Device Discover: A Component for Network Management System using SNMP

A

DISSERTATION

Submitted as a course in partial fulfillment of the

Requirements for the award of the degree

of

MASTER OF ENGINEERING

in

COMPUTER TECHNOLOGY & APPLICATIONS

By

GARIMA CHADHA

College Roll No: 19/CTA/05

University Roll No: 10604

Under the Guidance of

Dr. (Mrs.) Daya Gupta

DEPARTMENT OF COMPUTER ENGINEERING
DELHI COLLEGE OF ENGINEERING
BAWANA ROAD, NEWDELHI
DELHI UNIVERSITY
JUNE – 2008

CERTIFICATE

DELHI COLLEGE OF ENGINEERING
(Govt. of National Capital Territory of Delhi)
BAWANA ROAD, DELHI - 110042

Date:_____

This is to certify that the dissertation entitled **"Device Discover: A Component for Network Management System using SNMP"** has been completed by **Ms. Garima Chadha** in the partial fulfillment of the requirement for the award of degree of **Master of Engineering** in **Computer Technology & Application** of University of Delhi through Delhi college of Engineering under my supervision and guidance.

It is further certified that the work presented here has reached the standard of post graduation and to the best of my knowledge has not been submitted anywhere else for the award of any other degree or diploma.

Dr. (Mrs.) Daya Gupta

Head of the Department and Project Guide

Department of Computer Engineering

Delhi College of Engineering

ACKNOWLEDGEMENT

This work is not a solo endeavor but rather the amalgamate consequence of contribution from various people and sources. Therefore it would be discourteous to present it without acknowledging their valuable guidance.

I would like to express my profound gratitude to my project guide **Dr. (Mrs.) Daya Gupta,** Head of the Department, Department of Computer Engineering, Delhi College of Engineering, for her valuable and inspiring guidance throughout the progress of this dissertation. It is my privilege and honour to have worked under her supervision. Her invaluable guidance and timely discussions in every stage of this dissertation really helped me in materializing this dissertation. It is indeed difficult to put her contribution in few words.

Special thanks are due to my friends and classmates for their unconditional support and motivation during this dissertation.

Garima Chadha

Master in Engineering

(Computer Technology & Applications)

College Roll No: 19/CTA/05

Delhi University Roll No: 10604

ABSTRACT

Virtually all existing networked system management tools use a Manager/Agent paradigm. That is, distributed agents are deployed on managed devices to collect local information and report it back to some management unit. Even those that use standard protocols such as SNMP, fall into this model. Using standard protocol has the advantage of interoperability among devices from different vendors. However, it may not be able to provide customized information that is of interest to satisfy specific management needs. In this dissertation work, different approaches are used to collect information regarding the devices attached to a Local Area Network. An SNMP aware application is being developed that will manage the discovery procedure and will be used as data collector.

TABLE OF CONTENTS

TABLE OF FIGURES

Abbreviations and Acronyms

ASN.1:	Abstract Syntax Notation One.
BER:	Basic Encoding Rules.
FIFO:	First In First Out.
GDB:	GNU Debugger.
GLIBC:	GNU C Library.
ICMP:	Internet Control Message Protocol.
IGMP:	Internet Group Message Protocol.
IPC:	Inter Process Communication,
LAN:	Local Area Network.
MIB:	Management Information Base.
NMS:	Network Management System.
OID:	Object Identifier.
OS:	Operating System.
PDU:	Protocol Data Unit.
PID:	Process ID.
POSIX:	Portable Operating System Interface.
SNMP:	Simple Network Management Protocol
snmpd:	SNMP Daemon.
TCP:	Transmission Control Protocol.
TLV:	Tagged-Length-Value Format.
TTL:	Time to Live.
UDP:	User Datagram Protocol.

CHAPTER 1 - INTRODUCTION

1.1 Network Management:

Now a days, Networks and distributed processing systems are of growing importance and, indeed have become critical in the business world. Within a given organization, the trend is toward larger, more complex networks supporting more application and more users.

A large network cannot be put together and managed by human effort alone. The complexity of such systems dictates the use of automated network management tools.

The **Simple Network Management Protocol** (SNMP) is one step in the direction of standardization of Network Management. SNMP has been developed to provide a tool for multi-vendor, interoperable network management.

1. **Functional Areas and requirements:**

   Network management broadly includes five functional areas and requirements:

   - Fault Management: The facilities that enable the detection, isolation, and correction of abnormal operation of the OSI environment.
   - Configuration and Name Management: The facilities that exercise control over, identify, collect data from, and provide data to managed object for the purpose of assisting in providing for the continuous operation of interconnection services
   - Accounting Management: The facilities that enable charges to be established for the use of managed object and cost to be identified for the use of managed objects.
   - Performance Management: The facilities that needed to evaluate the behavior of managed objects and the effectiveness of communication activities.
   - Security Management: The facilities that address those aspects of OSI security essential to operate OSI network management correctly and to protect managed objects.

1.1.1 Network Management Systems:

A network management system is an integrated collection of tools for network monitoring and control. A network management system consists of incremental hardware and software additions implemented among existing network components. The software used in accomplishing the network management task resides in the host computers and communication processors (for example, front-end processors, terminal cluster controllers, bridges, and routers). A network management system is designed to view the entire network as a unified architecture, with addresses and labels assigned to each pint and the specific attribute of each element and link known to the system. The active elements of network provide regular feedback of status information to the network control center.

Figure 2 suggest the architecture of network management system. Each network node contains a collection of software devoted to the network management task, called as network management entity (NME). Each NME perfume the following tasks:

- Collect statistics on communications and network related activities.
- Store statistics locally.
- Responds to commands from network control center, including commands to:
    - Transmit collected statistics to network control center.
    - Change a parameter (for example, a timer used in transport protocol).
    - Provide status information (for example, parameter values, and active links).
    - Generates artificial traffic to perform a test.
- Send message to the network control center when local conditions undergo significant change.

At least one node in the network is designated as the network control host, or **manager.** In addition to the NME software, the network control host includes a collection of software called the network management application (NMA). The NMA include an operator interface to allow an authorized user to manage the network. The NMA responds to user commands by displaying information and/or by issuing commands to NMEs throughout the network. This communication is carried out using an application level network management protocol (for example, SNMP) that employs the communication architecture in the same fashion as any other distributed application.

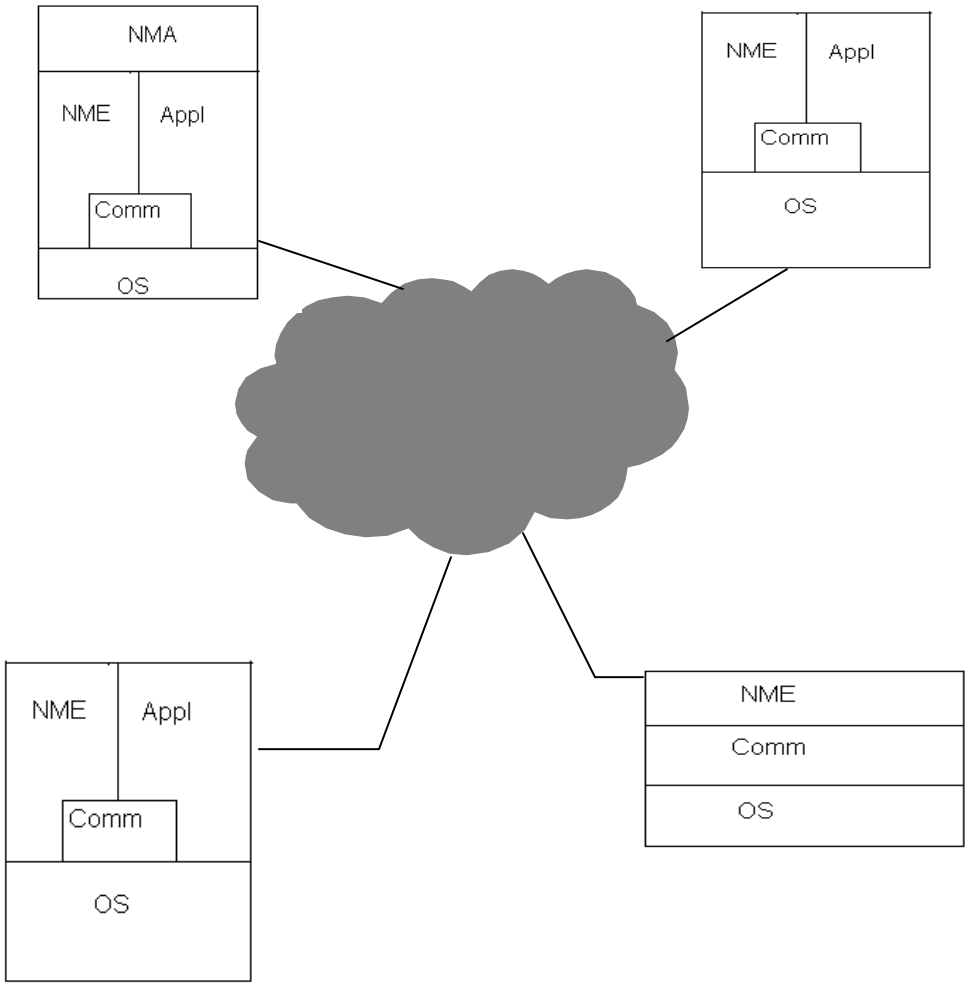


Figure 1: Elements of NMS

Other nodes in the network that are part of network management system include an NME that responds to requests from a manager system. The NME in such managed system is referred to as an agent module, or simply an *agent*. Agents are implemented in end systems that support end user applications as well as nodes that provide ab communication service, such as front-end processors, cluster controllers, bridges and routers.

For maintaining high availability of the network management function, two or more network control hosts are used. In normal operation, one of the centers is idle or simply collecting statistics, while the other is used for control. If the primary network control host fails, the backup system can be used.

The actual architecture of the network management software in a manager or agent varies greatly, depending on the functionality of the platform and details of the network management capability.

Network Management Platforms

A network management platform deployed in the enterprise manages an infrastructure that consists of multi-vendor network elements. The platform receives and processes events from network elements in the network. Events from servers and other critical resources can also be forwarded to a management platform. The following commonly available functions are included in a standard management platform:

- Network discovery.
- Topology mapping of network elements.
- Event handler.
- Performance data collector and grapher.
- Management data browser.

This research work will focus toward the first function of the most of the NMS systems i.e. Network Discovery. The discovery function of most network management platforms is intended to provide a dynamic listing of devices found in the network. Discovery engines such as those implemented in network management platforms should be utilized. A discovery engine provides detailed configuration information of network devices. Common information includes models of hardware, installed modules, software images, microcode levels, and so on. All these pieces of information are crucial in completing tasks such as software and hardware maintenance. The up-to-date listing of network devices collected by the discovery process can be used as a master list to collect inventory information using SNMP or scripting.

1.1.2 NMS Discovery

In many cases, changes are made to individual switches via the EMS (usually via the onboard CLI) and unless the user manually updates the NMS, then the EMS-NMS pictures may differ. This is where an NMS feature called network discovery is important. Network discovery

is the process by which an NMS uses SNMP (and also ICMP) to read, process, and store the contents of designated MIB tables. In this way, the NMS picks up any changes made via the EMS. This process of ongoing discovery and update is an important aspect of managing large networks. Network discovery also picks up the details of both simple objects and complex aggregate objects.

Not all NMS products provide automatic network discovery, because it introduces traffic into the managed network. Also, the workflows of the operator may (manually) provide the same service with no need for an automated solution.

So (using either manual or automatic network discovery), we now have our picture of the network and its higher level constructs (including aggregate objects such as VLANs and VPNs). Having a clear picture of the network objects leaves the operator free to effectively manage the network.

What kinds of things can the operator expect to happen to the network? Links and interfaces can go down; for example, if Link 1 goes down, the NMS should receive a notification from the network that the link has gone down. The NMS then has to cross-reference the notification with the associated aggregate object and infer that a particular LAN is no longer connected to the network. Similarly, if an NE in one VLAN becomes faulty—for example, if a NIC starts to continually broadcast frames—then the NMS should figure this out (by looking at interface congestion indicators) and reflect it back to the user. The user can then resolve the problem. This research work will targeting the NMS discovery feature, it will detect the network devices attached to a LAN and then keep track of the necessary information about the devices (for example, IP address, Links status, Ethernet address etc.).

1.2 Motivation

Complete and current knowledge of the network's topology, host availability, available protocols and services, and detailed application information, is required with minimal impact on the network and its devices. This knowledge could provide the solid understanding of what constitutes normal network traffic and behavior, which is critical for IDS and IPS systems to accurately and efficiently detect possible threats and respond appropriately. 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 sizing. Traditional topology discovery algorithms are based only on SNMP, which is not universally deployed. 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 of the older machines, and in newer machines, SNMP may be turned off or have restricted access.

Also, standard protocols have the advantage of interoperability among devices from different vendors. However, they may not be able to provide customized information that is of interest to satisfy specific management needs. Therefore, **an approach is proposed where the network discovery is done using the ICMP Echo Scan technique and SNMP Ping is then used to extract some more customized information related to device being discovered.**

1.3 Related Work

Knowledge of a network can be acquired by a variety of techniques that can be placed in two general categories:

- Passive Discovery Techniques.
- Active Discovery Techniques.

Passive Network Discovery is one of the important techniques in the field of Network device discovery. Passive network discovery and monitoring is a technology that processes captured packets from a monitored network in order to gather information about the network, its active elements, and their properties. It is usually installed at a network chokepoint. By definition, a passive system will analyze and draw conclusions about a monitored network, its elements, and their properties from network traffic observed at a monitoring location on the network.[1] Consequently, a passive solution cannot draw conclusions about an element and/or its properties if the related network traffic does not go through the monitoring point. Moreover, information that needs to be collected by a passive system might never be gathered, if there is no

[1] C Brain, "Algorithms and Techniques Used for Auto-discovery of Network Topology, Assets and Services", March 16, 2006.

network activity to disclose the information.[2] Passive techniques will always be dependent upon the timely availability of network traffic for analysis. Without any traffic to analyzed, there is no information about the network to be gleaned from it. A passive discovery system has no control over the type of information that passes through its monitoring point and its initiation.
A passive network discovery system cannot detect the physical network topology of a network it is monitoring, for several key reasons:

- It cannot detect the network switches that operate on the network.
- A passive system cannot query switches for their CAM tables, detecting which network element (or elements) are connected to which switch port.

The underlying methodology behind active profiling techniques involves actively probing a target device, for which there is a wide variety of techniques, and then analyzing the device's response. Through the use of these techniques, various pieces of information about the network and its devices can be discovered, including the network topology, device availability, the protocols in use, and much more.

The large number of active scanning techniques such as Network Topology Discovery, Host Link Layer Discovery, Host Internet-work Layer Discovery, Host Transport Layer Discovery, Host Application Layer Discovery etc. can be partly attributed to all the time people have put into developing and investigating them over the years.[3] As new discoveries were made, old techniques were improved or replaced until we arrive at the set of techniques we have today. Even with years of refinement, active techniques still invariably have an impact on the network. It is an unavoidable part of the underlying methodology, and as such its effect can be minimized but never full eliminated.

On the basis of analysis carried out till now, I propose the below mentioned approach towards the Network Discovery which will not only provide the updated and accurate information about the network devices, without consuming a large number of network resources, but also provide customized information about the devices which can be used by the management to take effective decisions.

[2] Annie De Montigny-Leboeuf and Frédéric Massicotte, Passive Network Discovery for Real Time Situation Awarenes., 2004
[3] C Brain, "Algorithms and Techniques Used for Auto-discovery of Network Topology, Assets and Services", March 16, 2006.

1.4 Proposed Work

Knowledge of the underlying network is becoming one of the key requirements of the new generation of Intrusion Detection Systems (IDS) and Intrusion Prevention Systems. Lack of this knowledge can result in numerous ambiguities when interpreting alerts and making decisions on adequate responses. Acquiring this knowledge can be accomplished by a variety of techniques that can be placed in two general categories: active and passive. The general methodologies behind these two categories, as well as the various techniques within each, have their own distinct advantages and disadvantages, which determine their applicability within real world implementations. Through careful examination of the characteristics of these techniques, the design of new IDS and IPS implementations can be improved, as well as making it easier for organizations to determine the best solution for their specific network security needs.

The work carried out is towards the Network discovery feature of the Network Management Systems. Different network discovery approaches (for example, passive discovery techniques, and active profiling techniques) are studied and analyzed; their advantages and shortcomings are explained.

The process being defined uses a combination of various techniques for the network discovery in the targeted system. ICMP echo scan (or ping scan) is used to discover the active devices on the network. In order to find extra information about the devices discovered ICMP scan is followed by SNMP Scan. SNMP Scan is used to obtain objects of 'system' group. This provides information about the device and system software running on the device. Whole discovery process is controlled and monitored via SNMP manager. And a non-standard MIB is designed and implemented to allow SNMP entities to control the process. This MIB include objects, and set/get on these objects allows managing the discovery process.

The traditional SNMP scanners are very inefficient as they spent a considerable amount of time waiting for replies. But SNMP scanner implemented in this work takes a different approach to SNMP scanning by taking advantage of the fact that SNMP is a connectionless protocol and sends all SNMP requests as fast as it can. The technique can be used by Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) in obtaining customized information about the devices on the network and preventing numerous ambiguities when interpreting alerts and making decisions on adequate responses.

1.5 Organization of Dissertation

This Dissertation mainly covers the network management concepts and network discovery techniques in detail.

Chapter 1 is the introductory chapter that gives the overview of the Network Management Systems and Network Discovery.

Chapter 2 presents the SNMP Network Management Concepts that is used for TCP/IP network management. The basic structure and advantages of the distributed network management system are also discussed in chapter 2.

Chapter 3 gives an understanding of the Network device discovery. It also discusses the various techniques of network discovery divided under the categories, Active Discovery Techniques and Passive Discovery Techniques.

Chapter 4 provides complete details about the new approach and an outline of the device discovery process that has been adopted in this dissertation work. It also discusses the overall architecture of Device Discovery.

Chapter 5 discusses the design and implementation aspects of all components of Device Discover in the proposed approach.

Chapter 6 explains the compilation and usage of Device Discover.

Chapter 7 is the concluding chapter that discusses the conclusions and summarization of the work and the future research directions.

Chapter 8 lists the papers that have been published during the preparation of the thesis.

Chapter 9 gives the list of references.

CHAPTER 2 - SNMP NETWORK MANAGEMENT CONCEPTS

2.1 Distributed Network Management:

The configuration depicted in the figure1 suggests a centralized network management strategy, with a single network control center and perhaps a standby center. This is the strategy that has traditionally been favored by both mainframe vendor and information system executives. A centralized network management system implies central control. A centralized network management system enables the manager to maintain control over the entire configuration, balancing resources against needs and optimizing the overall utilization of resources. But recently there is a shift from this traditional strategy to towards a distributed management system.

A distributed management system replaces the single network control center with interoperable workstations located on LANs distributed throughout the enterprise. This strategy gives departmental-level managers, who must watch over downsized applications for their local end users. A hierarchical architecture is typically used, with the following elements.

- Distributed management stations are given limited access for network monitoring and control, usually defined by departmental resources they serve.
- One central workstation, with a backup, has a global access rights and ability to manage all network resources. It can also interact with less-enabled management station to monitor and control their operation.

While maintaining the capacity for central control, distributed approach offers a number of benefits

1. Network management traffic overhead is minimized. Much of the traffic is confined to the local environment
2. Distributed management offers greater scalability. Adding additional management capability is simply a matter of deploying another inexpensive workstation at desired location.
3. The use of multiple, networked station eliminates the single point of failure that exists with centralized schemes.

Figure2 illustrate the basic structure used for most distributed network management system. Closest to users are management clients. These give the user access to management services and information and provide an easy-to-use graphical user interface. Depending on the access privileges, a client workstation may access one or more management servers. Management servers are the heart of the system. Each server supports a set of management applications and a management information base (MIB). They also store common management data models and route management information to applications and clients. Those devices to be managed that share the same network management protocol as the management server contain agent software and are managed directly by one or more management server. For other devices, management server can only reach the resources through a vendor-specific element manager, or proxy.

The distributed management model is flexible and scalable. As additional resources are added to the configuration, each is equipped with the agent software or linked to a proxy. In a centralized system, this growth might eventually overwhelm a central station. But in a distributed system, additional management servers and client

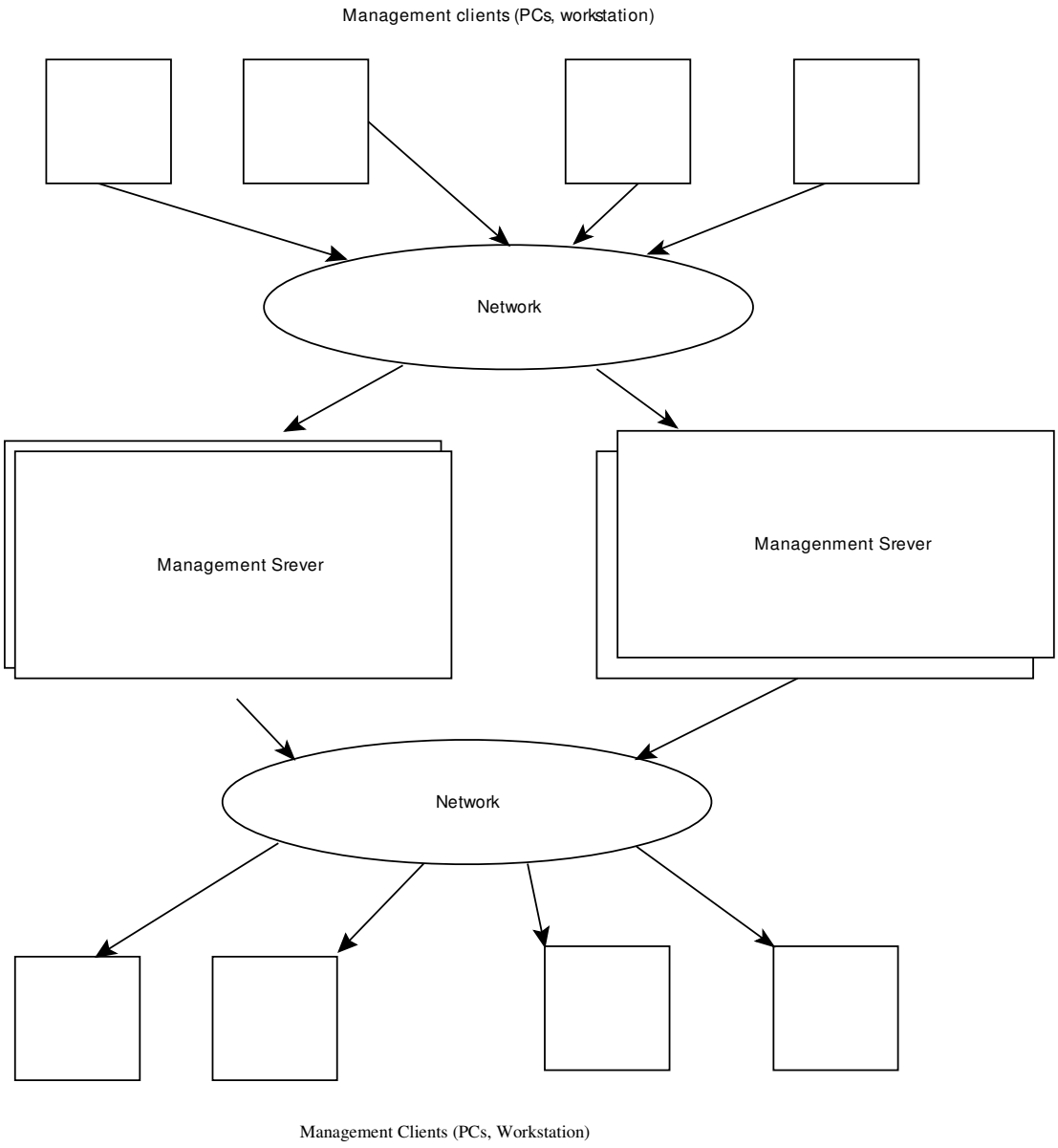


Figure 2: Distributed Management System Arch.

Workstations can be added to cope with the additional resources. Furthermore the growth of overall configuration will occur in a structured way (for example, adding an additional LAN with a number of attached PCs); the growth of the management system mirrors this underlying resources growth, with servers and clients added where the new resources are located.

2.2 SNMP Model of Network Management:

The model of network management that is used for TCP/IP network management includes the following key elements:

- Management station.
- Management agent
- Management information base
- Network management protocol.

The management station serves as the interface for the human network manager into the network management system. At a minimum management station will have,

- A set of management application for data analysis, fault recovery, and so on.
- An interface by which the network manager may monitor and control the network.
- The capability of translating the network manager's requirements into the actual monitoring and control of remote elements in the network.
- A database of information extracted from the MIBs of all managed entities in the network.

The other active element in the network management system is the *management agent*. Key platforms, such as hosts, bridges, routers, and hubs, may be equipped with SNMP agent so that they may be managed from management station. The management agent responds to request for information and actions from management station and may asynchronously provide the management station with important but unsolicited information.

Resources in the network may be managed by representing those resources as objects. Each object is, essentially, a data variable that represents one aspect of the managed agent. The collection of objects is referred to as a *management information base* (MIB). The MIB functions as a collection of access points at the agent for the management station. These objects are standardized across systems of a particular class (for example, a common set of objects is used for the management of various bridges). A management station performs the monitoring function by retrieving the value of MIB objects. A management station can cause an action to take place

at agent or can change the configuration settings at an agent by modifying the value of specific variables.

The management station and agents are linked by a *network management protocol*. The protocol used for the management of TCP/IP networks is the Simple Network Management Protocol (SNMP), which includes the following key capabilities.

- Get: enables the management station to retrieve the value of objects at the agent.
- Set: enables the management station to set the value of objects at the agent.
- Trap: enables the agent to notify the management station of significant events.

SNMP was designed to be an application-level protocol that is part of the TCP/IP protocol suite. It is intended to operate over the User Datagram Protocol (UDP). Each agent must implement SNMP, UDP, and IP. In addition, an agent process interprets the SNMP messages and controls the agent's MIB. For an agent device that supports other applications, such as FTP, both TCP and UDP are required.

Figure 3, provide a somewhat closer look at the protocol context of SNMP. From a management station, three types of SNMP messages are issued on behalf of a management application: GetRequest, GetNextRequest, and SetRequest. The first two are two variations of get function. All three messages are acknowledged by the agent in the form of a GetResponse message, which is passed up to the management application. In addition, an agent may issue a trap message in response to an event that affects the MIB and the underlying managed resources.

Because SNMP relies on UDP, which is a connectionless protocol, SNMP is itself connectionless. No ongoing connections are maintained between a management station and its agent. Instead, each exchange is a separate transaction between a management station and an agent.

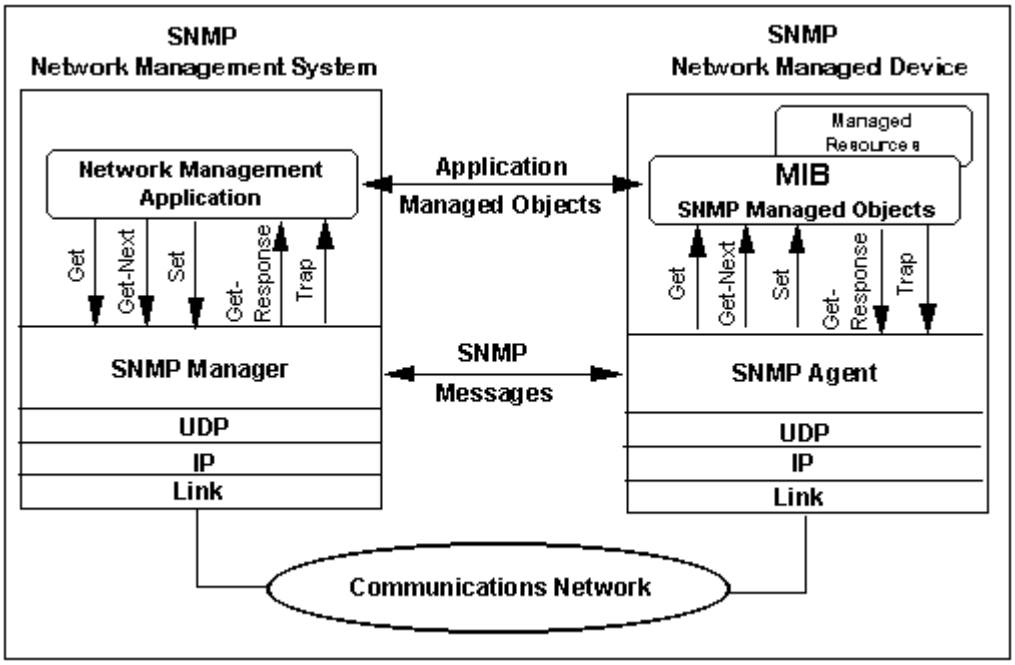


Figure 3: SNMP Protocol

2.2.1 SNMP: - Trap-Directed Polling

If a management station is responsible for a large number of agents, and if each agent maintains a large number of objects, then it become impractical for the management station to regularly to poll all agents for all of their readable object data. Instead, SNMP and the associated MIB are designed to encourage the manager to use technique referred to as *trap-directed polling.*

The preferred strategy is this. At initialization time, and perhaps at infrequent intervals, such as once a day, a management station can poll all of the agents it knows for some of the key information, such as interface characteristics and perhaps some baseline performance statistics, such as average number of packets sent and received over each interface over a period of time. Once this baseline is established, the management station refrains from polling. Instead, each agent is responsible for notifying the management station of any un-usual event; for example, the agent crashes and is rebooted, a link fails, or an overloaded condition as defined by the packet load crosses some threshold. These events are communicated to management station in SNMP messages known as *traps.*

Once a management station is alerted to an exception condition, it may choose to take some action. At this point, the management station may direct poll to the agent reporting the event and perhaps to some nearby agents in order to diagnose any problem and to gain more specific information about the exception condition.

Trap-directed polling can result in substantial savings of network capacity and agent processing time. In essence, the network is not made to carry the management information that the management station does not need, and agent are not made to respond to frequent requests for uninteresting information.

2.2.2 SNMP: - Proxy Configuration

The use of SNMP requires that all agents, as well as management station, must support the common protocol suites, such as UDP and IP. This limit direct management to such devices and exclude other devices, such as some bridges and modems that do not support any part of TCP/IP protocol suite.

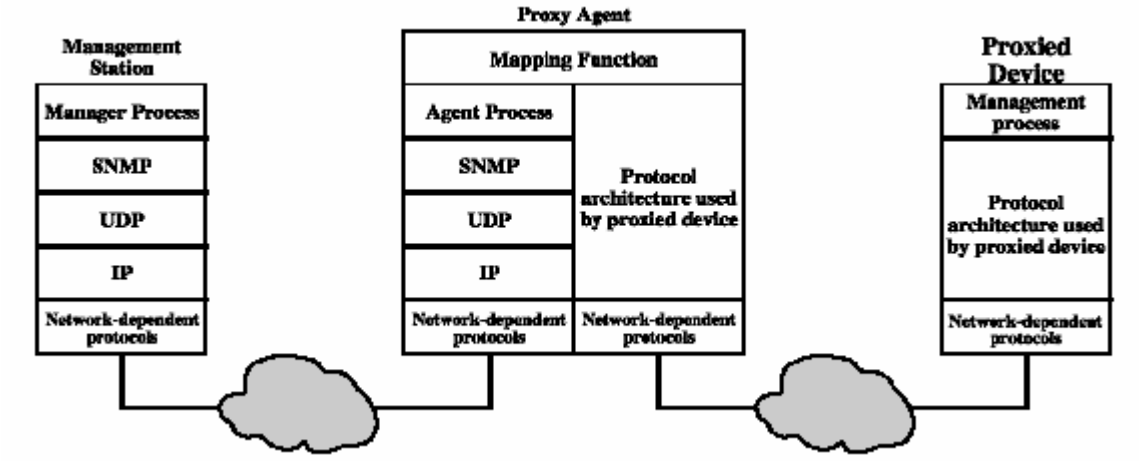


Figure 4: Proxy Configuration

To accommodate devices that do not implement SNMP, the concept of proxy was developed. In this scheme an SNMP agent act as a proxy for one or more other devices; that is, the SNMP agent acts on behalf of the peroxide devices.

Figure 4 indicates the type of protocol architecture that is often involved. The management station sends queries concerning a device to its proxy agent. The proxy agent

converts each query into the management protocol that the device is using. When the agent receives a reply to a query, it passes that reply back to management station. Similarly, if an event notification of some sort from device is transmitted to proxy, the proxy sends that on to the management station in the form of trap message.

SNMP agent makes use of SNMP-PROXY-MIB[4] to configure the parameters used by proxy forwarding application.

2.2.3 SNMP: - PDU description

For completeness, here we will have a brief description of the SNMP PDU[5] (Protocol Data Unit) for most common SNMP-v1 messages i.e. Get, GetNext, Set and Trap PDUs.

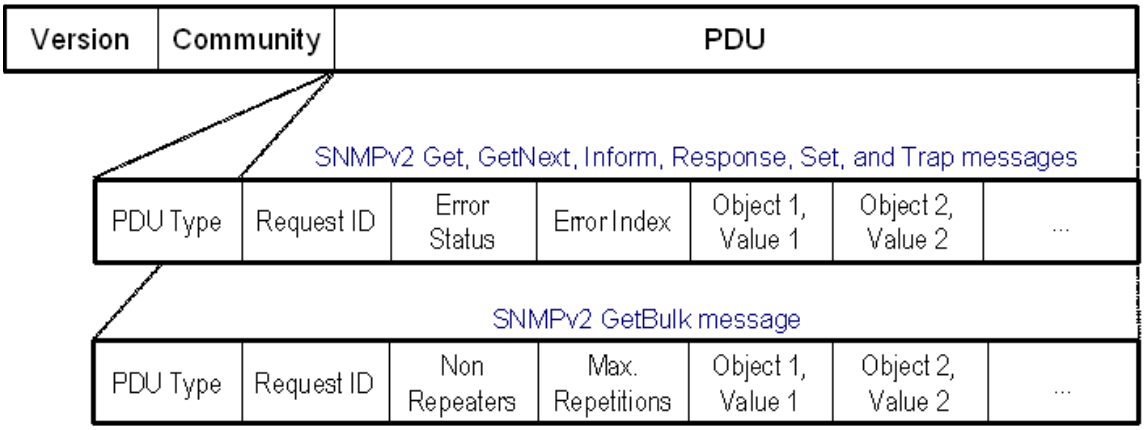


Figure 5: SNMP Message formats

[4] RFC 2273 defines the full PROXY MIB
[5] RFC 1155, 1157 explains SMIv1 and basics of SNMP protocols

CHAPTER 3 - NETWORK DEVICE DISCOVERY

Enterprise networks have gotten so complex that it is rare that any single person knows exactly what is connected to them. That could become an issue, particularly if someone brings an infected PC or if disaster strikes and a portion of the network go south.

Knowledge of the underlying network is becoming one of the key requirements of the new generation of Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS). Lacking this knowledge can results in numerous ambiguities when interpreting alerts and making decisions on adequate responses. Acquiring this knowledge can be accomplished by a variety of techniques that can be placed in two general categories:

- Active Discovery Techniques.
- Passive Discovery Techniques.

The general methodologies behind these two categories, as well as the various techniques within each, have their own distinct advantages and disadvantages, which determine their applicability within real world implementations.

3.1 Passive Network Discovery

Passive Network Discovery[6] is one of the important techniques in the field of Network device discovery. Passive network discovery and monitoring is a technology that processes captured packets from a monitored network in order to gather information about the network, its active elements, and their properties. It is usually installed at a network chokepoint.

The kind of information collected through passive network discovery and monitoring might include the following:

- Active network elements and their properties (e.g.., underlying operating system).
- Active network services and their versions.
- The distances between active network elements and the monitoring point on the network.
- Active client-based software and their versions.
- Network utilization information.

[6] Demystifying passive network discovery and monitoring systems, Ofir Arkin

- Vulnerabilities found for network elements residing on the monitored network

Such information can be used for the following purposes:

- Building the layer 3–based topology of a monitored network.
- Auditing.
- Providing network utilization information.
- Performing vulnerability discovery.
- Enhancing the operation of other security and/or network management system by providing context regarding the network they operate in (information about the network, the active element found on the network, and their properties).

3.1.1 Advantages of Passive Network Discovery:

Passive network discovery and monitoring system have important advantages related to their mode of operation.

Real-time operation: The operation (i.e., processing received network traffic and providing relevant information) is performed in real-time.

Zero performance impact: A passive network discovery and monitoring system has zero impact on the performance of the monitored network. This is because the monitored network's traffic is copied and fed into the system, the operation of which involves no active querying. This all means that passive monitoring poses no risk to the stability of a monitored network and can theoretically be installed on any network.

Data processing: Passive network discovery and monitoring systems have the ability to gather information from all TCP/IP layers of network traffic processed.

Detection of active network elements and their properties: A passive network discovery and monitoring system is able to detect network elements along with some of their properties, by observing network activity related to the network element, provided that it is receiving and responding to network traffic. This means a passive system can:

- Detect active network elements that transmit and/or receive data over the monitored network

- Detect network elements as they become active and transmit and/or receive data over the monitored network.

Detection of elements behind network obstacles: A passive system can detect active network elements that operate behind network obstacles and send and/or receive network traffic over the monitored network. A network obstacle is a network element that connects multiple networking elements to a network while filtering traffic from that network to these network elements (which are logically hidden behind it). Network obstacles include a network firewall, a NAT device, and a load balancer.

Granular network utilization information: A passive solution can provide information regarding the network utilization of its monitored network link. Unlike active monitoring solutions, which only provide basic network utilization information regarding the amount of traffic observed over a certain amount of time through SNMP, a passive network discovery and monitoring system supplies network utilization information by observing actual network traffic. A passive system has the ability to supply more granular and detailed network utilization information (for example, per network element, per service, etc.) than active solutions.

Network utilization abnormality detection: The ability to provide statistical information regarding network utilization information, per network element, per network service, and the ability to gather information from all TCP/IP layers, enables a passive solution to build usage profiles for any element using the network and for any service used over the monitored network. These usage profiles can later be used to detect network-related abnormalities.

Detection of NAT-enabled devices: A passive system might be able to discover network address translation (NAT)[7]–enabled devices that operate on the monitored network and to guess the number of network devices they might hide behind them.

[7] NAT – Network Address Translation

3.1.2 Weaknesses of Passive Network Discovery

Although associated with important advantages, passive network discovery and monitoring systems have a number of critical weaknesses that affect their discovery and monitoring capabilities.

What you see is only what you get: By definition, a passive system will analyze and draw conclusions about a monitored network, its elements, and their properties from network traffic observed at a monitoring location on the network. Consequently, a passive solution cannot draw conclusions about an element and/or its properties if the related network traffic does not go through the monitoring point. Moreover, information that needs to be collected by a passive system might never be gathered, if there is no network activity to disclose the information. A passive solution cannot detect idle elements, services, and applications. The discovery performed by a passive system will be partial and incomplete, since it is unable, technologically, to detect all network assets and their respective properties. Finally, a passive system is blind when it comes to encrypted network traffic.

No control over the pace of discovery: A passive discovery system has no control over the type of information that passes through its monitoring point and its initiation. Statistically, certain packets might not pass through the monitoring point for extended periods of time.

Limited IP address space coverage: Lacking control over the type of information that passes through its monitoring point, a passive network discovery system can generically cover only a limited IP address space.

Not everything can be passively determined: In some cases, information cannot be discovered by using passive network discovery. Passive vulnerability discovery is a good example: not all vulnerabilities can be determined passively.

Incomplete and partial network topology: A passive network discovery and monitoring system gathers network topology information based on the distances discovered between network elements and the monitoring point on the network, by relying on the time-to-live field value in the IP header of observed network traffic. The time-to-live field value is decremented from its default value by each routing enabled device that processes the IP header of the packet on its way from the sender to its destination. Some passive network discovery and monitoring systems

first determine the underlying operating system of a certain network element before relying on the time-to-live field value found with network traffic initiated by this network element.
The network topology information provided by a passive system relates only to layer 3–based information, i.e., routing-based information. A passive network discovery system cannot detect the physical network topology of a network it is monitoring, for several key reasons:

- It cannot detect the network switches that operate on the network. Usually a network switch will not generate network traffic other than the spanning tree protocol, sent only to its adjunct switches.
- A passive system cannot query switches for their CAM tables, detecting which network element (or elements) are connected to which switch port.

3.2 Active Network Discovery

The underlying methodology behind active profiling techniques involves actively probing a target device, for which there is a wide variety of techniques, and then analyzing the device's response. Through the use of these techniques, various pieces of information about the network and its devices can be discovered, including the network topology, device availability, the protocols in use, and much more.

3.2.1 Active Profiling Techniques

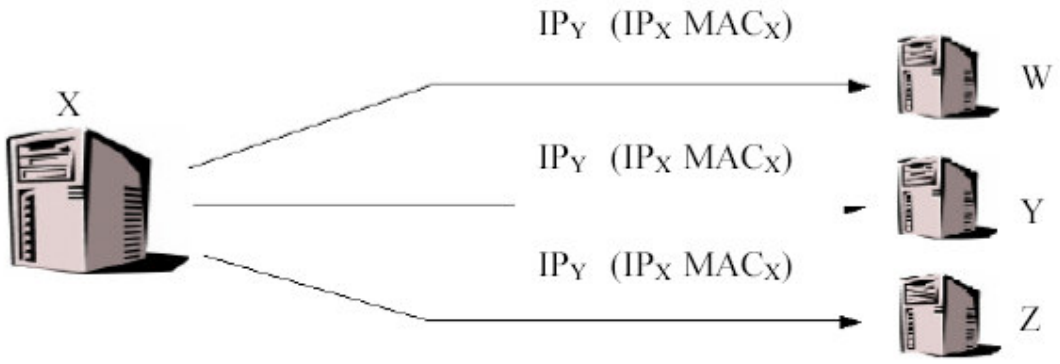
Following are the various Active Discovery Techniques being used commonly today.
Network Topology Discovery:
One common technique for discovering the topology of a network involves sending a series of groups of UDP or ICMP Echo Request packets to a destination host. The initial packets sent, have their IP header Time-to-Live (TTL) fields set to the value one and each subsequent group of packets have their TTL fields set one higher than the previous group.
Each time a packet arrives at a router, the TTL field will be decremented by one and then sent on to the next router or gateway between its current location and the destination host. If a packet arrives at a router and the TTL field has the value zero, the packet is dropped and an ICMP Time Exceeded in Transit error message is sent back to the packet's source IP address. When the

source host receives this error message, it sends out the next group of packets with the incremented TTL field. By examining the source IP address of the ICMP error message, the router that sent it is identified. This process continues until an ICMP Port Unreachable error message or ICMP Echo Reply, depending upon which type of packet is used, is received from the initial destination host. This produces a map of all the routers, gateways and hosts between the originating host and the destination host. Continuing this process for other hosts on the network, a complete map of all active hosts can be obtained. For this technique to function correctly, it requires that there be no prohibitive filtering or packet loss on the network.

Host Link Layer Discovery:

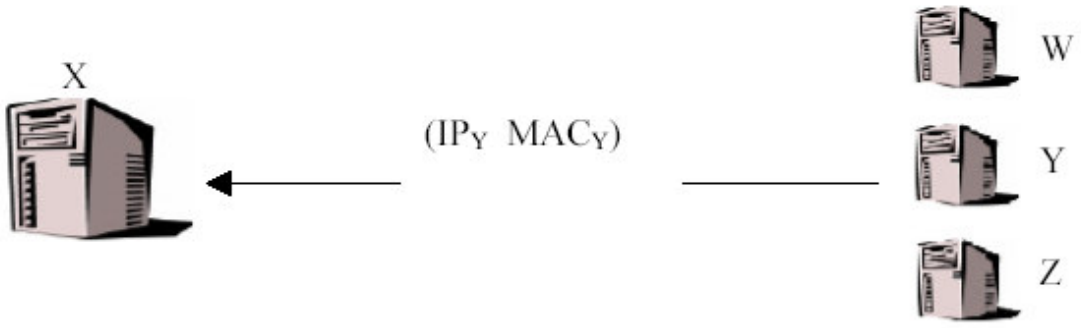
- **ARP Querying:** When a host wishes to communicate with another host on the network but only has knowledge of the other host's IP address, it must first learn the MAC address of the host it wishes to communicate with. As shown in Figure 1a, host X accomplishes this by sending out an

  ARP request broadcast to every host on the network. The ARP request contains the IP address of host Y, the host for which the MAC address is being sought, and the IP and MAC addresses of host X so that host Y will know which host to send the reply to.



ARP Broadcast by Host X

In above figure host Y has seen its IP address in the ARP broadcast and responds to host X with its IP and MAC addresses. All the other hosts on the network who received the ARP broadcast simply ignore it since the IP address in the broadcast did not match their own.



ARP Reply from Host Y

Through systematic querying of all the IP addresses on a network, a database containing the IP addresses and associated MAC addresses for all active hosts can be built. However, it is important to note that this technique relies on the network having a broadcast method, which is not something implemented by all LANs.

- **ICMP Echo Scan:** This technique involves sending an ICMP Echo (ICMP type 8) datagram to the destination host and waiting to see if there is a response. If the target host is active, an ICMP Echo reply (ICMP type 0) is sent back from the target host. If the source host receives no response to its ICMP Echo request, either the target host is inactive or the ICMP protocol is being filtered. Unfortunately, the ability to differentiate between the two causes of a failure by the target to respond with an ICMP Echo reply is beyond the ability of this scanning technique. It is also possible that an ICMP error message will be returned, such as a: ICMP Host Unreachable or ICMP Destination Unreachable port unreachable error. In the case of an ICMP Host Unreachable error, it indicates that the targeted host is either temporarily down or does not exist. For the port unreachable error, it can be determined that the host is alive and reachable, but the port is closed.

  One of the disadvantages of this scanning technique is that in the case where no response is received from the target host, it cannot be determined if the lack of response was due to an inactive or non-existent host or that the ICMP protocol is simply being filtered. However, if access to a network is available without any filtering issues along the connection path, such as firewalls, the ICMP Echo scan technique can be a useful way of creating a list of active stations on the network.

Other Common Active Profiling Techniques includes:

- Host Internet-work Layer Discovery: The purpose of this technique is to identify the various IP protocols (TCP, ICMP, IGMP, etc.) that are supported by the target host.
- Host Transport Layer Discovery.
- Host Application Layer Discovery: This is used for identifying the applications available on a host.

3.2.2 Advantages of Active Network Discovery:

Active network discovery and monitoring system have important advantages related to their mode of operation.

- An active network discovery system queries network elements for specific parameter values needed for the active network discovery process
- The initiation of an active network discovery scan or a specific query can be controlled. An Active Network Discovery System may allow complete control over the parameter values to query for, and the time to perform the queries
- The pace of an active network discovery scan can be controlled
- An active network discovery scan is able to cover the entire IP address range of an enterprise

3.3 Active vs. Passive Techniques

- While the monitoring methods employed in passive techniques create no additional traffic on the network, all active techniques create varying amounts of additional network traffic. As the number of hosts and ports to be scanned increases, the amount of increased traffic can interfere with normal network and host operation.

  Since passive techniques operate using only existing network traffic, they do not suffer from the same sort of difficulties in this regard as do their active counterparts. This lack of disruption to the network and devices connected to the network is one of the main advantages of passive techniques.

- Another major difference between active and passive techniques is in the acquisition of information about the network. Active techniques can be initiated at any time to provide a complete picture of the host or network in question, only requiring the time needed to run the scan. Unfortunately, the information obtained in this manner is only a snapshot of the network and can become outdated shortly thereafter. What this means is that any changes to the network that take place after a scan is run will not be accounted for until another scan is initiated. Contrast this with passive techniques that allow the picture of the network to be updated continuously.
- Passive techniques are only able to detect a host on the network if that host is communicating on the network and the traffic from the host is passing through the point on the network that the sensor is monitoring. As a result of these limitations, it generally takes longer to profile assets using passive techniques than it does using active ones

3.4 Conclusion

This chapter examines the strength and weaknesses of passive and active network discovery technology. It demonstrates that although the passive network discovery technology has some strong advantages, it cannot under any circumstances perform a complete, accurate and granular network discovery due to technological limitations that directly relate to the passive nature of the technology. Also, the active network discovery technology, in-spite of having some disadvantages along with the advantages mentioned, can be a good method to obtain network information, provided, means can be adopted to reduce the load on the network due to actively probing of the network.

CHAPTER 4 - DEVICE DISCOVERY MANAGEMENT

This section describes the basic overview/outline of the device discovery process that has been adopted in this dissertation work. Active discovery technique 'ICMP Echo Scan' extended with SNMP Scanner is being used to perform the discovery task.

Discovery process designed for 'Device Discover' is not limited to 'ICMP Echo Scan', but it also does the SNMP Ping on each device discovered during the ICMP Scan. SNMP Ping is used to extract some more information related to device being discovered.

4.1 Discovery Process - Outline

As described earlier, Device Discover has a Ping Scanner (for doing ICMP Echo Scan) and a SNMP Scanner (For doing SNMP Ping to the devices discovered by Ping Scanner).Overall functionality can be described using Figure 6.

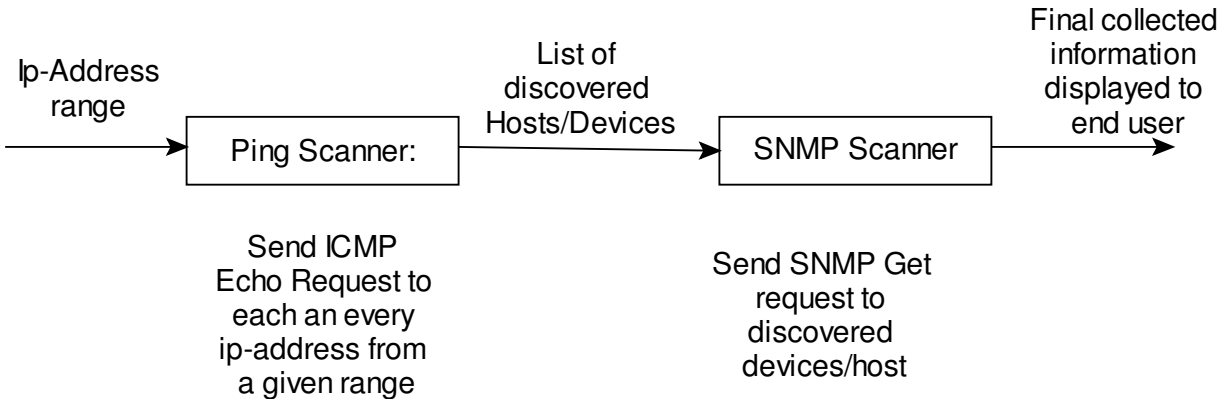


Figure 6: Discovery process outline

As depicted in the figure.

1. On receiving request to start discovery, check to see whether a discovery process is already in progress. If yes, then exit as only one instance of discovery process can run at a time.
2. Else start the discovery process, and pass all the required information to the process (to be discussed later, at a time of describing the design).

3. On start of discovery process, a range of IP-Address will be given as an input to the Ping Scanner module. This range can be specified either as start or an end ip-address or can be input as single ip-address and ask the Ping Scanner to look for all the address in the corresponding network.
4. Ping Scanner will send an ICMP Echo (ping) request to each and every ip-address within the range. And it will send the all ICMP packets in one go and then wait for ICMP echo replies. It will wait for configurable (configurable through FILE, to be discussed later) amount of time before moving on to next step.
5. After Waiting 'Ping scanner' will pass the list of discovered hosts/devices to SNMP scanner to do the further processing.
6. SNMP Scanner will send SNMP Get request (for few fixed OIDs, to be discussed later) on all the discovered hosts/devices in one go, and then wait for configurable amount of time to receive SNMP response.
7. Final collected information is stored in internal data-structures and displayed to user through a third party SNMP Browser, using internally developed MIB (DEV-DISCOVER-MIB).
8. At the end of the discovery process, compare the new data collected with that of the last time. If a new device is discovered i.e a device is discovered that was not present in the data collected during last scan, but present in the information collected during current scan, It means a new device is comes up on the network. A trap 'deviceUp' (as specified in DEV-DISCOVER-MIB) is raised.
9. Similarly, if a device goes down i.e. not discovered during the current scan but present in the last scan, a trap 'deviceDown' is raised.

4.2 Device Discover Architecture

Device Discover provides SNMP Interface for controlling and monitoring device discovery process. To accomplish this, a non-standard MIB DEVICE-DISCOVER-MIB is designed and implemented. A third party freely available SNMP MIB browser (acting as SNMP manager) is used for managing discovery process. DEVICE-DISCOVER-MIB is added on to that browser and same MIB is implemented in the CMU-SNMP agent. Both SNMP Manager and

agent (for demo. purpose) run on same machine. Figure 7 explains the architecture of Device Discover.

As depicted in the figure, Device Discover has following components:

- SNMP Browser (SNMP Manager): SNMP browser is acting as SNMP manager. It is used to start/stop and manage the discovery process. DEVICE-DISCOVER-MIB is compiled into the browser. SimpleMibBrowser™ from SimpleSoft is used for this purpose.
- SNMP Agent: CMU-SNMP (an open source project) is used as SNMP agent. A non-standard MIB i.e. DEV-DISCOVER-MIB is added into the CMU-SNMP. This agent is interacting with the manager to control and monitor the discovery process.
- Discovery Process: It is the main component of the Device Discover. This component includes Ping Scanner, SNMP Scanner and all the data-structure used to store the information collected during discovery process.

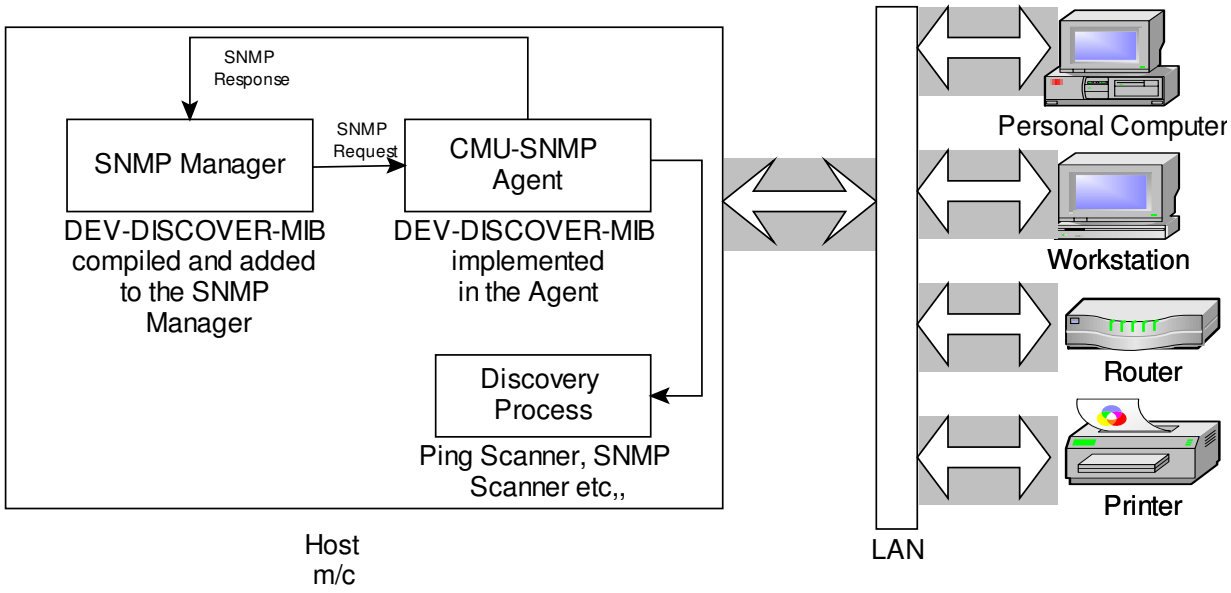


Figure 7: Architecture Diagram

Discovery process runs as a separate thread on the same machine on which SNMP Manager and SNMP agents are running. It directly interacts with the agent, and it does all the functional part of Device Discover.

On starting the discovery procedure via setting an object (described in next section) of DEVICE-DISCOVER-MIB, a SNMP set request is sent to the SNMP agent and then agent triggers the discovery process. Discovery process then performs steps described under 4.1.

4.3 DEVICE-DISCOVER-MIB

This section describe a non-standard MIB i.e. DEVICE-DISCOVER-MIB that is designed and implemented for managing the whole discovery process through SNMP manager. Figure 8 shows the hierarchal structure of this MIB, it shows how each and every object of this MIB are linked together in a hierarchal fashion.

This MIB is added under enterprises in mib-2 (defined in RFC 1213). OID of the first object in this MIB is "1.3.6.1.4.1.1024" (or can be represented as iso(1).org(3).dod(6).internet(1).private(4).enterprises(1). Device Discover(1024)). There are three group objects in this MIB:

1. devDiscoverCommon(1.3.6.1.4.1.1024.1): This contains many common objects e.g. Ethernet interface to be scanned, discovery state etc…

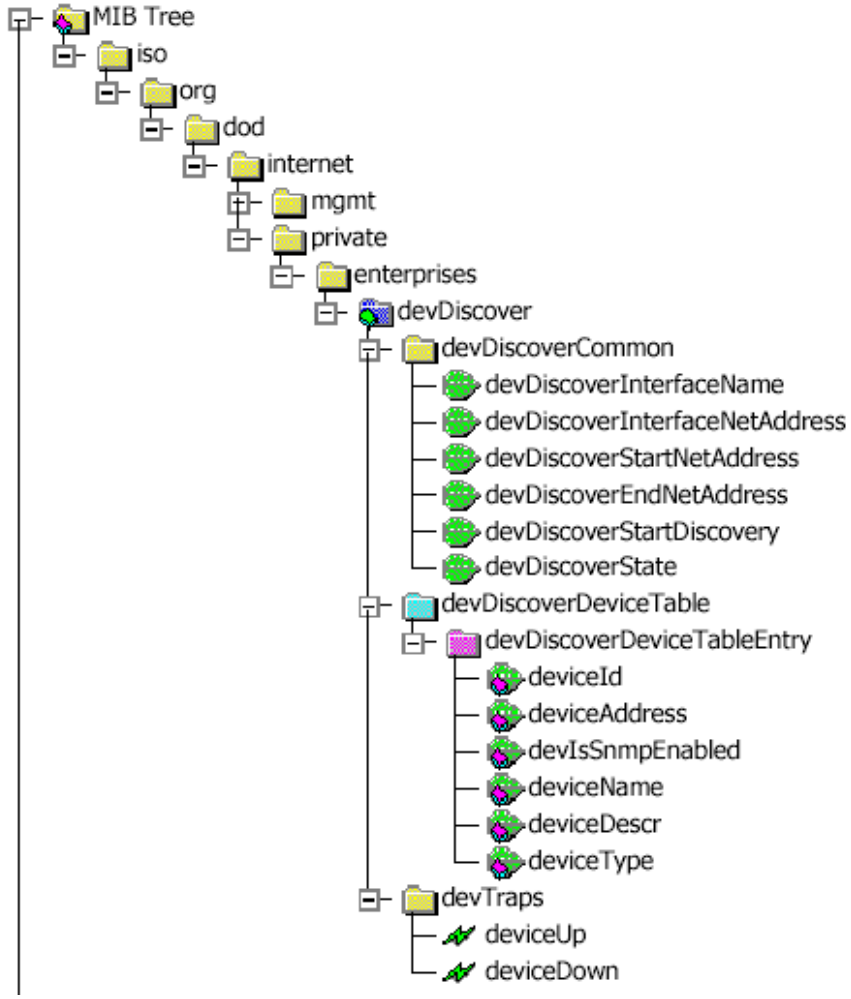


Figure 8: DEV-DISCOVER-MIB

2. devDescoverDeviceTable (1.3.6.1.4.1.1024.2): This object represent a two dimensional object i.e. a table containing information of discovered devices e.g. deviceName, deviceAddress, devIsSnmpEnabled etc…
3. devTraps (1.3.6.1.4.1.1024.3): This object defines two traps that will be sent to SNMP manager when a device comes up or goes down on a network.

Full DEVICE-DISCOVER-MIB is being explained under Appendix A. Description of each an every object fully explains the purpose and use of that object.

4.4 Configuration file

As describe earlier, some of the configuration for various common parameters of Device Discover is being done through a configuration file. Parameters that are configured through a file include:

- SNMP_DEFAULT_TIMEOUT: Its value indicates the time for which we should wait for a SNMP response to come. If SNMP response is not received in this much time, then SNMP request is time-out and depending up the value for SNMP_DEFAULT_RETRIES, SNMP request will be sent again or not.
- SNMP_DEFAULT_RETRIES: Its value indicates the number of retries that will be performed by the SNMP scanner. Usually its value is set to 0, as SNMP Scanner implemented in Device Discover claims to be very fast and if SNMP response is received on first time then its ok, otherwise it is assumed that, the device is not SNMP enabled.
- SNMP_MAX_COMMUNITY_COUNT: This is used by SNMP Scanner. It indicates number of communities with which a SNMP request packet will be sent to a device.
- NMS_PING_TIME_WAIT: Its value indicates the number of seconds to wait for after sending all the ICMP Echo request packets. Ping scanner waits for this much seconds and any ICMP Echo reply message received after waiting this much time is discarded.
- NMS_SNMP_TIME_WAIT: Its value indicates the number of seconds to wait for after sending all the SNMP request packets. SNMP scanner waits for this much seconds before declaring SNMP scans to be completed. Any SNMP response packet received after waiting this much time is discarded

- SNMP_IN_BW_PACKET_DELAY: Its value indicates the number of seconds to wait between each SNMP packet.
- NMS_DEBUG_LEVEL: It provides level of debugging enabled in the system.

There are many more common parameters defined in the configuration file.

4.4.1 Config File Format:

This section describes the configuration file format and this configuration file is being used for doing the basic configuration of Device Discover. A sample configuration file used in Device Discover is being added as Appendix B.

Each line of the configuration file is parsed by *configFileParser*. Each line of the file can be a name-value pair, a blank line or a comment line. Blank and comment lines are ignored by *configFileParser.* Comment line always starts with '#' symbol.

But for each non-blank and non-comment line, an entry in *configHashTable*(hash table to keep all the configurable information) is added. E.g. an entry could be of the form (name-value pair)

……

……

SNMP_DEFAULT_RETRIES 4

[name] [value]

…..

On parsing this line, configFileParser will add an entry into the configHashTable, name will be act as a key to the hash table and value will be value at that key in the hash table. GLIBC is used for maintaining configHashTable and many other data-structures used throughout the Device Discover.

Pseudo code for configFileParser:

```
/*For parsing configFile*/
{
GHashTable* configHashTable =  g_hash_table_new(NULL, strcmp);
```

```
readConfigLine(line);
parseConfigLine(line, key, value);
if parsing successful
{
        /*add the key to hash table*/
        g_hash_table_insert(configHashTable, key, value);
}

/*for reading config item from hash table*/
getConfigItem(Name,value)
{
        /*name is the name of configurable item,
        **value is the value returned
        */
        gpointer ptr;
        ptr = g_hash_table_lookup(configHashTable, name);
        value = (char *)ptr;
}
```

CHAPTER 5 - DESIGN AND IMPLEMENTATION

This section describes design and implementation of all components of Device Discover. The overall architecture of Device Discover is already being covered under chapter4; this would be an extension to that. As discussed Device Discover has following major components:

- SNMP Manager
- SNMP Agent
- Discovery Process

Device Discover uses third party SNMP manager, and open source SNMP agent. Following sub-section will discuss each of the components in sufficient detail.

5.1 SNMP Manager

Device Discover make use of SimpleMibBrowser (Linux version) by SimpleSoft Inc. as a SNMP manager. SimpleMIBBrowser is an easy to use, graphical tool that simplifies the process of interacting with SNMP enabled devices to retrieve and configure management information.

Management Information Base (MIB) ASN.1 documents can be loaded into the SimpleMIBBrowser and their contents displayed in a graphical tree. Queries can then be generated by selecting nodes on the managed object tree and sent to the device/SNMP agent and the responses shown in a tabular fashion.

SimpleMIBBrowser helps in:

- Making SNMP-GetNext (or walk) queries on the agent.
- Making SNMP-Set queries on the agent.
- Making SNMP-Get queries on the agent.
- Making SNMP-GetBulk queries on the agent.
- Displaying the contents of MIB tables in a corresponding tabular result form.
- Displaying Traps/Notifications/Informs received from agents.
- Conducting polling of performance related variables and displaying the result in the form of graph.

- Specifying various communication parameters.
- Comparing the results of the previous query and the latest query on same set of object.

Device Discover is supposed to be run on Linux machine (though most of the design is OS independent), so we need to install SimpleMIBBrowser on Linux machine. By default, installation copies the install files to 'opt/smbrowser/bin' directory. In order to start the browser one need to give following commands at prompt.

```
# cd /opt/smbrowser/bin
#./smbrowser
```

5.1.1 Adding MIB

As discussed in earlier chapters, a new non-standard MIB (DEV-DISCOVER-MIB) is designed and implemented to manage and control the discovery process. In order to control the discovery process through SimpleMIBBrowser, we need to import MIB definition file for DEV-DISCOVER-MIB. This can be done via menu options *'MIBs->Add/Remove MIBs'*

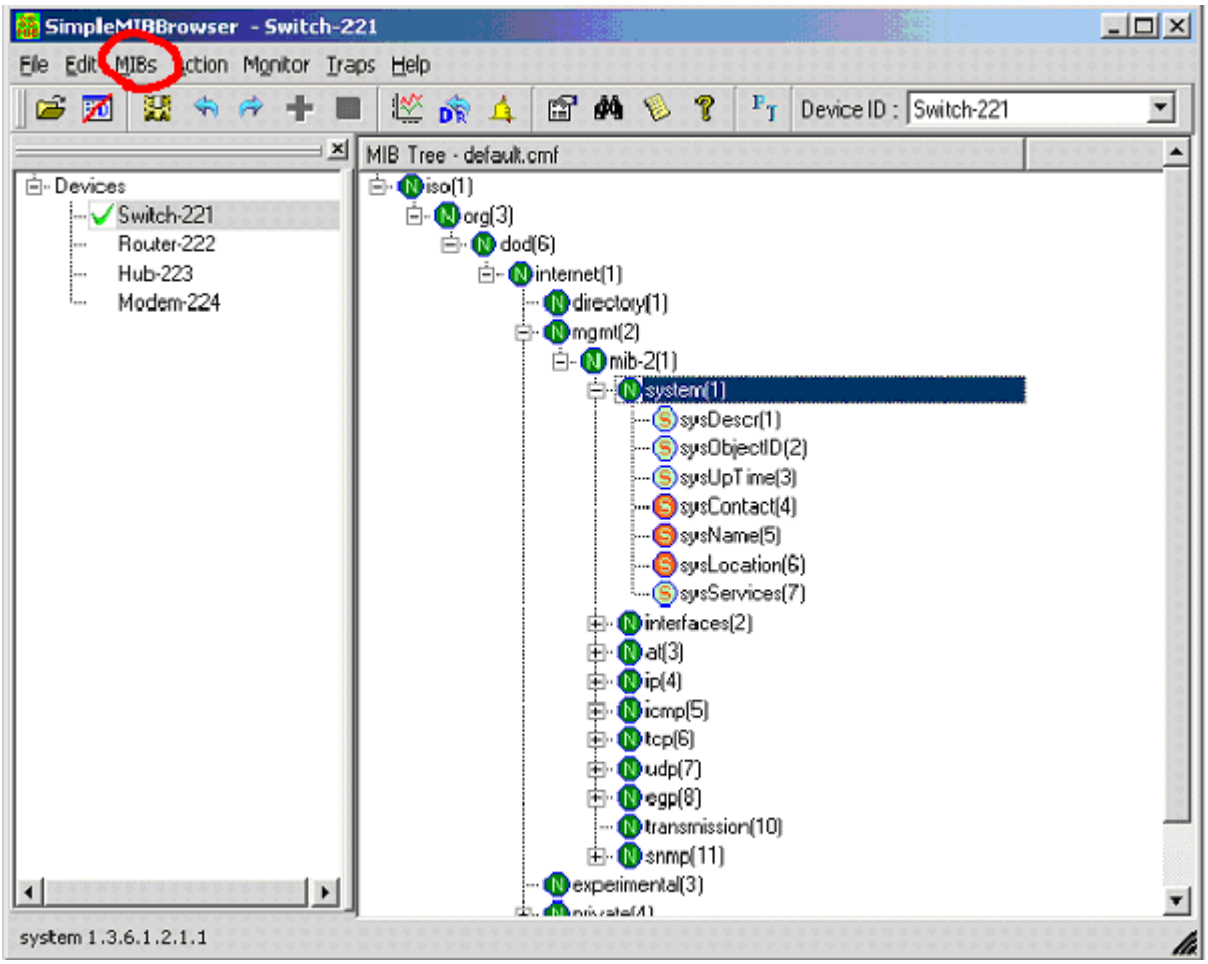


Figure 9: SimpleMIBBrowser

5.2 SNMP Agent

SNMP agent is another important component of Device Discover. Device Discover make use of OpenSource SNMP agent provided via CMU (called CMU SNMP). CMU SNMP agent has supports for various common MIBs like mib-2 (defined in RFC 1213) etc. We need to extend agent via writing the C code for our MIB (i.e. DEVICE-DISCOVER-MIB).

5.2.1 Extending Agent

As mentioned earlier, we need to extend the CMU-SNMP agent with a non-standard MIB for controlling and managing discovery procedure. For this we need to implement all the scalar

objects specified under 'devDiscoverCommon' object, we need to implement 'devDiscoverDeviceTable' tabular object and we need to implement two traps defined under 'devTraps' object. In order to implement new MIB we need to add three files to CMU-SNMP agent:

- A MIB definition file (devDiscover.my);
- A C header file (devDiscover.h);
- A C implementation files (devDiscover.c).

The first file needed is the MIB file that defines the MIB module to be implemented. If the MIB file is the definition of the module for external network management applications then the header file has traditionally served effectively the same purpose for the agent itself. Recent changes to the recommended code structure have resulted in the header file becoming increasingly simpler. It now simply contains definitions of the publically visible routines, and can be generated completely by *mib2c*. Header file will have the following function prototype definitions:

```
extern void  init_devDiscoverCommon (void);
extern FindVarMethod var_devDiscoverCommon;

and write method for each writable field

        WriteMethod write_devDiscoverInterfaceName;
        WriteMethod write_devDiscoverStartNetAddress;
        WriteMethod write_devDiscoverEndNetAddress;
        WriteMethod write_devDiscoverStartDiscovery;

Similar prototypes is required for settable fields of 'devDiscoverdeviceTable'
      …..
```

The core work of implementing the module is done in the C code file. This file contains the implementation for DEVICE-DISCOVER-MIB. Template code for the MIB can be generated using a mib-compiler tool 'mib2c'.

'mib2c' is a tool that comes with the Net-SNMP package (OpenSource package). This tool generates the code stubs for the objects defined in the MIB, and then these stubs are filled to implement our MIB.

Command for generating template code:

```
#env "MIBS=+DEV-DISCOVER-MIB" ./mib2c -c mib2c.old-api.conf devDiscover
```

Here follows the high level design for the various objects of DEV-DISCOVER-MIB.

5.2.1.1 Implementing - 'devDiscoverCommon'

'devDiscoverCommon' is a collection of few scalar objects. Some of these are 'read-only' and some are 'read-write'. For each object there is a corresponding variable being defined in the 'pingThread'. On SNMP-Get request value of that variable is returned. And on set, value is written to that variable. Here follows implementation for the objects under 'devDiscoverCommon' object.

- devDiscoverInterfaceName (OID - 1.3.6.1.4.1.1024.1.1):

This object is of 'read-write' type and specifies the interface to be scanned. Its value is stored in a global variable 'g_devDiscoverIfName'. Get will return the value stored in this variable and set will update the value stored at this variable.

```
#define IF_MAX_LEN 16
unsigned char g_devDiscoverIfName[IF_MAX_LEN];
get_devDiscoverInterfaceName()
{
   return g_devDiscoverIfName;
}
set_devDiscoverInterfaceName(name)
{
```

```
   strcpy(g_devDiscoverIfName, name);
}
```

- devDiscoverInterfaceNetAddress(OID - 1.3.6.1.4.1.1024.1.2)

This object is read-only (i.e. SNMP-Set request is not applicable to this object). SNMP-Get for this object will return IP-Address of the interface named by g_devDiscoverIfName. IP-Address is obtained by doing 'ioctl' call on any random socket.

- devDiscoverStartNetAddress(OID - 1.3.6.1.4.1.1024.1.3):

This is a read-write object, and if specified provide the starting network address from which scanning is to be start till endNetAddress. A global variable 'g_devDiscoverStartNetAddress' is used to store its value.

unsigned long g_devDiscoverStartNetAddress;

- devDiscoverEndNetAddress(OID - 1.3.6.1.4.1.1024.1.4):

This is a read-write object, and if specified provide the end network address for scanning procedure. A global variable 'g_devDiscoverEndNetAddress' is used to store its value.

unsigned long g_devDiscoverEndNetAddress;

- devDiscoverStartDiscovery(OID - 1.3.6.1.4.1.1024.1.5):

This is a 'read-write' object. If set with value true it will start the discovery procedure, and if set with value false it will stop the discovery procedure (if already running).

- devDiscoverState(OID - 1.3.6.1.4.1.1024.1.6)

This is a read-only object, used to specify the discovery state. A global variable 'g_devDiscoverState' is used to keep the state of discovery procedure. It can have one of the following values.

```
typedef enum
{
        IDLE   = 0;                     /*Discovery procedure not started yet*/
```

```
        RUNNING_ICMP_SCAN, /*ICMP Scanner is running*/
        RUNNING_ICMP_WAIT, /*Waiting for ICMP responses*/
        RUNNING_SNMP_SCAN, /*SNMP Scanner is running*/
        RUNNING_SNMP_WAIT, /*Waiting for SNMP responses*/
        COMPLETED,                /*Discovery process completed*/
        ABORTED                   /*Discovery process is aborted externally*/
}devDiscoverState
```

5.2.1.2 Implementing - 'devDiscoverDeviceTable'

'devDiscoverDeviceTable' is a SNMP tabular object. All objects in this table are read-only. This table do not contains 'rowStatus' or any such kind of objects i.e. user can't create any entry in this table. Entries in this table are created automatically one for each discovered device. Basically this table displays the entries present in the 'NMSDeviceDataTable' (describe later) – list of devices discovered. Each node in this list represents one row in 'devDicoverDeviceTable'

5.2.1.3 Implementing - 'devTraps'

Please see 'Trap Implementation' sub-section of next section for design detail of 'devTraps'

5.3 Discovery Process

As already explained, Discovery thread is the core of 'Device Discover' and active discovery method 'ICMP Scan' is extended with 'SNMP Scanner' to accomplish the task of discovery process.

This section will explain design and implementation for various components of discovery process. Design and implementation of 'ICMP Scanner' and 'SNMP Scanner' will be discussed. Various data-structures used throughout, and in last the design of sending 'deviceUp' and 'deviceDown' traps will be discussed.

5.3.1. Threaded Design

As mentioned in previous chapter, discovery process is implemented as a separate thread running parallel to SNMP agent daemon (‘snmpd’). For this a separate thread is created from the ‘main’ function of SNMP agent.

In SNMP agent after the initializations for ‘snmpd’ completes, a separate thread for discovery process is created and a communication infrastructure is setup between ‘snmpd’ and ‘discoveryThread’. A FIFO (standard IPC mechanism in POSIX compliant OS) is created for communication between ‘snmpd’ and ‘discoveryThread’ and message based communication is provided between these two threads. For example, if ‘snmpd’ received a request to start/stop the discovery procedure then ‘snmpd’ will allocate a message of particular type and send it to the ‘discoveryThread’ over the FIFO created for message exchange.

Since threaded approach has been chosen to implement discovery process, so passing a pointer to message structure is sufficient (as ‘doscoveryThread’ is having the same address space as that of ‘snmpd’ and so a particular pointer in both the thread will point to same object) and this will save amount of data that has to be transferred and will improve the overall efficiency of the discovery process.

Pseudo-code for communication setup:

```
snmpd’s main( )
{
        Start snmp agent as a daemon ‘snmpd’.
        Do the initializations.

        /*Communication setup*/
        Create a FIFO for communication between ‘snmpd’ and ‘discoveryTherad’.
        Create ‘discoveryThread’.
}
```

Here follows the message structure (that flow between ‘snmpd’ and ‘discoveryThread’) and its purpose is described. When ‘snmpd’ received a set request for an object of DEV-DISCOVER-MIB, it then creates a message containing that request and passes that message to ‘discoveryThread’. ‘discoveyThread’ continuously waits for messages coming on FIFO and on receiving a message it decide what has to be done. E.g. if a discovery process is running and it receive a message to modify ‘devDiscoverStartNetAddress’ then that message is simply freed without doing any operation.

Thus in this way the set operations is synchronized with respect to ‘discoveryThread’.

```
/*Message structure*/
typedef struct
{
        eMsgType      messageType;          /*Specify the object being set*/
        union
        {
                BOOL enable;                /*Start/Stop discovery*/
                char ifName[IFNAME_MAX_LAN], /*Interface name to be scanned*/
                unsigned long startAddr,    /*Start address for scanning*/
                unsigned long endAddr,      /*End address for scanning*/
        }obj;
}msg;
```

‘messageType’ field identifies the object for which this message is created. Figure 10 shows the simple scenario of starting the discovery process.

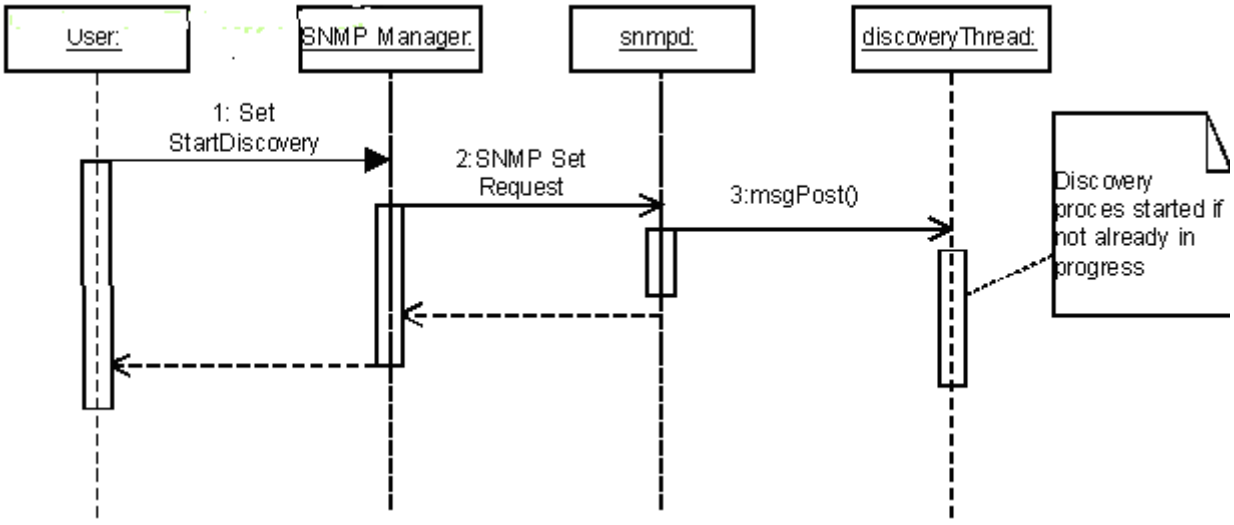


Figure 10: Sequence diag. for simple scenario

5.3.2. ICMP Scanner

ICMP Scanner is one of the main components of 'discoveryThread'. It scans the whole network or IP address range to find out the devices/machines that are active in the network. In order to start scanning process, either a network address or start and end IP-address are provided to ICMP scanner. ICMP Scanner then creates ICMP echo message and send it to all the ip-addresses one by one. And if ICMP reply is received then ICMP scanner comes to know that a particular device is active on the network. Thus when this activity completed, ICMP scanner is having the list of devices that are active on the network. Figure 11 shows the ICMP scanning process.

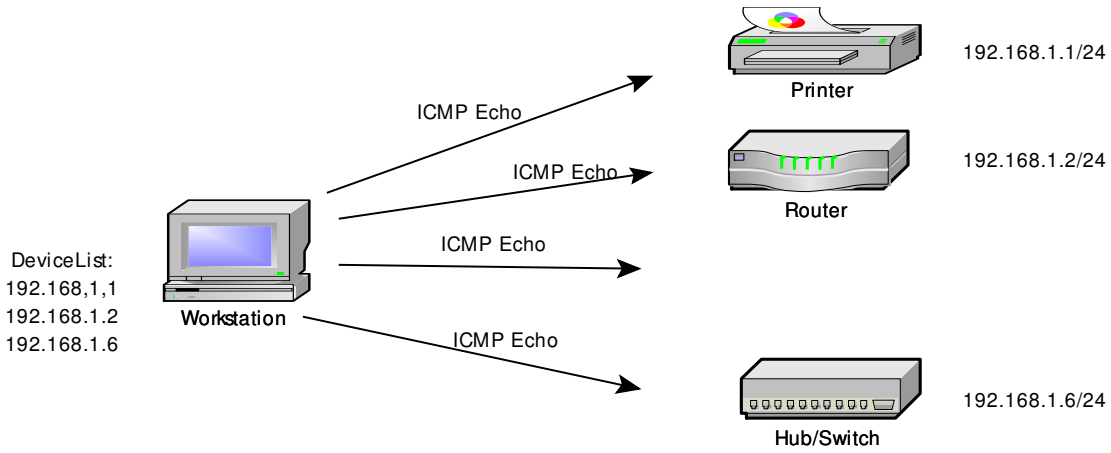


Figure 11: ICMP Scanning

5.3.2.1 Design

Design of ICMP scanner is kept simple, discoveryThread's entry point will create a raw socket 'pingSocket' for sending and receiving the ICMP packets. And for each ip-address to be scanned an ICMP echo packet is created and then send over this socket. In order to distinguish that a particular ICMP echo reply is for a ping request send by 'discoveryThread' 'icmp_id' of the ICMP echo packet is set to a unique value derived from the PID of the 'discoveryThread'.

```
ident = getpid () & 0xFFFF;
icp->icmp_id    = ident;
```

After all ICMP echo packets have been transmitted, ICMP scanner waits for 'NMS_PING_TIME_WAIT' time to receives the ICMP echo replies from devices on network. For each reply we check the 'icmp_id' (along with other verification) of the reply with 'ident' to make sure that reply is for the ICMP request that we have sent.

For each verified reply we create an entry in the list of discovered devices. And after a specific amount of time this list of discovered devices are passed on to SNMP scanner. Any ICMP reply received after waiting for 'NMS_PING_TIME_WAIT' is discarded.

This discovery scheme has one limitation that if ICMP has been disabled on a particular device or machine, that device can not be discovered.

5.3.3. SNMP Scanner

SNMP scanner is another important component of ‘discoveryThread’. It does the SNMP scan over the list of devices/machines discovered by the ICMP scanner in the last step.

The SNMP protocol is a stateless, datagram oriented protocol. SNMP scanner sends SNMP requests to multiple IP addresses (discovered during ICMP scan), trying different community strings and waiting for a reply. Unfortunately SNMP agent do not respond to requests with invalid community strings and the underlying UDP protocol does not reliably report closed UDP ports. This means that 'no response' from the probed IP address can mean either of the following:

- machine unreachable
- SNMP agent not running
- invalid community string
- the response datagram has not yet arrived

The approach taken by most SNMP scanners is to send the request, wait for n seconds and assume that the community string is invalid. If only 1 of every hundred scanned IP addresses responds to the SNMP request, the scanner will spend 99*n seconds waiting for replies that will never come.

This makes traditional SNMP scanners very inefficient.

But SNMP scanner implemented for ‘Device Discover’ takes a different approach to SNMP scanning. It takes advantage of the fact that SNMP is a connectionless protocol and sends all SNMP requests as fast as it can. Then the scanner waits for responses to come back and store them in the ‘NMSDeviceDataTable’. By default it waits for 10 milliseconds between sending packets, which is adequate for 100Mbs switched networks. The user can adjust this value via the configuration file parameter defining in between packet delay (SNMP_IN_BW_PACKET_DELAY). If set to 0, the scanner will send packets as fast as the kernel would accept them, which may lead to packet drop.

SNMP Scanner sends a request for the following OIDs:

1. system.sysName.0 (1.3.6.1.2.1.1.5.0)
2. system.sysDescr.0 (1.3.6.1.2.1.1.1.0)

These OIDs belongs to System group of MIB-II and are present on almost all SNMP enabled devices. Return values gives us more details of the system software running on the device.

5.3.3.1 Design

As explained, SNMP scanner will send the SNMP request for 'sysName' and 'sysDescr' to each ip-address discovered during ICMP scanning.

As SNMP packet include the community string and a particular SNMP agent will reply to SNMP get request only if the community string in the SNMP packet matches with that of configured in the SNMP agent. For this each ip-address is tried with the list of communities (most commonly used in the network) i.e. for each ip-address we send a SNMP get request with all the communities specified in the list.

```
#define MAX_COMMUNITIES       1024
int community_count = 3;      /*Number of communities present in the list*/
char* community[MAX_COMMUNITIES] = {
                                                {"public" },
                                                {"private" },
                                                {"ADSL"}
                                        };
```

E.g. Above configuration suggest that for each ip-address discovered during ICMP scan, SNMP scanner will send three SNMP get request one with each community name.

As the objects for which SNMP request has to be send are fixed, So SNMP packets is build locally (without making use of any library, that makes it even more efficient) in ASN.1 and TLV format. A local character array 'object' makes the OIDs in the TLV format.

```
char object[] =" \x30\x1c       /*Tag - length*/
\x30\x0c                        /*Tag – length*/
\x06\x08\x2b\x06\x01\x02\x01\x01\x01\x00\x04\x00 /*OID for sysName*/
\x30\x0c                        /*Tag - length*/
```

```
\x06\x08\x2b\x06\x01\x02\x01\x01\x05\x00\x04\x00 /*OID for sysDescr*/
";
```

A SNMP packet is created by prefixing the TLV format for 'SNMP version', 'community', 'PDU type: Get', 'Request ID', 'Error Status', and 'Error Index' (for detail see …SNMP message format - Figure 5)

After sending all the SNMP request packets in one go, SNMP scanner waits for some time to receive SNMP responses. After waiting for configurable (configurable through configFile) amount of time discovery process comes to an end and any packet received after that is discarded.

5.3.3.2 Device Data-structure

Discovery process collect the information about the devices that are active on the network and this collected information is displayed to user through DEV-DISCOVER-MIB. Structure of following type is kept to maintained data related to discovered devices

```
#define MAX_DEVICE_NAME_LEN 128
#define MAX_DEVICE_DESCR       255

typedef struct NMSDeviceData
{
        UINT8 deviceName[MAX_DEVICE_NAME_LEN];
        UINT8 deviceDescr[MAX_DEVICE_DESCR];
        UINT8 isSNMPEnabled;
        unsigned long deviceNetAddress;
        UINT8 devType;
}NMSDeviceData;

/*Link List to maintain the device data tales*/
GSList* currentNMSDeviceData = NULL;
GSList* prevNMSDeviceData = NULL;
```

Two linked-lists are created to keep device data (‘currentNMSDeviceData’ and ‘prevNMSDeviceData’). Current list keep the device data for currently running discovery process. For each node that get discovered, an entry is added into ‘currentNMSDeviceData’ list

Where as, ‘prevNMSDeviceData’ list contains device data collected during last instance of discovery process. Individual elements in each list are of type ‘NMSDeviceData’.

Purpose of keeping the collected information from last run is to identify the devices that comes UP or goes down between two runs of discovery process, and this allows raising the ‘deviceUp’ and ‘deviceDown’ traps at the end of discovery process.

CHAPTER 6 - COMPILATION AND USAGE

This chapter briefly explains how to compile and run 'Device Discover' and other component like SNMP manager and SNMP agent.

6.1. Installing and running SNMP Manager

As specified earlier, SimpleMibBrowser (Linux version) by SimpleSoft Inc is used as SNMP manager.

To install the standalone version of SimpleMIBBrowser:

1. Start Linux.
2. Insert the installation disk in your CD drive.
3. Invoke install_smbrowser_linux_nn.sh.
4. Follow the instructions on the screen.

The install program will begin copying files from the CDROM to the directory you specify. By default, it copies the install files to the '/opt/smbrowser' directory. After installing the files, add /opt/smbrowser/bin to the path environment variable and create an environment variable called, "SMB_DIR" and set it to the location of the SimpleMIBBrowser's installation.

In order to start the browser one need to give following commands at prompt.

```
# cd /opt/smbrowser/bin
#./smbrowser
```

6.2. Installing and running SNMP Agent/Discovery Thread

As specified, CMU-SNMP is used as SNMP agent (after extending it for DEV-DISCOVER-MIB).

The source distributions for CMU SNMP is named as

* cmu-snmp-linux-3.7-src.tar.gz

And same is available from ‘ftp.ibr.cs.tu-bs.de (134.169.34.15)’ in ‘/pub/local/linux-cmu-snmp’.

In order to install SNMP agent on to Linux machine please follow following steps.

Source Quick Install:

1. Extract the source by running:

   tar xvzf .../cmu-snmp-linux-3.7-src.tar.gz
   cd cmu-snmp-linux-3.7
   ./configure
   make

2. Switch to root and run
   make install

3. If install v3.7 the first time, do
   cd ./etc
   ./installconf -mini <password>

4. Have a look at /etc/snmpd.conf to be familiar with it and create an
   entry in /etc/rc.local:
   /usr/sbin/snmpd -f ; echo 'snmpd'
   This launches the agent (running in background) and it also start the ‘discoverThread’ (from main() function of ‘snmpd’).

Discovery process code is placed inside the CMU code and gets compiled along with CMU SNMP code.

6.3. Using the installed component

This section explains how to use the installed components.

1. Start SNMP Manager.
2. Start SNMP agent.
3. Set ‘devDisoverStartDiscovery’ to true for starting the discovery process.
4. Keep checking the discovery status by getting the value of ‘devDiscoverState’
5. When get on ‘devDiscoverState’ return completed, this marks the end of discovery process.
6. Now do a Get/GetBulk on ‘devDiscoverDeviceTable’ to find information about devices/machines discovered by discovery process.

6.4. Debugging Support

Discovery process code is compiled using ‘-g’ flags, thus the debugging support is enabled in the executable produced for ‘discoveryThread’. And one can make use of GDB to debug the discovery thread code.

Also various levels of debug traces have been added in the discovery process code. These levels are:

DEBUG_NONE (0) /*Debug traces are off*/

DEBUG_INFO (1)

DEBUG_RESULT (2)

DEBUG_ERROR (3)

Any one level of debug traces can be enabled by specifying the value of ‘NMS_DEBUG_LEVEL’ in configFile to one of these values.

CHAPTER 7 - CONCLUSION AND FUTURE WORK

This dissertation work is toward the Network discovery feature of the Network Management Systems. Different network discovery approaches (for example, passive discovery techniques, and active profiling techniques) are studied and analyzed; their advantages and shortcomings are analyzed and studied.

Active discovery techniques, **ICMP Echo Scan**, under Host link layer Techniques is used for the network discovery in the targeted system. ICMP echo scan (or ping scan) is used to discover the active devices on network. In order to find extra information about the devices discovered ICMP scan is followed by SNMP Scan. SNMP Scan (as currently implemented) used to obtain 'system.sysName' and 'system.sysDescr' objects of 'system' group. This provides information about the device and system software running on the device.

Whole discovery process is controlled and monitored via SNMP manager. And a non-standard MIB (DEV-DISCOVER-MIB) is designed and implemented to allow SNMP entities to control the process. This MIB include objects, and set/get on these objects allows managing the discovery process.

SimpleMIBBrowser from SimpleSoft is used as SNMP Manager. And open-source CMU-SNMP agent (extended with DEV-DISCOBER-MIB implementation) is used as SNMP agent. ICMP scanner and SNMP scanner (and other components of discovery process) is implemented as a separate thread 'discoveryThread'. All research and study related to the field of Network management and Network discovery is completed. SNMP daemon application has been ported as per our requirements.

This dissertation work can be extended to include the following:

- Design of a private MIB to manage and control the data collected through the discovery process.
- Adding the support for that MIB in the SNMP daemon application.
- Integration of the discovery thread with SNMP daemon.
- Adding support for the notification (trap) generation when any network device goes down or comes up on a local LAN.

CHAPTER 8 - PUBLICATION FROM THESIS

8.1. Communicated Paper

1. Daya Gupta, Garima Gupta, "DevDiscover: A component for Network Management system Using SNMP", in the International Journal of "Computer and Systems Engineering" (IJCSE), Serials Publications, New Delhi, India, June, 2008.

2. Daya Gupta, Garima Gupta, "Network Discovery: Integrating ICMP Echo Scan and SNMP Scanner", in the International Conference on "Pervasive computing and Governance 2008" (ICPCG08), Bhopal, November 23-25, 2008.

3. Daya Gupta, Garima Gupta, "An Approach to Network Discovery for Network Management Systems", in the National Conference on "Next generation Technologies" (NextGen T-2008), Ghaziabad, October 17-18, 2008.

CHAPTER 9 - REFERENCES

[1]. Marshall T. Rose, The Simple Book: An Introduction to Management of TCP/IP-based internets, Prentice-Hall, Englewood Cliffs, 1991.

[2]. Stalling W., SNMP,SNMPv2,SNMPv3 and RMON1 and 2, Pearson-Education,2004.

[3]. K.McCloghrie, M.Rose, "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", RFC1213, March 1991

[4]. W. Richard Stevens, Unix Network Programming: Networking APIs, Volume 1 (2nd Ed), Prentice Hall, 1998.

[5]. Annie De Montigny-Leboeuf and Frédéric Massicotte, Passive Network Discovery for Real Time Situation Awareness. Communication Research Centre Canada, Ottawa, Ontario, 2004.

[6] Ofir Arkin, ICMP Usage in Scanning: The Complete Know-How, Version3.0. Sys-Security Group, June 2001.

[7]. J. Treurniet, An Overview of Passive Information Gathering Techniques for Network Security, TM 2004-073. Defense R&D Canada – Ottawa. Ottawa, Ontario, May 2004.

[8]. CMU SNMP Library,
www.net.cmu.edu/groups/netdev/software.html

[9]. C Brain, “Algorithms and Techniques Used for Auto-discovery of Network Topology, Assets and Services”, Faculty of Computer Science, University of New Brunswick, Canada, March 16, 2006.

[10]. Annie De Montigny-Leboeuf and Frédéric Massicotte. Passive Network Discovery for Real Time Situation Awareness. Communication Research Centre Canada, Ottawa, Ontario, 2004.

[11]. Ofir Arkin. ICMP Usage in Scanning: The Complete Know-How, Version3.0.Sys-Security Group, June 2001, p 75.
http://www.syssecurity.com/index.php?page=icmp

[12]. Protocol Numbers. Internet Assigned Numbers Authority, September 2005.
http://www.iana.org/assignments/protocol-numbers

[13]. J. Treurniet. An Overview of Passive Information Gathering Techniques for Network Security, TM 2004-073. Defense R&D Canada – Ottawa. Ottawa, Ontario, May 2004.
http://www.ottawa.drdc-rddc.gc.ca/docs/e/TM2004-073.pdf

[14]. Internet Scanner User Guide Version 7.0, Service Pack 2. Internet Security Systems, February 28, 2005, p 182.
http://documents.iss.net/literature/InternetScanner/IS_UG_7.0_SP2.pdf

APPENDIX A: DEV-DISCOVER-MIB

This appendix contains full text definition of DEV-DISCOVER-MIB designed and implemented for Device Discover. It explains each and every node of the MIB, their purpose, their syntax etc …

A.1 Full Text-Definition:

-- file: DEV-DISCOVER-MIB.my
-- This MIB is propritary, and is being developed for
-- facilitating the Automatic device Discovery.
-- SNMP Manager (NMS) will use this MIB implementation for controling
-- Device Discovery process
-- Changes:
-- No changes needed.
--

DEV-DISCOVER-MIB *DEFINITIONS* ::= BEGIN

IMPORTS
MODULE-IDENTITY, OBJECT-TYPE, mib-2 ,enterprises, Counter32,
Gauge32,Unsigned32, IPAddress, TimeTicks FROM SNMPv2-SMI
RowStatus, TruthValue, DisplayString FROM SNMPv2-TC
MODULE-COMPLIANCE, OBJECT-GROUP FROM SNMPv2-CONF
InterfaceIndexOrZero, InterfaceIndex FROM IF-MIB
mgmt FROM RFC1213-MIB;

Device Discover *MODULE-IDENTITY*

CONTACT-INFO
" Garima Chadha

MAIT

Rohini, New Delhi

Phone: +91 9873293294

Email:garima.chadha@gmail.com

"

DESCRIPTION

" MIB Module for Device Discovery Management.

Used to control and manage the discovery process

"

::= { enterprises 1024}

```
--
-- Common Objects that controls the discovery process
--
devDiscoverCommon     OBJECT IDENTIFIER ::= { devDiscover 1 }

-- ethernet interface to be scanned
devDiscoverInterfaceName OBJECT-TYPE
   SYNTAX      DisplayString (SIZE(0..32))
   MAX-ACCESS  read-create
   STATUS      current
   DESCRIPTION
       "The devDiscoverInterfaceName specify the name of the ethernet interface, that has to be
scanned for linux machine this name could be like eth0, eth1 etc. It is a writable field; user
always has to set its value to a valid name before startinf the discovery procedure.
       "
   DEFVAL      { "H }   -- the empty string
   ::= { devDiscoverCommon 1}

-- IP-Address of the Ethernet interface
```

devDiscoverInterfaceNetAddress *OBJECT-TYPE*

SYNTAX IpAddress

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The devDiscoverInterfaceNetAddress specify the IPAddress of the interface specified in devDiscoverInterfaceName. This is read-only field. this field get value automatically on specifying the value in devDiscoverInterfaceName, i.e. it contains the network address assigned to interface name.

"

::= { devDiscoverCommon 2}

-- Ip address from where to start the scanning

devDiscoverStartNetAddress *OBJECT-TYPE*

SYNTAX IPAddress

MAX-ACCESS read-write

STATUS current

DESCRIPTION

"The devDiscoverStartNetAddress if specified, signifies the start address, i.e. Network discovery suit will try to discover all devices with the network address between devDiscoverStartAddress and devDiscoverEndAddress. devDiscoverStartAddress should always less then devDiscoverEndAddress. If any one of devDiscoverStartAddress or devDiscoverEndAddress is not specified, then the entire network range of devDiscoverInterfaceName will be scanned

"

::= { devDiscoverCommon 3}

-- IP address from where to start the scanning

devDiscoverEndNetAddress *OBJECT-TYPE*

SYNTAX IPAddress

MAX-ACCESS read-write

STATUS current

DESCRIPTION

"The devDiscoverEndNetAddress if specified, signifies the end address, i.e. Network discovery suit will try to discover all devices with the network address between devDiscoverStartAddress and devDiscoverEndAddress devDiscoverStartAddress should always less then devDiscoverEndAddress. If any one of devDiscoverStartAddress or devDiscoverEndAddress is not specified, then the entire network range of devDiscoverInterfaceName will be scanned

"

::= { devDiscoverCommon 4}

-- Start the discovery process

devDiscoverStartDiscovery *OBJECT-TYPE*

SYNTAX INTEGER {

enable(1), -- an invalidated mapping

disable(2)

}

MAX-ACCESS read-write

STATUS current

DESCRIPTION

"The devDiscoverStartDiscovery, set this object to value 1(enable) to start the discovery process, Setting it to 2(disable) will stop the discovery process if already in progress.

"

::= { devDiscoverCommon 5}

devDiscoverState *OBJECT-TYPE*

SYNTAX INTEGER {

idle(1), -- Discovery not yet started

progress(2), -- Discovery in progress

completed(3), -- Discovery completed

error(4) -- Discovery stopped because of error

}

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The devDiscoverState, this object signifies the state of discovery process. Discovery process can be in any one of the following state

idle(1), -- Discovery not yet started

progress(2), -- Discovery in progress

completed(3), -- Discovery completed

error(4) -- Discovery stopped because of error

"

::= { devDiscoverCommon 6}

devDiscoverDeviceTable *OBJECT-TYPE*

SYNTAX *SEQUENCE OF* DevDiscoverDeviceTableEntry

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"A table of Network address of the device that has been discovered during the discovery procedure

"

::= { devDiscover 2 }

devDiscoverDeviceTableEntry *OBJECT-TYPE*

SYNTAX DevDiscoverDeviceTableEntry

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"IP address of the devices detected so far. Entries in the devDiscoverDeviceTable are created and deleted automatically as discovery process proceeds.

"

INDEX { *IMPLIED* deviceAddress}

::= { devDiscoverDeviceTable 1 }

DevDiscoverDeviceTableEntry ::= *SEQUENCE* {

deviceId Integer32,

deviceAddress IpAddress,

devIsSnmpEnabled INTEGER,

deviceName DisplayString(SIZE(0..255)),

deviceDescr DisplayString(SIZE(0..255)),

deviceType INTEGER

}

deviceId *OBJECT-TYPE*

SYNTAX Integer32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Device ID, its a running number assigned to each device as they get discovered in the discovery procedure these number are not stored any where.

"

::= { devDiscoverDeviceTableEntry 1 }

deviceAddress *OBJECT-TYPE*

SYNTAX IPAddress

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Device Address,

Ip address of detected device

"

::= { devDiscoverDeviceTableEntry 2 }

devIsSnmpEnabled *OBJECT-TYPE*

SYNTAX *INTEGER*{

true(1), -- Device is SNMP Enabled

false(2) -- Device is not SNMP Enabled

}

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"This object specifies whether the discovered device is snmp enabled TRUE(1), or not FALSE(2). In the discovery process, once a device get discovered(through ping scan), it will get queried with SNMP GET request, If response for SNMP get arrived in a designated time, then that device is declared to be snmp enabled otherwise said to be not supporting SNMP protocol

"

::= { devDiscoverDeviceTableEntry 3 }

deviceName *OBJECT-TYPE*

SYNTAX DisplayString(SIZE(0..255))

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"This is read-only object, this object will have value only if the device is snmp enabled, it signifies the device name. And its value will corresponds to sysName of device system object.

"

::= { devDiscoverDeviceTableEntry 4 }

deviceDescr *OBJECT-TYPE*

SYNTAX DisplayString(SIZE(0..255))

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"This is read-only object, this object will have value only if the device is snmp enabled, it signifies the device description. And its value will corresponds to sysDescr of device system object.

"

::= { devDiscoverDeviceTableEntry 5 }

deviceType *OBJECT-TYPE*

SYNTAX INTEGER{

bridge(1), -- Device is a bridge

router(2), -- Device is a router

printer(3), -- Device is a printer

other(4) -- Device is of some other type

}

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"This object is read-only; it is applicable to a device if and only if, that device is snmp enabled. If a device is supporting the printer MIN then that device is declared to be printer. And if a device is supporting BRIDGE MIB then that device is declared to be of BRIDGE type...

"

::= { devDiscoverDeviceTableEntry 6 }

--

-- Device Traps

-- DeviceUp and DeviceDown traps are added for notifying, when a device comes up and goes down

devTraps *OBJECT IDENTIFIER* ::= { devDiscover 3 }

deviceUp *NOTIFICATION-TYPE*

OBJECTS {deviceAddress, deviceName}

STATUS current

DESCRIPTION

"A deviceUp trap signifies that a device is come up on the network."

::= { devTraps 1 }

deviceDown *NOTIFICATION-TYPE*

OBJECTS {deviceAddress, deviceName}

STATUS current

DESCRIPTION

"A deviceDown trap signifies that a device goes down on the network."

::= { devTraps 2 }

END

APPENDIX B: SAMPLE CONFIGURATION FILE

This appendix show a sample configuration file being used in Device Discover. Entries can be added into the file at compile time and configFileParser will make use of that entries.

#
 # Configuration File:- DevDiscover
 # This file shows the basic configuration that are required for Device Discover
 # Each configuration is specified as name-value pair
#

#SNMP_DEFAULT_TIMEOUT-
#Its value indicates the time for which we should wait for a SNMP response #to come. If SNMP response id not received in this much time, then SNMP #request is time-out and depending up the value for #SNMP_DEFAULT_RETRIES, SNMP request will be sent again or not. Time is #specified in sec.
SNMP_DEFAULT_TIMEOUT 5

#SNMP_DEFAULT_RETRIES -
#Its value indicates the number of retries that will be performed by the #SNMP scanner. Usually its value us set to 0, as SNMP Scanner #implemented in Device Discover claims to be very fast and if SNMP response is #received on first time then its ok, otherwise it is assumed that, the device #is not SNMP enabled.
SNMP_DEFAULT_RETRIES 0

#
 # More entries could be added or existing could be modified,
 # But changes will be effected from next program restart
#
#end