DIONEI MARCELO MORAES

UM SERVIÇO BASEADO EM SNMP PARA DETECÇÃO DE FALHAS EM SISTEMAS DISTRIBUÍDOS NA INTERNET

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Elias P. Duarte Jr.

CURITIBA

2009

SUMÁRIO

Resumo

Este trabalho apresenta uma implementação de detectores de falhas para sistemas distribuídos executados na Internet. A implementação é baseada em SNMP *(Simple Network Management Protocol)* e serviços Web. Detectores de falhas são oráculos distribuídos que fornecem informações sobre processos em um sistema distribuído. Aplicações distribuídas podem utilizar o sistema para detectar falhas em seus processos tanto em rede local, quanto na Internet. Cada processo é identificado pelo seu endereço IP, porta e identificador do processo no sistema operacional local. Um processo pode estar *falho, supeito* ou *sem-falha*. Um agente SNMP pode monitorar um ou mais processos em uma rede local. Existe um agente executando em cada rede local onde existam processos a serem monitorados. O agente disponibiliza as informações sobre o estado de execução dos processos através de uma MIB *(Management Information Base)*. Monitores atualizam e trocam essas informações entre si através de operações SNMP e serviços Web. Um processo monitorado envia *heartbeats* em um intervalo estipulado. Se o monitor não recebe um *heartbeat* de um processo sem-falha dentro de um tempo limite calculado, então o estado deste processo é atualizado para suspeito. Para calcular este tempo limite, é utilizado o algoritmo do TCP para determinar o *timeout*. O monitor considera um processo como falho se recebe esta informação do sistema operacional local. Monitores de diferentes redes locais se comunicam pela Internet através de serviços Web, de duas possíveis maneiras. Na primeira, cada mudança de estado de um processo local, identificada pelo agente, é notificada a todos os outros agentes, de modo que todos os agentes conhecem o estado de todos os processos da aplicação distribuída na Internet. Na segunda, o agente local obtém informações sobre processos que executam em outras redes locais apenas quando a aplicação necessita destas informações. O sistema foi implementado e avaliado com processos monitorados executando tanto em redes locais quanto distribuídos por diferentes regiões do mundo em nodos registrados no Planet Lab. Diferentes experimentos foram realizados, apresentando consumo de CPU, tempo de detecção de falha e taxa de engano.

Abstract

This work presents an implementation of failure detectors for Internet-based distributed systems. The implementation is based on SNMP (*Simple Network Management Protocol*) and Web Services. Failure detectors are distributed oracles that supply information about the execution state of processes of a distributed system. Each process is identified by its IP address, port and its process identier (*pid*) at the local operating system. A process can be in one of the following states: *crash, suspect or working.* An SNMP agent can monitor one or more processes in a LAN. An agent that acts a process monitor is executed at every LAN on which processes are running. The agent supplies information about process execution state through a MIB *(Management Information Base).* Monitors update and exchange information using SNMP operations and Web Services. A monitored process sends heartbeats at a given interval. If a monitor does not receive a heartbeat from a working process within the maximum computed time limit, then the state of the monitored process is toggled to suspect. The system employs the same algorithm of TCP's retransmission timer in order to compute the time-out interval that leads to a suspicion. The monitor identifies if a process has crashed only when it receives such information from an entity running at the process' local operating system. Monitors at different LANs communicate across the Internet using Web Services, in two possible ways. In the first, after a change in the state of a monitored process, information about the new state is notified to all monitors. Otherwise the local monitor gets information about the processes that execute in other LANs only when this information is explicitly requested by a application that needs that information. The system was implemented and evaluated for monitored process running both at a LAN and distributed throughout the world at the Planet Lab. The heartbeat interval ranged from a few milliseconds to hundreds of milliseconds. Different experiments were carried out, showing CPU usage, failure detection latency, and mistake rate.

Capítulo 1

Introdução

Este trabalho apresenta uma implementação de uma importante abstração de sistemas distribuídos, os detectores de falhas não confiáveis, utilizando o protocolo de gerência da *Internet*, o SNMP *(Simple Network Management Protocol)*. Esta introdução motiva e descreve brevemente o trabalho, estando organizada como segue. A seção seguinte apresenta brevemente conceitos de gerência de redes e o arcabouço SNMP. A seção 1.2 aborda o conceito de detectores de falhas e apresenta a proposta deste trabalho.

1.1 Gerência de Redes

Os primeiros computadores eram centralizados e não compartilhavam dados entre si. A comunicação entre computadores surgiu com terminais conectados a *mainframes*. A velocidade de transmissão era baixa e a única preocupação era manter a comunicação entre o mainframe e os terminais que o acessavam. O serviço de garantia de disponibilidade de comunicação era fornecido pelos próprios fabricantes de *mainframes*. A atividade de gerenciamento de redes era inexistente até então. Nos dias de hoje, com o avanço tecnológico dos equipamentos que compõem as redes, os sistemas de gerência de redes precisam atender às necessidades das instituições que confiam a segurança e a disponibilidade de suas informações às redes de computadores.

Os sistemas de gerência de redes surgiram da necessidade de disponibilizar, de forma integrada, ferramentas e sistemas que permitam monitorar e controlar os recursos e serviços das redes de computadores. Para definir a funcionalidade esperada destes sistemas, a *International Organization for Standardization* (ISO) propôs cinco áreas funcionais [1]:

gerência de falhas, gerência de configuração, gerência de segurança, gerência de desempenho e gerência de contabilização. Os principais componentes de um sistema clássico de gerência de redes são: estação de gerência, agente, base de informação de gerência e protocolo de gerência [2]. As três arquiteturas de gerência mais comuns são: centralizada, hierárquica e distribuída [1]. A arquitetura centralizada concentra sua estação de gerência em apenas um *host*. A arquitetura hierárquica utiliza várias estações de gerência, com uma estação atuando como principal ou central e outras estações atuando como estações de gerência intermediárias. Na arquitetura distribuída, ao invés de ter apenas um gerente ou gerentes hierárquicos, existem várias estações de gerência.

O protocolo SNMP é o padrão TCP/IP para gerência de redes de computadores [2]. Qualquer dispositivo que possui uma interface de rede pode ser gerenciado utilizando o protocolo SNMP, como por exemplo, roteadores, impressoras, dispositivos de armazenamento, dentre outros. A versão atual do SNMP é a terceira, SNMPv3. A arquitetura do protocolo SNMP [3] pode seguir as mesmas características das arquiteturas de gerência de redes citadas anteriormente. O SNMP também possui os componentes clássicos de um sistema de gerência de redes: gerente, agente e base de informação de gerência. O gerente concentra os alertas do sistema e serve de interface para o gerente humano. Os agentes monitoram os recursos da rede e enviam alertas ao gerente quando há alteração no estado de funcionamento dos recursos monitorados. O gerente pode ainda questionar os agentes sobre o estado de cada recurso. A base de informação de gerência (*MIB - Management Information Base*) organiza as informações de gerência utilizadas pelo SNMP para monitorar e controlar os dispositivos da rede. As operações do SNMP possibilitam a alteração e a inspeção do valor de objetos, que representam recursos da rede, nas MIBs [4]. As operações chave do SNMP são *GET*, *SET* e *TRAP*. A operação *GET* é utilizada pelo gerente para questionar um agente sobre o estado de um recurso. A operação *SET* é utilizada pelo gerente para alterar o estado de funcionamento de um recurso. A operação *TRAP* é uma notificação enviada de um agente para o gerente quando um evento inesperado acontece.

1.2 Serviço de Detecção de Falhas

Sistemas distribuídos assíncronos [5] são caracterizados por não possuírem limite de tempo de processamento e transmissão de mensagens. Isto implica no problema central deste tipo de sistema: suponha um processo p aguardando uma mensagem de outro processo q, que pode falhar. Passado um determinado tempo de espera sem receber tal mensagem, p pode parar de esperar, assumindo que q está falho. Neste caso, se q estiver com lentidão em seu processamento, p cometeu um engano ao considerar falho um processo que está apenas lento. Por outro lado, p pode esperar indefinidamente pela mensagem de q, assumindo que q está apenas lento. Neste caso, se q estiver falho, p vai esperar eternamente por uma mensagem que não irá chegar. Dentro deste contexto, Chandra e Toueg introduziram o conceito de detectores de falhas [6]. Detectores de falhas são oráculos distribuídos que fornecem informações a respeito do estado de execução de processos em um sistema distribuído. Entre as características dos detectores não-confiáveis, destaca-se o fato de que os agentes podem cometer enganos, ou seja, um processo sem falha pode ser suspeito por engano, ou um processo falho pode ser considerado sem falha. Detectores de falhas permitem que aplicações identifiquem falhas em seus componentes, sem ter que implementar um mecanismo interno para isto, favorecendo a modularidade.

O SNMP também pode monitorar e controlar processos. É esta funcionalidade do SNMP que permite sua utilização no desenvolvimento de serviços de detecção de falhas em processos de sistemas distribuídos. Wiesmann, Urbán e Défago apresentam em [7] um serviço de detecção de falhas baseado em SNMP. Este serviço foi denominado pelos seus autores como SNMP-FD *(SNMP-Failure Detector)*. Seu funcionamento consiste da utilização de várias MIBs, entre elas, *Host Resource MIB* [8], *Target and Notification MIBs* [9], *Alarm Description MIB* [10], *Event MIB* [11], e outras implementadas especificamente para o *framework*. Com a utilização destas MIBs, é possível obter informações sobre processos em execução, definir tipos de alertas, condições a serem satisfeitas para que um alerta seja disparado, especificar características de *heartbeats*, dentre outras funcionalidades. No SNMP-FD são implementados dois tipos de agentes: agente monitor e agente monitorado. Cada tipo de agente implementa MIBs diferentes. Para validar

o *framework*, foi implementado um detector de falhas simples baseado em *timeouts*. O detector de falhas retorna três possíveis valores para cada processo: confiável, suspeito ou falho.

O presente trabalho apresenta um serviço de detecção de falhas para sistemas distribuídos baseado em SNMP, para ser utilizado tanto em redes locais, quanto na Internet. Aplicações distribuídas que necessitam conhecer o estado de seus processos antes de tomar decisões podem consultar o detector de falhas para tal. A interface para o serviço é disponibilizada através de um agente SNMP. O serviço tem um agente executando em cada rede local onde houverem processos a serem monitorados. Esta monitoração é realizada através de *heartbeats* e do sistema operacional, que identificam se um determinado processo está falho, suspeito ou sem-falha. Os *heartbeats* são utilizados para identificar processos suspeitos. Um processo é considerado suspeito se não entregar um *heartbeat* dentro do intervalo estipulado. O sistema operacional é utilizado para confirmar processos falhos. Se o processo entrega seus *heartbeats* no intervalo estipulado, e se não são identificadas falhas através do sistema operacional, então o processo é considerado sem-falha.

O agente é composto de uma MIB contendo informações referentes aos processos locais e remotos, a saber, seus identificadores, compostos por endereço IP, porta e o identificador do processo no sistema operacional local *(PID)*, e seu estado de execução. Outra MIB é utilizada para armazenar informações estatísticas sobre os *heartbeats*, como o *timestamp* referente ao último *heartbeat* enviado pelo processo, média do tempo de envio de *heartbeats*, e desvio padrão. Os agentes devem trocar estas informações entre si através de operações SNMP. Agentes de diferentes redes locais se comunicam pela Internet através de serviços Web, de duas possíveis maneiras. Na primeira, cada mudança de estado de um processo local, identificada pelo agente, é notificada a todos os outros agentes, de modo que todos os agentes conhecem o estado de todos os processos da aplicação distribuída. Na segunda, o agente local obtém informações sobre processos que executam em outras redes apenas quando a aplicação necessita destas informações.

Neste trabalho, ao contrário de [7], o SNMP é utilizado apenas para manter informações sobre o estado de funcionamento de processos, e também é utilizado como

interface para a aplicação. Outras funcionalidades, como envio de *heartbeats*, serão implementadas sem a utilização do SNMP. Outra contribuição deste trabalho é a utilização do SNMP em conjunto com serviços Web para possibilitar a detecção de falhas de processos na Internet. Um processo pode verificar o estado de funcionamento de outros processos consultando seu agente local. Aplicações que necessitem manter seu funcionamento correto mesmo na presença de falhas de alguns de seus componentes podem utilizar este serviço para monitorar seus processos.

Para avaliar a implementação deste trabalho foram realizados cinco experimentos. Consumo de CPU dos processos monitorados e do monitor em relação ao intervalo entre *heartbeats*, tempo de detecção de falhas de processos em redes locais em relação à variação do intervalo entre *heartbeats*, taxa de engano em relação ao tempo, e tempo de notificação de falhas de nodos distribuídos em diferentes países na Internet. Além destes experimentos, também foi implementada uma aplicação distribuída que envolve a solução para o problema do acordo.

1.3 Organização deste Trabalho

Este trabalho está organizada como segue. No capítulo 2 são descritos conceitos gerais sobre gerência de redes e a aplicação do protocolo SNMP, além do uso de serviços Web em gerência. O capítulo 3 descreve o conceito de detectores de falhas, suas principais propriedades, sua classificação de acordo com as propriedades apresentadas, e as métricas para mensurar sua qualidade de serviço. No capítulo 4, é apresentado o serviço baseado em SNMP para detecção de falhas na Internet, incluindo resultados experimentais. Por fim, o capítulo 5 apresenta a conclusão do trabalho.

Capítulo 2

Gerência de Redes Baseada em SNMP

A gerência de redes permite monitorar e controlar toda a complexa coleção de dispositivos e equipamentos de comunicação que formam uma rede de computadores, com o objetivo de permitir seu uso eficiente e produtivo [1].

Este capítulo começa com uma descrição da funcionalidade dos sistemas de gerência de redes, seguida por uma descrição dos componentes e arquiteturas de gerência, bem como uma descrição do *framework* de gerência da Internet, o *Simple Network Management Protocol* (SNMP). Posteriormente é apresentada a gerência de redes com serviços Web.

2.1 Gerência de Redes: Funcionalidade

Muitas organizações investem quantidades significativas de recursos na construção de redes de dados e dependem fortemente do seu bom funcionamento. É frequente a alocação de um gerente humano que administra a rede e é responsável por garantir sua correta operação. É importante disponibilizar ao gerente humano ferramentas e sistemas que permitam automatizar a monitoração e o controle da rede, tanto quanto possível. Neste contexto surgiram os sistemas de gerência de rede. Para definir a funcionalidade esperada destes sistemas, a *International Organization for Standardization* (ISO) propôs cinco áreas funcionais [1]: gerência de falhas, gerência de configuração, gerência de segurança, gerência de desempenho e gerência de contabilização. Estas áreas são descritas a seguir.

A gerência de falhas permite localizar falhas na rede. Esta tarefa envolve a descoberta,

o isolamento e a resolução de problemas. Utilizando as técnicas de gerência de falhas o gerente humano pode localizar e resolver problemas de forma mais rápida do que sem a utilização de tais técnicas. Um sistema de gerência de falhas pode identificar um problema e notificar o gerente humano, eliminando a necessidade de uma busca exaustiva na rede na tentativa de encontrar o problema.

A gerência de configuração é um processo que envolve identificar quais são os dispositivos críticos para o funcionamento da rede e então configurá-los de forma adequada [1]. A configuração de certos dispositivos de uma rede determina o comportamento desta rede. Um sistema de gerência de configuração pode auxiliar o gerente humano nesta tarefa, provendo funcionalidades como, por exemplo, a reconfiguração de dispositivos como roteadores ou mesmo a atualização de versões de software.

A gerência de segurança permite monitorar e controlar o acesso aos recursos de uma rede. Um sistema de gerência de segurança pode auxiliar o gerente humano provendo monitores que recebem eventos sobre quem está acessando um determinado recurso. Em caso de intrusos, por exemplo, o gerente humano identifica o problema e toma as devidas providências para eliminar o acesso do intruso. O sistema pode ainda gravar registros sobre quais usuários acessaram um determinado recurso.

A gerência de desempenho envolve métricas de desempenho de *hardware* e *software* como, por exemplo, utilização de processador, consumo de disco, tempo de processamento, entre outras. Utilizando informações de desempenho o gerente humano pode assegurar que a rede terá capacidade para comportar os usuários necessários e pode expandir sua capacidade conforme haja necessidade. Um sistema de gerência de desempenho pode monitorar o percentual de uso dos componentes da rede notificando o gerente humano sobre qualquer excesso. O sistema pode ainda armazenar informações estatísticas sobre o uso destes componentes. De posse destas informações o gerente humano é capaz de determinar se a rede necessita ou não de expansão para atender às necessidades da organização.

A gerência de contabilização envolve o mapeamento da utilização de recursos da rede por cada usuário ou grupo de usuários, com o objetivo de assegurar que possuam recursos suficientes para executar suas tarefas. Esta área funcional envolve fornecer, manter ou

retirar permissões de acesso a recursos da rede. Um sistema de gerência de contabilização pode fornecer ao gerente humano informações sobre o percentual de utilização de dispositivos por parte de cada usuário da rede, classificando as informações de acordo com as necessidades do gerente humano. De posse destas informações, o gerente humano pode limitar, conceder ou manter o acesso a dispositivos de acordo com as necessidades da organização.

2.2 Componentes de um Sistema de Gerência de Redes

Os principais componentes de um sistema clássico de gerência de redes são: estação de gerência, agente, base de informação de gerência e protocolo de gerência [2]. Estes componentes são descritos a seguir.

A estação de gerência de rede, *Network Management Station* (NMS), ou simplesmente *gerente*, provê a interface entre o sistema de gerência e o gerente humano. É através desta estação que o gerente humano irá monitorar e controlar os dispositivos da rede. O gerente executa um conjunto de aplicações de gerência diversas, por exemplo para gerência de falhas, gerência de desempenho, entre outras; cada organização pode escolher o conjunto de aplicações que necessita. Um sistema de gerência pode ter apenas um NMS, ou vários, organizados de forma distribuída ou hierárquica.

O *agente* é o componente que mantém informações dos diversos elementos da rede. O agente se comunica com o gerente. Há duas formas de comunicação, descritas a seguir. Na primeira, o gerente faz requisições ao agente. Estas requisições podem ser pedidos de informação ou alteração no estado de um dispositivo da rede. Na segunda forma de comunicação, o agente envia mensagens assíncronas para a estação de gerência, frequentemente denominadas *alarmes*, informando o estado de um determinado dispositivo.

Os recursos da rede são modelados como objetos. Cada objeto representa um aspecto de um recurso gerenciado. Conhecendo o valor destes objetos, os agentes têm, por exemplo, condições de responder às requisições da estação de gerência, informando-a sobre o estado de um dispositivo. Uma base de informação de gerência, *Management Information Base* (MIB), é um conjunto destes objetos. O conceito de MIB é apresentado na seção

2.4.1.

O protocolo de gerência de rede permite a comunicação entre a estação de gerência e o agente. O protocolo utilizado para gerenciar redes TCP/IP *(Transmission Control Protocol/Internet Protocol)* é o SNMP, e é descrito na seção 2.4. As formas com que os componentes da gerência de redes se organizam são descritas na seção seguinte.

2.3 Arquiteturas de Gerência de Redes

Um sistema de gerência de redes pode utilizar várias arquiteturas na organização de seus diversos componentes. As três arquiteturas de gerência mais comuns são: centralizada, hierárquica e distribuída [1], descritas a seguir.

2.3.1 Arquitetura Centralizada

A arquitetura centralizada concentra seu gerente em apenas um *host*, que fica responsável por todas as tarefas de gerência da rede. A figura 2.1 ilustra esta arquitetura. O gerente central desta arquitetura é o destino de todos os alertas e eventos, sendo responsável pela monitoração integral da rede, armazenando toda informação correspondente sobre a rede e provendo acesso a todas as aplicações de gerência. O gerente comunica diretamente com todos os agentes na rede.

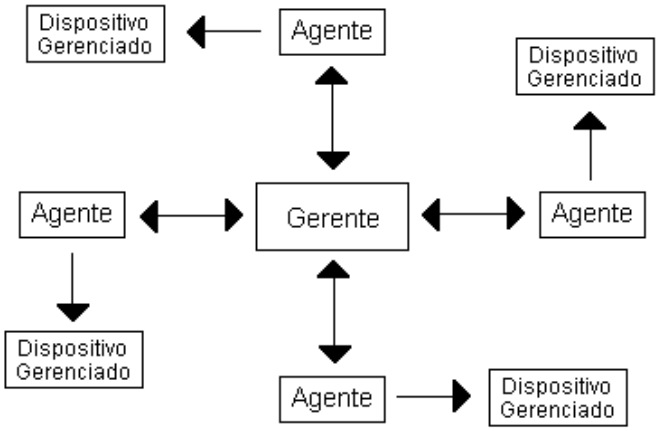

Figura 2.1: Arquitetura Centralizada de Gerência de Redes

Utilizando a abordagem centralizada, o gerente humano tem um único local para monitorar todos os alertas e eventos, o que aumenta a sua conveniência. Ter um único local para acessar todas as informações e aplicações de gerência também facilita a segurança do sistema. A estação de gerência pode ser, por exemplo, mantida em um local de acesso restrito.

Por outro lado, ter todas as funções de gerência instaladas em apenas uma máquina deixa o sistema vulnerável a falhas. Além disso, à medida em que equipamentos são adicionados à rede, a carga da estação de gerência aumenta. As arquiteturas hierárquica e distribuída foram propostas em virtude destes desafios.

2.3.2 Arquitetura Hierárquica

A arquitetura hierárquica utiliza várias estações de gerência, com uma estação atuando como principal ou central e outras estações atuando como gerentes intermediários, que se situam entre a principal e os agentes. A figura 2.2 ilustra esta arquitetura. Como há três tipos de componentes (gerente principal, gerente intermediário e agente) esta arquitetura é dita "em 3 camadas". De qualquer forma, como ilustra a figura 2.2, um gerente pode se comunicar tanto com outro gerente como com agentes, sempre formando uma hierarquia. Algumas funções de gerência residem no gerente principal, outras nos gerentes intermediários. O gerente humano pode configurar os gerentes intermediários para monitorar porções diferentes da rede.

As características chave da arquitetura hierárquica para gerência de redes são: não dependência de uma única estação de gerência e distribuição das tarefas de gerência nos gerentes intermediários que são distribuídos através da rede. Devido ao gerente central esta arquitetura ainda permite armazenamento centralizado de informações de gerência.

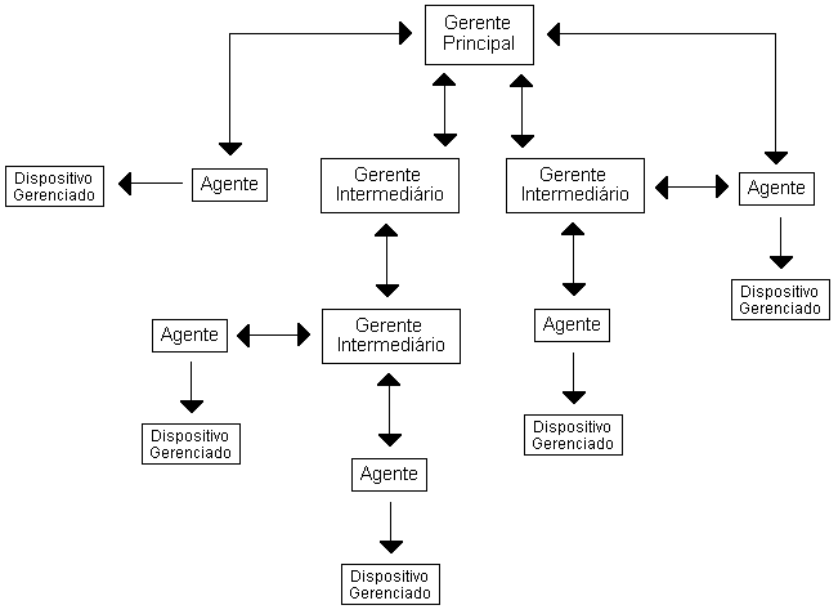

Figura 2.2: Arquitetura Hierárquica de Gerência de Redes

O problema de sobrecarga na estação de gerência, existente na arquitetura centralizada, é amenizado na arquitetura hierárquica. Os gerentes humanos podem distribuir os gerentes intermediários pela rede. Muitas tarefas de gerência de rede requerem a recuperação de informações sobre vários dispositivos monitorados. Consequentemente, ter uma base de dados centralizada geralmente é benéfico. Embora algumas tarefas de gerência estejam executando em gerentes intermediários na arquitetura hierárquica, esta abordagem provê um único local para armazenamento de informações e acesso à gerência da rede como um todo.

O fato da arquitetura hierárquica utilizar múltiplas estações para gerenciar a rede pode fazer com que o gerente humano tenha uma dificuldade a mais quando necessita centralizar todas as informações de gerência. Outro fato a ser considerado é que a distribuição de dispositivos gerenciados por cada gerente intermediário precisa ser pré-determinada e manualmente configurada. Se esta tarefa não for feita com cuidado, pode ocorrer de duas ou mais estações monitorarem o mesmo dispositivo. Se este problema ocorre em grande escala, o processamento e o consumo total de banda pelo sistema de gerência de rede pode aumentar significativamente.

2.3.3 Arquitetura Distribuída

A arquitetura distribuída combina as abordagens centralizada e hierárquica, como ilustra a figura 2.3. Ao invés de ter apenas um gerente ou gerentes hierárquicos, esta abordagem utiliza várias estações de gerência. Cada estação de gerência permite o acesso à gerência da rede como um todo. Um dos gerentes pode ser eleito líder, no contexto de alguma aplicação de gerência. O fato da arquitetura distribuída combinar as abordagens centralizada e hierárquica faz com que ela possua vantagens de ambas, como: informações da rede, alertas e eventos podem ser centralizados; aplicações de gerência podem ser acessadas de um único local; o sistema de gerência não depende de uma única estação; as tarefas de gerência são distribuídas entre diversas estações; monitores podem ser distribuídos através da rede. Por outro lado, esta arquitetura é mais complexa e possui maior redundância de informações.

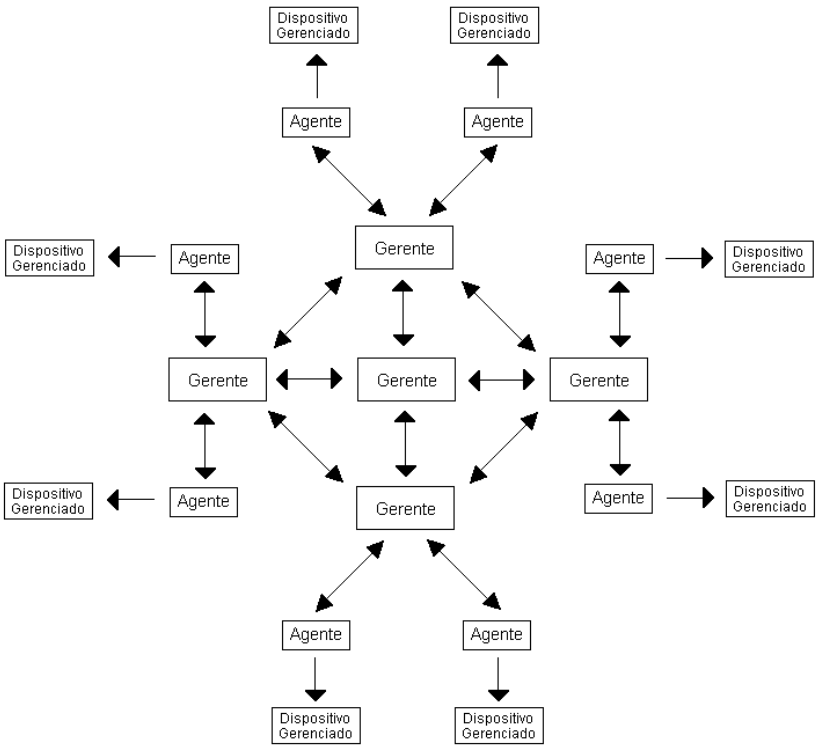

Figura 2.3: Arquitetura Distribuída de Gerência de Redes

2.4 *Simple Network Management Protocol* (SNMP)

O protocolo SNMP é o padrão TCP/IP para gerência de redes de computadores [2]. Qualquer dispositivo que possua uma interface de rede pode ser gerenciado utilizando-se o protocolo SNMP. Os componentes da rede monitorados e controlados pelo SNMP são denominados dispositivos gerenciados. Exemplos de dispositivos gerenciados podem ser roteadores, impressoras, dispositivos de armazenamento, dentre outros. A cada dispositivo estão associadas algumas informações de gerência. O SNMP monitora e controla estes dispositivos utilizando estas informações. Desta forma, vários fabricantes de dispositivos, desde *no-breaks* até computadores de grande porte, implementam mecanismos que permitem que os mesmos possam ser gerenciados através do SNMP. Além de dispositivos físicos, o SNMP também auxilia na gerência de *software* e serviços. A monitoração de processos via SNMP é o foco deste trabalho.

O SNMP é um protocolo simples e, consequentemente, fácil de ser implementado. O protocolo está situado no nível de aplicação e utiliza o *User Datagram Protocol* (UDP) na camada de transporte. O UDP é não orientado a conexão, portanto o SNMP não mantém conexão entre agentes e gerentes para transmissão de mensagens. O UDP provê um serviço de transporte rápido e com o mínimo de alocação de recursos, entretanto não provê um método confiável para troca de mensagens.

A primeira versão do SNMP é conhecida como SNMPv1 [12, 13, 14, 15] ou simplesmente SNMP. A segunda versão, conhecida como SNMPv2 [2, 4, 16, 17], é um melhoramento da primeira versão. A versão atual do protocolo é a 3, e é conhecida como SNMPv3 [3]. Em sua primeira versão, procurou-se estabelecer um protocolo com operações simples e eficientes, definir a base de dados e sua estrutura, além dos objetos gerenciáveis. O SNMPv1 tem poucos mecanismos de segurança. O SNMPv2 foi proposto com o intuito de suprir as deficiências identificadas na primeira versão. As mudanças propostas foram: melhorias na estrutura das informações, novas operações de protocolo, suporte à estratégia de gerência distribuída e novos mecanismos de segurança. Os mecanismos de segurança propostos no SNMPv2 não tiveram aceitação unânime na comunidade, o que resultou em diversas propostas alternativas [2]. O SMNPv3 propõe mecanismos eficazes de segurança

[18, 19]. Outros módulos referentes ao SNMPv2, incluindo as operações do protocolo, não foram alterados, facilitando a transição do SNMPv2 para o SNMPv3 [4, 16, 17].

A arquitetura do protocolo SNMP [3] pode seguir as mesmas características das arquiteturas de gerência de redes descritas na seção 2.3. O SNMP também possui os componentes clássicos de um sistema de gerência de redes; são eles: gerente, agente e base de informação de gerência, descritos na seção 2.2. As informações de gerência utilizadas pelo SNMP para gerenciar os dispositivos de uma rede são organizadas em MIBs. O conceito de MIB é descrito abaixo.

2.4.1 *Management Information Base* (MIB)

A MIB é uma base virtual de dados que reúne informações sobre os dispositivos gerenciados de uma rede [12, 13, 16]. Uma MIB contém um conjunto de *objetos* gerenciados. Um objeto gerenciado é uma visão abstrata de um dispositivo gerenciado de uma rede. Sendo assim, todos os dispositivos que devem ser gerenciados pelo SNMP devem ser modelados e as estruturas de dados resultantes desta modelagem são os objetos gerenciados, que devem estar presentes na MIB utilizada pelo SNMP para monitorar e controlar a rede. No SNMP os objetos não são realmente os objetos definidos em linguagens orientadas a objetos, mas sim variáveis simples com características básicas, como tipo e permissão de leitura e escrita. Portanto, de maneira simplificada, a MIB é um conjunto de variáveis acessadas por entidades SNMP.

Os objetos gerenciados podem ter permissão de acesso de leitura e/ou escrita, sendo que uma leitura a um objeto gerenciado retorna o estado do dispositivo gerenciado correspondente. Um acesso de escrita em um objeto gerenciado resulta na alteração do estado do dispositivo correspondente a tal objeto.

Os objetos gerenciados são organizados em uma hierarquia de árvore e são identificados por um OID *(Object Identifier)* [16]. As folhas desta árvore representam os objetos gerenciados e a sua estrutura separa os objetos em grupos. O OID de um objeto gerenciado é uma sequência de inteiros separados por pontos, e é construído com base na hierarquia de árvore da MIB, ilustrada na figura 2.4.

O OID para um objeto em particular pode ser encontrado pelo caminho percorrido da raiz da árvore até o objeto. Por exemplo a árvore da figura 2.4 mostra que o objeto Internet pode ser referenciado pelo OID `1.3.6.1` ou pelo nome `iso.org.dod.internet`.

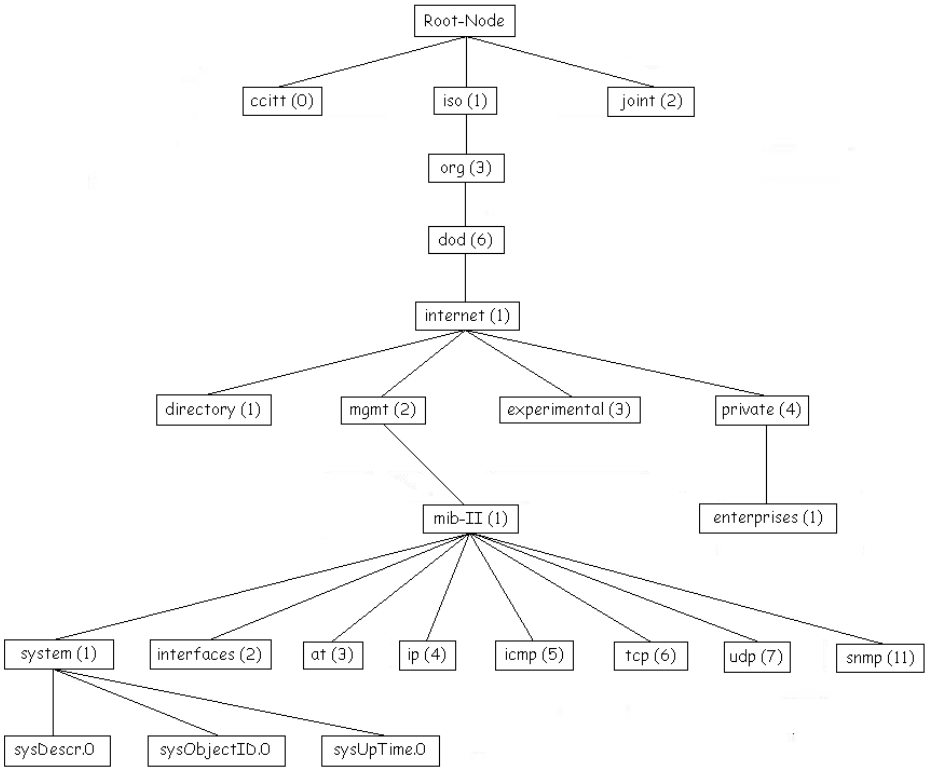

Figura 2.4: Uma Árvore de OID's

Cada agente na rede mantém uma MIB que reflete seu estado. O gerente pode monitorar e controlar estes agentes examinando e alterando, respectivamente, os valores dos objetos nas suas MIBs.

A RFC 1066 [12] especificou a MIB I, que é a primeira especificação da MIB. Esta especificação define o conjunto de informações de gerência necessárias para gerenciar redes baseadas em TCP/IP. A MIB II, uma evolução da MIB I, foi proposta e publicada na RFC 1213 [13] e posteriormente modificada e documentada na RFC 3418 [16].

A figura 2.4 mostra a definição de três tipos de MIBs: MIB-II, MIB experimental e MIB privada. A MIB-II é a especificação de MIB mais importante e contém apenas os objetos considerados essenciais. Esta MIB armazena informações gerais sobre os protocolos TCP/IP em dispositivos de uma rede. Através da MIB-II é possível obter informações como tempo decorrido desde a última inicialização do sistema, serviços oferecidos por um

determinado dispositivo e estado de uma interface de rede, entre outras. A MIB experimental contém conjuntos de objetos que ainda estão em fase de desenvolvimento e teste. A MIB privada contém objetos específicos dos dispositivos gerenciados, como por exemplo informações de configuração.

Utilizando estes objetos, além de outros objetos padrão que não foram descritos acima, é possível monitorar e controlar diversos dispositivos de uma rede. O SNMP permite ainda a implementação de novas MIBs. Desta forma, o usuário pode desenvolver objetos para gerenciar dispositivos de acordo com as necessidades de sua rede.

A MIB é descrita através de uma representação abstrata dos seus dados, utilizando a linguagem ASN.1 *(Abstract Syntax Notation One)*, e do SMI *(Structure of Management Information)* [14], que define a estrutura de informações de gerência. O ASN.1 e o SMI são descritos a seguir.

2.4.1.1 *Abstract Syntax Notation One* (ASN.1)

Abstract Syntax Notation One (ASN.1) é uma linguagem formal desenvolvida e padronizada para definir sintaxes abstratas de dados. Adicionalmente, o ASN.1 é utilizado para definir a estrutura de aplicações e o formato dos *Protocol Data Units* (PDUs). Finalmente, o ASN.1 é utilizado também para definir a base de informações de gerência do SNMP [2].

2.4.1.2 *Structure of Management Information* (SMI)

O SMI, especificado na RFC 1155 [14], define o *framework* geral no qual uma MIB pode ser definida e construída [2]. O SMI define o tipo de dados que pode ser usado na MIB e especifica como os recursos são representados e nomeados. O SMI não suporta estrutura de dados complexas. Esta filosofia do SMI simplifica a tarefa de implementação e aumenta a interoperabilidade entre MIBs. Um objetivo do SMI é fazer a descrição de uma MIB simples e extensível.

2.4.2 Operações do Protocolo

As operações do SNMP possibilitam a alteração e a inspeção do valor de objetos nas MIBs [4]. As operações chave do SNMP são *GET*, *SET* e *TRAP*.

A operação *GET* possibilita à estação de gerência requisitar o valor de um objeto a um agente. A operação *SET* possibilita à estação de gerência alterar o valor de um objeto, modificando assim o estado do dispositivo modelado por tal objeto.

Se a estação de gerência é responsável por um número grande de agentes, e se cada agente controla uma quantidade grande de objetos, então é impraticável para a estação de gerência efetuar regularmente operações *GET* para todos os objetos [2]. Ao invés disto, usa-se a operação *TRAP*. *TRAP* é uma notificação enviada de um agente para a estação de gerência quando um evento inesperado acontece.

As outras operações podem ser consideradas generalizações da operações acima descritas, entre elas [4]: GetRequest, GetNextRequest, GetBulkRequest, SetRequest, SNMPv2-Trap, InformRequest e Response.

2.4.3 Proxies

O uso do SNMP requer que todos os agentes, assim como as estações de gerência, suportem protocolos TCP/IP, tais como UDP e IP. Isto limita ou até mesmo impossibilita a gerência de alguns dispositivos [2].

O conceito de *proxy* foi desenvolvido para resolver este impasse. Um agente *proxy* é designado para mapear requisições em um protocolo comum à estação de gerência e aos agentes. Quando a estação de gerência envia uma requisição ao agente *proxy*, este então repassa a requisição ao recurso, mapeada em um protocolo comum ao mesmo. Quando o agente *proxy* recebe a resposta, ele a repassa à estação de gerência, mapeada no padrão SNMP. Do mesmo modo, se uma notificação de um dispositivo é transmitida ao agente *proxy*, ele envia a notificação à estação de gerência na forma de *TRAP*.

2.4.4 Segurança

As versões SNMPv1 e SNMPv2 utilizam o conceito de comunidade para definir as permissões de acesso do gerente a um objeto. Esta comunidade é definida no agente. Uma comunidade definida no agente especifica as permissões de acesso do gerente a um objeto, sendo possíveis as permissões de leitura, escrita ou não acessível. Definindo mais de uma comunidade o agente pode regular o nível de acesso de cada gerente. Este modelo de segurança procura assegurar que o rementente de uma mensagem é realmente quem diz ser. Toda mensagem do gerente para um agente inclui o nome da comunidade, para que o agente verifique o nível de acesso do gerente. Este nome funciona como uma senha, que trafega pela rede sem encriptação. Como esta estratégia é muito limitada, muitos gerentes de redes utilizam estas versões do SNMP apenas para monitoramento da rede, e não permitem alteração do valor dos objetos gerenciados.

O SNMPv3 possui suporte para múltiplos modelos de segurança. Dentre estes modelos destacam-se o VACM *(View-Based Access Control Model)* [18] e o USM *(User-Based Security Model)* [19]. O *USM*, definido na RFC 2574, verifica se cada mensagem SNMP não foi modificada durante sua transmissão, verifica a identidade do remetende de cada mensagem (autenticação), detecta mensagens com informações de gerenciamento obsoletas e evita divulgação indevida de mensagens (sigilo). O VACM concede, ou não, acesso a um objeto, de acordo com suas permissões, que podem ser de leitura, escrita ou notificação. O controle de acesso é acionado quando ocorre uma requisição de recuperação ou alteração de um objeto, por parte de uma entidade SNMP, ou ainda quando ocorre uma notificação *(TRAP)*. O VACM define na RFC 2275 um conjunto de serviços que uma aplicação pode utilizar para administrar direitos de acesso. É responsabilidade da aplicação efetuar as chamadas de serviço apropriadas para conferência de acessos. Tanto o *USM* quanto o *VACM* proveêm criptografia dos dados, o que aumenta a segurança desta versão do SNMP em relação às versões anteriores.

2.5 Gerência de Redes com Serviços Web

Os serviços Web formam um conjunto de tecnologias abertas, padronizadas pelo W3C *(World Wide Web Consortium)* [20] para comunicação entre aplicações na Internet. O serviço provê um conjunto de funcionalidades através de uma interface padrão, que garante a interoperabilidade entre arquiteturas diferentes, e provê uma abstração sobre a implementação de serviços. Serviços Web utilizam-se de protocolos Web e XML *(eXtensible Markup Language)* [21] para troca de mensagens. O fato destes protocolos serem abertos e bastante difundidos facilita o uso e integração dos serviços Web. Outro aspecto favorável ao uso dos serviços Web é a possibilidade de utilização de portas normalmente abertas, o que permite a comunicação através de *firewalls*.

Existem diferentes arquiteturas para implementação de serviços Web. Para garantir interoperabilidade, os serviços Web possuem alguns componentes em comum, descritos a seguir. Através do UDDI *(Universal Description, Discovery, and Integration)* [22] os serviços Web podem ser registrados, e localizados dinamicamente. A linguagem utilizada para descrever serviços é a WSDL *(Web Services Description Language)* [23]. WSDL é uma linguagem baseada em XML e descreve serviços de forma estruturada. Através desta linguagem é possível descrever tipos de dados, protocolo de ligação, operações, formato de mensagens, dentre outras características do serviço. O protocolo utilizado para troca de dados na Internet é o SOAP *(Simple Object Access Protocol)* [24]. SOAP é suportado pela maioria das implementações de serviços WEB [25].

Os serviços Web também têm sido utilizados na tarefa de gerência de redes. Além de outras funcionalidades, eles podem prover a comunicação entre protocolos de gerência com aplicações Web. Neste caso, serviços Web podem ser utilizados como gateways para protocolos de gerência que executam em redes locais.

Em [26], denomina-se serviço Web *ad hoc* o serviço Web construído a partir de uma linguagem de programação específica, e não utilizando um padrão de composição de serviços Web. No trabalho aqui apresentado é implementado um serviço Web utilizando-se a linguagem Python [27].

Aspectos como segurança e QoS *(Quality of Service)*, presentes em diversos sistemas,

também estão presentes nos serviços Web. Para um serviço que se comunica utilizando a Internet, mecanismos de segurança devem ser adotados. Os mecanismos de segurança presentes nos serviços Web incluem criptografia e autenticação. A criptografia garante que os pacotes que trafegam na rede não sejam lidos por entidades não autorizadas. Já a autenticação garante que apenas entidades autorizadas se comuniquem com um determinado serviço. A QoS de serviços Web é avaliada em termos de atraso na entrega de mensagens e consumo de banda. Um experimento medindo o atraso de mensagens do serviço Web implementado neste trabalho é apresentado adiante.

Capítulo 3

Detectores de Falhas para Sistemas Distribuídos

Um serviço de detecção de falhas é um *oráculo* distribuído que fornece informações a respeito do estado de processos em um sistema distribuído [7]. Este capítulo aborda conceitos clássicos relacionados a detectores de falhas. As seções 3.1 e 3.2 abordam a noção de detectores de falhas introduzida por Chandra e Toueg em [6]. A seção 3.3 aborda o conceito de qualidade de serviço de detectores de falhas, apresentando métricas que permitem avaliar suas funcionalidades e permitem a comparação entre diferentes detectores de falhas. A seção 3.4 apresenta o conceito de detectores de falhas quiescentes, e a seção 3.5 apresenta um trabalho relacionado, o *framework* SNMP-FD [7].

3.1 Detectores de Falhas: Definição Clássica de Chandra e Toueg

O problema central no desenvolvimento de sistemas distribuídos tolerantes a falhas é o consenso [28]. No problema do consenso, processos precisam decidir sobre um mesmo valor, que depende de uma entrada inicial, mesmo na presença de falhas [29]. Algoritmos que resolvem o problema do consenso podem ser utilizados para resolver vários outros problemas, como por exemplo, eleição de líder e agrupamento *(group membership)*. Em [30], Fisher, Lynch e Paterson provaram que o consenso não pode ser resolvido deterministicamente em sistemas assíncronos sujeitos a sequer uma falha. Este resultado é conhecido como a impossibilidade FLP, das iniciais de seus autores. Esta impossibilidade se dá devido à dificuldade de se determinar se um processo está falho ou apenas lento.

Os detectores de falhas não-confiáveis foram propostos por Chandra e Toueg em [6]. Eles representam uma abstração que corresponde à monitoração de processos de um sistema distribuído. Cada processo em um sistema distribuído tem acesso a um módulo do detector de falhas. Cada módulo monitora um subconjunto de processos no sistema e mantém uma lista dos processos suspeitos de terem falhado. O detector de falhas pode cometer erros; um módulo do detector de falhas pode adicionar processos à lista de suspeitos de forma equivocada. Se posteriormente este módulo identificar o engano que cometeu, o mesmo remove da lista de suspeitos os processos que não haviam falhado. Outro engano que o detector de falhas pode cometer é não suspeitar de um processo que falhou. Desta forma, cada módulo detector de falhas insere e remove processos da lista de suspeitos repetidamente. Além disso, em um determinado tempo, o detector de falhas, em processos distintos, pode ter listas de suspeitos diferentes. Cada processo identifica, através do detector de falhas, quais processos do sistema estão falhos.

Detectores de falhas podem ser usados para resolver o problema do consenso em sistemas assíncronos [6]. Uma classificação de detectores de falhas, em termos de duas propriedades abstratas, *completude* e *precisão*, é apresentada em [6]. Basicamente, *completude (completeness)* requer que um detector de falhas suspeite, em algum momento, de todos os processos que estão falhos, enquanto *precisão (accuracy)* restringe os enganos que o detector de falhas pode cometer.

É possível classificar os detectores de falhas de acordo com a maneira com que satisfazem as propriedades de completude e precisão. Estas propriedades podem ser satisfeitas das seguintes formas:

Completude forte *(strong completeness)*: Após um intervalo de tempo suficientemente grande, todo processo falho é permanentemente suspeito por *todo* processo correto.

Completude fraca *(weak completeness)*: Após um intervalo de tempo suficientemente grande, todo processo falho é permanentemente suspeito por *algum* processo correto.

Precisão forte *(strong accuracy)*: Nenhum processo é suspeito antes de falhar.

Precisão fraca *(weak accuracy)*: Existe ao menos um processo correto que nunca é

suspeito.

Como estas propriedades de precisão são difíceis de serem satisfeitas na prática, foram introduzidas outras duas, que requerem que as propriedades de precisão sejam satisfeitas apenas após um determinado tempo.

Precisão forte final *(eventual strong accuracy)*: Existe um tempo após o qual os processos corretos não são suspeitos por nenhum processo correto.

Precisão fraca final *(eventual weak accuracy)*: Existe um tempo após o qual algum processo correto nunca é suspeito por nenhum processo correto.

Estas propriedades combinadas, resultam em oito classes de detectores de falhas, descritas na próxima seção.

3.2 Classes de Detectores de Falhas

As diversas variações de precisão e completude são emparelhadas para definir classes de detectores de falhas. Existem oito classes, como mostra a tabela 3.1, obtidas selecionando uma das propriedades de completude e uma das propriedades de precisão introduzidas na seção anterior.

	Precisão			
Completude	Forte	Fraca	Forte Final	Fraca Final
Forte	*Perfeita* P	*Forte* S	*Finalmente Perfeita* $\Diamond\mathsf{P}$	*Finalmente Forte* $\Diamond\mathsf{S}$
Fraca	Q	*Fraca* W	$\Diamond\mathsf{Q}$	*Finalmente Fraca* $\Diamond\mathsf{W}$

Tabela 3.1: Classes de Detectores de Falhas

Um detector de falhas é dito *perfeito* se satisfaz completude forte e precisão forte. O conjunto de todos esses detectores de falhas, chamado de classe de detectores de falhas perfeitos, é denotado por P. Definição similar surge para cada classe. A classe S, denominada *forte*, é obtida combinando-se precisão fraca e completude forte. A classe $\Diamond\mathsf{P}$,

denominada *finalmente perfeita*, é obtida combinando-se precisão forte final e completude forte. A classe ◊S, denominada *finalmente forte*, é obtida combinando-se precisão fraca final e completude forte. A classe Q é obtida combinando-se precisão forte e completude fraca. A classe W, denominada *fraca*, é obtida combinando-se precisão fraca e completude fraca. A classe ◊Q é obtida combinando-se precisão forte final e completude fraca. Finalmente, a classe ◊W, denominada *finalmente fraca*, é obtida combinando-se precisão fraca final e completude fraca.

Um detector de falhas *D'* é redutível ao detector de falhas *D* se existe um algoritmo distribuído que pode transformar *D* em *D'* [6]. Neste caso, *D'* é dito "mais fraco"que *D*; dado este algoritmo de redução, qualquer problema que possa ser resolvido com *D'*, pode ser resolvido com *D*. Para ilustrar esta afirmação, suponhamos um algoritmo *A* que requer um detector de falhas *D'*, mas somente *D* está disponível. Juntamente com *A*, os processos executam um algoritmo *T*, que transforma *D* em *D'*. Sempre que *A* necessite que um processo *p* consulte seu detector de falhas, ao invés de efetuar esta consulta, *p* lê o valor corrente de saída que é mantido por *T*. Dois detectores de falhas são equivalentes se eles são redutíveis um ao outro. O conceito de redutibilidade se aplica também às classes de detectores de falhas. Dadas duas classes de detectores de falhas, C e C', se para cada detector de falhas *D* em C, existir um detector de falhas *D'* em C' tal que *D'* é redutível a *D*, então a classe C' é dita redutível à classe C, e é dita "mais fraca"que C. Desta forma, se um problema é resolvido utilizando a classe C', então é resolvido também utilizando a classe C de detectores de falhas. Se C' é redutível a C, e C é redutível a C', então C e C' são equivalentes. Usando o conceito de redutibilidade, é possível reduzir as oito classes de detectores de falhas, descritas na seção 3.2, em apenas quatro. Utilizando um algoritmo de redução no qual cada processo *p* periodicamente retorna seu valor de saída, as seguintes relações entre as classes de detectores de falhas são imediatas. Q é redutível à P, W é redutível à S, ◊Q é redutível à ◊P e ◊W é redutível à ◊S [6].

O detector de falhas mais fraco que resolve o problema do consenso em sistemas assíncronos é apresentado em [31]. Este detector é o $\Diamond W_0$, o detector de falhas mais fraco em ◊W. A grosso modo, $\Diamond W_0$ satisfaz as propriedades de ◊W, e nenhuma outra proprie-

dade. Assim, $\Diamond W_0$ é necessário e suficiente para resolver consenso em sistemas assíncronos com a maioria dos processos corretos. Em [6], Chandra e Toueg consideram este resultado como a maior evidência da importância de $\Diamond$W para a computação distribuída tolerante a falhas em sistemas assíncronos. Como o consenso e outros problemas clássicos de sistemas distribuídos, como difusão atômica, são equivalentes, $\Diamond W_0$ é tambem o detector de falhas mais fraco que resolve estes outros problemas.

Como já visto, a classe $\Diamond$W satisfaz as propriedades de completude fraca e precisão fraca final. Na prática não é necessário que estas duas condições sejam permanentemente satisfeitas. Quando se resolve um problema que termina, como o consenso, é suficiente que as condições sejam respeitadas até o tempo no qual o algoritmo atinja seu objetivo. Quando se está resolvendo um problema que não termina, como difusão atômica, é suficiente que estas propriedades sejam satisfeitas até que algum progresso ocorra, por exemplo, até que os processos corretos entreguem algumas mensagens.

Outra característica desejável para $\Diamond$W está relacionada a duas propriedades da aplicação. Se uma aplicação utiliza um detector de falhas com as propriedades de $\Diamond$W, e tal detector falha constantemente em satisfazer tais propriedades, a *aplicação* pode perder *progresso (liveness)*, mas não *segurança (safety)*. Progresso e segurança são duas propriedades desejáveis à aplicação. Um detector de falhas pode interferir negativamente no progresso de uma aplicação se permitir que um processo fique indefinidamente esperando uma resposta de outro processo que falhou. Já a segurança da aplicação pode ser afetada se o detector de falhas informa incorretamente uma suspeita. Deste modo, os processos podem ser prevenidos de decidir, mas nunca decidem por valores diferentes, ou por valores não permitidos. Similarmente, para o problema da difusão atômica, processos podem parar de entregar mensagens, mas nunca entregar fora de ordem.

Com relação à implementação prática da abstração $\Diamond$W, [6] explica que muitos detectores de falhas são baseados em *timeouts*, e que em um sistema assíncrono, uma seqüência ilimitada e prematura de *timeouts* pode fazer com que todos os processos corretos sejam repetidamente adicionados e removidos da lista de suspeitos de todos os processos corretos, violando a precisão de $\Diamond$W. Porém, em vários sistemas na prática, incrementar o

valor do *timeout* a cada engano assegura que em algum momento não haverá mais *timeout* prematuro em pelo menos um processo correto p. Isto assegura a propriedade de precisão de $\Diamond$W.

Larrea, Fernández e Arévalo apresentam em [32] um algoritmo que aumenta o intervalo de *timeouts*, satisfazendo a propriedade de *precisão fraca final*. Esta propriedade requer que, finalmente, algum processo correto nunca é suspeito por nenhum processo correto. Se a identidade do processo correto é conhecida, então basta que todos os processos aumentem seu *timeout* quando suspeitarem deste processo. Porém, se não há conhecimento sobre a corretude de nenhum processo, é preciso utilizar outro método. Para ilustrar o algoritmo, não são considerados agentes, mas apenas processos, que monitoram uns aos outros através de *pings*, e não de *heartbeats*. Os processos são organizados em uma topologia do tipo *anel*. A topologia é conhecida por todos os processos. Um processo, denominado "candidato a líder" e identificado por $cand_i$, é pré-determinado. Quando um processo p suspeita de $cand_i$, o valor do *timeout* de p para $cand_i$ é incrementado de uma unidade, e p considera como *novo* candidato a líder o sucessor de $cand_i$ na topologia anel. Desta forma, se p der uma volta completa na topologia anel, voltando ao candidato inicial $cand_i$, o *timeout* já estará incrementado, e, ocorrendo isso repetidamente, p, em algum momento, não vai mais suspeitar de $cand_i$, garantindo a propriedade de precisão fraca final. A prova formal pode ser encontrada em [32].

3.3 Qualidade de Serviço de Detectores de Falhas

Em [33], Chen, Toueg e Aguilera especificam a QoS (Quality of Service - Qualidade de Serviço) de detectores de falhas quantificando duas métricas: velocidade (quão rápido o detector de falhas detecta falhas reais) e precisão (quão bem ele evita enganos). Velocidade está relacionada à detecção de processos que falharam, e precisão está relacionada ao possível engano com relação à detecção de falhas de processos que não falharam. Estas métricas são propostas para sistemas com comportamento probabilístico, ou seja, para sistemas nos quais atrasos e perdas de mensagens seguem alguma distribuição de probabilidade. Um único tipo de falha é permitido, a falha por parada (*crash*). O detector de

falhas pode ser lento, isto é, ele pode levar muito tempo para suspeitar de um processo que falhou e, por outro lado, pode cometer enganos, isto é, ele pode suspeitar erroneamente de alguns processos que estão corretos. Tais enganos não são necessariamente permanentes: o detector pode parar de suspeitar de um processo correto mais tarde. Para ser útil, o detector tem que ser razoavelmente rápido e preciso. Em muitos casos, os detectores de falhas são especificados em termos de seus comportamentos finais. Estas especificações são apropriadas para sistemas assíncronos nos quais não há requisitos de tempo. Muitas aplicações, entretanto, possuem alguma restrição de tempo e, para tais aplicações, detectores de falhas com garantias finais não são suficientes. Por exemplo, um detector de falhas que suspeita de um processo uma hora após o processo ter falhado, pode ser usado para resolver consenso, porém o mesmo não é apropriado para uma aplicação que precisa resolver vários casos de consenso por minuto. Aplicações que possuem restrição de tempo requerem detectores de falhas que provêem qualidade de serviço com alguma garantia quantitativa de tempo.

A velocidade do detector de falhas é o tempo decorrente desde o momento quando um processo falhou até o tempo no qual o detector de falhas suspeita deste processo permanentemente. Esta métrica é chamada de *tempo de detecção*.

Medir a precisão de um detector de falhas não é uma tarefa trivial. Por exemplo, considere um sistema com dois processos, p e q, conectados por um *link* de comunicação com perda de mensagens, e um detector de falhas em q monitorando p. A saída deste detector é "p está correto" ou "p está falho", e ele pode alternar entre estas duas saídas de tempos em tempos. Para o propósito de medir a precisão do detector de falhas em q, suponha que p não falhou. Considere uma aplicação que questiona o detector em q em tempos aleatórios. Para tal aplicação, a medida natural de precisão é a probabilidade que, quando questionado em tempo aleatório, o detector em q indique corretamente que p está correto. Esta métrica, chamada de *probabilidade de precisão*, não é, entretanto, suficiente para uma completa análise da precisão do detector de falhas. Por exemplo, dois detectores podem possuir a mesma *probabilidade de precisão*, porém, o detector que possuir uma *taxa de engano* menor será mais útil para uma aplicação na qual todo engano

implique em uma interrupção cara. Para tal aplicação, a *taxa de engano* é uma importante métrica de precisão. O contrário também pode acontecer, isto é, a *taxa de engano* sozinha não é suficiente para caracterizar precisão: dois detectores de falhas podem ter a mesma *taxa de engano*, porém diferentes *probabilidades de precisão*.

Outra medida importante é a *duração do engano*, ou seja, quão rápido um detector de falhas corrige um engano que cometeu. Esta medida é importante para aplicações que operam em modo degradado enquanto processos são suspeitos incorretamente. Para estas aplicações, dois detectores de falhas podem ter a mesma *probabilidade de precisão* e a mesma *taxa de engano*, porém o detector que tiver a menor *duração de engano* será mais apropriado. Observa-se então que existem muitos aspectos diferentes que podem ser importantes para diferentes aplicações, e cada aspecto tem a correspondente métrica de precisão.

Em [33] são identificadas seis métricas de precisão, e a teoria de processos estocásticos é usada para quantificar a relação entre estas métricas. Duas métricas de precisão são selecionadas como primárias no sentido que elas não são redundantes e, juntas, elas podem ser usadas para derivar as outras quatro métricas de precisão. As métricas de QoS definidas em [33] são similares a algumas medidas de confiabilidade. Por exemplo, a métrica *probabilidade de precisão* é similar à medida de *disponibilidade* e a *duração de engano* é similar ao *tempo de recuperação*. Entretanto, existem diferenças significativas e é necessário ter cuidado para não se confundir. Medidas de confiabilidade se referem a falhas do sistema, enquanto as métricas de QoS se referem a enganos do detector de falhas, o qual monitora falhas do sistema. Por isso, nem todas as métricas de precisão têm correspondência com métricas de confiabilidade. Estas métricas de QoS são aplicáveis a todos os detectores de falhas, da maneira como forem implementados.

Com base na discussão acima, Chen, Toueg e Aguilera apresentam em [33] três métricas primárias e quatro métricas derivadas de QoS de detectores de falhas, descritas a seguir.

3.3.1 Métricas de QoS de Detectores de Falhas

Para ilustrar as métricas de QoS, algumas suposições são necessárias. Considera-se um sistema com dois processos, p e q. Assume-se que o detector de falhas em q monitora p e que q não falha. O tempo é contínuo e varia de 0 a ∞. A saída do detector em q no tempo t é S ou T, que significa que q suspeita ou confia em p no tempo t, respectivamente. Uma transição ocorre quando a saída muda: *transição-S* ocorre se a saída do detector de falhas em q muda de T para S; *transição-T* ocorre se a saída do detector em q muda de S para T. Assume-se que existe um número finito de transições durante algum intervalo finito de tempo. O sistema se comporta de forma probabilística. Os detectores de falhas alcançam um *estado fixo*, ou seja, q suspeita de p, ou não. As métricas de QoS propostas referem-se ao comportamento do detector de falhas após ele alcançar tal *estado fixo*.

Desta forma, as três métricas primárias são:

Tempo de detecção *(Detection time - T_D)*: Informalmente, T_D é o tempo que decorre da falha de p até o tempo em que q suspeita de p permanentemente. Mais precisamente, T_D é o tempo que decorre do tempo em que p falha até o tempo em que a *transição-S* final, do detector de falhas em q, ocorre e não há mais transições posteriores.

O *tempo de detecção* é uma métrica de velocidade. As próximas seis métricas apresentadas estão relacionadas à precisão, e supõe-se execuções livres de falhas, isto é, execuções em que p não falha.

Tempo de repetição de engano *(Mistake recurrence time - T_{MR})*: Mede o tempo entre dois enganos consecutivos. Mais precisamente, T_{MR} é o tempo decorrido de uma transição-S até a próxima.

Duração de engano *(Mistake duration - T_M)*: Mede o tempo em que o detector de falhas leva para corrigir o engano. Mais precisamente, T_M é o tempo que decorre da *transição-S* até a próxima *transição-T*.

As quatro métricas derivadas são:

Taxa média de engano *(Average mistake rate - λ_M)*: Mede a taxa em que o detector de falhas comete enganos, isto é, o número médio de *transições-S* por unidade de tempo. Esta métrica é importante para aplicações com tempo de vida longo, nas quais cada

engano do detector resulta numa interrupção cara, como é o caso do serviço de grupo *group membership*.

Probabilidade de precisão *(Query accuracy probability - P_A)*: Mede a probabilidade em que a saída do detector de falhas está correta em um tempo aleatório. Esta métrica é importante para aplicações que interagem com o detector de falhas questionando-o em tempos aleatórios.

Muitas aplicações podem fazer progresso apenas durante *bons períodos*, ou seja, períodos nos quais o detector de falhas não comete enganos. Para estas aplicações, as duas métricas seguintes são importantes:

Duração de bom período *(Good period duration - T_G)*: Mede o tamanho do bom período. Mais precisamente, T_G é o tempo que decorre da *transição-T* até a próxima *transição-S*.

Para aplicações de vida curta, entretanto, a próxima métrica pode ser mais relevante. Suponha que a aplicação é iniciada em um tempo aleatório, em um *bom período*. Se a parte restante do *bom período* é longa o suficiente, a aplicação de vida curta estará apta a completar sua tarefa.

Duração restante do bom período *(Forward good period duration - T_{FG})*: Mede o tempo decorrente do tempo aleatório no qual q confia em p até o tempo da próxima *transição-S*.

Para detectores baseados em *timeout*, a probabilidade de *timeouts* prematuros tem sido usada como métrica de precisão, isto é, a probabilidade em que, quando o contador é iniciado, ele terá *timeout* prematuro no processo que está correto. Em [33], esta métrica não é considerada apropriada porque é específica da implementação e não é útil para aplicações a menos que seja usada juntamente com outra métrica específica de implementação, por exemplo, com que freqüência a aplicação é iniciada, se o contador é iniciado em intervalos regulares ou variáveis, se o período de *timeout* é fixo ou variável, etc. As seis métricas de precisão identificadas acima não se referem a características específicas de implementação.

3.4 Detectores Quiescentes

Considere dois processos p e q que não falham conectados por um enlace não confiável, isto é, que perde mensagens. Um dos problemas básicos na comunicação de processos é a construção de um enlace confiável sobre este enlace não confiável. Uma solução clássica para este problema envolve retransmissões e confirmações. Retransmissões permitem tolerar perdas de mensagens, enquanto confirmações permitem que as retransmissões não ocorram eternamente: quando o receptor recebe uma mensagem, ele envia uma confirmação *(ack)* relacionada a tal mensagem, e para cada mensagem que um processo queira enviar, ele a reenvia repetidamente até receber um *ack* relacionado a ela. Este protocolo simples é *quiescente*, no sentido que, após algum tempo, não há mais transmissões da mensagem ou sua confirmação.

Soluções quiescentes para este problema não podem ser implementadas caso haja uma modificação no modelo: permitindo que o receptor falhe [34]. Para resolver este problema, Aguilera, Chen e Toueg mostram em [34] que é possível utilizar detectores de falhas, e que a classe de detectores de falhas mais fraca que resolve este problema é a dos detectores *finalmente perfeitos*. Porém, detectores desta classe não podem ser implementados no modelo de comunicação que utiliza *links* não confiáveis e que permitem falhas em processos.

Para resolver o problema, Aguilera, Chen e Toueg propõem em [34] um novo tipo de detectores de falhas, detectores de falhas baseados em *heartbeats*. Um detector de falhas baseado em *heartbeats* retorna, para cada processo p_i, um vetor HB_i[1..n] de contadores não decrescentes, satisfazendo as seguintes propriedades.

HB-completude: se p_j falha, então HB_i[j] pára de incrementar.

HB-precisão: se p_j está correto, então HB_i[j] nunca pára de incrementar.

A implementação de detectores de falhas baseados em *heartbeats* é não-quiescente, entretanto, estes detectores permitem encapsular uma parte não-quiescente de um protocolo de comunicação.

Além das primitivas de controle de falhas, o protocolo quiescente deve fornecer à aplicação as funções de envio e recebimento de mensagens, *send()* e *receive()* respectiva-

mente. Por exemplo, para enviar uma mensagem a um processo p_j, o processo p_i deve utilizar a primitiva *send()* do protocolo quiescente, que só repassa a mensagem ao processo destino se o mesmo não estiver falho, ou seja, se HB_i[j] estiver incrementando. As retransmissões acontecem até p_j transmitir um *ack()* referente a tal mensagem ou até que HB_i[j] páre de incrementar, caso que ocorre quando p_j está falho. Similarmente, a primitiva *receive()* do protocolo quiescente notifica a camada superior que uma nova mensagem chegou.

3.5 O *Framework* SNMP-FD

Wiesmann, Urbán e Défago apresentam em [7] o *framework* SNMP-FD - um serviço de detecção de falhas baseado em SNMP. A transmissão de mensagens do detector de falhas é feita utilizando mensagens SNMP. As mensagens incluem, por exemplo, *heartbeats* e notificações na mudança do estado de execução de um processo. Elas contém informações das diferentes MIBs *(Management Information Base)* do sistema. As seguintes informações são incluídas: o identificador do *host* do agente monitorado, o intervalo entre *heartbeats*, o identificador e o estado de cada processo monitorado, o intervalo entre *heartbeats* requerido (que precisa ser maior que o intervalo corrente), e o contador de *heartbeats*.

As MIBs mais relevantes para o serviço de detecção de falhas são descritas a seguir.

Host Resource MIB [8]: MIB padrão para a obtenção de informações sobre recursos em um *host*. Os recursos que podem ser monitorados via esta interface incluem dispositivos, sistemas de arquivos, *softwares*, e processos em execução. Este último item é o de interesse para o serviço de detecção de falhas. Processos são identificados por um inteiro de 32 *bits* sem sinal.

Target and Notification MIBs [9]: Estas MIBs especificam mecanismos de transporte, endereços de destino e filtros de mensagens SNMP.

Alarm Description MIB [10]: Descreve alarmes nos agentes. Um alarme é definido por uma indicação de falha persistente. Esta MIB define tabelas para descrever possíveis tipos de alarmes, as notificações associadas e os alarmes ativos atualmente.

Event MIB [11]: pode ser usada para descrever eventos que podem disparar noti-

ficações se certas condições são satisfeitas.

O serviço pode operar em dispositivos e serviços que tenham um agente SNMP: equipamentos de rede, servidores, serviços administrativos e ferramentas de gerenciamento.

A arquitetura do SNMP-FD é descrita a seguir. O sistema consiste de um conjunto de *hosts*. Cada *host* pode executar processos monitorados e processos monitores. Processos monitores capturam informações sobre processos monitorados. Um processo pode ser monitor, monitorado ou ambos. Cada *host* executa um agente SNMP local para implementar um processo monitor. Processos monitorados e monitores são registrados no agente SNMP local. Os agentes são responsáveis por troca de informações de monitoração. Em caso de falha do agente, o mesmo é automaticamente reinicializado pelo sistema operacional. Agentes possuem uma cópia de seu estado interno em memória estável. Na reinicialização, o agente segue a convenção SNMP para agentes e envia uma mensagem de início para todos os alvos registrados.

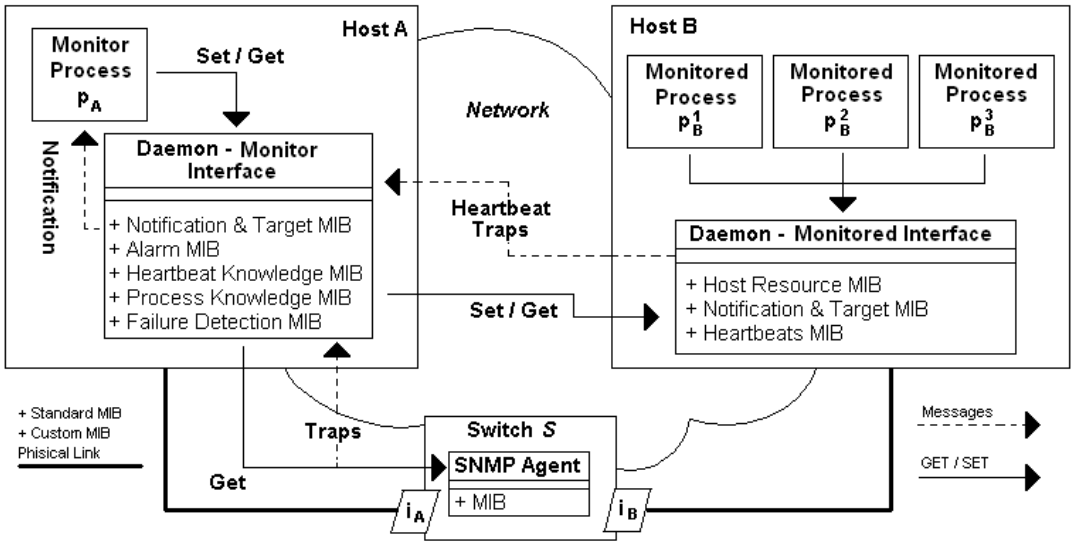

Figura 4.1: Arquitetura Geral do *Framework* SNMP-FD [7].

A figura 4.1 ilustra a arquitetura geral do SNMP-FD. Ela mostra dois hosts A e B. A contém um processo p_A que monitora três processos em B: p_{1B}, p_{2B} e p_{3B}. A rede entre A e B usa um *switch* S que disponibiliza informações de gerenciamento usando uma interface SNMP. Em S, A está conectado na interface i_A e B na interface i_B. Cada agente SNMP é implementado como um *daemon*, d_A e d_B, que rodam nos *hosts* A e B, respectivamente. Cada *daemon* funciona como monitor e monitorado. Todos os processos

monitorados (p_{1B}, p_{2B} e p_{3B}) são registrados no *daemon* d_B. Outras particularidades da figura 4.1 são explicadas ao longo desta seção.

O agente SNMP no *switch* S é configurado de acordo com suas MIBs *Notification* e *Target* para enviar *traps* para o agente no *host A* em caso de eventos, tal como a interface i_B parar ou reiniciar. Uma *trap* de *link* parado a respeito da interface i_B indica que o *host B* está inalcançável na rede (assumindo que não há *links* redundantes). Uma *trap* de *link* funcionando indica que a interface de rede subiu. Em caso de pequenas desconexões, por exemplo, reinicialização da máquina, a *trap* de *link* parado não deve ser gerada. Outras *traps*, por exemplo sinalizando congestionamento, podem ser usadas para ajustar parâmetros de detecção de falhas.

Na interface monitorada, o agente SNMP monitora processos locais e notifica agentes remotos em caso de falhas (*crashes*). A interface inclui quatro MIBs: *Host Resource, Notification, Target* e *Heartbeat MIB*. O agente implementa parte da *Host Resource MIB* padrão. O campo de interesse é o que contém o estado de execução dos processos. A *Heartbeat MIB* é específica do framework SNMP-FD. Ela contém as descrições dos diferentes *heartbeats* em forma de tabela. É necessária uma tabela adicional porque os parâmetros de *heartbeats* não cabem em tabelas das MIBS *Notification* e *Target*. Cada linha desta tabela contém o identificador do processo monitorado, e informações sobre os próprios *heartbeats*: intervalo entre *heartbeats* e contador de *heartbeats* enviados. Os destinos dos *heartbeats* são selecionados usando os filtros da *Notification MIB*.

A interface *A*, que monitora *B*, inclui cinco MIBs: *Notification, Target, Heartbeat Knowledge, Process Knowledge,* e *Failure Detection MIB*. O monitor inclui as MIBs *Notification* e *Target* padrão, assim como a interface monitorada. A *Heartbeat Knowledge MIB* é específica do *framework* SNMP-FD e contém estatísticas sobre os *heartbeats* remotos recebidos, como contador de *heartbeats*, taxa de recebimento e número de *heatbeats* perdidos. Esta informação é armazenada em uma tabela somente de leitura, com uma linha para cada origem de *heartbeats*. A *Process Knowledge MIB* também é específica do *framework* SNMP-FD e contém informações de monitoração de processos remotos. Esta informação é armazenada em uma tabela somente de leitura, onde cada linha é um processo re-

moto conhecido do agente. Para cada processo, ela contém o *hostname*, o identificador do processo, o último estado de execução e um *timestamp* da última atualização. Estas informações são atualizadas dinamicamente por mensagens SNMP. A interface A inclui ainda parte da *Alarm MIB*. Esta MIB contém descrições das notificações associadas às mudanças no estado dos processos. Contém também uma tabela de alarmes ativos: processos suspeitos ou falhos. Assim, ferramentas podem analisar informações de detecções de falhas sem conhecer detalhes do serviço de detecção.

Para validar a arquitetura apresentada, Wiesmann, Urbán e Défago implementaram o serviço SNMP-FD. A implementação foi feita em Java e conta com o sistema SNMP4J, que é aberto. Este serviço implementa as MIBs apresentadas, juntamente com uma biblioteca com a qual aplicações podem se registrar para receber notificações de mudanças nos estados de execução dos processos. Foi implementado um detector de falhas simples baseado em *timeouts*. Este detector assume estados para processos conforme proposto por Gorender [35]. O detector de falhas retorna três possíveis valores para cada processo: confiável, suspeito ou falho. Um processo é considerado confiável se o *heartbeat* foi recebido num intervalo de tempo menor que o *timeout* estipulado pelo detector de falhas. Após este intervalo, se o *heartbeat* não for recebido, o processo é considerado suspeito. Um processo é considerado falho se uma informação de falha for recebida do agente local que o monitora.

Capítulo 4

Um Serviço Baseado em SNMP para Detecção de Falhas na Internet

Este trabalho apresenta um serviço baseado em SNMP e serviços Web para detecção de falhas de processos de sistemas distribuídos na Internet. A seção 4.1 descreve a arquitetura e os componentes do detector de falhas, e a seção 4.2 apresenta os experimentos realizados.

4.1 Arquitetura e Componentes do Sistema

Nesta seção é apresentada a arquitetura do serviço de detecção de falhas e são detalhados os seus componentes. O serviço monitora o estado de execução de processos. Aplicações distribuídas podem utilizar o detector de falhas para consultar informações sobre o estado de seus processos. Essas informações são obtidas através de comandos SNMP, que podem ser executados de duas maneiras. Na primeira, o comando SNMP é executado diretamente na rede local, e é interpretado pelo agente SNMP local. Na segunda, quando o processo monitorado e a aplicação estão executando em redes distintas, o comando SNMP é enviado ao agente SNMP remoto através de serviços Web. Também através de serviços Web, o agente SNMP remoto responde ao comando SNMP da aplicação, enviando as informações sobre o estado do processo monitorado.

Os serviços Web são responsáveis por viabilizar a monitoração de processos da aplicação que possam estar executando fora da rede local, além de permitir que uma aplicação que esteja fora da rede local consulte o estado dos processos que estejam executando na própria rede local. O serviço Web foi implementado neste trabalho utilizando-se a lingua-

gem Python [27]. O módulo de serviços Web interage com um agente SNMP através de comandos SNMP, obtendo informações sobre o estado de execução dos processos monitorados, e também interage com a aplicação, recebendo requisições e devolvendo informações a respeito do estado de processos que estão executando fora da rede local. Módulos de serviços Web que executam em redes locais diferentes interagem entre si, com o intuito de trocar informações sobre os processos monitorados. É necessária a execução de um módulo de serviços Web para cada rede local onde existe ao menos um processo monitorado por uma aplicação externa à esta rede local, ou para cada rede local onde existe uma aplicação que monitora o estado de processos em outras redes locais.

A figura 4.1 ilustra a arquitetura da ferramenta. Na figura são ilustrados quatro processos monitorados pelo detector: P_{A1}, P_{A2}, P_{B1} e P_{B2}. Neste exemplo, os processos estão executando em *hosts* diferentes. Os *hosts* $HOST_{A1}$, $HOST_{A2}$ e $HOST_{A3}$ pertencem à LAN_A, e os *hosts* $HOST_{B1}$, $HOST_{B2}$ e $HOST_{B3}$ pertencem à LAN_B. Cada rede local possui ainda um agente SNMP que mantém a *Process Status (PS) MIB*, um monitor, uma interface com a Internet através de serviços Web, e uma aplicação usuária do serviço de detecção de falhas.

A aplicação é a entidade que utiliza o sistema para monitorar o estado de execução de seus processos, tanto dentro da rede local, quanto na Internet. O sistema fornece uma interface de comandos SNMP para a aplicação. Deste modo, o usuário do sistema pode implementar a aplicação conforme suas necessidades, desde que a mesma tenha acesso a comandos do protocolo padrão TCP/IP de gerência de redes, o SNMP. Por exemplo, observe o cenário da figura 4.1, e suponha que a aplicação da LAN_A necessita conhecer o estado do processo P_{B1}. A aplicação deve utilizar o SNMP através dos serviços Web da LAN_A para requisitar o estado do processo P_{B1} ao módulo de serviços Web da LAN_B. Deve ser informada a identificação do processo P_{B1}. O módulo de serviços Web da LAN_B consulta o agente SNMP local e retorna o estado do processo P_{B1} ao módulo de serviços Web da LAN_A. O módulo de serviços Web da LAN_A repassa este resultado para a aplicação. Desta forma, não é necessário que as MIBs tenham conteúdo sincronizado, pois o módulo de serviços Web busca as informações na Internet de acordo com a demanda da

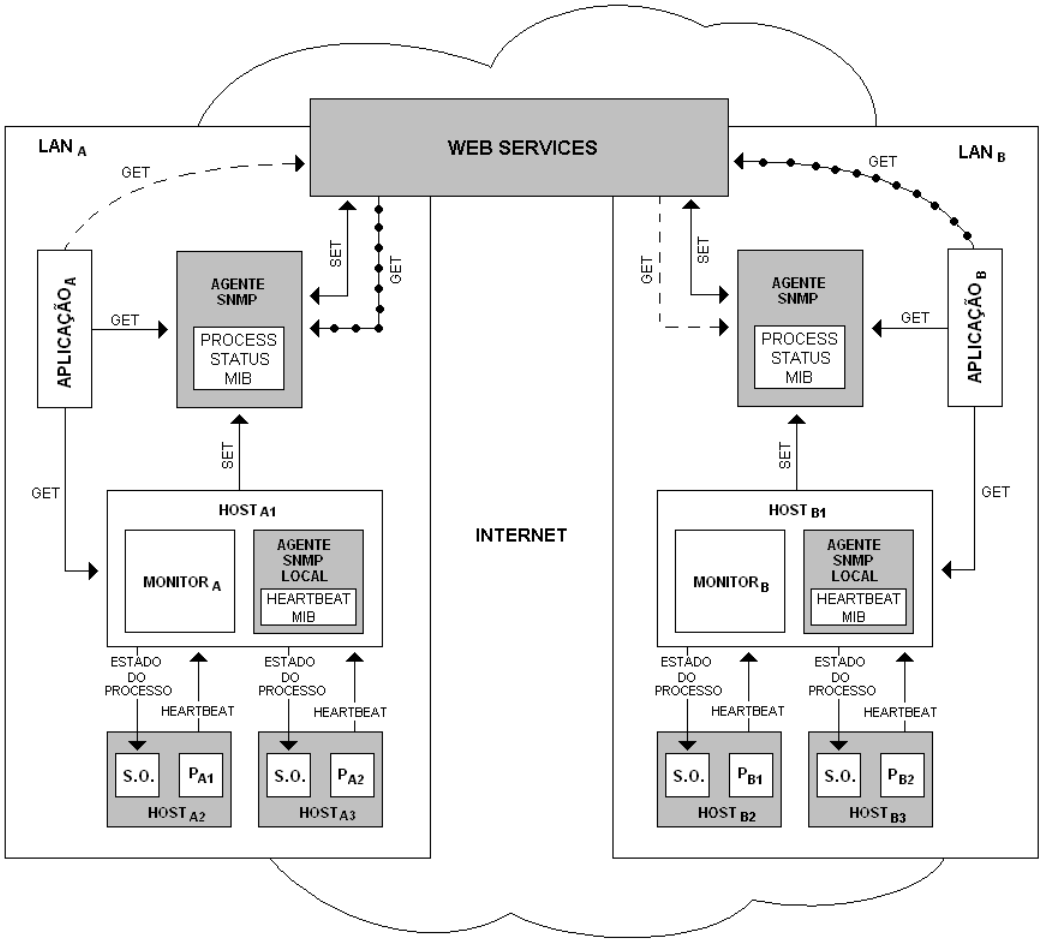

Figura 4.1: Arquitetura da ferramenta.

aplicação. Observando a mesma figura 4.1, e supondo que a aplicação da LAN_A necessita conhecer o estado do processo P_{A1}. Basta a aplicação acessar o módulo SNMP para obter o estado do processo.

Processos monitorados são processos de aplicação. Os três estados possíveis para os processos são: *sem-falha*, *suspeito* ou *falho*. Para ser monitorado, um processo precisa estar registrado no sistema. O processo deve incluir uma biblioteca que inicia uma *thread* para enviar *heartbeats* periodicamente ao monitor. O monitor continuamente calcula se esses *heartbeats* são recebidos antes do limite de *timeout*. Se estes *heartbeats* são recebidos pelo monitor dentro do intervalo de *timeout*, então o processo é dito sem-falha. Se os *heartbeats* não são recebidos pelo monitor dentro do intervalo de *timeout*, então o processo é dito suspeito. Ao ser considerado suspeito, o monitor verifica a existência do PID *(Process ID)* do processo no sistema operacional do *host* no qual o processo está sendo executado.

No sistema operacional Linux, isto pode ser feito, por exemplo, através do comando *ps*. Se o sistema operacional não retorna o PID do processo, então o processo é considerado falho, visto que o mesmo não existe mais na visão do sistema operacional. O *timeout* mencionado anteriormente é calculado utilizando uma variação do algoritmo utilizado pelo TCP para determinar o RTO $RetransmissionTimeout$. Este algoritmo é detalhado adiante.

O agente SNMP mantém na PS-MIB informações de estados de processos, e é responsável pela interface SNMP disponibilizada pelo sistema para interagir com aplicações usuárias. É através desta MIB que a aplicação conhece o estado de seus processos que estão executando na rede local. A PS-MIB também responde a requisições SNMP vindas dos serviços Web, ou seja, vindas da Internet. A PS-MIB é uma tabela com duas entradas, uma para identificar os processos atraves de endereço IP, porta (TCP ou UDP), e PID, e outra entrada para armazenar o estado de cada processo, que pode ser *sem-falha*, *suspeito* ou *falho*. Cada rede local, que possui ao menos um processo monitorado, deve ter um agente SNMP que mantém a PS-MIB. Se uma rede é utilizada apenas para monitorar processos que estão executando em outras redes locais, então não há necessidade de execução deste agente SNMP. Neste caso, o acesso é feito diretamente através dos serviços Web. A especificação ASN.1 da PS-MIB é descrita no apêndice 1.

O sistema pode ainda ser executado com uma opção de envio de notificações por parte da PS-MIB local para todas as outras redes locais, através dos serviços Web. Sendo assim, a PS-MIB além de armazenar as informações relacionadas aos processos locais, também armazena infomações dos processos remotos. Com essa opção, a MIB é atualizada tanto pelo módulo monitor, quanto pelos serviços Web.

O monitor é o módulo responsável por monitorar os processos pertencentes à aplicação na rede local. Para isto, utiliza uma estratégia baseada no envio de *heartbeats* por cada processo monitorado. O monitor também consulta o sistema operacional para monitorar o identificador de cada processo no sistema operacional. O monitor atualiza a MIB quando um novo processo se registra no sistema (através da biblioteca de envio de *heartbeats*), ou quando há mudança no estado de um processo. É necessária a execução de um monitor

em cada rede local onde exista ao menos um processo monitorado. Se uma rede local é utilizada apenas como estação de monitoração de processos que estão executando em outras redes locais, então não há necessidade de execução do módulo monitor. O monitor recebe *heartbeats* de todos os processos monitorados em um intervalo estipulado. Se o monitor não recebe um *heartbeat* de um processo sem-falha dentro de um tempo limite calculado, então o estado deste processo é atualizado para suspeito. O cálculo deste tempo limite é realizado utilizando-se uma variação do algoritmo utilizado no TCP para determinar o RTO *(Retransmission Timeout)* [36] [37], conforme a expressão abaixo.

$$Diferenca = Tempo\_hb_i - Tempo\_hb_{i-1}$$

$$Media_i = 0.9 * Media_{i-1} + 0.1 * Diferenca$$

$$Desvio_i = 0.9 * Desvio_{i-1} + 0.1 * |Media - Diferenca|$$

$$TempoLimite = Media_i + 4 * Desvio_i$$

Nesta expressão, $Tempo\_hb$ é a hora local no momento em que um *heartbeat* é recebido. $Diferenca$ corresponde à diferença entre a hora local no momento em que um *heartbeat* é recebido e a hora local correspondente ao recebimento do *heartbeat* anterior. $Media$ acumula o valor médio de $Diferenca$. Neste caso é dado um peso maior ao valor médio histórico, para que um único atraso pontual no valor da variável $Diferenca$ não interfira intensamente no valor do tempo limite. Isto permite uma redução na taxa de engano. $Desvio$ corresponde ao desvio médio, uma aproximação do desvio padrão. Quanto maior o ganho para $Desvio$, maior será o valor do tempo limite. $TempoLimite$ corresponde ao valor que é utilizado para determinar se um processo é suspeito ou não. Os valores iniciais de $Media$ e $Desvio$ são constantes atribuídas conforme o intervalo entre *heartbeats* enviados por cada processo. O cálculo do tempo limite se adapta ao intervalo entre *heartbeats*, sendo desnecessária a atribuição de um valor fixo. Desta forma, é possível ter processos enviando *heartbeats* com intervalos diferentes.

O monitor também mantém uma MIB, a HB-MIB, que armazena dados estatísticos sobre os *heartbeats*, como intervalo médio de recebimento, desvio padrão e hora do recebimento do último *heartbeat*. Esses dados podem ser consultados pela aplicação através de comandos SNMP. A especificação ASN.1 da HB-MIB é descrita no apêndice 2.

As principais contribuições que diferenciam a arquitetura proposta do trabalho relacionado SNMP-FD são descritas na sequência. A principal contribuição é a detecção de falhas de processos na Internet, e não somente em redes locais. Isto é feito através de serviços Web funcionando como gateway para o protocolo de gerência SNMP. Outra característica da arquitetura proposta é a utilização do SNMP apenas como interface e base de informações. O SNMP-FD utiliza um agente executando em cada host, enquanto o trabalho apresentado utiliza apenas um agente por rede local, independente do número de hosts que a compõem. O timeout utilizado para detectar suspeitas de processos não é fixo; a estratégia utilizada para este cálculo é análoga à estratégia utilizada pelo TCP para calcular o RTO. Outra característica que diferencia a arquitetura proposta do SNMP-FD é a implementação utilizando a linguagem C; O SNMP-FD é implementado em JAVA.

4.2 Avaliação do Sistema

O sistema foi implementado conforme segue. Processo monitorado, monitor, agente SNMP e aplicação foram implementados utilizando-se a linguagem C. No módulo de serviços Web, tanto cliente quanto servidor foram implementados utilizando-se a linguagem Python. O cliente utiliza a biblioteca *xmlrpclib*, que se comunica através de XML transportado via HTTP. Com essa biblioteca, o cliente pode acessar métodos com parâmetros em servidores remotos. O servidor utiliza a biblioteca *SimpleXMLRPCServer*, que provê um arcabouço básico para implementação de servidores escritos em Python. Para implementar agentes SNMP foi utilizado o arcabouço NET-SNMP, versão 5.2.4. NET-SNMP provê funcionalidades e ferramentas para implementação de agentes SNMP, como um agente extensível, biblioteca SNMP, e operações SNMP.

Para demonstrar o funcionamento do sistema, tanto em rede local quanto na Internet, foram elaborados cinco experimentos: Consumo de CPU (processos) vs. Intervalo entre *Heartbeats*, Consumo de CPU (monitor) vs. Intervalo entre *Heartbeats*, Tempo de Notificação vs. Nodo, Taxa de engano vs. Tempo, e Tempo de detecção vs. Intervalo entre *Heartbeats*. Para medir o tempo de notificação na Internet, o sistema foi executado no PlanetLab [38]. O PlanetLab é um laboratório mundial de pesquisa, que possui

nodos distribuídos ao redor do mundo e possibilita o desenvolvimento de sistemas distribuídos. Os gráficos apresentados foram gerados utilizando o software estatístico R [39]. R é um projeto GNU [40] e provê um ambiente computacional para estatística e geração de gráficos. A tabela 4.1 exibe a configuração dos computadores utilizados nos experimentos realizados em rede local. Os computadores e a rede foram utilizados de forma não-dedicada.

Recurso	**Configuração**
Fabricante	AMD
Processador	Dual-Core AMD Opteron(tm) Processor 2220 1 GHz
Número de processadores	4
Cache	1024 KB
Memória RAM	33.029.400 kB
Rede	1 Gbps
Sistema Operacional	GNU/Linux 2.6.27.21-vs2.3.0.36.4
SNMP	NET-SNMP (5.2.4)

Tabela 4.1: Configuração dos computadores da rede local.

Para medir o consumo de CPU, tanto do monitor quanto do processo, foram executados um monitor e um processo, em computadores diferentes, e a variação no intervalo entre *heartbeats* é feita no processo monitorado pelo monitor. A figura 4.2 demonstra o consumo de CPU do monitor. O eixo x representa o intervalo entre *heartbeats*, e o eixo y o consumo de CPU do monitor. Para cada ponto foram coletadas 50 amostras e o gráfico mostra a média e o intervalo de confiança de 95%. Durante todo o experimento, a média de consumo de CPU ficou entre 0 % e 0.3 %, o que caracteriza um baixo consumo de CPU por parte do monitor.

A figura 4.3 demonstra o consumo de CPU dos processos, variando o intervalo entre *heartbeats* do processo monitorado. O eixo x representa o intervalo entre *heartbeats*, e o eixo y o consumo de CPU do processo. Para cada ponto foram coletadas 50 amostras e o gráfico mostra a média e o intervalo de confiança de 95%. Para um intervalo entre

heartbeats de 1 milisegundo, a média de consumo de CPU do processo ficou em aproximadamente 2 %, caindo para 0.7 % quando o intervalo entre heartbeats é fixado em 2 milisegundos. Para intervalos entre heartbeats acima de 2 milisegundos, o consumo de CPU não superou 1 %.

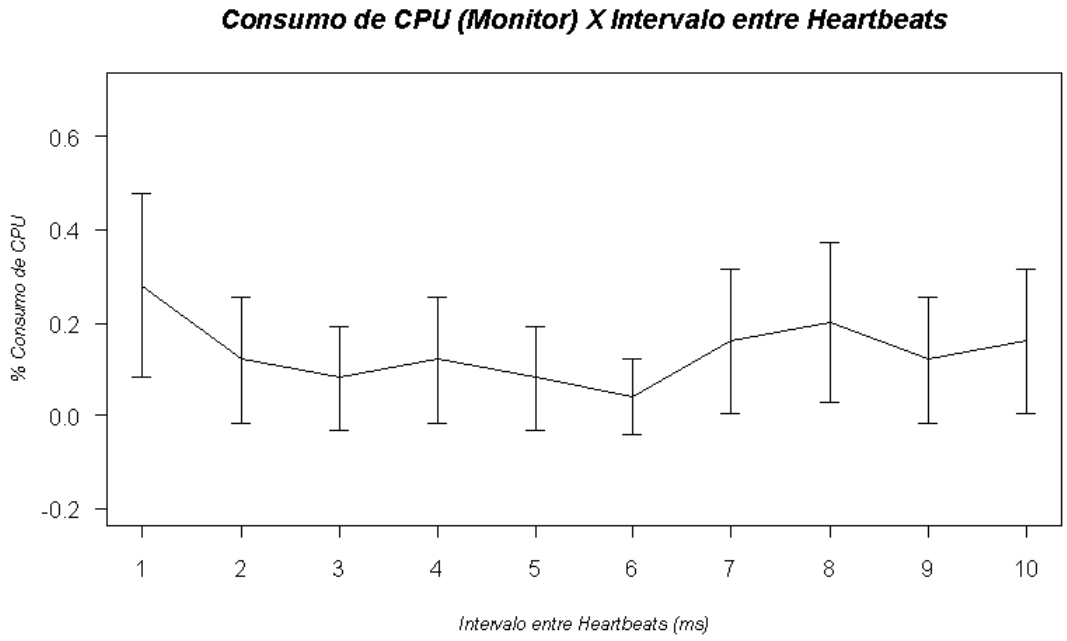

Figura 4.2: Consumo de CPU (monitor) X Intervalo entre *heartbeats* (ms).

A figura 4.4 demonstra o comportamento do sistema na Internet. O eixo x contém os nodos no Planet-lab, e o eixo y contém o tempo de notificação em segundos. Este tempo é calculado mantendo um servidor executando serviços Web no Brasil, no endereço planetlab1.c3sl.ufpr.br. Este servidor calcula a diferença de tempo entre o recebimento de uma mensagem e o recebimento da mensagem seguinte, ou seja, o cliente envia uma mensagem de aviso ao servidor através de serviços Web e logo em seguida força a suspeita de um processo monitorado, notificando a suspeita ao servidor logo em seguida. O intervalo de tempo entre essas duas mensagens determina o tempo de notificação. Para este teste foram coletadas 50 amostras por nodo, e o gráfico ilustra a média e o intervalo de confiança de 95% destas amostras. Os clientes executam em cinco nodos: Brasil (planetlab2.pop-mg.rnp.br), Estados Unidos (planetlab3.cs.uchicago.edu), Alemanha (mars.planetlab.haw-hamburg.de), Japão (node1.planet-lab.titech.ac.jp), e Nova Zelândia

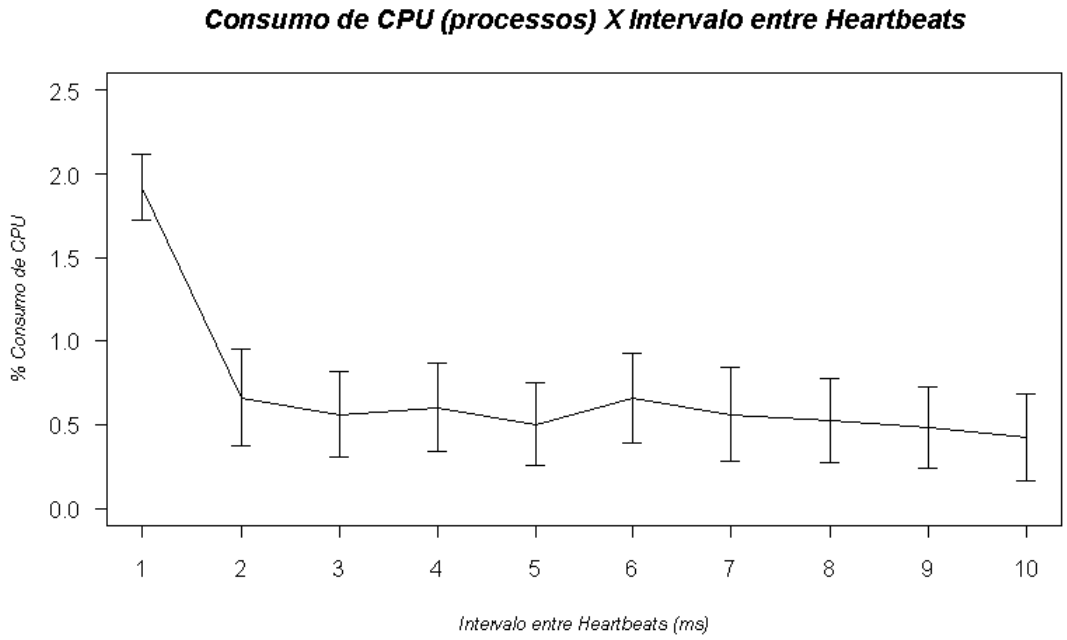

Figura 4.3: Consumo de CPU (processos) X Intervalo entre *heartbeats* (ms).

(planetlab-1.cs.auckland.ac.nz). O gráfico demonstra que o tempo de notificação respeita a distância física entre os nodos. O cliente que executa no nodo Brasil se encontra no mesmo país que o servidor, e para este nodo o tempo de notificação é de aproximadamente 0.1 segundos. Conforme a distância física entre os nodos aumenta, o tempo de notificação também aumenta, chegando a aproximadamente 1.1 segundos para cliente que executa na Nova Zelândia. Pode-se perceber também que o tempo de detecção de falhas em rede local, apresentado no gráfico anterior, é muito menor que o tempo de notificação na Internet, ou seja, o tempo necessário para notificar uma falha é consideravelmente maior do que o tempo necessário para se detectar a falha.

A figura 4.5 ilustra a taxa de enganos por segundo, a medida em que o tempo passa. O intervalo entre *heartbeats* foi fixado em 1ms para este experimento. Monitor e processo monitorado são executados em computadores diferentes numa mesma rede local. Para este experimento foi utilizado um processo que não falhou durante todo o período de teste, ou seja, toda falha detectada pelo monitor caracteriza um engano. Como é utilizado o algoritmo de RTO do TCP, espera-se que a quantidade de engano diminua com o passar do tempo, até estabilizar. Este resultado pode ser observado neste gráfico. Para este

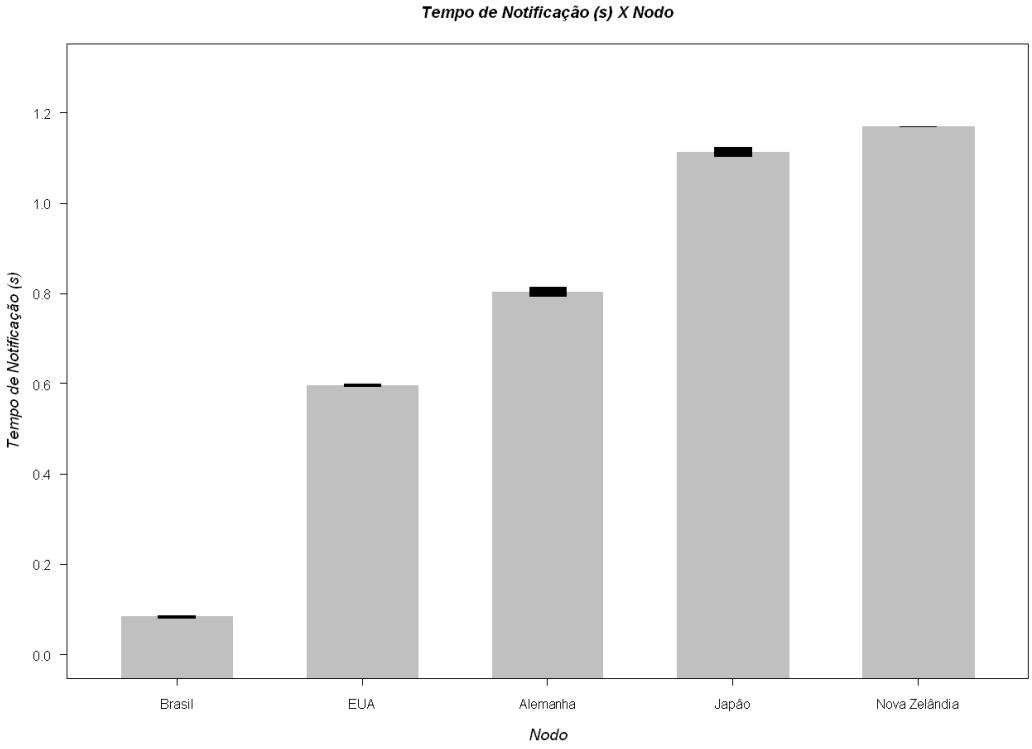

Figura 4.4: Tempo de notificação (s) X Nodo.

experimento foram coletadas 60 amostras para cada minuto, e o gráfico mostra a média e o intervalo de confiança de 95%. Observando o gráfico, percebe-se que a média da taxa de engano ficou em aproximadamente 1.9 enganos nos primeiros 3 minutos, caindo para 0.5 enganos por segundo no minuto 12, estabilizando a partir deste valor, permanecendo abaixo de 0.7 enganos por segundo no restante do tempo. A partir do minuto 27 até o restante do tempo, ocorre uma queda suave na taxa de engano, e nota-se também que o intervalo de confiança fica menor neste período, comprovando a tendência que o algoritmo tem de se adaptar às condições da rede.

A figura 4.6 ilustra o tempo de detecção de falhas em milisegundos, variando o intervalo entre *heartbeats* em milisegundos. Monitor e processo monitorado foram executados em um mesmo *host*, sendo desnecessária sincronia entre relógios. Neste experimento foi utilizado um processo que falha propositalmente. Tanto o tempo da falha, quanto o tempo no qual o monitor detecta essa falha são registrados. A diferença entre esses dois tempos é o chamado *tempo de detecção*. O gráfico de linha 4.6 reflete a média de 60 amostras, e as retas verticais ilustram o intervalo de confiança de 95 %. O eixo x contém o intervalo

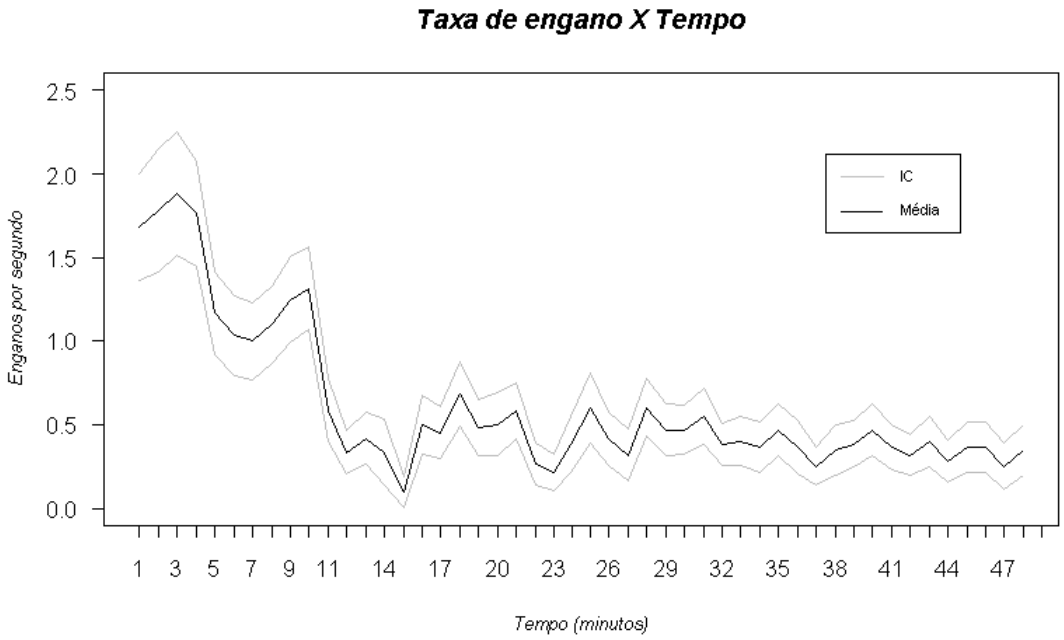

Figura 4.5: Taxa de engano X Tempo (m).

entre heartbeats, variando entre 7 e 21 milisegundos. No eixo y está representado o tempo de detecção em milisegundos. Teoricamente, o tempo de detecção nunca pode ser maior do que o intervalo entre heartbeats, e no gráfico pode-se perceber este comportamento. Para um intervalo entre heartbeats de 7 ms, o tempo de detecção é de aproximadamente 13 ms. Essa relação se mostra proporcional; a medida em que o intervalo entre heartbeats aumenta, o tempo de detecção também aumenta, sendo que no intervalo entre heartbeats de 21 ms, o tempo de detecção é de aproximadamente 26 ms.

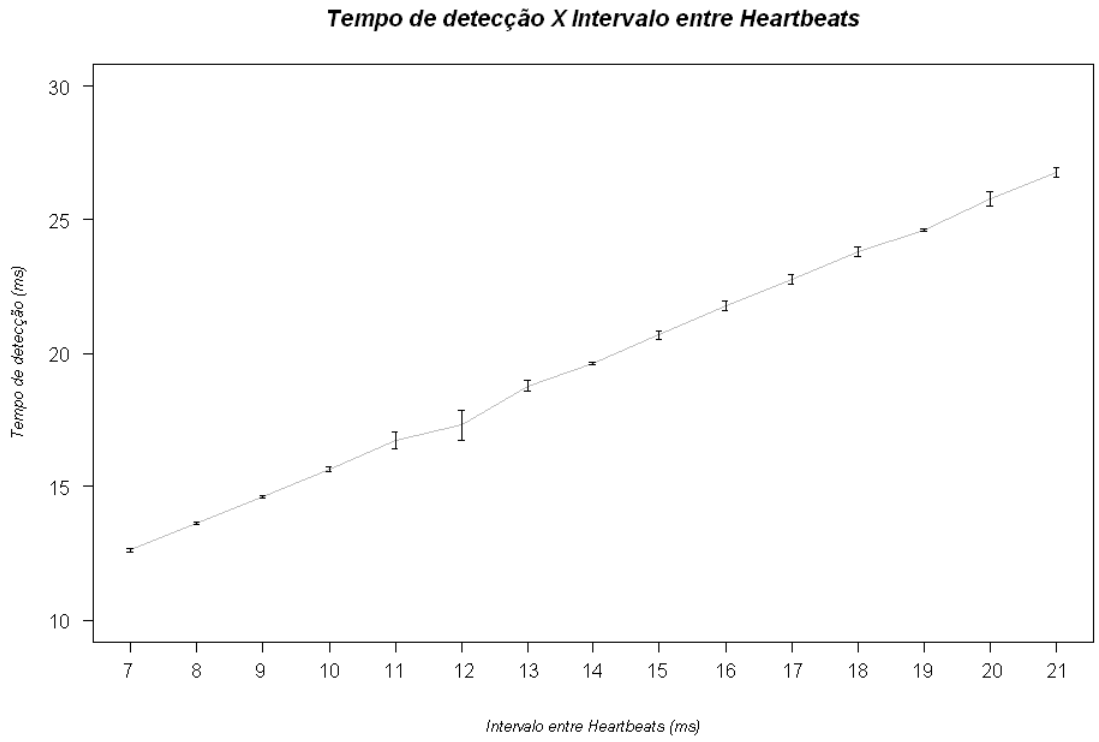

Figura 4.6: Tempo de detecção (ms) X Intervalo entre heartbeats (ms).

Capítulo 5

Conclusão

Neste trabalho apresentamos um serviço baseado em SNMP *(Simple Network Management Protocol)* e serviços Web para detecção de falhas em sistemas distribuídos. Foram também apresentados conceitos relacionados à gerência de redes de computadores e à utilização do protocolo SNMP e dos serviços Web na implementação do trabalho. Em seguida foi descrito o conceito de detectores de falhas, que são utilizados para detectar falhas de processos em sistemas distribuídos. Métricas de qualidade de serviço de detectores de falhas também foram apresentadas.

O trabalho seguiu com uma breve descrição da ferramenta SNMP-FD [7], relacionada a este trabalho. Na sequência foi apresentada a arquitetura implementada neste trabalho, um serviço para detecção de falhas em sistemas distribuídos, incluindo sua arquitetura. Este serviço pode ser utilizado por aplicações que necessitam manter seu funcionamento mesmo na presença de falhas de alguns de seus componentes, e podem utilizar o serviço para consultar o estado de execução de seus processos. Para avaliar a ferramenta foram realizados experimentos para medir consumo de CPU, tempo de detecção de falhas, tempo de notificação de falhas na Internet e taxa de engano.

Como trabalho futuro, será avaliada a funcionalidade desta ferramenta em serviços de grupo *(Group Membership)*.

Referências Bibliográficas

[1] A. Leinwand, K. Fang Conroy, *Network Management, A Pratical Perspective,* Vol. 2, Addison Wesley, 1998.

[2] W. Stallings, *SNMP, SNMPv2 and RMON,* Vol. 2, Addison Wesley, 1997.

[3] D. Harrington, R. Presuhn, B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks," *Request for Comments 3411 (RFC3411),* 2002.

[4] R. Presuhn, "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)," *Request for Comments 3416 (RFC3416),* 2002.

[5] M. Raynal, "Detecting Crash Failures in Asynchronous Systems: What? Why? How?," *Proc. Int. Conference on Dependable Systems and Networks (DSN'04),* Florence (Italy), 2004.

[6] T. Chandra, S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems," *Journal of the Association for Computing Machinery,* Vol. 43, No. 2, 1996.

[7] M. Wiesmann, P. Urbán, X. Défago, "An SNMP Based Failure Detection Service," *IEEE Symposion on Reliable Distributed Systems (SRDS),* 2006.

[8] S. Waldbusser, P. Grillo, "Host Resources MIB," *Request for Comments 2790 (RFC2790),* 2000.

[9] D. Levi, P. Meyer, B. Stewart, "SNMP Applications," *Request for Comments 2573 (RFC2573),* 1999.

[10] S. Chrisholm, D. Romascanu, "Alarm Management Information Base," *Request for Comments 3877 (RFC3877),* 2004.

[11] R. Kavasseri, B. Stewart, "Event MIB," *Request for Comments 2981 (RFC2981),* 2000.

[12] K. McCloghrie, M. Rose, "Management Information Base for Network Management of TCP/IP-based Internets," *Request for Comments 1066 (RFC1066),* 1988.

[13] K. McCloghrie, "Management Information Base for Network Management of TCP/IP-based Internets," *Request for Comments 1213 (RFC1213),* 1991.

[14] M. Rose, K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based Internets," *Request for Comments 1155 (RFC1155),* 1990.

[15] J. Case, M. Fedor, M. Schoffstall, J. Davin, "A Simple Network Management Protocol (SNMP)," *Request for Comments 1157 (RFC1157),* 1990.

[16] R. Presuhn, "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)," *Request for Comments 3418 (RFC3418),* 2002.

[17] K. McCloghrie, D. Perkins, J. Schoenwaelder, "Structure of Management Information Version 2 (SMIv2)," *Request for Comments 2578 (RFC2578),* 1999.

[18] B. Wijnen, R.Presuhn, K. McCloghrie, "View-Based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)," *Request for Comments 2275 (RFC2275),* 1999.

[19] U. Blumenthal, B. Wijnen, "User-Based Security Model (USM) for version 3 of the Simple Network Management Protocol SNMPv3," *Request for Comments 2574 (RFC2574),* 1999.

[20] The World Wide Web Consortium (W3C), http://www.w3.org/, Acesso em agosto/2009.

[21] Extensible Markup Language (XML), http://www.w3.org/XML/, Acesso em agosto/2009.

[22] Universal Description, Discovery, and Integration, http://uddi.xml.org/uddi-org, Acesso em agosto/2009.

[23] Web Services Description Language (WSDL), http://www.w3.org/TR/wsdl, Acesso em agosto/2009.

[24] Simple Object Access Protocol (SOAP), http://www.w3.org/TR/soap/, Acesso em agosto/2009.

[25] V. Dialani, S. Miles, L. Moreau, D. D. Roure, M. Luck, "Transparent Fault Tolerance for Web Services Based Architectures," *Eighth International Europar Conference,* 2002.

[26] R. L. Vianna, M. J. B. Almeida, L. M. R. Tarouco, L. Z. Granville, "Evaluating the Performance of Web Services Composition for Network Management," *IEEE International Conference on Communications,* 2007.

[27] Python Programming Language, http://www.python.org/, Acesso em agosto/2009.

[28] J. Turek, D. Shasha, "The Many Faces of Consensus in Distributed Systems," *Computer,* pp. 8-17, Vol. 25, No. 6, June 1992.

[29] S. Mullender (Editor), *Distributed Systems,* ACM Press, 1989.

[30] M. J. Fischer, N. A. Lynch, M. S. Paterson, "Impossibility of Distributed Consensus with one Faulty Process," *Journal of the Association for Computing Machinery,* Vol. 32, No. 2, 1985.

[31] T. Chandra, V. Hadzilacos, S. Toueg, "The Weakest Failure Detector for Solving Consensus," *Proceedings of the Eleventh ACM Symposium on Principles of Distributed Computing,* 1992.

[32] M. Larrea, A. Fernández e S. Arévalo, "On the Implementation of Unreliable Failure Detectors in Partially Synchronous Systems," *IEEE Transactions on Computers,* Vol. 53, No. 7, 2004.

[33] W. Chen, S. Toueg, M.K. Aguilera, "On The Quality of Service of Failure Detectors," *IEEE Transactions on Computers,* 2002.

[34] M.K. Aguilera, W. Chen, S. Toueg, "On Quiescent Reliable Communication," *SIAM Journal of Computing,* 2000.

[35] S. Gorender, R. Macêdo, M. Raynal, "A hybrid and adaptative model for fault-tolerant distributed computing," *Proc. of the Int. Conf. on Dependable Systems and Networks (DSN),* 2005.

[36] V. Jacobson, "Congestion Avoidance and Control," *Computer Communication Review,* Vol. 18, No. 4, 1988.

[37] V. Paxson, M. Allman, "Computing TCP's Retransmission Timer," *Request for Comments 2988 (RFC2988),* 2000.

[38] PlanetLab, http://www.planet-lab.org/, Acesso em agosto/2009.

[39] The R Project for Statistical Computing, http://www.r-project.org/, Acesso em agosto/2009.

[40] The GNU Project, http://www.gnu.org/, Acesso em agosto/2009.

Apêndice 1: PS-MIB

```
MIB-PS DEFINITIONS ::= BEGIN

IMPORTS
        enterprises, MODULE-IDENTITY, OBJECT-TYPE FROM SNMPv2-SMI;

-- definicao de "PS"
PS MODULE-IDENTITY
        LAST-UPDATED "200810050000Z"
        ORGANIZATION "PS"
        CONTACT-INFO "dioneimm@inf.ufpr.br, elias@inf.ufpr.br"
        DESCRIPTION  "Armazena informacoes de estado de processos."
        ::= { enterprises ufpr(18722) 1 }

processTable OBJECT-TYPE
        SYNTAX  SEQUENCE OF TabEntry
        ACCESS  not-accessible
        STATUS  mandatory
        DESCRIPTION
                "Tabela que armazena informacoes de processos."
        ::= { HB 1 }

tabEntry OBJECT-TYPE
        SYNTAX  tabEntry
        ACCESS  not-accessible
        STATUS  mandatory
        DESCRIPTION
                "Entrada da tabela com informações de estado de um processo."
        INDEX { tabIndex }
        ::= { processTable 1 }

tabEntry ::=
        SEQUENCE {
                tabIndex  INTEGER,
                processID OCTET STRING,
                status    OCTET STRING
        }

tabIndex OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "O indice da tabela."
        ::= { tabEntry 1 }

processID OBJECT-TYPE
        SYNTAX  OCTET STRING
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "ID do processo, composto por PID:IP:PORTA."
        ::= { tabEntry 2 }
```

```
status OBJECT-TYPE
        SYNTAX  OCTET STRING
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "Estado do processo: working, suspect ou crashed."
        ::= { tabEntry 3 }
END
```

Apêndice 2: HB-MIB

```
MIB-HB DEFINITIONS ::= BEGIN

IMPORTS
        enterprises, MODULE-IDENTITY, OBJECT-TYPE FROM SNMPv2-SMI;

-- definicao de "HB"
HB MODULE-IDENTITY
        LAST-UPDATED "2009060800000Z"
        ORGANIZATION "HB"
        CONTACT-INFO "dioneimm@inf.ufpr.br, elias@inf.ufpr.br"
        DESCRIPTION  "Armazena informacoes de heartbeats."
        ::= { enterprises ufpr(18722) 2 }

heartbeatTable OBJECT-TYPE
        SYNTAX  SEQUENCE OF TabEntry
        ACCESS  not-accessible
        STATUS  mandatory
        DESCRIPTION
                "Tabela que armazena informacoes de heartbeats."
        ::= { HB 1 }

tabEntry OBJECT-TYPE
        SYNTAX  tabEntry
        ACCESS  not-accessible
        STATUS  mandatory
        DESCRIPTION
                "Entrada da tabela com informações de heartbeats."
        INDEX { tabIndex }
        ::= { heatbeatTable 1 }

tabEntry ::=
        SEQUENCE {
                tabIndex        INTEGER,
                HB_interval     OCTET STRING,
                desvio          OCTET STRING,
                hour_HB         OCTET STRING
        }

tabIndex OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "O indice da tabela."
        ::= { tabEntry 1 }

HB_interval OBJECT-TYPE
        SYNTAX  OCTET STRING
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "Intervalo medio de recebimento dos heartbeats"
```

```
        ::= { tabEntry 2 }

desvio OBJECT-TYPE
        SYNTAX  OCTET STRING
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "Desvio padrao."
        ::= { tabEntry 3 }

hour_HB OBJECT-TYPE
        SYNTAX  OCTET STRING
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "Hora do recebimento do ultimo heartbeat."
        ::= { tabEntry 4 }
END
```