An SNMP based failure detection service

Matthias Wiesmann, Péter Urbán, and Xavier Défago

School of Information Science,
Japan Advanced Institute of Science and Technology (JAIST)

February 1, 2006

IS-RR-2006-001

Japan Advanced Institute of Science and Technology (JAIST)

School of Information Science
1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan

http://www.jaist.ac.jp/

ISSN 0918-7553

An SNMP based failure detection service

Matthias Wiesmann*, Péter Urbán, Xavier Défago
Japanese Advanced Institute of Science and Technology

February 1, 2006

Abstract

In this paper, we present the SNMP-FD system. This system is a novel failure detection service entirely based on the SNMP standard. The advantage of this approach is better interoperability, and the possibility to rely on different sources of information for failure detection, including network equipment. This, in turn, gives us more precise failure information. This paper presents the architecture of the SNMP-FD system and discusses its advantages, both from the system engineering and interoperability angle, and the quality of the resulting failure detection service.

Keywords: failure detection, SNMP, distributed systems, interoperability, fault tolerance.

1 Introduction

One definition of distributed systems was given by Leslie Lamport: *You know you have one when the crash of a computer you've never heard of stops you from getting any work done.* It illustrates one of the main issues in distributed systems: detecting and handling failures. In order to implement some form of fault tolerance in a distributed system, one needs a mechanism to detect remotely when hosts and processes fail. As the number of nodes and the complexity of distributed system increases, the need for standard means for detecting such failures becomes a necessity.

While failure detection is an important aspect for implementing distributed algorithms, the problem of detecting failed processes or nodes is more general. It affects cluster and network management, application deployment and distributed computing in the wide sense. Currently, solutions for process and host failure detection exist, but they tend to be restricted to system and network management usage. Application level failure detection is typically handled using fixed timeouts, if it is handled at all.

Fixed timeouts are ill-suited for failure detection. Consider the case of a web-browser issuing an HTTP request. A connection timeout can mean either that the server has failed or that it is simply overloaded.

*Swiss National Science Foundation Fellowship PA002-104979

This means that the user has to reload the page to figure out what happened, which is neither practical, nor reliable or even efficient—in fact, in case of an overloaded server, a retry will only add to the load. Setting the right timeout is a difficult task: if it is too short, the connection will fail before getting the answer from the remote site and, if it is too long, the connection will block, waiting needlessly for a crashed machine. As there is no standard means to detect if a host or an application is crashed, timeouts are, alas, often the only possible solution.

A failure detection service addresses these issues. Applications can use it to decide when a process has crashed and when reconfiguration is needed. The service can be used directly by applications, but also be integrated in the communication mechanisms of middleware systems, like CORBA or a grid infrastructure, or be used to build failure detectors that offer formal properties like $\diamond S$ [CT96], Ω [Lam98] or accrual failure detectors [DUHK05].

There are many advantages in having a failure detection service. First, a service factors out common functionalities. It is easier to implement complex, adaptive failure detection as an independent service and thus offer formal properties. Such a service can aggregate multiple information sources and use all the available information. Second, it gives a uniform interface to complex functionality. Third, the actual implementation and model of the underlying system can be changed without rewriting applications and high-level protocols. Fourth, a failure detection service can be configured independently of the applications.

A failure detection service has many advantages and there have been a few implementations, yet none of those have gained wide acceptance. Existing implementations have generally been ad-hoc implementations, with little regard for standardized interfaces or interactions with other services and standards. Many issues can prevent a distributed service from gaining acceptance, but providing no standard interface and no provision for interactions with existing services and infrastructure is certainly a factor.

We believe that failure detection is now mature enough to be standardized. In this paper, we present an architecture for a standard based failure detection service. This architecture is based on the SNMP standard [CFSD90]. The use of this standard gives us four advantages: 1) the SNMP protocol is designed to be lightweight; 2) SNMP makes it possible to inter-operate with network equipment and other management services; 3) using an existing standard prevents us from re-inventing the wheel and increases the prospect of wide adoption [WDS03]; 4) using a standard designed for monitoring makes it possible to use the abundance of existing tools and thus build a high quality detection service.

Our SNMP-FD framework offers low-level failure detection services and specifies interfaces to be used for application level failure reporting. Failure detection components can use a variety of strategies to detect failures and report such failures with different quality of service parameters and formal properties. In turn, these failure detection services can be used directly by applications, be integrated in middlewares, or be used for building distributed system protocols, like consensus, total order broadcast, or atomic commitment.

SNMP helps us in many ways: we use SNMP messages to transport heartbeats and failure reports; we use an SNMP interface to query and configure the failure detection service; and the failure detection service can use SNMP accessible information offered by other services, like network equipment. This makes it possible to rely less on heartbeat messages and improve the quality of failure detection.

Finally, we get all the advantages of using a standardized middleware. First, this means that different implementations of the failure detection service can coexist and share the same interface. This might be

needed because nodes may run in different administrative domains with a different infrastructure, or because there may be multiple versions of the failure detection service to avoid correlated bugs. Moreover, issues like external monitoring and security become important aspects if the distributed system is to run in a wide area network (WAN). Using SNMP allows us to reuse its security facilities and wide array of tools.

The remainder of this paper is structured as follows. Section 2 presents failure detection and the system model. Section 3 introduces the SNMP protocol. Section 4 describes the different SNMP mechanisms that can be used to build a failure detection service. Section 5 introduces the SNMP-FD framework architecture. Section 6 discusses the failure model and applications of the framework. Section 7 presents the related work. Section 8 concludes the paper and talks about future work. Appendix I contains the MIB description of the service.

2 Failure detection

Failure detection has always been an issue in distributed systems. Early implementations of failure detection have been proposed by Ricciardi et al. [RB91] and Beker [Bec91]. Conceptually, a failure detection service is a distributed *oracle* that gives you information about the state of processes in a distributed system. This oracle can be used to determine when a point-to-point communication should be aborted (typically, stop waiting for data from the other endpoint). In distributed algorithms, this oracle is used to trigger new phases and reconfigurations. For a general understanding of failure detectors, see the introductions on the subject by Gärtner [Gär01] and Raynal [Ray05]. Different failure detection models and implementations are described in Section 3.

While distributed systems concentrate on *processes*, monitoring systems typically monitor *resources*. A resource can be anything: a cooling fan, a single executing process, a host machine, or a whole computing rack. There is a relationship between resource failures. Depending on the amount of redundancy and the type of failure, one failure can cause the failure of another element: if a host crashes, all processes running on this host will also crash. Monitoring tools handle the failure relationship between resources. A failure detection service only monitors processes. Therefore this paper concentrates on processes and the resources whose failure can directly affect processes: hosts and network links.

We assume a set of hosts, each running a set of processes. Processes are the basic unit of monitoring. The failure detection service provides failure information about processes. We assume that the service is running within a Local Area Network (LAN); failure detection in Wide Area Networks (WAN) is beyond the scope of this paper. The failure detection delivers notifications each time a process is suspected to have failed (S-transition) or is trusted again not to have failed (T-transition). Attached to these notifications is some optional information about the level of suspicion. We concentrate on this failure detection interface because it is the most common, and it is also applicable in point-to-point communication.

Failure detection in a LAN is either done in a centralised way, with one management station monitoring all hosts and processes, or in a flat structure, where all processes monitor all other processes. There are two kinds of monitoring. With the first kind, also called *ping* (after the ICMP ping protocol [Pos81]), the monitoring host sends *are you alive* requests to the monitored process and expects a response. The lack of response leads to a suspicion. The second approach is to have the observed process send *heartbeat* messages (also called *I'm alive* messages) to the observer. The lack of *heartbeat* messages leads to a suspicion.

The characteristics of a LAN are as follows. Hosts are connected with high-bandwidth, direct links. There is little or no IP-level routing between the nodes and broadcast primitives like UDP-multicast can be used. Network latency is low, and packet loss is rare. Hosts can potentially interact with the network equipment that connects them.

3 SNMP

The Simple Network Management Protocol (SNMP) is a standard for network management activities [CFSD90] defined by the Internet Engineering Task Force (IETF). SNMP is the most common standard for managing network equipment and critical servers. SNMP agents can be found in all the higher-range equipment, and the current trend is to put SNMP agent in low-cost devices: nowadays, one can find SNMP agents in many network attached devices like Wi-Fi access points, ADSL routers, small smart switches, and even printers. We think this trend will continue. Open-source agent software and SNMP tools for personal computers and servers are available for most operating systems [NSN].

Many application specific extensions to SNMP have been proposed, from network performance monitoring [Wal95] to database management [BPD+94]. The Java Management toolkit specifies an SNMP interface [JMX99]. Other vendors provide tools to translate low level monitoring information like IPMI into SNMP [Vol01]. Thus, various aspects of a complete distributed system, from software components to power-supply voltage, can be monitored using SNMP. Many high-level diagnosis and monitoring tools rely on SNMP information for network and host management.

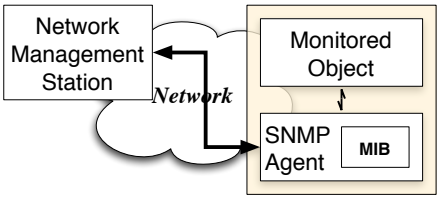

Figure 1: SNMP Architecture

Figure 1 illustrates the SNMP architecture. The Network Management Station (NMS), on the left, monitors some managed object on the right, by the way of an SNMP agent. The SNMP agent hosts a Management Information Base (MIB). The MIB can be queried and written by the NMS using synchronous queries. The agent can initiate communication with the NMS by sending messages. These messages can either be *traps* (which are not acknowledged), or *inform* messages (which are acknowledged). The actual type of message can be configured. SNMP specifies security and authentification. SNMP typically runs over UDP/IP but it can also use TCP/IP or other network technologies. Data marshalling is done according to the Abstract Syntax Notation's Basic Encoding Rules (ASN.1-BER) defined by the ITU [X.602].

The MIB is a tree structured database that contains information reflecting the state and configuration of both the agent and the monitored object. The MIB tree can be divided in sub-trees, each concerning a particular aspect of management. In addition to the standard parts of the MIB tree, each vendor can have its own proprietary sub-tree. The MIB supports scalar integer types, strings, structures, and tables. The position of a data item in the tree is identified by a reference, called Object IDentifier (OID). The OID is a sequence

of numbers, usually denoted `x.y.z`. Many OIDs have an associated human-readable name; in this paper, such names are written in sans serif font. An SNMP binding is a pair formed by an OID and an associated value. SNMP messages contain a sequence of bindings. In other words, each message contains the value of certain points in the tree.

The content of the MIB and of SNMP messages are described using schema files written in the SMI language. Those files are used by SNMP tools to display information in a human readable format, but also to configure tools, and to help building agents. The service specification given in Appendix I is written in the SMI language.

While SNMP was originally designed with a single NMS in mind, fault tolerance and scalability has been increasingly researched. A system that increase the scalability of SNMP monitoring system is presented in [SAF00]. In [DMNN98], techniques to make SNMP reporting fault tolerant are described.

4 Using SNMP

4.1 Goal

Our goal is not to implement one particular failure detector (for a description of different failure detector types and interfaces, see Section 7). Instead, we wish to offer common functionality needed to implement a failure detector and means to connect this failure detector with applications. For simplicity, we concentrate on one particular failure detection interface but the same technique can be used for other interfaces. With this interface, the application is simply told that a given process is suspected with some optional information giving some insight on the level of suspicion. We choose this interface mostly because it is the most common, and it can be used with point-to-point communication. Other interfaces, like leader election, are defined in terms of many processes, and hence cannot be applied to point-to-point communication.

The standard approach to build a complex distributed system is to rely on some kind of middleware. Middlewares have two advantages: they offer some abstraction level, and a standard for interoperability. Both aspects are relevant for a distributed failure detection service, although interoperability is more of a concern to us. There is an abundance of available middlewares: Java-RMI, CORBA, DCOM, JMS, SOAP, etc. SNMP is not traditionally considered a middleware, yet it offers all the basic facilities: it specifies client-server interactions, message format, data marshalling, standard services, and authentication.

Given our goal, SNMP is a clear choice, as the standard is designed for network monitoring, it is widely deployed, and was designed to be lightweight so as to be embedded in network equipment. In this section, we describe how SNMP's features can be used for the task of building a failure detection service. This includes the following functionalities: heartbeat messages, process information, failure detection configuration, failure reporting, and cooperation with networking equipment.

4.2 Messaging

SNMP specifies a mechanism for transmitting messages, as traps and inform messages. SNMP specifies the serialisation format and the message structure. A failure detection service will have to transmit messages. Those messages can be both internal (for instance, heartbeat messages), or to notify client applications (report changes in the suspicion level). By using the SNMP message format, messages can be parsed and understood by existing tools. This also means that client applications can use any existing SNMP handling library and is not tied to one specific implementation or language. One advantage of using SNMP messages over using the messaging structure of another middleware like CORBA is that the message format is lightweight (typically one UDP packet) and can be used in a connection-less mode.

4.3 Process information

SNMP specifies a standard MIB, called the *host resource* MIB [WG00] to expose information about resources on a host. Resources that can be monitored via this interface include devices, file systems, installed software, and running processes. The last item is of interest for a failure detection service. Using this MIB to expose information about whether local processes are running or not is a natural choice. If it uses this MIB, the failure detection system can be used hand in hand with classical network and cluster management systems.

4.4 Configuration interface

The MIB is the SNMP mechanism for querying and setting the configuration of the agent. The Notification and the Target MIB [LMS99] specify destinations of SNMP messages and how they should be addressed. Other aspects of the service can be described in application specific MIBs.

4.5 Cooperation with networking equipment

The SNMP agents on networking equipment are configured to send traps in case of events that have an influence on the failure detection service. Two traps are of particular interest: link-up and link-down. Link-down indicates that a network interface went down; the host behind this interface is therefore unreachable. Link-up indicates that the network interface went up. In case of short disconnections—for instance, the rebooting of a machine—the down trap might not be generated. Other traps, for instance those signaling congestion, could be used to adjust failure detection parameters [dAMeL04].

5 Failure detection service based on SNMP

This section describes the architecture of our SNMP-based failure detection service. We call it SNMP-FD.

We assume a set of hosts, running both *monitored* or *monitoring* processes. Monitoring processes want to get monitoring information from monitored processes. A process can be both monitoring and monitored.

A dæmon process is running on each host. This dæmon acts as a local agent for both monitoring and monitored processes. Monitored and monitoring processes are registered with the local dæmon. Dæmons are responsible for exchanging monitoring information. We assume that in case of crash of the dæmon, it is restarted automatically by the operating system.

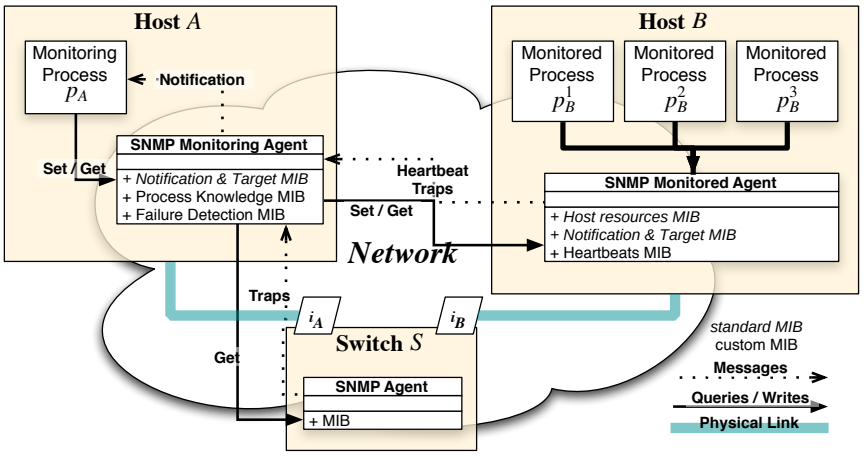

Figure 2: SNMP Usage

Figure 2 illustrates the general architecture of SNMP-FD. It shows two hosts A and B. Host A contains one process p_A that monitors three processes on host B: p_B^1, p_B^2 and p_B^3. The network between hosts A and B runs on smart switch S that exports management information using a SNMP interface. On smart switch S, host A is connected on interface i_A and host B is connected on interface i_B. Dæmons d_A and d_B run on nodes A and B, respectively. Both act as SNMP agents. Dæmon d_A implements the monitoring agent interface, while dæmon d_B implements the monitored agent interface. In an actual system, dæmons would implement both interfaces. All monitored processes $p_B^1 \ldots p_B^3$ are registered with dæmon d_B. Other parts of the figure are explained throughout the section.

5.1 Monitored agent

This agent monitors local processes and notifies remote agents in case of crashes. The agent implements three MIBs: the Host Resource MIB, the Notification MIB and the Heartbeat MIB. We assume that the agent is monitored by some operating system facility. In case of failure, the agent is automatically restarted. The agent has a copy of its internal structures in stable storage. Thus, we assume that d_B is running as long as host B is up. Upon restart, the agent follows the SNMP convention and sends a *cold-start* message to all the targets registered in the Notification MIB. The monitored agent sends two types of messages to other agents: heartbeats and messages signaling a change of process state.

Host Resource MIB The agent implements the part of the standard Host Resource MIB [WG00] that describes running processes. The system can export all running processes of the host or only the monitored processes. Processes are stored into the SNMP table called hrSWRunTable (OID .1.3.6.1.2.1.25.4.2). Of particular interest is the field named hrSWRunStatus that contains the status of the process. The agent

implements the part of the standard Host Resource MIB [WG00] that describes running processes. The system can export all running processes of the host or only the monitored processes. Processes are stored into SNMP table called hrSWRunTable (OID .1.3.6.1.2.1.25.4.2). The agent implements the portion of the Host Resource MIB that contains the process table.

Notification and Target MIB SNMP specifies the Notification MIB and the Target MIB [LMS99] to handle registration of remote SNMP entities interested in receiving messages if the state of the agent or its associated device changes. In particular, those MIBs specify tables that contain the addresses and parameters for each remote listener. The agent implements the standard Notification and Target MIB. Those MIBs contain the addresses and parameters of all entities that need to be notified in case of process state change.

For the case described in Figure 2, the Notification and Target MIB contains one entry with the address of A and the port number used by the SNMP Agent running on A, along with the relevant SNMP options (addresses, protocol used, SNMP version, type of message, filtering options, etc.).

Heartbeat MIB The agent also maintains a MIB that contains the descriptions of the different heartbeats in the form of one table. We need an additional table as heartbeat parameters do not fit in the notification and target tables. Each entry in this table contains the tag of one entry in the Notification MIB to send the heartbeat to, the process identifier of the monitored process, and information about the heartbeats themselves: time interval between heartbeats and a counter of the number of sent heartbeats. In the case illustrated by Figure 2, the heartbeat table contains three entries: for p_B^1, for p_B^2 and for p_B^3. This MIB is specific to the SNMP-FD framework and is specified in Appendix I.

Heartbeat messages Heartbeat messages can be either traps or inform messages. They contain a sequence of bindings from the different MIB of the monitored agent. The following bindings are included: the host identifier of the monitored agent SNMPv2-MIB::sysName (OID 1.3.6.1.2.1.1.5.0), the interval between heartbeats, then for each monitored process the process identifier, the process state (from the Host resource MIB), the requested heartbeat interval (which might be larger than the actual interval between heartbeats) and the heartbeat counter (both from the Heartbeat MIB).

5.2 Network equipment

The SNMP agent on smart switch S is configured by the way of its Notification and Target MIBs to send traps to the agent on host A in case of events. For instance if interface i_B goes up or down. A link-down trap concerning interface i_B indicates the host B is unreachable on the network. An link-up trap without a down trap before indicates a reboot – in this case, the agent on host A should query the demon on host B to see what its status is.

5.3 Monitoring agent

The monitoring agent A maintains four MIBs: the Notification MIB, the Target MIB, the Process Knowledge MIB, and the Failure Detection MIB. Each different failure detection policy will result in a different table in the Failure Detection MIB.

Notification and Target MIBs Like the monitored Agent (see Section 5.1), the Monitoring Agent implements the Notification and the Target MIB. Those MIBs contain the information for sending traps or notifications to remote hosts. All the entities known to the agent are registered here.

Process Knowledge MIB This MIB contains monitoring information for remote processes. This information is stored in a table, where each line is a remote process that the agent knows about. For each process, it contains the host name, the process identifier, the last known status (hrSWRunStatus), the state of the associated host and the time-stamp of the last update. This information is updated dynamically as SNMP messages arrive. This MIB is specific to SNMP-FD and specified in Appendix I.

Failure Detection MIB This MIB contains a table of failure detectors. Each failure detector is implemented as a different table, but they all share some common structure. Each entry of this table describes a remote process identified by a host name and a process identity, it also contains a reference to one local process knowledge table entry that describes the remote process, and the related heartbeat rate. Each line of the table also contains all the parameters related to the failure detection. Those columns of the table are specific to one failure detector. For example, a failure detector that implements a simple timeout-based suspicion scheme, will implement this MIB with a table where each entry will contain the status (suspected or not) and the timeout after which the lack of heartbeat will trigger a suspicion. This MIB is specific to each policy implemented within the SNMP-FD system. An example of a simple heartbeat base failure detector is specified in Appendix I.

Application notification Applications are notified using local SNMP notifications. The suspicion message contains the OID of the process's entry in the Failure Monitoring MIB, the binding of the process's status in the Failure Detection MIB (e.g, the OID and the value). The application can use the failure information in the message, or use the OIDs included in the message to lookup more complete information in the MIBs of the Monitoring Agent or even the host where the process runs.

5.4 Comparison to other services

This architecture follows that of other failure detection services like the Globus Heartbeat Monitor [SDF+99]. There are many similarities with the Hierarchical Service presented by Bertier and al. [BMS03]: the Process Knowledge MIB is similar to the *blackboard* element in the Basic Layer, and Failure Detection MIBs correspond to adaptation layers.

6 Failure detection types

In this section, we discuss the avantages of the SNMP-FD architecture for failure detection and the failure coverage.

We consider a system composed of hosts, processes, network links and network equipments. Hosts can have the following failures: operating system freeze (the hardware stays active, but the hosts and the processes running on it stop responding), operating system crash (the hosts reboots, all processes running on it are terminated), hardware stops (the hardware shuts down). Process failures are fail-stop and are detected by the operating system, that is, we consider that the operating system has means to detect other process failure types, like freezes [SS83]. We consider that the SNMP-FD agent can fail, but will be restarted by the operating system. Network links can fail (stop transmitting data). The network system can lose packets, but it is fair lossy, i.e., eventually messages will be transmitted. Network equipment can stop-crash (i.e physically stop from working), or freeze (stop transmitting data). We exclude malicious (byzantine) failures.

6.1 Available failure detection information

Cause	**Detection mechanism**	**Monitor**	**Notification**
Process stop-crash	OS	Monitored Agent	Process state trap
Agent stop-crash		OS	Agent cold start trap
Link failure	Link state	Network equipment	Link down trap
Host stop			
Host reboot			Link up trap
Equipment stop-crash			Link down trap

Table 1: Detection types

Many failures can be detected by components of the system. Table 1 summarizes the types of failures that can be detected and thus reported to the failure detection system.

Process failures Process stop-crashes can be communicated with no significant delay — the notification is triggered as soon as the host's operating system detects the crash. The crash of the monitored agent is detected indirectly. The crash is detected by the operating system and the monitored agent is restarted. Upon restart, the agent will send a cold start trap (as specified by the SNMP standard). The agent's data structures are assumed to be stored on stable storage.

Link crash failures Link crash failures mean that the network link goes physically down. This can be detected by the hardware of network equipment and are reported by link-down traps. Interpretation of those traps depends on the system's structure and the number of links to a given host. The processes on a given host should only be suspected when the host's last link fails. The loss of a link can affect some parameters of a failure detection scheme. For instance, if the failure detector reports a suspicion level for a host, this level would be increased when the host is connected to the network with a reduced set of links.

Host failures Host fail-stops also cause the network link to go down. A host reboot causes the network link to go down for a short amount of time and then go up again. This is often only signaled by the equipment by a link-up trap with no previous link-down. Detection time for changes in the link state are quite long: the traps are only generated after roughly 1 second. The hardware does its own transient suppression and thus waits for the link state to stabilise. Depending on the equipment, the agent can be configured to send notifications instead of traps and do retries to ensure the notifications are reliably delivered.

It is not possible to distinguish the crash of a single host from the failure of all its links: both are signaled by link-down traps. Both cases indicate that further messages from the host should be expected. Link-down traps can be correlated with ICMP Host unreachable packets [WNSS02] which also signal that a host is unreachable.

Network failure Fail-stops of network equipment, like host fail-stops are detected by the link connecting to them going down. It is also not possible to distinguish between the crash of some equipment and the failure of all of its links. While such failures should be handled by the networking infrastructure and do not concern directly the failure detection service, those failure could affect some failure detection parameters. For instance the loss of some of the network connectivity could mean that heartbeat parameters would be changed.

6.2 Proactive failure detection

The failures cases described in the previous section are very interesting because detecting them does not require any specific activity of the failure detection service: it must simply register for these notifications and react to them. When no failure occur, no processing and no message transmission is needed.[1] We call this *reactive* failure detection.

The main problem is that not all failures can be detected using *reactive* failure detection. For instance, an operating system freeze will not trigger any special signal, as the network link will stay up. Also while certain failures can be detected reactively, the detection time can be quite long. Because of this, we cannot rely solely on reactive schemes: we also have to use heartbeats. In that case, the failure detection system does not wait for external signals; it generates a data stream instead, and measures it. This uses both processing and network resources. We call this *proactive* failure detection. The quality of service of proactive failure detection is correlated with the amount of monitoring the traffic generated [CTA02].

The two approaches are not exclusive, and SNMP-FD is designed to support both.

6.3 Advantages of reactive failure detection

The advantage of reactive failure detection is that the detection time can be short. If the failure is detected immediately, the time until it is reported is basically the time for one message to go from the monitored

[1] This information might not be available via a push interface, so local polling might be needed.

host to the monitoring host. The detection time is constant. If you assume that reactive failure case are frequent then integrating reactive detection schemes in the failure detection service can improve your *average* detection time (it will not change the maximum detection time, though).

Let us consider the case illustrated in Figure 2, host A monitors processes on host B. The dæmon on host B sends heartbeat message to A every η time units. We consider a simple timeout based failure detector with the following model assumptions: the system has synchronized clocks, the average time for transmitting a message from B to A is δ, the maximum transmission time is $\delta + \epsilon$, and the failure detector on host A suspects B if it has not received a heartbeat at time $\delta + \epsilon$. Messages are never lost. In this system, the average detection time will be: $\bar{T_D} = \delta + \epsilon + \frac{\eta}{2}$.

If we consider P the probability that a given failure can be detected immediately, then the average detection time now will be: $\bar{T_D} = \delta + (1 - P)(\epsilon + \frac{\eta}{2})$. As P increases, the contribution of the heartbeat parameters becomes small, and the average detection time depends less on the the heartbeat period.

Beside crashes where the monitored process crashes and the rest of the system is fine, other failure cases can be detected without ambiguity. For instance, if a machine shuts down because its UPS is nearly empty. Another example is system reboot (by the way of link up traps and ICMP unreachable packets). In general, software failures are quite common [OGP03, Gra85].

6.4 Suspicions and certitudes

The basic interface of the failure detector returns two values: trusted or suspected. In the SNMP-FD there are cases where another value is returned: failure is not a suspicion but a certitude. For instance, if a process crashes but the host and the SNMP-FD agent are correct, this failure can be reported without any ambiguity. In fact, getting more accurate information is the main advantage of interacting with more entities on the network. Does this bring any advantage? What are the advantages of having a failure detector with three states trusted, suspected, failed?

A three state failure detector makes it possible to write algorithms with better characteristics. Take for instance the problem of consensus in an asynchronous system with unreliable failure detectors [CT96]. Consensus can only terminate if the there is a majority of correct processes, that is, a number of faulty processes $f < \frac{n}{2}$, now if we distinguish the number of suspected processes f_s from the proceses that are known for sure to have failed f_c, consensus can terminate if $f_s < \frac{n-f_c}{2}$.

Intuitively this can be explained in two ways. One is to consider that the system implements a simple view membership service [CKV01]. Processes are then safely excluded from the view when their failure is certain, so the algorithm will run with a view where all processes whose failure is certain have been excluded. Another way to see it is to consider the consensus algorithm for the crash-recovery model from Aguilera et al. [ACT00]. In this algorithm, when a node recovers, it simply sends a single message to say that it should be ignored in the next rounds of consensus. If process a is certain that process b is crashed, process a can behave as if b had crashed, recovered and sent this message. Thus from the point of view of the number of required processes to terminate, failure certitudes can be handled as recoveries.

A similar conclusion is reached by Gorender et al. [GMR05]. In their model, the system switches between

two models, one with a perfect failure detector and one with an imperfect one and uses this model to implement a three state failure detector, with similar results.

7 Related work

Chandra et al. have shown that an abstract failure detection service [CT96] with certain properties can solve the consensus and atomic broadcast problem in an asynchronous system [FLP85]. The same failure detectors can solve total order broadcast or atomic commitment problem [GLS95]. This failure detector returns a list of suspected processes. Other failure detection interfaces have been proposed: returning a leader [Lam98], giving an indication of slowness [SBCdF03], a suspicion level [HDYK04]. In [BO83], an oracle returning a random value is used instead of a failure detector. In this paper, we concentrated on the first kind of interface.

For each interface type, there have been numerous implementations and algorithms. Some refine the suspicion model [GS96, GM98], avoid the use of timeouts [ACT97], use a query-response based implementation [MMR03] or handle network partitions [DFKM97]. In [CTA02], Chen et al. propose a quantification of quality of service for failure detectors. Many techniques have been proposed to improve the quality of service of failure detectors: by minimizing traffic [LFA00], masking transient errors [BCG97], or by using some means of adaptation [CRV95, FRT01, NJP04]. Failure detectors that combine multiple approaches have also been proposed [AT99, MPR04]. In [dAMeL04], failure detection is implemented by the way of a neural network. This neural network uses information extracted from network equipment as one of its inputs. This information is extracted using SNMP, but the protocol is not used for other tasks. To the extent that they are used to build a failure detector that tells if a process is suspected or not, all these techniques can be used within our architecture.

The concept of a failure detection service as a separate entity has been proposed for some time [FDGO99]. Most distributed system toolkits feature a failure detection facility [Mur05], or a group membership service that excludes failed processes [KSMD02]. A standalone failure detection service have been proposed in both the distributed system community [DHJ+04] and the Grid community [SDF+99, WSGY00]. A system to adapt timeouts in Fault-tolerant CORBA is presented in [SM01]. Those services are different from the one presented here in two ways: first, they do not support standard interfaces; second, they offer high-level failure detection information. A first, incomplete and partially SNMP compliant prototype of the architecture in this paper was described in [Rei02]. The general architecture follows that of other failure detection services [SDF+99, BMS02].

The scalability of failure detection has also been studied. Two general types of wide area failure detectors have been proposed [HCK02]: hierarchy based [SFK+98, BMS03] and gossip based [vRMH98, GCG01]. Evaluations of wide area network failure detectors are presented in [BMS03, FB05]. The techniques described in those paper are all applicable to the system presented here, but due to space constraints they are not described here.

Tools to monitor processes for system administrations purposes have been around for a long time. Systems like Pulsar [Fin97] or Nagios [Kre03] include modules to monitor processes. Management systems like HP's Open View or Sun's Sun Management Center [MS01] integrate tools for process and host monitoring. Both tools rely on SNMP. While those tool can also monitor processes, this is not the focus of such systems.

They do not export a failure detection interface to be used by applications. The SNMP system has been designed for monitoring tasks and is used in many domains related to failure detection. A framework for application level recovery based on SNMP diagnosis is presented in [SIK02]. Distributed fault diagnosis based on SNMP is described in [STD02].

8 Conclusion and future work

We have described the architecture and design of a failure detection service based on the SNMP standard. This design shows that this standard can be adapted to fit the task of a failure detection service. While a few new Management Information Bases need to be defined, many existing, standardized interfaces can be used. By using an existing standard, we ensure interoperability with existing management systems and tools. A standard interface makes it possible for multiple implementations of the service to coexist and cooperate.

The SNMP-FD system offers a failure detection service, with lightweight heartbeat messages and process status and configuration information exposed using the standard Management Information Base interface mechanism. The service is designed for interoperability from the ground up and can interact directly with existing SNMP enabled network equipment. The system can be extended for WAN failure detection.

Future work includes the implementation of failure detection algorithms described in the literature within SNMP-FD, the integration of information gathered from SNMP devices into those algorithm, and the integration of SNMP-FD interfaces into existing group-communication toolkits. Further work on the use of the three state failure detector is also planned.

References

[ACT97] M. K. Aguilera, W. Chen, and S. Toueg. Heartbeat: A timeout-free failure detector for quiescent reliable communication. In *Workshop on Distributed Algorithms*, pages 126–140, 1997.

[ACT00] M. K. Aguilera, W. Chen, and S. Toueg. Failure detection and consensus in the crash recovery model. *Distributed Computing*, 13(2):99–125, 2000.

[AT99] M. K. Aguilera and S. Toueg. Failure detection and randomization: A hybrid approach to solve consensus. *SIAM Journal on Computing*, 28(3):890–903, February 1999.

[BCG97] S. Bondavalli, S. Chiaradonna, and F. Di and Giandomenico. Discriminating fault rate and persistency to improve fault treatment. In *Proceedings of the 27th International Symposium on Fault-Tolerant Computing (FTCS '97)*, pages 354–362. IEEE, 1997.

[Bec91] T. Becker. Keeping processes under surveillance. In *Proceedings of the Tenth Symposium on Reliable Distributed Systems (SRDS)*, pages 198–205. IEEE Computer Society, September 1991.

[BMS02] M. Bertier, O. Marin, and P. Sens. Implementation and performance evaluation of an adaptable failure detector. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks (DSN)*, pages 354 – 363. IEEE Computer Society, 2002.

[BMS03] M. Bertier, O. Marin, and P. Sens. Performance analysis of a hierarchical failure detector. In *Proceedings of the International Conference on Dependable Systems and Networks*, San Francisco, CA, USA, June 2003.

[BO83] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the Second Annual Symposium on Principles of Distributed Computing*, pages 27–30, Montréal, Quebec, Canada, 1983. ACM.

[BPD+94] D. Browner, R. Purvy, A. Daniels, M Yinkin, and J. Smith. Relational database management (rdbms) system management information based (mib). RFC 1697, IETF, 1994.

[CFSD90] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A simple network management protocol (SNMP). RFC 1157, Internet Engineering Task Force (IETF), 1990.

[CKV01] G. V. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: A comprehensive study. *ACM Computing Surveys*, 4(33):1–43, 2001.

[CRV95] F. J. N. Cosquer, L. Rodrigues, and P. Veríssimo. Using tailored failure suspectors to support distributed cooperative applications. In *Proceedings of the 7th International Conference on Parallel and Distributed Computing and Systems*, Washington D.C, USA, October 1995.

[CT96] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Communications of the ACM*, 43(2):225–267, 1996.

[CTA02] W. Chen, S. Toueg, and K. M. Aguilera. On the quality of service of failure detectors. *IEEE Transactions on Computers*, 51(2):561–580, 2002.

[dAMeL04] R. de Araújo Macêdo and F. Ramon Lima e Lima. Improving the quality of service of failure detectors with SNMP and artificial neural networks. In *Anais do 22o. Simpósio Brasileiro de Redes de Computadores*, pages 583–586, Gramado, RS, Brazil, 2004.

[DFKM97] D. Dolev, R. Friedman, I. Keidar, and D. Malkhi. Failure detectors in omission failure environments. In *Proceeding of the Symposium on Principles of Distributed Computing (PODC)*, pages 286–302, September 1997.

[DHJ+04] J. Dunagan, N. J. A. Harvey, M. B. Jones, D. Kosti, M. Theimer, and A. Wolman. FUSE: Lightweight guaranteed distributed failure notification. In *Proceedings of the 6th Symp. on Operating Systems Design and Implementation (OSDI'04)*, pages 151–166, 2004.

[DMNN98] E. P. Duarte, G Mansfield, T. Nanya, and S. Noguchi. Improving the dependability of network management systems. *International Journal of Network Management*, 8(4):244–253, July–August 1998.

[DUHK05] X. Défago, P. Urbán, N. Hayashibara, and T. Katayama. Definition and specification of accrual failure detectors. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 206–215. IEEE/IFIP, 2005.

[FB05] L. Falai and A. Bondavalli. Experimental evaluation of the QoS of failure detectors on wide area network. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN2005)*, pages 624–633, Yokohama, Japan, june 2005. IEEE Computer Society.

[FDGO99] P. Felber, X. Défago, R. Guerraoui, and P. Oser. Failure detectors as first class objects. In *Proceedings of the International Symposium on Distributed Objects and Applications (DOA'99)*, pages 132–141, Edinburgh, Scotland, 1999.

[Fin97] R. A. Finkel. Pulsar: an extensible tool for monitoring large unix sites. In *Software: Practice and Experience*, volume 27-10, pages 1163–1176. Wiley Interscience, 1997.

[FLP85] M. H. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[FRT01] C. Fetzer, M. Raynal, and F. Tronel. An adaptive failure detection protocol. In *Proceedings of the 8th Pacific Rim Symposium on Dependable Computing (PRDC-8)*, pages 146–153, Seoul, Korea, December 2001. IEEE Computer Society.

[Gär01] F. C. Gärtner. A gentle introduction to failure detectors and related problems. Technical Report TUD-BS-2001-01, Darmstadt University of Technology, Department of Computer Science, 2001.

[GCG01] I. Gupta, T. D. Chandra, and G. Goldszmidt. On scalable and efficient distributed failure detectors. In *Proceedings of the 20th Annual Symposium on Principles of Distributed Systems of Distributed Computing*, pages 170–179. ACM, 2001.

[GLS95] R. Guerraoui, M. Larrea, and A. Schiper. Non blocking atomic commitment with an unreliable failure detector. In *Proceedings of the 14th Symposium on Reliable Distributed Systems (SRDS-14)*, pages 41 – 50, Bad Neuenahr, Germany, September 1995. IEEE Computer Society, IEEE Computer Society.

[GM98] V. K Garg and J. R. Mitchell. Implementable failure detectors in asynchronous systems. In *Proceedings of the 18th Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer-Verlag, 1998.

[GMR05] S. Gorender, R. Macêdo, and M. Raynal. A hybrid and adaptive model for fault-tolerant distributed computing. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN2005)*, pages 412–421, Yokohama, Japan, June 2005. IEEE Computer Society.

[Gra85] J. Gray. Why do computers stop and what can be done about it. Technical Report 85.7, Tandem Computers Inc., Cupertino, CA, USA, 1985.

[GS96] R. Guerraoui and A. Schiper. Gamma-accurate failure detectors. In Springer-Verlag, editor, *Proceedings of the 10th International Workshop on Distributed Algorithms (WDAG-10)*, LNCS 1151, Bologna, Italy, October 1996.

[HCK02] N. Hayashibara, A. Cherif, and T. Katayama. Failure detectors for large-scale distributed systems. In *Proceeding of the 1st Workshop on Self-Repairing and Self-Configurable Distributed Systems (RCDS), 21st IEEE Int'l Symp. on Reliable Distributed Systems (SRDS-21)*, pages 404–409, Osaka, Japan, 2002.

[HDYK04] N. Hayashibara, X. Défago, R. Yared, and T. Katayama. The phi accrual failure detector. In *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS'04)*, pages 66–78, Florianpolis, Brazil, October 2004. IEEE Computer Society.

[JMX99] Java management extensions. White Paper, June 1999.

[Kre03] J. Kretchmar. *Open Source Network Administration*. Computer Networking and Distributed Systems. Prentice Hall Professional Technical Reference, October 2003.

[KSMD02] I. Keidar, J. Sussman, K. Marzullo, and D. Dolev. Moshe: A group membership service for WANs. *ACM Transactions on Computer Systems*, 20(3):191–238, August 2002.

[Lam98] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.

[LFA00] M. Larrea, A. Fernandez, and S. Arevalo. Optimal implementation of the weakest failure detector for solving consensus. In *Proceedings of the 19th Symposium on Reliable Distributed Systems (SRDS'00)*, pages 52 – 59, Nürnberg, Germany, 2000. IEEE Computer Society.

[LMS99] D. Levi, P. Meyer, and B. Stewart. SNMP applications. RFC 2573, IETF, 1999.

[MMR03] A. Mostéfaoui, E. Mourgaya, and M. Raynal. Asynchronous implementation of failure detectors. In *Proceedings if the International Conference on Dependable Systems and Networks (DSN'03)*, pages 351–360. IEEE, June 2003.

[MPR04] A. Mostéfaoui, D. Powel, and M. Raynal. A hybrid approach for building eventually accurate failure detectors. In *Proceedings of the 10th Pacific Rim International Symposium on Dependable Computing (PRDC'04)*, pages 57–65, Papeete, Tahiti, March 2004. IEEE.

[MS01] D. R. Mauro and K. J. Schmidt. *Essential SNMP*. O'Reilly and Associates, Sebastopol, CA, USA, july 2001.

[Mur05] P. Murray. A distributed state monitoring service for adaptative management. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, pages 200–205, Yokohama, Japan, June 2005. IEEE Computer Society.

[NJP04] R. C. Nunes and I. Jansch-Pôrto. Qos of timeout-based self-tuned failure detectors: The effects of the communication delay predictor and the safety margin. In IEEE Computer Society, editor, *Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04)*, 2004.

[NSN] The net-SNMP project. http://net-snmp.sourceforge.net/.

[OGP03] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why do internet services fail, and what can be done about it? In *Proceedings of the 4th Symposium on Internet Technologies and Systems (USITS '03)*, pages 1–16, Seatle, WA, USA, March 2003. USENIX.

[Pos81] J. Postel. Internet control message protocol. RFC 792, IETF, 1981.

[Ray05] M. Raynal. A short introduction to failure detectors for asynchronous distributed systems. *ACM SIGACT News*, 36(1):53–70, March 2005.

[RB91] A. M. Ricciardi and K. P. Birman. Using process groups to implement failure detection in asynchronous environment. In ACM, editor, *Proceedings of the tenth annual symposium on Principles of distributed computing PODC'91*, pages 341–353, 1991.

[Rei02] F. Reichenbach. Service SNMP de détection de faute pour des systèmes répartis. Diploma thesis, École Polytechnique Fédérale de Lausanne, Switzerland, 2002.

[SAF00] S. Subramanyan, J. M. Alonso, and J. A. B. Fortes. A scalable SNMP-based distributed monitoring system for heterogeneous network computting. In *Proceedings of Supercomputing 2000*, Dallas, Texas, USA, 2000. IEEE Computer Society.

[SBCdF03] M. R. Sampaio, F. V. Brasileiro, W. Cirne, and J. C. A. de Figueiro. How bad are wrong suspicions? towards adaptative distributed protocols. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN2003)*, pages 551–561, San Francisco, CA, USA, june 2003. IEEE Computer Society.

[SDF+99] P. Stelling, C. DeMatteis, I. Foster, C. Kesselman, C. Lee, and G. von Laszewski. A fault detection service for wide area distributed computations. *Cluster Computing*, 2(2):117 – 128, June 1999.

[SFK+98] P. Stelling, I. Foster, C. Kesselman, C. Lee, and G. von Laszewski. A fault detection service for wide area distributed computations. In *Proceedings of the 7^{th} Symposium on High Performance Distributed Computing*, pages 268–278, Chicago IL USA, 1998. IEEE.

[SIK02] H. Shinbo, A. Idoue, and T. Kato. A failure detection procedure for internet based on communication retrial. In *Proceedings of the 20th International Conference on Applied Informatics (AI 2002)*, pages 351–482, Innsbruck, Austria, February 2002. IASTED.

[SM01] I. Sotoma and E. R. M. Madeira. ADAPTATION - algorithms to adaptive fault monitoring and their implementation on CORBA. In *Proceedings of the Third International Symposium on Distributed Objects and Applications (DOA'01)*, pages 219–229, 2001.

[SS83] R. D. Schlichting and F. B. Schneider. Fail-stop processors: An approach to designing fault-tolerant computing systems. *Computer Systems*, 1(3):222–238, 1983.

[STD02] M. S. Su, K. Thulasiraman, and A. Das. A scalable on-line multilevel distributed network fault detection/monitoring system based on the snmp protocol. In *Proceedings of the Global Telecommunications Conference (GLOBECOM '02)*, volume 2, pages 1960– 1964. IEEE, November 2002.

[Vol01] R. Vollbrecht. The telecom system view. White paper, 12 2001.

[vRMH98] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-based failure detection service. In N. Davies, K. Raymond, and J. Seitz, editors, *Proceedings of Middleware '98, the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 55–70, The Lake District, UK, September 1998.

[Wal95] S. Waldbusser. Remote network monitoring management information base. RFC 1757, IETF, 1995.

[WDS03] M. Wiesmann, X. Défago, and A. Schiper. Group communication based on standard interfaces. In *Proceedings of the IEEE International Symposium on Network Computing and Applications (NCA-03)*, pages 140–147, Cambridge, MA, USA, 2003.

[WG00] S. Waldbusser and P. Grillo. Host resources MIB. RFC 2790, Internet Engineering Task Force (IETF), 2000.

[WNSS02] K. Wansbrough, M. Norrish, P. Sewell, and A. Serjantov. Timing UDP: Mechanized semantics for sockets, threads, and failures. In D. Le Métayer, editor, *Proceedings ot the 11th European Symposium on Programming, Programming Languages and Systems (ESOP 2002)*, volume 2305 / 2002, pages 278–294, Grenoble, France, April 2002. Springer-Verlag GmbH.

[WSGY00] A. Waheed, W. Smith, J. George, and J. Yan. An infrastructure for monitoring and management in computational grids. In S. Dwarkadas, editor, *Proceedings of the 5th International Workshop on Languages, Compilers, and Run-Time Systems for Scalable Computers (LCR 2000)*, page 235, Rochester, NY, USA, May 2000.

[X.602] ASN.1 encoding rules: Specification of basic encoding rules (BER), canonical encoding rules (CER) and distinguished encoding rules (DER). ITU-T recommandation X.690, 2002.

Appendix I SMI Failure detection service specification

```
SnmpFD DEFINITIONS ::= BEGIN

IMPORTS
  enterprises
    FROM    RFC1155-SMI
  Counter32, Integer32, TimeTicks, OBJECT-TYPE, MODULE-IDENTITY
    FROM    SNMPv2-SMI
  RowStatus, RowPointer, DisplayString, TEXTUAL-CONVENTION
    FROM    SNMPv2-TC ;

snmpFDMIB MODULE-IDENTITY
    LAST-UPDATED "200501300000Z"
    ORGANIZATION "JAIST"
    CONTACT-INFO "mail: wiesmann@jaist.ac.jp
                  Matthias Wiesmann
                  Japanese Advanced Institute of Science and Technology
                  Asahidai 1-1 Nomi-shi Ishikawa-ken 923-1211 Japan"
    DESCRIPTION  "Definitions for failure detection service."
    REVISION     "200501300000Z"
    DESCRIPTION  "Initial publication."
    ::= { enterprises 999 }

heartbeat       OBJECT IDENTIFIER ::= { snmpFDMIB 1 }
knowledge       OBJECT IDENTIFIER ::= { snmpFDMIB 2 }
basicfd         OBJECT IDENTIFIER ::= { snmpFDMIB 3 }

-- --------------------
-- Heartbeat Definitions
-- --------------------
```

```
HBTARGET ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "255a"
    STATUS current
    DESCRIPTION "Represents the tag associated with a hearbeat"
    SYNTAX       OCTET STRING (SIZE (0..255))


HeartBeatEntry ::=
    SEQUENCE {
        hbtargettag     HBTARGET,
        hbtargetnum     INTEGER,
        hbpid           INTEGER,
        hbinterval      TimeTicks,
        hbcounter       Counter32,
        hbstatus        RowStatus
    }

hBTable OBJECT-TYPE
    SYNTAX SEQUENCE OF HeartBeatEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION "Table of heartbeats"
    ::= { heartbeat 1 }

heartbeatEntry OBJECT-TYPE
    SYNTAX HeartBeatEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION "List of heartbeats"
    INDEX { hbtargettag, hbtargetnum }
    ::= { hBTable 1 }

hbtargettag OBJECT-TYPE
    SYNTAX HBTARGET
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "tag of target"
    ::= { heartbeatEntry 1 }

hbtargetnum OBJECT-TYPE
    SYNTAX INTEGER(1..32000)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "index for a given tag-target"
    ::= { heartbeatEntry 2 }

hbpid OBJECT-TYPE
    SYNTAX INTEGER
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "pid of monitored process"
    ::= { heartbeatEntry 3 }
```

```
hbinterval OBJECT-TYPE
    SYNTAX TimeTicks
    MAX-ACCESS read-write
    STATUS current
    DESCRIPTION "time interval between heartbeats"
    ::= { heartbeatEntry 4 }

hbcounter OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "count of sent heartbeats"
    ::= { heartbeatEntry 5 }

hbstatus OBJECT-TYPE
    SYNTAX RowStatus
    MAX-ACCESS read-write
    STATUS current
    DESCRIPTION "status of the row"
    ::= { heartbeatEntry 6  }

-- ---------------------------------
-- Knowledge Base Definitions
-- ---------------------------------

ProcessState ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION "state of a process -
                 extension of hrSWRunStatus in RFC 2790"
    SYNTAX INTEGER {
        running(1),
        trusted(2),
        waiting(3),
        invalid(4),
        unknown(5)
    }

KnowBaseEntry ::=
    SEQUENCE {
        kbindex         Integer32,
        kbhostname      DisplayString,
        kbpid           Integer32,
        kbstate         ProcessState,
        kbrecv          TimeTicks,
        kbcounter       Counter32,
        kbsender        DisplayString,
        kbsendtime      TimeTicks,
        kbeffinterval   TimeTicks,
        kbreqinterval   TimeTicks,
        kbhbindex       RowPointer
    }

kbTable OBJECT-TYPE
    SYNTAX SEQUENCE OF KnowBaseEntry
```

```
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION "Table of remote process information"
    ::= { knowledge 1 }

kbentry OBJECT-TYPE
    SYNTAX KnowBaseEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION "Remote information about a process"
    INDEX { kbindex }
    ::= { kbTable  1 }

kbindex OBJECT-TYPE
    SYNTAX Integer32(1..32000)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "index of row"
    ::= { kbentry 1 }

kbhostname OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "hostname for remote process"
    ::= { kbentry 2 }

kbpid OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "pid for remote process"
    ::= { kbentry 3 }

kbstate OBJECT-TYPE
    SYNTAX ProcessState
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "state of remote process"
    ::= { kbentry 4 }

kbrecv OBJECT-TYPE
    SYNTAX TimeTicks
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "reception time of last heartbeat"
    ::= { kbentry 5 }

kbcounter OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "Number of received heartbeats"
    ::= { kbentry 6 }
```

```
kbsender OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "sender of last heartbeat"
    ::= { kbentry 7 }

kbsendtime OBJECT-TYPE
    SYNTAX TimeTicks
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "send time of last heartbeat"
    ::= { kbentry 8 }

kbeffinterval OBJECT-TYPE
    SYNTAX TimeTicks
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "effective interval included in last heartbeat"
    ::= { kbentry 9 }

kbreqinterval OBJECT-TYPE
    SYNTAX TimeTicks
    MAX-ACCESS read-write
    STATUS current
    DESCRIPTION "requested interval included in last heartbeat"
    ::= { kbentry 10 }

kbhbindex OBJECT-TYPE
    SYNTAX RowPointer
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "heartbeat index that generated this entry"
    ::= { kbentry 11 }


-- ----------------------------------
-- Basic Failure Detector Definitions
-- ----------------------------------

BasicFDState ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION "State of a failure detector.
                 Suitable for P, diamond-S
                 and similar failure detectors"
    SYNTAX INTEGER {
        failed(1),
        suspected(2),
        trusted(3)
    }


BasicFDEntry ::=
```

```
    SEQUENCE {
        fdindex         Integer32,
        fdhostname      DisplayString,
        fdpid           Integer32,
        fdtimeout       TimeTicks,
        fdtarget        DisplayString,
        fdstate         BasicFDState,
        fdknowindex     RowPointer,
        fduptime        TimeTicks,
        fdinterval      TimeTicks,
        fdstatus        RowStatus
        }

fdtable OBJECT-TYPE
    SYNTAX SEQUENCE OF BasicFDEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION "Table of failure detectors"
    ::= { basicfd 1 }

basicFDEntry OBJECT-TYPE
    SYNTAX BasicFDEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION "List of failure detector entries"
    INDEX { fdindex }
    ::= { fdtable 1 }

fdindex OBJECT-TYPE
    SYNTAX Integer32(1..32000)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "index of the row"
    ::= { basicFDEntry 1 }


fdhostname OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "host of the process"
    ::= { basicFDEntry 2 }

fdpid OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "pid of the process"
    ::= { basicFDEntry 3 }

fdtimeout OBJECT-TYPE
    SYNTAX TimeTicks
    MAX-ACCESS read-only
    STATUS current
```

```
    DESCRIPTION "time out of failure detector"
    ::= { basicFDEntry 4 }

fdtarget OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "target failure detector"
    ::= { basicFDEntry 5 }

fdstate OBJECT-TYPE
    SYNTAX BasicFDState
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "state of the process"
    ::= { basicFDEntry 6 }

fdknowindex OBJECT-TYPE
    SYNTAX RowPointer
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "index in knowledge table"
    ::= { basicFDEntry 7 }

fduptime OBJECT-TYPE
    SYNTAX TimeTicks
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "uptime of process"
    ::= { basicFDEntry 8 }

fdinterval OBJECT-TYPE
    SYNTAX TimeTicks
    MAX-ACCESS read-write
    STATUS current
    DESCRIPTION "heartbeat interval (repeated from knowledge)"
    ::= { basicFDEntry 9 }

fdstatus OBJECT-TYPE
    SYNTAX RowStatus
    MAX-ACCESS read-write
    STATUS current
    DESCRIPTION "status of the row"
    ::= { basicFDEntry 10 }

END
```