Kaushik Srinivasan, Krishna Venkatasubramanian

Project ID: P-122 (***Geography of the web***)

CSE 450/598

Design and Analysis of Algorithm

Final Project Report

Table of Contents

Abstract

In this paper we look at IP address to location mapping techniques. Over the years the Internet has grown rather chaotically. The present infrastructure does not provide any means to find the location of the node given its IP address. IN this paper we will be analyzing three algorithms called Geo-Track, Geo-Ping and Geo-Cluster. These three algorithms use the present infrastructure and try to map an IP address to a location. We present a detailed description and analysis of these algorithms.

1. Introduction

Determining the geography of the web is the problem of mapping a given IP (for IPv4 only) address to a specific geographic location. The internet of today does not have any formal structure. Since its birth the growth of internet has always been rapid and chaotic to say the least. The original infrastructure developers did not feel the necessity to map any IP address to its location. The reason could have been that nobody initially envisioned the rapid growth of Internet and thus did not think of need to map IP address to geographic locations. In today's times this thinking may not be necessarily correct. Nowadays a large number of services are provided through the internet. E-commerce has become a reality and customers can purchase commodities while surfing the web. An example could be the Amazon online stores. Like any other retailer, e-commerce companies would like to know about there customers and thus provide customized features to them. One of the information that can be collected is the location of the customer. Given the location of the customer the online retailer can find the commodities that the customer would most likely buy and present them to make the shopping experience much easier and beneficial to both parties. Another example of knowing location information could be to provide information pertaining to that location: like the weather, local news, traffic conditions etc. to the customers. Mapping the IP-location can also be used to connect a client to the appropriate mirror site. A company with a large online presence usually manages a large number of servers, strategically placed to balance the load. For example Amazon has many mirror sites spread through out the continental US and if the location of a connecting customer can be found, the nearest mirror site to the customer's actual location can be used to provide contents. Using the

appropriate mirror site can do excellent load balancing (thus not overloading a server) and also provide fast connection to the clients. Nowadays technologies are present that map an IP address to the appropriate country or geographical region like North America, East Asia etc. But in order to provide a better service, we have to find locations at a lower granularity. In summary the need for IP-location mapping is as follows [1] [2] [3]:

- Web services could send a host location targeted information (local weather, events etc.)
- It would also help the content provider to customize content sent to a particular location in the world.
- If a content provider has many mirror sites, then by accurately finding the location of a host, the nearest mirror site can be used provide content.

In this paper we are going to analyze three IP-location mapping algorithms. They were developed at UC Berkeley and at Microsoft Research. The three algorithms are called GeoTrack, GeoPing and GeoCluster. These three are collectively called *IP2Geo* protocols. Each one of the protocol uses a different mechanism to solve the problem of location mapping. GeoTrack uses names assigned to DNS servers to find to find out the location of the DNS server and hence the geographic location. It uses the trace-root tool to find he names of DNS servers. The second algorithm is based on the assumption that the distance between two nodes is proportional to the round trip delay between the two nodes. The authors have done several experiments to find that this assumption holds. The third algorithm uses the BGP header information to group IP addresses into geographic clusters. They are based on partial location to IP mapping. The correctness of the results is depends on the correctness of the partial mapping.

One of the major problems in finding these IP-location mapping techniques is the serious dearth of material in the literature. Many people are working in allied areas like topology discovery and IP routing but actual open research in this topic seems very limited. Many content providing companies have similar algorithms in use but they are not at all open to public.

The paper is organized as follows: in section 2, we present the related work. In section 3, we present the GeoTrack, GeoPing and GeoCluster algorithms in detail and its analysis results, respectively. In section 4 we compare the three approaches. In section 5 we present our conclusions and future work that can be done in this area.

2. Related Work

Generally many location finding techniques exist in the real world. Some examples could be: the GPS system for wireless networks, another one called RADAR [radar] developed at Microsoft Research. The former one finds the location of an entity, but is not useful indoors, while the latter is useful in finding location of a person in building and may not be much use outside. But for the internet no such mechanism can be used. The only way to provide customized solution till recently (which is used extensively today) is ask the user about his/ her location and/or by using client based cookie. However such location finding solutions have the following disadvantages [2] [3]:

- They are burdensome on the user.
- Ineffective if the user a client which is different from the one where the cookie was stored.
- Prone to error, because very few users, when asked enter the right information.

Because of these reasons we need to build a system that automatically maps IP address to a geographic location that produces correct results. Therefore the results do not track the user's location but the location of the client machine. In this section we will present some work that has been done in this area prior to the work that we are presenting. Some of these works suggest the changes be made to the present infrastructure; others work with the existing system. These techniques are:

- Incorporating location information in DNS server. DNS (Domain Name Server) is the entity that given a URL finds the appropriate IP address [5]. Once deployed this system is very efficient as we can get the location information at the time of IP address location resolution. But the problem here is that, this is technique that would require substantial change to the present internet infrastructure. Therefore it faces deployment hurdles, even after this idea was proposed in RFC 1876. The update

would take place by placing a new RR (Resource Record) for the DNS. This will result in a new DNS entry called LOC (location) with a code for location. But the modification of this record structure is non trivial and also the network administrators have to enter the location data into the DNS servers. Most importantly, we cannot be sure if the location data entered is correct, as we cannot verify all the data entered.

- The second technique that was proposed was the use of *whois* [3] [4] database. This database gives the location of an organization to which as IP address was assigned. Several tools use the *whois* database to infer location information about geographic host. The presence of a database is fine but it has certain problems. First the *whois* database may have stale or incorrect data. Secondly, multiple servers may have inconsistencies regarding an IP address block. Thirdly, the address resolution being block oriented instead of individual address causes problem. Many organizations have blocks of IP addresses allocated to them. Therefore, in the database a whole IP address block belonging to an organization may show the organization headquarter as the address while there maybe many branches of the organization, located at far of places. For example 4.0.0.0/8 IP address block is allocated to BBN Planet and the query to a whois database gives location as Cambridge, MA for any IP address in this range. This may not be correct as BBN Planet may have branches located in different US cities that have hosts with IP addresses in the above mentioned block [3].
- The third way of traditional IP-location mappings done using *traceroute* tool [2] [3]. A *traceroute* tool sends out an ICMP packet from a source to the destination, as the packet goes towards the destination it sends back names of all the intermediate nodes back. By parsing these names we can try to find the location of these intermediate routers and finally the destination. Here the idea is to use the *traceroute* from the source to a target IP address. It is still difficult to find the location with the DNS names because, there is no naming convention for the names therefore the location names in DNS names may not be present. The GeoTrack algorithm uses the *traceroute* to function; the details will be explained later.
- Finally, the brute force technique can be used, like exhaustive mapping of IP address range to their corresponding locations. Many content providers like Akamai [5] depend on this technique for providing customized content. Example is Edge-Space.

According to [3] because of the good relationship that these companies have with web sites the data may not have the same problems as the *whois* database. Also as the algorithms used in Edge-Space, Digital Island and others are proprietary [6] [7], it is difficult to obtain and study them.

Many of these algorithms have one basic problem that they cannot distinguish between nodes with its own IP address and nodes behind proxies. Nodes are behind proxies in for most of the large ISP. For example for AOL, there is a centralized cluster of proxies in Virginia, therefore using the above techniques for a AOL IP address, will lead us to Virginia irrespective of where the actual node is located. In many cases the errors may be many thousands of miles to the actual location. Thus we see here that proxies pose big problem to IP-location mapping. Any network route analyzing techniques (traceroute) and network delay based techniques (GeoPing) will be susceptible. The third algorithm described here (GeoCluster) is not susceptible, as it does not fall in either these categories.

3. IP2Geo Algorithms

As stated before the three IP-location mapping protocols are collectively called IP2Geo. In this section we will explain the algorithms used in the three protocols in detail and then analyze them. But before doing this we need to describe the experimental setup that was used in these algorithms. An important point to be mentioned here is that, because of the lack of structure to IP addresses, an IP address can be located anywhere. Therefore any formal model for this problem is difficult to make. The only way to go by is to perform experiments and find patterns in them that would suit the needs of the protocols. The experimental design that was done in UCB and Microsoft was as follows. From [3] we found out that they had 14 probe nodes scattered all around the continental US. Any measurement to be done on a target IP address would be done from these 14 robe nodes. An important criterion in placement of these probe nodes is where to place them, in the sense that these kinds of experiments should be able to reflect the real world. The traffic pattern that is visible in these experiments should mimic the actual worlds for these experiments to have any real relevance. These experiment where conducted on nodes and

probe placed solely in continental US. But as US is a diverse region, this constraint is not necessarily an impediment to the measurement process. Also there are 265 Web hosts spread all over the US, these are referred to as UnivHosts. The 14 probes will use these 256 hosts to create the initial data that will be used as a standard to locate target IP addresses, using the known location of these 256 hosts.

3.1. GeoTrack

GeoTrack tries to infer the location based on the DNS names of the target hosts or other nearby network nodes. Very often routers are assigned geographically meaningful names. This is definitely not a requirement, but suggested for administrative convenience. For example, *Corerouter1.SanFransico.cisco.cu.net* would correspond to a router in San Francisco. Lots of Experimental results suggest that routers are indeed named according to their locations.

Not all routers contain meaningful names. A router is considered to be *recognizable* if its geographic location can be inferred from its name. Routers whose name does not reveal any meaningful location information are considered *non-recognizable*. GeoTrack is a tool that is based on the trace route application that uses these hints from the recognizable routers to estimate the location of the target host. A network path between the source and the destination nodes are obtained and then the recognizable router names are collected and parsed in order to get the location. The location of the destination host is assumed to be the location of the last recognizable router in that network path.

As an example, the following indicates a run of the Trace-route tool from ASU to www.microsoft.com.

1) **router-11193.inre.asu.edu** [129.219.140.193]
2) **dmz-gw-ascix1.inre.asu.edu** [149.169.254.11]
3) 209.147.191.42
4) **wsip-66-210-249-25.ph.ph.cox.net** [66.210.249.25]
5) **ip68-2-0-66.ph.ph.cox.net** [68.2.0.66]

6)	**chnddsrc01-gew0304.rd.ph.cox.net**	[68.2.14.9]
7)	**chndbbrc01-pos0101.rd.ph.cox.net**	[68.1.0.164]
8)	**fed1bbrc02-pos0102.rd.sd.cox.net**	[68.1.0.167]
9)	**fed1bbrc01-pos0100.rd.sd.cox.net**	[68.1.0.202]
10)	**fed1bbrc01-pos0200.rd.sd.cox.net**	[68.1.0.193]
11)		68.105.31.2
12)		207.46.34.65
13)	**pos15-0.core1.sea1.us.msn.net**	[207.46.33.29]
14)		207.46.36.66
15)		207.46.155.17
16)	* * * Request timed out.	
17)	* * * Request timed out.	
18)	iuscsecurc6m02-vl-101.msft.net	[207.46.129.10]

In our example the source is ASU and the destination is Microsoft. The table above shows the network path between the two. The router names indicated in bold are considered recognizable. We could also point out some routers that are non-recognizable (Shown in red). For these routers the IP address could not be mapped to a DNS name and thus these routers does not indicate any information about its location. Certain routers also do not respond immediately to the trace route packets and thus a *request timed out* occurs. The blue colored router name clearly indicates that this router is in Seatle, USA. This was the last router that is also recognizable before the destination was reached. Thus the location of this router is also considered to be the location of the destination host.
One could easily see from this example that these trials need to performed many times in order to get a good estimate of the destination location.

3.1.1. Extracting Geographic information

Geographic information, if any, is present in the router names as abbreviations of cities, states or countries. Since there is no standard convention followed for these abbreviations, the task of extracting the geographic information from the names is made difficult. Thus in order for the estimate to be accurate one has to rely on empirical results.

Certain results reveal that the abbreviations are normally city codes, airport codes or country codes. These results also indicate that various ISP's name their routers based on the airport code thus making the task of extracting the location information very easy because the airport codes are standardized all over the world. Routers sometimes also contain their country name. In these cases, the country name can be used to verify the location guessed based on the city codes.
For example, *asd-nr16.aus.sjc.kpnqwest.net,* would indicate that the router is in San Jose, CA. But when we verify this fact with the country code obtained from the name, we could convince ourselves that the router is indeed in Australia and not in USA. Thus the combination of these codes could make the errors less and thus making the estimate more accurate.

A table is constructed with the most often used codes of cities, states and countries and one could extract information from the router name and compare against this table to get the location information. This is *better said than done!* Just extracting information from the routers and directly performing a string match wont work. For example, a router could be named *charlotte.ucsd.edu*, if we were to do a longest string match with the elements in the table we could land up indicating that this router is in Charlotte, NC instead of the actual location, which is in San Diego. One possible solution for this problem is to divide the table into groups such that each group corresponds to a specific ISP (e.g. AT&T, Sprint, etc.). When trying to infer the location of a router associated with a particular ISP, only the codes associated with that ISP is considered. Splitting the routers according to the ISP also has one more advantage, the router names are split into multiple pieces separated by dots. Each ISP has its own strict naming conventions. For example the rule of Sprint Link specifies that the location code, if any, will only be present in the first piece from the left (e.g. sl-bb10-sea-9.sprintlink.net containing the code sea for Seattle). Using this information, the string extraction and matching are made easy and accurate.

GeoTrack thus partitions the codes obtained from experiments into sets corresponding to different ISP and uses this list to obtain information about the routers. Obviously the

performance of GeoTrack is dependent upon the intermediate routers participation and also the extent to which one gets a recognizable router as close to the final destination as possible. The complexity of GeoTrack is based on the implementation of the trace route functionality and the methods for string extraction and string matching.

The pseudocode for the GeoTrack is given below. Detailed implementations of the functions are not shown.

GeoTrack Algorithm

```
1.  Algorithm GeoTrack (Source, Destination) {
2.      TraceRoute (Source, Destination)
3.      Extract router IPs
4.      foreach (routers, i, extracted) {
5.          DNS_Name_IP ← getHostbyName(IP_i)
6.          if (DNS_Name_IP = Null) {
7.              router_i is non recognizable
8.              continue
            }
9.          ISP ← CheckISP (router_i)
10.         locationString ← ExtractLocationInfo (DNS_Name_IP)
              /* extracts location information according to that specific ISP's naming
                 convention */
11.           if (locationString is present in Table[ISP]) {
12.               router_i is recognizable
13.               location ←Table[ISP,locationString]
                  /* this returns the location, if any, corresponding to the location
                 code, locationString */
14.               LastRecogLocation ← location
              }
15.           else {
16.               router_i is non recognizable
```

```
                    }
            }
17.         return destination Location  is LastRecogLocation
```

Figure 1: GeoTrack Algorithm

3.2. GeoPing

This is the second algorithm for IP-location mapping. It is based on the assumption that there is a linear correlation between the packet delays between two nodes and the distance between the two nodes. In [3] they have empirically proved that this assumption makes sense and that there is indeed a linear correlation. However if there is network congestion then there are added problems because the delay-distance correlation may not hold. We can alleviate this problem by sending multiple packets to the destination spaced at certain time intervals and take a minimum value of the round trip times. We first take our set of destination nodes called the UnivNodes for which we know the locations. We then find their distances from our set of 14 probe nodes. When we have to find the destination of a target node we perform the same step and match the results with the results from the UnivNode set. The location of the UnivHost machine for which the two results match most closely is the location of the target machine.

Here they use Nearest Neighbor Delay Space approach (an empirical approach) based again on table mapping for finding the distance between two nodes. In NNDS we construct a table for a set of destination hosts (UnivHosts) for which we know the precise location (call it, distance-table). For each destination node in the distance-table we create a distance vector of the form *(DV = {d1, d2, d3…., dn})*, where DV is the distance vector and *d1, d2... dn* are distances of this destination node from the probe nodes *(Probe nodes, are used for making delay measurements for GeoPing and also to initiate traceroute for GeoTrack.)* [2].

Once the above information is tabulated we can move on to map locations of nodes for which we know the IP addresses. When a target host's location is to be found, we create another distance vector for this unknown host by using the same set of the probe nodes.

We call this vector DV1 and it is of the form *(DV1 = {d11, d12, d13…., d1n})*. Now we find the distance of unknown host from each of the destination hosts in the distance-table using the Euclidean distance formula (between DV1 and each of the DV). The destination host, for which this distance formula yields the lowest value, is considered the closest host to the unknown host and the location of the unknown host is considered the same as this host [3]. The GeoPing protocol algorithm is presented below. *This was not available in the literature, we have formulated this ourselves.* It has three parts: Initialization, Delay_Calculation and Location_Finding.

GeoPing Algorithm

```
/*Initialization */
1. delay_vector {d0, d1, d2,… dM} ← delay vector for a UnivHost node.
2.  IPVectorList ← contains location of UnivHost and its delay vector
3.  Algorithm Initialization (locations[N]) {
4.        delay_vector ← Null
5.        index_i ← index of locations, initially null.
6.        index_j ← index of probe nodes. (maximum is 14 here), initially null.
7.        foreach ( index_i & index_i <N) {
8.              foreach( index_j & index_j <M ) {
9.                    delay_vector_index_j ← delay to location[index_i] from probe index_j
                }
10.             Add (location[index_i], delay_vector) to IPVectorList
          }
     }
/* Calculate Delay  */
1. TDV {d0, d1, d2,… dM} ← delay vector for a target node
2.  Algorithm Delay_Calculation (Target_IP) {
3.        new_delay_vector ← Null
4.        index_i ← index of locations, initially null.
5.        index_j ← index of probe nodes. (maximum is 14 here), initially null.
6.        foreach( index_j & index_j <M ) {
```

```
7.              TDV index_j ← delay to Target_IP from probe index_j
            }
    }

/*  Finding actual location */
1.   Distance[N] ← distance vector for all N entries in the IPVectorList
2.   Algorithm Find_Location (TDV, delay_vector, IPVectorList) {
3.          index_i ← is the index of IPVectorList is initially null.
4.          foreach (index_i & index_i < N) {
5.                Distance[index_i] ← √ Σ (TDV j - delay_vectorj)^2
              }
6.           Final_Loc ← IPVectorList[min { Distance }].Location
7.           Print Final_Loc
    }
```

Figure 2: GeoPing Algorithm

The first part was the initialization, which does the job of creating the IPVectorList table with the distance vector of each location in the UnivHost list. The second phase takes a target IP address and finds its distance vector. The third phase takes this target IP distance vector and runs it through the IPVectorList and finds the distance between the table distance vector and target IP distance vector. The location of the UnivHost for which the result is minimum, is taken as target IP address's location.

Complexity and Correctness

The authors in [3] have experimented and found that there can be no mathematical model that can be used to estimate the geographic distance based on network delay. Hence a rigorous proof of correctness is not possible.

The complexity analysis is as follows: Let the number of nodes in the UnivHost list be *p* and the number of probe nodes be *q*.

For the initialization phase there are two loops of size p and q each so the complexity of that part is $O(p*q)$. Similarly for delay calculation of target IP address phase the complexity is $O(q)$ and for the final phase it is $O(p)$. Therefore the total complexity in the worst case of using GeoPing protocol and its related algorithms is

$$O(q*p) + O(q) + O(p) \quad [\text{here } p >> q]$$

$$\rightarrow O(p^2) + O(q) + O(p) \approx O(p^2).$$

3.3. GeoCluster

We now look at the GeoCluster protocol to map a given IP address to a geographic location. The protocol works on the basis of partial IP-to-Location mapping. This means that before this protocol is executed we have to obtain a few IP addresses and their corresponding geographic location data. This information is obtained by asking a set of users about their location and tabulating their IP addresses and the corresponding location information. This is called the partial IP-Location mapping table. For their project the authors of [3] claim that they collected information from hotmail users, fooTV users and bCentral users. GeoCluster is different the previous two algorithms in that it does not use the network delay or in general active network measurement to determine the location of a target IP address.

Initially the IP address space is broken down into geography based clusters. That means that the IP addresses that are grouped together are co-located. We then try to find out the location of a few nodes in the cluster. Formally, given a cluster G, we create a location set that has, as an element, a tuple of the form (<location, number of IP addresses found with that as a location>). Here the set basically represents the frequency distribution of the various locations at which IP addresses have been found in the cluster. We cab represent the location set as follows:

$$Location_G = \{<loc1, n1>, <loc2, n2> \ldots <locn, nn>\}.$$

If threshold is t, if the number of hosts at a given location, in the location set of G, is greater than a threshold value, then the cluster G is located at that location. Therefore if $n2 > t$ then G is located at loc2. If more than one location instances are greater than t,

then we choose the location of *G* as the one with the largest number of instances. Example if n1 > n2 > n6 > t, then we choose the location of G as n1.

But before we reach this stage, we have to look at how clusters are formed. Once the clusters are formed, we have to obtain the location set of the cluster. The address allocation and the routing in Internet is hierarchical. Routing information is aggregated across hosts that are under single administrative domain. An example of this would be, the IP addresses of a university will be aggregated on the address 156.190.0.0/16 instead of 65536 different addresses. This means that all the hosts belonging to the university will be of the form 156.190.x.x. Therefore while doing routing at a domain level all we need to do is to advertise the prefix. As you can see that using Border Gateway Protocol address prefixes (AP), as shown above, helps us in grouping a set of addresses into clusters. *(The BGP protocol is used to perform inter-domain routing. The BGP protocol defines a set of Autonomous Systems (AS) around which routing is done. A BGP router stores a destination AP and all the ASs that lead to it.)* The university here is a geographic cluster with 65536 IP addresses belonging to it. The number 156.190.0.0/16 is the addresses prefix of a BGP. It is of the form *x.y.z.u/a*, where *x.y.z.u* is the IP address slice and *a* is the size of the prefix. To reduce the latency of routing, we can also advertise a specific IP address in the BGP router, that way the routing will be done directly to the computer that to the Ass router and from there to the host in the AS.

We have defined the geographic cluster of IP addresses using the BGP routing information and APs. But even after this grouping we do not know where this cluster is located. For finding this point out we use a Sub Clustering algorithm. This algorithm forms the heart of GeoCluster. One important point to mention here is that, the grouping of IP addresses in geographic clusters using our technique may not give correct results. This is because of our implicit assumption that IP address clusters are actually geographically also co-located. For example our university has a IP address prefix 156.190.0.0/16, there are 65536 addresses in this domain. But they may not be co-located. Some of these IP addresses may be allocated to a different branch of the university that is in a different city. This is the same example as that of use of proxies in

ISP. The previous two protocols where susceptible to this problem. This protocol overcomes this to the extent that it knows when this situation occurs and intimates the source that initiates the search that geographic location cannot be found.

The proxy problems can be addressed using the sub-clustering problem. The sub-clustering algorithm helps in finding the location of a cluster. This is an offline algorithm that runs in the background produces a table that shows a list of domains (AS) and their respective locations. The final IP-location mapping is done using the table produced using the sub-clustering algorithm.

The sub-clustering algorithm works as follows: the algorithm takes as inputs the partial IP-to-Location mappings table and also a list of ASs (which is the geographic cluster determined using the AP information). It produces as output a table that maps each AS to a specific location. For each AP in the AP table we find if there is sufficient consensus about the location of the AP. If there is, we declare the location of the AP in a table. The algorithm is given following the notations [3]:

Notations:

IPLocList: Is the table of partial IP-location mapping that is sorted according to IP address.

BGPAPList: This is the table of APs obtained from BGP routing information. This list contains the IP address prefixes that belong to an AP and hence are co-located.

IPLocAPList: This is the table that is formed by a join of the previous table on AP and IP address of IPLocList.

sameAPList: This table stores the entries of IPLocAPList that have the same AP.

Sub Clustering Algorithm

1. IPLocList → sorted IP-location mapping
2. BGPAPList →APs derived from BGP information.
3. Algorithm sub-clustering (IPlocList, BGPAPList)
   /* initialization */
4. foreach ((IP,location) in IPLocList) {

```
5.      AP ← Longest_Prefix_ Matc (IP, BGPAPList))
6.      Add ((IP,Location,AP) to IPLoc APList)
}
       /* subdivide APs using IPLocAPList  */
7. curAP ← AP in the first entry of IPLocAPList
8. foreach ((IP,Location,AP) in IPLocAPList ) {
9.          if (AP in (IP,Location,AP) == curAP) {
10.                  Add  (IP, Location, AP) to sameAPList
          }
11.       else {
                /* subdivide curAP as appropriate */
12.               if ( |sameAPList| >= cthresh) {
13.                       if (sameAPList is geographically clustered) {
14.                             avgLocation ← average location of the cluster
15.                             Add (curAP, avgLocation) to newAPLocList
                          }
16.                        else {
17.                             divide curAP into two equal halves.
18.                             Recursively test if either or both subdivisions form a
                                 geographic  cluster
                          }
                    }
19.             sameAPList  ← Null
20.             Add (IP, Location, AP) to sameAPList
21.             curAP = AP
              }
    }
   /* newAPLocList is the new list used for IP-to-Location mapping. */
```

Figure 3: GeoCluster sub clustering Algorithm

The working of this protocol is as follows. This algorithm takes two tables as inputs. The first is the IPLocList, which is the mapping of IP addresses to locations. This list is not extensive but has a few 100s of entries. This table will form the basis of our location finding process. The second input is the BGPAPList, which is the table of APs of domains obtained from the E-BGP. This table contains entries for clusters that were discovered from the BGP header information prior to the sub-clustering algorithm. The IPLocList is sorted in ascending order on the IP address.

In the first step, we join the IPLocList and the BGPAPList entries into a new table called IPLocAPList. In the lines 4-6 we take each IP address entry of IPLocList and perform longest prefix match on the BGPAPList's AP. For example consider:

IPLocList

IP Address	Location
128.67.34.90	Loc1
128.67.34.102	Loc2
128.67.36.10	Loc3
…	…
…	…

Figure 4: IPLocList representation

BGPAPList

Address Prefixes
AP1
AP2
AP3
…
…

Figure 5: BGPAPList representation

For each IP address in IPLocList we find to which AP (from the BGPAPList) it belongs by performing longest prefix mapping. By doing so we create a new table called IPLocAPList.

IPLocAPList

IP Address	Location	AP
128.67.34.90	Loc1	AP1
128.67.34.102	Loc2	AP2
128.67.36.10	Loc3	AP3
…	…	…
…	…	…

Figure 6: IPLocAPList representation

IPLocAPList is our working table; it is from this table that we find out the location of a cluster (which is represented by AP).

Now we take the AP of the first entry of the IPLocAPList (call it curAP), and compare it with the AP of subsequent entries (steps 9 -18). Now as the IP addresses are sorted there is a high probability that the next few entries in the IPLocAP table have the same AP as they may belong to the same address block. Therefore as we go down the list we add all the entries to the sameAP table for which the AP is same as curAP. When we reach the first entry where the AP does not match the curAP we count if the size of sameAP is greater than some preset threshold. If it is, then we find out if the AP is geographically clustered just as we did before by creating a location set for the AP and seeing if any one loc is greater than threshold. If it is then that loc is the average location of the AP, then AP and average location values are written into newAPLocList. The sameAPList is cleared. If however, the size of sameAP is less than threshold. Then we split the address prefix into the next level IP address prefix. We perfume this task recursively and we keep on adding the AP to the newAPList as we find their locations. It is possible that a not all AP are mapped to their respective locations.

An example of this could be: Consider an address prefix of an ISP is 152.145.0.0/16. It means that the prefix is 16 bytes long covering the first two octets. Now the ISP has allocated one half of this address space to a customer in NY and the other to one in LA. The customer in NY has an address prefix of 152.145.0.0/17 and the one at LA has 152.145.128.0/17, this is because of the addition of the MSB of the third octet to the

prefix. Now when we had performed partial IP-location mapping we found that 50 addresses with the prefix 152.145.0.0/17 and 10 with the prefix 152.145.128.0/17. Now we start our algorithm at 152.145.0.0/16. We see that when we come down the list, the number of entries in newAPLocList does not equal to threshold of 20. We split the address space into two halves and try again. With the prefix as 152.145.0.0/17 we get 50 entries in newAPLocList. As this value is greater than threshold we can say that prefix 152.145.0.0/17 is in NY, but for the 152.145.128.0/17 we cannot find the location.

Once we have the list of AP to average location we have done the job of finding groups/ clusters of IP addresses and then finding their location. Now to map an IP address to a location we use the longest prefix match to find out to which AP it belongs to in the newAPLocList. The location of the AP will most likely be the location of the target IP address as well.

As we can here that, the correctness of the algorithm depends upon the accuracy of the partial IP-location mapping table. But the algorithm is fault tolerant in that even with few wrong entries in the partial IP-location mapping table, by choosing an appropriate threshold we can make an intelligent guess toward location of an AP and also an IP address.

Complexity and Correctness of GeoCluster

GeoCluster is an algorithm based on partial IP-location mapping information. This data is collected from email and other service providers. There is no way of finding out if any information is correct. This is a weak point in the algorithm that finds the location of an IP address without active network measurements. Because of the empirical nature of this algorithm we cannot actually write a proof for it correctness. There are no set mathematical constructs or models available here that can be used to prove the correctness of this protocol and its allied algorithms.

As for complexity we can prove that in terms of number of entries in the different tables used in the algorithms of GeoCluster. First we will consider the sub-clustering algorithm,

which is the heart of this protocol. Let *n* be the number of entries in the IPLocList, let *m* be the number of entries in the BGPAPList. For the steps which populate the IPLocAPList, we need *m*n* steps. Therefore the complexity of IPLocAPList formation is *O (n*m)*.

The complexity of creating the newAPLocList is as follows: The length of IPLocAPList is *n*. In the worst-case scenario, in each loop, we may have to split the AP into two halves and recursively do the whole thing again. In an IP address, at most three octets can be used for addressing. Therefore, if we are not able to reach the threshold we have to split till the AP has a prefix length of 24 (or 24 – prefix length times). For each prefix length we have to run the loop twice, because a prefix will be split into two halves. Now suppose on an average we stop adding entries to newAPLocList after traversing through *p* (where p <= n) entries in IPLocAPList, we if initial address prefix is a.b.c.d/g.

$2^{(24- g +1)}* p*m*n$. To explain the exponential constant, consider an AP with prefix length of 20. If we are not able to reach a threshold we have to keep splitting the AP till prefix length is 24, each time into two halves forming a binary tree like structure. The number of nodes in the tree is the number of recursive calls and each time p entries are traversed before another recursive call.

In figure 6, the example we see that if the length of prefix was 22 before, in the worst case we have to split the AP into two halves for two rounds producing 6 recursive calls. Therefore we have an exponential constant for the complexity.

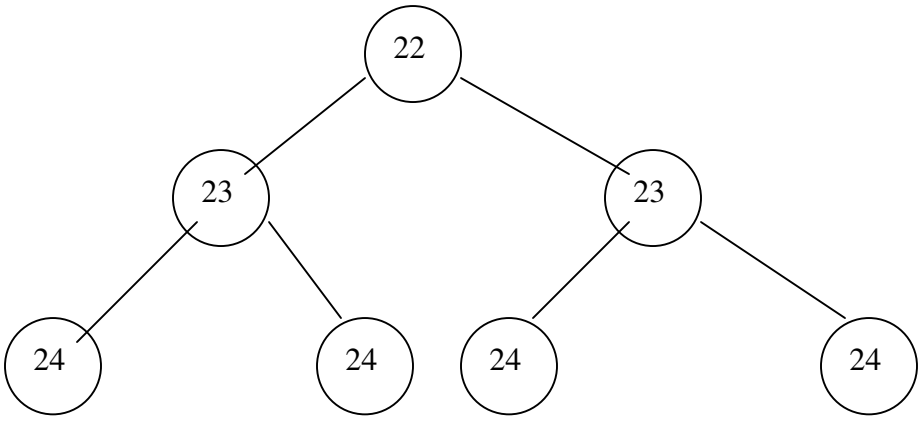

Figure 7: Binary tree representing number of recursive calls.

Therefore the total complexity in worst case for sub clustering algorithm is

$O(n*m) + 2^{(24-g+1)} * p * m*n.$

If we consider the value of p to n/2, then we have complexity as

$O(n*m) + [2^{(24-g+1)*} n * m*n]/2$

➔ $O(n*m) + O(n^2*m) \approx O(m^3)$. *[Because m >> n and if we take m = n*n]* -- (1)

Now given a target IP address we map the IP address to the newAPLocList table. In worst case this may take *r* entries to be visited, before we get our longest prefix match. The value of $r<=m$, as not all entries in the original BGPAPList may be mapped to the IPLocAPList. In the worst case we have to visit all r entries once. Therefore complexity *if O(r), if we take r = m /2 we will have O (m/2)* $\approx$ *O (m).* -- (2)

4. Comparison

So far we have presented three IP-location mapping algorithms. The first two, GeoTrack and GeoPing used active network measurements to find the location of a target IP address. The third scheme of GeoCluster used partial IP-location mapping information collected from service providers to group IP address prefixes into clusters, then we find the location of these clusters using sub clustering algorithm. There are certain difficulties in comparing the three protocols. Even though they perform the same task, they achieve it using different techniques. Therefore we lack any common ground for any comparison measurement.

GeoTrack itself is completely dependent on the implementation of traceroute and other string matching algorithms used. The complexity of GeoPing is dependent on the number of probe nodes used and the number of destination nodes used to create the distance table. Lastly, the GeoCluster algorithm complexity depends on the size of the partial IP to location mapping table and the BGPAPList table.

If we look at the performance intuitively, we see that the GeoCluster algorithm is the most reliable one, even though it is costlier than the others. In the worst case in GeoCluster we have an exponential constant multiplying factor $[2^{(24-g+1)}]$. But then it provides a reasonably sound location prediction, even though it is based on IP-location mapping information obtained from customers of online service providers, the

information is nicely averaged out to produce a result that modulates the granularity of the location accordingly. The biggest problem that we face in mapping IP to location is the presence of proxies. GeoCluster may not solve the problem but it at least detects it, no other technique available is able to do as much.

If we look at GeoPing and GeoTrack algorithms we see that GeoTrack seems to be a reliable source for IP-location mapping information. GeoTrack requires huge amount of research into understanding city, country or basically location codes. But once created they can be pretty reliable. As for GeoPing is concerned, the idea is neat but may not be very practically applicable. It used the network delay measurements to determine locations. But in the chaotic world of Internet the direct co-relation between distance and location may not be correct. In the literature the authors have shown experiments that have proved that delay is proportional geographic distance, but this is not a conclusive proof that indeed there exists such a relationship. Also the authors admit that a mathematical model for the above cannot be created because of the lack of uniform-ness. This makes GeoPing the least likely choice for practical implementation for IP-location mapping.

5. Conclusions and Future Work

In this paper we presented the topic of finding a location for a given IP address. This is a tough problem to crack, for the Internet has not real pattern and also the infrastructure does not provide any means to solve the problem. Updating the infrastructure is one way of achieving the aforesaid but it is very expensive.

In this paper we discuss three algorithms/ protocols that were used for mapping an IP address to a location; they were the GeoTrack, GeoPing and the GeoCluster. We presented the analysis for the three; it was difficult to perform the analysis because of the lack of any mathematical models to achieve it. We presented three pseudocodes for the algorithms used in each protocol and tried to analyze them. Each algorithm had a different metric to measure the complexity, which made it difficult to compare. But by looking at the accuracy of the results we see that GeoCluster is the best followed by

GeoTrack and GeoPing, as far as practical implementation is concerned GeoTrack and GeoCluster are equally implementable, while GeoPing may not be a very good option.

In the future we can come up with new algorithms for IP-location mapping. With the increase in the usage of IPv6, we can implement the above with much more ease. As of now the Internet is growing at a very fast pace, it is therefore possible that the idea behind GeoPing may work because of the huge number of routes available. We can also improve the Euclidian distance measurement scheme that is being used in GeoPing and incorporate a better scheme that would reflect the distances on the Internet.

References:

[1] D. Moore et al. “Where in the World is netgeo.caida.org?”, INET 2000, June 2000

[2] Lakshminarayanan Subramanian, “Geographic Properties of Internet Routing”, http://research.microsoft.com/~padmanab/papers/usenix2002.pdf

[3] Venkata Padmanabhan et al. “An Investigation of Geographic Mapping Techniques for Internet Hosts”, ACM SIGCOMM, SanDiego, USA, 2001

[4] K. Harrenstein et.al., “NICKNAME/WHOIS”, RFC-954, IETF, October 1985.

[5] D.C. Vixie et. al. “A Means of Expressing Location Information in the Domain Name System”, RFC – 1876, IETF, January 1996.

[6] Akamai Inc. http://www.akamai.com/

[7] Digital Island Inc. http://www.digitalisland.com/