UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ

CAMPUS LUIZ MENEGHEL

CÁSSIO FERNANDO GUERREIRO

SERVIDORES *WEB* DE ALTA DISPONIBILIDADE COM TOLERÂNCIA À FALHAS

Bandeirantes

2010

CÁSSIO FERNANDO GUERREIRO

SERVIDORES *WEB* DE ALTA DISPONIBILIDADE COM TOLERÂNCIA À FALHAS

Monografia apresentada à Universidade Estadual do Norte do Paraná – *Campus* Luiz Meneghel, como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Luiz Fernando Legore do Nascimento

Bandeirantes

2010

CÁSSIO FERNANDO GUERREIRO

SERVIDORES *WEB* DE ALTA DISPONIBILIDADE COM TOLERÂNCIA À FALHAS

Monografia apresentada à Universidade Estadual do Norte do Paraná – *Campus* Luiz Meneghel, como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

COMISSÃO EXAMINADORA

____
Prof. Luiz Fernando Legore do Nascimento
UENP – *Campus* Luiz Meneghel

____
Prof. Msc. Éderson Marcos Sgarbi
UENP – *Campus* Luiz Meneghel

____
Prof. Msc. Ricardo Gonçalves Coelho
UENP – *Campus* Luiz Meneghel

Bandeirantes, 10 de Dezembro de 2010

AGRADECIMENTOS

Agradeço, sobretudo à Deus, por ter me dado saúde, proteção e forças para elaborar o presente trabalho.

Agradeço também a minha família que sempre me auxiliou em todos os momentos da minha vida.

Agradeço à minha namorada, que soube entender as minhas ausências durante a elaboração deste trabalho.

Meus sinceros agradecimentos ao meu orientador, Prof. Luiz Fernando L. do Nascimento pela atenção, apoio, incentivo e direção oferecida durante a elaboração deste trabalho, obrigado pela orientação.

Aos membros da banca, Prof. Msc. Ricardo e Prof. Msc. Éderson, por terem aceitado fazer parte como membros da banca.

À todos, um sincero obrigado.

“O tempo amadurece todas as coisas.
Nenhum homem nasce sábio”
Cervantes

RESUMO

A necessidade de sistemas de computadores para ambientes cada vez mais complexos e heterogêneos e que ainda sejam disponíveis 24 horas por dia fez surgir também à necessidade de fazer com que esses sistemas de computadores também pudessem ser auto-diagnosticáveis. Num ambiente distribuído, a complexidade de identificar uma falha em uma unidade do sistema é maior. Nesse contexto, com o objetivo de encontrar procedimentos eficientes para detectar falhas que comprometam o funcionamento destes sistemas surgiu também o desafio de implementar sistemas confiáveis e tolerantes a falha. Dessa forma, quando uma falha é identificada em uma unidade do sistema, as unidades que estão sem falhas devem assumir as responsabilidades que a unidade em falha realizava, assim, os prejuízos acarretados seriam mitigados. Para que esses requisitos sejam atendidos, os sistemas precisam ser tolerantes a falhas e transparentes aos usuários.

Palavras-Chave: Sistemas Tolerantes à Falhas, Ambiente Distribuído, Auto-Diagnosticáveis, Falha.

ABSTRACT

The need for computer systems for environments increasingly complex and heterogeneous and that are still available 24 hours a day also gave rise to the need to make these computer systems could also be self-diagnosable. In a distributed environment, the complexity of identifying a flaw in a system drive is bigger. In this context, in order to find efficient procedures to detect flaws that compromise the functioning of these systems also arose the challenge of implementing reliable systems and fault tolerant. Thus, when a fault is detected in a system unit, the units are flawless shall assume the responsibilities that the failed unit performed well, the losses entailed would be mitigated. For these requirements are met, the systems need to be fault tolerant and transparent to users.

Keywords: Fault-Tolerant Systems, Distributed Environment, Self-Diagnosable, Failure.

LISTA DE FIGURAS

Figura 1: sistema com 4 servidores, onde 2 servidores estão com falhas.................16
Figura 2: a unidade 5 identificou a unidade 0 como falha...*23*
Figura 3: sistema com 8 unidades executando o algoritmo *Adaptive DSD...............25*
Figura 4: sistema executando o algoritmo *Hi-adsd..27*
Figura 5: árvore de teste da unidade *0 t*estando os *clusters.....................................27*
Figura 6: sistema executando o algoritmo *Hi-ADSD with Detours*............................28
Figura 7: sistema executando o algoritmo *Hi-ADSD with Timestamps*......................30
Figura 8: sistema executando o algoritmo *MM* baseado em comparações...............32
Figura 9: sistema executando o algoritmo de comparações generalizado................33
Figura 10: sistema executando o algoritmo *Broadcast* Confiável.............................34
Figura 11: arquitetura do *NS-2*...39
Figura 12: funcionamento básico de um *script Tcl*...40
Figura 13: cenário da simulação...44
Figura 14: servidor 1 com falha..44
Figura 15: servidor 0, 1 e 2 com falha...45
Figura 16: criação dos *links*...46
Figura 17: criação do agente de transporte...46
Figura 18: criação dos geradores de tráficos...47
Figura 19: trecho inicial do arquivo de *trace* do *NS-2* (editado)................................47
Figura 20: média de pacotes enviados e recebidos..49
Figura 21: média de pacotes descartados..50
Figura 22: taxa de entrega de pacotes..51
Figura 23: latência do algoritmo para diagnóstico de falha.......................................52

LISTA DE SIGLAS

CBQ	Class Based Queueing
CBR	Constant Bit Rate
DARPA	Defense Advanced Research Projects Agency
FTP	File Transport Protocol
HTTP	Hipertext Transmission Protocol
MIB	Management Information Base
NAM	Network Animator
NS-2	Network Simulator-2
OSI	Open System Interconnection
OTCL	Object Tool Command Language
RED	Random Early Detection
SNMP	Simple Network Management Protocol
TCL	Tool Command Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VBR	Variable Bit Rate
VINT	Virtual InterNetwork TestBed

SUMÁRIO

1 INTRODUÇÃO 11

1.1 CONTEXTO E DELIMITAÇÃO DO TRABALHO 11

1.2 OBJETIVOS 12

1.2.1 Objetivo Geral 12

1.2.2 Objetivos Específicos 12

1.3 JUSTIFICATIVA 12

2 FUNDAMENTAÇÃO TEÓRICA 14

2.1 CONSIDERAÇÕES INICIAIS 14

2.2 SISTEMAS DISTRIBUÍDOS 15

2.3 MODELOS SÍNCRONOS E ASSÍNCRONOS 16

2.4 CLASSIFICAÇÃO DE FALHAS EM SISTEMAS DISTRIBUÍDOS 17

2.5 MODELOS DE DIAGNÓSTICOS DE FALHAS EM NÍVEIS DE SISTEMAS 20

2.5.1 Modelo PMC 22

2.5.1.1 *Adaptive-DSD* 24

2.5.1.2 *Hi-ADSD* 26

2.5.1.3 *Hi-ADSD with Detours* 28

2.5.1.4 *Hi-ADSD with Timestamps* 29

2.5.2 Modelo de Diagnóstico Baseado em Comparações 30

2.5.2.1 Modelo de Diagnóstico *MM* baseado em Comparações 31

2.5.2.2 Modelo de Comparações Generalizado 32

2.5.2.3 Modelo *Broadcast* Confiável 33

2.6 TRABALHO RELACIONADO 34

2.6.1 Ferramenta Sapoti 34

3 DESENVOLVIMENTO 36

3.1 FERRAMENTAS UTILIZADAS 36

3.1.1 Simulação 36

3.1.2 NS-2 37

3.1.3 Nam e Gnuplot 40

3.1.4 Tcl/Tk 41

3.2 METODOLOGIA DE DIAGNÓSTICO DO ALGORITMO 42

3.3 SIMULAÇÃO DO CENÁRIO E ANÁLISE DOS TRACE 45

3.4 RESULTADOS E MÉTRICAS 48

4 CONSIDERAÇÕES FINAIS 53

4.1 CONCLUSÕES 53

4.2 TRABALHOS FUTUROS 54

REFERÊNCIAS 55

APÊNDICE A – INSTALAÇÃO DO *NETWORK SIMULATOR-2* 58

APÊNDICE B – LINGUAGEM *TCL* REFERENTE AO PROPOSTO 60

APÊNDICE C – ALGORITMO DE DIAGNÓSTICO EM PSEUDOCÓDIGO 73

APÊNDICE D – *SCRIPT* DE SIMULAÇÃO 75

1 INTRODUÇÃO

1.1 CONTEXTO E DELIMITAÇÃO DO TRABALHO

Com a evolução da *Internet* nos últimos anos de forma exponencial deu início também ao surgimento da necessidade de se ter sistemas cada vez mais heterogêneos capaz de compartilhar recurso em uma mesma rede. A necessidade de sistemas de computadores cada vez mais complexos e disponíveis 24 horas por dia fez surgir à necessidade de fazer com que esses sistemas de computadores também pudessem fazer diagnóstico automatizado.

Esses sistemas foram crescendo em função do tempo e houve uma segmentação desses sistemas em dois tipos: os sistemas paralelos e os sistemas distribuídos. Os sistemas paralelos são sistemas que compartilham os recursos disponíveis para realizarem a comunicação. Um sistema distribuído é uma coleção de computadores independentes que aparecem para os usuários do sistema como um único computador (TANENBAUM, 2002). Ou seja, os sistemas distribuídos são criados para permitir que cada unidade do sistema trabalhe e realize um determinado serviço comum em conjunto, mas de forma independente. Dessa forma, quando uma das unidades é identificada como falha, imediatamente outra unidade sem falha deve assumir as responsabilidades que a unidade em falha realizava.

Atualmente o termo diagnóstico automático de falhas tem sido um campo de pesquisa com bastante enfoque. Nesse contexto, onde os sistemas que apresentam falhas comportam-se diferentemente do esperado, o objetivo de encontrar procedimentos eficientes e automatizados para detectar falhas que venham a comprometer a disponibilidade do sistema, antevendo o “erro” e minimizando o impacto, tornam esse a justificativa para um sistema de alta disponibilidade e transparente ao usuário.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Objetiva-se neste trabalho fazer o levantamento bibliográfico dos diferentes tipos de falhas, as metodologias utilizadas para diagnosticar falhas e a partir disso implementar um algoritmo que possa manter e garantir o funcionamento de servidores *web* de alta disponibilidade. Dessa forma, será apresentado um cenário com cinco servidores *web* os quais devem garantir disponibilidade do serviço. O algoritmo proposto deve diagnosticar falhas e manter o serviço resiliente, ou seja, quando um servidor ficar indisponível seja por uma paralisação para manutenção planejada ou por paralisação não planejada devido a uma falha, outro servidor o substitui para fornecer os serviços ao usuário final, isso deve ser feito de forma automatizada.

1.2.2 Objetivos Específicos

- Conhecer os tipos de falhas enfrentados pelos sistemas distribuídos.
- Abordar os algoritmos de diagnóstico do modelo *PMC* e Comparativo para implementação do algoritmo.
- Estudo de trabalho relacionado, como exemplo a ferramenta *Sapoti*.
- Estudar o simulador de rede *Network Simulator-2*.
- Elaborar um algoritmo de diagnóstico de falhas.
- Relatar o desempenho do algoritmo.

1.3 JUSTIFICATIVA

A *Internet* desde o seu surgimento na década de 70 vem evoluindo exponencialmente e tornando-se cada vez mais um grande meio de propagação de informações e serviços. Sua evolução tem trazido grandes benefícios como também problemas para as pessoas de forma geral, em especial para as empresas, que passaram a dispor de um meio de comunicação com seus clientes para oferecer informações e serviços. A necessidade de oferecer serviços de melhor qualidade e

que sejam disponíveis a todo o momento é ainda um grande desafio para as organizações que trabalham nesse ramo.

Com a justificativa de manter serviços de alta disponibilidade, em instituições públicas e privadas, esse trabalho é motivado pela elaboração de um algoritmo que possa analisar e diagnosticar a falha, mitigando os prejuízos para as organizações.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 CONSIDERAÇÕES INICIAIS

O grande desafio para o projetista de uma rede de computadores é reconhecer que qualquer tipo de falha é possível de ocorrer, mesmo que sua probabilidade seja realmente muito pequena. A questão aí não é “se”, mas “quando” a falha ocorrerá. Porém, para alguns casos, os investimentos necessários para se prevenir uma falha são mais altos que os prejuízos acarretados pela sua ocorrência (VICENT, 2001).

Atualmente, várias empresas que trabalham no ramo de oferecer serviços através da *Internet* podem ter altos prejuízos quando acontecem falhas nas máquinas servidoras que disponibilizam seus serviços. Elas utilizam como meio de disseminação a *Internet* com a finalidade de aumentar a competitividade no mercado, trabalhar com estoques reduzidos e consequentemente diminuição de custos extras até então desnecessários, entre outros. Dessa forma, para as empresas disponibilizarem seus serviços, elas utilizam-se de altas tecnologias e que nem sempre são confiáveis para prover o serviço demandado. Uma falha que aconteça nesse sistema e deixa um servidor “fora do ar” pode acarretar grandes prejuízos para as organizações que atuam nesse ramo.

Dentre essas organizações pode-se citar como exemplo as empresas que trabalham no ramo de *e-comerce* (comércio eletrônico). O comercio eletrônico é definido como a compra e venda de produtos, serviços e informações através de redes de computadores (KALAKOTA e ROBINSON, 2002). Também são consideradas como comércio eletrônico as atividades de suporte para qualquer tipo de transação de negócios que ocorram através de infra-estrutura digital (BLOCH, PIGNEUR e SEGEV, 1996). Portanto, essas organizações precisam garantir a disponibilidade e confiabilidade dos serviços a todo momento, e caso alguma falha nos servidores dessas organizações seja iminente, os prejuízos que acarretarão as organizações serão enormes.

Organizações que, em geral, trabalham com sistemas de tempo real também podem sofrer consequências graves, pois esses sistemas, na maioria das vezes trabalham com um nível de segurança maior e exige um monitoramente constante,

neste caso, encaixa-se os sistemas de controle de tráfego aéreo, sistemas de transportes urbanos, sistemas de monitoração de viagens espaciais, pois caso alguma falha venha ocorrer, os danos serão catastróficos para a sociedade. Sistemas críticos envolvendo risco de vida, como sistemas hospitalares, sistemas de controle envolvendo perdas financeiras, como transações bancárias e mercado financeiro, sistemas envolvendo desastres ambientais, sistemas de usinas nucleares e sistema de telecomunicação também entram nessa classificação.

Portanto, garantir que o sistema atenda aos seus requisitos é uma tarefa complexa, principalmente se o sistema atua em ambientes vulneráveis, onde situações adversas como falhas são comuns. Assim esses tipos de sistemas que exigem alta disponibilidade e que sejam tolerantes à falhas em geral devem ter alguma estratégia definida que contorne esse desafio, fazendo com que as consequências dessas falhas tenham o mínimo possível de efeito para as organizações. Para isso, esses sistemas devem ter como prioridade principal a disponibilidade 24 horas por dia para terem a caracterização de alta disponibilidade e não fazer as organizações perderem a concorrência do mercado, e assim não deixando também os usuários ficarem frustados com esses tipos de problemas.

2.2 SISTEMAS DISTRIBUÍDOS

Um sistema distribuído é uma coleção de computadores independentes, apresentando-se como um sistema único, simples e coerente para os usuários, ou seja, um sistema distribuído é aquele em que as máquinas cooperam entre si na realização de tarefas com um objetivo em comum entre si, de tal forma que os recursos oferecidos pelo sistema estejam distribuídos fisicamente mas aparentando como um sistema único, caracterizando-se assim um sistema transparente ao usuário (TANENBAUM, 2002).

Assim, um sistema distribuído permite ao usuário compartilhar, acessar recursos e informações. Impressoras, computadores, unidades de armazenamento, dados, páginas *web*, e entre outros são exemplos de recursos. Uma das razões para compartilhar recursos é a economia, pois ao compartilhar uma impressora, por exemplo, é mais barato compartilhá-la na rede do que comprar uma impressora para cada usuário. Portanto, o compartilhamento de recursos reduz custos

desnecessários.

Um dos objetivos de se projetar um sistema distribuído é construí-lo de tal forma que se possa automaticamente recuperar de falhas sem que ela afete de forma considerável a performance do sistema, assim, quando uma falha acontece o sistema deve tolerar a falha e continuar a operar mesmo com a sua presença (FERNANDES, 2008). Assim um sistema distribuído tem a capacidade de se adaptar conforme surgem falha em alguma unidade do sistema e fazer estas falhas serem imperceptíveis ao usuário.

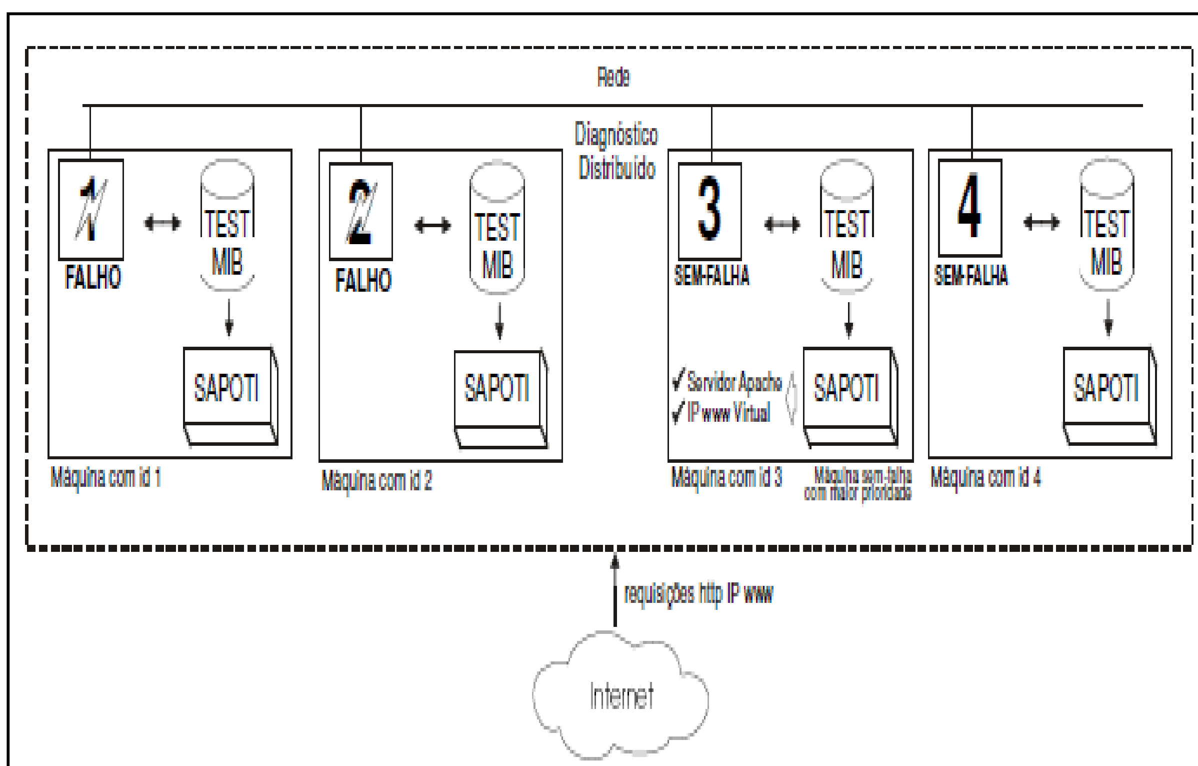

Figura 1: sistema com 4 servidores, onde 2 servidores estão com Falhas (HILGENSTIELER *et al*, 2003)

2.3 MODELOS SÍNCRONOS E ASSÍNCRONOS

Os modelos de sistemas distribuídos existentes podem ser classificados quanto a inúmeras características. O tempo é uma das principais, pois constitui uma característica estrutural do sistema. Por exemplo, um computador *X* emite uma mensagem a um computador *Y*, isto é, há uma troca de mensagens entre computadores, o tempo que esse processo levou é o que constitui essa

classificação. Quando há conhecimento desse tempo, é possível obter estimativas associadas à entrega das mensagens entre os computadores. Esses sistemas são conhecidos como sistemas síncronos (LAMPORT, 1982), isto é, o computador *X* emite uma mensagem e aguarda um certo tempo pela resposta. Quando não há o conhecimento, ou não se possa estimar o tempo da troca de mensagens entre os computadores, a classificação passa a ser de sistemas assíncronos (COULOURIS, 2004). Portanto, nos modelos síncronos, as unidades do sistema progridem e trabalham em passos simultâneos e seqüenciais, esses sistemas tem uma complexidade de implementação menor. Já nos modelos assíncronos, as unidades progridem em passos independentes, em ordem diferenciada e com velocidades diferentes, assim esses sistemas tendem a ser mais difíceis de ser implementados sendo as ferramentas de *e-mail* e os fóruns exemplos desse tipo de sistema.

2.4 CLASSIFICAÇÃO DE FALHAS EM SISTEMAS DISTRIBUÍDOS

Para tratar as falhas em um sistema distribuído é fundamental conhecer a diferença de falha, erro e defeito. Um sistema pode falhar ou porque não executou uma tarefa específica, ou porque a tarefa não executou corretamente a sua função. Essa situação é denominada falha. E falha quando não diagnosticada, pode levar a um erro e ocasionar um defeito, sendo considerado que falhas podem ser toleradas, defeitos não (AVIZIENIS, 1982). Portanto, a interrupção no fornecimento do serviço pode ser evitado pelo uso adequado de técnicas de diagnóstico.

As falhas são classificadas em três classes (COULOURIS, 2004):

Falhas de Duração

- Transitórias;
- Intermitentes;
- Permanentes;

Falhas de Causa

- Operacionais (Físicas);
    - Permanentes;

  - Temporárias;
- Designer (Humanas);
  - Intencionais;
  - Acidentais;

Falhas de Comportamento

- Falhas Crash (*Crash Fault*);
- Falhas de Omissão (*Omission Fault*);
- Falhas de Tempo *(Timing Fault*)*;*
- Falhas Bizantinas *(Byzantine Fault*);

As Falhas do tipo Duração acontecem quando o problema detectado é causado por lentidão na operação, ou seja, o tempo em que o problema permaneceu intervindo no funcionamento padrão do sistema. Elas subdividem em: Transitórias, Intermitentes e Permanentes. As Falhas Transitórias são falhas momentâneas que afetam o resultado de um teste e o seu surgimento é rápido, isto é, num intervalo curto de tempo ou ocorre em um único momento, podendo desaparecer sem ser tratada (BRANCO,1999). Segundo Branco, "as Falhas Intermitentes são aquelas que ocorrem raramente, ou se repetem a intervalos indefinidos. Falhas deste tipo são difíceis de diagnosticar", ou seja, as Falhas Intermitentes acontecem repetidas vezes no sistema entre uma falha e outra, pode-se citar como exemplo um conector *RJ-45* mal colocado. As Falhas Permanentes aparecem ativas por um período de tempo significativo, são mais simples de serem tratadas por possuírem um padrão na forma em que surgem e no seu tempo de duração.

As Falhas de Causa acontecem quando se conhece a origem da falha, ou seja, quando se conhece a causa raiz da falha. Elas são classificadas em Operacionais ou Físicas e *Designer* ou Humanas. Falhas Operacionais ou Físicas são aquelas que ocorrem durante a vida do sistema, afetando a estrutura mecânica ou eletrônica na qual o sistema esteja funcionando. Esse tipo de falha pode ser subdividido em Permanentes e Temporárias. A Falha Permanente surge em função de uma falha física e uma vez detectada no sistema, sempre irá repetir-se. Já a Falha Temporária acontece quando o principal motivo da falha é um desgaste da

unidade física. As Falhas Humanas ou Designer originam-se a partir de falhas humanas. São subdivididas em Intencionais, quando existem ataques premeditados com a intenção de prejudicar o sistema, e Acidentais, quando originam-se através do manuseio humano sem a intenção de prejudicar o sistema.

As falhas de Comportamento, as quais terão enfoque maior para o desenvolvimento desse trabalho, ocorrem quando há uma falha no comportamento de uma ou mais unidade do sistema, estando categorizadas e apresentadas de forma hierárquica abaixo, começando das falhas mais simples para as falhas mais complexas, respectivamente.

Falhas Crash (*Crash Fault*): a unidade do sistema encerra totalmente suas operações ou nunca revela seu estado válido, mas estava funcionando bem até parar, assim as demais unidades do sistema não conseguem detectar seu estado, isto é, quando um problema interno acontece dentro da unidade, como, por exemplo, a queima da fonte de alimentação, fazendo simplesmente a unidade parar de funcionar e parar de interagir com as demais unidades do sistema.

Falhas por Omissão (*Omission Fault*): quando uma unidade deixa de responder algumas requisições, ou seja, a unidade deixa de executar determinado serviço, mas algumas unidades do sistema conseguem detectar seu estado e outras não. Uma perda de mensagem pelo canal de comunicação é considerada uma falha de omissão, pois uma unidade pode fazer um *broadcast* na rede e nem todas as unidades receberem a mensagem.

Falhas por Tempo (*Timing Fault*): as unidades do sistema podem responder dentro ou fora de um limite de tempo predefinido, ou seja, a unidade não termina seu serviço no tempo esperado, alterando assim o desempenho do sistema.

Falhas Bizantinas (*Byzantine Fault*): quando a unidade tem um comportamento contrário quando entra no estado de falha, fazendo, por exemplo, com que as respostas das requisições possam ou não acontecer e caso aconteça, pode responder de forma incorreta, tornando este tipo de falha mais difícil de ser diagnosticada. Ou seja, é quando a unidade continua em operação enquanto está falha, realizando processamentos incorretos e enganosos, podendo dar a impressão de que está funcionando normalmente, quando na verdade não está (BRANCO, 1999). Um exemplo desse tipo de falha pode ser um vírus agindo em segundo plano dentro da unidade.

2.5 MODELOS DE DIAGNÓSTICOS DE FALHAS EM NÍVEIS DE SISTEMAS

Durante muitos anos, vários pesquisadores têm esforçado no intuito de propor modelos, estratégias para detecção e identificação de unidades falhas em sistemas distribuídos. O termo falha se refere a uma diminuição total ou parcial de desempenho de uma unidade, equipamento, processo ou sistema para atender à uma funcionalidade durante um determinado período de tempo. Portanto, num ambiente heterogêneo e distribuído, a dificuldade de um diagnóstico ser eficaz é alta.

O termo diagnóstico automático de falhas tem sido um campo de pesquisa bastante ativo nos últimos anos. Dessa forma, onde os sistemas que apresentam falhas comportam-se diferentemente do esperado, a necessidade de procedimentos para identificação de falhas fez surgir o desafio de implementar sistemas altamente confiáveis e tolerantes à falha. Isso pode ser alcançado por intermédio do isolamento da unidade falha e reconfiguração do sistema, de forma que as tarefas exercidas pela unidade falha passem a ser executadas por outra unidade do sistema. Portanto, a finalidade do diagnóstico de falha automatizado é a detecção rápida da falha onde as unidades do sistema sem falha detectem unidades que estejam falhas. Assim, através de um monitoramento contínuo do sistema, identificam-se os estados de funcionamento destas unidades e quando a unidade apresenta falha, toma-se uma ação corretiva antes que maiores danos ocorram no sistema, mascarando a falha, tornando o sistema tolerante a falha e transparente aos usuários.

Nesse contexto apresentado, com o objetivo de mitigar os efeitos destas falhas e diminuir os prejuízos causados por elas, vários pesquisadores proporam modelos e estratégias para detecção de falhas. Um modelo de falhas é um conjunto de restrições, que definem como as unidades falhas e as unidades sem falhas se comportam, como os testes sobre as unidades serão realizados e seus resultados são obtidos, e ainda como o diagnóstico das unidades é decidido, com o objetivo de identificar de forma precisa os estados de funcionamento destas unidades (BRANCO, 1999).

Na literatura de diagnóstico de falhas, há vários trabalhos propondo

algoritmos para diagnosticar falhas. O primeiro modelo para diagnosticar falhas foi introduzido no trabalho de Preparata, Metze e Chien (PREPARATA, METZE e CHIEN, 1967), onde é considerado que as unidades do sistema são capazes de realizar testes uma sobre as outras e seu estado não deve mudar durante o diagnóstico. Este modelo foi denominado Modelo *PMC*, sendo o nome do modelo as iniciais dos nomes dos autores e com base nesse modelo, foram desenvolvidos alguns algoritmos para diagnosticar falhas.

O modelo *PMC* assume a existência de um observador central que, baseado na síndrome do sistema, realiza o diagnóstico do sistema, sendo este modelo aplicável apenas para identificar falhas mais simples neste caso, falhas de comportamento do tipo *crash*. O conjunto de todos os resultados de testes é chamado de síndrome do sistema (ZIWICH, 2004).

Outro modelo para diagnóstico de falha, proposto por Malek (MALEK, 1980) e em seguida aperfeiçoado por Chwa e Hakimi (CHWA e HAKIMI, 1981), foi o modelo baseado em comparações. Neste modelo, os testes são realizados através de comparações dos resultados realizadas pelas unidades do sistema. Assim, uma tarefa é enviada para pelo menos duas unidades do sistema e os resultados são enviados à uma terceira unidade, denominado testadora, para serem comparados e realizar o diagnóstico do sistema. Uma vantagem desse modelo em relação ao Modelo *PMC* é poder identificar os outros tipos de falhas abordados e não apenas falhas do tipo *crash*, como mencionado no modelo *PMC*.

Estes dois modelos representam, graficamente, as unidades através de grafos. Nestes grafos, os vértices representam as unidades autônomas e as arestas representam uma topologia qualquer de interligação entre as unidades (BRANCO, 1999).

Na prática são utilizados três critérios para avaliar o desempenho dos algoritmos: o número de teste executados, a quantidade de informação transferida em cada teste, e a latência de diagnóstico (BRANCO, 1999). Dessa forma, uma etapa de testes é o período de tempo que todas as unidades têm para executar os testes em todas as unidades.

Assim, o número de testes por cada etapa é dada pela soma de todos os testes realizados por cada unidade.

A quantidade de informação transferida é o mais importante, pois é medido o tráfego de informações na rede de comunicação gerado pela execução do algoritmo de diagnóstico e quanto maior o tráfego maior será a banda da rede consumida.

A latência de diagnóstico é definida como sendo o tempo decorrido entre o acontecimento de uma falha ou reparação até o instante em que todas as unidades sem falha tenham obtido diagnóstico correto (BRANCO,1999).

O diagnóstico de falhas à nível de sistema também é feito de diferentes maneiras. Pode ser feito de forma distribuída, adaptativa e hierárquica.

O diagnóstico é distribuído quando permite que todas as unidades sem falhas determinem o estado, falho ou sem falhas, de todas as unidades do sistema (BONA, 2004). Os primeiros algoritmos de diagnóstico em nível de sistema, como por exemplo, o modelo PMC, considerava a existência de um monitor ou uma unidade, denominada observadora central a qual coletava o resultado de todos os testes executados e efetuava o procedimento de diagnóstico. Mas esta abordagem implica altos custos em termos de tempo, troca de informações e principalmente, confiabilidade.

Esses primeiros modelos dependem da disponibilidade do observador central, que em caso de falha, impede a obtenção do diagnóstico do sistema. Assim, em sistemas distribuídos, esta atividade deve ser compartilhada através de todas as unidades, de forma onde não haja um único ponto que, em caso de falha, impeça a porção restante do sistema de conhecer o seu estado (BRANCO,1999).

Quando o diagnóstico é adaptativo, os testes que cada unidade realiza são baseados em rodadas de teste e cada rodada é realizada com base nos resultados da rodada anterior, isto é, a cada etapa de teste é escolhido aleatoriamente uma unidade do sistema para realizar o diagnóstico. Além de distribuídos e adaptativos, os algoritmos de diagnóstico também podem ser hierárquicos. Quando o diagnóstico é hierárquico, as unidades são divididas e agrupadas em *clusters* e a cada rodada de testes o tamanho destes *clusters* aumenta (BONA, 2006).

2.5.1 Modelo *PMC*

Esse modelo foi um dos primeiros modelos propostos para diagnosticar falhas em nível de sistemas. Os algoritmos que implementam o modelo *PMC* representam, graficamente, um sistema *S* como um grafo completo direcionado,

onde os vértices são as unidades do sistema e as arestas são os canais de comunicação por onde serão feitos os testes. Os algoritmos que se baseiam no modelo *PMC* não levam em consideração as falhas dos canais de comunicação para realizar o teste. A Figura 2 representa um sistema de seis unidades onde a unidade 0 testa a unidade 1, a unidade 1 testa a unidade 2, e assim sucessivamente.

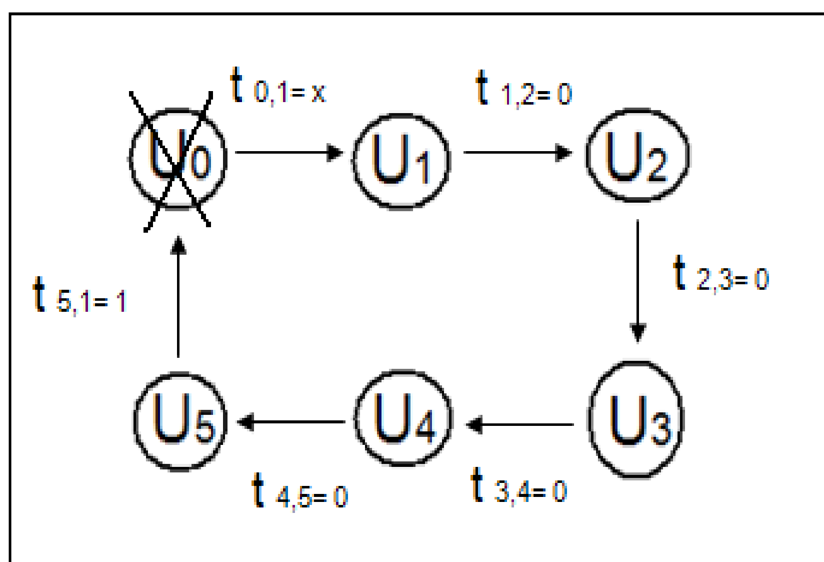

Figura 2: a unidade 5 identificou a unidade 0 como falha

Portanto, o resultado de um teste realizado por uma unidade *x* sobre a unidade *y* pode assumir apenas um estado (sem falha=0 ou falha=1). Neste exemplo, a unidade 5 identificou a unidade 0 como falha. O procedimento de diagnóstico neste modelo é a determinação da situação de falha de um sistema a partir da sua síndrome. Uma síndrome é composta pelo conjunto dos resultados dos testes feitos por todas as unidades sem falha, assim a síndrome desse sistema fica sendo como (x, 0, 0, 0, 0, 1) (ZIWICH, 2004).

No modelo *PMC*, ainda pressupõe-se a existência de um observador central que não faz parte do sistema, responsável por receber todos os resultados dos testes e determinar o diagnóstico dos estados de funcionamento de todas as unidades do sistema.

Quando menciona-se os algoritmos de diagnóstico de falhas, surge o termo *t-diagnosticabilidade,* que é a possibilidade de diagnosticar uma situação de falha com *t* ou menos unidades falhas, com base em uma dada síndrome do sistema (PREPARATA, METZE e CHIEN, 1967). Um sistema *S* é *t-diagnosticável* se todas

as unidades falhas em *S* podem ser identificadas, desde que o número de unidades falhas não exceda *t*, dessa forma surge o termo problema da diagnosticabilidade, ou seja, problema para determinar o número máximo de unidades que podem estar falhas (JALOTE, 1994).

Para que o diagnóstico funcione no modelo *PMC*, é necessário algumas considerações:

- Uma unidade sem falha é confiável e, portanto poderá testar as demais unidades do sistema, assim, as unidades com falha não são confiáveis, portanto podem trazer resultados incorretos;
- A falha não devem mudar durante toda rodada de teste;
- Um resultado *Rx,y* é associado a cada teste que a unidade *x* (número da unidade testadora) executa sobre a unidade *y* (número da unidade testada);
- Neste modelo, cada unidade conhece somente o resultado dos seus testes;

Baseado no modelo *PMC* são abordados quatro algoritmos de diagnóstico para detecção e diagnóstico de falhas:

1. *Adaptive-DSD - (Adaptive Distributed System-Level Diagnosis algorithm);*
2. *Hi-ADSD - (Hierarchical Adaptive Distributed System-Level Diagnosis algorithm);*
3. *Hi-ADSD with Detours;*
4. *Hi-ADSD with Timestamps;*

2.5.1.1 *Adaptive-DSD*

O algoritmo *Adaptive-DSD*, apresentado por Bianchini e Buskens (BIANCHINI e BUSKENS, 1992) é um dos primeiros algoritmos que realiza o diagnóstico de forma adaptativa, portanto não sendo necessário uma unidade central fixa que realize o diagnóstico, isto é, a cada rodada de teste é escolhida uma unidade para realizar o diagnóstico do sistema. Neste algoritmo, são atribuídos

identificadores seqüenciais ordenados às unidades, de 0 a *U*-1. Os testes são executados de forma seqüencial até encontrar uma unidade sem falha, isto é, a unidade *x* testa a unidade *x* +1, se estiver falho, a unidade *x* testa a unidade *x*+2 e assim por diante. Desta forma o grafo de testes do sistema é um anel, como mostrado na Figura 3.

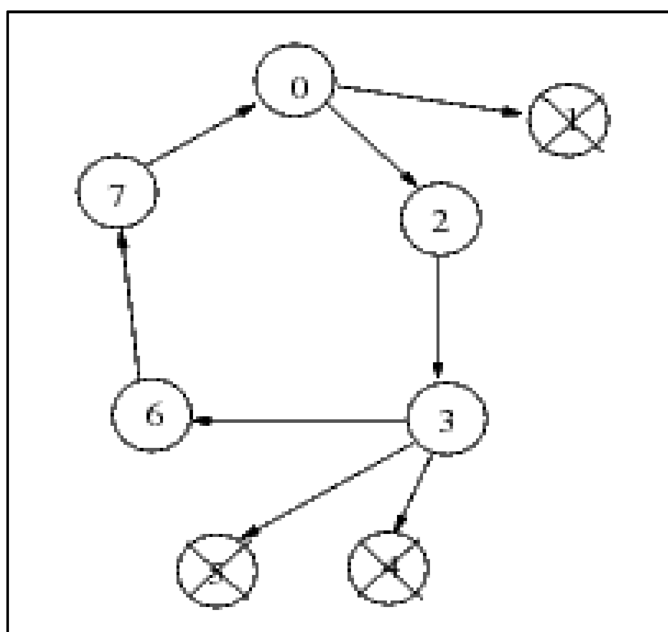

Figura 3: sistema com 8 unidades executando o algoritmo *Adaptive-DSD* (BONA, 2000)

Quando uma unidade sem falha é testada, a unidade testadora obtém informações de diagnóstico da unidade testada. Estas informações incluem os resultados dos testes realizados pelas unidades testadas, além das informações de diagnóstico por ela obtidas. A latência deste algoritmo é definida em termos de rodadas de testes.

O algoritmo *Adaptive-DSD* tem a propriedade de que cada unidade é testada exatamente uma vez a cada rodada de testes, e cada unidade sem falha testa exatamente uma outra unidade sem falha por rodada de testes. Entretanto, se apenas uma unidade estiver sem falha e *U-1* unidades estiverem falhas, a unidade sem falha testará todas as unidades *U-1*, ou seja, o algoritmo *Adaptive-DSD* consegue realizar o diagnóstico do sistema mesmo que exista somente uma unidade sem falha, o que faz com que o algoritmo seja classificado como (*U-1*)-diagnosticável.

Dessa forma, levando em conta o desempenho do algoritmo, o número total de testes realizado pelo algoritmo *Adaptive-DSD*, considerando todos as unidades, é *U* a cada rodada de testes, e a latência do algoritmo é *U* a cada rodada de testes. A quantidade de informações transferidas é *U-1* por rodada.

2.5.1.2 *Hi-ADSD*

O algoritmo *Hi-ADSD* é uma evolução do algoritmo *Adaptive-DSD,* ele foi proposto por Duarte e Nanya. O algoritmo *Hi-ADSD* utiliza uma estratégia de testes baseada na técnica “dividir para conquistar” (DUARTE e NANYA, 1998).

Este algoritmo agrupa as unidades em *clusters* de tamanho progressivo para realização dos testes com o objetivo de reduzir a latência, sendo este um algoritmo distribuído, adaptativo e hierárquico. *Clusters* são agrupamentos de unidades e o tamanho do cluster é calculado por uma potência de base dois, o teste começa pelo menor *cluster* (2^0) e prosseguindo para o segundo menor *cluster* e assim sucessivamente até chegar ao maior *cluster*, começando novamente a rodada de teste pelo menor *cluster* (BONA, 2004).

Quando a unidade é testada, o testador irá obter informações de diagnóstico sobre todas as unidades pertencentes ao *cluster* testado. Caso contrário, o testador ira continuar testando outras unidades deste *cluster,* e se ainda não encontrar, o testador irá continuar em outro *cluster*. A Figura 4 mostra um sistema com 8 unidades organizadas em *clusters* executando o algoritmo *Hi-ADSD*. Ela mostra a hierarquia de testes para oito unidades, do ponto de vista da unidade *0.*

Quando a unidade *0* testa um *cluster* de tamanho 2^2, isto é, com 4 unidades, ela primeiro testa a unidade *4*. Se a unidade *4* está sem falha, a unidade *0* copia a informação de diagnóstico a respeito da unidade *4*, *5*, *6* e *7*. Caso a unidade *4* está com falha, a unidade *0* testa a unidade *5*, e assim por diante.

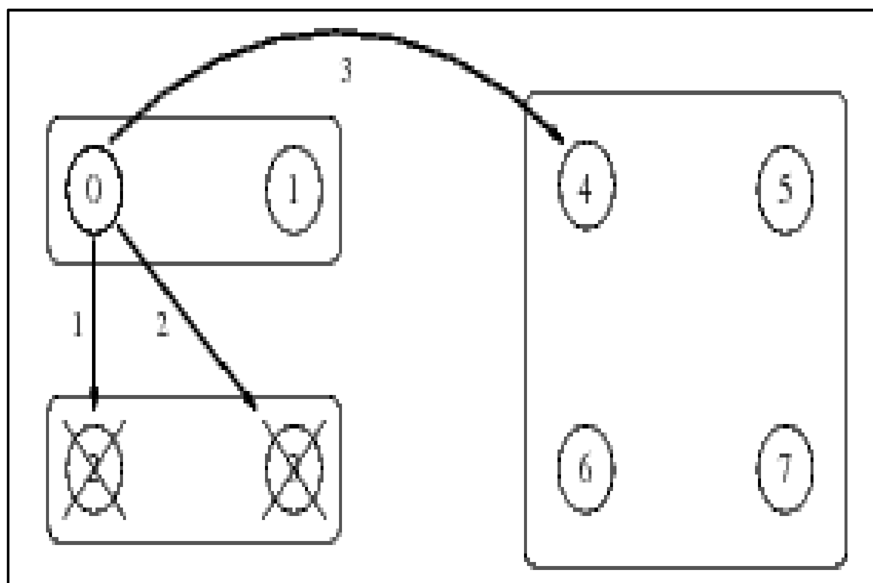

Figura 4: sistema executando o algoritmo *Hi-ADSD* (BONA, 2000)

Nesse algoritmo, a quantidade de informações transferidas é igual ao tamanho do *cluster* que está sendo testado. A latência do algoritmo por rodada de testes é $log^2 U$ e o número máximo de testes executados por rodada é $U^2 / 4$.

O algoritmo *Hi-ADSD* utiliza uma árvore para armazenar informações a respeito dos testes em todos os *clusters*. A Figura 5 mostra a árvore mantida na unidade *0* para um ambiente com 8 unidades onde todas as unidades estão sem falha.

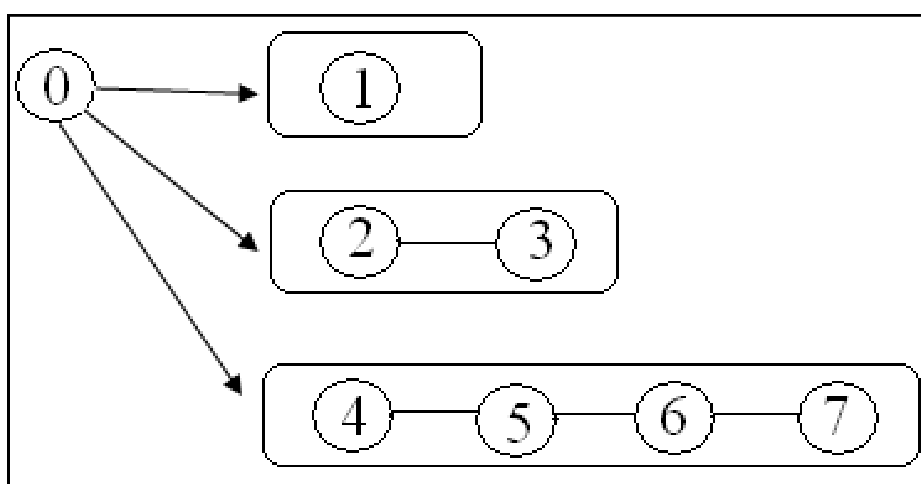

Figura 5: árvore de teste da unidade *0* testando os *clusters* (BRANCO,1999)

2.5.1.3 *Hi-ADSD with Detours*

O algoritmo *Hi-ADSD with Detours* é uma evolução do algoritmo *Hi-ADSD*. As unidades também são organizadas e agrupadas em *clusters*. O algoritmo *Hi-ADSD with Detours* apresenta uma melhora para o pior caso de testes do algoritmo *Hi-ADSD* sem prejudicar a latência, pois o número de teste necessário para completar o diagnóstico diminui consideravelmente.

No algoritmo *Hi-ADSD with Detours*, quando uma unidade testa uma unidade com falha em um determinado *cluster*, o algoritmo não continua o teste nas demais unidades deste *cluster*, neste caso, o *cluster* é dito bloqueado e a unidade que o testou procura obter informações sobre as unidades deste *cluster* quando testar outros *clusters* (BONA, 2006). Estes caminhos alternativos são chamados de *detours* e devem ter a mesma distância de diagnóstico que o caminho original. Distância de diagnóstico é a quantidade de arestas pelas quais as informações devem passar até atingir a unidade de destino, quando todas as unidades do sistema estão sem falhas.

Nesse algoritmo, uma rodada de testes é definida como o período de tempo no qual todas as unidades sem falha testam um *cluster*. O número máximo de testes executados a cada rodada é *U log U.* A quantidade de informações transferidas é definida pelo tamanho do cluster que está sendo testado e a latência do algoritmo é definida como o número de rodadas de testes necessárias para que todas as unidades sem falha do sistema realizem o diagnóstico, nesse algoritmo a latência é a mesma do algoritmo *Hi-ADSD*, isto é, $log^2 U$.

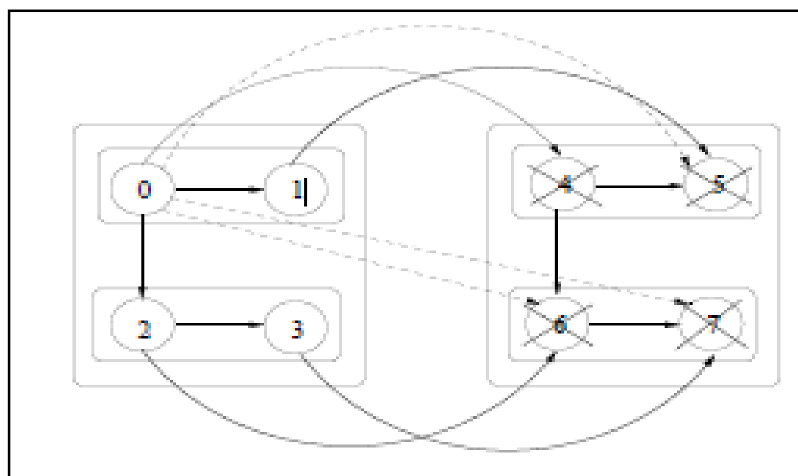

Figura 6: sistema executando o algoritmo *Hi-ADSD with Detours* (BONA, 2000)

2.5.1.4 *Hi-ADSD with Timestamps*

O algoritmo *Hi-ADSD with Timestamps* (DUARTE Jr., ALBINI e BRAWERMAN, 2000) é um aperfeiçoamento do algoritmo *Hi-ADSD with detours* e é também um algoritmo de diagnostico hierárquico, distribuído e adaptativo. No algoritmo *Hi-ADSD with Timestamps,* busca-se o diagnóstico de eventos, ou seja, mudanças de estado que podem acontecer antes do diagnóstico ter sido completado por todas as unidades, isto é, este algoritmo tem a capacidade de diagnosticar eventos dinamicamente, antes que uma rodada de teste termine. Um evento acontece quando uma unidade falha se torna sem falha ou vice-versa durante a realização do diagnóstico (BONA, 2006).

O termo *timestamp* é uma informação guardada sobre o estado de cada unidade do sistema, esta informação é um contador incrementado a cada mudança de estado. Dessa forma, cada testador pode obter informação sobre uma unidade especifica do sistema através de mais de uma unidade testada sem causar inconsistência, isto é, sem confundir um estado mais antigo com outro mais recente, dessa forma, o algoritmo permite datar as informações.

Ao testar uma unidade sem falha em um *cluster* no algoritmo *Hi-ADSD with Timestamps*, não são obtidas somente informações sobre o cluster da unidade testada e possíveis *detours* (BONA, 2006). Sempre que uma unidade *X* testa uma unidade *Y* como sem falha, são lidas informações de diagnóstico de todas as unidades que estão a uma distância de diagnóstico de até $log2\ U - 1$ da unidade *X* passando pela unidade *Y*, um conjunto que tem sempre $U/2$ unidades, ou seja, num sistema com oito unidades, terei 8 cluster com tamanho de $U/2$, isto é, com 4 unidades cada cluster. Como as informações sobre uma determinada unidade podem ser obtidas a partir de vários unidades testados, então emprega-se o *timestamp* para selecionar a informação de diagnóstico mais recente.

Portanto, o número máximo de teste é o mesmo que o do algoritmo *Hi-ADSD with Detours* e a latência é reduzida em relação ao *Hi-ADSD with Detours*, ou seja, este algoritmo apresenta somente uma melhoria para os piores casos de latência.

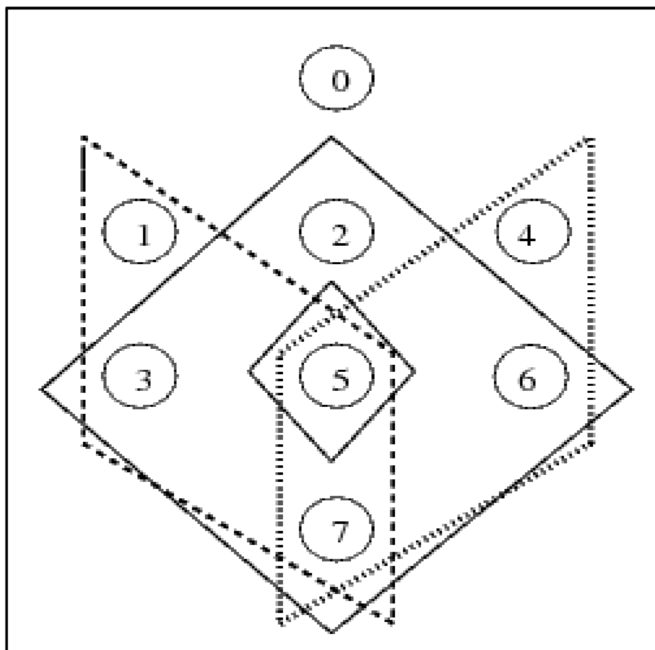

Figura 7: sistema executando o algoritmo *Hi-ADSD with Timestamps* (BONA, 2000)

2.5.2 Modelo de Diagnóstico Baseado em Comparações

Esse modelo foi proposto por Malek (MALEK, 1980). Nele, os testes são realizados através de comparações dos resultados realizadas pelas unidades do sistema. Esse modelo assume que existe um observador central que armazena as informações das saídas das tarefas e que através das comparações das saídas chega-se ao diagnóstico completo do sistema e se existe alguma falha ou não, localiza-se as unidades falhas. Este observador central é uma unidade confiável que não sofre nenhum evento (ZIWICH, 2004).

Nesse modelo uma tarefa ou rotina é enviada para pelo menos duas unidades do sistema, e os resultados da tarefa são enviados à outra unidade, denominado testadora, para serem comparados. Quando a comparação dos resultados dos testes da mesma rotina for igual, pressupõe que as unidades testadas estão sem falha, neste caso a unidade testadora, a qual fez as comparações, assume que as unidades que realizaram o teste estejam sem falha ou mesmo com falha, mas se os resultados forem divergentes, a testadora assume que uma das unidades esteja com falha, mas baseado apenas nestas comparações, a testadora ainda não sabe identificar qual das duas unidades testada está em estado

de falha (ALBINI, 2000). Neste caso, os resultados geram uma divergência e para contornar essa situação, algumas considerações devem ser tomadas. Assim, os resultados dos testes realizados pelas unidades sem falha para a mesma rotina devem ser iguais, um resultado de uma unidade falha tem que ser diferente de qualquer outro resultado produzido pela mesma rotina, tanto por unidades sem falha como para outras unidades falhas.

A desvantagem desse modelo é que o observador central, isto é, a unidade que receberá os resultados das comparações dos testes das rotinas e realizará o diagnóstico do sistema, deve ser uma unidade especial e não pode falhar em hipótese alguma porque nesse modelo apenas o observador central realiza o diagnóstico.

2.5.2.1 Modelo de Diagnóstico *MM* baseado em Comparações

O modelo MM, das iniciais dos autores Maeng e Malek (MAENG e MALEK, 1981), é um aperfeiçoamento do modelo baseado em comparações proposto por Malek (MALEK, 1980) e apresentado na seção 2.6.2. No modelo *MM*, as próprias unidades que realizam a tarefa também fazem as comparações, dessa forma dá a possibilidade de se ter diversas unidades comparadoras, ficando a unidade observadora responsável por realizar o diagnóstico do sistema e a unidade comparadora deve ser confiável, isto é, sem falha. Neste caso a unidade observadora saberá se as unidades testadas estão ou não falhas através da comparação feitas pela comparadora. Mesmo assim a observadora não consegue detectar qual unidade está falha quando acontecer dos resultados do testes serem divergentes e ainda se a unidade comparadora também estiver falha, a observadora nada poderá fazer (ALBINI, 2000).

A representação gráfica desse modelo é um multigrafo, ou seja, cada par de unidades está ligado por várias arestas (GOULD, 1988). Para diferenciar as unidades comparadoras de um mesmo par de unidades, um par de unidades (x,y) é denotado por $z(x,y)$, ou seja, z é a unidade comparadora para as unidades x e y (ALBINI, 2000). Para realizar o diagnóstico, a comparadora z vai comparar as saídas produzidas pelas duas unidades (x e y) quando submetidas à mesma tarefa, caso o resultado seja igual, o resultado da comparação será $r((x,y)\ z)= 0$, caso contrario

será r((*x,y*) *z*)= 1. Se a própria unidade comparadora estiver sem falha, um resultado da comparação r((*x,y*) *z*)= 0 implica que está sem falha, então as unidades *x* e *y* também estão sem falhas, no entanto, se o resultado das comparações for *r*((*x,y*) *z*)= 1 e o comparador *z* está sem falha, então sabe-se que a unidade *x* ou a unidade *y* ou até mesmo os duas unidades estão falhos, porém se o comparador *z* estiver falho, então nada se pode constatar de suas comparações.

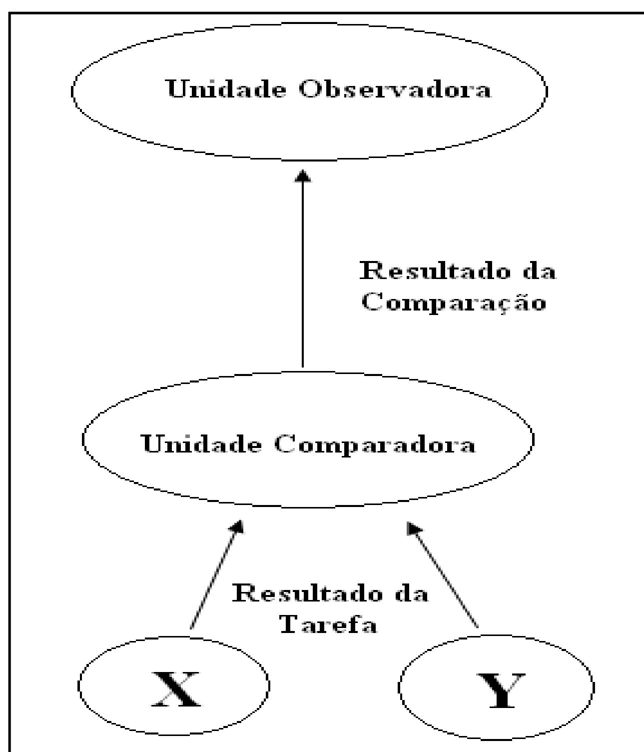

Figura 8: sistema executando o algoritmo *MM* baseado em comparações

2.5.2.2 Modelo de Comparações Generalizado

Neste modelo, os testes também são realizados através de comparações. Mas a comparadora será uma das duas unidades que executaram a mesma tarefa e não outra unidade do sistema como no modelo *MM*. Isto é, uma das unidades que executaram a tarefa de teste realiza também a comparação dos resultados e envia as comparações para a observadora central diagnosticar.

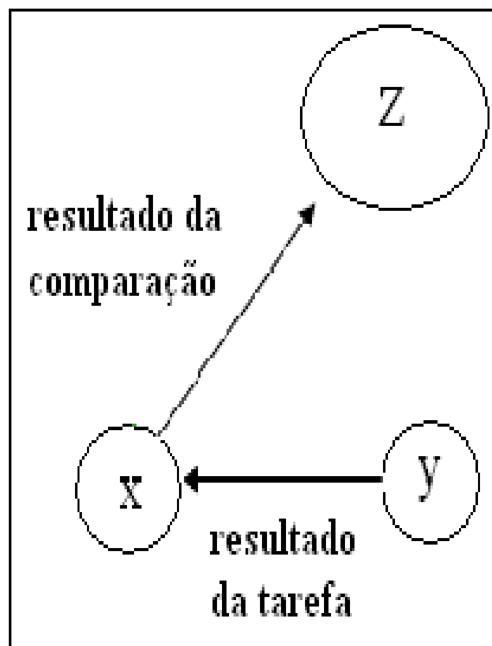

Figura 9: sistema executando o algoritmo de comparações generalizado

2.5.2.3 Modelo *Broadcast* Confiável

Nesse modelo, além do diagnóstico ser baseado em comparações também é distribuído. Ou seja, não existe um observador central. Quando duas unidades são submetidas a realizarem a mesma tarefa, os resultados são enviados a todas as unidades do sistema fazendo um *broadcast* na rede, inclusive as próprias unidades que realizam a tarefa, onde cada unidade receberá os resultados e fará as comparações e a seguir o diagnóstico do sistema (ZIWICH, 2004). Esse modelo consome muito recurso devido à grande quantidade de *broadcast* realizado na rede, assim a quantidade de informações transferidas é maior. Na Figura 10, a unidade *U1* enviou uma tarefa para a unidade *U2* e *U3*, estas por sua vez produziram um resultado e disseminou os resultados para as demais unidades do sistema, inclusiva para elas próprias e para a unidade *U1*. Se os resultados das tarefas são iguais, a comparação resulta 0, se os resultados das tarefas são diferentes, a comparação resulta 1 (ZIWICH, 2004). Além disso, qualquer mensagem de *broadcast* de uma unidade sem falha deve ser recebida corretamente por todas as outras unidades sem falha em um tempo limitado.

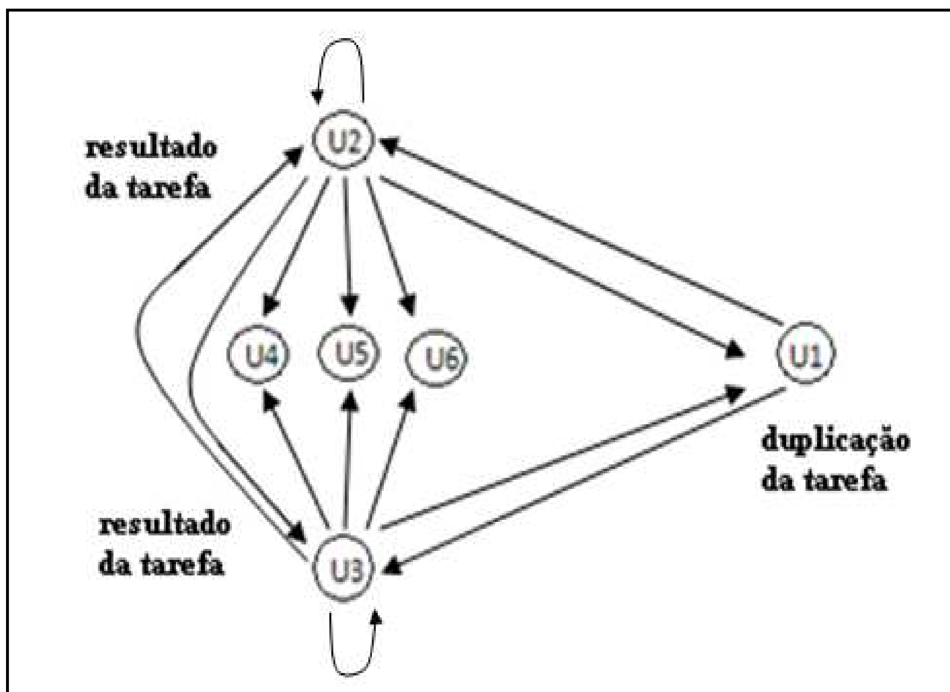

Figura 10: sistema executando o algoritmo *Broadcast* Confiável

2.6 TRABALHO RELACIONADO

2.6.1 Ferramenta *Sapoti*

Este trabalho partiu também do estudo da ferramenta *SAPOTI* (Servidores de APlicações cOnfiáveis *Tcp/Ip*). É uma aplicação em nível de sistema que garante, através da identificação e monitoramento de falhas, a alta disponibilidade de Servidores *Web*.

O *SAPOTI* utiliza uma ferramenta de gerência de rede para obter informação de diagnóstico e implementa o algoritmo de diagnóstico de falhas *Hi-ADSD with Timestamps* que será abordado nesse trabalho. Dessa forma, o *SAPOTI* utiliza o protocolo padrão da *Internet* para gerência de redes, denominado *SNMP* (*Simple Network Management Protocol*). A arquitetura do *SNMP* é composta por entidades denominadas agentes e gerentes, os quais utilizam uma *MIB* (*Management Information Base*). *MIB* é uma coleção de objetos que são mantidos por cada unidade do sistema, a qual é utilizada pelos gerentes do protocolo *SNMP* para obter

diversas informações referentes a uma unidade do sistema, essa unidade pode ser uma impressora, um modem, um *hub*, um computador, entre outros.

Nesse contexto, uma possível falha que ocorra em algum servidor *web*, pode impedir os agentes de realizarem os testes e obter informações. Para contornar essa situação, o *SAPOTI* distribui prioridades entre os servidores, assim, os serviços que eram oferecidos por um servidor devem ser recuperados em outro servidor sem falha de maior prioridade. Cada servidor tem um número *IP* e é definido também um *IP* virtual para o grupo de máquinas as quais tem também instalado o *Apache* que nada mais é do que um servidor de páginas *web*, isto é, um aplicativo rodando um serviço em uma máquina servidor que fornece páginas *web*. Para iniciar, o servidor de menor prioridade recebe o IP virtual e passa a disponibilizar os serviços, se esse servidor apresentar falha, outro servidor do grupo de maior prioridade que este receberá o *IP* virtual e assim sucessivamente. Dessa forma, apenas o servidor que estiver com o *IP* virtual passará a disponibilizar o serviço. Assim, a demanda de requisições poderá ser atendida de forma transparente, melhorando a confiabilidade, desempenho e qualidade dos serviços prestados.

3 DESENVOLVIMENTO

3.1 FERRAMENTAS UTILIZADAS

3.1.1 Simulação

Com o constante desenvolvimento das novas tecnologias, os responsáveis por elas necessitam de métodos para avaliar as propostas de melhorias ou soluções de problemas trazidos por essas tecnologias. Para que isso seja feito, existem três possibilidades de se avaliar e obter resultados (OLIVEIRA, 2005):

1. Experimentações: são as avaliações na prática do objeto a ser analisado. Esta metodologia tende a ser mais precisa, no entanto é mais difícil de ser realizada devido ao fato de ter alto risco de paralisação da rede ou instabilidade de alguns equipamentos quando exposto aos experimentos.

2. Analítica: consiste em modelar o sistema e definir as variáveis de interesse de forma que esse modelo possa ser tratado matematicamente. Através desse tratamento é possível obter diversos resultados de acordo com as situações desejadas através de modificações nestas variáveis. No entanto, essa metodologia normalmente implica em um profundo conhecimento matemático para ser elaborada a fim de obter resultados mais precisos.

3. Simulação: é uma metodologia similar à metodologia experimental, mas ao invés de testar na prática o objeto em estudo, é usado um programa de simulação como modelo. Assim, essa metodologia oferece mais dinamismo e resultados mais rápidos, definição de cenários mais complexos com menos envolvimento matemático e menor custo.

Formalmente simulação pode ser definida como (SHANNON, 1975):

> “...o processo de desenhar um modelo de um sistema real e conduzir os experimentos usando este modelo com o propósito de entender o comportamento do

> sistema ou avaliar várias estratégias (dentro dos limites impostos por um critério ou um conjunto de critérios) para a operação do sistema."

Simulações são necessárias atualmente, pois quando se modela elementos do mundo real requer recursos computacionais altos, como memória, processamento, entre outros, tendo assim custos financeiros altos. Dessa forma, as simulações oferecem condições para experimentações com menor custo, economizando tempo e dinheiro (ARAUJO, 2003).

Quando se faz uso de simulação, pode-se também analisar sistemas com comportamentos dinâmicos em função do tempo através da construção de modelos que ajudam a entender a estrutura do ambiente e analisar certos tipos de comportamentos do ambiente quando aplicado ações aleatórias e pré-definidas nesse ambiente.

3.1.2 *NS-2*

Para a realização das simulações deste trabalho foi usado o simulador *Network Simulator* ou *Network Simulator-2* devido a sua versão. O *NS-2* é um acrônimo de *Network Simulator-2* (Simulador de Rede), teve sua primeira versão lançada durante o desenvolvimento do software *Real Network Simulator* (Simulador Real de Rede) em 1989 com o objetivo de simular o funcionamento das primeiras redes e tornou-se bastante conhecido pelo fato de ser o pioneiro na área de redes.

Em 1995, o *NS-2* começou a fazer parte do projeto *VINT* (*Virtual InterNetwork TestBed*), o qual é um projeto de pesquisa oriundo da agência americana *DARPA* (*Defense Advanced Research Projects Agency, EUA*) e pela agência americana *NSF* (*National Science Foundation, EUA*), onde o objetivo do projeto era desenvolver um simulador de rede que permitisse o estudo de protocolos de redes que surgia na época. A partir de então vem evoluindo para atender a demanda das pesquisas na área de redes, tornando assim um simulador de redes de computadores amplamente utilizado no mundo inteiro na área de redes (*NS*).

Atualmente, o *NS-2* encontra-se na versão 2.34 e pode ser usado principalmente em plataformas *Unix*, como *HP-UX* e *Solaris*, plataforma *Unix-like*, como *Linux* e *FreeBSD* e em plataforma *Microsoft Windows*. É um software disponibilizado gratuitamente e *open source* (código aberto), ou seja, tem código

aberto o qual se pode modificá-lo para adaptar-se a cada necessidade específica. Caracterizado também por ter seu funcionamento baseado e dirigido à eventos discretos, permitindo facilidade quanto a criação de inúmeros cenários muito próximos da realidade. Portanto, o *NS-2* é um simulador abrangente podendo simular tecnologias existentes que abrangem desde a camada de enlace até a camada de aplicação do modelo referenciado e padronizado internacionalmente como modelo *OSI* (*Open System Interconnection Reference Model*). Assim o *NS-2* pode simular diversos tipos de tecnologias de redes, ou seja, pode trabalhar simulando protocolos *TCP* (*Transmission Control Protocol*) e *UDP* (*User Datagram Protocol*), roteamento de pacotes, como *unicast*, *multicast*, alguns protocolos da camada de enlace, rede cabeada e rede sem fio (*wireless*), simular comportamento dos protocolos de aplicação, como por exemplo, *FTP* (*File Transport Protocol*), *HTTP* (*Hipertext Transmission Protocol*), *CBR* (*Constant Bit Rate*), *VBR* (*Variable Bit Rate*), *Telnet*, simular mecanismos de gerenciamento de filas de roteamentos como *DropTail*, *RED* (*Random Early Detection*) e *CBQ* (*Class Based Queueing*), e geração programada e pré-definida de falhas e eventos dos componentes que compõem uma rede, entre outros. Este trabalho descreverá o uso do simulador *NS-2* no sistema operacional *Linux* (distribuição *Ubuntum* 9.10). No apêndice *A* pode-se obter informações sobre a instalação do *NS-2*.

Como já mencionado acima que o *NS-2* é baseado em eventos discretos, define-se evento como sendo qualquer seqüência de ações pré-programada que possa acontecer em função do tempo durante uma simulação. Portanto, o *NS-2* possui um agendador ou escalonador para gerenciar o momento que cada evento ocorrerá. A queda de um enlace de comunicação é um exemplo de evento que podemos simular no *NS-2*.

O *NS-2* usa as linguagens *C/C++* e *Tcl* (*Tool Command Language* - Linguagem de Comandos de Ferramentas) (ARAUJO, 2003). Os *scripts Tcl* são usados para descrever o ambiente a ser simulado, sendo o simulador um interpretador destes *scripts* que usa as bibliotecas desenvolvidas em *C++* as quais possuem os objetos para criação dos elementos da rede e escalonadores de eventos. Portanto, para iniciar a simulação deve-se escrever um *Script Tcl* o qual é interpretado pelo *NS-2* através do interpretador *OTcl* (*Object Tool Command Language*). A Figura 11 descreve a arquitetura do *NS-2*. Nesta figura, o programador escreve *script Tcl* usando a biblioteca *OTcl*. Os escalonadores de evento e a maioria

dos elementos de rede são implementados em *C++* e disponíveis para o interpretador *OTcl* através da utilização da biblioteca *Tclcl*.

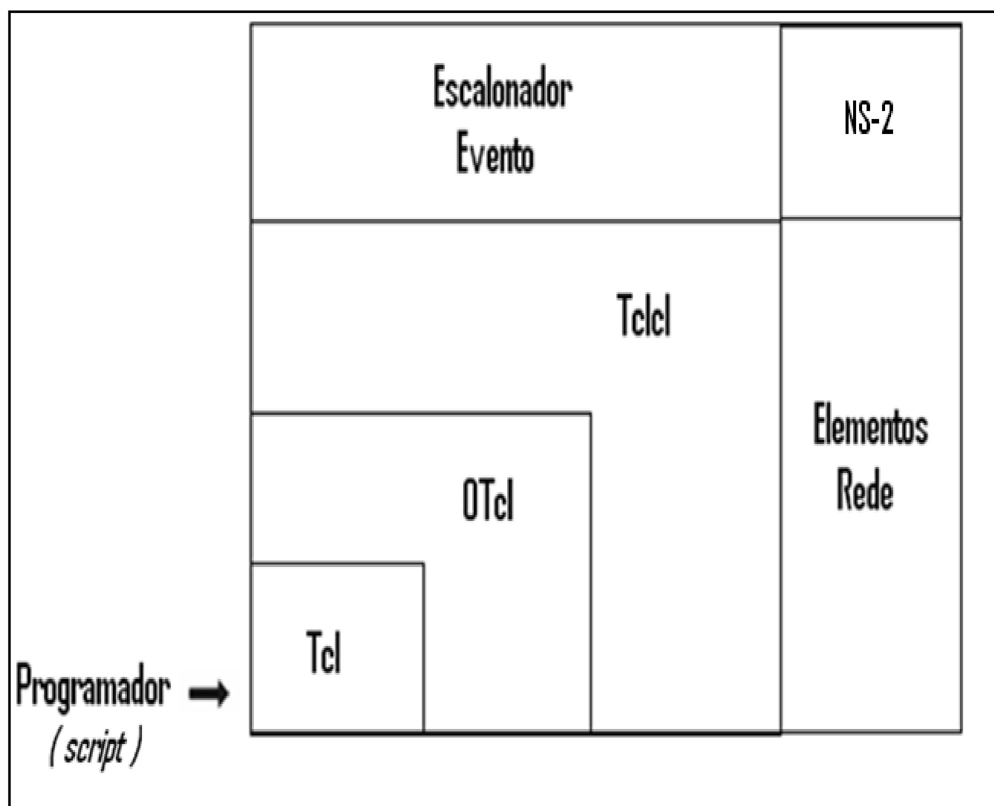

Figura 11: arquitetura do *NS-2* (Araújo, 2003)

Através do *Script Tcl* pode-se fazer atribuições, chamar manipuladores de eventos, esperar por eventos, configurar a topologia de rede, informar as fontes de tráfego, indicando quando esta deve iniciar ou parar de transmitir pacotes em determinado instante de tempo, favorecendo a criação de um cenário bem próximo do mundo real. A Figura 12 mostra como funciona a interpretação de um *Script*. No entanto, para escrever um *Script Tcl*, existem alguns passos básicos que devem ser seguidos:

1. Definição dos parâmetros iniciais da simulação como, por exemplo, tempo da simulação, quantidade de nós, entre outros.

2. Criar o objeto simulador ou o escalonador de eventos.

3. Abrir arquivos de *trace* para posteriores análises.

4. Criação da topologia de rede, onde se definem os nodos, tipos de *links*, tipos de filas e tipos de tráficos.

5. Criação e vinculação dos agentes de transportes.

6. Criação dos geradores de tráficos e a vinculação deles com os agentes de transporte (criado na etapa 5).

7. Definição dos acontecimentos como início e fim da transmissão de pacotes.

8. Finalização e fechamento da simulação, através do uso de ferramentas de animação e geração de gráficos estatísticos.

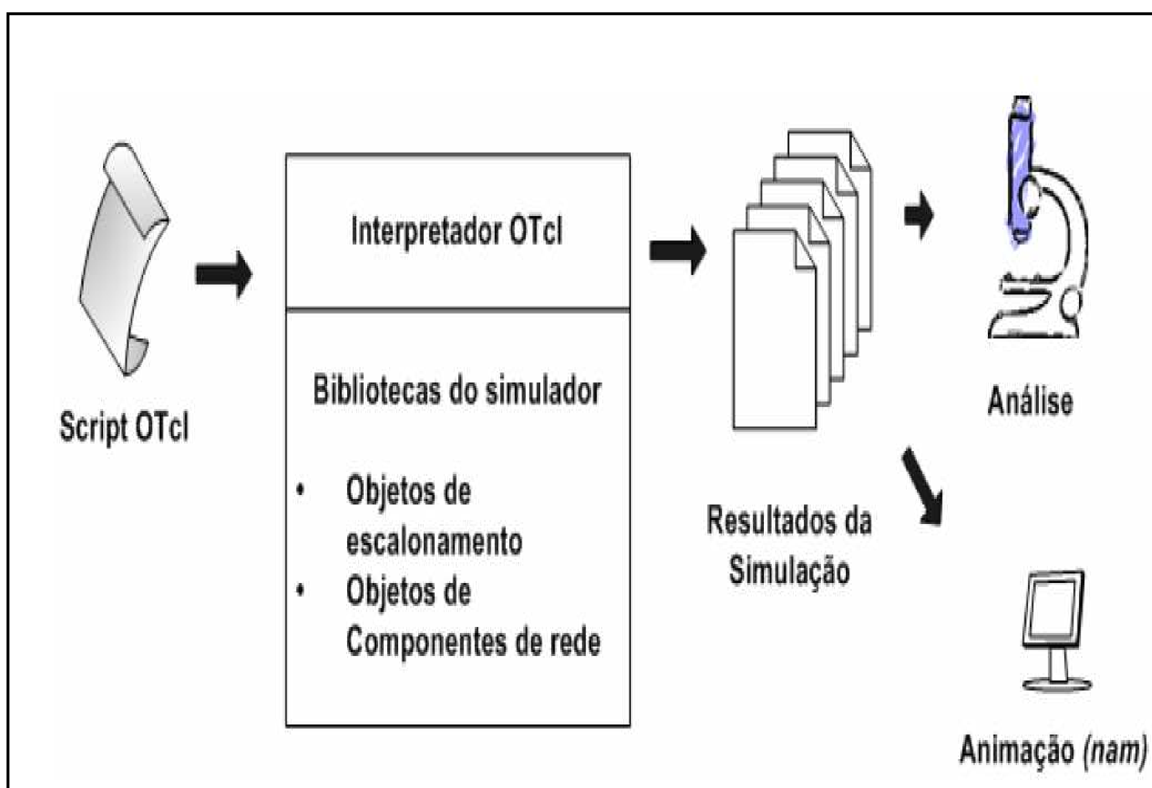

Figura 12: funcionamento básico de um *Script Tcl* (Araújo, 2003)

3.1.3 *Nam e Gnuplot*

No entanto, apenas o *NS-2* não é suficiente para análise dos dados da simulação devido ao resultado da simulação estar em um formato incompreensível para leitura humana. Portanto, durante a simulação, é criado um arquivo com os

dados da simulação, denominado arquivo de *trace*, esse arquivo então pode ser utilizado por outras ferramentas para realizar as análises e visualização animada desses *traces.* Esses arquivos *traces* podem então ser usados como entrada para ferramentas gráficas como o *Nam* (*Network Animator* – Animador de Rede) e *Gnuplot*, e se necessário gerar gráficos, ou seja, os dados podem ser usados como entrada para ferramentas gráficas como o *Nam* (*Network Animator* – Animador de Rede) e *Gnuplot*, ambas utilizadas neste trabalho para analisar os dados. Essas duas ferramentas, criadas na linguagem *TCL/TK* em 1986, são ferramentas *open source* (código fonte aberto) e faz parte do projeto *VINT* onde também está inserido o *NS-2*.

A ferramenta *Gnuplot* é aqui utilizada para gerar os gráficos a partir de um conjunto de dados de saída elaborada pelos *Scripts* personalizados ou algoritmo, com objetivo de análise futura desses resultados colhidos durante o processo de simulação. Embora se possa gerar gráficos a partir de pares ordenados em qualquer outra ferramenta de plotagem gráfica, o *Gnuplot* é de valor, com capacidade de renderização de gráficos com duas e três dimensões, e fornecendo, no seu leque de funcionalidades, formas de proceder ao tratamento estatístico de dados. Trata-se ainda de uma ferramenta de distribuição gratuita disponível para diversos sistemas operacionais, como *Linux* e *Windows*. Dentre as vantagens em sua utilização podemos citar a portabilidade e embora não pareça a princípio, a facilidade no seu manuseio. O mesmo se dá de forma interativa através da linha de comando ou elaborando um *Script*, contendo todas as instruções a serem executadas.

3.1.4 *Tcl/Tk*

Tcl é acrônimo de *Tool Command Language* (Ferramenta de Linguagem de Comandos), é a terceira linguagem de programação estruturada *open source* (código fonte aberto) mais utilizada no mundo, sendo uma linguagem interpretada baseada em comandos para uso em *scripts*. Isso permite a linguagem ser muito simples, flexível e gratuita, facilitando a rapidez no desenvolvimento de aplicações e poderosa para criação de diversos tipos de aplicações. Também uma das principais características é permitir a extensibilidade. Isto é, quando uma aplicação requer alguma funcionalidade que não é oferecida pela linguagem, estas funcionalidades

podem ser implementadas usando outra linguagem de programação, como *C/C++* ou *Java* e depois integrar a nova funcionalidade na linguagem *Tcl*.

Desenvolvida pela equipe de John Ousterhout 1988 na Universidade da Califórnia e mantida, posteriormente, pela *Sun Microsystems* e *Ajuba Solution*, pode ser utilizada em diferentes sistemas operacionais, como *Linux, Microsoft Windows* , *Mac OS* e *Solaris* e até em sistemas embarcados, como exemplo *Smartphones* e Roteadores *CISCO*.

Atualmente a linguagem *Tcl* encontra-se na versão 8.4. Já a *OTCL* é a linguagem *Tcl* orientada a objetos, pode ser utilizada numa grande quantidade de aplicações, mas muito utilizada em programas com interface gráfica, onde geralmente se utiliza a extensão *Tk Toolkit*, que é uma biblioteca com interface gráfica padrão para suporte à linguagem *Tcl*. O suporte dessa biblioteca para a linguagem *Tcl* permite aos programadores criar interfaces gráficas amigáveis, através da inserção de menus, botões e ações que não poderiam ser feitos apenas com a própria linguagem *Tcl*.

3.2 METODOLOGIA DE DIAGNÓSTICO DO ALGORITMO

De acordo com o contexto apresentado, este trabalho propõe uma solução com base nos modelos de diagnóstico de falhas abordados e a ferramenta *SAPOTI,* a criação de um algoritmo que possa atender o que é proposto, ou seja, o diagnóstico de falhas em servidores *web* de alta disponibilidade. Para isso, o algoritmo identificará servidores *web* falhos e realizará uma medida corretiva para contornar situações de falhas.

O código do algoritmo encontra-se no apêndice C deste trabalho. Esse algoritmo mantém em cada servidor uma matriz 5x2, onde cada linha é composta, no primeiro campo pelo *IP* do servidor e o segundo campo corresponde ao estado (com falha ou sem falha) de cada servidor. Durante a execução do algoritmo, essa matriz é atualizada em cada servidor e a cada rodada de teste o algoritmo identificará através dessa matriz, se existe algum servidor falho, e caso não encontra, inicia a rodada novamente. Se caso identificar algum servidor falho, enviará um e-mail para o responsável técnico, avisando que servidor x caiu e além de avisar o responsável, o algoritmo redirecionará o serviço para outro servidor sem falha e assim iniciará a rodada de teste novamente, isso sendo feito de forma automatizada e transparente.

Portanto, o diagnóstico desses servidores será realizado por uma estratégia adotada para que quando um servidor deixar de fornecer o serviço ou deixar de responder devido à uma falha ocorrida, outro servidor do grupo assumirá as responsabilidades que este até então vinha realizando e dará continuidade para prover os serviços demandados e caso ainda encontra-se inoperante, o servidor continuará a verificação até encontrar, dentro do grupo de servidores, outro servidor apto a prover o serviço, sendo essas verificações feitas de modo assíncrono entre eles. Portanto existirá a possibilidade de o servidor, quando sendo testado por outro, também iniciar os testes. Como essa estratégia é baseada no modelo *PCM*, mesmo quando apenas uma máquina estiver sem falha o serviço ainda continuará sendo disponibilizado.

Quanto a simulações, foram feitas de forma a forçar a falha com um servidor, depois com dois, três e com quatro servidores. Embora houvesse cinco máquinas servidoras, a premissa do modelo *PMC* define como sendo diagnosticável se puder ser feito com no máximo *n-1* máquinas.

Portanto, foi criado um cenário contendo uma rede cabeada caracterizada e estruturada conforme demonstrado na Figura 13, para ser simulado e aplicado na contextualização de um ambiente contendo cinco servidores *web*. Este cenário de simulação foi modelado através do *Script tcl*, que pode ser encontrado no apêndice B deste trabalho. Esse ambiente é formado por cinco servidores *web* (servidor de página) e considera-se ainda que haja uma política de replicação dos dados nos servidores. Eles permaneceram em simulação e dentro desse intervalo de tempo, eventos pré-definidos aleatoriamente foram executados para analisar o desempenho do algoritmo no ambiente descrito durante a execução do algoritmo proposto.

Além do uso do simulador *Network Simulator-2* para simular e analisar o ambiente proposto, ferramentas como *Gnuplot* e *NAM* foram aqui empregadas para a geração de gráficos estatísticos e a visualização dos arquivos de *log* em formato animado, para ao final relatar a performance do algoritmo proposto.

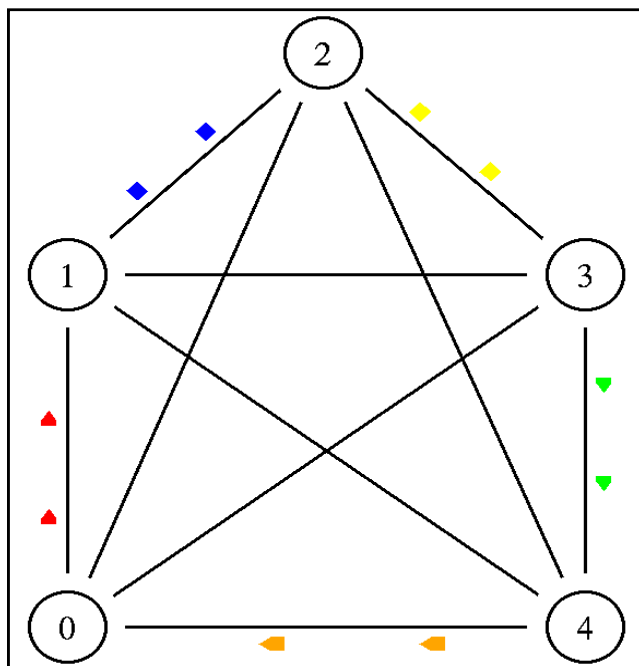

Figura 13: cenário da simulação

Na Figura 13 é apresentado o instante onde todos os servidores estão sem falha e, portanto um servidor testa sempre o próximo servidor do grupo. Já a Figura 14, o instante da simulação onde o servidor 1 apresenta uma falha, neste instante, o servidor 0 começa a testar o servidor 2. Três servidores apresentam falhas e pode ser vista através da Figura 15, neste caso o servidor 3 testa o servidor 4 e vice-versa, e o servidor número 3 assume a disponibilidade do serviço.

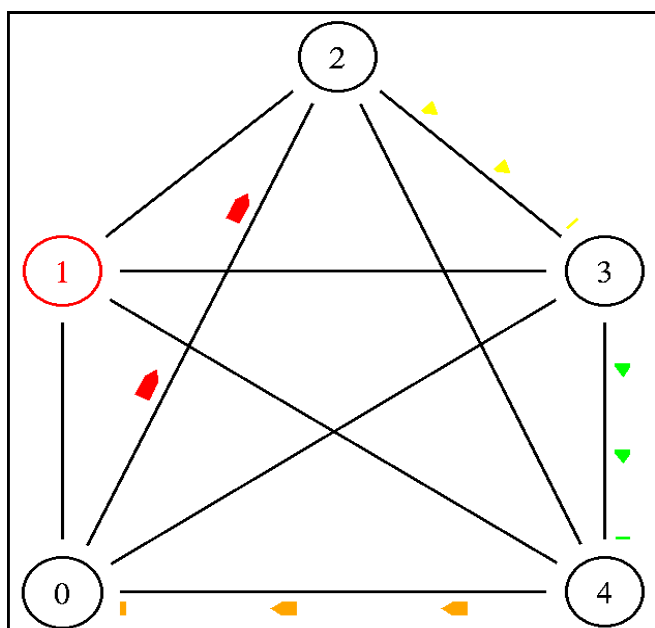

Figura 14: servidor 1 com falha

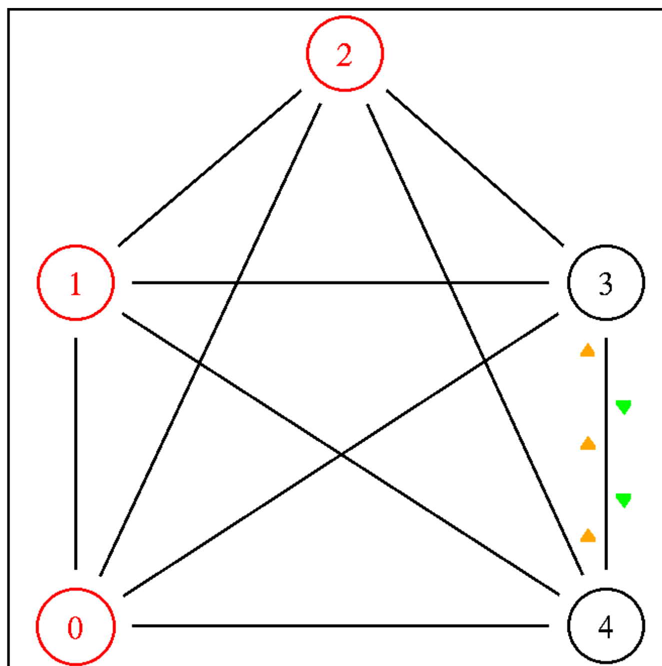

Figura 15: servidor 0, 1 e 2 com falha

3.3 SIMULAÇÃO DO CENÁRIO E ANÁLISE DOS TRACE

No apêndice B encontra-se o *Script* completo para simulação do cenário definido na Figura 13. Considera-se que todos os servidores enviam pacotes a uma taxa constante e em um intervalo de 0.006 segundos, através de um gerador de tráfego *CBR* sobre *UDP*. Considera-se também que os *links* são fixos de *4* Mb (megabits) e tem um atraso de 15ms (milisegundos), usando a política de fila *DropTail*, esta política implementa o algoritmo *FIFO* (*First-In;First-Out*), ou seja, o primeiro pacote que chega na fila é o primeiro a sair.

A seguir, os principais trechos do código fonte *tcl* encontrado no apêndice B. A Figura 16 representa o trecho do código onde são criados os *links* entre os servidores.

Cada linha da figura representa a criação de um *link*. O primeiro campo da primeira linha diz ao escalonador (*$ns*) para criar um *link* do tipo *duplex-link* (2º campo), tendo como origem o servidor_0 (3º campo) e servidor de destino (4º campo), o 5º campo representa o tamanho do *link,* o 6º campo representa o *delay* (atraso do *link* em milésimo de segundos) e o último campo, o tipo de fila implementado no *link*.

```
$ns duplex-link $servidor_0 $servidor_1 4Mb 15.0ms DropTail
$ns duplex-link $servidor_1 $servidor_2 4Mb 15.0ms DropTail
$ns duplex-link $servidor_2 $servidor_3 4Mb 15.0ms DropTail
$ns duplex-link $servidor_3 $servidor_4 4Mb 15.0ms DropTail
$ns duplex-link $servidor_4 $servidor_0 4Mb 15.0ms DropTail
$ns duplex-link $servidor_0 $servidor_2 4Mb 15.0ms DropTail
$ns duplex-link $servidor_0 $servidor_3 4Mb 15.0ms DropTail
$ns duplex-link $servidor_2 $servidor_4 4Mb 15.0ms DropTail
$ns duplex-link $servidor_1 $servidor_3 4Mb 15.0ms DropTail
$ns duplex-link $servidor_1 $servidor_4 4Mb 15.0ms DropTail
```

Figura 16: criação dos *links*

A Figura 17 representa a etapa de criação do agente de transporte, onde na primeira linha, é criado o agente de transporte usando o protocolo *UDP*, e na segunda linha, a vinculação desse agente com o servidor 1. Na terceira linha é criado outro agente do tipo *Null*, o qual será responsável por receber os pacotes, na quarta linha é vinculado esse agente *Null* com o servidor 2 e na última linha é conectado esses dois agentes.

```
set s_1_2 [new Agent/UDP]
$ns attach-agent $servidor_1 $s_1_2

set r_1_2 [new Agent/Null]
$ns attach-agent $servidor_2 $r_1_2
$ns connect $s_1_2 $r_1_2
```

Figura 17: criação do agente de transporte

A Figura 18 representa o trecho do código onde é criado os geradores de tráficos, onde a primeira linha representa a criação de um tráfico *CBR – Constant Bit Rate*, a seguir atribui-se ao gerador, o tamanho dos pacotes em *bytes* à gerar (2ª

linha) e um intervalo de 0.006 segundos (3ª linha) para geração de pacotes, e por último, a vinculação desse gerador de tráfico ao agente de transporte definido na Figura 17.

```
set traf_CBR_1_2 [new Application/Traffic/CBR]
$traf_CBR_1_2 set packetSize_ 480
$traf_CBR_1_2 set interval_ 0.006
$traf_CBR_1_2 attach-agent $s_1_2
```

Figura 18: criação dos geradores de tráficos

Como já mencionado no presente trabalho que durante a simulação é gerado arquivos de *trace*, então segue abaixo o trecho inicial do arquivo *trace* do primeiro pacote enviado pelo servidor 0.

```
+ 0 0 1 cbr 480 ------- 1 0.0 1.0 0 0
- 0 0 1 cbr 480 ------- 1 0.0 1.0 0 0
r 0.01596 0 1 cbr 480 ------- 1 0.0 1.0 0 0
```

Figura 19: trecho inicial do arquivo de *trace* do *NS-2* (editado)

Cada linha do *trace* corresponde à um evento. O primeiro campo define o tipo de evento ocorrido que pode ser:

- Colocado na fila (+)
- Enviado (-)
- Recebido (r)
- Descartado (d)

O segundo campo corresponde ao tempo em que o evento ocorreu. O terceiro e quarto definem, respectivamente, o servidor de origem e o servidor de

destino do pacote do tipo *CBR* de tamanho 480 *bytes*. A seguir, o identificador do tráfego, seguido pelo servidor de origem e a porta e servidor de destino e a porta. Os dois últimos campos são números usados pelo *NS-2* para informar o identificador único do pacote.

Neste *trace*, pode-se ver que inicialmente o servidor 0 gera um pacote *CBR* e o coloca na fila (+) no tempo 0s, ou seja, no inicio da simulação. Logo em seguida, como a fila esta vazia, o pacote é retirado da fila e enviado para o destino (-). No tempo 0.01596s, o pacote é recebido (r) pelo servidor 1.

3.4 RESULTADOS E MÉTRICAS

Durante a realização da simulação e análise dos resultados, foram feitos medições quanto à quantidade de pacotes enviados, pacotes recebidos, pacotes descartados, taxa de entrega de pacotes e a latência do algoritmo de diagnóstico. Para aumentar a confiabilidade das simulações, foram executadas 35 simulações diferentes, alternando e variando os eventos de forma aleatória. No apêndice D encontra-se o código fonte do *Script* que realiza as 35 simulações aleatoriamente.

O resultado obtido permitiu gerar gráficos, como na Figura 20 mostrando a quantidade de pacotes enviados e recebidos que diminui conforme há um aumento na quantidade de máquinas com falha. Assim a quantidade de pacotes descartados tende a aumentar conforme aumenta a quantidade de máquinas falhando. Esse fato pode ser observado através da Figura 21.

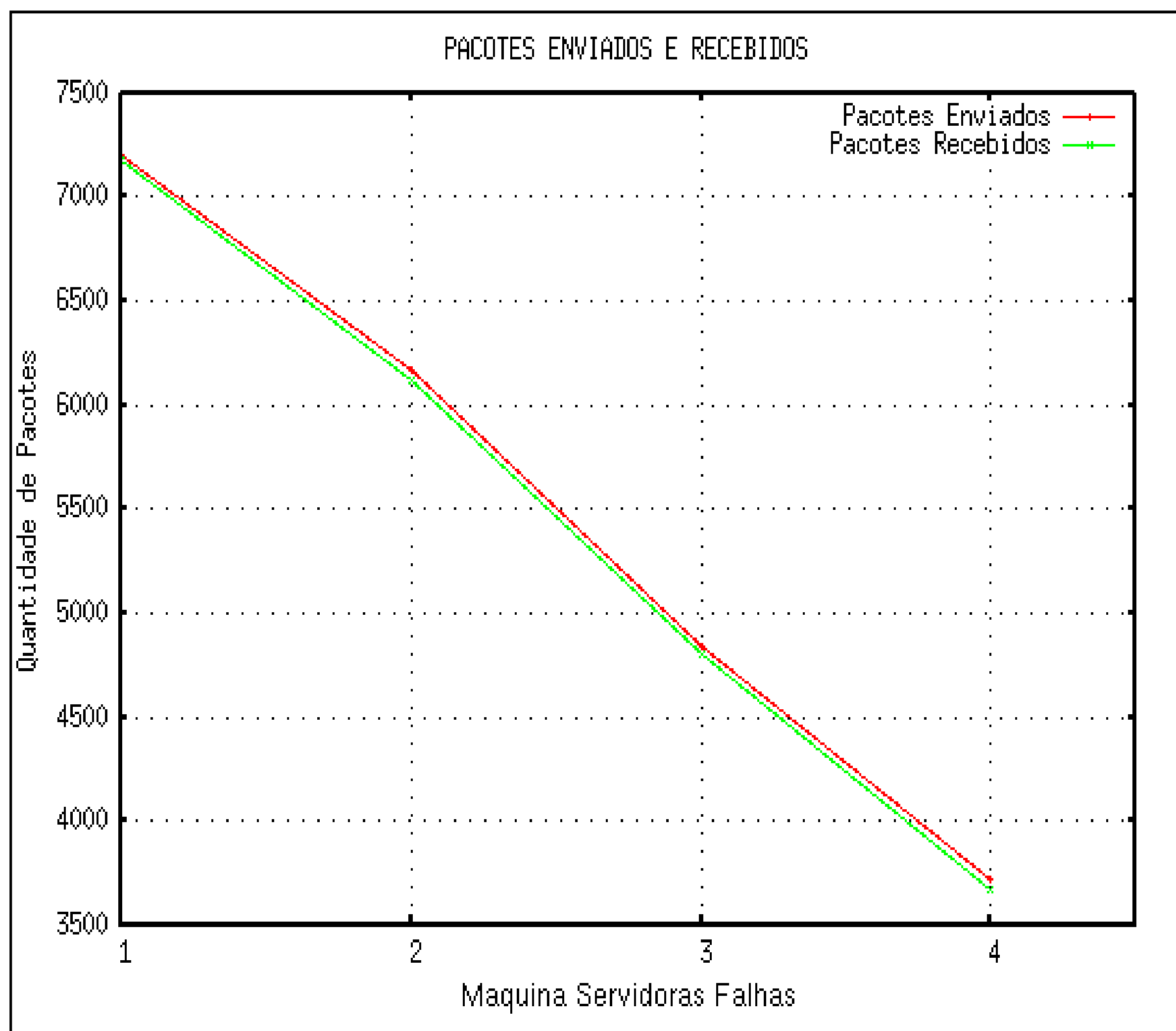

Figura 20: média de pacotes enviados e recebidos

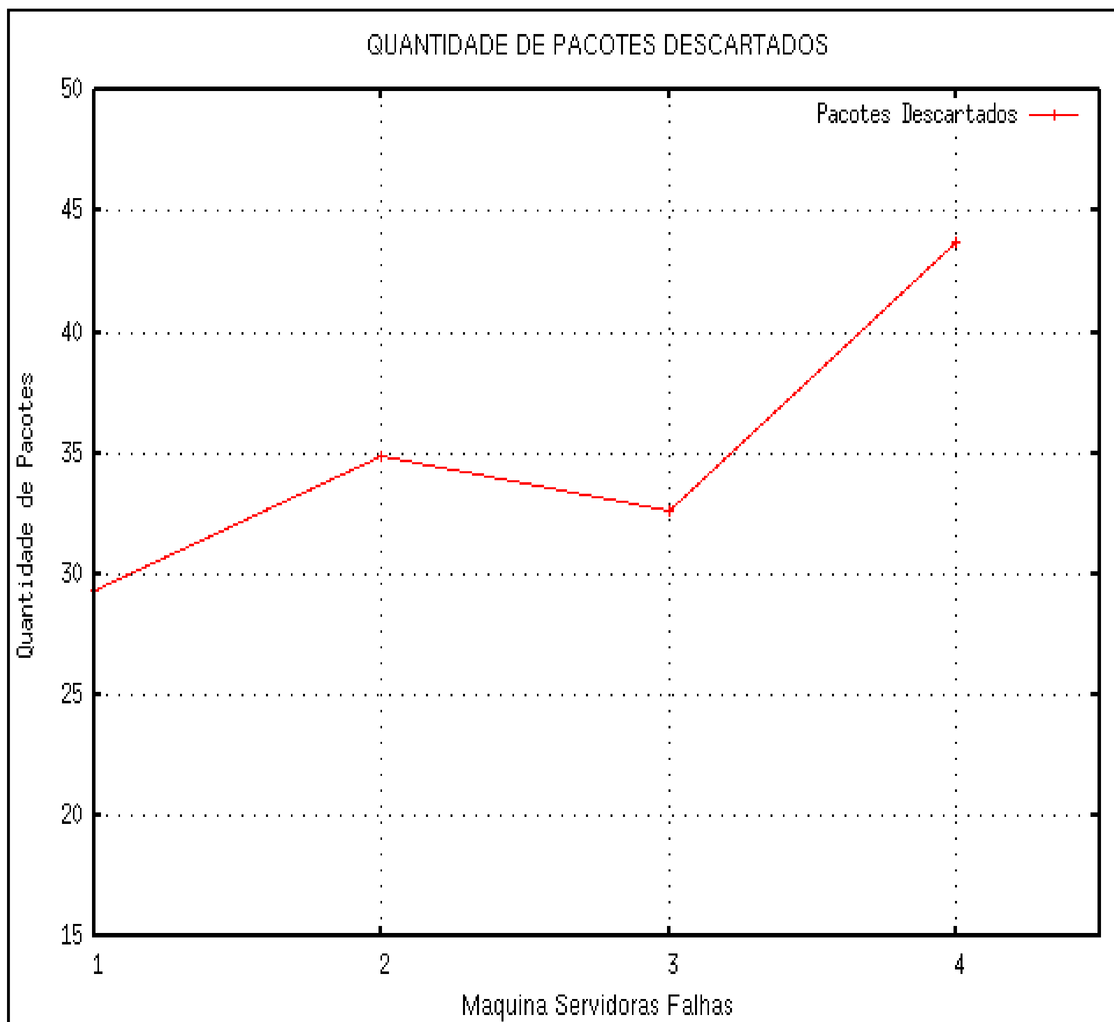

Figura 21: média de pacotes descartados

Portanto, a quantidade de pacotes enviados e recebidos é inversamente proporcional a quantidade de máquinas falhando, ou seja, quanto mais máquina falhando, menor será a quantidade de pacotes enviados e por conseqüente os recebidos. No entanto, a quantidade de pacotes descartados é proporcional a quantidade de máquinas falhando, ou seja, quanto mais máquina falhando, maior será o descarte de pacotes.

Quanto à taxa de entrega, onde é medida a porcentagem de entrega de pacotes através da razão de (recebidos/enviados), pode-se observar na Figura 22 que há uma estabilidade e também uma pequena variância na taxa de entrega conforme aplica-se o algoritmo de diagnóstico. Ou seja, ele é praticamente eficaz e confiável quando até quatro máquinas estão em falhas, isto é, quando apenas existir uma máquina sem falha, o algoritmo ainda consegue fazer o diagnóstico correto e a

partir disso tomar uma ação para identificar qual servidor do grupo irá disponibilizar o serviço.

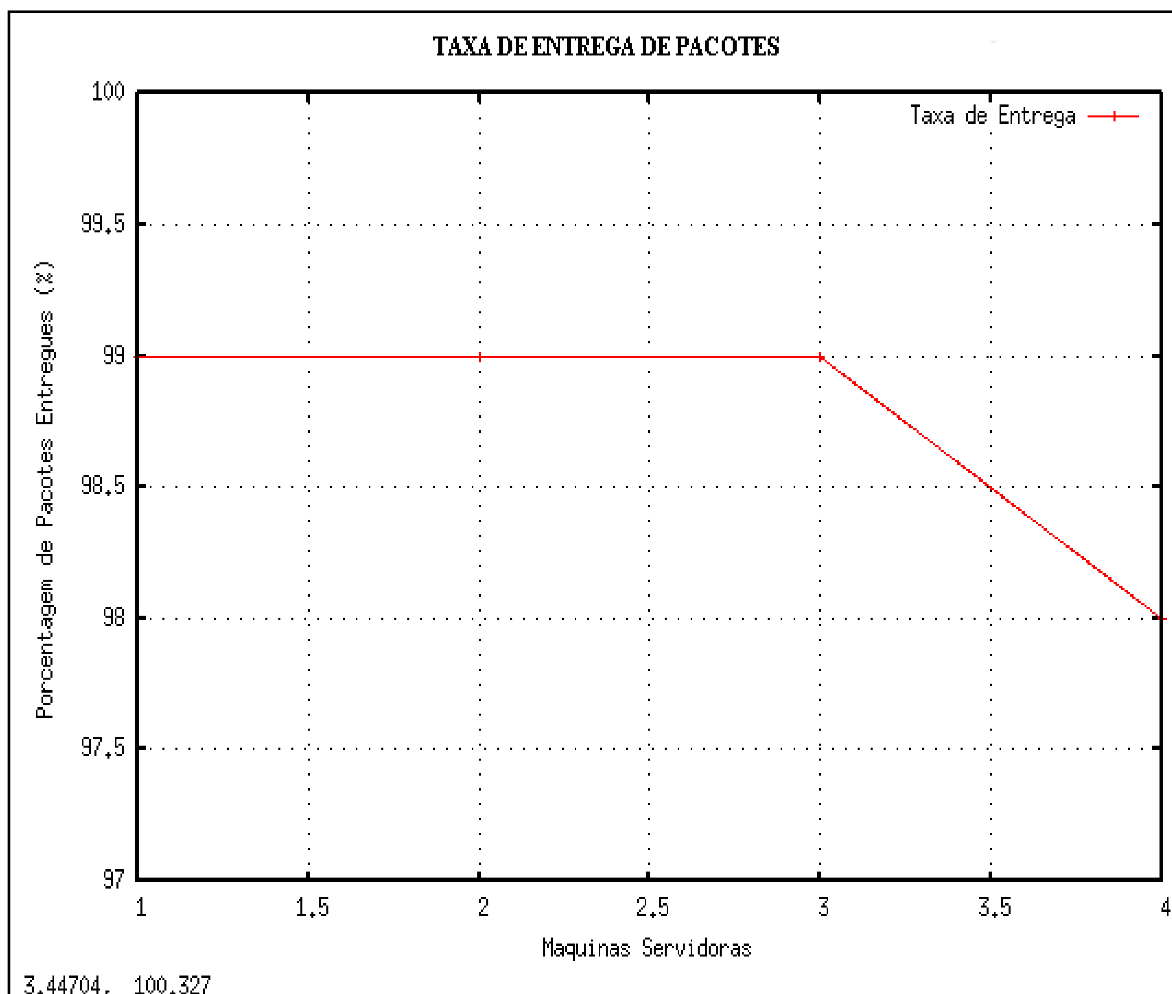

Figura 22: taxa de entrega de pacotes

Para a latência do algoritmo, observou-se que a quantidade de testes realizados no pior dos casos é a duração de 4 testes, quando no pior caso, isto é, quando ocorrer de todas as máquinas menos uma falharem. E no melhor dos casos quanto todas as máquinas estiverem conexas e sem falha, apenas será necessária a duração de um teste. Uma vez que no melhor dos casos todas as máquinas conseguiram se testarem no mesmo momento da realização de um único teste. Este fato pode ser observado através da Figura 23.

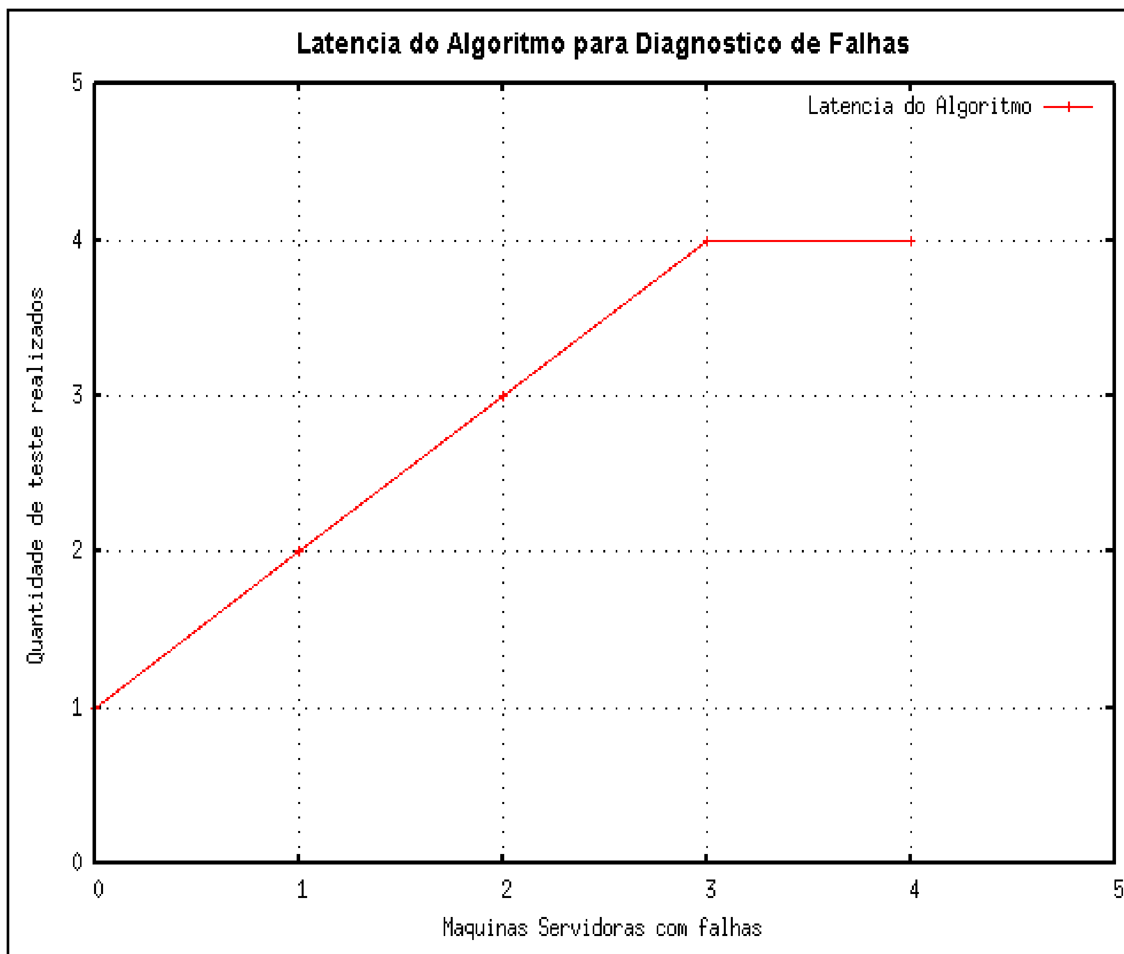

Figura 23: latência do algoritmo para diagnóstico de falhas

4 CONSIDERAÇÕES FINAIS

4.1 CONCLUSÕES

Através do desenvolvimento deste trabalho sobre diagnóstico de falhas em sistemas distribuídos, em especial à servidores *web*, possibilitou ter um melhor entendimento dos mecanismos utilizados nesse tipo de sistema tão susceptível à falha, para garantir a confiabilidade e disponibilidade no funcionamento dos serviços.

Na questão de diagnóstico de falhas, analisou-se, principalmente, os tipos de falhas, abordando posteriormente os modelos de diagnóstico de falhas e ferramentas relacionadas.

Como resultado do trabalho, pode-se concluir que é possível mitigar as falhas se empregadas técnicas de diagnóstico de falhas adequadas à esses sistema. Além disso, desenvolver um algoritmo é uma tarefa complexa, sendo necessárias, técnicas que auxiliem a implementação. Assim a simulação, através do uso do *NS-2*, é uma técnica utilizada neste trabalho que possibilitou realizar essa tarefa, permitindo testar e relatar o algoritmo proposto com base nas métricas apresentadas.

Um dos desafios encontrados na execução do trabalho foi também o estudo do *NS-2* e principalmente devido à deficiência de materiais sobre a ferramenta. Mas pelo contrário, ferramentas adicionais como o visualizador *Nam* e *Gnuplot* auxiliaram na verificação do funcionamento correto do algoritmo para colher os resultados necessários. Através dessas ferramentas, provou-se que o algoritmo de diagnóstico de falha apresentado garante a disponibilidade dos servidores *web*, mesmo quando surgem falhas aleatórias. Portanto se o algoritmo fosse implementado, teria um ótimo desempenho em relação às métricas utilizadas e definidas neste presente trabalho.

4.2 TRABALHOS FUTUROS

Durante a elaboração deste trabalho, várias limitações foram sentidas e que talvez possam ser possibilidades para dar continuidade para novos trabalhos. Segue algumas sugestões para trabalhos futuros:

- Colocar o algoritmo em execução para permitir melhorar o funcionamento dos servidores *web* em relação à disponibilidade dos serviços prestados;
- Desenvolver um novo modelo de tráfico para o simulador *NS-2*;
- Implementar um algoritmo baseado no modelo de diagnóstico comparativo e posterior comparação com os resultados deste trabalho, ou seja, ao modelo *PMC*;
- Aumento da quantidade de servidores e uma escolha mais bem elaborada do modelo *PMC*, que permita trabalhar em *clusters*, como sugestão o modelo *Hi-ADSD with Detours*;

REFERÊNCIAS

ALBINI, Luiz Carlos Pessoa.; *Um Algoritmo Baseado em Comparações para Diagnostico Distribuído Hierárquico.* 2000. 112 f. Dissertação (Mestrado em Informática). Universidade Federal do Paraná, Curitiba.

ARAUJO, Rafael G. B. de. *A ferramenta de Simulação NS (Network Simulator) – Estudos de Caso.* 2003. 59 f. Monografia. Universidade de Salvador, Salvador.

AVIZIENIS, A.; *The Four-Universe Information System Model For Study of Fault Tolerance***.** [Publicação] Proceedings of the 24th Annual International Symposium on Fault-Tolerant Computing Santa Monica, California,1982.

BIANCHINI, R. P.; BUSKENS, R.; *Implementation of On-Line Distributed System-Level Diagnosis Theory. IEEE Transactions on Computers*. vol 41. pp. 616-626, 1992.

BLOCH, M.; PIGNEUR, Y.; SEGEV, A.; *On the Road of Eletronic Commerce – a Business Value Framework, Gaining Competitive Advantage and Some Research Issues*. Disponível em: http://inforge.unil.ch/yp/Pub/ROAD_EC/EC.HTM. Acesso em: 12/05/2010.

BONA, Luiz Carlos E.; *Um algoritmo baseado em comparações para diagnóstico distribuído hierárquico*. 2000. 112 f. Dissertação (Mestrado em Informática). Universidade Federal do Paraná, Curitiba.

BONA, Luiz Carlos E.; *HyperGrid: Uma plataforma distribuída e confiável para computação em grade*. 2004. 89 f. Tese (Doutorado em Engenharia Elétrica e Informática Industrial). Centro Federal de Educação Tecnologica do Paraná-CEFET, Curitiba.

BONA, Luiz Carlos E.; *Hyperbone: uma Rede Overlay baseada em Hipercubo Virtual para Computacão Distribuída na Internet*. 2006. 102 f. Tese (Doutorado em Ciências). Universidade Tecnológica Federal do Paraná, Curitiba.

BRANCO, Moisés Almeida Castelo.; *Um Algoritmo para Diagnóstico Distribuído de Falhas em Redes de Computadores.* Dissertação. Ceará. 198 f. 1999. Disponível em http://www.frb.br/ciente/2005.2/BSI/BSI.LIMA&.F3.pdf. Acessado em 22/05/2010.

CHWA, K. Y.; HAKIMI, S. L.; *Schemes for Fault-Tolerant Computing: A Comparison of Modularly Redundant and t-Diagnoable Systems. Information and Control,* Vol. 49, pp. 212-238, 1981.

COULOURIS, George; Dollimore, Jean; Kindberg,Tim; *Distributed System Concepts and Design*. 2004. ADDISON WESLEY, Cap. 11 - 451 a 462 / Cap. 565 a 572.

DUARTE JR, E. P.; NANYA, T. *A Hierarchical Adaptive Distributed System-Level Diagnosis Algorithm. IEEE Transactions on Computers,* Vol.47, pp.34-45, No.1, 1998.

DUARTE Jr., E. P.; ALBINI, L. C. P.; BRAWERMAN, A. *An algorithm for distributed diagnosis of dynamic fault and repair events*. 7th IEEE International Conference on Parallel and Distributed Systems, IEEE/ICPADS'00, p. 299–306, 2000.

FERNANDES, Manuel J. M. de M.; *Sistemas Distribuídos Imune a Falhas Bizantinas.* 2008. 64 f. Dissertação (Mestrado em Engenharia Eletrotécnica e de Computadores). Universidade Técnica de Lisboa, Portugal.

GOULD, R.; *Graph Theory.* The Benjamim/Cummings Publishing Company Inc., 1988.

HILGENSTIELER, Egon et al. *SAPOTI: Servidores de Aplicações cOnfiáveis Tcp/IP.* 2003. 8 f. Universidade Federal do Paraná, Curitiba.

JALOTE, P.; *Fault Tolerance in Distributed Systems.* P T R Prentice Hall, 1994.

KALAKOTA, R.; ROBINSON, M.*; E-Business: Estrategias para Alcançar o Sucesso no Mundo Digital*. 2.ed. Porto Alegre: Bookman, 2002. 470 f.

LAMPORT, Leslie; SHOSTAK, Robert; PEASE, Marshall. *The Byzantine Generals problem*. ACM Transactions on Programming Languages and Systems, 1982.

MALEK, M.; *A Comparison Connection Assigment for Diagnosis of Multiprocessor Systems.* Proc. Seventh International Symp. Computer Architecture, 1980.

MAENG, J.; Malek, M.; *A Comparison Connection Assigment for Self-Diagnosis of Multiprocessor Systems. Digest 11 th Int'l Symp. Fault Tolerant Computing*, 1981.

NS, *The Network Simulator – NS*. [online] Disponível em http://www.isi.edu/nsnam. Acessado em 13/09/2010.

NAM, *The Network Animator – Nam*. [online] Disponível em http://www.isi.edu/nsnam/nam/index.html. Acessado em 15/09/2010.

OLIVEIRA, Márcio Izaquiel de. *Obtenção de QoS através de MPLS e DIFFSERV.* 2005. 56 f. Monografia (Curso de Ciência da Computação). Centro Universitário Feevale, Novo Hamburgo.

PREPARATA, F. P.; METZE, G.; CHIEN, R. T.; *On the Connection Assignment Problem of Diagnosable Systems*. IEEE Transactions on Electron Computers, New York, vol. EC-16, pp. 848-854, Dec, 1967.

SHANNON, Robert E. *System Simulation: The Art and Science*. Prentice-Hall. Englewood Cliffs, New Jersey. 1975.

TANENBAUM, Andrew; STEEN, Maarten van.; *Distributed System Principles and Paradigms*. Pearson Education 2002 - Cap. 7 - p. 361 a 410 / Cap. 9 - p. 494 a 522.

VICENT, J.; *High Availability Networking with Cisco*. EUA: Addison Wesley, 2001.

ZIWICH, Roverli Pereira**;** *Detecção Distribuída de Alterações em Sistemas com Conteúdo Replicado Utilizando Diagnóstico Baseado em Comparações*. 2004. 129 f. Dissertação (Mestrado em Ciências Exatas). Universidade Federal do Paraná, Curitiba.

APÊNDICE A – Instalação do *Network Simulator-2*

Existem duas formas de instalar o simulador *NS-2*. Primeiro, baixando o pacote completo *ns-allinone*, o qual contém todos os pacotes descritos abaixo ou baixando cada pacote separadamente e instalando cada pacote conforme a necessidade, sendo que não é necessário ter todos os pacotes descritos abaixo para que o simulador funcione. Esses pacotes podem ser encontrados para *download* em *http://www.isi.edu/nsnam.*

Antes de iniciar a instalação do *NS-2*, é necessário ter um compilador *C++* na máquina onde será instalado o *NS-2* e também é necessário alguns pacotes externos do sistema operacional (para maiores informações consultar manual do *NS-2* em *http://www.isi.edu/nsnam).*

O pacote completo do simulador *NS-2* é composto pelos módulos:

- *Tcl/Tk*: Interpretador de linguagem *Tcl*, que é a interface do simulador com o usuário.
- *OTcl*: biblioteca de suporte para orientação a objetos para o *Tcl.*
- *Tclcl*: implementação de classes para Tcl.
- *ns*: classes do simulador propriamente dito.
- *nam*: visualizador e animador gráfico de topologias de rede e simulação.
- *xgraph*: ferramentas para plotagens de gráficos.
- *CWeb* e *SGB*: bibliotecas requeridas para *sgb2*-ns (idem abaixo) e gt-itm (idem abaixo).
- *Gt-itm*, *sgb2-ns* e *gt-itm*: gerador de topologias.
- *Zlib*: ferramenta para compressão de arquivos.

Após fazer o *download* e verificar os pacotes necessários para a instalação do simulador, deve-se descompactar os pacotes dentro do diretório do usuário e executar o *Script* de instalação (*./install*).

Assim os pacotes serão então compilados apropriadamente e após essa etapa ser concluída, o *script* dará uma mensagem para realizar modificações na variável de ambiente *PATH* do sistema operacional, sendo estas modificações realizadas e configuradas de forma manual e específica para cada usuário.

A próxima etapa é fazer a validação do *NS-2* que nada mais é do que realizar simulações pré-programadas e comparar os resultados com os resultados padrão do simulador. Isso pode ser feito executando o *Script* de validação (*./validate*). Após essa etapa ser concluída com sucesso, o simulador *NS-2* está pronto para uso.

APÊNDICE B – Linguagem *TCL* referente ao proposto

```
#
======================================================================
==============================================
# 01 - Parametros Iniciais
#
======================================================================
==============================================
  set temp_simulacao 10;
  set qtde_servidor 5;

#
======================================================================
==============================================
# 02 - Cria o Escalonador de Eventos - Variavel (ns)
#
======================================================================
==============================================
  set ns [new Simulator]

# define as cores
  $ns color 1 red
  $ns color 2 blue
  $ns color 3 yellow
  $ns color 4 green
  $ns color 5 orange

#
======================================================================
==============================================
# 03 - Define os arquivos de trace
#
======================================================================
==============================================
  set nf [open nam_4-1.nam w]                ;# cria ou abre um arquivo de nam .nam
  $ns namtrace-all $nf                       ;# grava os passos da simulacao no formato
NAM
  set tf [open tr_4-1.tr w]           ;# cria ou abre um arquivo de trace .tr
  $ns trace-all $tf                          ;# usa a funcao trace-all em formato geral para
posterior analise


#
======================================================================
==============================================
# 04 - Cria os nodos
#
======================================================================
==============================================
# laço para criar os nodos
  for {set i 0} {$i < $qtde_servidor} {incr i} {
        set servidor_$i [$ns node]
  }
```

```
#
=============================================================================
================================================
# 05 - Cria os Enlaces/Links entre os nodos
#
=============================================================================
================================================
# largura_Banda - atraso - tipo da fila(DropTail(first in - first out)

  $ns duplex-link $servidor_0 $servidor_1 4Mb 15.0ms DropTail
  $ns duplex-link $servidor_1 $servidor_2 4Mb 15.0ms DropTail
  $ns duplex-link $servidor_2 $servidor_3 4Mb 15.0ms DropTail
  $ns duplex-link $servidor_3 $servidor_4 4Mb 15.0ms DropTail
  $ns duplex-link $servidor_4 $servidor_0 4Mb 15.0ms DropTail
  $ns duplex-link $servidor_0 $servidor_2 4Mb 15.0ms DropTail
  $ns duplex-link $servidor_0 $servidor_3 4Mb 15.0ms DropTail
  $ns duplex-link $servidor_2 $servidor_4 4Mb 15.0ms DropTail
  $ns duplex-link $servidor_1 $servidor_3 4Mb 15.0ms DropTail
  $ns duplex-link $servidor_1 $servidor_4 4Mb 15.0ms DropTail

#
=============================================================================
================================================
# 06 - Cria agentes de transporte e gerador de trafico
#
=============================================================================
================================================
  #
=============================================================================
==========================================
  # cria agente UDP
    set s_0_1 [new Agent/UDP]                          ;#Cria um agente de transporte UDP
    $ns attach-agent $servidor_0 $s_0_1                ;#vincula o agente de transporte UDP
ao servidor

    set r_0_1 [new Agent/Null]                         ;#cria um agente de transporte para
receber o trafico
    $ns attach-agent $servidor_1 $r_0_1                ;#vincula o agente de transporte UDP
ao servidor
    $ns connect $s_0_1 $r_0_1                          ;#conecta os agente de transporte


    set s_1_2 [new Agent/UDP]
    $ns attach-agent $servidor_1 $s_1_2

    set r_1_2 [new Agent/Null]
    $ns attach-agent $servidor_2 $r_1_2
    $ns connect $s_1_2 $r_1_2


    set s_2_3 [new Agent/UDP]
    $ns attach-agent $servidor_2 $s_2_3

    set r_2_3 [new Agent/Null]
    $ns attach-agent $servidor_3 $r_2_3
```

```
$ns connect $s_2_3 $r_2_3


set s_3_4 [new Agent/UDP]
$ns attach-agent $servidor_3 $s_3_4

set r_3_4 [new Agent/Null]
$ns attach-agent $servidor_4 $r_3_4
$ns connect $s_3_4 $r_3_4


set s_4_0 [new Agent/UDP]
$ns attach-agent $servidor_4 $s_4_0

set r_4_0 [new Agent/Null]
$ns attach-agent $servidor_0 $r_4_0
$ns connect $s_4_0 $r_4_0


set s_0_2 [new Agent/UDP]
$ns attach-agent $servidor_0 $s_0_2

set r_0_2 [new Agent/Null]
$ns attach-agent $servidor_2 $r_0_2
$ns connect $s_0_2 $r_0_2


set s_0_3 [new Agent/UDP]
$ns attach-agent $servidor_0 $s_0_3

set r_0_3 [new Agent/Null]
$ns attach-agent $servidor_3 $r_0_3
$ns connect $s_0_3 $r_0_3


set s_0_4 [new Agent/UDP]
$ns attach-agent $servidor_0 $s_0_4

set r_0_4 [new Agent/Null]
$ns attach-agent $servidor_4 $r_0_4
$ns connect $s_0_4 $r_0_4


set s_1_3 [new Agent/UDP]
$ns attach-agent $servidor_1 $s_1_3

set r_1_3 [new Agent/Null]
$ns attach-agent $servidor_3 $r_1_3
$ns connect $s_1_3 $r_1_3


set s_1_4 [new Agent/UDP]
$ns attach-agent $servidor_1 $s_1_4

set r_1_4 [new Agent/Null]
```

```
$ns attach-agent $servidor_4 $r_1_4
$ns connect $s_1_4 $r_1_4


set s_1_0 [new Agent/UDP]
$ns attach-agent $servidor_1 $s_1_0

set r_1_0 [new Agent/Null]
$ns attach-agent $servidor_0 $r_1_0
$ns connect $s_1_0 $r_1_0


set s_2_4 [new Agent/UDP]
$ns attach-agent $servidor_2 $s_2_4

set r_2_4 [new Agent/Null]
$ns attach-agent $servidor_4 $r_2_4
$ns connect $s_2_4 $r_2_4


set s_2_0 [new Agent/UDP]
$ns attach-agent $servidor_2 $s_2_0

set r_2_0 [new Agent/Null]
$ns attach-agent $servidor_0 $r_2_0
$ns connect $s_2_0 $r_2_0



set s_2_1 [new Agent/UDP]
$ns attach-agent $servidor_2 $s_2_1

set r_2_1 [new Agent/Null]
$ns attach-agent $servidor_1 $r_2_1
$ns connect $s_2_1 $r_2_1


set s_3_0 [new Agent/UDP]
$ns attach-agent $servidor_3 $s_3_0

set r_3_0 [new Agent/Null]
$ns attach-agent $servidor_0 $r_3_0
$ns connect $s_3_0 $r_3_0



set s_3_1 [new Agent/UDP]
$ns attach-agent $servidor_3 $s_3_1

set r_3_1 [new Agent/Null]
$ns attach-agent $servidor_1 $r_3_1
$ns connect $s_3_1 $r_3_1


set s_3_2 [new Agent/UDP]
```

```
$ns attach-agent $servidor_3 $s_3_2

set r_3_2 [new Agent/Null]
$ns attach-agent $servidor_2 $r_3_2
$ns connect $s_3_2 $r_3_2


set s_4_1 [new Agent/UDP]
$ns attach-agent $servidor_4 $s_4_1

set r_4_1 [new Agent/Null]
$ns attach-agent $servidor_1 $r_4_1
$ns connect $s_4_1 $r_4_1


set s_4_2 [new Agent/UDP]
$ns attach-agent $servidor_4 $s_4_2

set r_4_2 [new Agent/Null]
$ns attach-agent $servidor_2 $r_4_2
$ns connect $s_4_2 $r_4_2


set s_4_3 [new Agent/UDP]
$ns attach-agent $servidor_4 $s_4_3

set r_4_3 [new Agent/Null]
$ns attach-agent $servidor_3 $r_4_3
$ns connect $s_4_3 $r_4_3


 #
===========================================================================
==========================================
 # cria gerador de Trafico CBR sobre o UDP
  set traf_CBR_0_1 [new Application/Traffic/CBR]            ;#Cria um gerador de trafego
CBR('constant bit rate')
  $traf_CBR_0_1 set packetSize_ 480                                   ;#tamanho dos
pacotes em bytes
  $traf_CBR_0_1 set interval_ 0.006                                   ;#intervalo de envio
entre um pacote e o próximo pacote
  $traf_CBR_0_1 attach-agent $s_0_1                                   ;#vincula o gerador de
trafico sobre o agente UDP criado


  set traf_CBR_1_2 [new Application/Traffic/CBR]
  $traf_CBR_1_2 set packetSize_ 480
  $traf_CBR_1_2 set interval_ 0.006
  $traf_CBR_1_2 attach-agent $s_1_2




  set traf_CBR_2_3 [new Application/Traffic/CBR]
  $traf_CBR_2_3 set packetSize_ 480
```

```
$traf_CBR_2_3 set interval_ 0.006
$traf_CBR_2_3 attach-agent $s_2_3


set traf_CBR_3_4 [new Application/Traffic/CBR]
$traf_CBR_3_4 set packetSize_ 480
$traf_CBR_3_4 set interval_ 0.006
$traf_CBR_3_4 attach-agent $s_3_4


set traf_CBR_4_0 [new Application/Traffic/CBR]
$traf_CBR_4_0 set packetSize_ 480
$traf_CBR_4_0 set interval_ 0.006
$traf_CBR_4_0 attach-agent $s_4_0


set traf_CBR_0_2 [new Application/Traffic/CBR]
$traf_CBR_0_2 set packetSize_ 480
$traf_CBR_0_2 set interval_ 0.006
$traf_CBR_0_2 attach-agent $s_0_2


set traf_CBR_0_3 [new Application/Traffic/CBR]
$traf_CBR_0_3 set packetSize_ 480
$traf_CBR_0_3 set interval_ 0.006
$traf_CBR_0_3 attach-agent $s_0_3


set traf_CBR_0_4 [new Application/Traffic/CBR]
$traf_CBR_0_4 set packetSize_ 480
$traf_CBR_0_4 set interval_ 0.006
$traf_CBR_0_4 attach-agent $s_0_4


set traf_CBR_1_3 [new Application/Traffic/CBR]
$traf_CBR_1_3 set packetSize_ 480
$traf_CBR_1_3 set interval_ 0.006
$traf_CBR_1_3 attach-agent $s_1_3


set traf_CBR_1_4 [new Application/Traffic/CBR]
$traf_CBR_1_4 set packetSize_ 480
$traf_CBR_1_4 set interval_ 0.006
$traf_CBR_1_4 attach-agent $s_1_4


set traf_CBR_1_0 [new Application/Traffic/CBR]
$traf_CBR_1_0 set packetSize_ 480
$traf_CBR_1_0 set interval_ 0.006
$traf_CBR_1_0 attach-agent $s_1_0


set traf_CBR_2_4 [new Application/Traffic/CBR]
$traf_CBR_2_4 set packetSize_ 480
$traf_CBR_2_4 set interval_ 0.006
```

```
$traf_CBR_2_4 attach-agent $s_2_4

set traf_CBR_2_0 [new Application/Traffic/CBR]
$traf_CBR_2_0 set packetSize_ 480
$traf_CBR_2_0 set interval_ 0.006
$traf_CBR_2_0 attach-agent $s_2_0

set traf_CBR_2_1 [new Application/Traffic/CBR]
$traf_CBR_2_1 set packetSize_ 480
$traf_CBR_2_1 set interval_ 0.006
$traf_CBR_2_1 attach-agent $s_2_1

set traf_CBR_3_0 [new Application/Traffic/CBR]
$traf_CBR_3_0 set packetSize_ 480
$traf_CBR_3_0 set interval_ 0.006
$traf_CBR_3_0 attach-agent $s_3_0

set traf_CBR_3_1 [new Application/Traffic/CBR]
$traf_CBR_3_1 set packetSize_ 480
$traf_CBR_3_1 set interval_ 0.006
$traf_CBR_3_1 attach-agent $s_3_1

set traf_CBR_3_2 [new Application/Traffic/CBR]
$traf_CBR_3_2 set packetSize_ 480
$traf_CBR_3_2 set interval_ 0.006
$traf_CBR_3_2 attach-agent $s_3_2

set traf_CBR_4_1 [new Application/Traffic/CBR]
$traf_CBR_4_1 set packetSize_ 480
$traf_CBR_4_1 set interval_ 0.006
$traf_CBR_4_1 attach-agent $s_4_1

set traf_CBR_4_2 [new Application/Traffic/CBR]
$traf_CBR_4_2 set packetSize_ 480
$traf_CBR_4_2 set interval_ 0.006
$traf_CBR_4_2 attach-agent $s_4_2

set traf_CBR_4_3 [new Application/Traffic/CBR]
$traf_CBR_4_3 set packetSize_ 480
$traf_CBR_4_3 set interval_ 0.006
$traf_CBR_4_3 attach-agent $s_4_3

#
==========================================================================
==========================================
# 07 - Adiciona (escalona) os eventos na simulacao em funcao do tempo
```

```
#
=======================================================================================================================
# Associa trafico à uma cor para facilitar a visualizacao no NAM
  $s_0_1 set fid_ 1
  $s_1_2 set fid_ 2
  $s_2_3 set fid_ 3
  $s_3_4 set fid_ 4
  $s_4_0 set fid_ 5
  $s_0_2 set fid_ 1
  $s_0_3 set fid_ 1
  $s_0_4 set fid_ 1
  $s_1_3 set fid_ 2
  $s_1_4 set fid_ 2
  $s_1_0 set fid_ 2
  $s_2_4 set fid_ 3
  $s_2_0 set fid_ 3
  $s_2_1 set fid_ 3
  $s_3_0 set fid_ 4
  $s_3_1 set fid_ 4
  $s_3_2 set fid_ 4
  $s_4_1 set fid_ 5
  $s_4_2 set fid_ 5
  $s_4_3 set fid_ 5



  $ns at 0.0 "$traf_CBR_0_1 start"          ;#inicia a simulacao do trafico em 0 segundo
  $ns at 0.0 "$traf_CBR_1_2 start"
  $ns at 0.0 "$traf_CBR_2_3 start"
  $ns at 0.0 "$traf_CBR_3_4 start"
  $ns at 0.0 "$traf_CBR_4_0 start"
  $ns at [expr $temp_simulacao-0.01] "$traf_CBR_0_1 stop"    ;#paraliza o fluxo antes (0.01
seg) do fim da simulacao
  $ns at [expr $temp_simulacao-0.01] "$traf_CBR_1_2 stop"
  $ns at [expr $temp_simulacao-0.01] "$traf_CBR_2_3 stop"
  $ns at [expr $temp_simulacao-0.01] "$traf_CBR_3_4 stop"
  $ns at [expr $temp_simulacao-0.01] "$traf_CBR_4_0 stop"
  $ns at [expr $temp_simulacao-0.01] "$traf_CBR_0_2 stop"
  $ns at [expr $temp_simulacao-0.01] "$traf_CBR_0_3 stop"
  $ns at [expr $temp_simulacao-0.01] "$traf_CBR_0_4 stop"
  $ns at [expr $temp_simulacao-0.01] "$traf_CBR_1_3 stop"
  $ns at [expr $temp_simulacao-0.01] "$traf_CBR_1_4 stop"
  $ns at [expr $temp_simulacao-0.01] "$traf_CBR_1_0 stop"
  $ns at [expr $temp_simulacao-0.01] "$traf_CBR_2_4 stop"
  $ns at [expr $temp_simulacao-0.01] "$traf_CBR_2_0 stop"
  $ns at [expr $temp_simulacao-0.01] "$traf_CBR_2_1 stop"
  $ns at [expr $temp_simulacao-0.01] "$traf_CBR_3_0 stop"
  $ns at [expr $temp_simulacao-0.01] "$traf_CBR_3_1 stop"
  $ns at [expr $temp_simulacao-0.01] "$traf_CBR_3_2 stop"
  $ns at [expr $temp_simulacao-0.01] "$traf_CBR_4_1 stop"
  $ns at [expr $temp_simulacao-0.01] "$traf_CBR_4_2 stop"
  $ns at [expr $temp_simulacao-0.01] "$traf_CBR_4_3 stop"
  $ns at $temp_simulacao "finaliza"
```

```
#
================================================================================
============================================
# erros
# considerando um atraso de 0.1 segundos para comecar o teste no proximo nodo
#
================================================================================
============================================
   set temp [expr {rand()}]    ;#Gera um número aleatório entre 0 e 1.

   $ns rtmodel-at $temp down $servidor_1
   $ns at [expr $temp+0.0001] "$servidor_1 color red"
   $ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_0 $servidor_1 color black"
   $ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_1 $servidor_2 color black"
   $ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_1 $servidor_3 color black"
   $ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_1 $servidor_4 color black"
   $ns at $temp "$traf_CBR_0_1 stop"
   $ns at $temp "$traf_CBR_1_2 stop"
   $ns at $temp "$traf_CBR_1_3 stop"
   $ns at $temp "$traf_CBR_1_4 stop"
   $ns at $temp "$traf_CBR_1_0 stop"
   $ns at [expr $temp+0.1] "$traf_CBR_0_2 start"   ;# testa o proximo nodo depois de um
tempo


   set temp [expr {rand()+$temp+0.1}]

   $ns rtmodel-at $temp down $servidor_4
   $ns at [expr $temp+0.0001] "$servidor_4 color red"
   $ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_3 $servidor_4 color black"
   $ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_4 $servidor_0 color black"
   $ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_4 $servidor_1 color black"
   $ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_4 $servidor_2 color black"
   $ns at $temp "$traf_CBR_3_4 stop"
   $ns at $temp "$traf_CBR_4_0 stop"
   $ns at $temp "$traf_CBR_4_1 stop"
   $ns at $temp "$traf_CBR_4_2 stop"
   $ns at $temp "$traf_CBR_4_3 stop"
   $ns at [expr $temp+0.1] "$traf_CBR_3_0 start"


   set temp [expr {rand()+$temp+0.1}]

   $ns rtmodel-at $temp down $servidor_2
   $ns at [expr $temp+0.0001] "$servidor_2 color red"
   $ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_1 $servidor_2 color black"
   $ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_2 $servidor_3 color black"
   $ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_2 $servidor_4 color black"
   $ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_2 $servidor_0 color black"
   $ns at $temp "$traf_CBR_1_2 stop"
   $ns at $temp "$traf_CBR_2_3 stop"
   $ns at $temp "$traf_CBR_2_4 stop"
   $ns at $temp "$traf_CBR_2_0 stop"
   $ns at $temp "$traf_CBR_2_1 stop"
   $ns at [expr $temp+0.1] "$traf_CBR_0_3 start"
```

```
set temp [expr {rand()+$temp+0.1}]

$ns rtmodel-at $temp up $servidor_4
$ns at [expr $temp+0.0001] "$servidor_4 color black"
$ns at $temp "$traf_CBR_3_0 stop"
$ns at [expr $temp+0.1] "$traf_CBR_3_4 start"
$ns at [expr $temp+0.1] "$traf_CBR_4_0 start"


set temp [expr {rand()+$temp+0.1}]

$ns rtmodel-at $temp down $servidor_0
$ns at [expr $temp+0.0001] "$servidor_0 color red"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_4 $servidor_0 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_0 $servidor_1 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_0 $servidor_2 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_0 $servidor_3 color black"
$ns at $temp "$traf_CBR_4_0 stop"
$ns at $temp "$traf_CBR_0_1 stop"
$ns at $temp "$traf_CBR_0_2 stop"
$ns at $temp "$traf_CBR_0_3 stop"
$ns at $temp "$traf_CBR_0_4 stop"
$ns at [expr $temp+0.1] "$traf_CBR_4_3 start"


set temp [expr {rand()+$temp+0.1}]

$ns rtmodel-at $temp up $servidor_1
$ns at [expr $temp+0.0001] "$servidor_1 color black"
$ns at $temp "$traf_CBR_4_3 stop"
$ns at [expr $temp+0.1] "$traf_CBR_4_1 start"
$ns at [expr $temp+0.1] "$traf_CBR_1_3 start"


set temp [expr {rand()+$temp+0.1}]

$ns rtmodel-at $temp down $servidor_1
$ns at [expr $temp+0.0001] "$servidor_1 color red"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_0 $servidor_1 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_1 $servidor_2 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_1 $servidor_3 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_1 $servidor_4 color black"
$ns at $temp "$traf_CBR_0_1 stop"
$ns at $temp "$traf_CBR_1_2 stop"
$ns at $temp "$traf_CBR_1_3 stop"
$ns at $temp "$traf_CBR_1_4 stop"
$ns at $temp "$traf_CBR_1_0 stop"
$ns at [expr $temp+0.1] "$traf_CBR_4_3 start"


set temp [expr {rand()+$temp+0.1}]

$ns rtmodel-at $temp up $servidor_2
$ns at [expr $temp+0.0001] "$servidor_2 color black"
$ns at $temp "$traf_CBR_4_3 stop"
```

```
$ns at [expr $temp+0.1] "$traf_CBR_4_2 start"
$ns at [expr $temp+0.1] "$traf_CBR_2_3 start"


set temp [expr {rand()+$temp+0.1}]

$ns rtmodel-at $temp up $servidor_1
$ns at [expr $temp+0.0001] "$servidor_1 color black"
$ns at $temp "$traf_CBR_4_2 stop"
$ns at [expr $temp+0.1] "$traf_CBR_4_1 start"
$ns at [expr $temp+0.1] "$traf_CBR_1_2 start"


set temp [expr {rand()+$temp+0.1}]

$ns rtmodel-at $temp down $servidor_1
$ns at [expr $temp+0.0001] "$servidor_1 color red"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_0 $servidor_1 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_1 $servidor_2 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_1 $servidor_3 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_1 $servidor_4 color black"
$ns at $temp "$traf_CBR_0_1 stop"
$ns at $temp "$traf_CBR_1_2 stop"
$ns at $temp "$traf_CBR_1_3 stop"
$ns at $temp "$traf_CBR_1_4 stop"
$ns at $temp "$traf_CBR_1_0 stop"
$ns at [expr $temp+0.1] "$traf_CBR_4_2 start"


set temp [expr {rand()+$temp+0.1}]

$ns rtmodel-at $temp up $servidor_1
$ns at [expr $temp+0.0001] "$servidor_1 color black"
$ns at $temp "$traf_CBR_4_2 stop"
$ns at [expr $temp+0.1] "$traf_CBR_4_1 start"
$ns at [expr $temp+0.1] "$traf_CBR_1_2 start"


set temp [expr {rand()+$temp+0.1}]

$ns rtmodel-at $temp down $servidor_1
$ns at [expr $temp+0.0001] "$servidor_1 color red"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_0 $servidor_1 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_1 $servidor_2 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_1 $servidor_3 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_1 $servidor_4 color black"
$ns at $temp "$traf_CBR_0_1 stop"
$ns at $temp "$traf_CBR_1_2 stop"
$ns at $temp "$traf_CBR_1_3 stop"
$ns at $temp "$traf_CBR_1_4 stop"
$ns at $temp "$traf_CBR_1_0 stop"
$ns at [expr $temp+0.1] "$traf_CBR_4_2 start"


set temp [expr {rand()+$temp+0.1}]
```

```
$ns rtmodel-at $temp down $servidor_3
$ns at [expr $temp+0.0001] "$servidor_3 color red"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_2 $servidor_3 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_3 $servidor_0 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_3 $servidor_1 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_3 $servidor_4 color black"
$ns at $temp "$traf_CBR_2_3 stop"
$ns at $temp "$traf_CBR_3_4 stop"
$ns at $temp "$traf_CBR_3_0 stop"
$ns at $temp "$traf_CBR_3_1 stop"
$ns at $temp "$traf_CBR_3_2 stop"
$ns at [expr $temp+0.1] "$traf_CBR_2_4 start"



set temp [expr {rand()+$temp+0.1}]

$ns rtmodel-at $temp down $servidor_4
$ns at [expr $temp+0.0001] "$servidor_4 color red"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_3 $servidor_4 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_4 $servidor_0 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_4 $servidor_1 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_4 $servidor_2 color black"
$ns at $temp "$traf_CBR_3_4 stop"
$ns at $temp "$traf_CBR_4_0 stop"
$ns at $temp "$traf_CBR_4_1 stop"
$ns at $temp "$traf_CBR_4_2 stop"
$ns at $temp "$traf_CBR_4_3 stop"


set temp [expr {rand()+$temp+0.1}]

$ns rtmodel-at $temp up $servidor_0
$ns at [expr $temp+0.0001] "$servidor_0 color black"
$ns at [expr $temp+0.1] "$traf_CBR_2_0 start"
$ns at [expr $temp+0.1] "$traf_CBR_0_2 start"



set temp [expr {rand()+$temp+0.1}]

$ns rtmodel-at $temp down $servidor_0
$ns at [expr $temp+0.0001] "$servidor_0 color red"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_4 $servidor_0 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_0 $servidor_1 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_0 $servidor_2 color black"
$ns at [expr $temp+0.0001] "$ns duplex-link-op $servidor_0 $servidor_3 color black"
$ns at $temp "$traf_CBR_4_0 stop"
$ns at $temp "$traf_CBR_0_1 stop"
$ns at $temp "$traf_CBR_0_2 stop"
$ns at $temp "$traf_CBR_0_3 stop"
$ns at $temp "$traf_CBR_0_4 stop"


set temp [expr {rand()+$temp+0.1}]
```

```
   $ns rtmodel-at $temp up $servidor_0
   $ns at [expr $temp+0.0001] "$servidor_0 color black"
   $ns at [expr $temp+0.1] "$traf_CBR_2_0 start"
   $ns at [expr $temp+0.1] "$traf_CBR_0_2 start"



#
=========================================================================
===============================================
# 08 - Fechamento da Simulacao
#
=========================================================================
===============================================

  proc finaliza {} {
   global ns nf tf
   $ns flush-trace                    ;#atualiza o buffer dos arquivos trace
   close $nf                                 ;#fecha o arquivo.nam
   close $tf                                 ;#fecha o arquivo.tr
  # exec nam nam_4-1.nam                     ;#executa a ferramenta nam
   exit 0
  }

#
=========================================================================
===============================================
# 09 - Executa a Simulacao
#
=========================================================================
===============================================
  $ns run                                    ;#inicia a simulacao
```

APÊNDICE C – Algoritmo de diagnóstico em pseudocódigo

```
algoritmo_testa_nodos
{   int id_nodo, id_rodada, qtde_nodo=5;
    string mat [qtde_nodo] [2];

    cont=2;
    id_nodo=0;
    id_rodada=0;

    enquanto (id_nodo < qtde_nodo)
    {
      mat [id_nodo] [0]= '10.1.1.'+cont;
      incrementa cont;
      incrementa id_nodo;
    }

    id_nodo=0;
    cont = 0;

    enquanto (id_nodo < qtde_nodo)
    {
      se (id_rodada = 0)
      {
        ping -c 1 leia(mat[id_nodo+1] [0]);
              se (id_nodo+1 = 0)
              {
                mat[id_nodo+1][1] = 'ok';
              }
              senao
              {
                mat[id_nodo+1][1] = 'falho';
              }
              incrementa id_nodo;
              se (id_nodo = 4)
              {
                id_nodo =0;
                ping -c 1 leia(mat[id_nodo] [0]);
                se (id_nodo = 0)
                {
                  mat[id_nodo][1] = 'ok';
                }
                senao
                {
                  mat[id_nodo][1] = 'falho';
                }
                id_rodada = 1;
              }
      }
            senao     //para rodada igual a 1 (faz atualizacao)
            {
              enquanto (id_nodo < qtde_nodo)
              {
                se (mat[id_nodo][1] = 'falho' )
```

```
                {
                    named0.conf = named.conf
                    #  avisa ao suporte tecnico que servidor caiu;
                    echo "Aviso de pane no Servidor $id_nodo" | mail -s "O sistema de
diagnóstico detectou falha ocorrida no Servidor $id_nodo";
                          }
                senao
                {
                  named.conf = named.conf;
                }
                  incrementa id_nodo;
              }
              id_rodada = 0;
              id_nodo=0;
            }
    }
 }
}
```

APÊNDICE D – *Script* de simulação

```
#!/bin/bash

pktS=0
pktR=0
pktD=0
pktS0=0
pktS1=0
pktS2=0
pktS3=0
pktS4=0
pktR0=0
pktR1=0
pktR2=0
pktR3=0
pktR4=0
pktD0=0
pktD1=0
pktD2=0
pktD3=0
pktD4=0


rm -r trace_Nam trace_Tr
mkdir trace_Nam trace_Tr


clear
echo
==========================================================================
=
echo Rodando Simulações...Aguarde
echo
==========================================================================
=
echo




for exp in  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
31 32 33 34 35;
do

  mkdir trace_Nam/$exp-trace_Nam trace_Tr/$exp-trace_Tr


  rodada=1;
  while [ true ]
  do
```

```
   #Gera valores aleatórios de 1 a 3, soma-se 1 ao resultado para que o valor gerado não
inclua o "0" e vai até o 3
   valor_aleatorio="$(($RANDOM % 3 + 1))"

   ns tcl_$rodada-$valor_aleatorio.tcl      #chama o .tcl



   tam_nodo=$(grep -c ' ' tr_$rodada-$valor_aleatorio.tr);   #pega tamanho vetor/ numero
linhas arquivo
   nodo_src=($(cut -f3 -d ' ' tr_$rodada-$valor_aleatorio.tr));  #carrega vetor com os nodos de
origem
   nodo_dst=($(cut -f4 -d ' ' tr_$rodada-$valor_aleatorio.tr));  #carrega vetor com os nodos de
destino
   evento=($(cut -f1 -d ' ' tr_$rodada-$valor_aleatorio.tr));   #carrega vetor tipo de evento



  #
==============================================================================
=
  # calcula quantidade de pacotes
  #
==============================================================================
=
  echo Analisando TR_$rodada-$valor_aleatorio.tr no Experimento $exp...


  cont=0;
  while [ true ]
   do
    if [ ${evento[$cont]} == - ]; then       #só pacotes enviados (sai da fila)
     pktS=$(($pktS+1));
     if [ ${nodo_src[$cont]} == 0 ]; then
      pktS0=$(($pktS0+1));
     fi
     if [ ${nodo_src[$cont]} == 1 ]; then
      pktS1=$(($pktS1+1));
     fi
     if [ ${nodo_src[$cont]} == 2 ]; then
      pktS2=$(($pktS2+1));
     fi
     if [ ${nodo_src[$cont]} == 3 ]; then
      pktS3=$(($pktS3+1));
     fi
     if [ ${nodo_src[$cont]} == 4 ]; then
      pktS4=$(($pktS4+1));
     fi
    fi
    if [ ${evento[$cont]} == r ]; then       #só pacotes recebidos no nodo
     pktR=$(($pktR+1));
     if [ ${nodo_dst[$cont]} == 0 ]; then
```

```
        pktR0=$(($pktR0+1));
      fi
      if [ ${nodo_dst[$cont]} == 1 ]; then
        pktR1=$(($pktR1+1));
      fi
      if [ ${nodo_dst[$cont]} == 2 ]; then
        pktR2=$(($pktR2+1));
      fi
      if [ ${nodo_dst[$cont]} == 3 ]; then
        pktR3=$(($pktR3+1));
      fi
      if [ ${nodo_dst[$cont]} == 4 ]; then
        pktR4=$(($pktR4+1));
      fi
    fi
    if [ ${evento[$cont]} == d ]; then       #só pacotes descartados
      pktD=$(($pktD+1));
      if [ ${nodo_src[$cont]} == 0 ]; then
        pktD0=$(($pktD0+1));
      fi
      if [ ${nodo_src[$cont]} == 1 ]; then
        pktD1=$(($pktD1+1));
      fi
      if [ ${nodo_src[$cont]} == 2 ]; then
        pktD2=$(($pktD2+1));
      fi
      if [ ${nodo_src[$cont]} == 3 ]; then
        pktD3=$(($pktD3+1));
      fi
      if [ ${nodo_src[$cont]} == 4 ]; then
        pktD4=$(($pktD4+1));
      fi
    fi
    if [ $cont == $(($tam_nodo-1)) ]; then     #qdo ultima linha para o laço
      break;
    fi
    cont=$(($cont+1));
  done

 #
=========================================================================
=
 # grava no arquivo
 #
=========================================================================
=
  echo ==========================================================
>>resul_Experimento-$exp
  echo Resultados da análise de $exp-TR_$rodada-$valor_aleatorio.tr
>>resul_Experimento-$exp
  echo ==========================================================
>>resul_Experimento-$exp

  echo 'Nodo 0:' pkts_S.:$pktS0  pkts_R.:$pktR0  pkts_D.:$pktD0 >>resul_Experimento-
$exp
```

```
   echo 'Nodo 1:' pkts_S.:$pktS1  pkts_R.:$pktR1  pkts_D.:$pktD1 >>resul_Experimento-
$exp
   echo 'Nodo 2:' pkts_S.:$pktS2  pkts_R.:$pktR2  pkts_D.:$pktD2 >>resul_Experimento-
$exp
   echo 'Nodo 3:' pkts_S.:$pktS3  pkts_R.:$pktR3  pkts_D.:$pktD3 >>resul_Experimento-
$exp
   echo 'Nodo 4:' pkts_S.:$pktS4  pkts_R.:$pktR4  pkts_D.:$pktD4 >>resul_Experimento-
$exp

   echo --------------------------------------------------- >>resul_Experimento-$exp
   echo Total pkts_S. na rede:$pktS >>resul_Experimento-$exp
   echo Total pkts_R. na rede:$pktR >>resul_Experimento-$exp
   echo Total pkts_D. na rede:$pktD >>resul_Experimento-$exp
   echo >>resul_Experimento-$exp
   echo TxEntrega_Pacotes=`echo "scale=3; $pktR/$pktS*100" |bc` % >>resul_Experimento-
$exp
   echo =====================================================
>>resul_Experimento-$exp
   echo >>resul_Experimento-$exp
   echo >>resul_Experimento-$exp
   echo >>resul_Experimento-$exp


   pktS=0
   pktR=0
   pktD=0
   pktS0=0
   pktS1=0
   pktS2=0
   pktS3=0
   pktS4=0
   pktR0=0
   pktR1=0
   pktR2=0
   pktR3=0
   pktR4=0
   pktD0=0
   pktD1=0
   pktD2=0
   pktD3=0
   pktD4=0

   mv tr_$rodada-$valor_aleatorio.tr trace_Tr/$exp-trace_Tr/tr_$rodada-$valor_aleatorio.tr
#move arquivos para pasta
   mv nam_$rodada-$valor_aleatorio.nam trace_Nam/$exp-trace_Nam/nam_$rodada-
$valor_aleatorio.nam



   if [ $rodada == 4 ]; then
    echo 'Análise do Experimento' $exp 'concluída.'
    mv resul_Experimento-$exp trace_Tr/resul_Experimento-$exp
    echo ---------------------------------------------------
    echo
    echo
```

```
      break;
    fi
  rodada=$(($rodada+1));
  done
done
```