Graduate School in Computer Science
(2000-2001)
Pre-doctoral Project

Network Aware Failure Detection

by
Sirajuddin Shaik Mohammed
Graduate School in Computer Science 2000/2001
École Polytechnique Fédérale Lausanne
(sirajuddin.shaikmohammed@studi.epfl.ch)

Under the Guidance of
Prof. André SCHIPER
&
Matthias Wiesmann
Laboratoire de Systèmes d'Exploitation
École Polytechnique Fédérale Lausanne
Ecublens
CH-1015 Lausanne

9th July 2001

Acknowledgements

I would like to thank LSE project supervisors Prof.A.Schiper and M.Wiesmann for giving me an opportunity to work on this project that enabled me to learn the key concepts about SNMP. I forward a special thank to M.Wiesmann for his support, encouragement and advice to structure my report.

.

Contents

List of Figures

Chapter 1

Introduction

Reliability is one of the key design goals of distributed systems which can be defined as the degree of tolerance against errors and faults. One important aspect in distributed computing is the possibility that nodes can fail.The occurence of a failure pose a reliability problem. Detecting such failures is both useful for network administration and implementation of distributed algorithms. The failure detection abstraction is a very powerful one and is used a lot in distributed programming.

Traditionally, the failure detection is done mostly with ping like protocols in which a machine sends a message to another and waits for the answer to this message. This approach has two drawbacks. Firstly, it is costly, as a part of network bandwidth is used by such messages, posing both performance and scalability problems. Secondly, such systems can be unreliable, as they depend on time-outs.

The goal of this project is to explore the concept of the hardware failure(node failure) and software failure (application failure) by extracting information from the network architecture instead of solely relying on ping protocol and exploit this information to build a scalable and fast failure detection which could be used be used both for network administration and as building block for distributed systems. Simple Network Management Protocol (SNMP) is used to accomplish our goal as it a standard for network management and widely accepted.

The workable approach to fault detection in this report employs external detection [5] which implies that responsibilities for detection of node failures are given to facilities external to the node. For example, a node B is tested by another node A. If node A is healthy, it is assumed node A is capable

of detecting any deviation of node B from expected behaviour. If node A is faulty, the diagnosis result can be random.

The failure detection abstraction is explained with a network scenario (Section 2.3) in which we firstly explain the failure detection of application instances running on the server, when the server is up and then explain the failure detection of the server, when the server is down.

We make use of a probe (defined in Section 2.1.10), to help in the failure detection of the server and show that the probe satisfies the properties of $\diamond$S failure detector introduced by Chandra and Toueg [19] and hence can be used to solve the failure detection. The concept of unreliable failure detector is used to detect the failure of probe and we specifically implement $\diamond$S failure detector. This concept of failure detection considering a network scenario can be extended to detect the failure of all nodes in the network.

Finally we conclude the report by showing that our approach of using SNMP is better than traditional approach in terms of network bandwidth, performance and scalability, heterogeneity and interoperability.

The remainder of this report is organized as follows:

Chapter 2 gives the definitions of basic terms that are used in the entire report, the concept of Simple Network Management Protocol and Failure detectors and the network scenario considered in this report.

Chapter 3 gives the detection of failure of application instances running on the server.

Chapter 4 deals with the failure detection of the server. It presents the different approaches to detect the server failure and explains the logic used by the probe, a key component in the approaches used for the server failure detection.

Chapter 5 deals with configuring the probe to make the probe perform the logic that it uses in the approaches for failure detection of server.

Chapter 6 deals with the failure detection of the probe.

Chapter 7 gives the conclusion.

Chapter 2

Concepts, Definitions and Scenario

This chapter gives the definition of technical terms that are used in this report, explains the concept of Simple Network Management Protocol (SNMP) and also gives the network scenario that is considered in this report. The meaning of terms might differ from other definitions found in the literature; these definitions will serve as a reference for this report.

Section 2.1 gives definitions of technical terms that are used in the entire report. Section 2.2 explains the concept of Simple Network Management Protocol (SNMP). Section 2.3 gives the network scenario that is considered in this report.

2.1 Definitions

2.1.1 Distributed system

A Distributed System is a collection of computers connected by a communication subnet and logically integrated in varying degrees by a distributed operating system and/or distributed algorithms. The communications subnet may be a widely geographically dispersed collection of communication processors or a local area network. Typical applications that use distributed computing include multi-media telecommunications, e-mail, web, teleconferencing and support for general purpose computing in academic and industrial settings.

2.1.2 Fault

A fault [5] is a physical defect, imperfection, or flaw that occurs within some hardware or software unit. An error is the manifestation of a fault. It is deviation from accuracy or correctness. If the error results in the system performing one of its functions incorrectly, a system failure has occurred. This report uses fault and failure interchangeably. The failures considered in this report are node (hardware) failures and Application (software) failures.

2.1.3 Network

A *network* is a collection of nodes in which the nodes are connected together through a communication medium. The word *network* in this report will always refer to TCP/IP Ethernet local area network in which all nodes are connected by Ethernet interfaces to communicate with other nodes.

2.1.4 Nodes

The computer network is comprised of *nodes*, which can be thought of as points of connection together with a *transport medium* (Ethernet) connecting them, along which data propagates. Nodes are either the *end points* at which data originates or is consumed, or *redistribution points* that store, possibly duplicate or filter, and then forward data to other nodes. Examples of redistribution points are routers, gateways, switches and hubs. These nodes can often be configured to change redistribution behaviour and monitored to give a view of data passing through them. The extent of monitoring and configuration that is provided depends on the device. To conclude, the node is the generic term that is used for any point in the network.

2.1.5 Server

A *server* is a node in a network that provides a set of services to the clients from the applications running on it.

2.1.6 Service

A *service* can be anything that can be used by clients over the network. The services may be devices as well as software.

2.1.7 Client

The user of service provided by the server is a *client*. The client is itself a node that acts as user to use the service provided by the server.

2.1.8 Management Station

It is a node in the network that runs a set of management applications to monitor and control other nodes in the network like server, clients, routers etc by communicating with them. It uses SNMP to communicate with other nodes in the network. The nodes that are monitored by management station are called managed nodes. There can be many management stations in the network. For further detail about management station, see Section 2.2.1.

2.1.9 Managed node

It is a node that is monitored by a management station in the network. Any node in the network can be monitored by a management station. For example, the monitored node can be a server, a client, a router, a probe, a switch, a hub or a gateway. There can be many managed nodes in the network. The communication between management station and the managed node is done using SNMP.

2.1.10 Probe

It is a managed node in the network that is configured by management station to monitor the traffic on the network segments on which it resides. There can be many probes in the network one each per network segment.

2.1.11 MIB

MIB stands for Management Information Base. It is the interface that is maintained at the managed node or probe. A management station configures the managed nodes or probes through this interface. The MIB consists of objects, some of which are organized in the form of tables. The management station sets and gets the value of these objects through SNMP. For further detail about MIB, see Section 2.2.3.

2.1.12 SNMP agent

It is daemon that resides on the managed nodes. It communicates with management station on behalf of managed nodes and probes on which it is present. It processes the SNMP requests from management stations and generates SNMP response on behalf of managed nodes and probes on which it is present. The agent processes the requests of the management stations that are in the community of the agent. For further detail about agent, see Section 2.2.2.

2.1.13 SNMP Community

An SNMP Community is a relationship between an SNMP agent and a set of management stations that defines authentication, access control. The community concept is a local one, defined at the managed node or probe. Each community is given a unique (within this agent) community name, and the management stations within that community must employ the community name in all operations.

The SNMP agent may establish a number of communities, with overlapping management station membership.

Since communities are defined locally at the SNMP agent, the same name may be used by different agents. This identity of names is irrelevant and does not indicate any similarity between the defined communities. Thus, a management station must keep track of the community name or names associated with each of the agents that it wishes to access.

2.1.14 Failure Detectors

A failure detector is viewed as a distributed oracle that provides hints about the failures in the distributed system [18]. The system components of the system being monitored by failure detectors could be entire sites, specific computers, processors within a computer, processes, threads, network interfaces, network connections, or any number of other low-level abstractions. Each component has its own failure detector. Factors such as the highly variable communication latency and best-effort service provided by today's wide area networks, and need to construct a scalable service impact dictate to consider unreliable fault-detection service [17]. The unreliable failure detectors are based on time-outs. They may erroneously indicate that a component

has failed only to correct this error at a later time [17].

The unreliable failure detector [19] used for monitoring processes, in spite of making mistakes can be used to solve consensus in distributed systems if it satisfy two properties namely completeness (which means that if a process is faulty, it should be suspected by the detector) and accuracy (which means that, if a process is suspected, it should be faulty). There are eight classes of failure detectors defined in terms of completeness and accuracy properties. $\diamond$S is the weakest unreliable failure detector that can be used to solve consensus in asynchronous distributed computations [19].

$\diamond$S Failure detector :

The $\diamond$S failure detector is one of the eight classes of failure detector proposed by Chandra and Toueg [19]. $\diamond$S failure detector is the weakest unreliable failure detector that satisfies the following properties:

- **Stong completeness**: Eventually every process that crashes is permanently suspected by every correct process.
- **Eventual weak accuracy**: There is a time after which some correct process is never suspected by any correct process.

and can be used to solve fundamental problems like consensus in distributed systems [19].

2.2 Simple Network Management Protocol (SNMP)

The Simple Network Management Protocol (SNMP) is the standard for management of data communication networks using TCP/IP. SNMP was issued in 1988 and was designed for network management of routers, hosts, workstations with low overhead and supporting multivendor network management. It is used to get management information from the managed resources of network architecture [6].

The model of network management that is used for SNMP includes the following key elements:

- Management station
- Management agent (it resides on the managed node)

- Management Information Base (MIB)
- Network Management Protocol

Figure 2.1 illustrates a SNMP managed network comprising these elements. SNMP defines a client/server relationship. The client program

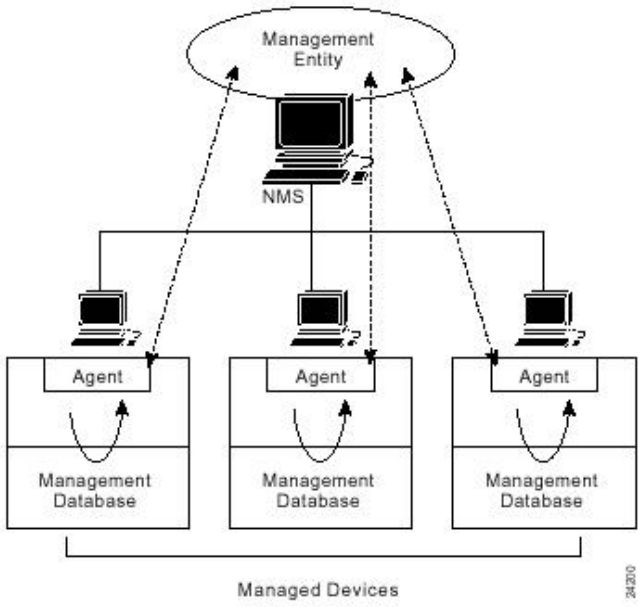

Figure 2.1: A SNMP managed network consisting of NMS, managed nodes, agent.

(network manager) makes virtual connections with the server program (the SNMP agent) and provides some information regarding the device status by accessing the management information base controlled by the agent [7].

Each of the components of the network management architecture of SNMP is explained in brief as following: Section 2.2.1 deals with management station, Section 2.2.2 deals with management agent, Section 2.2.3 deals with management information base (MIB), Section 2.2.4 deals with the management protocol (SNMP).

2.2.1 Management Station

A management station is typically a standalone device. The management station [6] will have, at minimum:

- A set of management applications for data analysis, fault recovery, and so on.
- An interface by which the network manager may monitor and control network.
- A protocol by which the management station and managed entities exchange control and management information.
- A database of information extracted from the management databases of all the managed entities in the network.

A management station executes applications that monitor and control managed devices. Management stations provide the bulk of the processing and memory resources required for network management.

2.2.2 Management agent

The SNMP-Managed devices (or nodes) all contain the SNMP agent software and MIB. Managed devices collect and store management information and make this information available to management station using SNMP. Managed devices, sometimes called network elements, can be routers and access servers, switches and bridges, hubs, computer hosts, or printers.

An agent has local knowledge of management information and translates that information into a form compatible with SNMP. The agent is a daemon, which is a background server process, which receives, authenticates and processes SNMP requests from management applications.

The SNMP agent software is usually quite small (typically less than 64 KB) because the SNMP protocol is simple. The Snmpd (means snmp daemon) agent uses an authentication scheme to determine which management station can access it's MIB, SNMP community, access mode and MIB view. The access policies are user configurable [8][9][10].

SNMP enables proxy management, which means that a node with an SNMP agent and MIB can communicate with other node that do not have the full SNMP agent software. This proxy management lets other nodes be controlled through a connected machine by placing the node's MIB in the agent's memory. It is also useful for managing nodes that do not support any part of TCP/IP protocol suite like some modems, bridges etc and to offload some nodes that are under heavy load [6][8].

2.2.3 Management Information Base (MIB)

Each managed node in a network maintains a MIB that reflects the status of the managed resources at that system. Each resource to be managed is represented as an object and an MIB is a structural collection of such objects. The MIB is defined using the structure of management information (SMI), which identifies the data types, naming and specifying the resources in a MIB.

All managed objects in the SNMP environment are arranged in a tree structure. The leaf objects are the actual managed objects. Associated with each type of object in a MIB is an object identifier. The objects have the access policies either "read-only", "read-write" or "write-only" [7].

The managed objects are comprised of one or more object instances, which are essentially variables and there are only two types of managed objects namely scalar and tabular. Scalar objects have a single instance and tabular objects have multiple instances. Objects in a MIB are specified in groups. The object is a "unit of implementation" - if an agent implements one object in a group, it should implement every object in that group, sometimes however it may not be possible [4].

The implementors of managed system and the management station should know the MIB definitions. The MIB is instantiated within the managed system [4].

MIB-II (management information base for network management of TCP/IP based internets) is the full standard MIB (STD#17) consisting of most of the standard MIB objects common to a wide variety of nodes. In addition to the standard objects defined under MIB-II, the MIB also supports a number of objects defined by vendors.

Vendors may support more management objects than those defined under MIB-II, to improve the ability to manage their nodes. These objects get added under another portion of the MIB tree, called private. The private subtree has a number of subtrees under it, which typically consist of one subtree for each enterprise. Enterprises allocate objects under their own subtrees to represent their device specific MIB objects [11].

2.2.4 Management protocol (SNMP)

The management station and management agent are linked by a network management protocol namely SNMP [1]. Managed nodes [10] are monitored

and controlled using four basic SNMP commands: get, set, trap, and traversal operations.

- The **get** command is used by a management station to monitor managed nodes. The management station examines different variables that are maintained by managed nodes.
- The **set** command is used by a management station to control managed nodes. The management station changes the values of variables stored within managed nodes.
- The **trap** command is used by managed nodes to asynchronously report events to the management station. When certain types of events occur, a managed device sends a trap to the management station. More information on traps is presented in Section 2.2.5.

Traversal operations are used by the management station to determine which variables a managed device supports and to sequentially gather information in variable tables, such as a routing table.

The SNMP has been through several iterations SNMPv1, SNMPv2, and SNMPv3. SNMP message contains two parts namely message header and protocol data unit. Figure 2.2 gives the basic format of SNMP message.

Figure 2.2: An SNMP message consists of a header and a PDU.

SNMP message headers contain two fields: Version Number and Community Name. The PDU in general contains PDU type, request type, error status, error index and variable bindings [1]. SNMP is advantageous to operate over UDP [4].

2.2.5 Trap messages

One of the basic commands of SNMP is trap command. The trap command is used by the managed nodes to asynchronously report events to the management station. When certain types of events occur, a managed node sends a trap to the management station.

The different events [1] that generates traps are:

- *coldStart* - the sender is reinitializing and it's configuration may change.
- *warmStart* - the sender is reinitializing but it's configuration will not change.
- *linkDown* - failure in one of the agent's links.
- *linkUp* - one of the agent's links has come up.
- *authenticationFailure* - the agent received a protocol message unproperly authenticated.
- *egpNeighborLoss* - an Exterior Gateway Protocol neighbour is down
- *enterpriseSpecific* - The trap is identified as not being one of the basic one.

As SNMP is advantageous to operate over UDP[4], the traps may be lost and they may never reach the management station.

One possible solution to prevent the loss of traps is to use TCP instead of UDP as the underlying transport protocol for SNMP. As the TCP is connection oriented, the SNMP traps are never lost. But the disadvantage of using TCP is that the network bandwidth is wasted for maintaining TCP connections. The main objective of SNMP is to provide network management with low overhead. The use of TCP is not advisable because of the overhead involved in maintaining the TCP connections.

The notion of the trap messages is extensively used in this report.

2.3 Network scenario considered

This section explains the network scenario that is considered to explain the failure detection of software and hardware failures in a network.

In abstract, the network scenario considered is that of a network in which one node acts as a server and one node acts as management station. The management station is present on another network segment than that of the server. The server is the important one that we want to observe. First the failure of application instances (software failure) running on the server is detected and then the failure of the server (hardware failure). It is the

management station that should know about both the software and hardware failures of the server.

The server has applications running on it and the failure of these applications is detected when the server is up. The server serves the requests from clients and so it will generate traffic in the network. If the server fails, then there will be no traffic from the server on the network. To detect the failure of server, the knowledge about the behaviour of server, in terms of the traffic generated by the server is important.

The information about the behaviour of the server presents an idea about the functioning of the server to the management station. The information about the IP packets coming from the server and the time elapsed since a last packet is seen from the server gives an idea of the working of the server. This knowledge about the last time an IP packet has been seen from the server is theoritically possible but not in practice.

To know the last time a packet is seen from the server, the packets coming from the server should be observed regularly at some predefined time interval. So the behaviour of server that is considered in this report is such that the server cannot be idle without sending any IP packet for a definite interval of time.

The management station has certain knowledge about the behaviour of the server. When the server deviates from it's behaviour, the management station suspects the server. The management station may suspect the server while the server is up or it may suspect the server while the server is down. Thus the failure detection of server reduces to imperfect failure detection as the management station may make the mistake of suspecting the server while it is actually up.

[19] introduced the concept of imperfect failure detection and unreliable failure detector. They claimed that unreliable failure detectors, which satisfies the properties of Completeness and Accuracy can be used to solve the fundamental problems like consensus, impossibility result associated with the asynchronous distributed computations.

In Chapter 4, we present three approaches that can be used by management station to detect the failure of the server. Two of the three approaches use a probe. The probe provides the information regarding the behaviour of the server to the management station. The probe in these approaches functions like a unreliable failure detector of [19] and which may make mistake of suspecting the server while it is actually up. We prove (in Section 4.4.2) that the probe satisfies the properties of strong completeness and eventual weak

accuracy of $\diamond$ S failure detector and so can be used to solve failure detection.

This concept of failure detection can be extended to detect failure of any node in the network.

Chapter 3

Application Instance Failure Detection

3.1 Introduction

In distributed systems, the computers perform an integrated computing facility by interacting with each other. Some computers provide a service to other computers. These computers that provide the service are termed as servers. The servers provide the service through the applications running on them. This chapter deals with detecting application instance failure when the server is working. The failure detection of server when the server is down is explained in chapter 4.

3.2 How do we know application instance failure?

The failure of application instance is known from the status of the application instance.

3.3 Detecting application instance failure

This section deals with detecting the failure of application instances running on the server by the management station.

As the application instance failure detection is done using SNMP, the server has an SNMP agent and maintains an interface in the form of Management Information Base (MIB). The MIB contains the information regarding the applications running or previously run on the server. A table in the MIB namely *sysApplrunTable*, contains information regarding the applications running on the server. An entry, *sysApplRunCurrentState*, in the table contains the current status of the application instance running.

The management station sends an SNMP request to the server requesting the value of *sysApplRunCurrentState* and the SNMP agent running on the server sends a SNMP response message that contains the value of the variable *sysApplRunCurrentState*. The possible values [12] of this variable are:

- Running
- Runnable but waiting for a resource such as CPU
- Waiting for an event
- Exiting or
- Other

The path in the MIB where this variable is located is /system application run group /sysApplrunTable/SysApplRunEntry/sysApplRunCurrentState.

So from the value of the variable *sysApplRunCurrentState*, the management station can know only whether

- The application instance is running on the server
- The application instance is waiting for a resource
- The application instance is waiting for an event
- The application instance is exiting.

Any other behaviour of application instance makes the SNMP agent return the value of *sysApplRunCurrentState* as *Other*. Thus this value *Other* can be configured by the management station to indicate only the application instance failure.

The server can also configured to generate trap message to the management station when an application instance fails.

3.4 Prerequisites

This section gives the prerequisites that should be satisfied to detect application instance failure.

The prerequisites for failure detection of application instances running on the server are:

1. The server is up and working.
2. The SNMP agent is up and working.
3. The applications are registered in the MIB maintained on the server.
4. The management station is registered with the SNMP agent.

Figure 3.1 presents these prerequisites in the form of a diagram.

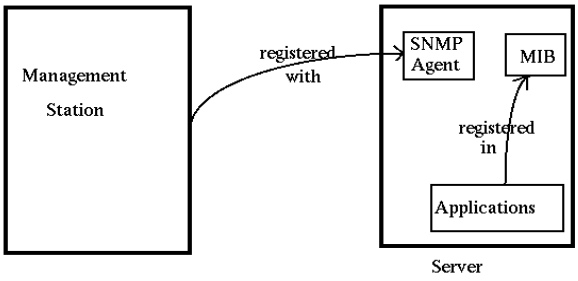

Figure 3.1: Diagramatic representation of prerequisites for detecting failure of Application Instances

3.5 Implementation

In order to provide the information about the application instance running on the server, Definitions of System-Level Managed Objects for Applications [12] (System Application MIB, RFC 2287, proposed standard) should be

implemented on the server. An SNMP agent should also be implemented on the server in order to communicate the management information regarding the applications to the management station.

According to [12], the implementation of the MIB on the operating system may require some considerable processing power to obtain the status information from the managed database. The management station polls the agent to get the status of different application instances running on the agent's node. In order to have the most recent information regarding the status of application instances, the agent should poll the MIB on the agent's node at regular intervals. This time interval can be set by manager.

The manager should use the variable *sysApplAgentPollInterval* present in the MIB at the server. The default value of 60 seconds is defined to keep the processing overhead low, while providing usable information for long-lived processes [12]. A manager is expected to adjust this value if more accurate information about short-lived applications is needed, or if the amount of resources consumed by the agent is too high [12].

Chapter 4

Server Failure Detection

4.1 Introduction

In Distributed Systems, the computers connected through an interface perform an integrated computation. The computers perform the computing by exchanging services with each other. Some computers provide a service to other computers. These computers that provide the service are termed as *servers* and the computer that utilize the service provided by them are termed as *clients*. The failure of servers leads to degradation in the performance and reliability of the distributed systems. Thus the failure detection of servers in distributed systems is very important.

4.2 What is server failure?

The *server* is expected to behave in a particular way and if it deviates from the expected behaviour then it is said to have failed. For example, the server is expected to behave in such away that it cannot be idle for more than some definite time-interval. If it is idle for more than the definite time-interval, then it is suspected and said to have failed if it remains idle forever. This chapter uses this example of behaviour about the server to detect the failure of the server.

4.3 Approaches to detect server failure

This section presents the different approaches that can be used by the management station to detect the failure of the server.

The approaches that can be used by the management station to detect the failure of server are Direct Approach, Indirect Approach and Mixed approach. Section 4.3.1 explains the Direct approach, Section 4.3.2 explains the Indirect approach and Section 4.3.3 explains the Mixed Approach.

4.3.1 Direct Approach

This section explains the Direct approach that can be used by the management station to detect the failure of the server.

In this approach, as the name implies, the failure of server is detected by directly contacting the server. An interface in the form of MIB is maintained on the server. The management station sends a SNMP request to the server asking the value of any object present in the MIB maintained at the server. One example for the object can be *sysuptime* if the MIB-II [15] is implemented on the server. If the server already has the MIB [12], which contains information about applications running on the server, the management station can send SNMP request asking the value of any object present in the MIB.

If the management station gets the SNMP response to the SNMP request from the server, then the server is up. If the management station does not get the response, then the server is down.

Thus this direct approach used for the failure detection of the server can also be used to detect the failure of application instances if [12] is implemented on the server.

Figure 4.1 illustrates the direct approach that can be used by the management station.

Drawbacks

The drawbacks of this approach are as follows:

1. As SNMP is advantageous to operate over UDP [4], the SNMP requests or SNMP response or both can get lost in the network. It may happen

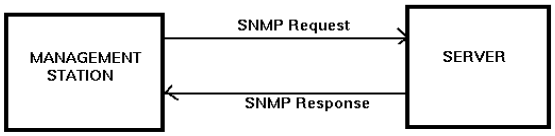

Figure 4.1: Direct approach for server failure detection

that the SNMP request from the management station may not reach the server or the SNMP response from the server may not reach the management station. Thus the management station may sometimes make the mistake of declaring the server to be dead while it is actually up. More over, this approach does not take into account the behaviour of the server.

2. The management station should poll server using SNMP requests at regular intervals. Thus increasing the messages in the network.

3. This approach does not give fast failure detection of server when the server has failed. The reason for this is that the management station knows about the failure of the server only when it polls the server. If the management station does not poll the server at the instant it failed, then it knows about the failure of the server only when it polls the server the next time.

4.3.2 Indirect approach

This section presents the Indirect approach that can be used by the management station to detect the failure of the server.

In this approach, the management station receives the help of a probe to detect the failure of the server. The probe helps the management station by notifying it about the deviation in the behaviour of the server. The probe

has an interface in the form of MIB. The management station configures the probe through the interface to make the probe provide the information about the behaviour of the server.

The management station configures the probe through the MIB such that the probe monitors the packets coming from the server. The probe checks for the count of number of packets coming from the server at every pre-defined time interval. This time interval is set by the management station in the MIB at the probe. The probe generates events when the difference between the count at two successive time intervals is either zero or one. These events are logged in the MIB at the probe and are reported to the management station in the form of traps.

The probe sends the trap indicating server failure when the difference between two successive counts is zero and a trap indicating server alive when the difference is one. After receiving the trap indicating server failure, the management station declares that the server has failed.

The traps may be lost and they may never reach the management station. So for this reason the management station configures the probe in such away that it logs the events indicating server failure and server alive before sending these events as traps to the management station. So the management station has to periodically check the log table in the MIB at the probe for the occurrences of these events.

Figure 4.2 illustrates the Indirect approach that can be used by the management station to detect failure of the server.

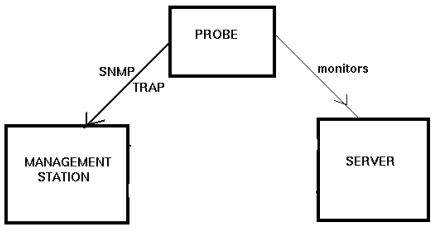

Figure 4.2: Indirect approach for server failure detection

Drawback

In the Indirect approach, the management station declares the server to be dead when it receives a trap message indicating server failure from the probe. This declaring of failure of the server based solely on trap has a drawback that can be explained with the following example:

Consider that the server is expected to behave in such away that it cannot be idle without sending any packet for more than 20 seconds. If no packet is seen from the server during 20 seconds time interval, the probe sends a trap to the management station indicating that the server has failed. The management station after receiving the trap declares that the server has failed. It may happen that due to traffic problem in the network, a packet from the server may reach the probe at 22nd second. In this case, the server is declared to be dead by the management station while it is actually up.

Thus if the decision to declare the failure of server is based solely on traps, then the time-interval over which the probe checks for count of the packets from the server is very critical for the correct failure detection of server.

Merit

The advantage of this method is that only the probe and management station should support SNMP. The server need not support SNMP.

Requirements

In order to follow the indirect approach, the following requirements have to be met:

1. The management station is up
2. The probe is up.
3. The management station is registered with the SNMP agent of the probe.
4. The probe should have it's interfaces in promiscuous mode because it need to monitor the packets being received on it's interfaces.
5. The probe is configured by management station to monitor the packets coming from the server.

6. The management station and probe should support SNMP.

4.3.3 Mixed Approach

This section explains the Mixed approach that can be used by the management station to detect the failure of the server.

This Mixed approach is a combination of Indirect approach (Section 4.3.2) and Direct approach (Section 4.3.1).

In this approach, the management station first waits for a trap indicating failure of the server from the probe (Indirect approach). After receiving the trap, the management station becomes suspicious about the working of the server. To confirm it's suspicion that the server has failed, the management station now uses the Direct approach (Section 4.3.1) i.e. it now sends a SNMP request to the server asking the value of any object present in the MIB at the server and waits for SNMP response.

It may happen that initial SNMP request or response can get lost. If the management station does not get the response to it's initial request from the server, the management station should send SNMP requests to the server repeatedly until it gets the SNMP response for a time-interval equal to the twice the time-interval over which the probe checks for the count of packets coming from the server or the time interval expires.

The reason for this is that the management station is trying to confirm the failure. If the server is alive, the probe will send a trap indicating server alive during the next time interval when it sees packet from the server. So if there is no SNMP response to the initial SNMP request, then the management station should keep sending SNMP requests until it receives the response before the end of next time-interval or the time-interval expires.

If there is no SNMP response from the server for the SNMP requests sent by the management station and no trap from the probe indicating server alive in the next time-interval, then the management station declares the server is dead.

The traps may be lost and they may never reach the management station. The management station configures the probe in such away that it logs the events indicating server failure and server alive before sending these events as traps to the management station. So the management station has to periodically check the log table in the MIB at the probe for the occurrences of these events.

The mixed approach can also be used to detect the failure of application instances if [12] is implemented on the server. Figure 4.3 illustrates the Mixed approach.

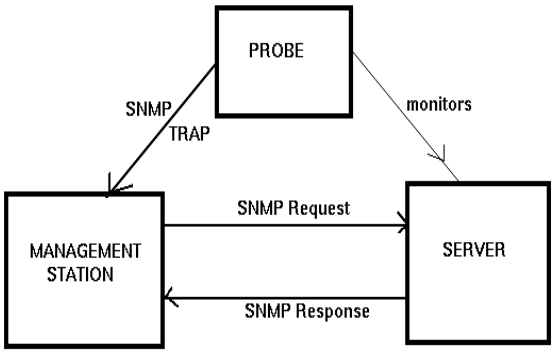

Figure 4.3: Mixed approach for server failure detection

This working of Mixed approach is explained with an example as follows:

Example

Consider that the server is expected to behave in such away that it cannot be idle with out sending any packet for more than 20 sec. The management station configures the probe to check the number of packets from server every 20 sec. If the difference between number of packets seen during two successive time-intervals is either zero or one, the probe sends a trap to management station. When the difference is zero, the probe sends a trap indicating that the server has failed.

After receiving the trap indicating failure of the server, the management station sends SNMP request to the server asking the value of any object present in the MIB at the server and waits for SNMP response. The management station sends SNMP requests to the server until it gets SNMP response from the server for duration of 40 seconds.

If there is no SNMP response during that 40 seconds and even no traps from the probe to the management station indicating that the server is alive, then the management station declares that the server is dead.

Requirements

In order to follow the Mixed approach, the following requirements have to be met:

1. The management station is up
2. The probe is up.
3. The management station is registered with the SNMP agent of the probe.
4. The probe should have it's interfaces in promiscuous mode because it need to monitor the packets being received on it's interfaces.
5. The probe is configured by management station to monitor the packets coming from the server.
6. The management station, probe and server should support SNMP.

4.4 Logic to be followed by the probe

This section explains the logic used by the probe to report the traps indicating failure of the server and liveliness of the server to the management station in the Indirect (Section 4.3.2) and Mixed approach (Section 4.3.3) to detect the failure of the server. The management station configures the probe through the MIB at the probe to make the probe perform the logic mentioned in Section 4.4.1.

4.4.1 Logic

The probe has it's interfaces in promiscuous mode in the Indirect and Mixed approach used by the management station to detect the failure of the server. Because of this mode, the probe monitors each and every packet travelling on the network segment on which it is present [1].

In order to have information about the server, the probe first filters the packets coming from the server. For every packet received on it's interface, the probe sees for a match of source address in the packet with that of server's address. If a match occurs, then it filters the packet. These filtered packets form a channel [1].

The channel keeps a count of the number of filtered packets passing through it. This value of the count of the number of filtered packets passing through the channel is sampled regularly at some pre-defined time interval.

If the difference between two successive samples is zero then the probe logs the event "server failure" and sends a trap message indicating the failure of the server to the management station. If the difference between two successive samples is one, then the probe logs the event "server alive" and sends a trap message indicating that the server is alive to the management station. The events that are to be logged when there is either zero or one packet from the server namely "server failure" and "server alive" are defined by the management station in the MIB at the probe.

Thus the configuration of the probe by the management station makes the probe perform the following functions:

1. Filter packets coming from the server.
2. Pass the filtered packets through a channel and keep a count of number of filtered packets passing through the channel.
3. Sample the count of number of filtered packets passing through the channel regularly at predefined time interval and generate events.
4. Log the events and generate a trap message indicating failure of the server or liveliness of the server.

This logic is represented in the form of flow chart as shown in figure 4.4.

4.4.2 Properties satisfied by the probe

The logic used by the probe (Section 4.4.1) accomplishes the properties namely strong completeness and eventual weak accuracy of $\diamond$S failure detector. The justification for stating this as follows:

Strong completeness:

It states that eventually any failure of a process is suspected permanently by every correct process[19].

When the probe is up, the probe may initially make the mistake of sending the trap indicating server failure when it does not see a packet from the server during a predefined definite time-interval. It later corrects the mistake by generating the trap indicating server alive when it sees a packet from the server during a definite time-interval.

For example, the server may be slow but not completely dead and hence may send less number of IP packets. The probe checks for the number of packets from the server during every predefined definite time-interval. The probe initially will generate the trap indicating server failure when it does not see a packet during the predefined definite time-interval. Because of the slow working of the server, a packet will be seen by the probe after some time-intervals. At that point, the probe corrects it's mistake of declaring the failure of the server by sending the trap indicating server alive to the management station.

If the server is dead, it will no longer send any packets and the probe will initially generate a trap indicating server failure when it does not see a packet during a predefined time-interval. The probe will never generate the trap indicating server alive as it will not see a packet from the server in the future time-intervals. Thus eventually inspite of making mistakes in the beginning, the failure of server is detected. Thus the strong completeness property is satisfied.

Eventual weak accuracy:

It states that there is a time after which a correct process is never suspected by any other correct process [19].

The probe may initially make the mistake of sending the trap indicating server failure to the management station when it does not see a packet from the server during a predefined time interval only to correct it later.

If the server is correct and is functioning properly, it will be sending the IP packets regularly and so the probe will see the packets at definite time-intervals and so it will not send the trap indicating server failure to the management station.

Thus there is a time after which the probe will never suspect the server

and will not generate traps indicating server failure. Thus satisfying the eventual weak accuracy property.

[19] claims that $\diamond$S failure detector can be used to solve fundamental problems in distributed systems. As the probe satisfies the strong completeness and eventual weak accuracy properties of $\diamond$S failure detector, by induction the probe can be used to solve the failure detection of nodes.

4.4.3 Short note on traps generated by probe and ports

The fourth function of the probe, as mentioned in Section 4.4.1, is to log the events and generate trap message indicating server failure and server alive. The probe can also be configured to generate only one trap namely server failure or server alive. But doing so has its own drawbacks which can be explained with an example.

Example

Consider that the server is alive and the probe is configured to generate only trap indicating server failure. The probe checks for the number of packets coming from the server at definite predefined time interval. Due to network delay, a packet may reach the probe after the definite time interval.

In that case, when the probe does not see a packet from the server during the time interval, it sends a trap indicating server failure to the management station. The management station then sends SNMP request to the server until it gets SNMP response or finally gives up and declares the server to be dead. If the network condition is very bad and the messages are lost, then the management station spends long time in polling the server to confirm the failure. Thus increasing the number of messages in the network.

In the above case, if the probe is configured to generate both the traps ie the trap indicating server failure and trap indicating server alive, then it will generate the trap indicating server alive when it sees the packet from the server. There is a possibility that the trap indicating server alive may reach the management station and the management station will stop polling the server if it has not got the SNMP response till that time. Thus preventing the management station from sending more messages on the network.

The probe can also be configured to monitor the traffic with respect to the protocols used in the network. Some of these protocols use a fixed port number and monitoring the traffic with respect to a protocol implicitly means

monitoring the port used by the protocol.

For example, if the probe is configured to monitor the traffic associated with SNMP, and then it implicitly means that the probe monitors the port 161 as SNMP uses port 161. The management station defines the protocols that the probe should interpret in the MIB at the probe.

4.5 Implementation

The section presents the implementation for the failure detection of the server. The implementation depends on the approach that is used by the management station to detect the server failure.

4.5.1 Implementation for Direct approach

In order to use the Direct approach for server failure detection, the server should implement any MIB. For example, the server can implement the MIB [12] used to detect application instance failure or any other MIB. An SNMP agent should also be implemented on the server to communicate with the management station.

4.5.2 Implementation for Indirect approach

This section deals with the implementation for the Indirect approach used by the management station to detect the failure of the server.

The probe is an integral part of indirect approach used by the management station to detect the failure of server. The probe should implement filter group, alarm group, event group of RMON MIB [13] if the probe filters the packets coming from the server based on MAC address. The filter group, Alarm Group, Event Group of RMON MIB [13] and Protocol Directory Group of RMON MIB-2 [14] should be implemented on the probe if the probe is configured by the management station to filter the packets coming from the server based on IP address.

[13] tells that the implementation of [13] must also implement system group of MIB-II [15] and the IF-MIB [16]. The probe should also implement an RMON agent, which is not different from any SNMP agent [1] to communicate with the management station using SNMP.

4.5.3 Implementation for Mixed approach

The probe is an integral part of mixed approach used by the management station to detect the failure of the server. The probe should implement the filter group, Alarm Group, Event Group of RMON MIB [13] if the probe filters the packets coming from the server based on MAC address. The filter group, Alarm Group, Event Group of RMON MIB [13] and protocol directory group of RMON MIB-2 [14] should be implemented on the probe if the probe is configured to filter the packets coming from the server based on IP address. [13] tells that the implementation of [13] must also implement system group of MIB-II [15] and the IF-MIB [16].

The probe should also implement an RMON agent to communicate with the management station using SNMP and the server should implement SNMP agent and any MIB. An example of MIB that can be implemented on server is [12].

4.6 Extensions

This section deals with extending the concept of the server failure detection to detect the failure of all nodes in the network.

In this chapter, the approaches that can be used by the management station to detect the failure of the server were discussed. Two of the three approaches presented for failure detection of server has a probe as an integral component. The functionality of the probe is that it sits on network segment and monitor the traffic of the network segment.

The failure detection scenario presented so far can be extended by configuring the probe to monitor the packets coming from all nodes present on the network segment and notify about the behaviour of each node on the segment to the management station in the form of logs and traps. The network can be structured in such away that there exists one probe per each network segment that gathers statistics and information about the behaviour of nodes on the network segment and notify it to the management station.

The management station can use the approaches presented in this chapter to detect the failure of each node in the network.

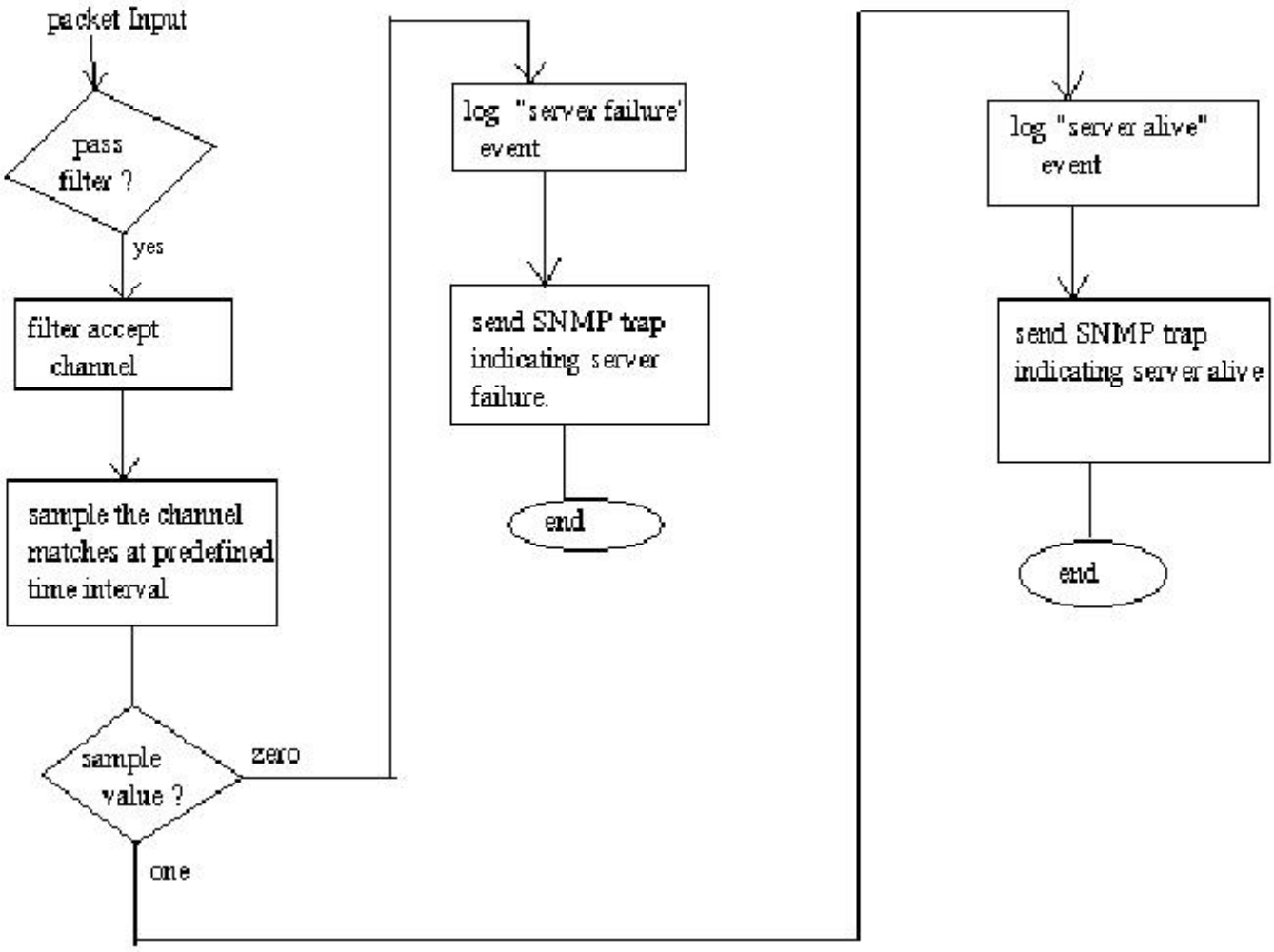

Figure 4.4: Flow chart of the logic used by probe.

Chapter 5

Configuring the probe

This chapter deals with configuring the probe by the management station to make the probe perform the logic mentioned in Section 4.4.

In this chapter, Section 5.1 gives an introduction, Section 5.2 deals with configuring the probe to perform the filtering function, Section 5.3 deals with configuring the probe to accept the filtered packets into a channel, Section 5.4 deals with configuring the probe to perform the sampling on the number of packets passing through the channel at pre-defined time interval and Section 5.5 deals with configuring the probe to log the events and generate trap messages to the management station.

5.1 Introduction

The probe is an integral part of the indirect approach (Section 4.3.2)and mixed approach (Section 4.3.3) used for the failure detection of the server. The probe sits on a network segment and monitors the packets on that segment.

The probe does a lot [2], but the management station should ask the probe to report the failure of the server. Thus the management station configures the probe in such away that it reports the failure of server to it. The probe has an interface in the form of MIB on it. The management station configures the probe through the MIB. The MIB consists of groups and several of the groups in the MIB consists of pairs of tables namely *control tables* and *data tables* [2].

The management station configures the probe by setting values to the

variables present in the control tables [2]. In the logic (Section 4.4) the probe follows to report the failure of the server, the probe performs four functions namely

1. Filter packets coming from the server
2. Pass the filtered packets through a channel and keep a count of number of filtered packets passing through the channel.
3. Sample the count of number of filtered packets passing through the channel regularly at predefined time interval and generate events.
4. Log the events and generate a trap message indicating server failure or server alive.

Thus the management station should configure the probe to make the probe perform these four functions.

5.2 Filtering

This section deals with configuring the probe by the management station to make the probe perform the filtering function (Section 4.4) in the logic followed by the probe.

The probe has it's interfaces in promiscuous mode and so it monitors all the packets which arrive or pass through it's interface. Hence the probe will receive packets from even other nodes in the network. Thus in order to have information of packets only from the server, the probe should filter packets sent by server only. The management station configures the probe through the Filter table present in the MIB at the probe. This configuring of filter table makes the probe perform the filtering function [1].

In order to filter the packets coming from the server, the probe checks the source address field in the packets received on it's interfaces. If the source address is server's address then the probe filters the packet. The probe can check for a match of server's MAC address or IP address in the packets received.

The values that are to be entered by the management station in the objects of the Filter table are given in Section 5.2.4. Filtering the packets coming from the server based on it's MAC address and IP address has its own positive and negative points which are discussed later in Sections 5.2.1 and 5.2.2 respectively.

In order to make the probe filter the packets based on IP address, the IP should be defined in the MIB at probe. This defining of protocols in the MIB is mentioned in Section 5.2.3.

5.2.1 Filtering on the basis of MAC address

This section focusses on the filtering of packets coming from the server on the basis of MAC address by the probe.

Principle

The management station configures the probe to filter the packets coming from the server based on MAC address of the server. The management station configures the probe through the filter table present in the MIB at the probe. The probe checks for a match of the source MAC address in the packets received with that of the server's MAC address. If a match occurs then it filters the packets. If a match does not occur, it does not filter the packet.

Placement of the probe

Any node in the network can be configured as a probe. The filtering of the packets from the server based on MAC address is useful only when the probe is on the same network segment as that of the server. The reason for this is explained in the drawbacks. If a redistribution node like router is configured as a probe, then it should have one of the interfaces connected to the network segment on which the server is present.

Advantages

Only the Filter group, Alarm group, Event group of RMON MIB [13] be implemented on the probe. There is no need to implement Protocol Directory Group of RMON MIB-2 [14].

Drawbacks

1. The probe should reside on the same network segment as that of the server in order to filter the packets coming from the server based on

server's MAC address. Thus restricting the existence of probe on the same network segment as that of the server. If the probe is on different network segment than that of server, the filtering of packets coming from the server on the basis of MAC address will not work.

For example, consider that the server is on network segment 1 and the probe is on network segment 3. These two segments are connected by redistribution node (eg:router). The packet from the server passes through the redistribution node to reach the probe. When the redistribution node gets the packets from the server, it removes the MAC layer addresses in the packet and attaches new MAC layer address. Thus when the packet reaches the probe it will have the source MAC address of the redistribution node but not the MAC address of the server.

2. If the network card of the server is changed, then the server MAC address will change. So the probe should be reconfigured every time the network card of the server is changed.

Requirements

If a redistribution node is to be configured by the management station as a probe, then the management station should have the necessary permission to configure it.

5.2.2 Filtering on the basis of IP address

This section focusses on the filtering of packets coming from the server on the basis of IP address by the probe.

Principle

The management station configures the probe to filter the packets coming from the server based on IP address of the server. The management station configures the probe through the filter table and the Protocol directory group present in the MIB at the probe. The probe checks for a match of source IP address in the packet with that of the server's IP address. If a match occurs, the probe filters the packet. Otherwise it will not filter the packet.

Placement of the probe

The probe can reside on the same network segment as that of the server, or the probe can reside on any segment if it is able to receive packets from the server. For example, if the server uses IP Multicast to send packets and the probe is on the network segment that receives the multicast packets sent by the server , then it will be able to monitor the packets from the server.

Merits

There is no need to reconfigure the probe when the server's network card is changed because filtering of packets is done based on IP address and IP address of the server will not change when the network card is changed.

Requirements

1. If the probe is on another segment then that of server
   (a) The server should be programmed with IP multicast.
   (b) The probe should register as a member of multicast group to receive packets from the server.
2. If a redistribution node is to be configured by the management station as a probe, then the management station should have the necessary permission to configure it.

5.2.3 Defining protocols in the MIB

This section deals with configuring the probe by the management station to make the probe interpret IP.

If the management station wants the probe to filter the packets based on IP address, then it should define IP in the protocol directory group that is present in the MIB at the probe. The filter that is defined in the filter table to filter the packets coming from the server based on IP address should have a pointer to IP defined in the protocol directory group. Thus to make the probe interpret the IP, the management station should configure this protocol directory group.

The protocol directory group has a *protocolDirTable* and the management station configures the objects in this table. The objects in the *protocolDirTable* that are configured and the values that are set to these objects as follows:

- *protocolDirID* - In this object, the management station should specify an unique octet string for a specific protocol. For example, for IP running over Ethernet the *protocolDirID* is 8.0.0.0.1.0.0.8.0. [1].
- *protocolDirParameters* - The value of this object should be set to 0.0.0.1.0 .0.8.0.4.0.1.0.0 [1] to make the probe count the fragments correctly for IP and above.
- *ProtocolDirType* - The manangement station should set this object to *addressRecognitionCapable(1)*, to make the probe not only count packets for the IP but also to recognize source and destination address fields [1].

Other variables in the protocol directory group should be set to *notsupported(1)*.

5.2.4 Configuring Filter table

This section deals with configuring the filter table present in the MIB at the probe, to make the probe perform the filtering function.

Each entry in the filter table is called a filter and the management should define a filter in this table in such away that the probe should only filter the packets coming from the server. The management station defines filter by setting values to the objects present in the filter table. The objects and the values the management station should set are as follows:

- *FilterChannelIndex* - The management station should set this variable to an index pointing to a channel in the channel table that accepts the filtered packets from this filter.
- *FilterPktOffset* - In this object, the management station should specify the offset in the packet where the filter begins to find a match. If the filtering is done based on MAC address, the value of this object should be equal to the bit position in the packet at which the source MAC address starts. If the filtering is done based on IP address, the value

of this object should be equal to the bit position in the IP packet at which the sources IP address starts.

- *FilterPktData* - The management station should set this object to server's MAC address or IP address depending on the choice to filter the packets based on MAC address or IP address.

- *FilterOwner* - In this object, the management should write it's name in the form of owner string. This object is used to recognize which management station created the entry.

There are other objects in the filter table that are concerned with the status of packets. To detect the failure of the server, it does not matter whether a packet received is short or long because what matters is only whether a packet from the server is seen or not. So these objects concerned with the status of the packet are assigned empty values.

[1] gives further information about the filtering concept.

5.3 Channeling the filtered packets

This section deals with configuring the probe by the management station to make the probe perform it's second function. The second function of probe is to pass the filtered packets from the server through a channel and keep a count on the number of packets passing through the channel.

After the filtering function (Section 4.4), the packets that are filtered form a stream of information called as channel [3]. The channel can perform many functions [2] like:

- It accepts the packets when the packets pass the filter or fail the filter.

- It can generate events or capture the packet when a packet enters the channel.

- It can be turned on or off by events occurring else where in the MIB.

- Keeps a count of number of channel matches i.e. keeps count of number of packets passing through the channel.

But the management station should configure the channel only to accept filtered packets of server and keep a count of number of filtered packets

passing through the channel. The management station configures the channel through the channel table present in the MIB at the probe. The values that the management station should enter in the objects present in the channel table are given in Section 5.3.1.

After configuring the channel, the object *channelMatches* contains the count of number of filtered packets passing through the channel.

5.3.1 Configuring the channel table

This section deals with configuring the channel table by the management station to make the channel accept only the filtered packets coming from the filter and keep a count of number of filtered packets passing through the channel.

The management station configures the channel table by setting values to the objects present in the channel table. The objects and the values that the management station should enter are as follows:

- *channelIfIndex* - The management station should set this object to index that identifies the interface of the probe to which the filter is applied to allow the data into the channel. This value should be one of the values of interfaces defined in the IF-MIB [16].
- *ChannelAcceptType* - The management station should set this object to acceptMatched(1) in order to make the channel accept only the filtered packets.
- *ChannelDataControl* - The management station should set this object to off(2) (default value) as it does not want the channel to generate an event .
- *channelOwner* - In this object, the management should write it's name in the form of owner string. This object is used to recognize which management station created the entry [1].
- *ChannelStatus* - The management station should set this object to valid (1). It ensures security for the entry made in the table by allowing only the management station that created it change the entry [1].

The remaining objects in the table should be assigned empty values in order to make the channel accept only the filtered packets and keep count on the number of filtered packets passing through the channel.

[1] gives further information about the concept of channel.

5.4 Sampling the number of filtered packets

This section deals with configuring the probe by the management station to make the probe perform it's third function.

The third function of the probe is to sample the count of the number of filtered packets passing through the channel regularly at predefined time interval and generate events corresponding to failure of the server and liveliness of the server. The management station configures the probe through the Alarm table present in the MIB maintained on the probe to make the probe perform this function.

An entry in the Alarm table identifies a variable, a time interval and the kind of count that should be tested. It chooses whether the management station want to check the value of the variable or the change in the value of the variable. An entry also includes the thresholds: a Rising Threshold with corresponding event, a Falling Threshold with corresponding event [2].

The management station should specify *channelMatches* as the variable, a time-interval and the count to be the difference between two successive values of *channelMatches* that should be tested. The management station specifies the Rising Threshold as 1 with corresponding event because an event should be generated when there is one packet from the server. The management station specifies the Falling Threshold as 0 with corresponding event because an event should be generated when there is zero packet from the server.

The objects in the Alarm table and the values the management station should set for these objects are given in Section 5.4.1.

5.4.1 Configuring the Alarm table

This section deals with configuring the Alarm table by the management station to make the probe perform it's third function. The management station configures the Alarm table by setting values to the objects present in the Alarm table. The objects and the values that the management station should enter for the objects are as follows:

- *alarmInterval* - In this object, the management station should specify the interval (in seconds) over which the sampling should be done in

this object.

- *alarmVariable* - The management station should set this variable to object identifier of *channelMatches* object of channel table.
- *alarmSampleType* - The management station should set this object to deltaValue (2) as the difference between two consecutive samples is compared to zero or one.
- *alarmStartupAlarm* - The management station should set this object to *risingOrFallingAlarm (3)* in order to make the probe generate an event when there are zero or one packets from server during the predefined time interval.
- *alarmRisingThreshold* - The management station should set this object to 1 as the upper bound for the difference between two consecutive samples to generate an event is 1.
- *alarmFallingThreshold* - The management station should set this object to 0 as the lower bound for the difference between two consecutive samples to generate an event is 0.
- *alarmRisingEventIndex* - The management station should set this object to the index of the event which is generated when there is one packet from server during the predefined time interval. In other words, this object should have the value of the index that identifies the event "server alive". The event is defined in the Event table.
- *alarmFallingEventIndex* - The management station should set this object to the index of the event which is generated when there is zero packet from server during the predefined time interval. In other words, this object should have the value of the index that identifies the event "server failure". The event is defined in the Event table.
- *alarmOwner* - In this object, the management should write it's name in the form of owner string. This object is used to recognize which management station created the entry [1].
- *alarmStatus* - The management status should set this object to valid (1) in order to make the entry to be valid one [1].

After configuring the above values, the object *AlarmValue* contains the value of the statistic i.e. the count of the packets during the last sampling

period. It is this value that is compared to the rising and falling thresholds [1].

[1] gives further information about the alarm table.

5.5 Logging the events and generation of a trap message

This section deals with configuring the probe by the management station to make the probe perform it's fourth function. The fourth function of the probe is to log the events corresponding to failure of server and liveliness of server and generation of trap messages indicating server failure and server alive to the management station.

The management station configures the probe through the Event table and log table present in the MIB at the probe to make the probe perform it's fourth function.

Each entry in the Event table identifies an event and an action, such as logging or sending a trap or both and an event community to which the trap should be sent [2]. The management station defines the events that are to be generated when there is zero or one packet from the server during the predefined time interval in the event table.

The objects *alarmRisingEventIndex* and *alarmFallingEventIndex* present in the Alarm table (Section 5.4.1) have pointers to the events defined in the Event table. The log table records the events that are to be logged. The values that the management station sets to the objects in the Event table and Log table are given in Section 5.5.1.

5.5.1 Configuring Event table and Log table

This section deals with configuring the Event table and Log table by the management station to make the probe perform it's fourth function. The fourth function of the probe is to log the events corresponding to failure of server and liveliness of server and generation of trap messages indicating server failure and server alive to the management station.

The management station configures the Event table and Log table by setting values to the objects present in the Event table and Log table. The objects and the values that the management station should enter for the

objects are as follows:

Event table

The Event table should have two entries. One entry for an event that will be generated when there is zero packet from the server and the other when there is one packet from the server during the predefined time interval.

- *eventDescription* - The management station should specify the description of event in this object. The management station for one entry should specify this object as "server failure" and for other entry as "server alive".
- *eventType* - The management station should set this object to log-and-trap (4) as the management station wants the event to be logged and sent as trap to it [1].
- *eventCommunity* - The management station in this object should specify the community to which the trap should be sent. The management should have to be in the community to receive the trap [1].
- *eventOwner* - The management should specify it's identity in the form of owner string. This object is used to recognize which management station created the entry [1].
- *eventStatus* - The management status should set this object to valid (1) in order to make the entry to be valid one [1].

Log table

The events that are generated from the event table are logged in the log table. There is no object in the Log table that can be set by the management station. The setting of the object *eventType* of Event table to 4 automatically creates log entries in the log table.

Each entry in the log table has an *logEventIndex* which identifies the event that is logged. There is an object called *logIndex* that identifies a particular log entry among all entries associated with the same event type. The object *logtime* gives the value of *sysUpTime* corresponding to each log entry.

5.6 Interaction between tables of MIB

As explained in Sections 5.2 to 5.5, protocol directory group, filter table, channel table, event table and log table present in the MIB implemented at the probe are configured to make the probe accomplish its functions. The interrelationship between the tables is as shown in figure 5.1. The figure 5.1 is shown with tables containing objects that are used to interrelate with other tables when the probe filters the packets based on IP address. The figure contains some sample values to explain the interrelationship between tables.

In the protocolDirTable of the protocol directory group, *protocolDirLocalIndex* uniquely identifies each entry. The filter table identifies the protocol that it should filter from *filterProtocolDirLocalIndex* and *filterProtocolDirDataLocalIndex*. Both of these objects in filter table has the same index value as that of ProtocolDirLocalIndex. In the figure 5.1, these objects all have a value 1.

The filter table identifies to which channel the packets are sent through *filterChannelIndex*. The *filterChannelIndex* identifies a channel in the channel table. In the figure 5.1, the *filterChannelIndex* identifies a channel with *channelIndex* 1.

In the alarm table, the object that is to be sampled is mentioned in *alarmVariable*. The *alarmVariable* identifies the object *channelMatches* in the channel table. The *alarmFallingEventIndex* and *alarmRisingEventIndex* uniquely identifies an event in the event table. In the figure 5.1, *alarmFallingEventIndex* identifies an event "server failure" because the value of *alarmFallingEventIndex* in alarm table and *eventindex* for the event "server failure" in Event table is 2. Like wise, *alarmRisingEventIndex* identifies an event "server alive".

In the log table, the object *logEventIndex* identifies the event that is logged. The object *logEventIndex* and *eventIndex* have the same value for a particular event. In the figure 5.1, the event "server alive" has the *eventIndex* as 2. The log of the event "server alive" also has the *logEventIndex* as 2.

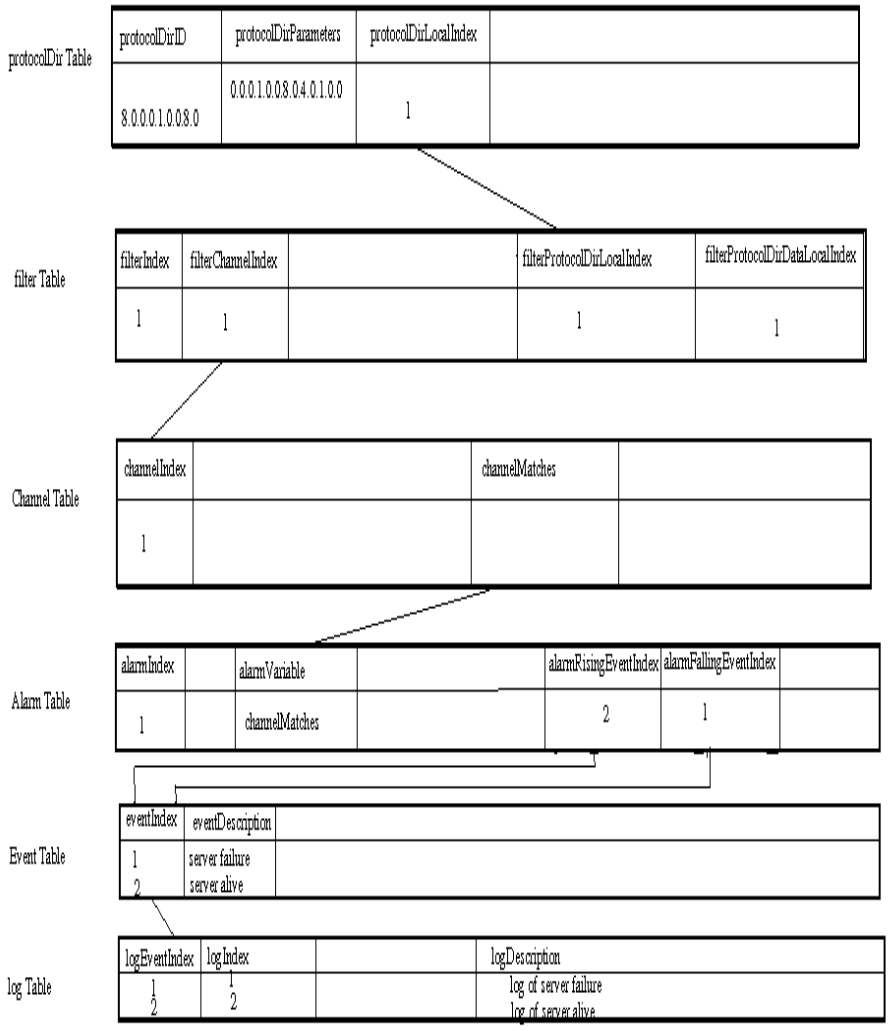

Figure 5.1: Interrelationship between tables of MIB maintained at the probe

Chapter 6

Probe Failure Detection

6.1 Introduction

In the Indirect approach (Section 4.3.2) and Mixed approach (Section 4.3.3) used by the management station to detect the failure of the server, the probe is considered to be reliable and it never fails. As the probe is also a node in itself, it may also fail. So the management station should also detect the failure of the probe.

6.2 Detecting probe failure

This section deals with detecting the failure of the probe by the management station.

The probe has a local monitor maintained on it. The local monitor generate heart beat message periodically to inform the management station that the probe is still alive. The management station has a $\diamond$S unreliable failure detector (Section 2.14) that receives the heartbeats from the local monitor of the probe.

If the failure detector on the management station does not receive the heartbeat message from the probe in a specific time bound, the management station sends an SNMP request to the probe requesting the value of any object present in the MIB maintained on the probe and waits for SNMP response from the probe for a definite time interval. During the time-interval it waits for SNMP response, if there is no SNMP response and even no heartbeat from the probe to the failure detector, then the management station

declares the probe to be dead.

6.3 Requirements

The $\diamond$S unreliable failure detector should be registered with the local monitor of the probe to receive heartbeats from local monitor.

6.4 Implementation

In order to detect the failure of the probe, the probe should implement a local monitor on it. The probe should also implement the filter group, alarm group, event group of RMON MIB [13] or the filter group, alarm group, event group of RMON MIB [13] and protocol directory group of RMON MIB-2 [14] depending on the choice of filtering the packets on the basis of MAC address or IP address.

The probe should also implement a RMON agent. The RMON agent is same as that of SNMP agent [1]. The management station apart from management applications should also implement a failure detector.

6.5 Extensions

This chapter till now dealt with the failure detection of a probe by the management station and this was done using the concept of failure detectors. This failure detection concept can be extended to detect the failures of all probes in the network with each probe in the network sending heartbeat messages to the management station. The time-interval over which the probe sends heartbeat messages can be relaxed as the management station polls the probe at regular time-intervals to check the log table present in the MIB at the probe for occurrence of events related to failure of nodes on the network segment.

If the network has multiple management stations, then a hierarchial approach can be followed to detect the failure of probes. The hierarchial approach can be explained with an example as follows:

Example:
Consider that there are two management stations in the network and these

management stations should know about the failure of all the probes in the network. The probes in the network can be divided into two groups such that one group of probes sends heartbeats to one management station and the other group of probes sends heartbeats to other management station. These two management stations then exchange information about the suspected probes between them. In this way, the management stations will know about the failure of all the probes in the network.

Chapter 7

Conclusion

This chapter compares the traditional approach and SNMP approach.

7.1 Comparison between traditional approach and SNMP approach

This section presents the comparision between the traditional failure detction approach and failure detection using SNMP in terms of network bandwidth, performance and scalability, heterogeneity and interoperability. This section presents the comparision by comparing the two methods from the point of view of management station.

7.1.1 Network Bandwidth

Network Bandwidth is related to number of messages travelling in the network. The network consists of network segments on which the nodes reside. Using the traditional approach, the management station should periodically ping each and every node in the network to check it's availability. So for a particular time period and network consisting of n network segments with N nodes per each network segment where n is very small compared to N, the minimum number of messages involved in the network will be $n \times N \times 2$.

Using the SNMP approach, the management station should have to poll periodically only the probe, which is one per each network segment. The failure of other nodes in the network is reported by probes to the management station. The management station polls the probes to check the entries in the

log table of the MIB maintained in the probe. So for a particular time period and network consisting of n network segments with N nodes per each network segment where n is very small compared to N, the minimum number of messages involved in the network will be $n \times 2$. If the heartbeats from the probe are taken into account, then the minimum number of messages will be $(n \times 2)$+n.

Example
Consider a network consisting of 5 network segments with 10 nodes per each network segment. The total number of nodes in the network is equal to $5 \times 10 = 50$. If the traditional approach is used, then the management station should poll 50 nodes periodically. The minimum number of messages for one time-period of polling is 50×2=100. If the SNMP approach is used, then there will be 5 probes in the network (one per each network segment). The management station should periodically check for occurrence of events in the log table of MIB maintained at the probe. The management station polls periodically only the probes in the network and so the minimum number of messages for a particular time-period is 5×2=10. If the heartbeats from the probe are taken into account, then the minimum number of messages are (5×2)+5=15. Thus for a particular time-period the minimum number of messages in the network using traditional approach are 100 and that of SNMP approach are 15.

If all the nodes want to monitor all the other nodes then the number of messages involved in the network with the traditional approach are n $\times N \times [(n \times N) - 1] \times 2$ and that of SNMP approach are $((n \times N) - n) \times n \times 2) + [n \times ((n - 1) \times 2]$.

Example
Consider the network with 5 network segments and 10 nodes per segment.(i.e. n=5 and N=10. then the number of messages involved in the network by the traditional approach are $5 \times 10 \times [(5 \times 10) - 1] \times 2 = 4900$.The number of messages involved in the network by the SNMP approach is $((5 \times 10) - 5) \times 5 \times 2) + [5 \times ((5 - 1) \times 2] = 490$.

Thus the numbers of messages involved in the SNMP approach are less than traditional approach.

7.1.2 Performance and Scalability

Performance is related to how fast the failure detection is done. In the traditional approach, as the number of nodes in the network increases, the

time-period over which the management station should poll all the nodes in the network decreases. Thus the management station will always be busy polling nodes almost all the time and it may happen that it will not be able to poll all nodes in the network during a particular time-period. If a node that is not polled by the management station fails, it's failure is detected by the management station only when it next polls that node.

In the SNMP approach, as the management station polls periodically only the probes, it will be able to poll all probes in the same network in which it was not possible to poll all nodes using the traditional approach. More over, when a node fails, the probe logs the event and will send a trap at the instant the node fails. So the management station in this approach knows about the failure very fast compared to traditional approach. The probe helps in remote monitoring of networks by performing proactive monitoring, offline operation.

In the worst case, when the trap and SNMP requests and responses are lost in the network, the time taken to detect the failure is equal to the time taken until a trap or SNMP response reaches the management station.

As the SNMP approach offers better performance and consumes less network bandwidth than the traditional approach, it is scalable.

7.1.3 Heterogeneity

The traditional approach does not offer heterogeneity.

The SNMP approach offers heterogeneity because of the following:

1. Each probe can be configured by multiple management stations. This makes the repository of data that is available at the probe to be shared by multiple management stations.

2. The SNMP supports multivendor equipments and so new nodes can be added to the network with out changing the specification of SNMP.

7.1.4 Interoperability

SNMP has been through many versions namely SNMPv1, SNMPv2 and SNMPv3. The SNMP offers interoperability because of proxy agents and bilingual network-management systems. Bilingual network management systems

support both SNMPv1 and SNMPv2. The management station communicates with the agent through the version of SNMP supported by the agent.

7.2 My opinion about the project

I was new to the concept of SNMP, failure detection and networking concepts. This project has given me depth in the knowledge about the working and utilization of SNMP and also made me understand the concept of failure detection, one of the key goals of distributed systems and failure detectors.

I feel that SNMP approach can effectively detect failures in a network in which it was not possible to detect the failure using ping effectively. The SNMP failure detection approach can be scalable to certain extent. But managing very large networks again possesses the problems related to network bandwidth because SNMP uses polling.

I hope that this report will be helpful to the Operating Systems Laboratory and will serve as a reference to future projects related to failure detection and SNMP.

Bibliography

[1] Stallings, W. *SNMP, SNMPv2, and RMON : Practical Network Management*, 2nd ed., Reading, MA: Addison-Wesley, 1996.

[2] Feit; Sidnie. *SNMP : A Guide To Network Management*, McGraw-Hill, 1995.

[3] Leinwand, A. and Conroy, K.,F.*Network Management: A Practical Perspective*, 2nd ed., Addision-Wesley, 1996.

[4] Rose, M.T. and MeCloghrie, K. *How to manage your network using SNMP.* New Jersey: Prentice-Hall, 1995.

[5] Wu; Jie. *Distributed System Design*, CRC press, 1998.

[6] Stallings,W. *SNMP and SNMPv2: the infrastructure for network management*, IEEE Communications Magazine, volume: 36 3, March 1998, Page(s):37-43.

[7] *An Overview of SNMP*, DDRI, Diversified Data Resources, 1999. available at http://www.ddri.com/Doc/SNMP_Overview.html.

[8] Parker, T., *Managing and trouble shooting TCP/IP.* SunSITE India Virtual Library, Publication Date: 1996. available at http:// sunsite.iisc.ernet.in /collection/virlib/ tcpip/parker/ tyt13fi.htm#E66E13.

[9] *Network Management.* AIX Version 4.3 Documentation. Communication programming concepts, first edition, october1997. available at http://nim.cit.cornell.edu/ doc_link/en_US/ a_doc_lib/ aixprggd/ progcomc/ ch7_netmgmt.htm

[10] *Network Management.* AIX Version 4.3 System Management Guide: Communications and Networks, second edition, October 1998. Available at http://nim.cit.cornell.edu /doc_link/en_US /a_doc_lib/ aixbman/commadmn/ ch9_snmp.htm

[11] CyberManage, *"A White Paper on Network Management"*. available at http://cybermanage.Wipro.com/wpaper-nm.htm

[12] Krupczak , C. and J.Saperia , *"Definition of system-Level Managed Objects for Applications"*, RFC 2287, Empire Technologies, BGS Systems, February 1998.

[13] Waldbusser, S. "*Remote Network Monitoring Management Information Base*", RFC 2819, STD 59, Lucent Technologies, May 2000.

[14] Waldbusser, S. "*Remote Network Monitoring Management Information Base version 2 using SMIv2*", RFC 2021, INS, January 1997.

[15] Rose M., and K. McClogherie, *Management Information Base for Network Management of TCP/IP-based internets*: MIB-II, RFC 1213, Performance Systems International, Hughes LAN Systems, March 1991.

[16] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB using SMIv2", RFC 2233, November 1997.

[17] Stelling, P.; Foster, I.; Kesselman, C.; Lee, C.; Von Laszewski, G. *A Fault detection service for wide area distributed computations.* High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on, 1998, Page(s): 268 -278.

[18] Felber, P.; Defago, X.; Guerraoui, R.; Oser, P. *Failure detectors as first class objects.* Distributed Objects and Applications, 1999. Proceedings of the International Symposium on, 1999, Page(s): 132 -141.

[19] Chandra T.D. and Toueg S., *Unreliable failure detectors for reliable distributed systems.* Journal of the ACM, 43(2), pp: 225–26, (March 1996).