EGON HILGENSTIELER
EMERSON FABIANO FONTANA CARARA
ROVERLI PEREIRA ZIWICH

SAPOTI: SERVIDORES DE APLICAÇÕES CONFIÁVEIS TCP/IP

Trabalho apresentado ao curso de Bacharelado em Ciência da Computação da Universidade Federal do Paraná, como requisito parcial à obtenção do título de Bacharel em Ciência da Computação.
Orientador: Prof. Elias Procópio Duarte Jr.

Curitiba
2002

Sumário

Resumo

Este trabalho apresenta uma estratégia distribuída que garante a alta disponibilidade de servidores de aplicações TCP/IP (*Transfer Control Protocol/Internet Protocol*) aplicada especificamente para a implementação de um servidor Web (*World Wide Web*) tolerante a falhas. O serviço tolerante a falhas é baseado no algoritmo de diagnóstico distribuído *Hi-ADSD with Timestamps* que permite o monitoramento dos componentes de uma rede. O diagnóstico é baseado no protocolo SNMP (*Simple Network Management Protocol*), o padrão TCP/IP para gerência de redes. Uma MIB (*Management Information Base*) é utilizada pelo algoritmo *Hi-ADSD with Timestamps* que permite a especificação de testes em máquinas e recursos de uma rede. Através da determinação de uma relação ordenada de máquinas desta rede e com base nas informações obtidas através da MIB, a aplicação apresentada neste trabalho garante a alta disponibilidade de um servidor Web. Desta forma, uma outra máquina assume a disponibilização do serviço Web no caso de falha da máquina que antes oferecia o serviço. É designado um endereço IP (*Internet Protocol*) virtual que é assumido sempre pela máquina que estiver executando o serviço, tornando as mudanças transparentes. Experimentos foram realizados em um intervalo de cerca de 12 horas, em máquinas monitoradas pela ferramenta *Hi-ADSD with Timestamps* e que executavam a aplicação apresentada neste trabalho. Como resultado foi constatado que em configurações de rede com máquinas onde ocorreram 210 falhas distribuídas entre todas as máquinas, a disponibilidade do servidor Web foi de 97,3%. Em configurações de rede não tão instáveis, onde algumas máquinas juntas falharam 27 vezes, a disponibilidade do servidor Web foi de 99,5%.

Capítulo 1

Introdução

As organizações dependem do bom funcionamento de suas redes. Em muitos casos é necessário que a disponibilidade seja garantida. Para alcançar este objetivo é necessário um sistema de gerenciamento de redes para monitoração e controle da rede. Os servidores de aplicações TCP/IP (*Transfer Control Protocol/Internet Protocol*) [1] estão entre os componentes críticos para o bom funcionamento da rede de uma organização.

Este trabalho apresenta uma estratégia para implementação de servidores de aplicações confiáveis TCP/IP, em específico um servidor HTTP (*Hyper Text Transfer Protocol*) [2], chamado servidor Web (*World Wide Web*). Estes servidores confiáveis são implementados sobre uma ferramenta de monitoração de máquinas e recursos de uma rede, baseada em diagnóstico distribuído e adaptativo em nível de sistema. Para implementar uma monitoração confiável e eficiente, pode-se utilizar um algoritmo de diagnóstico distribuído [3]. Assim, é possível identificar falhas em um servidor Web e recuperar o serviço iniciando-o em uma máquina sem-falha. Este trabalho também apresenta uma interface Web para visualização das informações de diagnóstico do sistema.

A seguir é apresentada uma visão geral de diagnóstico em nível de sistema, uma introdução ao gerenciamento de redes, uma breve descrição da ferramenta SAPOTI, da interface e dos resultados obtidos.

1.1 O Algoritmo Hi-ADSD with Timestamps

O objetivo do diagnóstico em nível de sistema é identificar quais unidades do sistema estão falhas e quais estão sem-falhas. Quando o diagnóstico é distribuído [3] cada nodo tem a habilidade de realizar o diagnóstico de todo o sistema. Quando o diagnóstico é adaptativo [4], os testes que cada nodo realiza são baseados em rodadas, cada rodada é realizada com base nos resultados da rodada anterior. Quando o diagnóstico é hierárquico [5], os nodos são divididos em clusters e a cada rodada de testes o tamanho destes clusters aumenta. Inicialmente é testado um cluster de tamanho *1*. O tamanho do cluster é dobrado a cada rodada até chegar a $n/2$, sendo n o número de nodos do sistema. A cada rodada, cada nodo realiza o teste de um cluster inteiro. O algoritmo *Hi-ADSD with Timestamps* [6] é um exemplo de um algoritmo hierárquico, distribuído e adaptativo de diagnóstico em nível de sistema. Este algoritmo é descrito no capítulo 2 e é a base da ferramenta proposta no capítulo 4.

1.2 O Protocolo SNMP

O protocolo SNMP (*Simple Network Management Protocol*) [7] é o padrão da Internet para gerência de redes. Foi projetado para ser simples, como seu nome sugere. Devido à sua simplicidade, o protocolo foi amplamente aceito e implementado nos mais diferentes

dispositivos de rede. Este protocolo permite que os mais distintos componentes de uma rede TCP/IP sejam monitorados e controlados. Cada nodo da rede é visto como um conjunto de objetos. Ao ler estes objetos, o nodo é monitorado, e ao atribuir algum valor a estes objetos, o nodo é controlado. Outras operações possíveis são a de determinar quais objetos um determinado nodo contém e a de permitir que um determinado nodo reporte um determinado evento para um nodo gerenciador. Este conjunto de objetos que podem ser vistos também como uma base de dados é chamado de MIB (*Management Information Base*) [7].

1.3 A Ferramenta SAPOTI

A ferramenta SAPOTI foi construída para garantir a alta disponibilidade de servidores TCP/IP, aplicada em particular para um servidor Web. A ferramenta atua de forma distribuída em uma rede de computadores monitorados pelo algoritmo *Hi-ADSD with Timestamps*. Através das informações de diagnóstico geradas pelo algoritmo, a ferramenta disponibiliza o servidor Web em uma máquina da rede, caso exista pelo menos uma máquina sem-falha que esteja sendo monitorada nesta rede.

O algoritmo *Hi-ADSD with Timestamps* utiliza o protocolo padrão de gerência de redes SNMP em sua implementação. O algoritmo implementa uma MIB contendo todos os dados necessários para o monitoramento. Com isso, os nodos trocam informações entre si, a partir dos dados de suas respectivas MIB's. Além de fornecer as informações de diagnóstico da rede, a implementação do algoritmo *Hi-ADSD with Timestamps* permite especificar testes configuráveis para cada elemento e serviço da rede.

Na configuração inicial, o algoritmo *Hi-ADSD with Timestamps* especifica um índice a cada máquina monitorada. Este índice também é usado como identificador da ordem de prioridade das máquinas pela ferramenta SAPOTI. A implementação da ferramenta SAPOTI foi feita de forma que a estratégia é executar o algoritmo como um *daemon* [8] em cada máquina da rede. Cada máquina que executa o algoritmo da ferramenta verifica qual máquina de menor identificador está disponibilizando o servidor Web, inclusive ela própria. Usando estas informações a ferramenta disponibiliza o servidor Web na máquina sem-falha de maior prioridade. Para que o atendimento às requisições destinadas ao servidor Web seja transparente, um endereço IP virtual único é utilizado associado ao servidor Web. Desta forma, toda vez que uma máquina estiver disponibilizando o serviço Web, estará configurada uma interface de rede para o endereço IP virtual associado, como mostrado na figura 1.1.

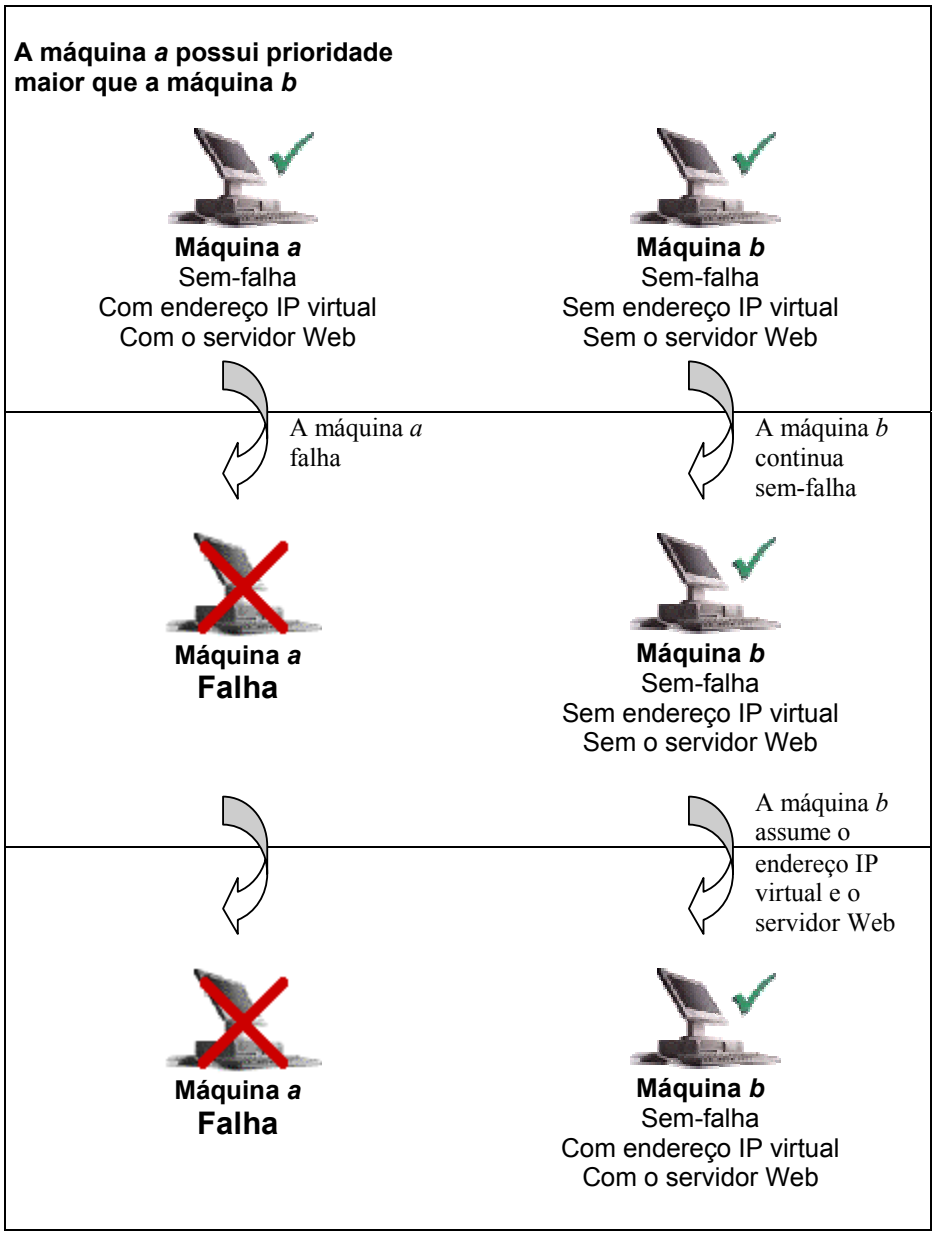


Figura 1.1: Outra máquina assume o endereço IP virtual

no caso da falha do servidor Web.

Uma interface Web foi implementada para visualizar as informações de diagnóstico da rede monitorada pelo algoritmo *Hi-ADSD with Timestamps*. Esta interface permite a visualização dos resultados de todos os testes da MIB utilizada pelo algoritmo, inclusive a disponibilidade do servidor Web a partir de qualquer máquina da rede.

Experimentos foram realizados em um intervalo de cerca de 12 horas, em máquinas monitoradas pela ferramenta *Hi-ADSD with Timestamps* e que executavam a aplicação apresentada neste trabalho. Como resultado foi constatado que em configurações de rede com máquinas onde ocorreram 210 falhas distribuídas entre todas as máquinas, a disponibilidade do servidor Web foi de 97,3%. Em configurações de rede não tão instáveis, onde algumas máquinas juntas falharam 27 vezes, a disponibilidade do servidor Web foi de 99,5%.

1.4 Organização do Trabalho

O restante deste trabalho está organizado da seguinte maneira. O capítulo 2 descreve o algoritmo de diagnóstico distribuído *Hi-ADSD with Timestamps*. Em seguida, o capítulo 3 contém uma visão geral de gerenciamento de redes com o protocolo SNMP, que foi utilizado para a implementação do algoritmo *Hi-ADSD with Timestamps* e da ferramenta SAPOTI. No capítulo 4 é descrita a ferramenta SAPOTI e a interface Web utilizada. O capítulo 5 apresenta os resultados experimentais do uso da ferramenta em uma rede com injeção de falhas.

Capítulo 2

Hi-ADSD with Timestamps

Diagnóstico em nível de sistema permite que os componentes sem-falha do sistema determinem quais componentes estão falhos e quais estão sem-falha [3]. Os algoritmos hierárquicos adaptativos e distribuídos de diagnóstico em nível de sistema entre eles o *Hi-ADSD with Timestamps* [6] permitem a monitoração prática de máquinas e recursos de uma rede local.

A seguir é apresentada uma breve descrição de diagnóstico em nível de sistema e do algoritmo *Hi-ADSD with Timestamps* incluindo o conceito de eventos dinâmicos, a especificação do algoritmo, sua estratégia de testes e por último sua avaliação segundo os critérios de latência e número de testes requeridos.

2.1 Diagnóstico em Nível de Sistema

Considere um sistema S formado por um conjunto de N nodos, $n_0, n_1, \ldots, n_{N-1}$. Cada nodo n_i pode estar em um dos seguintes estados: falho ou sem-falha. Um evento ocorre quando um nodo muda de estado, ou seja, quando um nodo sem-falha fica falho ou quando um nodo falho se recupera. Neste trabalho o nodo n_i é também chamado nodo i. Assuma que o

sistema S é representável por grafo completo, isto é, há ligação entre quaisquer pares de nodos (n_i, n_j) e não existem falhas de enlace.

O primeiro modelo de diagnóstico em nível de sistema foi o modelo PMC [9]. O conjunto de testes deste modelo forma um grafo direcionado em que os vértices são os nodos do sistema e as arestas, que vão de i paxra j, representam um teste do nodo i para o nodo j. Todos os resultados de testes são chamados de síndrome do sistema. O modelo PMC assume a existência de um observador central que, baseado na síndrome, realiza o diagnóstico do estado de todos os nodos do sistema. Além disso, é assumido também que todo nodo sem-falha executa de modo confiável todos os testes, isto é, um nodo sem-falha nunca realiza um teste de forma imprecisa.

Se cada nodo do sistema é capaz de executar testes em outros nodos e se os resultados dos testes dependem dos resultados de testes anteriores, então o algoritmo utilizado é dito adaptativo [4]. Se o algoritmo não assume a existência de um nodo monitor, mas considera que os próprios nodos que realizam os testes fazem o diagnóstico do sistema então o algoritmo também é dito distribuído [10]. Nestes algoritmos ao realizar um teste em um nodo sem-falha, o nodo testador recebe informações do nodo testado. Estas informações são chamadas de informações de diagnóstico.

Algoritmos adaptativos e distribuídos de diagnóstico em nível de sistema trabalham com rodadas de testes, que são intervalos de tempo em que cada nodo executa seus testes.

Os algoritmos hierárquicos adaptativos e distribuídos de diagnóstico em nível de sistema são: o algoritmo *Hi-ADSD* [5], o algoritmo *Hi-ADSD with Detours* [11] e por último o *Hi-ADSD with Timestamps* [6]. Estes algoritmos utilizam uma estratégia de agrupar os nodos em clusters lógicos de tamanho progressivo. O número de nodos em um cluster, isto é, seu tamanho, é sempre uma potência de dois. Estes algoritmos executam seus testes de maneira assíncrona, isto é, em um determinado intervalo de testes, os nodos sem-falha podem testar clusters de tamanhos diferentes.

Três critérios são geralmente utilizados para avaliar estes algoritmos, são eles: a quantidade de informação de diagnóstico transferida a cada teste, o número máximo de testes do sistema e a latência, isto é, o tempo total necessário para que todos os nodos completem o diagnóstico do sistema.

O algoritmo *Hi-ADSD* possui latência de, no máximo, log^2N rodadas de testes para um sistema de N nodos. Neste trabalho, todos os logaritmos estão em base 2. O número máximo de testes por rodada pode chegar a $N^2/4$ testes e a quantidade de informações de diagnóstico transferidas a cada teste varia de *1* até $N/2$ nodos, dependendo do cluster testado [5].

O algoritmo *Hi-ADSD with Detours* requer no máximo $NlogN$ testes por $logN$ rodadas de testes, a latência no pior caso é de log^2N rodadas de testes e o número de informações de diagnóstico transferidas a cada teste é variável e sempre menor que $N-1$ nodos [11].

Por último, no algoritmo *Hi-ADSD with Timestamps* apesar do pior caso da latência continuar log^2N rodadas e o número de testes que é de *NlogN* também continuar o mesmo, a latência média é consideravelmente menor que a latência dos algoritmos *Hi-ADSD* e *Hi-ADSD with Detours*. É transferido um número fixo de informações de diagnóstico de *N/2* nodos a cada teste. Este algoritmo é brevemente descrito abaixo, para uma descrição completa sugere-se [6].

2.2 O Algoritmo Hi-ADSD with Timestamps

O algoritmo *Hi-ADSD with Timestamps* foi apresentado por Duarte, Brawerman e Albini [6]. Este algoritmo utiliza a mesma estratégia de agrupar os nodos em clusters lógicos utilizada pelos outros algoritmos hierárquicos. É o primeiro algoritmo hierárquico a tratar de falhas dinâmicas [6], isto é, eventos podem ocorrer antes que outros eventos tenham sido completamente diagnosticados pelo sistema. Além disso, um procedimento de recuperação [6] é especificado, o que permite aos nodos que tenham sido reparados unirem-se àqueles que já estão rodando o algoritmo.

2.2.1 Especificação do Algoritmo Hi-ADSD with Timestamps

O grafo dirigido dos nodos sem-falhas testados, *T(S)*, tem arestas dirigidas do nodo *a* para o nodo *b* caso o nodo *a* teste o nodo *b* como sem-falha. Quando não temos falhas em nenhum nodo do sistema, *T(S)* é um hipercubo, como mostra a figura 2.1 para *N=8*.

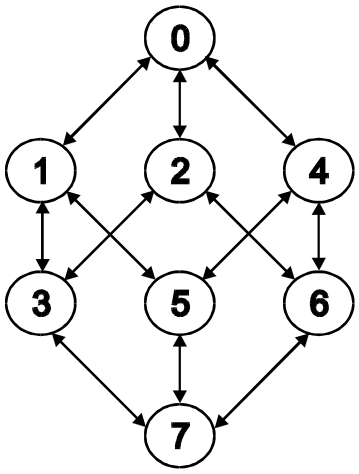


Figura 2.1: *T(S)* para um sistema com 8 nodos.

A menor distância entre o nodo *i* e *j* no grafo *T(S)* é chamada de distância de diagnóstico entre o nodo *i* e o nodo *j*. Chamamos *TFFi* o grafo de um nodo sem-falha testado. *TFFi* é um grafo dirigido onde os nodos estão em *S*. Existe uma aresta na direção do nodo *a* para o nodo *b,* se esta aresta está no conjunto de arestas do grafo *T(S)* e a distância de diagnóstico do nodo *i* para o nodo *a* é menor que a distância de diagnóstico do nodo *i* para o nodo *b.*

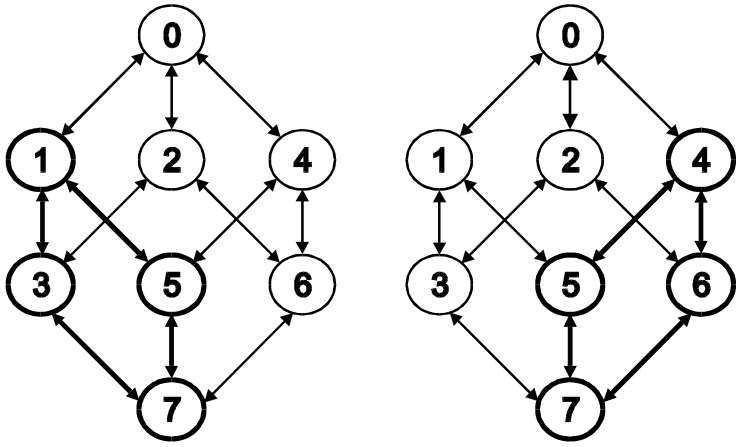
Seja $c_{i,s,p}$ definida como a lista ordenada dos nodos que podem ser alcançados pelo nodo *i* partindo do nodo *p* com distância de diagnóstico menor ou igual a *s–1*, ou seja, se o caminho do nodo *i* para o nodo *j* através do nodo *p* na *TFFi* tem no máximo *s* arestas, então o nodo *j* pertence a $c_{i,s,p}$.

2.2.2 Estratégia de Testes

A função $c_{i,s,p}$ fornece os nodos de um cluster quando o nodo *i* é o testador. Neste algoritmo *s* é sempre *logN,* isto é, cada cluster contem *N/2* nodos. Quando um nodo é testado, o nodo

testador obtém informação sobre todo o cluster do nodo testado. Em cada intervalo de teste, cada nodo testa um determinado nodo em um cluster. Em *logN* intervalo de testes todos os clusters são testados. O processo continua indefinidamente.

No algoritmo *Hi-ADSD with Timestamps* é possível para o nodo *i* obter informação de diagnóstico sobre o nodo *j* a partir de dois ou mais nodos sem-falha. Neste caso é necessário garantir que o nodo *i* seja capaz de determinar a informação mais recente sobre o estado do nodo *j.* Por exemplo, na figura 2.2 o nodo 0 obtém informação de diagnóstico do nodo 5 através do nodo 1 e do nodo 4 porque o nodo 5 pertence à $c_{i,logN,1}$ e $c_{i,logN,4}$. Consequentemente é preciso garantir que o nodo 0 obtém a informação mais recente sobre o nodo 5.



$c_{0,3,1}$ – lista da $c_{i,s,p}$ é {1,3,5,7} $c_{0,3,4}$ – lista da $c_{i,s,p}$ é {4,5,6,7}

Figura 2.2: Exemplos de $c_{i,s,p}$: $c_{0,3,1}$ e $c_{0,3,4}$.

Para garantir que o nodo *i* mantenha somente a informação mais recente sobre o estado do nodo *j,* foi adotada uma estratégia que tem sido usada nos algoritmos de diagnóstico de

sistema para redes de topologia arbitrária [11, 12]. Esta estratégia consiste no uso de *timestamps,* isto é, informação datada. Os *timestamps* são implementados como contadores de estados. Um contador deve ser incrementado quando um teste é executado e o testador detecta que o nodo testado mudou de estado.

Para certificar que o nodo *i* mantenha a informação mais recente sobre o estado do nodo *j,* o nodo *i,* quando testa o nodo *p,* compara o seu *timestamp* com o *timestamp* do nodo *p* com respeito ao nodo *j.* Se o *timestamp* do nodo *i* é igual ou maior que o do nodo *p*, ou seja, o nodo *i* está obtendo informação mais antiga sobre o nodo *j,* o nodo *i* deve manter seu *timestamp*. Mas, caso contrário, o nodo *i* estará obtendo uma informação mais nova que a sua sobre o nodo *j,* então o nodo *i* atualiza a informação sobre o estado do nodo *j* e seu *timestamp* correspondente.

Por outro lado, se o nodo *i* testa um cluster e encontra um nodo falho, o nodo *i* não testa outro nodo no cluster no mesmo intervalo de testes. Não é testado um outro cluster porque o nodo *i* pode obter informação sobre este cluster através de nodos sem-falha localizados fora deste cluster, usando caminhos alternativos chamados desvios [11]. Um teste extra é executado somente quando não existem desvios do nodo *i* para aquele nodo.

2.2.3 Procedimento de Recuperação

Um nodo que acaba de se recuperar não possui informações atualizadas de diagnóstico sobre qualquer nodo do sistema, pois muitos eventos podem ter acontecido enquanto aquele

nodo estava falho. Deste modo, ao começar a execução de seus testes, o nodo atualiza suas informações de diagnóstico a partir do primeiro nodo sem-falha testado.

A informação de diagnóstico que cada nodo mantém sobre os outros nodos do sistema pode ser armazenada numa tabela onde cada entrada é usada para manter o estado de um nodo do sistema. Cada entrada deve ter um campo que indica se a entrada foi atualizada depois que o nodo foi reparado pela última vez. Este campo chamado *u-bit*, pode ser implementado com um bit que se estiver em *1* indica que as informações já foram atualizadas, caso esteja em *0* não foram atualizadas.

Quando o nodo inicializa o algoritmo, todos os *u-bits* são setados para *0*. Assim que o nodo começa a rodar o algoritmo ele obtêm informações de diagnóstico dos nodos sem-falhas. No momento em que cada testador atualiza suas informações locais, *u-bits* correspondentes são setados para *1*. Para cada informação de diagnóstico obtida de um nodo sem-falha testado, o testador checa se o *u-bit* correspondente no nodo testado é *1*, neste caso atualiza sua informação local.

2.2.4 Eventos Dinâmicos

Quando todos os nodos sem-falha do sistema obtém informações sobre um determinado evento, dizemos que o evento foi completamente diagnosticado. Se ocorrerem muitos eventos no sistema é necessário garantir quais nodos realizarão o diagnóstico de quais eventos. Se todos os nodos ficarem falhos e forem recuperados rapidamente eles podem nunca diagnosticar qualquer evento no sistema.

Enquanto um nodo que acabou de se recuperar não atualizar suas informações locais de diagnóstico, não é esperado que ele complete o diagnóstico dentro da latência do algoritmo. Um nodo que for reparado neste intervalo reiniciará o algoritmo e pode levar *logN* intervalos de testes para atualizar suas informações de diagnóstico. Após este intervalo o nodo é dito estável.

Considere agora um nodo *i,* que sofreu eventos consecutivos. Alguns desses eventos podem ter sido diagnosticados, outros não. Outros algoritmos hierárquicos possuem uma asserção que impede tal ocorrência. Já para garantir que o algoritmo *Hi-ADSD with Timestamps* complete o diagnóstico de um evento, é preciso que todos os nodos estáveis sem-falha do sistema mantenham informação sobre este evento por um período de tempo longo o suficiente para todos os testadores obterem esta informação.

2.2.5 Latência e Número Máximo de Testes Requeridos

Quando o nodo *i* testa um nodo sem-falha em um dado cluster, ele obtém informação de diagnóstico sobre o conjunto dos nodos cuja distância de diagnóstico é no máximo *logN–1*. Este conjunto tem sempre *N/2* nodos. Entretanto se o nodo testado é falho, duas situações podem ocorrer. A primeira vez que o nodo *i* testa um nodo falho, ele não continua os testes neste cluster. O nodo *i* procura por desvios nas próximas *logN* rodadas de testes, de clusters menores que o corrente. A segunda situação ocorre quando o testador determina que não há desvios de clusters menores para os nodos daquele cluster. Neste caso, o

testador executa testes sequenciais nos nodos naquele cluster até um nodo sem-falha ser encontrado ou até que reconheça que todos os nodos do cluster estão falhos.

No pior caso existe somente um caminho através do qual todas as informações de diagnóstico podem passar. A figura 2.3 mostra o pior caso da latência de um sistema com 8 nodos rodando o algoritmo *Hi-ADSD with Timestamps*. Neste exemplo existe somente um caminho por onde toda a informação de diagnóstico pode passar, isto é, somente um caminho que conecta o nodo 0 ao 7.

O número máximo de testes é *NlogN* para cada *logN* rodadas de testes, que é menor que no *Hi-ADSD with Timestamps* onde é $N^2/4$ testes em uma rodadas de testes.

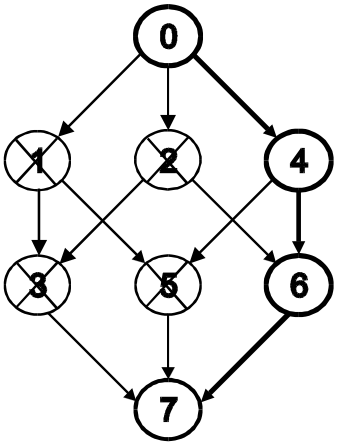


Figura 2.3: Pior caso da latência
de um sistema com 8 nodos.

Capítulo 3

O Protocolo SNMP

O objetivo de um protocolo de gerência de rede é permitir a monitoração e o controle da rede e todo dispositivo conectado a ela, com seus recursos associados. Os dispositivos gerenciados através do protocolo SNMP (*Simple Network Management Protocol*) se comunicam uns com os outros através do protocolo de gerência. Mais especificamente, o SNMP é um protocolo da camada de aplicação sobre UDP/IP (*User Datagram Protocol/Internet Protocol*) [1].

A versão 1 do protocolo, chamada SNMPv1, foi especificada nos RFCs 1155, 1157 e 1223 [13, 14, 15]. Devido à sua simplicidade e extensibilidade, o SNMP foi amplamente aceito, tornando-se uma solução de protocolo de gerenciamento de rede para uma variedade cada vez maior de plataformas e ambientes de rede. Seu sucesso foi assegurado pela grande proliferação de dispositivos de rede que necessitavam ser gerenciados de uma forma padrão. Com mais experiência e tendo em vista a necessidade de mecanismos eficazes de segurança, foi criada uma segunda e posteriormente uma terceira versão do protocolo (SNMPv2 e SNMPv3).

A seguir é apresentado com base em [7, 16] o modelo gerente-agente, os objetos SNMP, o protocolo SNMP e finalmente a MIB (*Management Information Base* – base de informação de gerência).

3.1 O Modelo Gerente-Agente

No modelo Gerente-Agente, existem dois tipos de entidades que implementam o protocolo SNMP: agentes e gerentes. Um agente é responsável pela disponibilização das informações sobre um determinado elemento de rede. Dessa forma, os agentes podem ser simples e têm baixo impacto sobre os dispositivos nos quais estão sendo executados [7]. A figura 3.1 mostra um exemplo para o modelo Gerente-Agente.

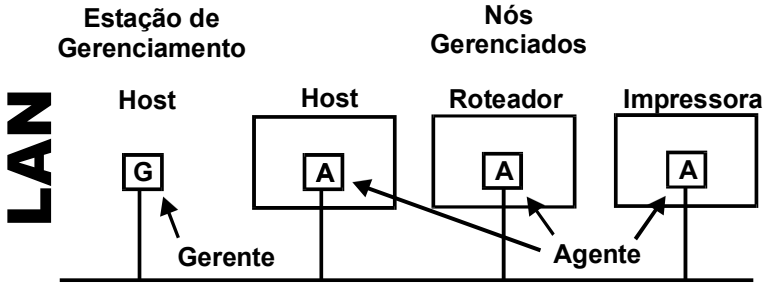


Figura 3.1 Modelo Gerente-Agente.

Um agente pode ser qualquer dispositivo capaz de comunicar informações que descrevam o seu estado (hosts, roteadores, pontes, impressoras, etc.). Essas informações são armazenadas em variáveis chamadas de objetos, que são capazes de descrever seu estado, histórico e também afetar sua forma de funcionamento. O conjunto formado por todos os

objetos possíveis em uma rede é estruturado e disponibilizado através da chamada MIB (*Management Information Base*).

Os gerentes localizam-se em estações de gerenciamento de rede, sendo responsáveis por manter um ou mais processos que implementam aplicações de gerência. Estas aplicações de gerência se comunicam com os agentes espalhados pela rede, emitindo comandos, obtendo respostas e, caso seja necessário, alterando o estado de objetos.

3.2 Objetos SNMP

O cerne do SNMP é o conjunto de objetos mantidos pelos agentes e manipulado pelos gerentes. Para que haja compatibilidade entre os diversos tipos, modelos e fabricantes de equipamentos é fundamental que os objetos sejam definidos de uma forma padronizada e neutra. É necessária também uma forma padronizada para que os objetos sejam decodificados para transferência através da rede.

A linguagem de definição de objetos padronizada e as regras de codificação utilizadas pelo SNMP são baseadas num subconjunto da linguagem ASN.1 (*Abstract Syntax Notation One*) [14].

A linguagem de declaração de dados permite que objetos mais complexos sejam criados a partir da combinação de objetos mais primitivos. Já a sintaxe de transferência ASN.1 define como os valores dos tipos ASN.1 são convertidos em uma sequência de bits para a transmissão e decodificação, sem qualquer ambiguidade [17].

3.3 O Protocolo SNMP

No protocolo SNMP, o gerente envia uma solicitação a um agente pedindo informações ou solicitando a atualização de seu estado de alguma forma. O agente responde com informações solicitadas pelo gerente ou confirma a atualização de dados. Esta comunicação é feita através do protocolo SNMP, onde os dados são enviados com base na sintaxe de transferência ASN.1.

O protocolo SNMP contém apenas 7 primitivas de requisição/resposta, sumarizadas na tabela 3.1: *get-request*, *get-next-request*, *get-bulk-request*, *set-request*, *inform-request*, *trap* e *get-response.* Os comandos *get*, *get-next* e *get-bulk* servem para solicitar informações, enquanto *set* armazena uma nova informação. A primitiva *get-response* é a resposta do agente a aceitação do *get*, *get-next*, *get-bulk* e *set* requeridos pelo gerente. A primitiva *trap* permite ao agente enviar um alerta assíncrono ao sistema de gerência de rede para sinalizar condições predefinidas. A primitiva *inform* permite que um gerente informe a outro variáveis que está gerenciando. A especificação de um objeto contém a permissão de acesso associada: somente leitura, somente escrita, ou leitura e escrita.

Get	Solicita o valor de uma ou mais variáveis.
Get-next	Solicita a variável seguinte à esta corrente.
Get-Bulk	Extrai um grande volume de dados.
Set	Atualiza uma ou mais variáveis.
Inform	Mensagem enviada entre os gerentes, cujo objetivo é descrever o valor de um objeto.
Trap	Informa ao gerente que uma determinada condição ocorreu.
Response	Resposta a uma solicitação.

Tabela 3.1: Mensagens do protocolo SNMP.

3.4 A Base de Informação de Gerência (MIB)

A chamada MIB contém os objetos de interesse para as diversas aplicações de gerência. Como já foi dito, os objetos são descritos usando a linguagem ASN.1. Por conveniência, esses objetos são agrupados de acordo com sua funcionalidade. Por exemplo, o grupo *system* se refere ao sistema gerenciado e permite que o gerente descubra o nome do dispositivo, quem o fabricou, sua localização e sua funcionalidade, entre outras coisas.

A organização dos objetos de uma MIB é hierárquica, assim como os domínios da Internet, e pode ser representada através de uma árvore. A figura 3.2 mostra a hierarquia com o prefixo de todos os identificadores de objetos SNMP *.iso.org.dod.internet*.

O identificador é usado como prefixo para grupos de objetos. Por exemplo, o grupo *system,* tem identificador *.iso.org.dod.internet.mgmt.mib.system.* Em cada grupo ocorrem ramificações que levam até os objetos de gerência. Por exemplo, o objeto *sysDescr*, que é um dos objetos do grupo *system* tem como identificador *.iso.org.dod.internet.mgmt.mib.system.sysDescr.0*.

Para obter o valor de um objeto qualquer, sempre deve haver um número no final de um identificador. Os objetos podem ser variáveis simples ou tabelas, sendo de tipo definido pelo ASN.1. Se o final do identificador for o número 0 (zero), trata-se de uma variável simples. Caso contrário trata-se de um conjunto unidimensional de objetos, onde o número representa o índice do elemento na tabela. O identificador de um objeto é conhecido como *Object Identifier* (*OID*).

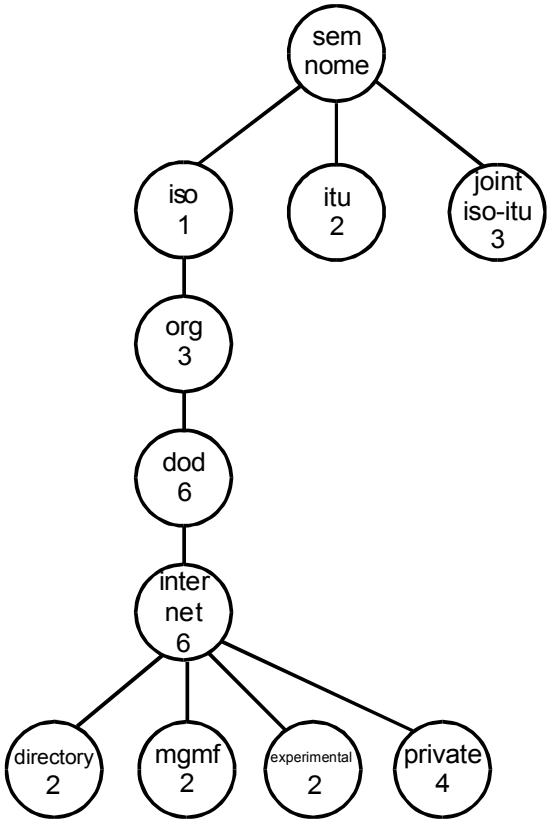


Figura 3.2: Árvore hierárquica do *identificador.iso.org.dod.internet*.

Capítulo 4

SAPOTI: Servidores de APlicações cOnfiáveis Tcp/Ip

SAPOTI é uma ferramenta que garante a alta disponibilidade de servidores de aplicações TCP/IP, aplicada em particular para um servidor Web. A ferramenta atua de forma distribuída em uma rede de computadores monitorados pelo algoritmo *Hi-ADSD with Timestamps.* Através das informações de diagnóstico geradas pelo algoritmo, a ferramenta disponibiliza o servidor Web em uma máquina da rede, caso exista pelo menos uma máquina sem-falha que esteja sendo monitorada nesta rede.

Uma interface Web foi implementada para visualizar as informações de diagnóstico da rede monitorada pelo algoritmo *Hi-ADSD with Timestamps*. Esta interface permite a visualização dos resultados de todos os testes da MIB utilizada pelo algoritmo, inclusive a disponibilidade do servidor Web, a partir de qualquer máquina da rede.

A seguir é apresentado o funcionamento e a implementação da ferramenta SAPOTI e também o funcionamento da interface para visualização dos dados da MIB.

4.1 Considerações Iniciais

Para a implementação do algoritmo *Hi-ADSD with Timestamps* é utilizado o protocolo padrão de gerência de redes SNMP. Assim, o algoritmo atualiza e recupera informações de diagnóstico de qualquer máquina da rede através dos comandos *snmpset, snmpwalk* e *snmpget* do pacote NET-SNMP [18]. Desta forma, cada máquina que será monitorada pelo algoritmo deverá ter o agente SNMP instalado.

Na configuração inicial, o algoritmo *Hi-ADSD with Timestamps* especifica um índice a cada máquina monitorada. Além de fornecer as informações de diagnóstico da rede, a implementação do algoritmo [28] permite especificar testes configuráveis para cada elemento e serviço desta rede e especificar a frequencia com que estes testes são executados. Um exemplo de teste é o que determina o estado do servidor Web de cada uma das máquinas monitoradas.

4.2 Esquema de Prioridades

O índice configurado para as máquinas no algoritmo *Hi-ADSD with Timestamps* é usado como identificador da ordem de prioridade das máquinas pela ferramenta SAPOTI. Uma máquina *a* tem maior prioridade que uma máquina *b* se a máquina *a* possuir um identificador menor que o da máquina *b*. Considerando duas máquinas *i* e *j* sem-falha em um dado momento da execução da ferramenta SAPOTI, se a máquina *i* possui maior prioridade que a máquina *j*, a máquina *i* tem maior prioridade em disponibilizar o servidor Web.

4.3 A Ferramenta SAPOTI

A ferramenta SAPOTI pode ser utilizada em uma rede TCP/IP com máquinas que utilizam o sistema operacional GNU/Linux [19]. Para estas máquinas é utilizado o servidor Web Apache [20] como servidor de aplicações TCP/IP. O servidor Web deve ser instalado em um grupo de máquinas da rede que disponibilizam o serviço confiável. A replicação do conteúdo disponibilizado pelo servidor Web é feito através do aplicativo RSYNC [21] que sincroniza entre todas as máquinas da rede qualquer alteração neste conteúdo. Para uma atualização periódica deste conteúdo foram utilizadas chamadas previamente agendadas ao RSYNC. Para que o atendimento às requisições destinadas ao servidor Web seja transparente, um endereço IP virtual único é utilizado associado ao servidor Web. Desta forma, toda vez que uma máquina estiver disponibilizando o serviço Web, estará configurada nesta máquina uma interface de rede para o endereço IP virtual associado.

A implementação da ferramenta SAPOTI foi feita na linguagem *bash script* [22] e a estratégia é executar o algoritmo como um *daemon* [8] em cada máquina da rede. Cada máquina que executa o algoritmo da ferramenta SAPOTI verifica qual máquina de menor identificador está disponibilizando o servidor Web, inclusive ela própria. Este teste foi especificado através de configuração na MIB do algoritmo *Hi-ADSD with Timestamps*. Todas estas informações são obtidas única e exclusivamente através da MIB da máquina local. Como a MIB é local, a ferramenta SAPOTI não executa nenhuma requisição pela rede. Através de todas estas informações o algoritmo da ferramenta disponibiliza o servidor Web na máquina sem-falha de maior prioridade da rede. Com isso, caso o algoritmo

executando na máquina *a* detecte que o servidor Web não está disponível em nenhuma máquina, o algoritmo inicia o servidor Web na máquina *a* e configura a interface de rede para o endereço IP virtual associado, como ilustrado na figura 4.1.

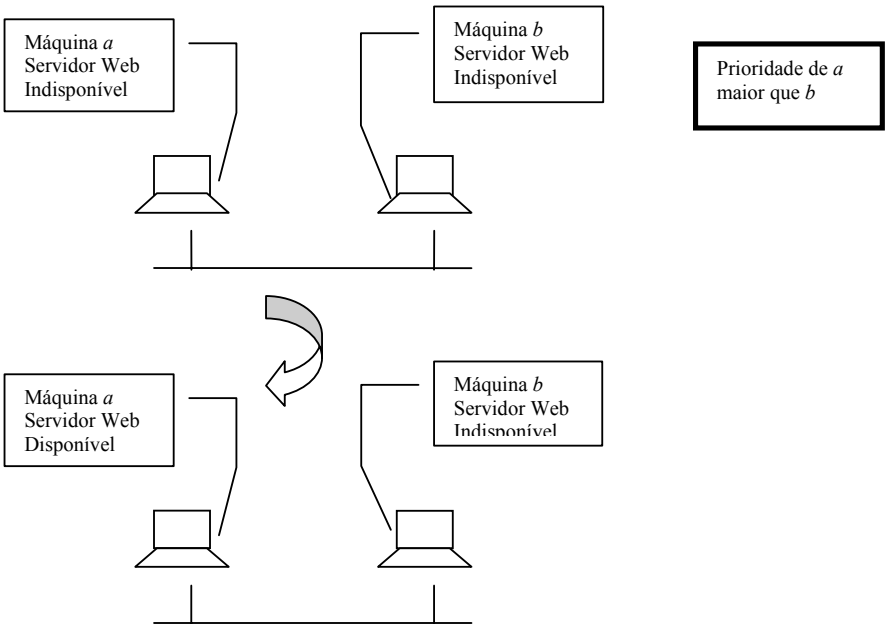


Figura 4.1: Inicialização do servidor Web após a detecção
de que nenhuma máquina o disponibilizava.

Caso o algoritmo executando na máquina *a* detecte que o servidor Web está disponível em uma máquina de maior prioridade e a própria máquina *a* não esteja disponibilizando o servidor Web, nenhuma ação é tomada, como mostrado na figura 4.2.

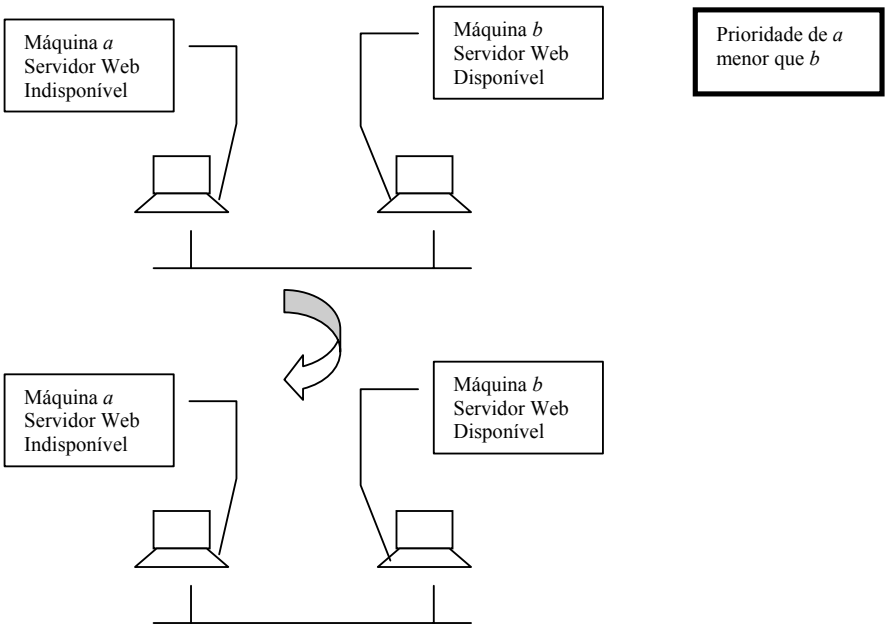


Figura 4.2: Nada se altera após a detecção de que o servidor Web está disponível em uma máquina de maior prioridade.

Caso o algoritmo executando na máquina *a* detecte que o servidor Web está disponível em uma máquina de maior prioridade e a própria máquina *a* também esteja disponibilizando o servidor Web, o algoritmo indisponibiliza o serviço Web na máquina *a* através do encerramento do processo do Servidor Web e da retirada da interface de rede do endereço IP virtual associado, como mostrado na figura 4.3.

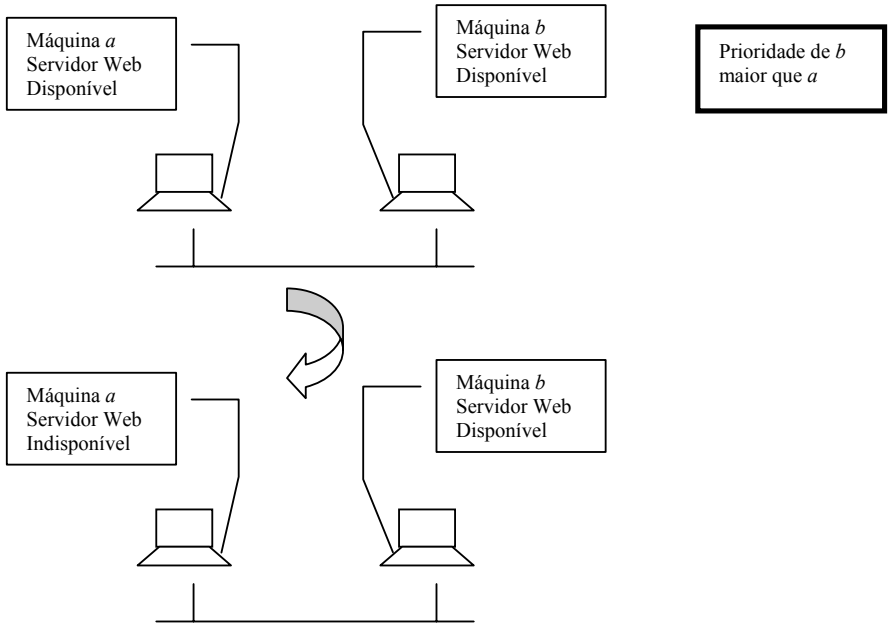


Figura 4.3: Indisponibilização do servidor Web na máquina *a* após a detecção de que o servidor Web também está disponível em uma máquina de maior prioridade.

Caso o algoritmo executando na máquina *a* detecte que o servidor Web está disponível em uma máquina de menor prioridade e a própria máquina *a* não disponibiliza o servidor, o algoritmo inicia o serviço Web na máquina *a* através da inicialização do processo do servidor Web e da configuração da interface de rede para o endereço IP virtual associado, como mostrado na figura 4.4.

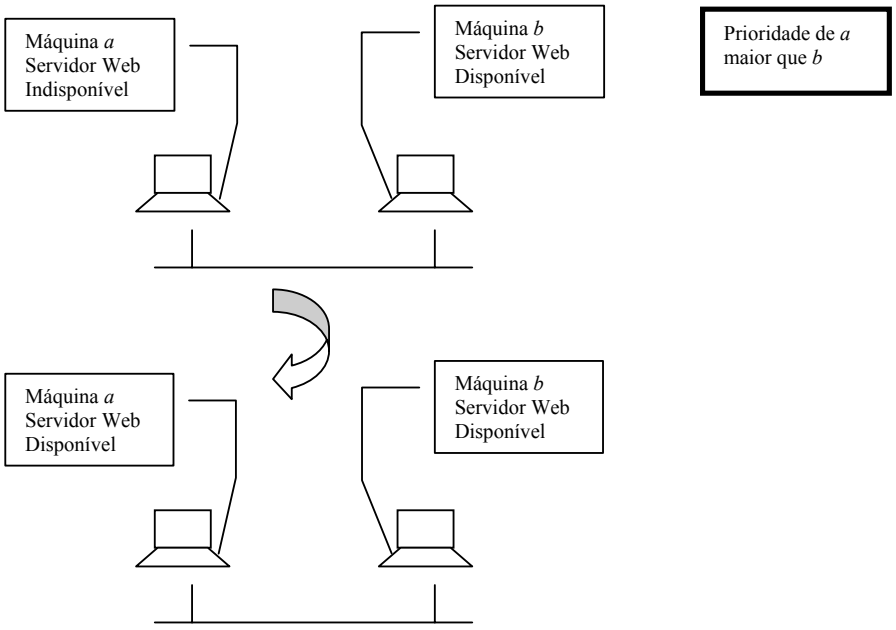


Figura 4.4: Disponibilização do servidor Web após a detecção de que o servidor Web está disponível em uma máquina de menor prioridade.

Caso o algoritmo executando na máquina *a* detecte que a máquina de maior prioridade que disponibiliza o servidor Web é a própria máquina *a*, nenhuma ação é tomada, como mostrado na figura 4.5.

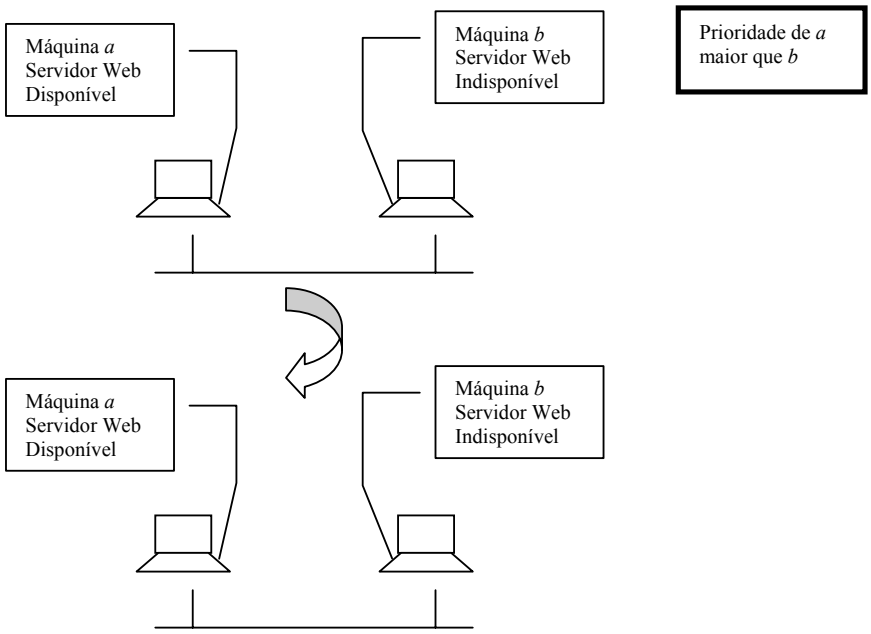


Figura 4.5: Nada se altera após a detecção de que a máquina de maior prioridade que disponibiliza o servidor Web é a própria máquina *a*.

Desta forma, se tivermos uma rede em que todas as máquinas estejam sendo monitoradas pelo algoritmo *Hi-ADSD with Timestamps* e estiverem executando a ferramenta SAPOTI, temos a garantia de que se a máquina responsável pela disponibilização do servidor Web falhar, seguramente outra máquina irá assumir automaticamente esta responsabilidade, caso exista alguma outra máquina sem-falha nesta rede.

Abaixo o algoritmo em alto nível da ferramenta SAPOTI é mostrado em fonte destacada.

```
Algoritmo da ferramenta SAPOTI em Alto-Nível

declare my_num, node_num

my_num='meu identificador segundo a configuração
            do algoritmo Hi-ADSD with Timestamps'
```

```
while [TRUE]
do
       while 'Agente SNMP está fora de serviço'
       then

              if 'Estou com o IP Virtual do servidor Web'
              then
                     'Retira a interface do IP Virtual do servidor Web'
              end if

              if 'O servidor Web está no ar'
              then
                     'Retira o servidor Web do ar'
              end if

              sleep 10
       end while

       # Descobre qual a primeira máquina está disponibilizando o #
       # servidor Web (de acordo com a ordem de identificadores #
       # configurada no algoritmo Hi-ADSD with Timestamps) #

       node_num='número da primeira máquina que está com o servidor Web
                    no ar'

       # Se existe alguma máquina disponibilizando o servidor Web e esta #
       # máquina tem maior prioridade que a minha máquina... #

       if ( node_num < my_num ) AND (node_num is not null)
       then
              if 'Estou com a interface do endereço
                     IP Virtual do servidor Web'
              then
                     'Retira a interface do endereço
                            IP Virtual do servidor Web'
              end if

              if 'O servidor Web está no ar'
              then
                     'Retira o servidor Web do ar'
              end if


       # Se_não existe nenhuma máquina disponibilizando o servidor Web #
       # ou a máquina que está disponibilizando o servidor Web tem menor #
       # prioridade que a minha máquina... #
       else

              if 'Não estou com a interface do endereço
                     IP Virtual do servidor Web'
              then
                     'Inicializa a interface do endereço
                            IP Virtual do servidor Web'
              end if

              if 'O servidor Web está fora ar'
              then
                     'Inicializa a disponibilização do servidor Web'
              end if

       end if

       sleep 10

end while
```

Nesta descrição do algoritmo em alto nível da ferramenta SAPOTI, o comando `sleep 10`, utilizado duas vezes, é mostrado da forma como foi utilizado na implementação. O objetivo dos sleeps foi criar uma pausa de 10 segundos durante cada iteração do algoritmo. Com a diminuição do tempo dos sleeps, podemos conseguir uma maior precisão na detecção e disponibilização do servidor Web.

4.3.1 Uma Limitação da Ferramenta SAPOTI

A ferramenta SAPOTI possui uma limitação referente ao serviço tolerante a falhas oferecido pela aplicação TCP/IP. Transações que exijam algum tipo de alteração no servidor através de modificação de conteúdo armazenado não podem ser garantidas, como por exemplo, atualização de registros contadores de acessos de paginas Web. Por outro lado, esta limitação não se torna relevante se no controle do armazenamento das informações for usado um SGBD (Sistema Gerenciador de Banco de Dados) [23] que possua controle de transações, pois neste caso, o próprio SGBD garante o controle de transações [23] no banco de dados.

4.4 Interface

Uma interface Web foi implementada em PHP *(PHP: Hypertext Preprocessor)* [24] e *bash script* para visualizar as informações de diagnóstico da rede monitorada pelo algoritmo *Hi-ADSD with Timestamps*. Esta interface não está diretamente relacionada com a ferramenta para alta disponibilidade do servidor Web, ao invés disso, ela é uma interface genérica para a infra estrutura de diagnóstico baseada no algoritmo *Hi-ADSD with Timestamps*.

4.4.1 Funcionamento da Interface

As informações contidas na MIB podem ser visualizadas em uma página Web. A cada acesso a esta página, as informações são extraídas de um dos testadores executando o algoritmo *Hi-ADSD with Timestamps*.

Como se trata de uma interface Web, qualquer máquina pode ter a interface instalada, para tal, a máquina deve ter um servidor Web instalado. A partir de uma máquina escolhida, pode-se obter informações de qualquer testador executando o algoritmo. Para isso, a interface obtém as informações consultando a MIB da máquina escolhida. Estas consultas à MIB são feitas através dos comandos *snmpwalk* e *snmpget* do pacote NET-SNMP.

As informações apresentadas na interface são obtidas através de um conjunto de scripts que realizam uma chamada para o comando *snmpwalk*, comando que exibe todo o conteúdo da MIB mostrando na saída padrão as informações desejadas no formato de um arquivo CSV (*comma separated values* ou valores separados por ponto-e-vírgula). Estes scripts realizam uma filtragem do resultado da execução deste comando. A saída do script, por sua vez, é lida por uma página PHP cada vez que a página é requisitada ou atualizada, mostrando as informações de maneira adequada. Esta atualização da página pode ser feita automaticamente em um período configurado na própria página. O fluxo dos dados da MIB de um testador é representado e mostrado na figura 4.6.

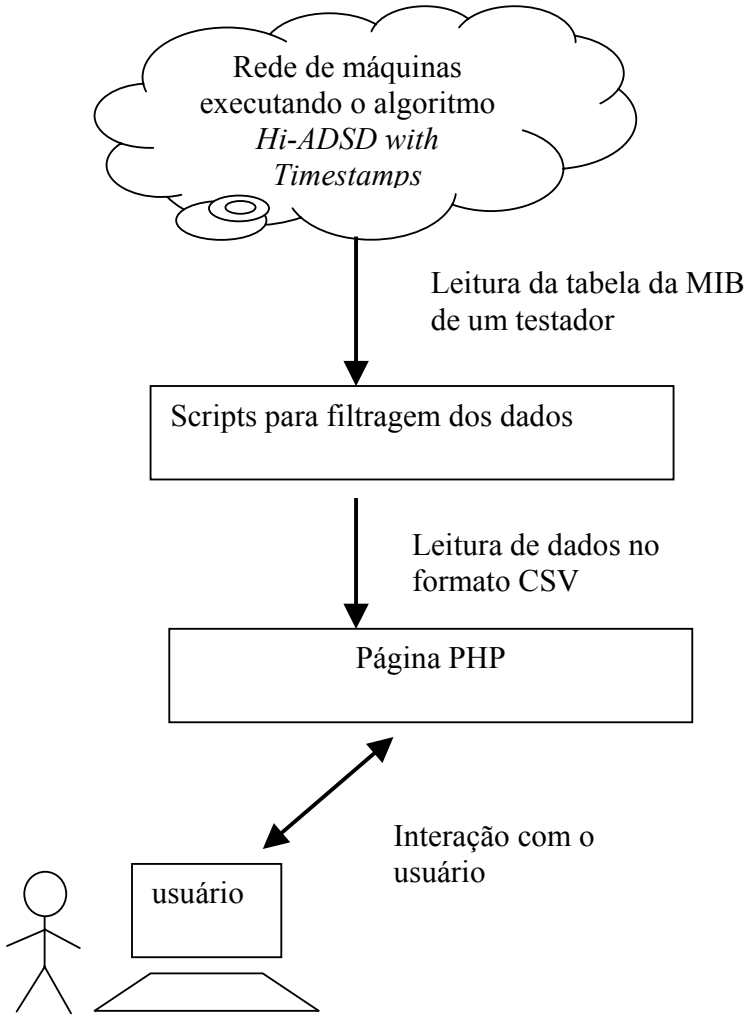


Figura 4.6: Fluxo dos dados da MIB de um testador

até a disponibilização para o usuário.

4.4.2 Utilização da Interface

Para acessar a interface é necessário conhecer o nome ou endereço IP de pelo menos uma máquina pertencente ao grupo de máquinas que executam o algoritmo *Hi-ADSD with*

Timestamps. Caso nenhuma seja especificada, a própria máquina que possui a interface instalada será considerada como a máquina inicial.

A partir desta máquina inicial, os dados adequados da MIB são lidos, gerando a página Web. Um exemplo é mostrado na figura 4.7.

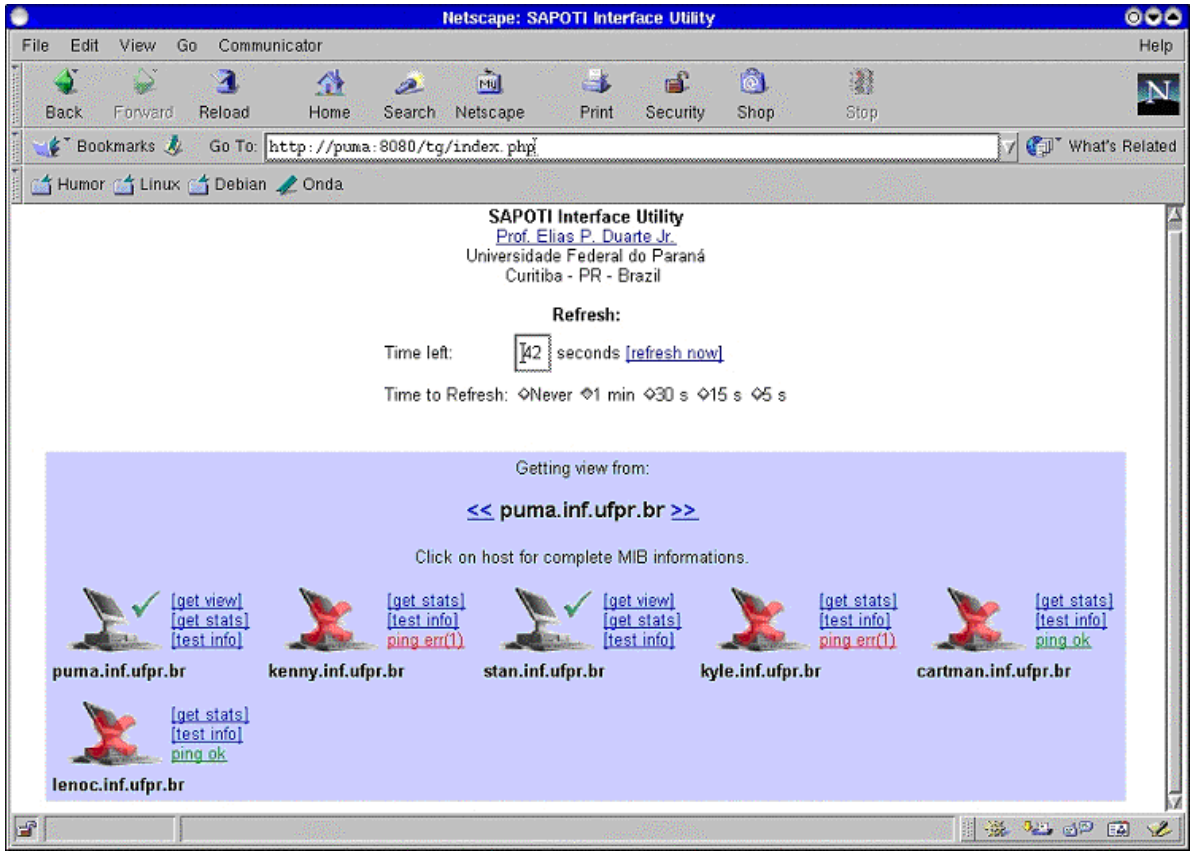


Figura 4.7: A interface.

As máquinas são representadas pelos ícones de computadores, que indicam o estado destas máquinas, que pode ser falho, sem-falha ou estado desconhecido (inicializando), que é mostrado respectivamente através das figuras 4.8.


Estado com falha


Estado sem-falha


Estado desconhecido

Figura 4.8: Ícones representando os estados das máquinas.

Para cada máquina há, além do estado, um conjunto de informações adicionais. Caso a máquina esteja falha, é indicado o resultado da execução do comando *ping* [1]. Caso o resultado do comando *ping* indique que a máquina está no ar, a máquina está apenas com alguns serviços indisponíveis (em particular, o agente SNMP), mas continua na rede.

Para cada máquina, caso a máquina esteja sem-falha, a interface indica os resultados dos testes secundários, indicando também o número de testes falhos para cada categoria, como é mostrado nas figuras 4.9 e 4.10.

Figura 4.9: Erro em testes tipo SERVICE.

Figura 4.10: Informações sobre o PING.

É possível alterar o testador do qual se lê a MIB a partir de qualquer testador executando algoritmo *Hi-ADSD with Timestamps*, para isto existe um link *get view* ao lado de cada ícone que altera a "visão" para esta outra máquina. É possível também detalhar cada teste realizado por cada máquina, assim como as estatísticas de seu estado. Para ambos os casos existe um link, respectivamente *tests info* e *get stats*, ao lado do ícone que abre uma nova janela com as informações desejadas como, por exemplo, as janelas mostradas respectivamente através das figuras 4.11 e 4.12.

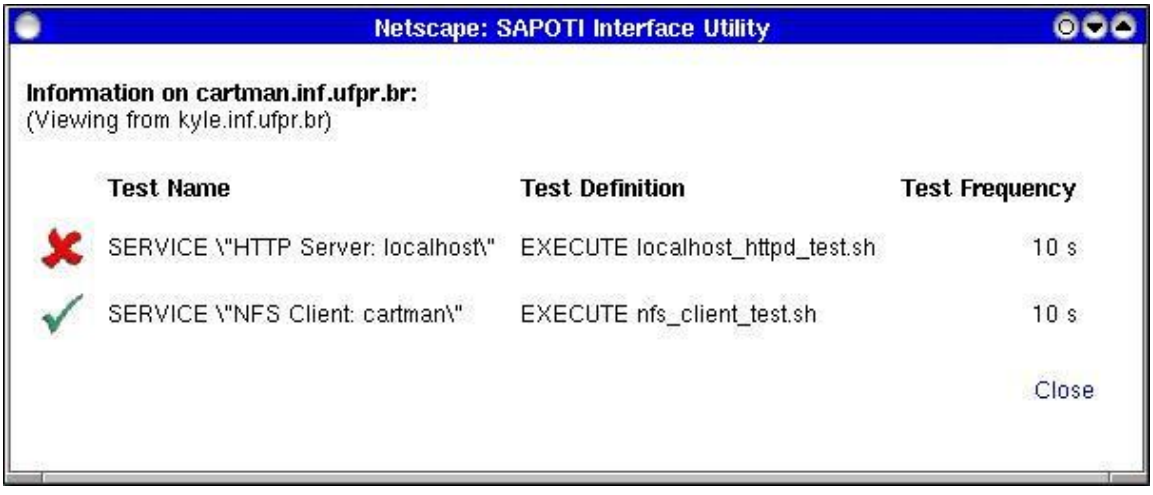


Figura 4.11: Janela com informações sobre os testes.

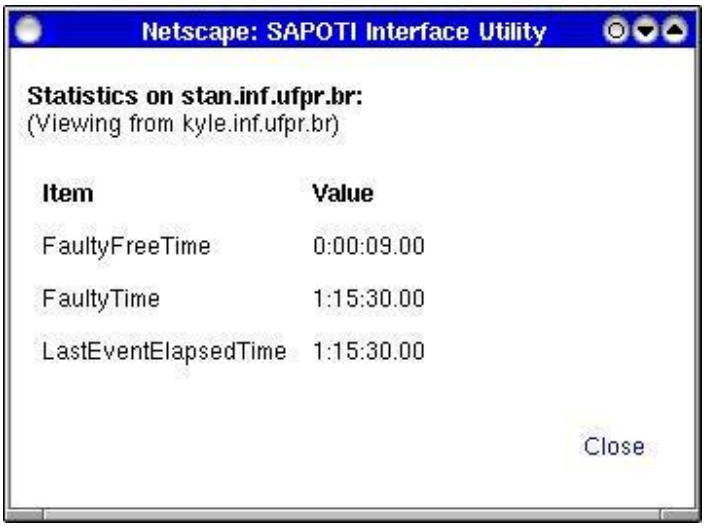


Figura 4.12: Janela com estatísticas dos estados.

4.4.3 Interface para Configuração Remota

A interface contém uma página Web que permite a configuração remota dos arquivos de configuração do algoritmo *Hi-ADSD with Timestamps*. Esta configuração é atualizada somente na máquina alterada, ou seja, sem a propagação das alterações para outras máquinas. Esta interface de configuração remota é mostrada na figura 4.13.

Esta pagina Web é um módulo opcional e tem como objetivo fornecer um serviço mais conveniente para administradores da rede. A interface tem maior praticidade se utilizada em uma rede NFS (*Network File System*) [25, 26] ou em uma rede que utilize o serviço RSYNC.

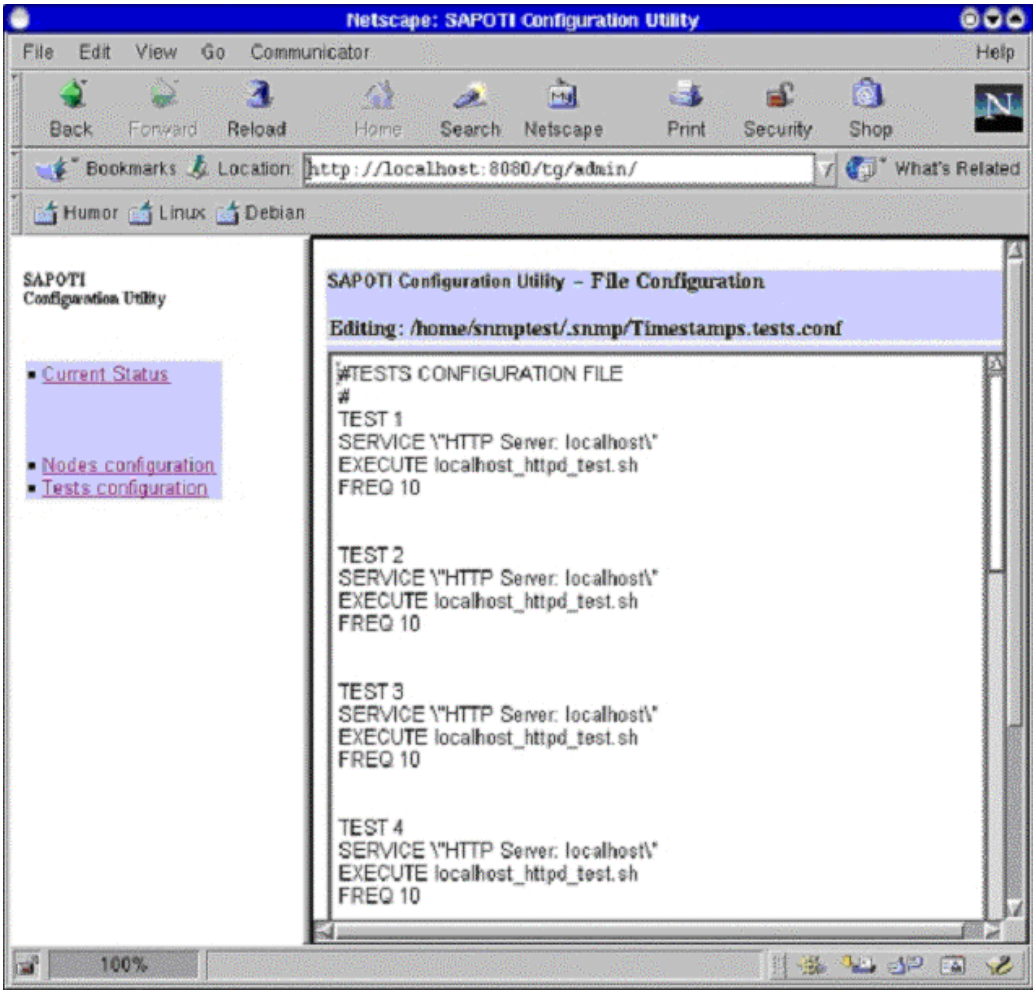


Figura 4.13: Exemplo de interface para configuração remota

do algoritmo *Hi-ADSD with Timestamps*.

Capítulo 5

Resultados Experimentais

Neste capítulo descrevemos os resultados obtidos através da realização de dois experimentos com a ferramenta SAPOTI. Estes dois experimentos foram realizados em um período de cerca de 12 horas. No primeiro experimento foi simulada uma configuração de rede com escalonamento de falhas frequentes. Este primeiro experimento ocasiona inclusive a falha simultânea de todas as máquinas, causando a indisponibilidade do servidor Web. No segundo experimento foi simulada uma configuração de rede na qual, apesar de algumas máquinas falharem frequentemente, havia a garantia de sempre uma máquina não estar falha e em condições de assumir a tarefa de executar o servidor Web.

5.1 Primeiro Experimento

No primeiro experimento foram usadas seis máquinas. As falhas das máquinas foram simuladas através do encerramento do processo do agente SNMP. Uma máquina é considerada sem-falha quando executa o agente SNMP. As máquinas foram configuradas de forma a falhar frequentemente, permanecendo sem-falha de cerca de 30% a cerca de 85% do tempo do experimento, dependendo da máquina. Ocasionalmente todas as máquinas podem estar falhas ocasionando a indisponibilidade do servidor Web. A máquina

sem-falha de menor identificador disponibiliza o servidor Web obtendo o endereço IP virtual configurado.

As máquinas utilizadas no experimento estavam em uma rede Ethernet, executando TCP/IP e NFS. Todas as máquinas rodavam o sistema operacional Linux Debian [27]. Em todas as máquinas estava instalada a versão 4.2.1 do pacote NET-SNMP [7, 18]. Qualquer uma das seis máquinas estava apta a disponibilizar o servidor Web. O servidor Web usado no experimento foi o Apache. Os testes do experimento foram feitos e monitorados durante 12 horas e 23 minutos. Durante a monitoração, os dados eram coletados de 5 em 5 segundos. Toda a monitoração do experimento foi realizada através de scripts escritos na linguagem *bash script* que geravam arquivos de *logging* para a análise dos dados.

5.1.1 Máquinas Utilizadas no Primeiro Experimento

As máquinas utilizadas no primeiro experimento são mostradas na tabela 5.1, bem como seus identificadores associados. Todas as máquinas consideradas no experimento executavam a ferramenta SAPOTI.

Puma – máquina número 01
Kenny – máquina número 02
Stan – máquina número 03
Kyle – máquina número 04
Cartman – máquina número 05
Lenoc – máquina número 06

Tabela 5.1: Máquinas utilizadas no primeiro experimento e seus identificadores.

Com a disposição das máquinas mostradas na tabela 5.1, uma máquina *a* tem maior prioridade que uma outra máquina *b* quando o identificador da máquina *a* for menor que o identificador da máquina *b*.

5.1.2 Injeção de Falhas das Máquinas no Primeiro Experimento

A frequência com que as máquinas utilizadas no experimento falhavam e se recuperavam foi previamente definida. Estas disposições de falhas foram definidas como mostra a tabela 5.2.

Puma	5 minutos no ar 10 minutos fora do ar
Kenny	8 minutos no ar 9 minutos fora do ar
Stan	13 minutos no ar 7 minutos fora do ar
Kyle	17 minutos no ar 6 minutos fora do ar
Cartman	21 minutos no ar 5 minutos fora do ar
Lenoc	25 minutos no ar 4 minutos fora do ar

Tabela 5.2: Configuração das falhas das máquinas no primeiro experimento.

Ao iniciar os testes, cada máquina tinha sua disposição de falhas pré-configurada, por exemplo, se em um dado momento a máquina *puma* já estivesse falha há 2 minutos, ela se recuperaria em 8 minutos; se a máquina *kenny* estivesse no ar há 4 minutos, ela iria falhar em 4 minutos; se a máquina *stan* acabou de falhar, ela se recuperaria em 7 minutos, e assim por diante.

Em um dado momento a máquina responsável por disponibilizar o servidor Web é a máquina sem-falha com menor identificador. Quando ao menos a máquina *puma* está sem-falha, a máquina *puma* disponibiliza o servidor Web. Se a máquina *puma* estiver falha e ao menos a máquina *kenny* não estiver falha, a máquina *kenny* será a responsável por disponibilizar o servidor Web. Se as máquinas *puma* e *kenny* estiverem falhas e ao menos a máquina *stan* não estiver falha, a máquina *stan* disponibiliza o servidor Web, e assim por diante.

5.1.3 Monitoramento Parcial do Experimento

A seguir é descrito o resultado do monitoramento parcial realizado durante 21 minutos, iniciado após 3 horas e 10 minutos do início do experimento. No início deste monitoramento parcial, a máquina *puma*, a máquina *kyle* e a máquina *cartman* estavam sem-falhas e a máquina *kenny*, a máquina *stan* e a máquina *lenoc* estavam falhas. Para melhor descrever este segmento do monitoramento, definiremos este momento inicial como sendo o tempo 00:00 (mm:ss). A seguir é apresentado um *logging* do experimento em fonte destacada.

```
No momento 00:00 a máquina puma está disponibilizando o
servidor Web.
No tempo 00:05 a máquina lenoc entra no ar.
```

Como a máquina *puma* tem maior prioridade que a máquina *lenoc*, a máquina *lenoc* que acabou de entrar no ar não inicializa o servidor Web.

```
No tempo 00:20 a máquina stan entra no ar.
```

Novamente, como a máquina *puma* tem maior prioridade que a máquina *stan*, a máquina *stan* que acabou de entrar no ar não inicializa o servidor Web.

No tempo 01:12 a máquina *kyle* fica falha.

Como não era a máquina *kyle* que disponibilizava o servidor Web, sua falha não influencia e não ocasiona nenhuma mudança nas máquinas consideradas no experimento.

No tempo 01:27 a máquina *cartman* fica falha.

Novamente, como não era a máquina *cartman* que disponibilizava o servidor Web, sua falha não influencia e não ocasiona nenhuma mudança nas máquinas consideradas no experimento.

No tempo 03:43 a máquina *puma* fica falha.

A máquina *puma* estava com o servidor Web iniciado. Com sua falha, o servidor Web se torna indisponível.

No tempo 04:12 o servidor Web se torna indisponível na máquina *puma*.

No tempo 04:19 a máquina *stan* inicializa o servidor Web.

Neste momento, a máquina *stan* detecta pela ordem de prioridades que deve iniciar o servidor Web e o faz.

No tempo 05:26 a máquina *kenny* entra no ar.

No tempo 05:49 o servidor Web se torna indisponível na máquina *stan*.

A máquina *stan* estava com o servidor Web iniciado. Como ela detectou que a máquina *kenny* tinha entrado no ar, e como a máquina *kenny* tem maior prioridade que a máquina *stan*, a máquina *stan* torna o servidor Web indisponível.

No tempo 05:57 a máquina *kenny* inicializa o servidor Web.

Neste momento, a máquina *kenny* detecta que ela deve iniciar o servidor Web e o inicia.

No tempo 06:31 a máquina *cartman* entra no ar.

No tempo 07:18 a máquina *kyle* entra no ar.

No tempo 13:23 a máquina *stan* fica falha.

No tempo 13:48 a máquina *puma* entra no ar.

No tempo 14:00 a máquina *puma* inicializa o servidor Web.

Neste momento, a máquina *puma* detecta que deve iniciar o servidor Web e o inicia. Mesmo a máquina *kenny* ainda não tendo detectado que deveria retirar o servidor Web do ar, ao iniciar o servidor Web na máquina *puma*, automaticamente todas as requisições que forem feitas serão atendidas corretamente, por uma das duas máquinas.

No tempo 14:16 a máquina *kenny* retira o servidor Web do ar.

Como a máquina *kenny* que disponibilizava o servidor Web detectou que a máquina *puma* já tinha entrado no ar, e como a máquina *puma* tem maior prioridade que a máquina *kenny*, a máquina *kenny* retira o servidor Web do ar.

No tempo 14:29 a máquina *kenny* fica falha.

No tempo 18:54 a máquina *puma* fica falha.

No tempo 19:34 o servidor Web se torna indisponível na máquina *puma*.

A máquina *puma* estava com o servidor Web iniciado. Com sua falha, o servidor Web se torna indisponível.

No tempo 19:46 a máquina *kyle* inicializa o servidor Web.

Neste momento, a máquina *kyle* detecta que a máquina *puma* está falha e inicia o servidor Web.

5.1.4 Análise dos Resultados do Primeiro Experimento

Durante todo o primeiro experimento, o servidor Web trocou de máquina 177 vezes, como mostrado na tabela 5.3. Destas 177 vezes em que o servidor Web trocou de máquina, 72 vezes a troca foi ocasionada pela falha do servidor Web, ocasionando a inicialização do servidor Web em outra máquina da rede. As demais trocas foram ocasionadas pela recuperação de máquinas com maior prioridade que a máquina que disponibilizava o servidor Web no momento desta recuperação. Assim, a máquina que acabou de se recuperar inicializa o servidor Web sem que nenhuma requisição Web deixe de ser atendida.

Número de Vezes que o servidor Web trocou de máquina
177 vezes

Tabela 5.3: Número de vezes que o servidor Web trocou de máquina.

A figura 5.1 mostra o número de vezes que cada máquina utilizada no experimento falhou. Nesta figura notamos que a máquina *puma* falhou e se recuperou 50 vezes, a máquina *kenny* falhou e se recuperou 44 vezes. Já a máquina *lennoc* esteve falha 26 vezes. A porcentagem do tempo do experimento em que as máquinas ficaram falhas é mostrada na figura 5.2. Podemos notar que a máquina *puma* ficou falha 68,4% do tempo do experimento, já a *kyle* ficou 26,3% do tempo fora do ar durante o experimento.

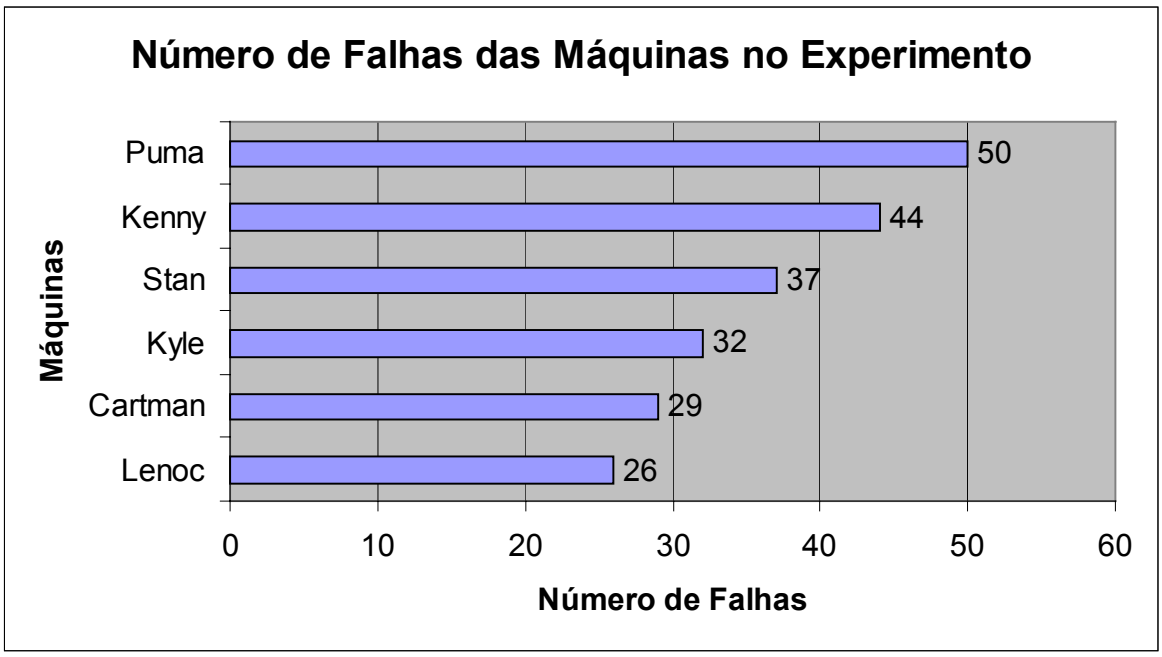


Figura 5.1: Número de falhas de cada máquina no primeiro experimento.

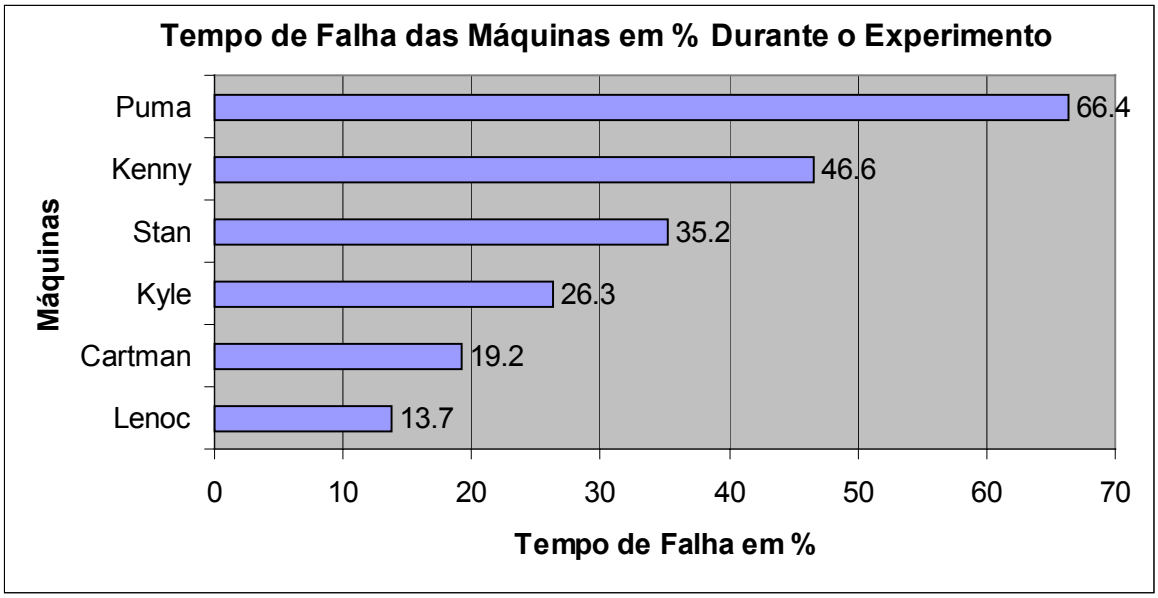


Figura 5.2: Porcentagem do tempo do experimento em que as máquinas ficaram falhas.

A figura 5.3 mostra a latência de recuperação do servidor Web. Podemos notar que 20 vezes das 72 vezes em que o servidor Web falhou, o tempo de recuperação foi de 7 segundos, ou seja, aproximadamente 28% das vezes em que o servidor Web falhou, o tempo de recuperação foi de 7 segundos. No melhor caso a recuperação do servidor Web após a máquina que disponibilizava o servidor falhar foi em 6 segundos. No pior caso a recuperação deu-se em 53 segundos. A média ponderada do tempo de recuperação do servidor Web foi de 16,4 segundos.

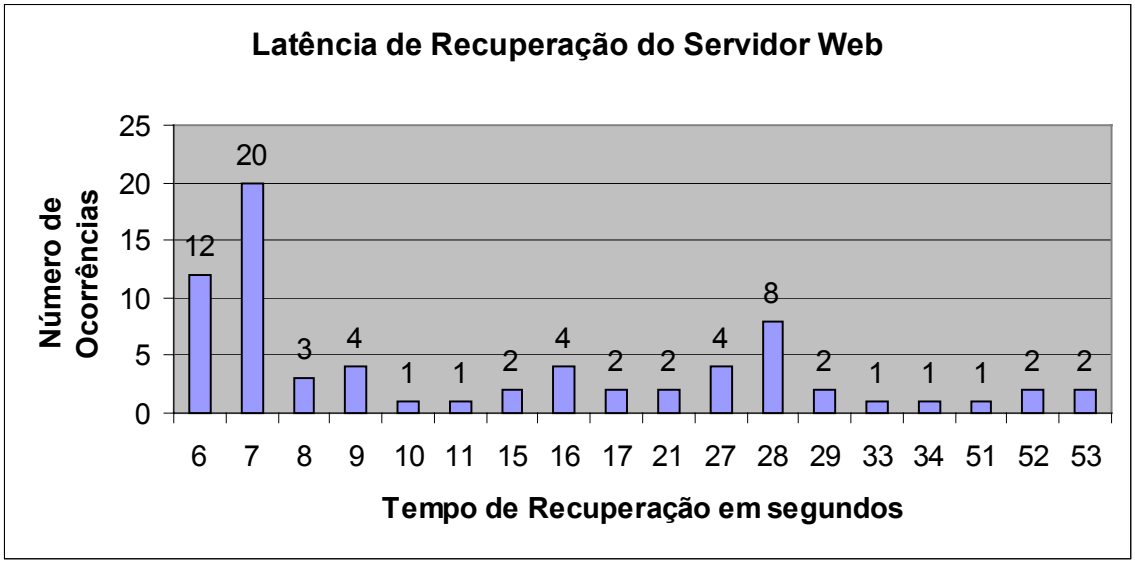


Figura 5.3: Latência de recuperação do servidor Web.

Durante o experimento, 54% das vezes que o servidor Web caiu, o tempo de recuperação foi menor que 10 segundos e 90% das vezes que o servidor Web caiu o tempo de recuperação foi menor que 30 segundos, como mostrado na figura 5.4. Deve ser considerado que a frequência configurada no algoritmo *Hi-ADSD with Timestamps* para a execução dos testes da MIB foi de 10 segundos. Deve ser considerado também que durante

a monitoração, os dados eram coletados de 5 em 5 segundos e que, na implementação da ferramenta SAPOTI existe uma intervalo de 10 segundos entre cada iteração do algoritmo. Com a diminuição do tempo destes intervalos, podemos conseguir uma maior precisão na detecção e disponibilização do servidor Web.

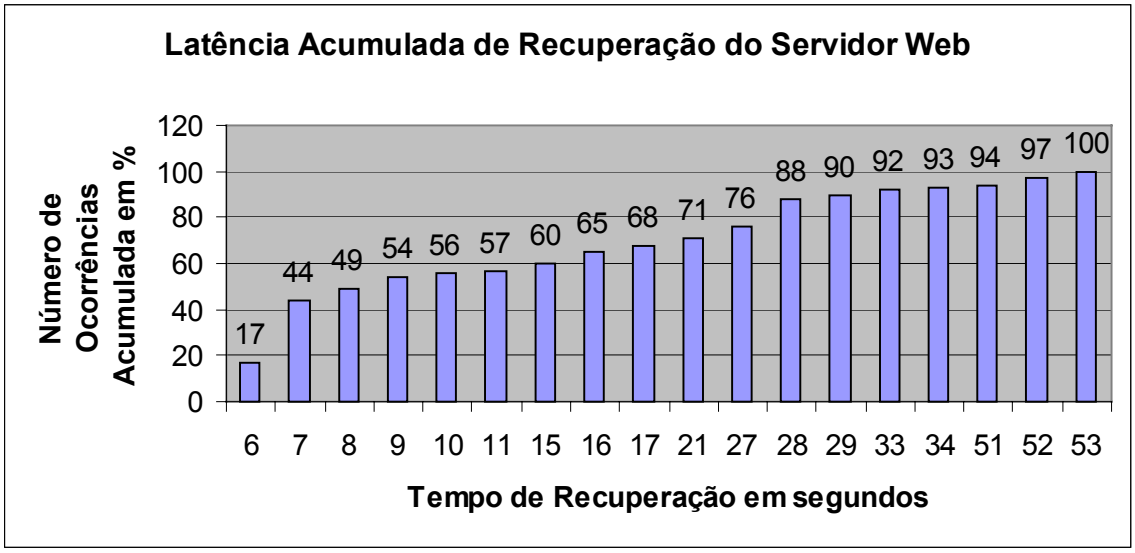


Figura 5.4: Latência acumulada de recuperação do servidor Web.

Durante o experimento, o servidor Web foi iniciado na máquina *Kenny* por 53 vezes, e foi iniciado na máquina *Lenoc* por 2 vezes, como mostrado na figura 5.5. No total o servidor Web trocou de máquina 177 vezes.

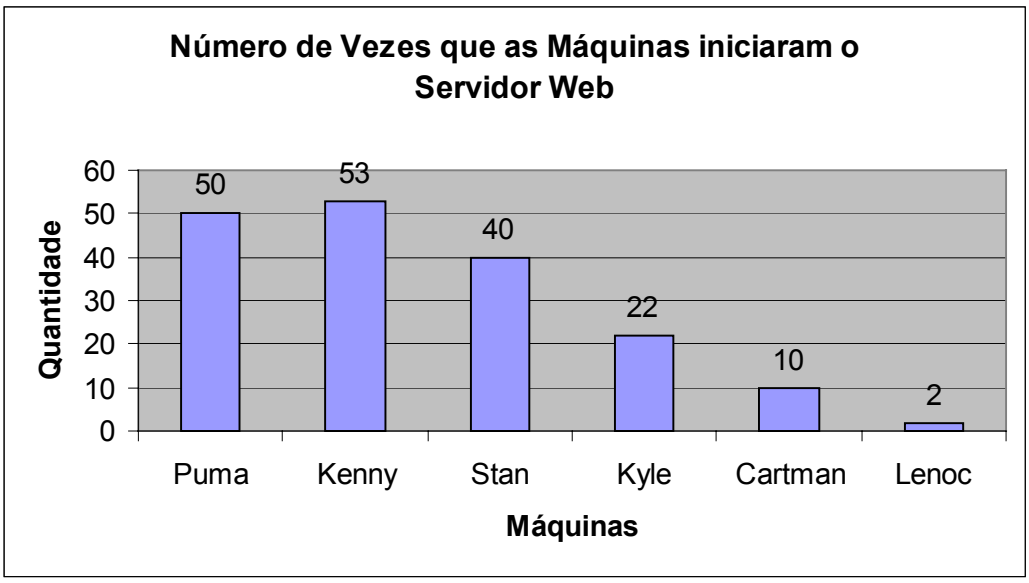


Figura 5.5: Número de vezes que cada máquina iniciou o servidor Web.

Mesmo com um ambiente com muitas falhas distribuídas entre todas as máquinas, a disponibilidade do servidor Web durante o experimento foi de 97,35% do tempo do experimento, como mostrado na figura 5.6. Portanto, durante as 12 horas e 23 minutos do experimento a indisponibilidade do servidor Web foi de apenas 2,65%, ou seja, não passou de 20 minutos. Deve ser frisado que durante parte deste tempo todas as máquinas estavam falhas, o que não ocorre no próximo experimento descrito.

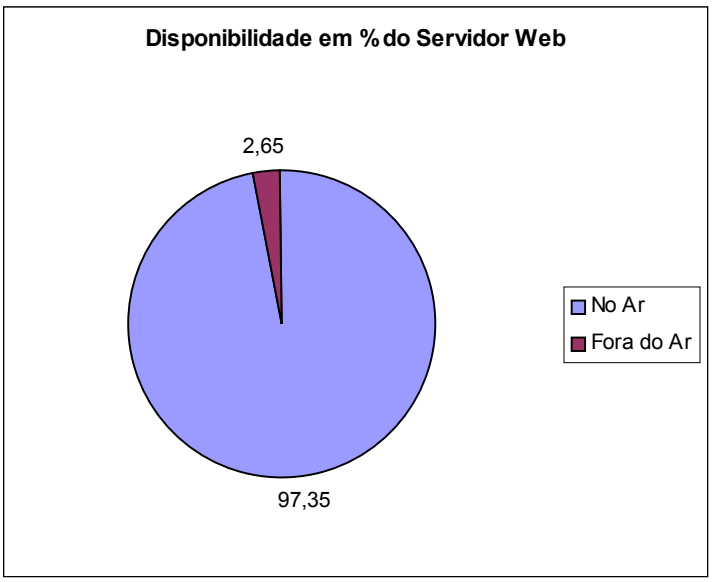


Figura 5.6: Disponibilidade do servidor Web

durante o primeiro experimento.

5.2 Segundo Experimento

Neste experimento foram usadas cinco máquinas. As falhas das máquinas foram simuladas através do encerramento do processo do agente SNMP. Uma máquina é considerada sem-falha quando executa o agente SNMP. As máquinas foram configuradas de forma a sempre existir uma máquina sem-falha em qualquer momento do experimento e não permitindo que mais de uma máquina fique falha no mesmo instante, mais de uma máquina se recupere no mesmo instante e ainda não permitindo que uma máquina fique falha enquanto outra está se recuperando de uma falha. Assim, existe a garantia de que sempre existirá uma máquina em condições de disponibilizar o servidor Web. A máquina de menor identificador disponibiliza o servidor Web obtendo o endereço IP virtual configurado.

As máquinas utilizadas no experimento estavam em uma rede Ethernet, executando TCP/IP e NFS. Todas as máquinas rodavam o sistema operacional Linux Debian. Em todas as máquinas estava instalada a versão 4.2.1 do pacote NET-SNMP. Qualquer uma das cinco máquinas estava apta a disponibilizar o servidor Web, que foi o Apache. Os testes do experimento foram feitos e monitorados durante 12 horas e 40 minutos. Durante a monitoração, os dados eram coletados de 5 em 5 segundos. Toda a monitoração do experimento foi realizada através de scripts escritos na linguagem bash script que geravam arquivos de *logging* para a análise dos dados.

5.2.1 Máquinas Utilizadas no Segundo Experimento

As máquinas utilizadas no segundo experimento são mostradas na tabela 5.4, bem como seus identificadores associados. Todas as máquinas consideradas no experimento executavam a aplicação SAPOTI.

Puma – máquina número 01
Kenny – máquina número 02
Stan – máquina número 03
Kyle – máquina número 04
Cartman – máquina número 05

Tabela 5.4: Máquinas utilizadas no segundo experimento e seus identificadores.

Com a disposição das máquinas mostrada na tabela 5.4, uma máquina *a* tem maior prioridade que uma outra máquina *b* quando o identificador da máquina *a* for menor que o identificador da máquina *b*.

5.2.2 Injeção de Falhas das Máquinas no Segundo Experimento

A frequência com que as máquinas utilizadas no experimento falhavam e se recuperavam foi previamente definida. Estas disposições de falhas foram definidas como mostrado na tabela 5.5.

Puma	20 minutos no ar 40 minutos fora do ar
Kenny	40 minutos no ar 20 minutos fora do ar
Stan	sempre no ar
Kyle	sempre no ar
Cartman	sempre no ar

Tabela 5.5: Configuração das falhas das máquinas no segundo experimento.

Ao iniciar os testes, cada máquina tinha sua disposição de falhas pré-configurada, por exemplo, se em um dado momento a máquina *puma* já estivesse no ar há 10 minutos, ela

iria falhar em 10 minutos; se a máquina *kenny* estivesse falha há 15 minutos, ela se recuperaria em 5 minutos, e assim por diante.

Em um dado momento a máquina responsável por disponibilizar o servidor Web é a máquina sem-falha com menor identificador. Quando ao menos a máquina *puma* está sem-falha, ela disponibiliza o servidor Web. Se a máquina *puma* estiver falha e ao menos a máquina *kenny* não estiver falha, a máquina *kenny* será a responsável por disponibilizar o servidor Web, e assim por diante.

5.2.3 Monitoramento Parcial do Experimento

A seguir é descrito o resultado do monitoramento parcial realizado durante 1 hora e 40 minutos, iniciado após 3 horas e 10 minutos do início do experimento. No início deste monitoramento parcial, a máquina *kenny*, a máquina *stan*, a máquina *kyle* e a máquina *cartman* estavam sem-falhas e a máquina *puma* estava falha. Para melhor descrever este segmento do monitoramento, definiremos este momento inicial como sendo o tempo 00:00 (mm:ss). A seguir é apresentado um *logging* do experimento em fonte destacada.

```
    Neste momento a máquina kenny está disponibilizando o
servidor Web.
    No tempo 00:14:09 a kenny fica falha.
```

A máquina *kenny* estava com o servidor Web iniciado. Com sua falha, o servidor Web se torna indisponível.

No tempo 00:14:31 o servidor Web se torna indisponível na máquina *kenny*.

No tempo 00:14:44 a máquina *stan* inicializa o servidor Web.

Neste momento, a máquina *stan* detecta pela ordem de prioridades que deve iniciar o servidor Web e o faz.

No tempo 00:34:19 a *puma* entra no ar.

No tempo 00:34:32 a *puma* inicializa servidor Web.

Neste momento, a máquina *puma* detecta pela ordem de prioridades que deve iniciar o servidor Web e o inicia. Mesmo a máquina *kenny* ainda não tendo detectando que deveria retirar o servidor Web do ar, ao iniciar o servidor Web na máquina *puma*, automaticamente todas as requisições que forem feitas serão atendidas corretamente, por uma das duas máquinas.

No tempo 00:34:46 a *stan* retira o servidor Web do ar.

Como a *stan* estava com o servidor Web iniciado e detectou que a *puma* já tinha entrado no ar, e como a *puma* tem maior prioridade que a *stan*, a *stan* retira o servidor Web do ar.

No tempo 00:44:13 a *kenny* entra no ar.

No tempo 00:54:24 a *puma* fica falha.

No tempo 00:54:40 o servidor Web se torna indisponível na máquina *puma*.

A máquina *puma* estava com o servidor Web iniciado. Com sua falha, o servidor Web sai do ar.

No tempo 00:54:46 a máquina *kenny* inicializa o servidor Web.

Neste momento, a máquina *kenny* detecta que deve iniciar o servidor Web e o inicia.

No tempo 01:14:17 a máquina *kenny* fica falha.

No tempo 01:14:38 o servidor Web se torna indisponível na máquina *kenny*.

A máquina *kenny* estava com o servidor Web iniciado. Com sua falha, o servidor Web se torna indisponível.

No tempo 01:14:45 a máquina *stan* inicializa servidor Web.

A máquina *stan* detecta que deve iniciar o servidor Web e o inicia.

No tempo 01:34:27 a máquina *puma* entra no ar.

No tempo 01:34:41 a máquina *puma* inicializa o servidor Web.

A máquina *puma* detecta que deve iniciar o servidor Web e o inicia.

No tempo 01:34:42 o servidor Web se torna indisponível na máquina *stan*.

A máquina *stan* estava com o servidor Web iniciado. Como ela detectou que a máquina *puma* já tinha entrado no ar, e como a máquina *puma* tem maior prioridade que a máquina *stan*, a máquina *stan* torna o servidor Web indisponível.

5.2.4 Análise dos Resultados do Segundo Experimento

Durante todo o experimento, o servidor Web trocou de máquina 41 vezes, como mostrado na tabela 5.6. Destas 41 vezes, 14 vezes a troca foi ocasionada pela falha do servidor Web, ocasionando a inicialização do servidor Web em outra máquina da rede. As demais trocas foram ocasionadas pela recuperação de máquinas com maior prioridade que a máquina que disponibilizava o servidor Web no momento desta recuperação. Assim, a máquina que acabou de se recuperar inicializa o servidor Web sem que nenhuma requisição Web deixe de ser atendida.

Número de Vezes que o servidor Web trocou de máquina
41 vezes

Tabela 5.6: Número de vezes que o servidor Web trocou de máquina.

A figura 5.7 mostra o número de vezes que cada máquina utilizada no experimento falhou e se recuperou. Nesta figura notamos que a máquina *puma* falhou e se recuperou 14 vezes, a máquina *kenny* falhou e se recuperou 13 vezes. Já a máquina *stan*, a máquina *kyle* e a máquina *cartman* não falharam em nenhum momento durante o experimento. A porcentagem do tempo do experimento em que as máquinas ficaram falhas é mostrada na figura 5.8. Podemos notar que a máquina *puma* ficou falha 68,5% do tempo do experimento e a máquina *kenny* ficou falha 51,4% do tempo do experimento. A máquina *stan*, a máquina *kyle* e a máquina *cartman* não falharam durante o experimento.

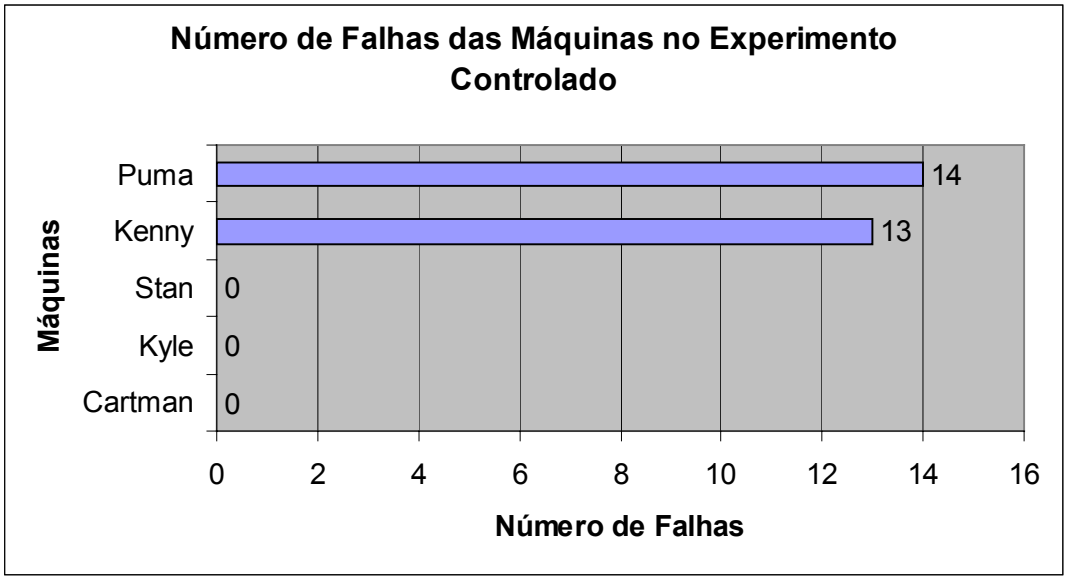


Figura 5.7: Número de falhas de cada máquina no segundo experimento.

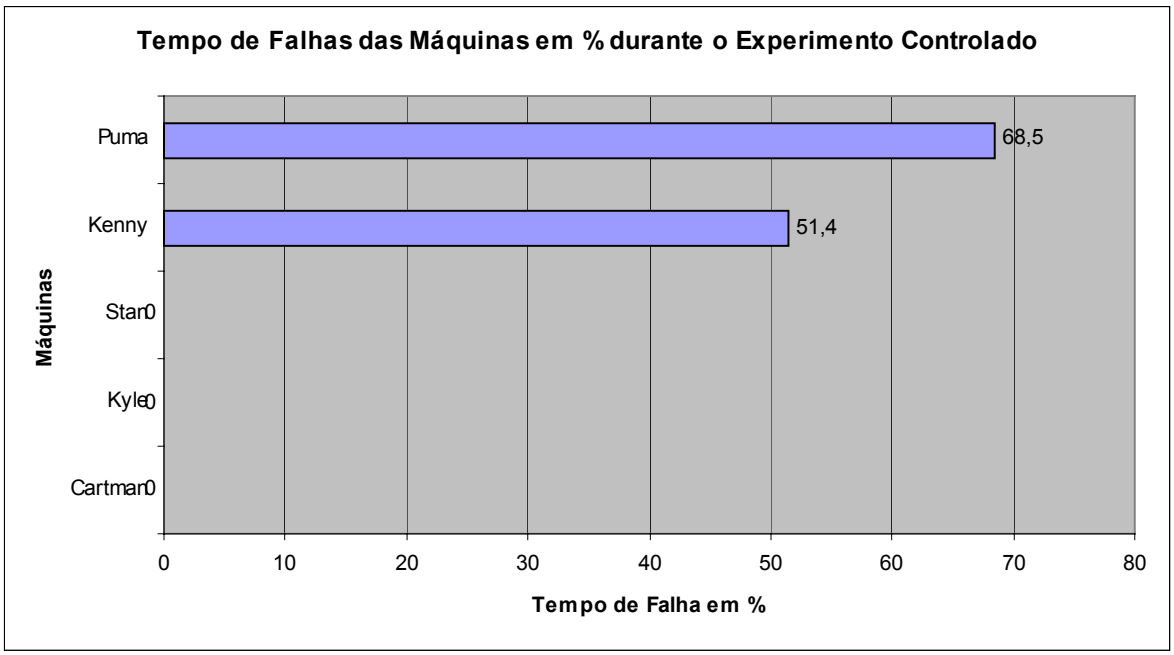


Figura 5.8: Porcentagem do tempo do experimento em que as máquinas ficaram falhas.

A figura 5.9 mostra a latência de recuperação do servidor Web. Podemos notar que 7 vezes das 14 vezes em que o servidor Web falhou o tempo de recuperação foi de 8 segundos, ou seja, 50% das vezes em que o servidor Web falhou o tempo de recuperação foi de 8 segundos. No melhor caso a recuperação do servidor Web após a máquina que disponibilizava o servidor Web falhar foi de 7 segundos. No pior caso a recuperação deu-se em 29 segundos. A média ponderada do tempo de recuperação do servidor Web foi de 14,0 segundos.

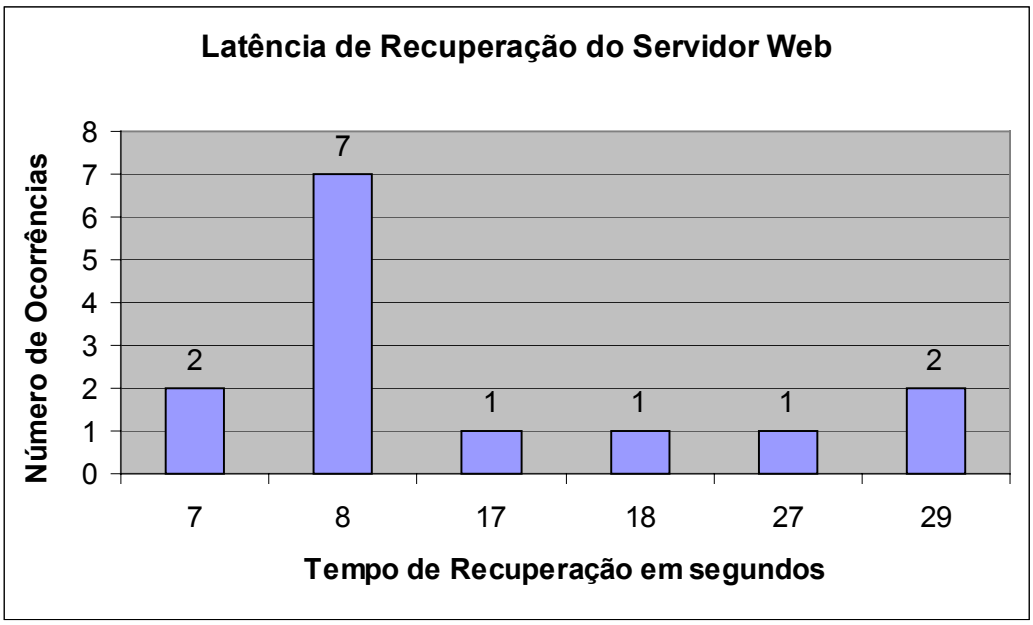


Figura 5.9: Latência de recuperação do servidor Web.

Durante o experimento, 64% das vezes que o servidor Web caiu o tempo de recuperação foi menor que 10 segundos e 100% das vezes que o servidor Web caiu o tempo de recuperação foi menor que 30 segundos, como mostrado na figura 5.10. Deve ser considerado que a frequência configurada no algoritmo *Hi-ADSD with Timestamps* para a execução dos testes da MIB foi de 10 segundos. Deve ser considerado também que durante a monitoração, os dados eram coletados de 5 em 5 segundos e que, na implementação da ferramenta SAPOTI existe uma intervalo de 10 segundos entre cada iteração do algoritmo. Com a diminuição do tempo destes intervalos, podemos conseguir uma maior precisão na detecção e disponibilização do servidor Web.

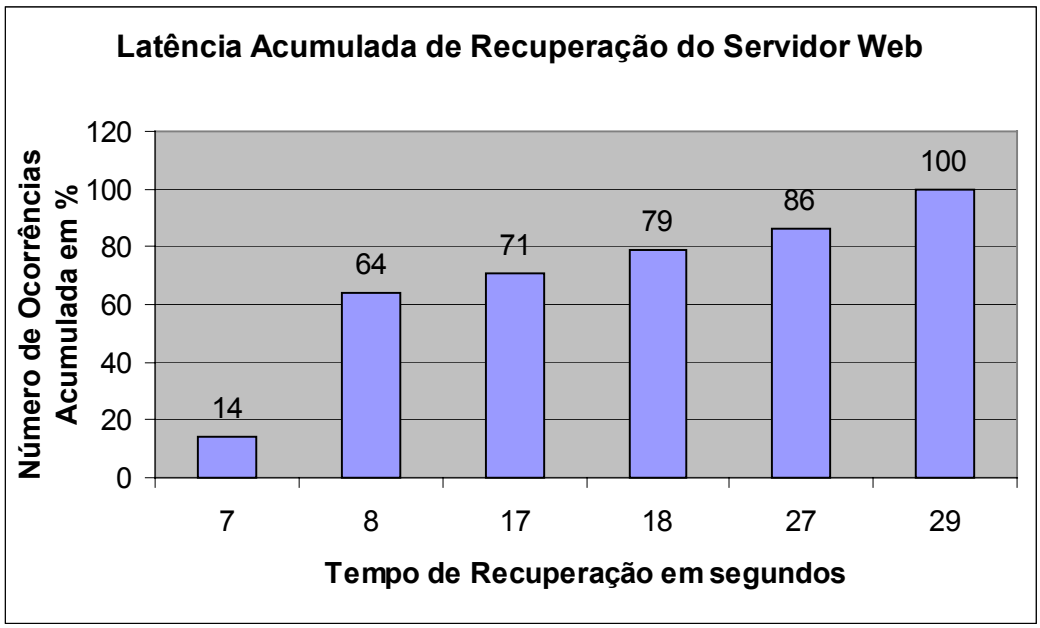


Figura 5.10: Latência acumulada de recuperação do servidor Web.

Durante o experimento, tivemos o servidor Web foi iniciado na máquina *Puma* e na máquina *Kenny* por 14 vezes, e também tivemos o servidor Web iniciado na máquina *Stan* por 13 vezes, como mostrado na figura 5.11. No total o servidor Web trocou de máquina 41 vezes.

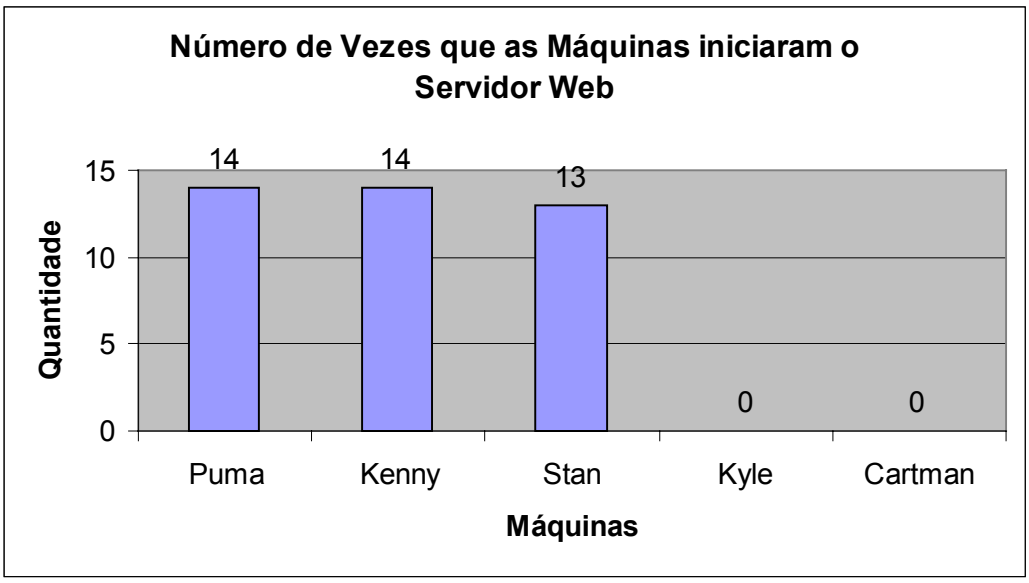


Figura 5.11: Número de vezes que cada máquina iniciou o servidor Web.

Mesmo em um ambiente mais controlado, mas com muitas falhas em algumas das máquinas, a disponibilidade do servidor Web durante o experimento foi de 99,58% do tempo do experimento, como mostrado na figura 5.12. Portanto, durante as 12 horas e 40 minutos do experimento a indisponibilidade do servidor Web foi de apenas 0,41%, ou seja, não passou de 4 minutos.

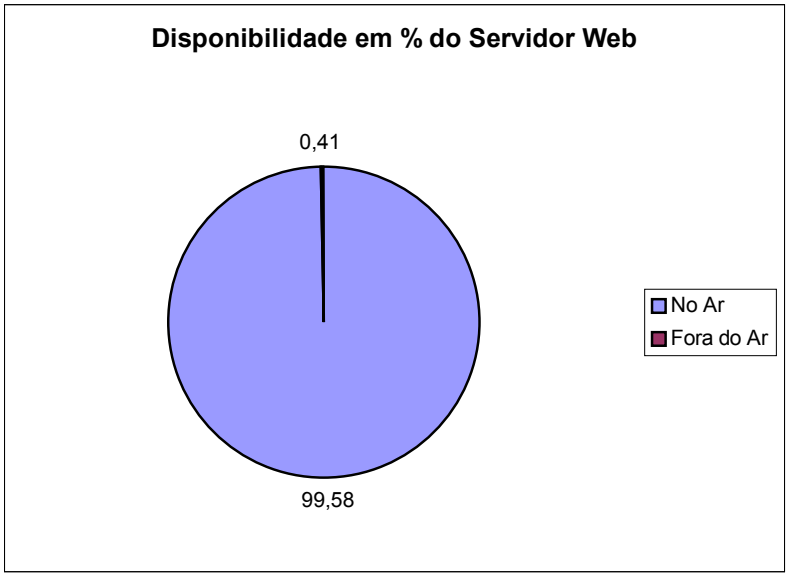


Figura 5.12: Disponibilidade do servidor Web

durante o segundo experimento.

Capítulo 6

Conclusão

Neste trabalho foi apresentada uma estratégia distribuída que garante a alta disponibilidade de servidores de aplicações TCP/IP aplicada especificamente para a implementação de um servidor Web tolerante a falhas. A ferramenta SAPOTI atua de forma distribuída em uma rede de computadores monitorados pelo algoritmo *Hi-ADSD with Timestamps.* Através das informações de diagnóstico geradas pelo algoritmo a ferramenta disponibiliza o servidor Web em uma máquina da rede, caso exista alguma máquina sem-falha que esteja sendo monitorada nesta rede. Uma interface Web foi implementada para visualizar as informações de diagnóstico da rede monitorada pelo algoritmo *Hi-ADSD with Timestamps*. Experimentos com injeção de falhas foram realizados em um intervalo de observação de cerca de 12 horas, em máquinas monitoradas pela ferramenta *Hi-ADSD with Timestamps* e que executavam a aplicação apresentada neste trabalho. Como resultado foi constatado que em configurações de rede com máquinas onde ocorreram 210 falhas distribuídas entre todas as máquinas, a disponibilidade do servidor Web foi de 97,3%. Em configurações de rede não tão instáveis, onde algumas máquinas juntas falharam 27 vezes, a disponibilidade do servidor Web foi de 99,5%.

A ferramenta SAPOTI possui uma limitação referente ao serviço tolerante a falhas oferecido pela aplicação TCP/IP. Transações que exijam algum tipo de alteração no

servidor através de modificação de conteúdo armazenado não podem ser garantidas. Em trabalhos futuros esta limitação da ferramenta SAPOTI deverá ser estudada e solucionada. Além disto será estudada a criação de um framework baseado em diagnóstico para a construção e de aplicações distribuídas genéricas com alta disponibilidade.

Referências Bibliográficas

[1] D. E. Comer, *Internetworking with TCP/IP – Principles, Protocolos, and Architectures*, Editora Prentice Hall, 4a ed., Vol. 1, 1995.

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, e T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1", *RFC 2616*, Jun 1999.

[3] P. Jalote, *Fault Tolerance in Distributed Systems.* Editora Prentice Hall, 1994.

[4] S. L. Hakimi, e K. Nakajima, "On Adaptive System Diagnosis", *IEEE Transactions on Eletronic Computers*, Vol. 33, 1984.

[5] E. P. Duarte Jr., e T. Nanya, "A Hierarquical Adaptive Distributed System-Level Diagnosis Algotithm", *IEEE Transactions on Eletronic Computers*, Vol. 47, No.1, Jan 1998.

[6] E. P. Duarte Jr., A. Brawerman, e L. C. P. Albini, "An Algorithm for Distributed Hierarquical Diagnosis of Dynamic Fault and Repair Events", *Proceedings of the IEEE International Conference on Parallel and Distributed Systems 2000*.

[7] M. T. Rose, *The Simple Book – An Introduction to Internet Management*, Editora Prentice Hall, 2a ed., 1994.

[8] A. S. Tanenbaum, *Sistemas Operacionais Modernos*, Editora LTC, 1992.

[9] F. Preparata, G. Metze, e R. T. Chien, "On The Connection Assignment Problem of Diagnosable Systems", *IEEE Transactions on Eletronic Computers*, Vol. 16, 1968.

[10] R. P. Bianchini e Buskens, "An Adaptive Distributed System-Level Diagnosis Algorithm and Its Implementation", *Proc. FTCS-21*, 1991.

[11] E. P. Duarte Jr., A. Brawerman, e L. C. P. Albini, “A Diagnosis Algorithm based on Clusters with Detours”, disponível em http://www.inf.ufpr.br/~elias.

[12] S. Rangarajan, A. T. Dahbura, e E. A. Ziegler, “A Distributed System-Level Diagnosis for Arbitrary Network Topologies”, *IEEE Transactions on Eletronic Computers*, Vol. 44, 1955.

[13] M. Rose, e K. McCloghrie, “Structure of Management Information for TCP/IP-based internets”, *RFC 1155*, Performance Systems International, Hughes LAN Systems, Mai 1990.

[14] J. Case, M. Fedor, M. Schoffstall, e J. Davin, “Simple Network Management Protocol (SNMP)”, *RFC 1157*, SNMP Research, Performance Systems Internacional, MIT Laboratory for Computer Science, Mai 1990.

[15] K. McCloghrie, e M. Rose, Editors, “Management Information Base for Network Management of TCP/IP-based internets: MIB-II”, *RFC 1213*, Hughes LAN Systems, Performance Systems Internacional, 1991.

[16] E. S. Silva, e P. R. S. Rezende, “SNEMP: Uma ferramenta Java/Web para Monitoração de Objetos SNMP Replicados em Grupos de Agentes”, 2001, disponível em http://www.inf.ufpr.br/~elias.

[17] A. S. Tanenbaum, *Redes de Computadores*, Editora Campus, 1997.

[18] The NET-SNMP Project Home Page – http://www.net-snmp.org.

[19] The Linux Home Page at Linux on Line – http://www.linux.org.

[20] The Apache Software Foundation – http://www.apache.org.

[21] RSYNC – http://www.rsync.org.

[22] Bash – http://www.gnu.org/software/bash/bash.html.

[23] R. Elmasri, e S. B. Navathe, "*Fundamentals of DataBase Systems*", Editora Addison Wesley, 3a ed., 2000.

[24] PHP Hypertext Preprocessor – http://www.php.net.

[25] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, e B. Lyon, "Design and Implementation of the Sun Network File System", *Proc. Summer Techn. Conf. USENIX*, 1985.

[26] S. Shepler, "NFS Version 4 Design Considerations.", *RFC 2624*, Jun 1999.

[27] Debian GNU/Linux - The Universal Operating System – http://www.debian.org.

[28] L. C. E. Bona, "Gerência Confiável de Redes Locais baseada em Diagnóstico Distribuído", disponível em http://www.inf.ufpr.br/~elias.