Number of valid IP addresses up to two away from another valid address	460 (91.6%)	6763 (82.6%)
Number of valid IP addresses up to three away from another valid address	472 (94.0%)	6927 (84.6%)

Table 4: Valid adjacent addresses in a domain.

Table 4 shows measurements of the number of adjacent IP addresses in the CUCS and cornell.edu domains. The first row shows the total number of nodes in each domain. The next rows show the number of valid addresses that are up to one, two, three, four, and five addresses away from another valid address. Consider the second row. It tells us that in the cornell.edu domain, given a valid IP address, the probability that one of its adjacent IP addresses is valid is roughly 72.3%. If we add four addresses to the temporary set, two on either side of a valid address, the probability that we have added at least one valid address to the temporary set increases to 82.6%. The CUCS domain, which is more heavily populated, is even more easily discovered with Heuristic 3.

[2] Percentages are with respect to the total number of subnets in that domain

[3] Percentages are with respect to the number of valid addresses in the domain.