Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 63 (2015) 104 – 111

Procedia Computer Science

The 6th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2015)

A Network Topology Discovery Tool for Android Smart Phones

Bilal Saeed[a], Tarek Sheltami[a], Elhadi Shakshuki[b,*]

[a] *King Fahd University of Petroleum and Minerals, Dhahran, 31261, Saudi Arabia*
[b] *Acadia University, Wolfville, NS, B4P 2R6 318 Canada*

Abstract

Android has invaded the majority of the smartphone market. Many developers in this domain are building up applications and addressing many problems. In this paper, we present Topology Discovery Algorithm (TDA) to discover nodes and the topology of the network for the Android Platform. The algorithm is based on Net Inventory System which is used to quickly discover the network topology without generating unnecessary traffic. To speed up the topology discovery process, we used multi-threading; thus, multiple threads run in parallel and handle all the network discovery queries.

Peer-review under responsibility of the Program Chairs

Keywords: Android; TDA; Network; ICMP

1. Introduction

One of the most challenging tasks for network administrators is managing a network. In past, when there were few tools available for probing the network, any hidden error would have caused so much trouble for network administrators. Nowadays, network tools provide network administrators very efficient ways to probe and discover the network. Expensive tools such as HPOpenView [1] are able to pin point the location of the error and send notifications of many events including losses in the networks, utilization of the links, etc. Other tools such as HPOpenView deliver on their purpose; however, they are very costly and not affordable for most small and mediocre size organizations.

Most network discovery tools are developed for desktop computers only and very little attention is given to mobile platforms. In this paper ,we present a Topology Discovery Algorithm (TDA) that uses multi-threading to quickly discover the network topology. We implemented this algorithm for Android hand-held devices. The TDA results are compared with the existing topology discovery algorithms and results have proven to be more efficient than existing algorithms.

* Corresponding author. Tel.: +966-59-842-6200
E-mail address: g201207200@kfupm.edu.sa

Peer-review under responsibility of the Program Chairs
doi:10.1016/j.procs.2015.08.318

2. Related Work

Internet Control Message Protocol (ICMP)[2] is a very basic and easy to implement method of discovering the network. ICMP adds more load and utilizes more bandwidth, because it pings all the active devices in the network successively. The following are three major limitations of ICMP: 1) PING can detect the active devices in the network, but cannot trace the packets routed through routers from the source to the destination. 2) Discovering network devices using ICMP has blindness. For example, ICMP protocol can detect the device of a given IP address, but it is very difficult to find the range of IP subnet. 3) Most of the network firewalls block ICMP packets; hence, ICMP cannot detect devices behind the firewall. ICMP can quickly detect the network devices within the firewall, but it is difficult to create the network devices relationship map.

Simple Network Management Protocol (SNMP)[3] is a network management protocol based on UDP/IP protocols. Devices running SNMP manager are able to extract the routers Management Information Base (MIB)[4], which has support for network information. Currently, most of the network devices have SNMP agents installed. Information on the network topology is contained in the MIB file, which is stored at both agent and management system sides. By accessing the network topology information, we can easily analyze the network devices connections. SNMP has very little overhead on the network and it is also very efficient. However, there are three major limitations of SNMP protocol. 1) SNMP cannot manage devices that do not support SNMP or the SNMP agent is not installed on the device. 2) Routing table may contain a large amount of redundant or useless information. 3) There is a multicast routing problem with SNMP, routing table corresponds to multiple IP addresses. This means that one IP for one port. As a record of factors, routing tables are based on unique IP addresses. Thus, it is not efficient to reflect the topology connection information. SNMP works like a charm for backbone topology discovery. At backbone level it reflects the overall state of the network.

The work proposed by Kuangyu Qin[5] presented an algorithm for Network Topologic Discovery based on SNMP. In the proposed algorithm. the authors assumed that the manager is connected to the border router and sends the first management packet to the router to discover the network. Management station start discovering the network by reading the routing table of the router. It issues an SNMP GETNEXT request to ipRouteTable. In ipRouteTable, there is an object called ipRouteNextHop which tells the next connected router. Thus, by getting the routing table of one router, other routers will be found. ipRouteTable also contains ipRouteDest, ipRouteMask, and ipRouteType. ipRouteType provides the relationship between the router and the network. If it is a direct connection, it means that the network is in its proximity and is adjacent. Since not every PC in the network is running SNMP agent, SNMP cannot be used to get their information. To get these PCs information, ipNetToMediaTable object in the MIB of a switch is used. ipNetToMediaTable contains the records of relationship between switch ports and PCs. Their algorithm is coded in Java using SNMP4J libraries. This algorithm is capable of eliminating redundant devices and generating an efficient topology even when there are VLANs in the network.

In another direction, Narayan et al.[6] proposed an algorithm for topology and service discovery for heterogeneous networks using NetInventory system. Their proposed algorithm is based on the following two assumptions: 1) each switching domain must have one subnet, and 2) address forwarding tables are complete. For the topology discovery, NetInventory system first enumerates all the IP addresses and sends ECHO or ICMP messages. Based on the acknowledgement, NetInventory system determines the nodes are alive. In case PING is disabled, a secondary mechanism is used to determine the presence of nodes. NetInventory uses ipRouteTable and ipNetToMediaTable from a router to discover the liveliness of nodes.

A survey paper by Ahmed et al.[7] provided different techniques and algorithms for network topology discovery. The authors stated that most of the physical topology discovery algorithms are based on SNMP protocol.[8] In their work, the cost and overhead of the de facto standard approach to topology discovery is evaluated and currently implemented by the major SDN controller frameworks. They also proposed simple and practical modifications which achieve a significantly improved efficiency and reduced control overhead

An improved network topology is discussed in Yin et al.[9] for auto-discovery algorithm based on Simple Network Management Protocol(SNMP), Internet Control Message Protocol (ICMP), and Address Resolution Protocol(ARP). The algorithm is based on standard MIB-II and ICMP command, according to TCP/IP protocol addressing relevant principles, to achieve rapid network topology discovery and further reduce the network load. The problem with SNMP based topology discovery algorithm is that it cannot discover devices that do not support SNMP protocol. Xio et al.[10] proposed an Ethernet data link layer topology discovery algorithm based on MAC Address Table. The algorithm is based on graphs, and it iterates through MAC addresses stored in each layer to switch in the network.

3. Proposed Architecture and Contribution

The network topology in which TDA is tested is shown in Figure 1. An Android smart phone running network discovery application is connected to the network through WiFi. The scope of the application is confined to the default gateway; therefore, it cannot discover the network topology beyond gateway - subnet. From Figure 1, it is noticeable that the application is connected to the default gateway and can only discover what is connected up till the gateway. The TDA has been tested in a campus cafeteria area network where 250 devices are connected.

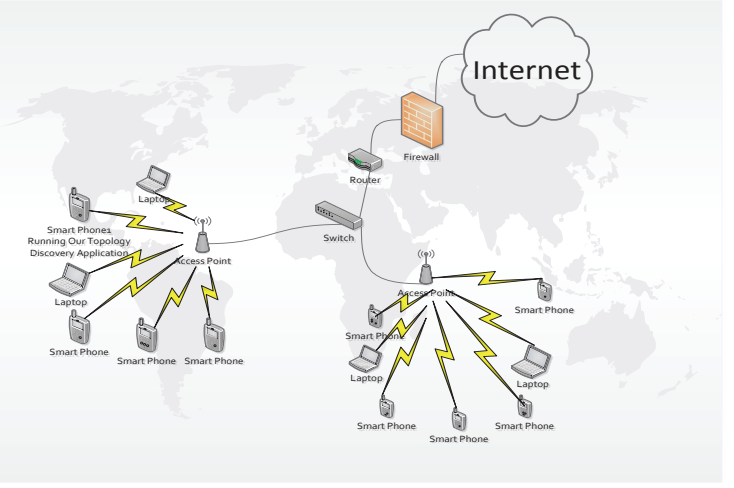

Fig. 1: Network topology

In this paper, Topology Discovery Algorithm (TDA) is proposed for discovering the network nodes. TDA begins by calculating the subnet size of the connected network. Since the proposed approach is based on Net-Inventory System, the algorithm iterate through all the nodes available in the network. After calculating the subnet size of the network, TDA executes n threads in parallel and divides the whole subnet among n threads. Each thread iterate through its assigned subnet and sends ICMP or PING request to its assigned IP addresses. When a node does not respond to PING request, it is most likely that PING is blocked. Consequently, TDA will send ICMP requests on other TCP ports (e.g., 22, 135, 139, 111, 80). If any node still does not respond, TDA marks that node as inaccessible or dead.

If we assume a node is alive and it sends back the response to ICMP request. Then, in this case, the application first checks whether the responding node is a default gateway or a simple PC/Laptop/Phone. It is assumed that the algorithm discovers a router in its first iteration. The TDA updates the DiscoveryTable (DTable) by adding its IP_i to the table along with its MAC address as an Image, and its response time. Since TDA discovers router in the first iteration, it creates a XML file and makes it the parent node of the XML and also saves its IP and MAC addresses in the XML file.

Now, let's consider the case when the responding node is a PC or a laptop. In this case, the algorithm first checks whether the router has been discovered or not. If the router is not discovered yet, the algorithm perfumes an update

in the DiscoveryTable by adding the Host IP, Host MAC and response time. If the router or the default gateway is already discovered, the algorithm also updates the DiscoveryTable as it did when router was not discovered. It also makes this Host the child node of the router in the XML file. It also saves its IP, MAC addresses and response time in the XML. The pseudo code of TDA algorithm is shown in 1. Once the XML file has been created, it is saved in the phone memory card which contains the tags - router being the parent node and all the hosts are its child nodes.

In order to quickly discover all the network devices and topology, TDA assigns the whole network subnet to n threads. As shown in Figure 2, n threads run in parallel and each thread iterate through $\frac{IPSubnet}{n}$ of addresses. All the threads run in parallel and the whole subnet is discovered in a very short amount of time. After threads execution, each thread forwards its results to the application which shows all the discovered nodes.

Algorithm 1 Topology Discovery Algorithm (TDA)

IP[] = ipSubnetRange(IP_1, IP_2, IP_3,....,IP_N) # TDA calculates the IP addresses of its connected Network
Thread N $\in \mathbb{R}$ #Initializes n number of threads assigns each thread $\frac{IP[]}{N}$ number of IP addresses
for all $IP_i \in \frac{IP[]}{N}$ and **for every thread in parallel** # Iterate through every IP
  if IP_i is not iterated yet **then** # If thread has not iterated this IP yet then send ICMP packets
    send ICMP or ECHO to IP_i or try different TCP ports
  end if
  if IP_i is Reachable **then**
    if IP_i is default gateway **then** # If the responding node is a router then make it the parent node of XML
      $XML.Parent \leftarrow IP_i$ # Make this IP_i the root node of XML file
      $DTtable.RouterIP \leftarrow (IP_i)$ # Add this IP_i to the DTable
      $DTtable.RMac \leftarrow ipNetToMedia$ # Resolve IP_i to mac address and save it to DTable
      $DTtable.RouterImage \leftarrow RImage$ # Add router Image to DTable
      $DTtable.ResponseTime \leftarrow RTime$ # Add node response time to DTable
    else if IP_i is a host **then** # If the responding node is a HOST instead of router
      if Router is not discovered yet **then**
        $XML.Child \leftarrow IP_i$ # make this IP_i the child node of Router
        $DTtable.HOST_iip \leftarrow IP_i$ # Add host IP_i to DTable
        $DTtable.HOST_iimage \leftarrow PcImage$ # add host image to the DTable
        $DTtable.Mac_i \leftarrow ipNetToMedia(IP_i)$ # resolve host IP_i to mac address
        $DTtable.ResponseTime \leftarrow RTime$ # add response time to DTable
      end if
    else
      $XML.Child \leftarrow IP_i$
      $DTtable.HOST_iip \leftarrow IP_i$
      $DTtable.HOST_iimage \leftarrow PcImage$
      $DTtable.ResponseTime \leftarrow RTime$
      $DTtable.Mac_i \leftarrow ipNetToMedia(IP_i)$
    end if
  else
    IP_i is marked inaccessible # if IP_i is not reachable mark this IP_i in inaccessible
  end if
UpdateDTable(DTable); # Update DTable
  $i \leftarrow i + 1$ # increment counter and iterate the next IP_{i+1}
end for

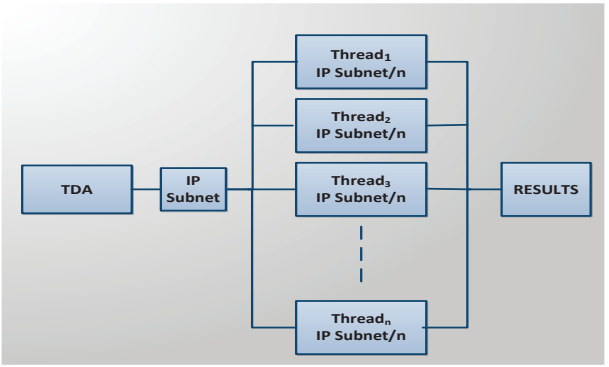

Fig. 2: Multi-Threading

4. Implementation

The hardware specifications used for developing the android application are shown in Table 1. Whereas, the software specifications used are shown in Table 2. First, Eclipse is installed on the laptop which is freely available at[11] along with Android Development Tools (ADT)[12]. ADTs are installed separately to develop Android applications. It is also kept in mind that most Android phones are currently running on Android 4.0+; therefore, we set the minimum Software Development Kit (SDK) to 4.0. This means that any phone that is running Android 4.0 or 4.0+ will be able to run it.

Name	Laptop	Android Phone
CPU	Intel(R) Core(TM) i7-2670 CPU @ 2.20GHz	Quad-core 1.6 GHz
OS	Windows 7 64-bit	Android 4.3
Memory	8GB	2GB
Network Connectivity	Wireless 54Mbps and Ethernet 1Gbps	Wireless 54Mbps

Table 1: Hardware specifications

The Network Mapper(NMAP) and INET API's of Android Development Tools are utilized for discovering the nodes in the network. INET API's enables us to use ICMP in Android; thus, application uses ICMP response to discover nodes in the network. As soon as the application discovers a node, if it is a default gateway, it creates a XML and makes this node (default gateway) the parent node. If the discovered node is a Host or PC, it will make this Host the child node of the router in XML file. All the discovered nodes are saved in a XML file along with their IP address, MAC address, corresponding image, and response time. Once the discovery results are saved in XML, we can use graphical representation tool to see the connectivity among nodes.

Name	Version	Purpose
Android Developer Tools	22.2.1-833290	For Android Libraries
Eclipse	4.2.1	Develop Android Application

Table 2: Software specifications

4.1. Experimental Setup

The hardware and software used in developing the application are described in Table 1 and 2. Android development tools are installed which contain Eclipse in its package. Besides, we installed Software Development Kit (SDK) 4.0, 4.2 and 4.3. The application can be run on any Android phone which is running Android 4.0+.

After setting up the environment, we developed custom Graphical User Interface (GUI) of the application. We tried our best to make the application user friendly so that non-technical people can utilized. In the next section, we describe the functionality of the application and we also show the process of how the application discovers the network devices.

5. Experimental Results

The application is tested in a network where it is connected to an access point and 250 devices that are connected to the network as well. When the user taps the top left icon, it opens a drawer which shows the application functionality. The main functions of the proposed application are:

- **Discover Network** Looks for the nodes available in the network.
- **PING** User can ping any individual host.
- **TraceRoute** User can TraceRoute any IP address.
- **Local Info** Opens a fragment showing the information related to the device, e.g., phone info, WiFi info, location and interfaces' information of the device.
- **About** Pops up a dialog showing the information about application.
- **Settings** Opens a new activity where users can set several parameters for discovering the network.
- **XML** Network discovery results are saved in an XML file
- **Help** Pops up a dialog showing how to use application.

Let's consider the case when a user wants to discover the whole network. In the case of just one thread, the application takes much longer to discover the network. In the case of one thread, TDA takes around 18 minutes to discover 250 nodes, as shown in Figure 3. As we increase the number of threads, the discovery time decreases. It can be seen that in case of 20 threads, TDA takes only 4 minutes to discover 250 nodes.

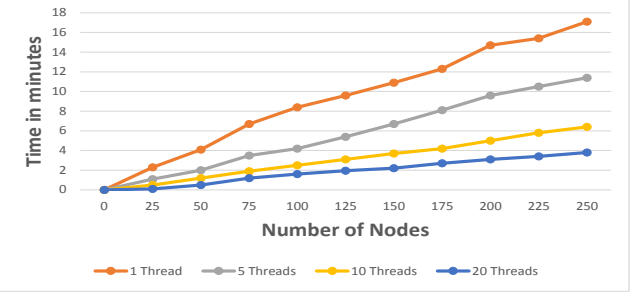

Fig. 3: Threaded Network Discovery

When the user taps the discovery button, the application executes n threads in the background. All the threads are run in parallel and each thread sends ICMP packets to its assigned IP address. As the application starts to getting responses from the nodes, it will resolve host IP to Host name. Figure 4 shows the discovery process of the application. It is worth noting that according to the proposed algorithm, the router or the default gateway is discovered first. Hence, the application in its first iteration sends ICMP request to the default gateway. When the application discovers a router, it creates an XML file in the background and makes it the parent node. As it starts discovering nodes in the network, it keeps updating the GUI (Host table), which contains Host name, Host IP, Host image, and response time. While in the background, it keeps on updating XML file by making each host a child node of the router (default gateway). Once the discovery is finished, the phone will vibrate and all the discovered nodes will be shown.
Figure 5 shows what will appear when the application finishes discovery. All the hosts including default gateway will be shown and the device will vibrate so that user knows the discovery has finished. If the node is alive, it is marked with a Green check. Otherwise a red mark.

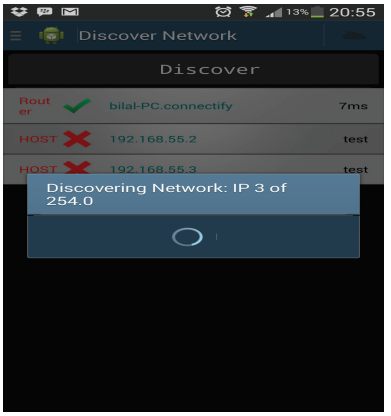

Fig. 4: Nodes are being discovered

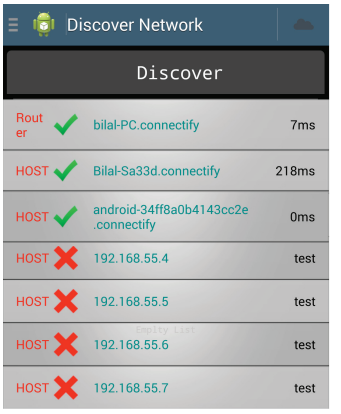

Fig. 5: Discovered Nodes

6. Conclusion and Future Work

In this paper, we discussed a developed Topology Discovery Algorithm (TDA) to discover the network topology. Many of the topology discovery tools to date are based on desktop computers and only a few tools are available for mobile platform. The TDA algorithm smoothly discovers the nodes in the network with quick response time. Moreover, TDA can only discover nodes that are in one subnet. However, we plan to enhance this work so that TDA can discover nodes that are beyond its own subnet. The TDA stores all the discovered nodes in an XML file along with their associated information. In the future, we plan to utilize the XML to plot the topology.

Acknowledgment

The authors would like to thank King Fahd University of Petroleum and Minerals (KFUPM) for supporting the research. The authors would like to also thank the research and graduate office at Acadia University for their support.

References

1. What is HP OpenView. 2012. URL: `http://www.protocolsoftware.com/hp-openview.php`.
2. Postel, J.. INTERNET CONTROL MESSAGE PROTOCOL. 1997. URL: `http://tools.ietf.org/html/rfc792`.
3. J, C.. Simple Network Management Protocal (SNMP). 2009. URL: `http://www.ietf.org/rfc/rfc1157.txt`.
4. ROSE, . Management information base for network management of TCP/IP based internets: MIB2II. 2009. URL: `http://www.ietf.org/rfc/rfc1213.txt`.
5. Qin, K., Li, C.. Network Topology Discovery Based on SNMP. In: *Ubiquitous Information Technologies and Applications (CUTE), 2010 Proceedings of the 5th International Conference on*. 2010, p. 1–3. doi:10.1109/ICUT.2010.5678177.

6. Breitbart, Y., Garofalakis, M., Jai, B., Martin, C., Rastogi, R., Silberschatz, A.. Topology discovery in heterogeneous IP networks: the NetInventory system. *Networking, IEEE/ACM Transactions on* 2004;**12**(3):401–414. doi:10.1109/TNET.2004.828963.
7. Alhanani, R., Abouchabaka, J.. An overview of different techniques and algorithms for network topology discovery. In: *Complex Systems (WCCS), 2014 Second World Conference on*. 2014, p. 530–535. doi:10.1109/ICoCS.2014.7061004.
8. Pakzad, F., Portmann, M., Tan, W.L., Indulska, J.. Efficient topology discovery in software defined networks. In: *Signal Processing and Communication Systems (ICSPCS), 2014 8th International Conference on*. 2014, p. 1–8. doi:10.1109/ICSPCS.2014.7021050.
9. Yin, J., Li, Y., Wang, Q., Ji, B., Wang, J.. Snmp-based network topology discovery algorithm and implementation. In: *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*. 2012, p. 2241–2244. doi:10.1109/FSKD.2012.6233879.
10. Xiao, W., Wang, R., Huang, X.. Design and implementation of ethernet topology discovery algorithm. In: *Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on*; vol. 02. 2012, p. 767–770. doi:10.1109/CCIS.2012.6664279.
11. ECLIPSE, . ECLIPSE IDE. 2002. URL: `http://www.eclipse.org/downloads/`.
12. AndroidSDK, . Android Development Tools. 2005. URL: `http://www.eclipse.org/downloads/`.