FD-SENSI - UM DETECTOR DE FALHAS ADAPTATIVO E SUA APLICAÇÃO A UM SISTEMA DISTRIBUÍDO EM LARGA ESCALA

EVERTHON VALADÃO

FD-SENSI - UM DETECTOR DE FALHAS ADAPTATIVO E SUA APLICAÇÃO A UM SISTEMA DISTRIBUÍDO EM LARGA ESCALA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Dorgival Olavo Guedes Neto
Coorientador: Prof. Dr. Ricardo de Oliveira Duarte

Belo Horizonte
Setembro de 2009

Valadão, Everthon
D????p FD-Sensi - Um detector de falhas adaptativo e sua aplicação a um sistema distribuído em larga escala / Everthon Valadão. — Belo Horizonte, 2009
xviii, 103 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Prof. Dr. Dorgival Olavo Guedes Neto
Coorientador: Prof. Dr. Ricardo de Oliveira Duarte

1. Sistemas Distribuídos. 2. Tolerância a Falhas. 3. Detectores de Falhas. I. Título.

CDU ???.?*??.??

[Folha de Aprovação]

Quando a secretaria do Curso fornecer esta folha,
ela deve ser digitalizada e armazenada no disco em formato gráfico.

Se você estiver usando o `pdflatex`,
armazene o arquivo preferencialmente em formato PNG
(o formato JPEG é pior neste caso).

Se você estiver usando o `latex` (não o `pdflatex`),
terá que converter o arquivo gráfico para o formato EPS.

Em seguida, acrescente a opção `approval={`*nome do arquivo*`}`
ao comando `\ppgccufmg`.

Resumo

Os detectores de falhas consistem num componente essencial em qualquer sistema distribuído tolerante a falhas. Um detector de falhas ideal deve adaptar-se a variadas condições de rede e carga do sistema, de maneira a prover informações rápidas e precisas sobre processos falhos aos outros módulos do sistema tolerante a falhas. Este trabalho apresenta a avaliação de um novo algoritmo de detecção de falhas, FD-Sensi, que é capaz de lidar com sistemas distribuídos altamente sobrecarregados e redes de comunicação com grandes variações de latência de mensagens. Foi realizada a avaliação do FD-Sensi tanto em cenários com cargas sintéticas quanto cenários com cargas reais da Internet, utilizando dados coletados em uma centena de nós do PlanetLab. Os dados foram utilizados para comparar seu desempenho com um dos melhores algoritmos de detecção da atualidade, *Adaptive Accrual*. Os resultados mostram que o FD-Sensi teve um desempenho superior ao *Adaptive Accrual*, apresentando uma significativa redução da emissão de falso-positivos com a manutenção de um baixo tempo médio de detecção. O *trace* coletado no PlanetLab poderá ser utilizado na avaliação de novos algoritmos de detecção de falhas e, através de sua análise, este trabalho fornece também as distribuições estatísticas mais apropriadas para a modelagem de atrasos de rede em ambientes globalmente distribuídos. Por fim, propomos uma técnica de aperfeiçoamento para algoritmos de detecção que, baseando-se na correlação entre a carga do nó monitorado e os atrasos percebidos possibilitou-nos melhorar significativamente a precisão e velocidade da detecção de falhas.

Palavras-chave: Sistemas Distribuídos, PlanetLab, Tolerância a Falhas, Detectores de Falhas.

Abstract

The failure detector is an essential component of any distributed dependable system solution. An ideal failure detector must adapt to varying network/system conditions in order to provide fast and accurate information about faulty nodes to other modules of a dependable system. This work presents a new adaptive failure detection algorithm, FD-Sensi, which is able to cope with heavily loaded distributed systems and networks in a wide range of message delay scenarios. We evaluated our failure detector algorithm in an Internet scenario, using data collected in one hundred PlanetLab nodes. The data were used to compare the performance of our algorithm with one of the best failure detection algorithms of the present day, Adaptive Accrual. Our results show FD-Sensi outperformed Adaptive Accrual, presenting a significant reduction in the emission of false-positives with the maintenance of a low average detection time. The trace collected on PlanetLab may be used in the evaluation of new algorithms for failure detection and through its analysis this work also provides the best fitted statistical distributions to model network delays in globally distributed environments. Finally, we propose a technique for improving detection algorithms that, based on the correlation between the resource load of the monitored node and the observed delays allowed us to significantly improve the accuracy and speed of failure detection.

Keywords: Distributed Systems, PlanetLab, Fault Tolerance, Failure Detectors.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Sistemas computacionais consistem numa diversidade de componentes de hardware e software suscetíveis a ocorrência de eventuais falhas, que podem direcioná-los a um comportamento não planejado, potencialmente causando a indisponibilidade de um serviço. Porém, alguns sistemas são projetados para serem tolerantes a falhas, demonstrando um comportamento bem definido na ocorrência de falhas ou mascarando-as para os usuários, mantendo assim o serviço disponível. Esses sistemas são ditos "tolerantes a falhas" e são projetados para prover serviço confiável de maneira contínua mesmo na presença de falhas em alguns de seus componentes. Tais sistemas podem tolerar falhas de software (ex.: falha no projeto das aplicações) e/ou hardware (ex.: falha de processamento, falha da máquina) [Sampaio & Brasileiro, 2001], devendo mascará-las para o usuário ou acusá-las para que a camada de gerenciamento possa tomar uma decisão.

Fidedignidade (*dependability*) é a propriedade de sistemas que apresentam atributos como confiabilidade (*reliability*), disponibilidade (*availability*), seguridade (*safety*), integridade (*integrity*) e manutenibilidade (*maintainability*) [Avizienis et al., 2004]. Mesmo se o sistema for projetado para mascarar os erros, a detecção de falhas ainda contempla um importante papel no sistema distribuído fidedigno, pois é a técnica que permite apontar qual é o componente falho para que o sistema de gerenciamento possa tomar uma decisão [Verissimo & Rodrigues, 2004]. Estas características são fundamentais para que o software siga, por exemplo, o modelo de qualidade estabelecido pela norma ISO 9126 [NBR-13596, 1996] e sejam responsáveis por atuações em ambientes críticos como mercado financeiro, sistemas que controlam aeronaves ou mísseis e naqueles que fornecem suporte à vida humana [Turchetti & Nunes, 2006].

Detectores de falhas são considerados como uma parte crucial para o desenvolvimento de sistemas fidedignos, robustos e auto-reparáveis, consistindo num componente essencial para a tolerância a falhas. Ao prover informações sobre defeitos nos

componentes desses sistemas, permitem a camada de gerenciamento tomar as devidas providências no caso de falhas. Porém, os detectores de falha (FD) são considerados como oráculos não confiáveis. A razão disso é que deve-se considerar que FDs perfeitos não existem em ambientes assíncronos, pois neles há a impossibilidade de distinguir com certeza quando um processo falhou ou quando a rede de comunicação está apenas lenta devido a atrasos causados por congestionamentos, espera em filas de roteadores, dentre outros fatores.

Neste contexto, é possível que um processo tenha falhado mas não se suspeite disso ou que tenha sido considerado suspeito mas não tenha realmente falhado, este último conhecido como falso-positivo. Falso-positivos podem ser muito prejudiciais ao desempenho do sistema, pois cada suspeita de falha pode disparar um custoso protocolo de consenso entre os membros do grupo de detecção, devendo isso ser evitado ao máximo. Ainda, um detector de falhas pode suspeitar corretamente de um processo que tenha falhado, mas levar um longo tempo para apontar a ocorrência da falha. Dentro da gama existente de sistemas distribuídos assíncronos, há uma série de aplicações que possuem limitações temporais, necessitando de detectores de falhas que provejam uma "qualidade de serviço", ou seja, detectar as falhas de maneira rápida e precisa.

1.1 Motivação

A elaboração de sistemas disponíveis, corretos, seguros, escaláveis, persistentes e ubíquos[1] foi apontado pela SBC [2006] como um dos os cinco maiores desafios para a pesquisa em computação no Brasil até 2016. Há inúmeros benefícios advindos da pesquisa nesta área, onde vários deles são decorrentes das vantagens resultantes do desenvolvimento de sistemas confiáveis e seguros, contribuindo para a melhoria da qualidade e a redução dos custos de desenvolvimento e uso de software e sistemas computacionais em geral.

O desenvolvimento de uma técnica de detecção adequada a um sistema distribuído assíncrono que experimente diferentes situações de sobrecarga e variações no atraso das mensagens durante sua execução é uma tarefa desafiadora, pois é extremamente dependente da aplicação e do ambiente que a compreende [Cui et al., 2003; Chandra & Toueg, 1996]. Sendo assim, dado o compromisso entre velocidade e precisão de detecção, o campo de detectores de falhas tem recebido constantes contribuições, visando torná-los mais velozes, precisos e também adaptáveis às condições da rede.

[1] Ubiquidade é a condição de estar em toda parte ao mesmo tempo, ser onipresente. No contexto, significa sistemas larga e esparsamente distribuídos.

Os FDs tradicionais enfrentam problemas ao lidar com sistemas distribuídos de larga escala (redes de longa distância - WANs), considerando-se que suas escolhas foram feitas para redes locais (LANs) e sistemas de escala limitada. Sendo assim, as implementações tradicionais não foram feitas para lidar com um número muito grande de processos, alto nível de perda de mensagens, topologia dinâmica de rede e a imprevisibilidade das WANs. Porém, apesar de constantemente aparecerem novas propostas de detectores de falhas, não é percebida uma larga e fácil utilização dos mecanismos de detecção de falhas e, até onde pudemos verificar, não existe atualmente uma infra-estrutura distribuída de larga escala que proveja este tipo de serviço de detecção de falhas de maneira a conseguir uma maior cobertura, precisão e velocidade de detecção.

Em minha monografia de graduação, sob orientação do Prof. Dr. Ricardo de Oliveira Duarte, propus o algoritmo de detecção de falhas FD-Sensi [Valadão, 2007]. Ele utiliza uma abordagem adaptativa que melhora sua sensibilidade a variações no atraso das mensagens recebidas. O FD-Sensi foi disponibilizado como uma alternativa aos protocolos de detecção de falhas com *timeout* fixo encontrados no *framework* JGroups [Montresor, 2006] de então. Neste trabalho percebemos a oportunidade de ampliar o escopo do FD-Sensi para sistemas distribuídos em larga escala, onde buscamos avaliar e melhorar seu desempenho em WANs.

1.2 Objetivos

O objetivo deste trabalho é implementar, avaliar e melhorar uma versão do algoritmo de detecção de falhas FD-Sensi, otimizando seu desempenho quanto à velocidade/precisão de detecção de falhas para um ambiente largamente distribuído.

O modelo que justifica o trabalho é o PlanetLab [Peterson & Roscoe, 2006], uma plataforma *overlay* globalmente distribuída onde se percebe uma lacuna no provimento de um serviço mais fidedigno de detecção de falhas, no que diz respeito a uma maior velocidade e precisão na detecção das falhas.

1.3 Principais Contribuições

No ambiente distribuído PlanetLab, analisamos e melhoramos o desempenho do FD-Sensi, um novo algoritmo de detecção de falhas adaptativo. Disponibilizamos também um *trace* coletado durante 10 dias em 100 nós do PlanetLab, bem como a caracterização do RTT (*round-trip time*) e análise da assimetria e distribuição dos atrasos de rede.

Através do *trace* coletado e sua análise, este trabalho fornece tanto uma carga real quanto as distribuições estatísticas mais apropriadas para a modelagem de uma carga sintética para o cenário do PlanetLab.

O algoritmo de detecção de falhas FD-Sensi é aplicável a cenários de sistemas distribuídos heterogêneos, especialmente aqueles que apresentem sobrecarga e alta variabilidade na comunicação. Os resultados mostraram, no cenário do PlanetLab, uma significativa redução da emissão de falso-positivos com a manutenção de um baixo tempo médio de detecção.

Propomos uma técnica de aperfeiçoamento para algoritmos de detecção baseada na correlação entre a carga do nó monitorado e os atrasos percebidos. De posse das informações de carga é possível perceber o sobrecarregamento do nó monitorado, que associado aos atrasos permite minimizar falso-positivos. Através desta técnica foi possível melhorar significativamente o desempenho do FD-Sensi em relação à velocidade e à precisão da detecção de falhas. Porém, ela não fica restrita ao FD-Sensi, podendo ser aplicada a outros detectores de falhas que tenham acesso a informações de carga dos nós monitorados.

1.4 Estrutura

Este trabalho está estruturado nos seguintes capítulos: no capítulo 2 nós fornecemos uma revisão da detecção de falhas em sistemas distribuídos, com a importância dos detectores, suas principais características e estratégias de monitoramento, bem como o estado da arte. No capítulo 3 apresentamos o detector adaptativo FD-Sensi, juntamente com uma avaliação de seu desempenho com uma carga sintética. Então, no capítulo 4 introduzimos o PlanetLab como ambiente de testes e fornecemos a análise do *trace* de atrasos de rede nele coletado. No capítulo 5 propomos uma técnica para melhoria de desempenho dos detectores baseada na correlação entre carga do nó e atrasos percebidos, bem como a avaliação de desempenho do FD-Sensi com uma carga real. Finalmente, o capítulo 6 apresenta as conclusões e trabalhos futuros e fecha esta dissertação.

Capítulo 2

Detecção de Falhas em Sistemas Distribuídos

A detecção de falhas está presente no projeto, análise e implementação de vários algoritmos distribuídos tolerantes a falhas que constituem o núcleo do *middleware* de sistemas distribuídos. Sendo assim, detectores de falhas são considerados como uma parte crucial para o desenvolvimento de sistemas fidedignos, robustos e auto-reparáveis, consistindo num componente essencial para a tolerância a falhas. Ao prover informações sobre defeitos nos componentes desses sistemas, permitem a camada de gerenciamento tomar as devidas providências no caso de falhas.

Exemplos de sistemas distribuídos que necessitam ser tolerantes a falhas são aqueles baseados em grupos de processos executando em redes assíncronas e não-confiáveis, como por exemplo, protocolos de gerenciamento de grupo, *clusters* e *grids* de computadores, dentre outros. Assim sendo, grupos de processos em aplicações e serviços distribuídos confiam completamente nos detectores de falhas para detectar falhas em processos, preferencialmente da maneira mais rápida, precisa e escalável possível (mesmo na presença de entrega não-confiável de mensagens). Portanto deve-se considerar num detector os requisitos específicos da aplicação, comumente a rápida detecção de falhas e precisão da detecção de falhas, mas pode haver restrições quanto a largura de banda e/ou carga computacional. Gupta et al. [2001] considera que a habilidade de detectar falhas em processos de maneira completa e eficiente, mesmo na presença de entrega não-confiável de mensagens, na arbitrariedade das falhas e na recuperação de processos, pode ter um grande impacto no desempenho desses sistemas.

2.1 Modelo de Sistema Distribuído Assíncrono

Sistemas distribuídos podem ser classificados como síncronos e assíncronos, dependendo da camada subjacente de comunicação. Segundo Cristian & Fetzer [1999], num sistema síncrono, uma mensagem enviada entre dois processos corretos é recebida numa quantidade de tempo finita, ou seja, a probabilidade da mensagem não ser recebida e processada a tempo é desprezível. Já um sistema assíncrono não provê esta garantia, não havendo um limite conhecido para a frequência com a qual as falhas possam ocorrer, ou seja, estes sistemas não proveem garantias temporais para o funcionamento das aplicações. Um modelo formal de um sistema assíncrono é apresentado em Cristian & Fetzer [1999], onde podem ser encontrados detalhes da caracterização de uma especificação de serviços, relógios lógicos, semântica de falhas, garantias de sincronismo e suposições acerca dos processos.

Modelo de Processo Considera-se um sistema consistindo de um conjunto finito de n processos, no qual um processo pode falhar por queda ou parada prematura. Um processo correto é aquele que se comporta de acordo com a sua especificação, processando as entradas recebidas e entregando as saídas corretamente. No modelo considerado, o processo se comporta corretamente (de acordo com sua especificação) até que ele (possivelmente) falhe e não há restrições de tempo para que um processo correto execute uma tarefa. Consideramos um modelo de falha *crash-stop* (*fail-stop*), onde um processo finaliza e não executa mais nenhuma operação.

Modelo de Comunicação Processos comunicam e sincronizam-se pela troca de mensagens através de enlaces de comunicação. Cada par de processos é conectado por um enlace, que pode ser de dois tipos:

- O enlace conectando o processo p_i ao p_j é *confiável* se ele não cria ou duplica mensagens e cada mensagem enviada por p_i para p_j é eventualmente recebida por p_j (caso ele não falhe).

- O enlace conectando o processo p_i ao p_j é não-confiável (*fair lossy*) se, apesar de não criar ou duplicar mensagens ele pode perdê-las. Porém, se p_i envia uma quantidade ilimitada de mensagens para p_j que por sua vez permanece na ação de recebê-las indefinidamente, então p_j receberá uma quantidade infinita de mensagens de p_i.

Modelo Computacional Reynal [2005] considera dois tipos de modelo computacional assíncrono:

- O modelo FLP que considera processos propensos a falhas e enlaces confiáveis.
- O modelo FLL que considera processos propensos a falhas e enlaces não-confiáveis.

2.2 A Importância dos Detectores de Falhas

Sistemas distribuídos assíncronos podem ser caracterizados por não possuírem um limite de tempo para um processo executar uma determinada computação ou para uma mensagem trafegar até o destinatário da mesma. Tais sistemas apresentam certas vantagens ao facilitar a compreensão e prova de corretude das aplicações para eles planejadas, por não dependerem de nenhuma premissa temporal [Reynal, 2005]. Porém tais vantagens são inúteis caso possam ocorrer falhas nos processos ou nós. Considere o caso onde um nó *pode* falhar. Nesse cenário, o problema consiste num determinado nó decidir se um outro nó falhou ou não. A possibilidade de falha num nó juntamente com a assincronia da comunicação leva a uma situação onde não há um modo seguro de saber se um nó falhou ou não. Observe que ao assumir-se que o tal nó falhou, pode-se estar enganado pois talvez a mensagem esperada está apenas atrasada. Por outro lado, se o nó realmente tiver falhado mas, para evitar eventuais enganos, permanece-se pacientemente esperando pela mensagem. Esta espera vai se prolongar eternamente e a propriedade de *liveness*[1] da aplicação não será satisfeita.

Muitos problemas de concordância entre processos distribuídos (como por exemplo o problema do Consenso) não podem ser resolvidos deterministicamente em sistemas assíncronos se ao menos um processo puder falhar [Fischer et al., 1985; Lynch, 1989] e, conforme observa-se no cenário previamente descrito, esta impossibilidade decorre do fato de que nesses sistemas um processo falho não pode ser diferenciado de um processo lento. Felizmente, tal impossibilidade não se mantém se ao sistema for adicionado um oráculo distribuído não-confiável para detecção de falhas, mais conhecido como detector de falhas, noção esta introduzida e investigada por Chandra & Toueg [1996]. Tais oráculos não alteram o padrão de ocorrência das falhas, provendo apenas uma adivinhação ou palpite acerca de quando uma falha poderá ocorrer. Porém, como o oráculo não é confiável, é possível que um processo tenha falhado mas não seja suspeitado bem como um processo tenha sido suspeitado sem ter realmente falhado.

[1] *liveness* é a propriedade de uma aplicação continuar seu propósito de existência

Ainda, um oráculo pode mudar de ideia ao parar de suspeitar de um processo ao qual tenha previamente suspeitado.

Segundo Sampaio & Brasileiro [2005], muito do sucesso da abordagem de oráculos de detecção de falhas para projetar e implementar protocolos distribuídos tolerantes a falha é baseado no fato de que esta abstração permite uma separação muito elegante de conceitos. Os oráculos são definidos em termos de propriedades abstratas, de maneira que os protocolos possam ser projetados considerando tais propriedades mas ignorando sua implementação prática. O sucesso de tal abstração fez com que oráculos também fossem utilizados em outras áreas, como por exemplo no roteamento de mensagens em redes tolerantes a interrupção [Mota et al., 2009]. Observe que apesar da informação provida pelo oráculo de um detector de falhas pode não ser confiável, ela é precisa o suficiente para permitir soluções determinísticas para vários problemas de concordância [Brasileiro & Sampaio, 2001], conforme veremos a seguir.

2.2.1 Resolvendo o Problema do Consenso

O problema do consenso é um paradigma de problemas de concordância. Ele surge quando processos necessitam concordar em algo, como por exemplo, numa ação comum a ser executada em conjunto ou numa mesma decisão a ser tomada, dentre outros. Um exemplo clássico de consenso é o *broadcast* atômico, um problema que aparece como uma camada básica em vários esquemas de tolerância a falhas via replicação. O *broadcast* atômico requer que os processos corretos entreguem o mesmo conjunto de mensagens na mesma ordem. Observe que consiste ao mesmo tempo de um problema de comunicação (pois todos os processos têm de entregar o mesmo conjunto de mensagens) e um problema de consenso (pois eles devem entregá-las na mesma ordem). No problema do consenso, cada processo correto p_i propõe um valor v_i e todos os processos corretos têm de decidir por um valor v, em relação ao conjunto de valores propostos [Reynal, 2005].

Chandra & Toueg [1996] definem o problema do consenso nas três seguintes propriedades:

- Finalização: Cada processo eventualmente decide por algum valor.
- Validade: Se um processo decide pelo valor v, então v foi proposto por algum processo.
- Concordância: Dois processos corretos não decidem diferentemente.

Observe que a propriedade de concordância se aplica somente aos processos corretos, portanto é possível que um processo decida-se por um valor distinto logo antes de falhar. O *Consenso Uniforme* impede tal possibilidade, pois possui as mesmas propriedades de Finalização e Validade, mas acrescidas da seguinte propriedade:

- Concordância Uniforme: Não existem dois processos (corretos ou não) que decidam diferentemente.

Conforme observado por Fischer et al. [1985], o problema do consenso não pode ser resolvido em sistemas assíncronos suscetíveis a falha de um único processo sequer. E para contornar esta impossibilidade o conceito dos detectores de falhas foi proposto por Chandra & Toueg [1996]. Para ser útil, uma classe de detectores de falhas tem que satisfazer certas propriedades e para o detector ser implementável elas devem ser tão fracas quanto possível, desde que ainda permitam que o problema de interesse seja resolvido. Na prática, nenhuma das classes de detectores de falhas propostos por Chandra & Toueg [1996] pode ser implementada em um ambiente puramente assíncrono sem a possibilidade de se cometerem enganos [Nunes & Jansch-Pôrto, 2002]. Assim, apesar dos detectores de falhas serem uma poderosa abstração que simplifica o projeto de protocolos distribuídos, é impossível construir em sistemas assíncronos tal dispositivo, de maneira que uma solução prática necessitará de suposições adicionais. Em Brasileiro & Sampaio [2001] é proposta uma abordagem prática para validar o projeto de protocolos distribuídos tolerantes a falhas para sistemas assíncronos, onde tais suposições adicionais (ex.: serviço de *broadcast* confiável) foram validadas no contexto do problema do consenso.

Para garantir a consistência da aplicação, a maioria das implementações de detectores de falhas primeiro garante deterministicamente a propriedade de completude, de maneira que a precisão passa a ser garantida apenas probabilisticamente. Sendo assim, das diversas classes de detectores por eles propostas, a classe mais fraca para resolver o consenso inclui todos os detectores de falhas que satisfaçam às seguintes propriedades:

- **Completude Forte**: Eventualmente, cada processo que falhe é permanentemente suspeitado por cada outro processo correto.

- **Precisão Eventualmente Fraca**: Exite um momento após o qual algum processo correto nunca será suspeitado pelos processos corretos.

Reynal [2005] apresenta um protocolo de consenso baseado no quórum entre os detectores de falhas [Mostéfaoui & Reynal, 1999], que requer uma maioria de processos corretos (por ser um requisito dos protocolos indulgentes, propostos por Guerraoui

[2000]), considerando-se o modelo FLP (vide seção 2.1). A propriedade de completude forte é utilizada para mostrar que o protocolo nunca bloqueia e a propriedade de precisão eventualmente fraca é utilizada para garantir a finalização, onde a maioridade dos processos corretos é utilizada para prover o consenso de concordância.

2.2.1.1 Resolvendo o Problema da Consistência Interativa

Este problema foi inicialmente introduzido no contexto de sistemas síncronos onde alguns processos podem se comportar de maneira Bizantina [Pease et al., 1980]. Dizer que um processo se comporta de maneira Bizantina significa que ele pode falhar arbitrariamente. Como os processos podem vir a falhar, um processo defeituoso pode enviar valores diferentes a processos distintos, ainda que esses valores sejam referentes à mesma informação. Para alcançar um *Consenso Bizantino*, os processos não defeituosos devem concordar sobre os valores corretos, onde cada processo deve tomar uma decisão baseada nos valores recebidos dos outros processos e todos os processos não defeituosos devem decidir pelo mesmo valor.

Observe que este problema é mais difícil que o consenso simples pois os processos devem concordar não num valor proposto, mas num vetor de valores propostos. Neste caso, cada processo p_i propõe um valor v_i e tem que decidir por um vetor de valores propostos, de maneira que as seguintes propriedades sejam satisfeitas (considere a versão uniforme do problema):

- Finalização: Cada processo eventualmente decide um vetor.
- Validade: Cada vetor decidido D é tal que D[i] $\in \{v_i, \perp\}$ e é v_i se p_i não falhou.
- Concordância: Dois processos não decidem diferentemente.

A classe de detectores de falhas mais fraca que permite o problema de consistência interativa ser resolvido é a classe dos detectores perfeitos, denotada por P, que contém os detectores de falhas que satisfazem as seguintes propriedades:

- **Completude Forte**: Eventualmente, cada processo que falhe é permanentemente suspeitado por cada outro processo correto.
- **Precisão Forte**: Nenhum processo é suspeitado antes que ele falhe, ou seja, não são cometidos enganos (falso-positivos).

Reynal [2005] apresenta um protocolo que provê a solução para o problema da consistência interativa através da implementação de um detector de falhas perfeito, considerando-se o modelo FLP.

2.2.2 Resolvendo o Problema do *Commit* Atômico Não-Blocante

Originado da área de banco de dados, o problema do *Commit* Atômico Não-Blocante (*Non-Blocking Atomic Commit*, NBAC) é um dos mais antigos problemas de concordância encontrado em computação distribuída. O mecanismo funciona da seguinte maneira: cada processo emite um voto, que pode ser "sim"ou "não"e de acordo com o conjunto de votos e o fato de que alguns processos possivelmente falharam, os processos não falhos devem decidir em um valor único: *commit* ou *abort*. Reynal [2005] define o problema nas seguintes propriedades:

- Finalização: Cada processo correto eventualmente decide-se.
- Validade: Um valor decidido é *commit* ou *abort*, onde:
  - Justificação: Se um processo decide por *commit*, todos os processos devem ter votado "sim".
  - Obrigação: Se todos os processos votam "sim"e não houve falha de processo, então o valor da decisão é *commit*.
- Concordância: Dois processos não decidem diferentemente.

Sendo assim, a maior diferença entre a especificação do consenso e do NBAC é que o último explicitamente menciona falhas de processos ocorrendo durante a execução deste protocolo de concordância. Para resolver o problema NBAC no modelo FLP, é necessário que este modelo seja enriquecido com os detectores de falhas propostos por Guerraoui [2002]; Guerraoui & Kouznetsov [2002]. Há ainda outros trabalhos que buscam resolver este problema, de maneira rápida e/ou distribuída, a saber [Dutta et al., 2004; Greve & Narzul, 2005; Wang & Buehrer, 2008].

2.2.3 Implementando Comunicação Quiescente

Considere o problema de construir um enlace confiável em cima de um enlace não-confiável (*fair lossy*). Se os dois processos comunicantes não falham, mecanismos básicos como retransmissões e reconhecimento (*ack*) permitem que este problema seja resolvido. Este protocolo é quiescente[2] no sentido que após algum tempo, nenhum processo envia ou recebe mensagens de controle relacionadas à transmissão de uma determinada mensagem. Agora considere que os processos podem falhar: é possível

[2]quieto, dormente, que permanece em repouso

que o processo receptor falhe antes de receber a mensagem, de maneira que o processo emissor vai consequentemente enviar cópias da mensagem indefinidamente, de maneira que este protocolo não pode mais ser considerado quiescente [Reynal, 2005]. Aguilera et al. [2000] demonstram que o problema da comunicação quiescente não pode ser resolvido em um modelo FFL puro, devendo ele ser enriquecido com um detector de falhas apropriado (observe que para parar de retransmitir uma mensagem, o emissor deve saber se o destinatário efetivamente falhou). Os autores mostram ainda que a classe mais fraca para resolver este problema é a classe dos detectores de falhas eventualmente perfeitos que, após um tempo desconhecido mas finito, passam a suspeitar de todos os processos falhos (e somente eles). Porém, tal detector de falhas não pode ser implementado no modelo FFL, fato pelo qual propuseram uma outra classe de detectores de falha implementável neste modelo: os detectores de falha do tipo *heartbeat* [Aguilera et al., 1997].

2.3 Os Detectores de Falhas

Nesta seção serão apresentadas as principais estratégias de monitoramento utilizadas por detectores de falhas e as propriedades que definem sua qualidade de detecção.

2.3.1 Estratégias de Monitoramento

As principais estratégias de monitoramento utilizadas por detectores de falhas são a *push* e a *pull*. Numa abordagem *push* o processo monitorado toma um papel mais ativo, enquanto que numa abordagem *pull*, ele adota um papel mais passivo [Chen et al., 2002]. Segundo Satzger et al. [2007], detectores de falhas que utilizem o paradigma *push* possuem alguns benefícios se comparados àqueles que utilizem uma abordagem *pull*: necessitam de apenas a metade das mensagens para uma qualidade de detecção equivalente.

2.3.1.1 *Push*

Técnicas de detecção de falhas em sistemas distribuídos popularmente seguem uma estratégia *push*, sendo baseadas na troca de mensagens periódicas, conhecidas como *heartbeats*, utilizadas para suspeitar acerca de falhas em processos. O mecanismo funciona da seguinte maneira: seja p o processo monitorado e q o processo detector de falhas. O processo p envia periodicamente um *heartbeat* ("eu estou vivo.") para o processo q. Quando q recebe um *heartbeat* ele confia em p por um certo período de tempo, porém caso novas mensagens não cheguem até expirar este *timeout*, q assume

que ocorreu uma falha do processo p. Naturalmente, o *timeout* deve ter um valor maior que o intervalo de *heartbeat*.

2.3.1.2 *Pull*

Esta estratégia é também baseada na troca de mensagens periódicas para suspeitar acerca de falhas em processos, porém estas mensagens são do tipo *query-response*, também conhecidas como *ping*. O mecanismo funciona da seguinte maneira: seja p o processo monitorado e q o processo detector de falhas. O processo q envia periodicamente uma *query* ("você ainda está vivo?") para o processo p, que deve responder ("eu estou vivo!") em tempo hábil, porém, caso tal resposta não chegue num certo *timeout*, q assume que ocorreu uma falha do processo p.

2.3.2 Qualidade do Serviço de Detecção

Um detector de falhas pode ser basicamente definido por duas características, a saber: *completude* é a garantia de que a falha num membro do grupo é eventualmente detectada por cada outro membro não-falho, enquanto *eficiência* significa que falhas são detectadas rápida e precisamente, sem que se cometam muitos enganos. Como exemplo de aplicação que requer rapidez na detecção podemos citar aquelas que confiam num único ou mesmo poucos computadores centrais para agregar as informações de detecção de falhas ao longo do sistema. Estes computadores são responsáveis por disseminar tais informações ao longo de todo o sistema e portanto, a eficiência na detecção de uma falha depende do tempo em que a falha é inicialmente detectada por um membro não-falho. Conforme observado por Gupta et al. [2001], mesmo na falta de um servidor central, a notificação de uma falha é tipicamente comunicada pelo primeiro membro a detectá-la a todo o grupo via *broadcast* (possivelmente não-confiável). Portanto, mesmo sendo importante alcançar completude, uma eficiente detecção de falhas é mais comumente relacionada com o tempo da primeira detecção da falha.

O primeiro trabalho teórico a analisar as propriedades dos detectores de falhas foi Chandra & Toueg [1996], no qual os autores demonstram porque é impossível para um algoritmo de detecção de falhas deterministicamente alcançar ambas as características de completude a precisão estando numa rede assíncrona e não-confiável. Como consequência, a maioria das aplicações distribuídas têm optado por contornar esta impossibilidade ao confiar em algoritmos de detecção de falhas que garantam completude deterministicamente enquanto alcançam eficiência apenas probabilisticamente, conforme observado por Gupta et al. [2001]. Esta área da literatura tem tratado os detectores de falhas como oráculos utilizados resolver o problema do Consenso Distri-

buído [Fischer et al., 1985], que não pode ser resolvido nos modelos gerais de redes assíncronas.

A caracterização das propriedades dos detectores de falhas provida em Chandra & Toueg [1996] prevê dois tipos de completude e quatro tipos de precisão. Reynal [2005] resume esta caracterização nas seguintes propriedades principais para detectores de falhas:

- **Completude** (Forte/Fraca): falhas em qualquer membro do grupo são detectadas por (todos/alguns) membros não-falhos.
- **Precisão** Forte: nenhum membro não-falho do grupo será declarado falho por qualquer outro membro não-falho do grupo.

Uma precisão fraca admitiria que alguns poucos membros corretos sejam considerados falhos, desde que isso não afete a maioria necessária para o problema abordado. Porém, a maioria das abordagens concentra-se na classe de detectores de falhas eventualmente perfeitos, por ser suficiente para resolver o problema do consenso dadas suas propriedades de *completude forte*, onde há um momento a partir do qual cada processo que tenha falhado é permanentemente suspeitado por todos os outros processos corretos; e *precisão eventualmente forte*, onde há um momento a partir do qual processos corretos não são suspeitados por qualquer outro processo correto [Hayashibara et al., 2004].

2.3.2.1 Compromisso entre Velocidade X Precisão

Existe um compromisso inerente entre a qualidade do serviço prestado por um detector de falhas mais conservativo, que reduz o risco de falsas suspeitas acerca de um processo e um detector de falhas mais agressivo, que rapidamente detecta a ocorrência de uma falha real. Naturalmente, há uma faixa contínua de escolhas válidas entre estes dois extremos, sendo que aquilo que define a escolha apropriada está fortemente relacionado com os requisitos da aplicação.

2.3.2.2 Considerações

Em redes assíncronas, tal qual outros protocolos distribuídos baseados em tempo, grande parte dos detectores de falhas baseia-se em *timeouts* para garantir a terminação sob atrasos de comunicação sem limite definido. Consequentemente, esta abordagem faz com que a qualidade do serviço de detecção de falhas seja dependente da escolha do *timeout* correto [Nunes & Jansch-Porto, 2004]. Por outro lado, o intervalo de envio

de *heartbeats* é um fator que contribui para o tempo de detecção, porém ao contrário do senso comum, este intervalo não é prioritariamente determinado pelos requisitos de qualidade de serviço na detecção (QoS), sendo mais determinado pelas características do sistema sobrejacente. O tempo de detecção é determinado por três parâmetros: o intervalo de checagem, a latência de transmissão da rede e a margem de segurança α, adicionada no *timeout*. Se por um lado o intervalo de checagem é muito menor que a latência, reduzi-lo gera pouco efeito no tempo de detecção, pois o tempo de detecção não pode ser menor que o tempo de transmissão. Assim, reduzindo o intervalo de checagem fará com que se gere mais tráfego e mais atividade na pilha de rede, podendo aumentar a latência de transmissão. Por outro lado, se o intervalo de checagem é muito maior que a latência, então ele vai determinar majoritariamente o tempo de detecção. Aumentá-lo ainda mais vai aumentar o tempo de detecção, mas não diminuirá significativamente a carga já baixa da rede. Sendo assim, segundo Hayashibara et al. [2004] qualquer valor razoável para o intervalo de checagem deve ser aproximadamente igual à média da latência de transmissão da rede, onde a exceção é quando é determinado um limite superior para o uso aceitável da largura de banda da rede para mensagens de controle.

2.3.3 Características dos Detectores de Falhas

A seguir, serão apresentadas as principais características dos detectores de falhas, sendo elas sua flexibilidade, adaptabilidade, economia, escalabilidade e organização.

2.3.3.1 Flexibilidade

Se um detector de falhas não é flexível, ele disponibiliza como saída uma suspeita booleana de falhas (correto ou faltoso), ao passo que um detector flexível responderá com um nível contínuo de suspeita de falha. Portanto, a flexibilidade existe quando é possibilitado à aplicação configurar um limiar para interpretação da saída do detector de falhas, de maneira que atenda aos seus requisitos de QoS: um baixo limiar é propenso a gerar muitas suspeitas errôneas mas garante uma rápida detecção no evento de uma falha real, por outro lado, um alto limiar gerará menos falso-positivos mas necessitará de um tempo maior para detectar falhas reais. Ao configurar um limiar apropriado, as aplicações podem disparar suspeitas e desempenhar as ações apropriadas. É necessário observar que o compromisso que o limiar de suspeita define é dependente da natureza da ação a ser disparada, bem como seus custos em termos de desempenho e utilização de recursos.

Resposta Booleana Conceitualmente, a implementação dos detectores de falhas no lado receptor pode ser decomposta em três partes básicas: *monitoramento* no qual o detector coleta informação de outros processos; *interpretação* na qual a informação colhida é utilizada para decidir se um processo falhou ou não; e *ação* na qual são tomadas medidas em resposta às suspeitas. Dessas partes, normalmente a última é realizada dentro das aplicações. Como os detectores tradicionais provêm apenas interpretações booleanas (confiança *vs.* suspeita), eles não provêem muita flexibilidade de ação para as aplicações. Além disso, têm-se sugerido que os detectores de falhas devam ser providos como uma forma de serviço genérico, similar ao *lookup* de endereço IP ou sincronização de tempo [Reynal, 2005]. Porém, as implementações tradicionais não têm tido êxito neste quesito por não terem sido projetadas para satisfazer aos diversos requisitos das aplicações simultaneamente.

Resposta Numa Escala Contínua (*Accrual*) Hayashibara [2004] propôs uma nova abstração para detectores de falhas, chamada *accrual*, que enfatiza a flexibilidade: ao invés de prover informação de maneira booleana, propõe-se que a saída do detector seja um nível de suspeita numa escala contínua. A grosso modo, este valor captura o grau de confiança de que um dado processo tenha falhado: quanto maior o valor, maior é a chance de que o processo monitorado tenha falhado. Se o processo verdadeiramente tiver falhado, o valor de saída do detector é garantido de ir-se acumulando ao longo do tempo e tender ao infinito, donde vem o nome deste tipo de detector.

Para melhor ilustrar a vantagem desta abordagem, considere o simples exemplo de uma aplicação distribuída na qual um dos processos é nomeado mestre enquanto todos os outros desempenham o papel de escravos. O mestre possui uma lista de trabalhos que devem ser computados e mantém uma lista dos processos escravos disponíveis. Enquanto houver trabalhos na lista, o mestre envia-os aos escravos ociosos e coleta seus resultados após terem sido computados. Considere agora que alguns dos processos escravos possam ter falhado (e por questões de simplicidade, considere que o mestre nunca falha). Ao falhar um determinado processo escravo, o mestre deve ser capaz de identificá-lo como tal e tomar as devidas ações corretivas, do contrário o sistema pode ficar paralisado. A utilização de um detector de falhas *accrual* poderia ser feita da seguinte maneira: quando o nível de confiança atingir um determinado limiar inferior, o mestre sinaliza o processo escravo em questão e temporariamente para de enviar novos trabalhos para ele. Então, quando atingir um limiar intermediário, o mestre cancela todas as computações incompletas que estavam executando neste escravo e reenvia-as a algum outro processo escravo. Por fim, quando atingir um limiar superior, confiança de que o processo escravo em questão falhou é alta e portanto o mestre remove-o da lista

de processos escravos disponíveis e libera os recursos correspondentes. Segundo Défago et al. [2005], caso fosse utilizado um detector booleano convencional, a implementação de tal comportamento preventivo seria mais complexa e desafiadora. Como a saída é a suspeita ou não de falha de um processo, não haveria uma maneira da aplicação tomar as medidas preventivas definidas para os limites inferiores e intermediários citados acima.

2.3.3.2 Adaptabilidade

É importante observar que é difícil estimar o *timeout* ótimo a ser utilizado pelos detectores de falhas. Isso é em grande parte devido a instabilidades da rede de comunicação, tais como perdas e atrasos na transmissão das mensagens e também às próprias variações nestes atrasos (*jitter*), dentre outros fatores. Há portanto um compromisso, onde se o *timeout* é curto, falhas são detectadas mais rapidamente porém com uma probabilidade maior de falso-positivos. Inversamente, se o *timeout* é longo, falso-positivos tornam-se menos frequentes, mas ao custo de um tempo de detecção maior [Hayashibara et al., 2004]. Segundo Nunes & Jansch-Porto [2004], detectores adaptativos adaptam dinamicamente seu *timeout* ao comportamento do atraso de comunicação de acordo com uma margem de segurança, visando melhorar a qualidade do serviço.

***Timeout* Fixo** Nas implementações tradicionais o *timeout* para a detecção de falha é fixo, portanto esses detectores de falhas são incapazes de se adaptar a mudanças na condição do sistema. Deve-se considerar que em sistemas reais as condições da rede podem variar muito ao longo do tempo [Nunes & Jansch-Pôrto, 2002], especialmente naqueles esparsamente distribuídos.

***Timeout* Adaptativo** Na prática, é do senso comum que os sistemas estão sujeitos a variações entre longos períodos de instabilidade e estabilidade. Isto pode levar às situações onde um detector de falhas pode repetidamente remover e adicionar um dado processo de sua lista de processos suspeitados. Para melhor lidar com tais situações, os detectores de falhas adaptativos são capazes de lidar com mudanças nas condições da rede, modificando dinamicamente seu *timeout*. Pode-se ajustar o *timeout* de acordo com o máximo atraso observado, repetir o último atraso observado ou realizar uma estimativa acerca do próximo atraso acrescida de uma margem de segurança [Nunes & Jansch-Porto, 2004]. Isto pode ser alcançado ao observar no histórico das mensagens recebidas sua distribuição temporal e estatísticas de atrasos e perdas, de maneira a viabilizar estimativas para os tempos de chegada futuros.

2.3.3.3 Economia

Os detectores de falhas devem ser econômicos quanto ao envio de mensagens, de maneira a diminuir seu impacto no consumo de recursos necessários aos outros sistemas. Abaixo são listados alguns esquemas de envio de mensagens, onde pode-se enviar as mensagens de monitoração a todos ou somente alguns nós, bem como aproveitar as mensagens da aplicação para uma maior economia.

Mensagens de Todos pra Todos Esquemas tradicionais de *heartbeats* enviam mensagens num esquema todos-para-todos, o que pode gerar uma quantidade imensa de mensagens com o aumento do tamanho da quantidade de processos no grupo de detecção. Dada a ordem quadrática deste esquema de envio, ele não escalaria para um grupo com centenas ou milhares de processos, dependendo do intervalo de comunicação entre eles.

Mensagens de Todos pra Alguns (*Randomized*) Neste esquema de envio de mensagens, a cada iteração envia-se *heartbeats* somente a uma quantidade reduzida de processos aleatoriamente escolhidos. Normalmente esta estratégia é utilizada em conjunto com um esquema colaborativo aumentando assim sua escalabilidade, como será visto adiante.

Mensagens para Alguns Selecionados (*Overlay*) Pode-se aproveitar a localidade de rede ao olhar para a estrutura dos domínios e subdomínios da Internet através do IP do processo, de maneira que se diminua a latência entre nós e a maioria das mensagens seja trocada dentro da sub-rede, poucas entre sub-redes e menos ainda entre subdomínios. Desta maneira ele pode escalar bem, pois o número de mensagens pode ser mantido baixo independentemente da topologia da rede e o número de mensagens num domínio depende unicamente do número de subdomínios. Pode-se construir uma topologia que selecione os nós com menor latência e/ou maior confiabilidade, explorando assim a localidade de rede. Ainda, de acordo com a necessidade da aplicação é possível configurar os detectores para minimizar a latência de detecção ou a utilização de banda, conforme analisado por So & Sirer [2007].

***Piggyback* de Mensagens (*Lazy*)** Quando o módulo de detecção de falhas faz parte da aplicação, é possível interceptar mensagens da pilha de protocolos superiores e inserir nelas uma camada de detecção contendo um *heartbeat*. Desta maneira este tipo de detector aproveita as mensagens da aplicação para monitorar os processos sempre que possível, economizando no envio de mensagens.

2.3.3.4 Escalabilidade

Os detectores de falhas de um sistema distribuído composto por vários processos podem colaborar ou não entre si, característica essa que determina sua maior ou menor escalabilidade:

- **Não-colaborativo (*single-node*)**: A detecção de falhas é realizada entre cada par de processos apenas com as informações locais, ou seja, não há colaboração entre detectores. É retornada a suspeita ou não de um processo. Como se suspeita individualmente do processo monitorado, é necessária uma quantidade maior de mensagens trocadas entre os processos, o que reduz sua escalabilidade.

- **Colaborativo (*multi-node*)**: A detecção de falhas é realizada de maneira a integrar mais os processos, ou seja, colaborativamente, onde os detectores se beneficiam da troca de informações para buscar atingir uma melhor qualidade de detecção. Neste caso, os detectores retornam uma lista de processos dos quais suspeitam terem falhado. Zhuang et al. [2005] apresentam um estudo com os benefícios da troca de informações de monitoramento entre os algoritmos de detecção nos nós.

No modelo colaborativo há várias maneiras de compartilhar as informações sobre os processos, a saber:

Disseminação Epidêmica (*Gossip*) São baseadas na informação de que rumores/doenças se propagam muito eficientemente num sistema, mesmo que ele seja grande. Um processo escolhe outro processo de maneira aleatória, com o qual ele trocará informações sobre processos suspeitos e é garantido que as suspeitas sejam propagadas para todo o sistema com uma certa probabilidade. Como este tipo de abordagem ignora completamente a topologia do substrato, mudanças nesta topologia não afetam seu desempenho (apresentando um bom dinamismo pois a rota de propagação das informações não é estática), uma boa escalabilidade (porém não funciona muito bem quando muitos processos finalizam ou a rede é particionada). Detectores epidêmicos podem utilizar uma abordagem básica ou apresentar uma estrutura multi-nível para melhorar a escalabilidade, podendo também explorar a localidade de rede para melhorar a latência.

Disseminação Hierárquica São baseadas numa hierarquia multi-nível para manter local parte do tráfego gerado pela detecção. Um detector de falhas é responsável por

monitorar todos os processos que estejam no mesmo nível e as informações são propagadas pela hierarquia. Assim, geram menos mensagens de controle, pois as informações são propagadas via estruturas do tipo árvore ao invés de um grafo completo, podendo ainda se aproveitar da topologia física do sistema. Tipicamente apresenta uma boa escalabilidade porém não têm dinamismo.

2.3.3.5 Organização

Nos esquemas de colaboração entre os nós é possível organizá-los de diversas maneiras, de maneira a buscar uma maior abrangência e desempenho na detecção de falhas, podendo ser explorada a localidade de rede.

Group Membership *Group Membership* ou Composição de Grupo é uma abordagem popular para garantir tolerância a falhas em aplicações distribuídas, na qual é realizado o acompanhamento de quais processos pertencem à computação distribuída e quais não pertencem [Chockler et al., 2001]. Normalmente é necessário excluir processos que tenham falhado ou sofrido um particionamento na rede. Segundo Hayashibara et al. [2004], a Composição de Grupo pode também ser vista como um mecanismo de detecção de falhas de alto-nível, que provê informação consistente sobre suspeitas e falhas. Uma variação interessante do *Group Membership* chamada *Fuzzy* foi proposta por Friedman [2003], onde um nível de *fuzziness* deve ser associado a cada processo para determinar o quão um processo pertence a um grupo. É também proposto um detector de falhas *fuzzy* [Friedman et al., 2005] que retorna um valor inteiro e se utiliza de dois limiares para definir três níveis de desconfiança: confiável, *fuzzy* (indistinto) e suspeito, porém não são dados detalhes da arquitetura ou implementação do mesmo.

Overlay P2P Explorando Localidade de Rede Considere que numa arquitetura mais centralizada, caso haja um interesse crescente na utilização do serviço de detecção de maneira a extrapolar os limites de hardware, com o crescimento do interesse pelo serviço a qualidade da detecção poderia ser fatalmente comprometida, considerando-se que o número fixo de detectores não conseguiria mais atender a demanda de usuários. A vantagem de se utilizar uma arquitetura par-a-par (P2P) é que ela considera que todos os nós têm capacidade de compartilhar informações com outros membros da rede. Juntamente com a possibilidade de qualquer um poder se juntar à rede, leva a um crescimento escalável do sistema, pelos nós compartilharem recursos como largura de banda, espaço de armazenamento e processamento. Além disso, a natureza distribuída de redes P2P aumenta sua robustez em caso de falhas nos pares (nós mem-

bros), ao replicar os dados entre múltiplos pares, evitando pontos críticos de falha. As redes *overlay* provêm uma maior resiliência pela replicação dos dados e recuperação no roteamento de mensagens. Nesse contexto, Zhuang et al. [2005] realizaram um estudo de algoritmos de detecção de falhas que compartilham informações de monitoramento. Eles observaram que as melhorias de desempenho se mostram mais visíveis com o aumento da vizinhança e que a troca de informações também possibilita uma maior tolerância a alta rotatividade de nós. So & Sirer [2007] examinaram o problema de como alocar banda de rede para monitoração de nós num esquema colaborativo (*multi-node*), buscando uma baixa latência de detecção.

2.3.4 Problemas Enfrentados em Sistemas Distribuídos de Larga Escala

Os detectores de falhas tradicionais enfrentam problemas ao lidar com sistemas distribuídos de larga escala (WANs), considerando-se que foram projetados para LANs e sistemas de escala limitada. Sendo assim, as implementações tradicionais não foram feitas para lidar com um número muito grande de processos, alto nível de perda de mensagens, topologia dinâmica de rede e a alta imprevisibilidade de WANs [Sotoma, 2006]. Segundo Hayashibara [2004], um detector de falhas para um sistema distribuído de larga escala (ex.: *Grid*) deve abordar problemas de:

- *Economia de Mensagens*: detectores de falhas podem gerar mensagens desnecessárias e se o número de processos for muito grande, pode não ser possível para um processo monitorar mais do que uma parte do sistema.

- *Escalabilidade*: detectores de falhas podem ter mensagens complexas, mas deve ser capaz de difundir informações de suspeita para todos os outros detectores de falhas.

- *Flexibilidade*: diferentes aplicações têm diferentes padrões de monitoramento e o detector de falhas deve ser calibrável e adaptável às aplicações.

- *Perda de Mensagens*: detectores de falhas geram falsos positivos, caso não se adaptem às condições da rede. A perda de mensagens é mais frequente em WANs do que em LANs, às vezes levando até a particionamentos da rede.

- *Dinamismo*: o detector de falhas se comporta de maneira correta apenas quando o sistema está estável por um longo período. A topologia e composição de sistemas distribuídos de larga escala são altamente dinâmicas (processos entrando e

saindo, padrões de uso mudando). Os detectores de falhas devem estar cientes de reconfigurações e se adaptar.

Os detectores de falhas podem ser implementados de diversas maneiras, sendo elas diretamente dependentes do contexto no qual serão aplicados e de acordo com as características particulares oferecidas pelo sistema subjacente. Além disso, sua complexidade será determinada pelo problema que deseja-se resolver, de maneira que será selecionado o detector de falhas mais fraco (mais simples) com o qual o sistema assíncrono subjacente deve ser equipado para que o problema possa ser resolvido. Observe que as propriedades que definem o detector de falhas mais fraco para o problema exprimem as condições necessárias e suficientes para que o problema seja resolvido, então o problema só poderá ser resolvido nos sistemas assíncronos que forem enriquecidos com os mecanismos adicionais necessários a implementação deste detector de falhas [Reynal, 2005].

2.4 Estado da Arte

Chen et al. [2002] propõem uma abordagem baseada numa análise probabilística do tráfego de rede, utilizando amostragens dos tempos de chegada para calcular uma estimativa acerca do tempo de chegada da próxima mensagem. O *timeout* é configurado de acordo com esta estimativa e é acrescido de uma margem de segurança constante α, que é calculada uma única vez, baseada nos requisitos de QoS. A estimativa do tempo de chegada do próximo *heartbeat* é recalculada a cada nova mensagem recebida, porém ela cresce mais lentamente pois o último desvio tem o mesmo peso que todos os anteriores no cálculo do *timeout*. São propostas duas versões do protocolo de detecção de falhas: uma que conta com relógios sincronizados e uma segunda versão que utiliza relógios dessincronizados, porém com um escorregamento individual desprezível.

Os trabalhos teóricos de Chen et al. [2002] motivaram o aparecimento de diversos trabalhos práticos com a implementação de detectores de falhas usando diferentes abordagens. Bertier et al. [2002] ampliam o trabalho anterior, substituindo a margem fixa de segurança por uma calculada com o algoritmo de Jacobson [1995] para a estimativa do RTT (*round-trip time*). Esta nova estimativa supõe que o comportamento do sistema não é constante, adaptando sua margem de segurança a cada mensagem recebida, passando a ser adaptável ao estado da rede de comunicação. Porém ele pode levar um longo tempo para convergir, tornando a estimativa menos precisa. Para a prova da classe do algoritmo como eventualmente perfeito foi considerado o modelo de sincronia parcial. Seu mecanismo de comunicação é baseado na troca de *heartbeats*

todos-pra-todos, porém se for possível utilizar as capacidades de IP-Multicast torna-se mais escalável (especialmente se utilizar divisão em grupos de detecção e organização hierárquica). A validação foi realizada utilizando-se um *cluster* de seis computadores, sendo a rede compatível com o protocolo de comunicação IP-Multicast. O intervalo de *heartbeat* foi estipulado como sendo de 5 s e foram consideradas as métricas de QoS tempo de detecção, tempo de recorrência de falso-positivos e duração do falso-positivo. Foi aplicada aos processos cargas pontuais e constantes e comparado com o detector de Chen. Na carga pontual eles obtiveram um tempo de reação médio muito próximo, porém sob carga constante o detector de Bertier obteve um tempo de detecção menor ao custo de mais falso-positivos e sua respectiva duração. Porém, conforme observado em Hayashibara et al. [2004] o detector de falhas de Bertier é muito sensível a perda de mensagens e *jitter*, pois foi originalmente planejado para ser utilizado em LANs, onde altas flutuações de rede raramente acontecem. Nesta linha ainda temos o detector de falhas adaptativo de Fetzer et al. [2001], que utiliza o intervalo máximo de comunicação para evitar falso-positivos e aproveita as mensagens trocadas pela aplicação para economizar mensagens de detecção de falhas. O trabalho de Sotoma & Madeira [2001] ajusta o intervalo de monitoramento e o *timeout* baseado no histórico dos últimos tempos observados, buscando assim construir estimativas dos atrasos futuros nas mensagens e prover um tempo de resposta rápido em falhas reais.

Hayashibara et al. [2004] propuseram um detector de falhas *accrual* cuja particularidade é ajustar dinamicamente a escala (na qual o nível de suspeita é expresso) às condições atuais da rede. A distribuição das amostras recebidas anteriormente é utilizada como uma aproximação da distribuição probabilística das futuras mensagens de *heartbeat*. De posse desta informação é possível calcular um valor φ numa escala que se altera dinamicamente para corresponder às alterações recentes nas condições da rede. A validação foi realizada num esquema contendo dois computadores ligados por um enlace intercontinental e também numa rede local, onde mensagens de *heartbeat* foram enviadas a cada 100 ms durante o período de uma semana. Das métricas propostas por Chen et al. [2002], a análise da qualidade de serviço da detecção de falhas considerou o *tempo de detecção* de falhas e a *taxa média de falso-positivos*. Foram analisados as taxas e rajadas de perda do trace coletado, bem como a variação do RTT. Os experimentos verificaram a taxa média de falso-positivos e o tempo médio de detecção em relação ao limiar de suspeita configurado, bem como o efeito da escolha do tamanho da janela de amostragem na reatividade do protocolo. Foram realizadas comparações de QoS com alguns detectores [Chen et al., 2002; Bertier et al., 2002] onde se verificou que o detector φ *accrual* apresentou um pequeno ganho de desempenho, comportando-se melhor na faixa agressiva de detecção de falhas. Os autores ainda concluem que uma

distribuição normal provê uma estimativa aceitável para os tempos de chegada dos *heartbeats* e que a flexibilidade provida não implica em nenhum custo significativo no desempenho do sistema.

Alguns novos trabalhos têm sido produzidos seguindo esta tendência, como por exemplo o novo detector *accrual* adaptativo proposto por Satzger et al. [2007], com características para aumentar a flexibilidade e diminuir os custos computacionais. É armazenado um histórico dos tempos de chegada entre mensagens, de maneira que é possível realizar uma estimativa da probabilidade de que nenhuma mensagem chegue após determinado atraso, ou seja, do processo ter falhado. Este histórico de mensagens provê o histograma do atraso das mensagens, que por sua vez pode ser visto como uma aproximação da PDF (função densidade de probabilidade) da distribuição dos tempos de chegada entre mensagens. A partir do histograma é possível então calcular as frequências cumulativas dos valores, que por sua vez pode ser vista como uma aproximação da CDF (função de distribuição cumulativa). E como a CDF representa a probabilidade de uma mensagem chegar com um atraso menor que o atual, pode-se utilizá-la para computar o valor de suspeita de falha através de uma função que calcula a porcentagem de valores que têm um atraso menor ou igual ao valor atual. A validação foi realizada através uma simulação que utilizou uma carga sintética com distribuição normal N(0;0,5) e 1% de perda de mensagens injetada, com um intervalo de *heartbeat* de 10 s. Da simulação foram colhidas as métricas "quantidade de falso-positivos" e "tempo médio de detecção", sendo realizadas comparações com os detectores de Chen et al. [2002], Bertier et al. [2002] e Hayashibara et al. [2004]. No cenário sem injeção de perdas de mensagens os detectores de Chen e Bertier obtiveram resultados melhores que ambos os *accrual*, porém no cenário com injeção de perdas de mensagens o detector de falhas de Satzger obteve desempenho superior a todos os demais. Ainda, o autor aponta para o fato de que seu algoritmo é computacionalmente menos intenso que o de Hayashibara.

Na abordagem hierárquica, podemos citar os detectores de falhas Globus [Stelling et al., 1999a] e CORBA [Felber et al., 1999], sendo que este último monitora os processos direta ou indiretamente, via outros detectores de falhas, reduzindo o tráfego pela combinação de diversas informações numa só mensagem, além de temporariamente replicar alguma informação em diversas partes do sistema. Eles apresentam uma boa escalabilidade porém não têm dinamismo. Já no detector de falhas de Bertier et al. [2003], a organização hierárquica é mapeada sobre a topologia da rede, utilizando grupos locais e globais (com seus respectivos líderes), reduzindo a complexidade das mensagens e também se recuperando de falhas, especialmente nos nós líderes. É dividido em duas camadas: uma básica, que provê um tempo de detecção curto e adapta o

intervalo de emissão às condições da rede e uma camada de adaptação, que personaliza a QoS provida pela camada básica de acordo com as necessidades da aplicação.

Em Gupta et al. [2001] é discutido porque os esquemas tradicionais de *heartbeat*, muito populares por garantirem a propriedade de completude, são inerentemente não-escaláveis de acordo com a carga ótima. Os autores ainda apresentam um detector de falhas distribuído aleatório, que impõe uma carga esperada igual por membro do grupo de detecção e apresenta uma carga de rede que difere do ótimo por um fator de sub-otimalidade que não varia com o tamanho do grupo. Também preocupado com a escalabilidade dos esquemas tradicionais de *heartbeat*, Yang et al. [2006] propõem uma abordagem baseada em notificações de falha onde as falhas em processos são localmente detectadas, através de um processo de monitoramento rodando separadamente porém na mesma máquina. Assim que uma falha é detectada, o processo monitor envia mensagens de notificação aos processos remotos interessados. Porém, um problema desta abordagem é que os processos remotos podem não ser informados acerca de uma falha se a mensagem de notificação for perdida ou a máquina como um todo falhar, resultando na perda da propriedade de completude. Uma abordagem híbrida de notificações de falha e *heartbeat* foi proposta para lidar com este problema e manter uma economia de mensagens. Turchetti & Nunes [2006] realizam aproveitamento das mensagens da aplicação para detectar a operacionalidade dos processos, além de realizar *piggybacking* das mensagens da aplicação para suprir mensagens de controle, economizando assim seu envio.

Na abordagem com disseminação epidêmica podemos citar como exemplo os detectores de falhas FD-PROB [Montresor, 1999], Gossip [van Renesse et al., 1998] e SWIN [Das et al., 2002], sendo que este último possui mecanismos redundantes para evitar falso-positivos, podendo ser calibrado para garantir um tempo de detecção em termos dos períodos do protocolo. A sua precisão depende de diversos parâmetros como as condições da rede, porém só pode lidar com um número fixo de processos. Chu [1998] apresenta um algoritmo para transformar um detector de falhas não-confiável num detector de falhas Ω, onde cada processo mantém um contador associado a cada outro processo. Estes contadores são incrementados e os processos trocam informações sobre seus valores utilizando disseminação epidêmica. Segundo Hayashibara et al. [2004], o processo com o menor valor é considerado o mais confiável e portanto o candidato mais desejável para ser o líder.

Capítulo 3

O Detector de Falhas Adaptativo FD-Sensi

A maior parte das soluções de detecção de falhas propostas em sistemas distribuídos são baseadas no pressuposto de que há um conhecimento prévio dos atrasos das mensagens, sendo isso utilizado para estimar o valor do *timeout* para os *heartbeats* sinalizarem uma possível falha [Stelling et al., 1999b; Chen et al., 2002]. Soluções baseadas nesse pressuposto provêem protocolos simples e rápidos que não afetam significantemente o desempenho do sistema. Porém, tais soluções de detecção não foram projetadas para tolerar pequenas flutuações de carga durante a sua execução, frequentemente produzindo falso-positivos.

Essas soluções tipicamente não toleram variações no atraso das mensagens trocadas entre os processos, mesmo que estas variações sejam pequenas. Como exemplo prático disso, podemos citar os protocolos de detecção de falhas implementados no JGroups [Montresor, 2006], que são pouco sensíveis a essas situações, não se adaptando bem ao ambiente não-homogêneo de um sistema distribuído. Neste contexto, os protocolos de detecção de falhas frequentemente geram falso-positivos ou mesmo não detectam as falhas em tempo hábil, quando elas realmente ocorrem. Isso é em grande parte devido ao fato desses protocolos utilizarem um *timeout* fixo, que apesar de poder ser pré-configurado, não se adaptará ao contexto da aplicação, que pode ser variável.

Segundo Nunes & Jansch-Porto [2004], o comportamento do *round-trip time* (RTT) na Internet corresponde a uma variável estocástica não-estacionária durante a maior parte do dia, ou seja, predições baseadas em séries não-estacionárias podem aumentar a precisão das predições dos atrasos. Uma escolha inadequada para o *timeout* deve ser evitada pois pode diminuir a precisão do detector ao aumentar a quantidade de falso-positivos, podendo também diminuir o seu desempenho ao aumentar o tempo

para detectar uma falha [Nunes & Jansch-Pôrto, 2002]. É necessário ressaltar que falso-positivos podem ser muito prejudiciais ao desempenho do sistema, especialmente em protocolos de consenso [Sampaio et al., 2003]. Neles, cada suspeita de falha dispara um custoso protocolo de concordância entre os membros do grupo de detecção, devendo ser evitada ao máximo a emissão desses falso-positivos [Montresor, 2006]. Assim, em situações reais onde as cargas da rede e das aplicações podem ser altamente variáveis e não previsíveis, a efetividade destes detectores de falhas pode comprometer a tarefa de gerenciamento dos processos. Esta situação se torna ainda mais crítica quando os processos estão espalhados numa WAN (*Wide Area Network*), onde o desempenho é influenciado pelo estado dinâmico dos processos pertencentes do sistema distribuído [Cui et al., 2003].

Um ponto crucial no projeto de serviços de detecção de falhas é a questão da calibração dos *timeouts* utilizados para identificar processos falhos [Sampaio et al., 2003]. Para atender a esta demanda propomos o detector de falhas FD-Sensi como uma técnica para se alcançar tolerância a falhas em grupos lógicos de maneira mais dinâmica, adaptando-se ao contexto e comportamento dos processos monitorados. O algoritmo deste novo detector foi projetado de maneira que ao invés de manter somente o *timestamp* da última comunicação, cada detector mantém uma lista das últimas atualizações recebidas. Esta lista de atualizações é então utilizada para determinar se o comportamento observado na comunicação de um processo está de acordo com o comportamento estatístico estimado pelo histórico dele.

O FD-Sensi utiliza uma abordagem estatística, baseando-se na variação observada no histórico de comunicação entre os processos. Assim, é possível aceitar atrasos de comunicação dentro de um certo intervalo de tolerância , calibrando automaticamente o *timeout* de acordo com o contexto no qual o sistema está inserido. De maneira geral, FD-Sensi busca obter ganhos de desempenho num ambiente distribuído heterogêneo e imprevisível, ao evitar grande parte dos falso-positivos observados em protocolos de *timeout* fixo. Conforme observado por Nunes & Jansch-Porto [2004], a predição de atrasos baseada somente na média (global ou amostral) não oferece uma boa precisão. Sendo assim, além da média, o FD-Sensi considera o desvio padrão amostral para incorporar à predição dos atrasos a variabilidade observada nos intervalos de recebimento de mensagens. Observe que o detector *accrual* adaptativo de Satzger et al. [2007] também é baseado numa abordagem estatística, utilizando as frequências cumulativas dos intervalos de recebimento para definir seu nível de suspeita. Porém, ao contrário dos detectores *accrual* [Hayashibara et al., 2004; Satzger et al., 2007], o FD-Sensi não assume que os intervalos de recebimento das mensagens sigam uma distribuição normal.

3.1 O Algoritmo do FD-Sensi

O pseudo-código simplificado do FD-Sensi pode ser observado no Algoritmo 1, que funciona da seguinte maneira: considere dois processos p e q onde q é monitorado por p e sua única tarefa (linha 2) é enviar mensagens do tipo *heartbeat* para p a cada Δ_i segundos. O processo p (linha 4) gerencia uma lista L com informações acerca dos intervalos de recebimento dos últimos η *heartbeats* (linha 10). Baseado neste histórico de *heartbeats*, o FD-Sensi constantemente adapta seu *timeout* (linha 25) baseando-se nas estatísticas descritivas *média* e *desvio padrão* dos intervalos entre atualizações observadas (linha 27) e caso um *heartbeat* de q não chegue dentro deste *timeout* "dinâmico", p assume que o processo q falhou (linha 29).

Algorithm 1 Detector de Falhas: FD-Sensi

```
1: Processo q:
2: envia uma mensagem do tipo heartbeat para p a cada △_i
3:
4: Processo p:
5: t_u ← −1 {timestamp da última atualização}
6: L ← new L() {lista com os intervalos de recebimento dos heartbeats}
7: η ← 1000 {tamanho máximo da lista}
8: κ ← 3 {fator de confiança}
9:
10: {invocado quando um heartbeat h_j é recebido}
11: função recebeMensagem(h_j)
12: t ← tempoAtual()
13: if t_u = −1 then
14:    t_u ← t
15: else
16:    t_△ ← (t − t_u)
17:    L.add(t_△)
18:    if tamanho de L > η then
19:       remove o elemento mais antigo de L
20:    end if
21:    t_u ← t
22: end if
23: return
24:
25: {invocado quando deseja-se identificar falhas em q no tempo t}
26: função detectaFalha()
27: timeout ← (T̄ + κσ)
28: elapsedtime ← (tempoAtual() − t_u)
29: return (elapsedtime > timeout)
```

O *timeout* do FD-Sensi é calculado através da média e desvio padrão dos tempos de recebimento dos *heartbeats*, conforme segue:

$$\bar{T} = \frac{(\sum_{i=1}^{\eta} t_i)}{\eta} \tag{3.1}$$

onde, $\bar{T}$ é a média aritmética dos intervalos de recebimento t_i dos últimos η *heartbeats* armazenados na lista L.

$$\sigma = \sqrt{\frac{1}{\eta - 1}\left(\sum_{i=1}^{\eta} t_i^2 - \eta \bar{T}^2\right)} \tag{3.2}$$

Na equação 3.2, σ representa o desvio padrão amostral do intervalo de recebimento de *heartbeats* da lista L. Unindo estas duas medidas, a expressão a seguir é utilizada para calcular o *timeout* estimado para o próximo *heartbeat*:

$$timeout = \bar{T} + \kappa\sigma \tag{3.3}$$

onde κ é uma constante que determina o fator de confiança a ser utilizado no limite superior do tempo de espera. Esta abordagem é semelhante ao mecanismo do protocolo TCP para estimar *round-trip times* (RTTs) [Jacobson, 1995].

Fator de Confiança (κ)

O fator de confiança κ é um importante parâmetro do FD-Sensi, definindo a sensibilidade do algoritmo às mudanças no comportamento do processo monitorado. Ele é utilizado para determinar qual nível de atraso será tolerado pelo *timeout* dos *heartbeats*. O valor de κ deve ser estipulado de acordo com a distribuição dos atrasos no recebimento dos *heartbeats*, porém estimar qual o comportamento dessa distribuição é uma tarefa nem sempre viável, devido a grande variedade de cenários de execução. Portanto, é necessário considerar um *valor mais conservador* para a escolha dos intervalos de confiança a serem utilizados. Na teoria da probabilidade, a Desigualdade de Chebyshev atesta que em qualquer amostra de dados ou distribuição de probabilidades, quase todos os valores estão perto do valor da média e de acordo com o número de desvios padrões que se toma dela, é possível estimar que porcentagem dos valores estará contida no intervalo de confiança especificado. A fórmula a seguir representa a Desigualdade de Chebyshev [Mo, 1984]:

$$Pr(\mid X - \mu \mid \geq \kappa\sigma) \leq \frac{1}{\kappa^2} \tag{3.4}$$

onde X é uma variável aleatória com valor esperado μ e uma variância finita de σ^2, sendo definida para qualquer número real $\kappa > 0$.

Por exemplo, não mais do que 1/4 dos valores estão mais do que dois desvios padrões além da média, não mais do que 1/9 dos valores estão além de 3 desvios padrões da média e não mais do que 1/25 estão além de 5 desvios padrões e assim por diante. Portanto, se não é conhecido nada a respeito da distribuição, uma boa estimativa é utilizar os valores provenientes da Desigualdade de Chebyshev, conforme pode ser observado nos exemplos da tabela 3.1.

Tabela 3.1. Intervalos de Confiança da Desigualdade de Chebyshev

Desvios Padrão	*% Valores no Intervalo de Confiança*
$1,4\sigma$	ao menos 50%
2σ	ao menos 75%
3σ	ao menos 89%
4σ	ao menos 94%
5σ	ao menos 96%
6σ	ao menos 97%
7σ	ao menos 98%
$\kappa\sigma$	ao menos $(1 - 1/\kappa^2)$%

Um valor muito baixo deve ser evitado, pois faria com que o FD-Sensi se comporte de maneira menos conservadora, afetando consequentemente a sua precisão. Por outro lado, deve-se evitar valores muito altos, pois estes tornariam o detector demasiadamente tolerante aos atrasos, degradando a velocidade com que as falhas seriam detectadas. O parâmetro κ define basicamente o custo benefício entre precisão e velocidade do algoritmo. Sendo assim, para que o algoritmo seja sensível somente a determinados valores discrepantes, o κ deve ser escolhido de forma a adequar-se às necessidades reais da aplicação.

Tamanho da Lista de *Heartbeats* (η)

O tamanho máximo da lista L também deve ser configurado, definindo o tamanho da janela de amostragem estatística onde somente o intervalo de recebimento dos últimos η *heartbeats* será utilizado no cálculo do *timeout*. É um parâmetro que define a reatividade do detector, onde caso ele seja muito pequeno o detector apresentará uma baixa histerese[1] , mas caso seja demasiado grande as estatísticas calculadas serão

[1] histerese consiste na tolerância a pequenas variações onde o estado atual depende das propriedades dos estados anteriores e é intencionalmente adicionada para prevenir chaveamentos (troca de estados) rápidos.

amenizadas pelo peso de *heartbeats* antigos, que não necessariamente representam o estado atual do sistema.

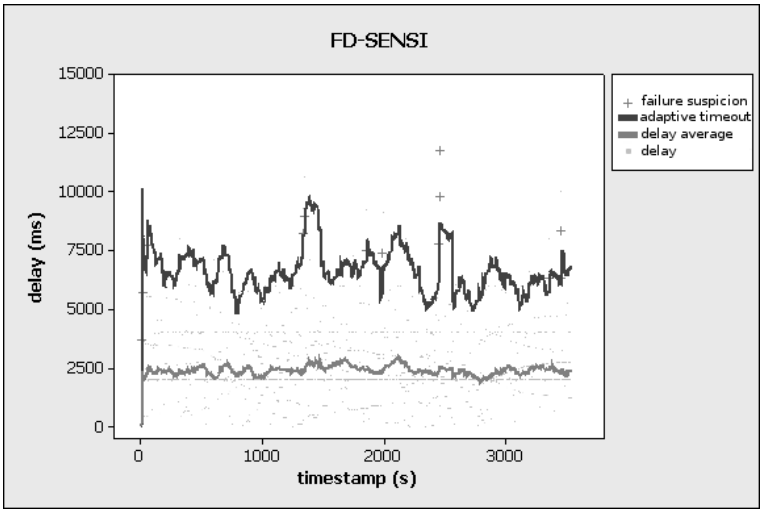


Figura 3.1. *Timeout* do FD-Sensi Acompanhando as Variações nos Atrasos

Assim, através do histórico armazenado, o intervalo médio de recebimento dos *heartbeats* e principalmente a variação nesse intervalo podem se calculados estatisticamente, visando obter uma melhor estimativa acerca de possíveis falhas no processo q. Desta forma, o *timeout* passa a ser dinâmico e adaptativo, determinando um valor mais apropriado para a situação de carga que o sistema esteja experimentando naquele momento, conforme ilustrado na figura 3.1. Assim, o FD-Sensi somente suspeitará de atrasos no recebimento dos *heartbeats* caso estes sejam muito discrepantes em relação aos últimos atrasos registrados.

3.2 Desempenho do Algoritmo com uma Carga Sintética

Para avaliar o algoritmo e possibilitar comparações de desempenho, realizamos simulações baseando-nos na análise de desempenho realizada por Satzger et al. [2007], onde uma carga sintética de dados é gerada segundo uma distribuição para a ocorrência de atrasos nas mensagens e uma taxa de injeção de falhas. Para comparar o desempenho escolhemos o *Adaptive Accrual* pois na avaliação de Satzger et al. [2007] este apresentou ótimos resultados em comparação com os detectores de falhas de Chen et al. [2002] Bertier et al. [2002] e *Accrual* φ de Hayashibara et al. [2004]. Conforme observamos na figura 3.2 o *Adaptive Accrual* consistiu na curva que mais se aproximou da origem,

apresentando assim um melhor custo-benefício na qualidade de detecção (velocidade X precisão).

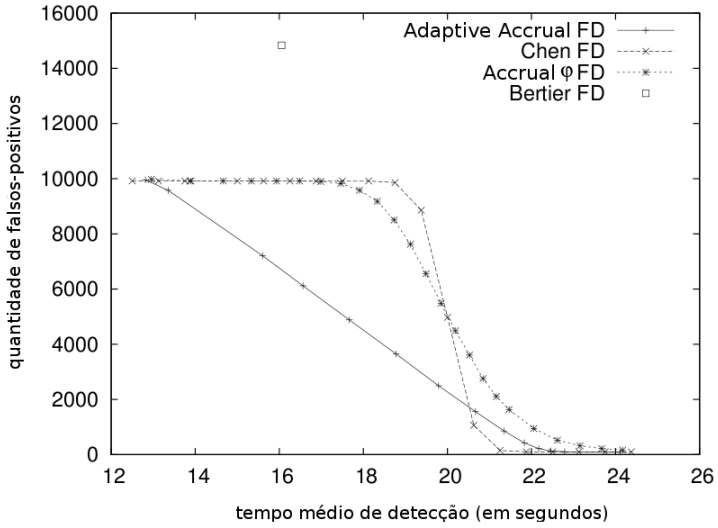


Figura 3.2. Qualidade de Detecção do *Adaptive Accrual*

Fonte: Satzger et al. [2007], pág. 5. Copyright 2007 ACM.

Porém, acreditamos que utilização de uma carga seguindo a distribuição normal não é suficiente para a validação do algoritmo, visto que os atrasos de rede não necessariamente seguem tal distribuição. Propomos a verificação do impacto no desempenho causado pela utilização de uma distribuição com a presença de valores mais extremos, visando assim prover uma maior variabilidade aos atrasos de mensagens. Assim, além da distribuição normal utilizada por Satzger et al. [2007], empregaremos cargas geradas com a distribuição exponencial para repetir os mesmos testes, por ser este um modelo que proverá uma maior dispersão nos atrasos. Observe que exponencial não é proposta como um melhor modelo que a normal para os atrasos de rede, porém utilizámo-la para checar o impacto no desempenho dos FDs causado por um cenário com atrasos não-normais. As métricas, parâmetros e as cargas de trabalhos utilizadas serão detalhados nas seções seguintes.

3.2.1 Métricas

A qualidade dos detectores de falhas é classicamente medida pela sua completude e precisão, onde a completude refere-se à capacidade dos FDs de eventualmente suspeitar de processos falhos e a precisão restringe os enganos cometidos ao suspeitar de um processo não-falho [Chandra & Toueg, 1996]. Ainda, existe um compromisso entre

precisão e velocidade, onde por um lado, ao se diminuir a ocorrência de falso-positivos (aumentando sua precisão ao tolerar atrasos maiores de comunicação), o tempo de detecção aumentará pois as falhas só serão detectadas num tempo maior.

Assim, um FD deve ser rápido e/ou preciso, dependendo das necessidades da aplicação. A velocidade de um detector de falhas pode ser medida como o tempo decorrido entre o momento em que um processo p falha e o tempo em que o FD passa a suspeitar de p. Já a precisão de um FD pode ser avaliada por uma série de métricas, conforme observado por Chen et al. [2002]. Para medir a qualidade de detecção e prover meios de comparação entre diferentes FDs, a qualidade de um FD pode ser dada, dentre outras, em termos de duas métricas:

- *Tempo Médio de Detecção*: mede a **velocidade** em termos do tempo decorrido entre a falha de um processo e a suspeita dessa falha.
- *Quantidade de falso-positivos*: mede a **precisão** em termos da quantidade de enganos cometidos pelo detector de falhas ao afirmar que um processo correto falhou.

Falso-negativos não são admitidos pois conforme vimos na seção 2.2.1, a propriedade de *completude forte*, necessária para resolver o problema do consenso, garante que eventualmente cada processo que falhe é permanentemente suspeitado pelos processos corretos. Sendo assim, haverá um momento no qual os *timeouts* dos detectores serão ultrapassados pelo intervalo de tempo sem atualizações devido à falha no processo.

3.2.2 Parâmetros e Fatores

Os parâmetros que influenciam no desempenho dos algoritmos de detecção de falhas são:

- $\triangle_i$ Intervalo de envio dos *heartbeats*.
- Existência ou não de perdas no envio/recebimento das mensagens.
- Existência ou não de falha injetada no processo num dado momento.

Cada algoritmo possui seus respectivos parâmetros de calibragem, sendo eles:

FD-Sensi: seu *timeout* é calculado como sendo $\bar{T} + \kappa.\sigma$, onde $\bar{T}$ é a média observada nos intervalos de recebimento dos *heartbeats* e σ é o seu desvio padrão. Sendo assim, os parâmetros deste algoritmo são:

- κ: define o fator de confiança acima da média observada nos tempos dos *heartbeats*

- η: tamanho da janela de amostragem estatística (histórico de *heartbeats*)

Adaptive Accrual: seu *timeout* é calculado utilizando $\alpha.\triangle_i$, onde $\triangle_i$ é o intervalo de envio de *heartbeats*. Portanto, os parâmetros deste algoritmo são:

- α: fator de escala, que define qual a porcentagem máxima a ser aceita acima do intervalo observado para o *heartbeat*
- η: tamanho da janela de amostragem estatística (histórico de *heartbeats*)

Para a avaliação dos algoritmos de detecção de falhas, será fixado o parâmetro η, considerando-se uma janela de amostragem com os 1000 últimos *heartbeats*, sendo os demais parâmetros eleitos como fatores, ou seja, aqueles que iremos variar. O η define a reatividade do algoritmo de detecção, pois quanto menor a janela de amostragem, mais a média observada nos tempos dos *heartbeats* será afetada, tal qual a frequência do seu histograma e vice-versa.

3.2.3 Carga de Trabalho

Os dados necessários para avaliar o desempenho do FD são os tempos de chegada dos *heartbeats*, que são mensagens periódicas de monitoração trocadas entre os processos. No caso, consideramos sistemas distribuídos onde processos podem falhar e mensagens podem chegar atrasadas ou serem descartadas pelos enlaces de comunicação e esses atrasos e perdas seguem uma certa distribuição de probabilidade.

Existe um conjunto ilimitado de ambientes envolvendo os processos e seus meios de interconexão nos quais o FD poderia ser testado, porém baseando-nos na abordagem utilizada por Satzger et al. [2007], decidimos para uma primeira avaliação gerar uma massa de dados sintética para a avaliação do FD, de maneira que os experimentos possam ser reproduzidos e os resultados sejam independentes de peculiaridades do canal de comunicação. Portanto, o tempo de chegada dos *heartbeats* será gerado conforme o seguinte modelo:

$$t_j = \begin{cases} j \ . \ \triangle_i \ + \ \delta, & com\ probabilidade\ 1 \ - \ \theta \\ mensagem\ perdida, & com\ probabilidade\ \theta \end{cases}$$

O tempo de chegada do j-ésimo *heartbeat* é composto por $j.\triangle_i$ (onde $\triangle_i$ é o intervalo de envio) e θ é a probabilidade de uma mensagem se perder. A este tempo de chegada é somado o atraso no recebimento da mensagem, representado por δ, cujas características podem ser geradas segundo distribuições probabilísticas. o trabalho de Sotoma [2006] propõe o uso de um modelo baseado em cadeias de Markov para capturar

a informação da distribuição de probabilidade do comprimento de rajadas de perdas de mensagens, porém por critérios de simplicidade e para possibilitar comparações com a avaliação realizada por Satzger et al. [2007], mantivemos uma distribuição aleatória para a perda de mensagens. As distribuições para os atrasos, utilizadas nesta primeira avaliação, são a normal e a exponencial, sendo δ uma variável aleatória gerada de acordo com estas duas distribuições.

Distribuição Normal: Satzger et al. [2007] utilizaram a distribuição normal para simular os atrasos inseridos nos tempos de chegada dos *heartbeats*. Portanto, fizemos uma comparação entre os algoritmos utilizando essa distribuição para avaliarmos de maneira semelhante a apresentada por ele. Para tal foi gerada uma amostragem de pontos normalmente distribuídos com média 0 segundos e desvio padrão 1, δ = N(0;1)

Distribuição Exponencial: Conforme ponderamos, a utilização de uma distribuição normal para amostragem de atraso pode não refletir um comportamento real de chegadas de mensagens, cujos atrasos podem atingir valores muito distantes da média e apresentar uma alta variabilidade. Acreditamos ainda que a utilização de uma normal para avaliação do *Adaptive Accrual* foi devido a sua característica de adaptação baseada na normalidade dos *heartbeats* da janela de amostragem. Desta forma, buscando modelar uma maior variabilidade nos atrasos e também possibilitar uma análise comparativa entre o comportamento dos algoritmos com diferentes tipos de cargas foi utilizada além da normal, uma distribuição exponencial com o parâmetro $\lambda = 1$ segundo. Visamos com isso obter uma maior dispersão nos atrasos e medir seu impacto nos algoritmos.

Sendo assim, dado o modelo de geração dos *heartbeats*, para cada distribuição, serão obtidos quatro cenários com diferentes cargas de trabalho, configurados com os seguintes níveis:

- Cenário 1: $\triangle_i = 10$ segundos e $\theta = 0{,}00$
- Cenário 2: $\triangle_i = 10$ segundos e $\theta = 0{,}02$
- Cenário 3: $\triangle_i = 2$ segundos e $\theta = 0{,}00$
- Cenário 4: $\triangle_i = 2$ segundos e $\theta = 0{,}02$

Diferentemente de Satzger et al. [2007], também iremos avaliar um intervalo de envio de *heartbeats* menor (2 segundos), para verificarmos o impacto da magnitude do atraso de mensagem no desempenho do FD ao diminuir a proporção entre $\triangle_i$ e δ. As falhas também são injetadas na carga com 2% de probabilidade de ocorrência, aleatoriamente distribuídas. Seu significado é que naquele instante não foi recebido o *heartbeat* correspondente, proveniente de uma falha no processo. Observe que o não recebimento de *heartbeats* também pode ser proveniente de ocorrência de perda daquela mensagem. E uma perda de mensagem não significa uma falha no sistema, mas sim que o *heartbeat* esperado para aquele instante foi corrompido e/ou descartado em algum lugar. Por isto a nomenclatura "falso-positivos" indica as falhas erroneamente detectadas, visto que é justamente nos cenários com perdas de mensagens que os algoritmos devem apresentar uma maior permissividade para tolerar os atrasos no recebimento dos *heartbeats* e a variação destes atrasos.

O tamanho da amostra é de 100.000 pontos para todas as cargas geradas. Esta quantidade foi obtida a partir da análise de diversos tamanhos de amostra das distribuições geradas, variando de 1.000 a 1.000.000 mensagens. A partir de aproximadamente 100.000 mensagens, os resultados mantinham proporcionalmente os mesmos resultados porém o tempo de processamento aumentava substancialmente.

3.2.4 Simulador

Foi construído um simulador em Java para realizar a avaliação de desempenho dos protocolos de detecção de falhas, automatizando todo o processo desde a obtenção da carga de trabalho até a análise de resultados dos experimentos.

As cargas de trabalho consistiam em arquivos de texto simples, contendo os valores de atraso das mensagens a serem simuladas e geradas segundo uma distribuição (normal ou exponencial). Num primeiro passo, estas cargas eram carregadas para a memória principal de maneira a simular o intervalo de comunicação do *heartbeat*. As perdas de mensagem e falhas são nelas injetadas e após isso as cargas são repassadas aos algoritmos de detecção de falhas para serem processadas.

No detector de falhas, a cada iteração o tempo simulado é incrementado de acordo com o intervalo de comunicação configurado e então são lidos e armazenados no histórico os *heartbeats* "recebidos" com tempo inferior ao tempo simulado atual. Além disso, cada iteração é também uma rodada de detecção de falhas, onde é verificado se algum *heartbeat* não chegou dentro do tempo esperado. Caso alguma suspeita de falha seja acusada, é escrito num *log* o *timestamp* da suspeita e o motivo dela, para que no final este *log* possa ser comparado com as falhas injetadas na carga de trabalho

e as métricas possam ser coletadas. Após a carga ter sido completamente processada, tanto ela como o *log* de saída do protocolo de detecção de falha são repassados a um analisador, que compara-os e coleta as seguintes métricas:

Quantidade de Falso-positivos: conforme anteriormente explicado, falsas suspeitas são provenientes de mensagens que não chegaram no tempo esperado (ex.: estavam apenas atrasadas) e foram consideradas equivocadamente como falhas do processo. Para identificá-las, basta contabilizar as suspeitas do *log* e subtrair da quantidade de falhas efetivamente injetadas.

Tempo Médio de Detecção: o tempo de detecção de uma falha é o tempo decorrido desde a ocorrência da falha até a sua efetiva suspeita por parte do protocolo de detecção de falhas. O seu cálculo é realizado da seguinte maneira: para uma determinada suspeita de falha do *log*, é verificado o *timestamp* da suspeita e diminuído do tempo em que a falha ocorreu (obtido a partir da carga de trabalho). A relação entre suspeitas e falhas é feita heuristicamente da seguinte maneira: para uma determinada falha considera-se como suspeita a primeira suspeita observada com tempo de simulação superior à falha em questão. A partir destes tempos de detecção é calculada a média e esta é normalizada pelo intervalo de *heartbeat* da configuração utilizada.

Após coletar estas métricas, o analisador imprime-as, bem como de qual configuração de experimento elas foram obtidas. Por fim, a unidade macro do simulador é o próprio projeto experimental, onde as etapas do experimento são encadeadas de maneira a aplicar cada carga de trabalho aos dois protocolos de detecção, passando por cada configuração prevista no projeto do experimento, numa ordem que possibilite que os resultados impressos sejam imediatamente transferidos para a planilha de análise dos resultados.

Para agilizar a execução dos experimentos, o simulador foi ainda construído de maneira *multi-threading*, de maneira a aproveitar o processamento *multi-core* hoje em dia cada vez mais comum nos computadores.

3.2.5 Projeto Fatorial 2^3

Um projeto experimental fatorial completo utiliza cada combinação possível de todos os níveis de todos os fatores, o que pode levar a um número muito grande de experimentos se a quantidade de fatores ou de níveis for muito grande [Jain, 1991]. Assim sendo, realizamos um projeto fatorial para cada tipo de distribuição probabilística da carga

(normal ou exponencial), variando como fatores os algoritmos, existência de perda de mensagens e intervalo de *heartbeat*. Desejávamos com isso verificar qual a influência de cada fator escolhido no desempenho para escolher aqueles que tenham um impacto mais significativo, de maneira que nosso projeto fatorial 2^3 possui três fatores com dois níveis cada:

$$A:\ Algoritmo\ =\ \begin{cases} FD-Sensi\ =\ -1, \\ Adaptive\ Accrual\ =1 \end{cases}$$

$$B:\ Carga\ =\ \begin{cases} Com\ 2\%\ de\ perda\ =\ -1, \\ Sem\ perda\ =\ 1 \end{cases}$$

$$C:\ Intervalo\ de\ heartbeat\ =\ \begin{cases} 2\ segundos\ =\ -1, \\ 10\ segundos\ =\ 1 \end{cases}$$

Lembrando, as falhas foram injetadas com 2% de probabilidade de ocorrência e aleatoriamente distribuídas. Como este parâmetro não é variado, ele não é considerado um fator.

3.2.6 Projeto Simples

Após a execução do projeto fatorial, refinamos nossa análise através de um projeto simples. No projeto simples, inicia-se com uma configuração típica e varia-se um fator de cada vez para checar como aquele fator influencia no desempenho [Jain, 1991]. Variamos assim os parâmetros de ajuste de cada algoritmo e para cada distribuição probabilística. Cada algoritmo teve seu parâmetro de ajuste variado em 23 níveis, representando uma faixa de valores que vai desde alta precisão com baixa velocidade até alta velocidade com baixa precisão, com detalhamentos em setores intermediários de custo-benefício:

- FD-Sensi: parâmetro κ variando de 0 a 8: 0,0 ; 0,05 ; 0,1 ; 0,25 ; 0,5 ; 1,0 ; 1,1 ; 1,5 ; 1,75 ; 2,0 ; 2,25 ; 2,75 ; 3,0 ; 3,5 ; 4,0 ; 5 ; 5,5 ; 6 ; 6,0 ; 7 ; 7,5 ; 8

- *Adaptive Accrual*: parâmetro α variando de 0 a 1,5: 0,0 ; 0,125 ; 0,25 ; 0,375 ; 0,50 ; 0,625 ; 0,75 ; 0,875 ; 1,0 ; 1,01 ; 1,03 ; 1,03 ; 1,05 ; 1,09 ; 1,1 ; 1,125 ; 1,185 ; 1,2 ; 1,25 ; 1,3 ; 1,35 ; 1,4 ; 1,45 ; 1,5

3.3 Resultados

Os resultados de cada projeto experimental (fatorial e simples) são apresentados a seguir, para um dos dois tipos de distribuição probabilística.

3.3.1 Projeto Fatorial 2^3

Conforme detalhado na seção anterior, foi realizado um Projeto Fatorial 2^3 buscando identificar o impacto de cada fator no desempenho da detecção de falhas.

3.3.1.1 Distribuição Normal

Através do Projeto Fatorial 2^3 para a distribuição normal, obtivemos os resultados da tabela de alocação de variância 3.2, onde observamos que a variação dos falso-positivos é melhor explicada pelos fatores A (algoritmo), B (perdas na carga) e AB (interação dos fatores algoritmo e perdas na carga).

Tabela 3.2. Efeito dos fatores - Distribuição Normal

Frações	*A*	*B*	*C*	*AB*	*AC*	*BC*	*ABC*
falso-positivos	**25%**	**20%**	8%	**23%**	8%	7%	9%
tempo de detecção	1%	**45%**	**19%**	4%	**18%**	10%	2%

A variação no tempo médio de detecção é melhor explicada pelos fatores B (perdas na carga), C (intervalo de *heartbeat*) e AC (interação dos fatores algoritmo e intervalo de *heartbeat*). Isso dá sinais de que a variação da magnitude do intervalo de *heartbeat* em relação aos atrasos simulados pela distribuição têm uma influência significativa no tempo médio de detecção. Por exemplo, um atraso de 300 ms em relação a um intervalo de *heartbeat* de 2 segundos tem um impacto maior do que com um intervalo de 10 segundos. Isso fez com que o desempenho variasse mais nos cenários com configurações menores para o intervalo de *heartbeat* (2 segundos).

3.3.1.2 Distribuição Exponencial

Realizando o Projeto Fatorial 2^3 para a distribuição exponencial, obtivemos os resultados da tabela de alocação de variância 3.3, onde observamos que os fatores que mais influenciam na variação dos falso-positivos para a distribuição exponencial são A (algoritmo), B (perdas na carga) e AB (interação de A e B) e no tempo de execução são A (algoritmo), C (intervalo de *heartbeat*) e AC (interação de A e C).

Tabela 3.3. Efeito dos fatores - Distribuição Exponencial

Frações	*A*	*B*	*C*	*AB*	*AC*	*BC*	*ABC*
falso-positivos	**31%**	**27%**	2%	**31%**	3%	2%	3%
tempo de detecção	**41%**	11%	**24%**	0%	**15%**	1%	8%

Comparando com os resultados da alocação de variância das distribuições exponencial e normal, é interessante notar que nesta última, o algoritmo (fator A) explica muito pouco da variabilidade (cerca de 1%) da métrica tempo médio de detecção, fazendo com que a existência de perdas de mensagens na carga explique uma considerável fatia da variação, seja individualmente, seja na interação com outros fatores. Já na distribuição exponencial, este fenômeno não é observado, de maneira que o algoritmo passa a explicar uma grande parcela da variabilidade (41%) do tempo médio de detecção.

Assim, percebe-se que ao utilizar uma distribuição normal para simular os atrasos nos recebimentos, a escolha do algoritmo de detecção não faz muita diferença na variação do tempo médio de detecção, dando sinais de que ambos os algoritmos aproximam muito seu comportamento quanto a esta métrica. Isso não é verdade quando se utiliza a distribuição exponencial, pois nela os valores dos atrasos no recebimento dos *heartbeats* estão mais dispersos, apresentando uma maior ocorrência de valores grandes (em contrapartida à concentração de valores encontrada na distribuição normal), o que faz com que os algoritmos passem a influenciar bastante no tempo médio de detecção.

Quanto à métrica quantidade de falso-positivos, em ambas as distribuições os algoritmos explicam a maior parte da variabilidade encontrada, juntamente com a existência de perdas de mensagem na carga de trabalho. Assim sendo, a forma como os algoritmos se adaptam à ocorrência de atrasos e falhas também têm um impacto muito forte na emissão de falso-positivos, conforme era de se esperar. Vale notar que um algoritmo muito reativo, quanto à ocorrência de atrasos e falhas, proporcionará uma significativa diminuição na quantidade de falso-positivos, mas ao custo de um aumento no tempo médio de detecção dada sua maior tolerância aos atrasos.

3.3.2 Projeto Simples

Com o objetivo de realizar uma comparação mais detalhada entre os algoritmos FD-Sensi e *Adaptive Accrual*, realizamos um projeto simples, variando os parâmetros de ajuste de cada algoritmo. As falhas foram injetadas de maneira aleatória na proporção de 2% do tamanho da carga. Para cada tipo de distribuição da carga (normal ou

exponencial) foram realizados experimentos em quatro tipos de cenários, variando a existência ou não de perdas na carga (0% e 2%) e o intervalo do *heartbeat* (2 ou 10 segundos). Conforme visto na seção 3.2.3, os cenários são:

- Cenário 1: $\triangle_i$ = 10 segundos e θ = 0,00
- Cenário 2: $\triangle_i$ = 10 segundos e θ = 0,02
- Cenário 3: $\triangle_i$ = 2 segundos e θ = 0,00
- Cenário 4: $\triangle_i$ = 2 segundos e θ = 0,02

Tabela 3.4. Intervalos de Confiança para os Falso-positivos

	Normal		Exponencial	
Cenário				
1	-0,42	1,64	-0,42	1,64
2	-15,85	12,12	-0,45	0,63
3	**0,28**	**1,45**	**0,42**	**1,32**
4	-0,55	1,47	-0,54	1,42

As tabelas 3.4 e 3.5 exibem um resumo dos intervalos de confiança do teste T de Student para a média da diferença de desempenho entre os detectores. Estes intervalos de confiança são utilizados para comparar o desempenho das 23 amostras (níveis para κ e α) através do teste de diferenças entre as médias populacionais. Através deste teste queremos identificar quais dos cenários ocasionaram uma alteração de desempenho diferenciável com 95% de confiança, ou seja, aqueles em que os intervalos de confiança obtidos não incluem o zero.

Para a métrica *quantidade de falso-positivos*, no cenário 3 (sem ocorrência de perdas e com *heartbeats* a intervalos de 2 segundos), para ambas as distribuições pode-se afirmar que os FDs são diferentes com um nível de confiança de 95%. Tal fato justifica-se pela magnitude do intervalo dos *heartbeats* em relação aos atrasos no seu recebimento.

Já para a métrica *tempo médio de detecção*, no cenário 3 (sem perdas de mensagem e com *heartbeats* a intervalos de 2 segundos), apenas para a distribuição exponencial pode-se afirmar que os FDs são diferentes com um nível de confiança de 95%. Além da magnitude do intervalo dos *heartbeats* em relação aos atrasos, isso se explica também pelo maior espalhamento de valores proporcionado pela distribuição exponencial, gerando grandes atrasos com uma maior frequência que a distribuição normal.

Tabela 3.5. Intervalos de Confiança para o Tempo de Detecção

	Normal		Exponencial	
Cenário				
1	-0,42	1,64	-1,72	0,45
2	-0,10	0,10	-0,60	0,40
3	-1,87	1,09	**-1,85**	**-0,09**
4	-1,35	1,29	-2,91	0,99

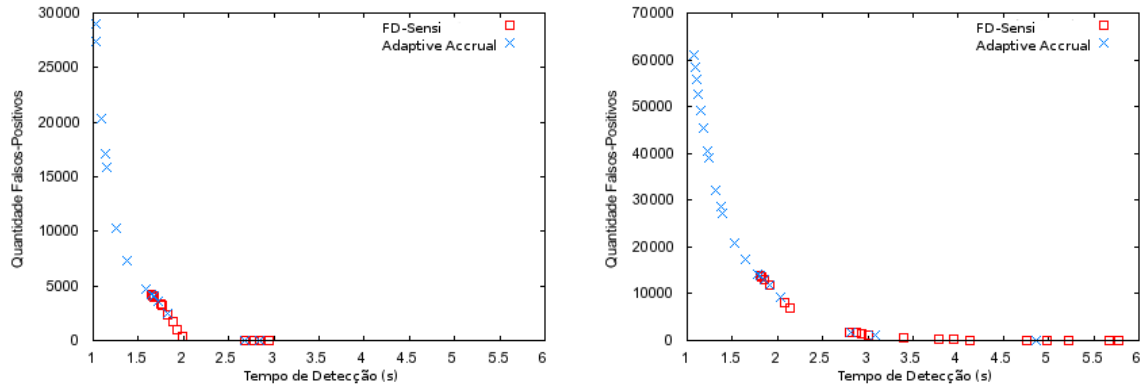
Observe que nos demais cenários, não é possível afirmar que o FD-Sensi possui um desempenho superior, dado que os ICs de ambas as métricas incluem o zero, indicando que os FDs não são diferentes com um nível de confiança de 95%. Todavia, isso deve ser considerado como um resultado promissor para uma primeira análise, pois demonstra que o FD-Sensi conseguiu um desempenho tão bom quanto o *Adaptive Accrual*, que pode ser considerado como um excelente detector de falhas.

3.3.3 Análise da Qualidade dos Detectores

Consideremos nessa análise o cenário 3, onde os algoritmos tiveram seu desempenho diferenciável com 95% de confiança. Nesse cenário não houve ocorrência de perda de mensagens e o intervalo de *heartbeat* foi de 2 segundos. Conforme pode ser observado nas figuras 3.3b, a utilização da distribuição exponencial fez com que ambos os algoritmos gerassem mais falsas suspeitas em relação aos experimentos com a distribuição normal, além de provocar também um aumento do tempo médio de detecção de ambos os algoritmos. Isso é de se esperar pois na carga exponencialmente distribuída, nota-se uma maior ocorrência de maiores valores de atrasos, conforme pode ser observado no histograma da distribuição de atrasos representado na figura 3.4, para uma população de 100.000 amostras.

Isso nos dá sinais de que ambos os algoritmos funcionam melhor em cenários com atrasos de comunicação normalmente distribuídos, onde se pode de certa forma realizar melhores previsões de acordo com a implementação de suas heurísticas. Porém, uma distribuição normal para os atrasos na comunicação não é tipicamente encontrada em cenários reais, especialmente aqueles que envolvam processos dispersos numa WAN. Como não se conhece necessariamente a distribuição apresentada pelos atrasos recebidos na comunicação dos detectores de falhas, a utilização de um detector que se adapte rapidamente aos atrasos e perdas e seja de certa forma independente da distribuição dos mesmos pode trazer benefícios.

Via de regra, em gráficos do tipo "falso-positivos X tempo médio de detecção" é



(a) Normal (b) Exponencial

Figura 3.3. Comparação da Qualidade de Detecção com Cargas Normal e Exponencial

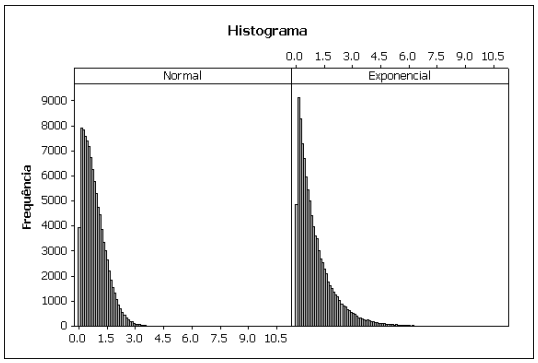


Figura 3.4. Histogramas das Distribuições de Atrasos Normal e Exponencial

interessante que o comportamento do FD se aproxime mais da origem, gerando menos falso-positivos e detectando falhas num tempo menor, dado o compromisso entre essas duas métricas. Porém, como os ICs encontrados na seção 3.3.2 não pudemos atestar a superioridade de desempenho entre estes dois FDs. A opção por um deles dependerá das prioridades das aplicações quanto às métricas aqui analisadas. Por exemplo, em esquemas distribuídos do tipo mestre-escravo a precisão é mais importante que a velocidade. Isso se deve ao fato de que o custo do mestre cancelar um processamento e atribuí-lo a outro escravo é grande, devendo-se optar por uma alta precisão ao custo de uma velocidade pior.

Olhando novamente para as figuras 3.3a e 3.3b, pode-se perceber que de maneira geral o FD-Sensi gera menos falso-positivos que o *Adaptive Accrual*. Para aplicações onde o custo de se tratar um falso-positivo é um processo oneroso (ex.: consenso no JGroups, atribuição de trabalhos em esquema mestre-escravo), é interessante gerar

o mínimo de falso-positivos possível. Já para aplicações onde seja mais importante detectar uma falha num curto intervalo de tempo, o *Adaptive Accrual* seria mais indicado, por apresentar tempos médios de detecção menores que o FD-Sensi, naturalmente ao custo de uma maior emissão de falso-positivos. Porém, o *Adaptive Accrual* exige ambientes onde a distribuição dos atrasos é normal.

Observe que a aparente limitação do FD-Sensi quanto ao tempo de detecção é devida ao seu fator de confiança κ ter sido limitado a valores maiores que zero pela definição da desigualdade de *Chebyshev*. Na seção 3.2.6 é dito que também utilizamos o valor 0,0 para κ e isso foi devido a querermos testar o comportamento do algoritmo sem o uso de desvios padrões, somente com a média. Isso nos deu a ideia de utilizar, nos próximos experimentos, também valores negativos para κ. Apesar de não previstos pela desigualdade de *Chebyshev*, os valores negativos visarão ganhos de desempenho no tempo médio de detecção ao custo de mais falso-positivos.

É também interessante observar que para a distribuição exponencial, ambos os FDs passaram a gerar falsos-suspeitos para uma quantidade maior de configurações . Isso dá sinais que a utilização de distribuições não-normais para avaliar o desempenho dos FDs pode ser promissora. Isso talvez revele aspectos de seu comportamento não observados no uso da distribuição normal, pois esta não necessariamente reflete o comportamento observado para atrasos em mensagens em LANs e WANs. De uma maneira geral, percebe-se que as diferenças entre os algoritmos são menos acentuadas nos experimentos com distribuição normal dos atrasos e com diferentes intervalos de comunicação (10 segundos), ao passo que estas diferenças são mais visíveis com a utilização de uma distribuição exponencial e intervalos menores (2 segundos). Porém, reiterando, através desta análise inicial, não é possível realizar nenhuma afirmação em relação a qual dos algoritmos de detecção de falhas possui desempenho superior, dado que os intervalos de confiança obtidos não foram conclusivos para um nível de confiança de 95% (ou mesmo 90% conforme também verificado).

3.4 Considerações

Através dos experimentos foi possível observar que o FD-Sensi gera uma quantidade consideravelmente menor de falso-positivos e apesar dos ICs obtidos para os ganhos relativos em relação ao *Adaptive Accrual* não permitirem afirmar que o FD-Sensi é melhor (salvo certos cenários específicos), é possível ainda afirmar que o desempenho do FD-Sensi é tão bom quanto do *Adaptive Accrual*, não sendo possível diferenciá-los com um nível de confiança de 95%, consistindo num resultado promissor.

Foi também observado que a utilização de um intervalo de *heartbeat* menor fez com que os atrasos simulados em seu recebimento tivessem um impacto maior no desempenho dos detectores de falha estudados, especialmente quanto ao tempo médio de detecção. As comparações de desempenho realizadas também apresentaram alguns resultados interessantes, como por exemplo o fato de que a utilização da distribuição exponencial faz com que os algoritmos degradassem seu desempenho, gerando mais falso-positivos que nos experimentos com a distribuição normal e também fazendo com que ambos os algoritmos detectassem as falhas num tempo médio maior. Isso dá sinais de que a utilização de uma carga de trabalho que reflita cenários reais onde os atrasos não são normalmente distribuídos pode trazer resultados mais interessantes.

De fato, a utilização de uma distribuição não-normal fez com que os detectores de falhas tivessem seu desempenho degradado. Além disso, a utilização de cargas sintéticas sempre deixa margem para discussão da validade dos resultados perante cenários reais. Por exemplo, tal como Satzger et al. [2007] utilizamos a injeção de aleatória de perdas de mensagens, porém, num cenário real é conhecido de que as falhas possam ocorrer em rajadas (*bursts*). Cientes disso, pensamos no potencial de se utilizar uma carga de trabalho que represente de maneira mais fiel os atrasos e perdas observados no mundo real, seja através de simulações que utilizem uma distribuição mais adequada ou ainda *traces* de algum ambiente distribuído real, donde optamos por este último.

Um bom exemplo de ambiente globalmente distribuído é o PlanetLab, um consórcio de diversas instituições do qual a UFMG participa. O PlanetLab oferece uma infra-estrutura distribuída em vários continentes e compartilhada entre as instituições integrantes. Através dele, não somente é possível o uso de vários nós distribuídos para o teste de novas aplicações e protocolos, como também é possível analisar os efeitos de se utilizá-los em grandes distâncias e nas condições reais de tráfego da Internet [Redígolo et al., 2007]. Assim, ao utilizar tal plataforma é possível realizar experimentos que possibilitem uma análise mais próxima da realidade.

Capítulo 4

Coleta e Análise do *Trace* do PlanetLab

Neste capítulo forneceremos uma visão geral do ambiente distribuído PlanetLab e após será detalhado como a análise experimental foi conduzida: o modelo de sistema adotado, a metodologia de coleta e análise dos dados e da avaliação de desempenho.

4.1 O Ambiente Distribuído PlanetLab

O PlanetLab é uma plataforma geograficamente distribuída para implantação, avaliação e acesso a serviços de rede de escala planetária. Consiste numa rede *overlay* global para o desenvolvimento e acesso a serviços de rede de larga cobertura, ou seja, que se beneficiem em estar largamente distribuídos na Internet. Uma rede overlay pode ser descrita, basicamente como uma rede lógica formada pela segregação de diversos elementos de sua rede base (ou substrato), a qual é então usada como canal de comunicação para a rede lógica formada pela agregação dos nós segregados [Redígolo et al., 2007]. A plataforma do PlanetLab é compartilhada, construída e mantida por uma comunidade de milhares de pesquisadores em dezenas de países. Em troca por hospedar um pequeno número de nós (servidores), os participantes obtém acesso a recursos compartilhados através da plataforma para que possam implantar e avaliar serviços de rede de escala planetária.

Em Março de 2002 o PlanetLab foi proposto como um *testbed* para serviços de rede de escala planetária e em Outubro do mesmo ano era lançada a versão 1.0. Nesse mesmo ano os dois primeiros nós brasileiros vieram da Universidade de Princeton para o Departamento de Ciência da Computação da UFMG. Naquela época, o PlanetLab consistia em 100 nós ao longo de 42 locais. Em Janeiro de 2004 a versão 2.0 foi

lançada e em Agosto mais nós chegaram ao Brasil, doados pela HP e Intel à Rede Nacional de Ensino e Pesquisa (RNP). Em Junho de 2007 foi superada a marca de 800 nós, sendo que atualmente o PlanetLab consiste em mais de 1000 nós em quase 500 locais espalhados pelo globo (Américas, Europa e Ásia, dentre outros). Desde o início de 2003, mais de 1000 pesquisadores das mais renomadas instituições acadêmicas e laboratórios de pesquisa industrial têm utilizado o PlanetLab para o desenvolvimento de novas tecnologias para armazenamento distribuído, mapeamento de rede, sistemas P2P, tabelas *hash* distribuídas e processamento de requisições [PlanetLab, 2008].

Um dos requisitos primários da arquitetura do PlanetLab é que ela proveja capacidade ao sistema de evoluir eficientemente, de maneira flexível e dirigida pelos requisitos atuais da comunidade e não por uma "visão idealista" de como o ambiente deveria ser. Desta maneira, é possível prover uma plataforma na qual novos serviços de rede serão implementados e servirão a uma comunidade de usuários reais, catalizando assim a evolução da Internet numa arquitetura orientada a serviços [Peterson & Roscoe, 2006]. Observe que por ser desenvolvida e utilizada pela comunidade de pesquisa, o PlanetLab não precisa necessariamente considerar uma gama completa de escalabilidade, segurança e questões de autonomia, que seria necessária a uma plataforma que visasse atender os usuários da Internet. A curto prazo, o objetivo inicial do PlanetLab é possibilitar a experimentação de serviços largamente distribuídos e a médio prazo é objetivado suportar também serviços que executem continuamente, potencialmente disponibilizados para uma comunidade de clientes [Chun et al., 2003]. Ainda, o PlanetLab poderá ser visto a longo prazo como um "microcosmo" para a Internet da próxima geração.

Um objetivo explícito e importante característica do PlanetLab é que é permitido a organizações independentes (grupos de pesquisa) implementar serviços alternativos em paralelo, permitindo aos usuários escolherem aqueles que mais lhe convenham [Peterson & Roscoe, 2006]. Isto é válido tanto para serviços voltados para usuários finais quanto para serviços de infra-estrutura utilizados para gerenciar e controlar o PlanetLab em si. Assim, múltiplos serviços competidores estão implementados no PlanetLab e evoluem simultaneamente. O PlanetLab busca maximizar a oportunidade de serviços competirem uns com os outros, mas num nível amigável. Porém, há de se decidir entre funcionalidades que possam ser implementadas por sistemas competindo paralelamente versus mecanismos que por sua natureza só devem ser implementados uma única vez. Estes últimos são evitados, mas resta um pequeno núcleo de funcionalidades que devem ser únicas no nó [Peterson & Roscoe, 2006].

O PlanetLab já oferece diversos serviços inovadores, cada qual executado em

um *slice*[1] próprio. Tais serviços visam tanto oferecer suporte aos experimentos em execução em outros *slices* como oferecer serviços distribuídos em escala global para usuários externos do PlanetLab [Redígolo et al., 2007]. Uma breve lista destes serviços é exibida a seguir:

CoDeeN: rede de distribuição de conteúdo (CDN)

Coral: rede par-a-par para Web caching

CoBlitz: transferência de arquivos de grandes dimensões de forma escalável

OpenDHT: serviço de tabela *hash* distribuída

OASIS: provê a seleção do "melhor servidor" baseado na localização, disponibilidade e carga

i3: rede *overlay* para roteamento

TMesh e ESM: redes par-a-par para a transmissão de áudio e vídeo

ScriptRoute: medições de tráfego de máquinas arbitrárias a partir dos nós do PlanetLab

4.1.1 Serviços de Monitoramento e Descoberta

Além dos diversos serviços em operação atualmente no PlanetLab (CDNs, DHTs, DNS *Resolution*, *Routing Overlays*, etc), há uma seção que vale ser ressaltada: os serviços de monitoramento e descoberta de nós, utilizados para monitorar o comportamento de nós e dos *slices* a que pertencem, bem como também auxiliar os usuários a descobrir quais recursos estão disponíveis em quais nós. Dentre eles, podemos citar:

- MON: um sistema leve e resiliente para gerenciar aplicações distribuídas, permitindo aos usuários executar comandos de gerenciamento tais como verificar o estado da aplicação e iniciar/parar um processo em um nó distribuído [Liang et al., 2005].
- CoMon: provê estatísticas de monitoramento tanto a nível dos nós quanto dos *slices*, tais como processamento, memória, limites de disco e banda de rede, dentre outros. Além disso, também relata sobre alguns problemas tais como " o SSH não responde há mais de uma hora", podendo indicar falhas nos nós/*slices* [Park & Pai, 2006].

[1] *slice* é a coleção de máquinas virtuais a qual o usuário tem acesso.

- PlanetFlow: arquiva o tráfego de saída proveniente de atividades de todos os nós do PlanetLab, sendo utilizado para auditorias e direcionamento das reclamações aos responsáveis pelo tráfego [Huang et al., 2006].
- SWORD: um serviço escalável para descoberta de recursos em sistemas largamente distribuídos. Ele coleta as informações disponíveis sobre os recursos dos nós e responde as requisições de usuários interessados de acordo com o critério definido (carga, memória livre, espaço em disco livre, latência entre nós) [Oppenheimer et al., 2005].

4.1.2 Considerações

Segundo Redígolo et al. [2007], a possibilidade de uso do PlanetLab, não apenas como bancada de testes e pesquisas, mas também como plataforma de implementação pode ser ressaltada como um dos principais pontos de diferenciação desta iniciativa, especialmente quando comparadas a *"overlays"* de arquitetura assemelhada. Enquanto os pesquisadores têm, de seu ponto de vista, acesso a um grande número de nós geograficamente distribuídos em ambiente experimental realista (ex. congestionamentos e falhas em uma topologia complexa), devido às características estruturais intrínsecas (ex. a dimensão física e componencial), estes têm, também, a possibilidade de exposição de seu experimento a uma carga de trabalho realista.

O PlanetLab é portanto formado por um conjunto de nós em diferentes sítios geograficamente distribuídos, apresentando *hardware*/infra-estrutura de rede heterogêneos e experimentando tanto sobrecarga como sub-utilização dos *slices*. Assim, é de se esperar que o comportamento de rede dos nós do PlanetLab seja bastante variável com o tempo, obviamente não sendo eficiente estipular *timeouts* fixos para um detector de falhas num cenário tão dinâmico. Até onde pude verificar, percebe-se uma lacuna no provimento de um serviço mais fidedigno de detecção de falhas para o PlanetLab. O mais próximo disso é o CoMon [Park & Pai, 2006], que fornece informações sobre problemas nos nós que não respondem via SSH por mais de uma hora. Porém, há sistemas que necessitam de um intervalo consideravelmente menor e uma maior precisão na detecção das falhas e aí reside uma oportunidade para o projeto da eficiente detecção de falhas no PlanetLab. Além do mais, neste ambiente distribuído há o objetivo explícito de que sejam implementados serviços alternativos em paralelo, permitindo aos usuários escolherem aqueles que mais lhe convenham [Peterson & Roscoe, 2006]. Sendo assim, mesmo que haja algum serviço que proveja algum nível de detecção de falhas, pode-se implementar um outro serviço, que possivelmente irá atender a uma gama de usuários com necessidades diferentes.

4.2 Coleta do *Trace*

A avaliação do desempenho de um sistema (computacional) é significativa somente no contexto de uma carga de trabalho e a utilidade de um modelo de carga depende do quão bem o modelo sintético representa o efeito observado em cargas reais e se ele tem poder preditivo. Cargas são válidas na medida em que permitem avaliar corretamente o comportamento do sistema no que diz respeito às características de interesse. Segundo Smith [2007], na medida em que o desempenho é sensível a mudanças na carga de trabalho, para outros cenários de carga as estimativas de desempenho serão pouco confiáveis ou então o modelo ou metodologia de estimativa serão considerados suspeitos. Ou seja, os resultados de desempenho obtidos para uma determinada carga de trabalho são válidos para o contexto daquela carga.

Conforme observado na análise experimental com cargas sintéticas (seção 3.2), a distribuição dos atrasos têm um grande impacto no desempenho dos algoritmos de detecção de falhas. Utilizando uma distribuição normal os algoritmos de detecção se comportam de uma forma, mas ao utilizar uma distribuição com maior dispersão como a exponencial, o desempenho dos mesmos é consideravelmente degradado. Ao invés de checarmos o desempenho dos algoritmos em outras distribuições de carga que talvez melhor representassem os atrasos percebidos na Internet, resolvemos realizar a coleta de um *trace* no PlanetLab, de maneira a obter uma carga real que representasse exatamente os tipos de situações que os algoritmos de detecção iriam enfrentar: congestionamentos de rede, enfileiramento em *buffers* de roteadores, sobrecarga das máquinas, etc.

Segundo Redígolo et al. [2007], dentre os mecanismos de obtenção de conhecimento, o método científico mostra-se, não apenas como o de êxito mais provável, mas como o mais convincente. O empirismo, cerne do método científico, consiste na investigação controlada de um fenômeno, organizada na forma de experimentos. Com efeito, os resultados positivos de um experimento prestam-se ao intuito de comprovar ou ao menos, reforçar uma hipótese formulada e investigada na condução deste. Tendo isto em mente, propomos a utilização de um *trace* na análise dos detectores, possibilitando a investigação controlada e a repetibilidade dos experimentos, de maneira que comparações de desempenho possam ser realizadas num cenário real. Além disso, uma carga real possibilitaria uma análise de desempenho considerando também efeitos inesperados, resultantes da interação entre sistemas complexos, produzindo resultados com alto valor científico e válidos como cenários reais.

Coletas anteriormente realizadas no PlanetLab [Tang et al., 2007] não apresentam um intervalo entre *pings* suficiente para direcionar o projeto de um detector de falhas

para o Planetlab: 20 minutos na coleta de Pathak et al. [2008], 15 minutos no APP [Stribling, 2005; Yoshikawa, 2006] e 5 min no CoMon [Park & Pai, 2006]. Portanto, foi desenvolvido um programa para realizar a coleta dos atrasos de rede a intervalos menores entre diversos nós do PlanetLab, durante um período de tempo prolongado. Uma coleta de 10 dias foi realizada de 27/05/2009 a 05/06/2009 em diversos nós do Planetlab, escolhidos de maneira a apresentar uma boa distribuição geográfica. Por questões de escalabilidade e sobrecarga do slice, a quantidade de nós foi limitada a 100 nós com um intervalo de coleta de 10 segundos. Resolvemos realizar tal coleta para aproximar nossa análise da realidade, tornar os resultados reproduzíveis por outros e por fim, para prover um *trace* que possa ser utilizado na avaliação de novos algoritmos de detecção de falhas ou mesmo outras aplicações não-correlatas.

4.2.1 Arquitetura do Coletor

O RTT (*round-trip time*) entre dois nós A e B é definido como sendo a soma do atraso direto de A para B e do atraso inverso de B para A. Ele tem sido utilizado largamente como métrica em diversas aplicações de rede que necessitam inferir atrasos e/ou localização dos nós. Normalmente assume-se o valor desses atrasos individuais como sendo 1/2 RTT, porém a existência de assimetrias nesses atrasos individuais prejudica a precisão e desempenho das aplicações que utilizam-se deste pressuposto [Pathak et al., 2008]. Portanto, se possível, a utilização dos atrasos individuais de cada direção do enlace é desejável por trazer consigo benefícios decorrentes de sua maior precisão.

Para a realização da coleta dos atrasos de rede e dados de utilização de recursos dos nós no PlanetLab foi necessário realizar o projeto e implementação de um software próprio, visto que o *ping* (*ICMP echo request*) não nos proveria as informações necessárias para inferir os atrasos individuais de cada direção da comunicação. Optamos por utilizar uma abordagem "*Three-Way-Ping*", de maneira a ter acesso ao RTT nos dois sentidos em momentos próximos.

O *Three-Way-Ping* (TWP) funciona da seguinte maneira: o nó A envia periodicamente solicitações ao nó B, através de mensagens UDP. A solicitação inicial é denominada PING, ao passo que o nó B responderá um PING-ACK. Ao receber a resposta do nó B, o nó A devolve um ACK para fechar a rodada de troca de mensagens deste protocolo. Assim, o TWP possibilita que em um nó A qualquer calculemos o RTT de A para B como sendo o intervalo de tempo entre a emissão um PING e o recebimento da resposta PING-ACK. Para praticamente o mesmo momento, é também possível calcular em B o RTT de B para A como sendo o intervalo entre a emissão de

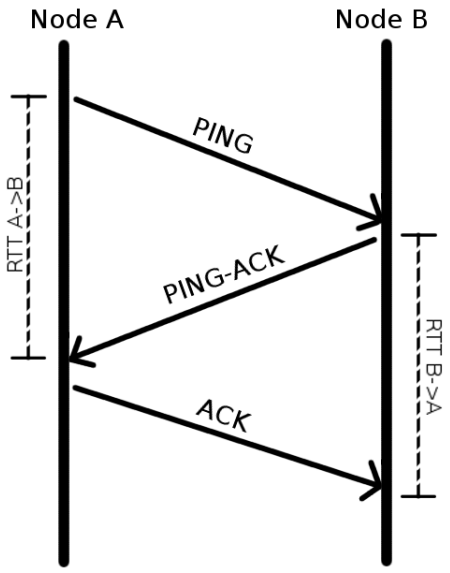


Figura 4.1. Sequência de Troca de Mensagens no TWP

um PING-ACK e o recebimento do ACK, conforme ilustra a figura 4.1.

Caso ambos os nós A e B estejam com seus relógios sincronizados, é possível também calcular os atrasos individuais se estes foram de ordem de grandeza superior ao escorregamento dos relógios (*clock-drift*). Segundo Pathak et al. [2008], os relógios dos nós do PlanetLab não escorregam muito de seus servidores NTP: 60% apresentam um escorregamento de menos de 2 milissegundos e 40% apresentam um erro máximo estimado pelo NTP de menos de 10 milissegundos. O atraso direto de A para B é calculado subtraindo-se o tempo de chegada da solicitação PING em B do seu tempo de envio em A. O atraso inverso também poderá ser calculado ao subtrair-se o tempo de chegada da resposta PING-ACK em A do seu tempo de envio em B. Ademais, uma estimativa da precisão do atraso direto pode ser realizada se for considerado também o atraso direto disponibilizado pelos tempos de envio e chegada da confirmação final ACK. Estes atrasos individuais serão utilizados na análise da assimetria dos atrasos (conforme será visto na seção 4.3.4), porém não são necessários ao funcionamento do FD-Sensi uma vez que ele não assume sincronização dos relógios. Para o FD-Sensi basta o intervalo entre dois TWPs, mais especificamente o intervalo entre dois PINGs, consistindo numa aproximação razoável entre o recebimento de dois *heartbeats*.

Implementamos o TWP em Java, como um software para coleta dos atrasos de rede no PlanetLab. Cada mensagem do protocolo TWP consiste num datagrama UDP contendo em seu cabeçalho o endereço e porto do destino e como dados 1 byte especificando o tipo de mensagem (PING, PING-ACK ou ACK) e 2 bytes para o número de

sequência. Este número de sequência é necessário para a ordenação das mensagens no *trace*, provendo mais de 65 mil identificadores em cada ciclo de incrementação de seu valor. Isso é suficiente para diferenciar as mensagens, pois aquelas de mesmo identificador de sequência estarão muito espaçadas temporalmente. Este processamento é o mesmo feito pelo TCP, por exemplo.

No ato de envio e recebimento das mensagens, cada nó armazena em seu *trace* a quíntupla:

- *timestamp* (8 bytes)
- número de sequência (2 bytes)
- tipo de mensagem (1 byte)
- ID do nó fonte (2 bytes)
- ID do nó destino (2 bytes)

Por economia de espaço, tais dados são armazenados não em ASCII mas como binários, com cada evento de envio ou recebimento ocupando apenas 15 bytes no *trace*. Os IDs dos nós fonte e destino não consistem em seus endereços de rede, mas num identificador provido pela sua respectiva ordem numa java.util.Hashtable contendo todos os nós e seus respectivos portos. Assim, após a tabela *hash* estar preenchida com todos os nós do experimento, obtém-se uma listagem do conjunto de chaves, onde o ID de cada nó será a posição dele neste conjunto. Com isso foi possível economizar no tamanho do *trace*, pois 2 bytes a menos em centenas de milhões de mensagens armazenadas diminuem em alguns GB no tamanho total do *trace*. Além disso, a utilização dos IDs incrementais facilitou o processamento da análise dos dados através de laços (*loops*) com contadores.

Para economizar na quantidade de mensagens trocadas evitamos que haja duplicação de esforços. Fazemos isso de maneira que um nó somente realize solicitações a metade do conjunto de nós e consequentemente, responda às solicitações da outra metade do conjunto. Para tal utilizamos um iterador circular sobre o conjunto de chaves da tabela *hash*. Assim, a cada intervalo de comunicação um nó A somente ou enviará ou receberá requisições TWP de um outro nó B. Isso é possível pois a sequência das chaves é a mesma em todos os nós.

Para coordenar os nós participantes da coleta, foi necessário eleger um servidor para controlar a execução do experimento e coletar os resultados de cada nó participante. Tal coordenador é especialmente crítico nos momentos iniciais do experimento, onde apesar de termos definido um porto padrão para a troca de datagramas, nem

sempre ele estará disponível no nó. Portanto, assim que um nó instancia o soquete no qual ouvirá as requisições TWP ele envia seu porto ao coordenador e espera pela autorização para o início da troca de mensagens. Por sua vez, ao receber todos os portos dos nós participantes, o coordenador replica a lista de portos para cada um dos nós e após todos terem acusado seu recebimento, dá a autorização ao início do experimento. Ao final do experimento, o coordenador envia um comando para os nós participantes finalizarem a troca de mensagens e enviarem a ele os resultados restantes.

Porém, contando com a possibilidade de perda de dados, ao invés de coletar os resultados só ao final dos vários dias de experimento, optamos por, a cada intervalo de uma hora, cada nó participante passaria a armazenar os dados coletados em um novo arquivo com um nome incremental. Após isso o arquivo anterior seria compactado e enviado ao coordenador, que ao final do experimento iria ordenar e reunir os pedaços do *trace* de cada nó em um arquivo único. O *daemon* que realiza o envio dos dados ao servidor foi projetado para evitar que todos os nós enviem seus dados ao mesmo tempo. Além da possibilidade de sobrecarregar o coordenador, isto poderia gerar atrasos altamente correlacionados durante os envios simultâneos, portanto, adicionamos um tempo aleatório antes de cada ciclo de envio. Observe que apesar da compactação e envio das partes do *trace* influenciarem momentaneamente na carga de processamento e utilização da banda no nó, isso não é de todo condenável pois desejamos observar os atrasos de rede em situações de utilização real das máquinas, inclusive em condições de sobrecarga. O valor do intervalo de envio foi selecionado de maneira que possibilite um bom compromisso entre um tempo de transmissão pequeno e um tamanho/quantidade de arquivos que compense a compactação.

Além dos dados acerca dos atrasos, armazenamos também os seguintes dados de utilização de recursos informados pelo CoTop (*daemon* que faz parte do CoMon e responde no porto 3121):

- carga média dos últimos 1, 5 e 15 minutos
- % de utilização de CPU
- % de utilização de memória principal
- vazão de banda de *upstream* e *downstream*

Estes dados foram coletados em cada nó a um intervalo igual ao intervalo de requisições TWP, também armazenado como binário, mas em um arquivo à parte. Ao todo, juntamente com o momento no qual a coleta foi efetuada (*timestamp*), cada entrada de utilização de recursos ocupou 120 bytes. E tal como os arquivos contendo

os TWPs, também foram divididos em arquivos incrementais e enviados a cada uma hora e 15 minutos ao coordenador.

Caso algum nó fosse reiniciado durante o experimento, ele comunicaria-se com o coordenador para obter a lista dos portos dos demais nós participantes e voltaria a trocar requisições TWP com eles.

4.3 Resultados da Análise do *Trace*

4.3.1 Filtragem

Numa análise do progresso evolutivo do PlanetLab, Tang et al. [2007] observou que na maior parte do tempo, mais de 80% dos nós estavam online, sendo que certos conjuntos de nós apresentavam uma confiabilidade maior que os demais. Tipicamente o *uptime* dos nós do PlanetLab é em média 5 dias [Zhang & Honeyman, 2008] e dos 100 nós que selecionados para nossa coleta 14 foram reiniciados durante a execução do experimento. É importante observar que a precisão dos atrasos individuais (e sua associação com os dados de carga dos nós) depende criticamente da sincronização dos relógios: quando um nó é reiniciado, seu relógio não está sincronizado e consequentemente as suas medidas não são utilizáveis; após ser reiniciado, o nó contacta o servidor NTP e corrige seu relógio, mas este ajuste consome algum tempo [Pathak et al., 2008], inutilizando as medidas realizadas até aquele momento.

Além dos 14 nós reiniciados, outros 5 nós tiveram problemas ao contactar o servidor NTP. Decidimos então por remover de nossas análises estes 19 nós problemáticos, sendo utilizados os *traces* dos 81 nós restantes. A lista completa com os 19 nós desconsiderados e os outros 81 considerados na análise encontra-se nos apêndices, através das tabelas B.1, B.2 e B.3. Nos 81 *traces* a analisar, ignoramos os períodos iniciais e finais do experimento para evitar valores discrepantes nas perdas e atrasos devidos a soquetes fechados, transferência dos últimos resultados e afins. Adicionamos uma faixa de segurança de 3 horas após e antes do final do experimento ao carregar os *traces* para sua análise.

4.3.2 Estatísticas do *Trace*

O tamanho médio dos *traces* colhidos foi de 85000 ($\simeq$3600s*24h*10d/10s) *Three-Way-Pings*/nó, o que dá cerca de 255000 mensagens trocadas por cada par de nós (considerando-se que cada TWP gera três mensagens: PING, PING-ACK e ACK). Como cada nó troca TWPs com os (N-1) outros nós, ao todo os *traces* coletados nos

81 nós "sadios" possuem 81*80=6480 enlaces de comunicações entre dois nós. Porém, como vimos na seção 4.2.1, cada nó envia TWPs a N/2 nós e responde da outra metade. Assim, ao todo coletamos cerca de 85000*(6480/2)≃275 milhões de TWPs ou 826 milhões de mensagens trocadas durante os 10 dias da coleta. Conforme explicado na seção 4.2.1, através dos tempos das mensagens contidas no *trace* cada troca de TWPs possibilita calcular 2 RTTs. Calculamos assim cerca de 551 milhões de medidas de RTTs. Ao todo, os 81 *traces* binários ocupam 20 GB de espaço em disco, que compactados com o GZip caíram para 7 GB. Após o término da coleta, realizamos um recompactamento utilizando o 7Zip, que resultou numa queda para 4 GB ocupados em disco.

Conforme visto na seção 4.2, a coleta foi realizada dos dias 27/05/2009 a 05/06/2009, de maneira que o meio da coleta coincidiu com um fim de semana, conforme observa-se na tabela 4.1.

Dia	**1º**	**2º**	**3º**	**4º**	**5º**	**6º**	**7º**	**8º**	**9º**	**10º**
Data	27/05 qua	28/05 qui	29/05 sex	30/05 sáb	31/05 dom	01/06 seg	02/06 ter	03/06 qua	04/06 qui	05/06 sex

Tabela 4.1. Datas de Realização da Coleta

Para o RTT, foi obtida uma média de 181 ms e conforme pode ser observado na figura 4.2, 95% dos nós apresentaram uma média abaixo de 386 ms, com um máximo de 537 ms para o RTT médio. A figura 4.3 exibe a evolução diária do RTT médio ao longo do *trace*. Nela é possível observar que do 2º ao 6º dia houve um período de queda no RTT médio, que foi de 183 ms a 169 ms, correspondendo ao fim de semana. Após este período observa-se uma breve subida do 7º ao 8º dia para o valor de 177 ms e uma nova queda até o fim da coleta no 10º dia, alcançando então o menor valor para o RTT médio: 168 ms. As barras verticais consistem os intervalos com 99% de confiança na média calculada.

Consideramos como perda de mensagem os PINGs (e PING-ACKs) enviados mas não respondidos e podemos afirmar com exatidão qual mensagem não foi respondida pois todas elas recebem um número de sequência (conforme explicado na seção 4.2.1). Na figura 4.4 podemos observar que em média, a taxa de perda de mensagens foi de 0,41% apenas, sendo que grande parte dos nós apresentou uma baixa taxa de perda entre 0,02% e 0,26%. Foi observado que 90% dos nós apresentaram perdas abaixo de 0,90% e 95% abaixo de 3,07%, com um máximo de 8,55% de perda de mensagens.

A figura 4.5 exibe a evolução diária da perda média de mensagens ao longo do

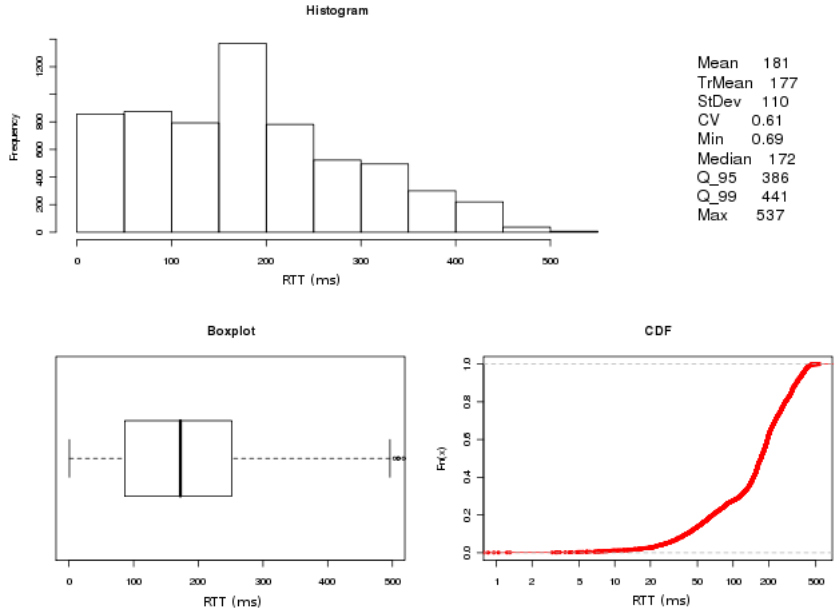


Figura 4.2. Resumo Estatístico do RTT Médio

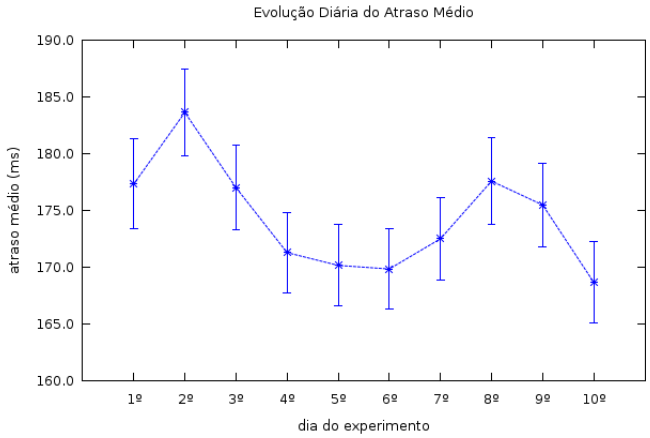


Figura 4.3. Evolução Diária do RTT Médio

trace. Observamos que a perda de mensagens apresentou uma maior variabilidade nos 3 primeiros dias da coleta, passando a demostrar valores mais homogêneos a partir do 4º dia. O 3º dia apresentou o maior valor global de 0,86% para a perda média de mensagens e o 9º dia apresentou o menor valor global de 0,18%. As barras verticais consistem os intervalos com 99% de confiança na média calculada.

O coeficiente de variação (CV) indica a variabilidade dos dados em relação à média, sendo que quanto menor o CV mais homogêneo é o conjunto de dados. Foram observadas altas variabilidades nos atrasos das mensagens, com um CV médio de 1,89, porém com grande parte dos nós apresentando variabilidades de 0,22 a 1,74 e 50%

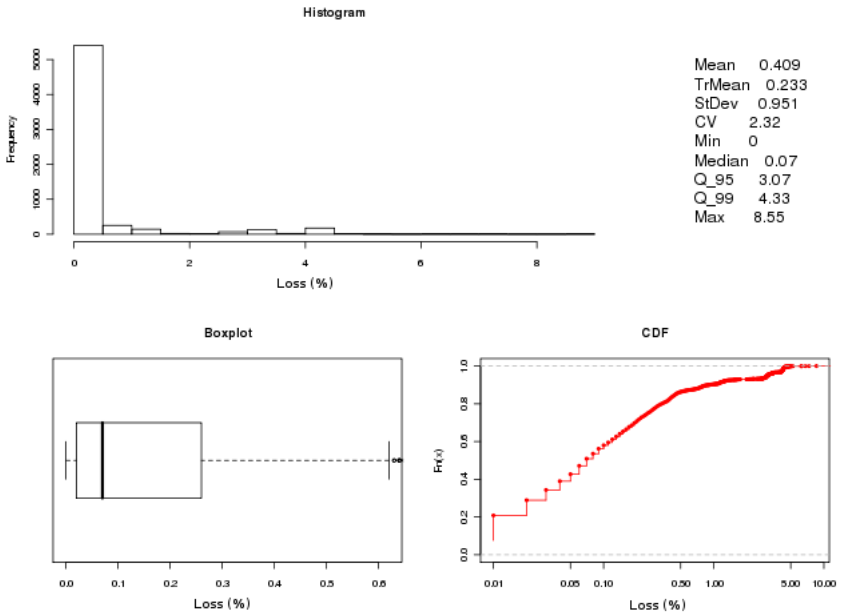


Figura 4.4. Resumo Estatístico da Perda de Mensagens

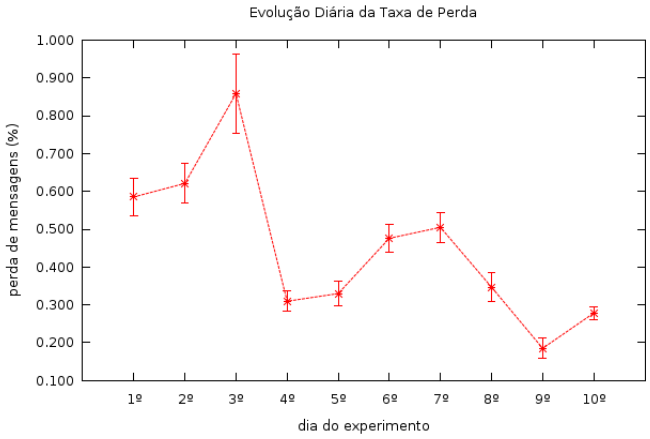


Figura 4.5. Evolução Diária da Perda Média de Mensagens

abaixo de 0,56, conforme pode ser observado na figura 4.6.

A figura 4.7 exibe a evolução diária da variabilidade média dos atrasos ao longo do *trace*, onde as barras verticais consistem os intervalos com 99% de confiança na média calculada. Nela é possível observar que, tal como observado no RTT médio, do 2º ao 8º dia houve um período de queda na variabilidade, seguido de uma subida do 8º ao 9º dia. Ou seja, ao mesmo tempo que o RTT médio decaiu, observou-se uma maior homogeneidade no coeficiente de variação dos atrasos do referido período.

Já acerca dos extremos do RTT, observamos que:

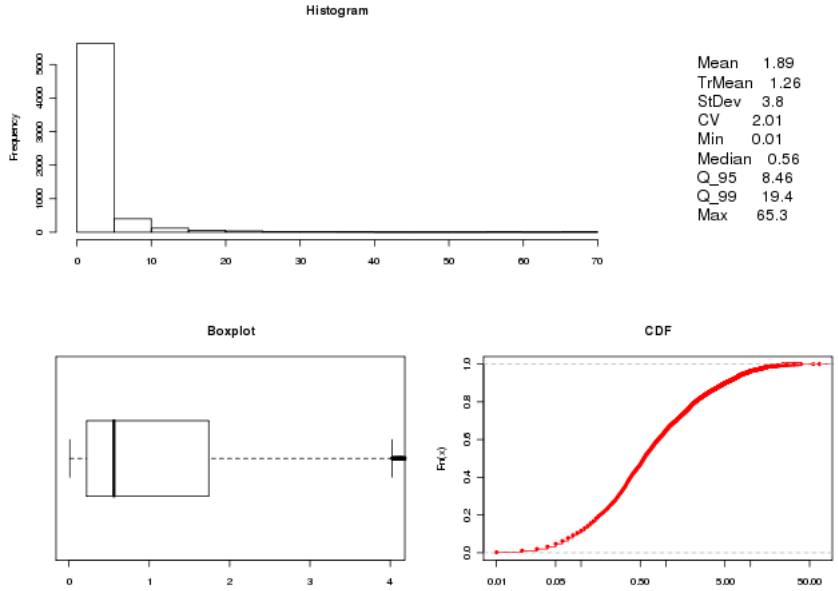


Figura 4.6. Resumo Estatístico da Variabilidade dos Atrasos

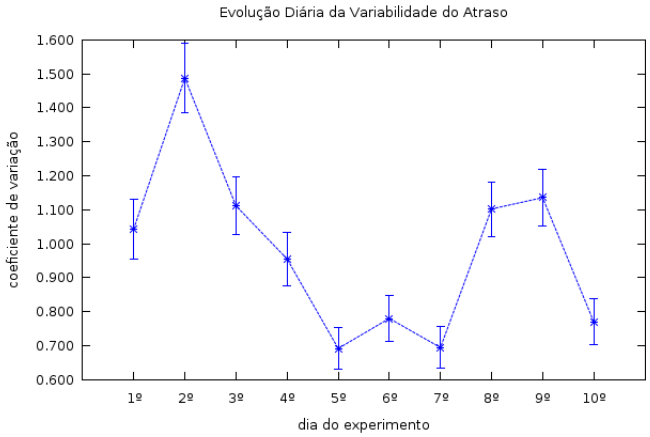


Figura 4.7. Evolução Diária da Variabilidade Média do RTT

- A média dos RTTs mínimos foi de 145 ms, com 75% dos nós abaixo de 198 ms e 95% abaixo de 330 ms. O mínimo absoluto foi de 1 ms (entre nós de uma mesma LAN) e valor máximo observado para o RTT mínimo foi de 409 ms.

- 75% dos nós apresentaram um RTT abaixo de 353 ms, 90% abaixo de 565 ms, 95% abaixo de 721 ms e 99% abaixo de 982 ms.

- A média dos RTTs máximos foi de 27 segundos, com 75% dos nós abaixo de 32 s e 95% abaixo de 112 s. O máximo absoluto foi de 170 segundos, ou seja, 2,83 minutos de atraso. Porém, em geral as mensagens seguintes aos atrasos

máximos não foram afetadas, ou seja, não foi observada uma rajada de atrasos nestes eventos.

Os valores muito altos do RTT como o máximo absoluto de 170 segundos foram verificados e neles foi percebida alteração nas próximas mensagens, ou seja, não consistiram numa rajada de atrasos mas em acontecimentos individuais. Observe que como as mensagens foram enviadas por UDP, não houve retransmissão daquelas perdidas, portanto os atrasos máximos observados provavelmente foram provenientes de congestionamentos na rede, roteamentos ineficientes e esperas em filas de roteadores. Consideremos o impacto de um atraso de mais de 2 minutos numa mensagem que seja esperada por um detector de falhas: caso este atraso seja um acontecimento individual poderá ser gerado um falso-positivo ou esperar-se-á pela próxima mensagem; caso o atraso esteja embutido numa rajada de atrasos, o detector que não considerar o histórico dos atrasos já recebidos muito provavelmente irá suspeitar que uma falha tenha ocorrido onde na verdade experimenta-se problemas de rede transientes ou uma tendência de sobrecarga.

A figura 4.8 exibe a evolução diária do 99-quantil médio ao longo do *trace*. Nos dois primeiros dias da coleta, 99% do RTT médio ficou abaixo de 406 ms, porém a partir do 3º ao 7º dia, houve uma queda brusca no 99-quantil médio, girando em torno de 220 ms, ou seja, quase a metade do valor inicialmente observado. Nesse período os dados também se apresentaram mais homogêneos. Após, seu valor subiu brevemente e voltou a descer, mantendo-se abaixo de 307 ms.

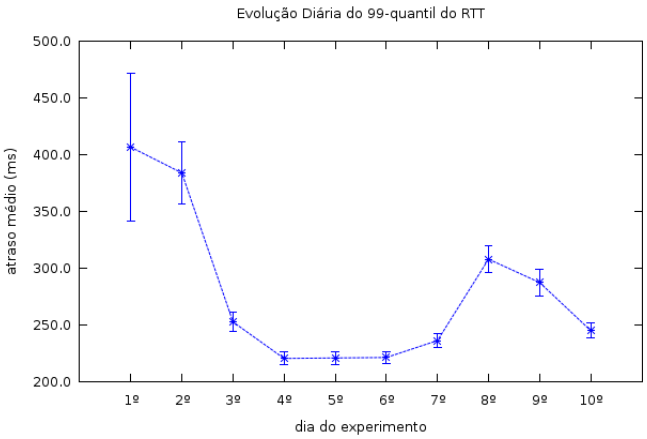


Figura 4.8. Evolução Diária do 99-quantil Médio do RTT

4.3.3 Distribuição dos Atrasos

Para determinar qual o melhor modelo de distribuição para os atrasos observados utilizamos o software Minitab [Ryan et al., 2005] para encontrar dentre as distribuições candidatas aquela que melhor se ajuste aos dados. Devido a limitações da quantidade máxima de linhas permitidas e do tempo necessário para o processamento, dos 551 milhões de medidas de RTT coletadas obtemos uma amostra de 0,1% ou 551 mil medidas.

Os melhores parâmetros da distribuição podem ser encontrados através de testes padrões para determinar o *best-fit*: Chi-square, Kolmogorov-Smirnov (KS) e Anderson-Darling (AD). Desses, o teste utilizado pelo Minitab foi o teste AD, também conhecido como um teste baseado na ECDF (*empirical cumulative distribution function*), consistindo num refinamento do KS [Stephens, 1974]. Há uma grande quantidade de distribuições que consistiriam em candidatos a modelo de distribuição dos dados. A tabela 4.2 exibe a qualidade do ajuste daquelas que se mostraram mais apropriadas para a modelagem de atrasos de rede. Na segunda coluna, o coeficiente AD fornece-nos uma medida relativa de qualidade do ajuste e portanto, quando compararmos o ajuste de múltiplas distribuições para o conjunto de dados, a distribuição com a menor estatística de Anderson-Darling oferecerá o melhor ajuste. Já o *p-value* correspondente (se disponível), caso seja menor que um determinado limiar escolhido (por exemplo, limiar=0,005), pode-se rejeitar a hipótese nula de que os dados vieram daquela distribuição.

Tabela 4.2. Qualidade do Ajuste da Distribuição dos Atrasos

Distribution	***AD***	***P-Value***
3-Parameter Lognormal	202	*
Gamma	207	<0,005
3-Parameter Loglogistic	214	*
Lognormal	267	<0,005
3-Parameter Weibull	272	<0,005
Weibull	471	<0,010
Normal	985	<0,005
2-Parameter Exponential	2376	<0,010

Para melhor ilustrar os resultados, as figuras 4.9 e 4.10 exibem os gráficos de probabilidade de ajuste das distribuições, consistindo numa técnica para avaliar se o conjunto de dados segue ou não uma dada distribuição. Os dados são plotados sobre a distribuição candidata de maneira que os pontos devam formar aproximadamente

uma linha reta e desvios na reta indicam desvios da distribuição especificada [Croarkin & Tobias, 2002]. Sendo assim, nosso critério para a distribuição com melhor ajuste é aquela com o gráfico de probabilidade mais linear. O coeficiente de correlação AD provê uma medida da linearidade do gráfico, assim como uma medida do quão bem a distribuição ajusta-se aos dados. O *p-value*, seria utilizado como critério de desempate entre duas distribuições, porém, como todas as distribuições apresentaram um *p-value* muito próximo, não foi possível utilizá-lo como critério de descarte.

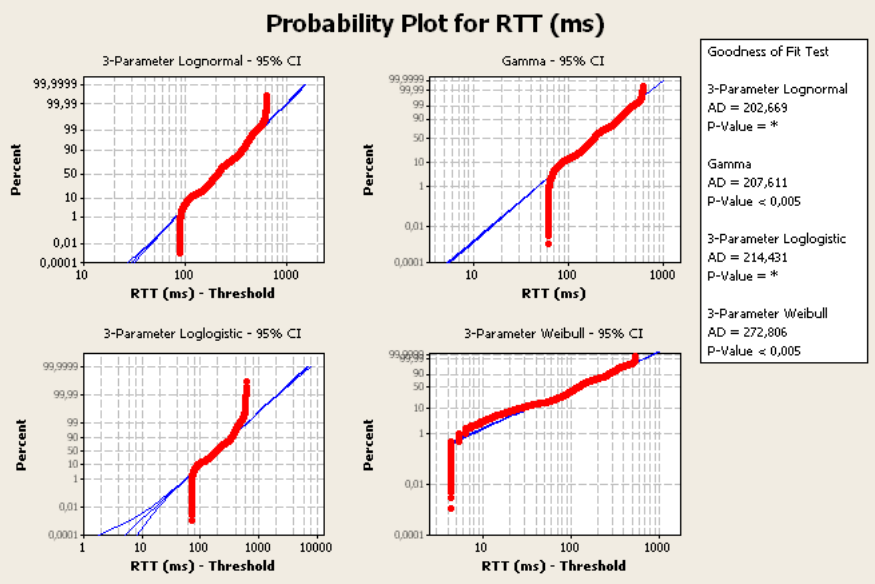


Figura 4.9. Gráfico de Probabilidades das Distribuições I

Dito isso, ao olhar para as figuras 4.9 e 4.10 verificamos que as distribuições *Lognormal de 3 parâmetros* e *Gamma* foram as que melhor se ajustaram aos atrasos observados. Das duas, a distribuição Gamma foi a que melhor se ajustou aos extremos (atrasos <100 ms e >500 ms). Portanto, propomos a utilização da distribuição Gamma por prover um balanço entre melhor ajuste, bons resultados nos extremos e simplicidade, consistindo assim num modelo de distribuição apropriado para a modelagem de atrasos na rede.

A distribuição Gamma é uma distribuição de probabilidade contínua que possui dois parâmetros: um parâmetro de escala Θ e um parâmetro de forma λ. Ela é frequentemente utilizada para modelar tempos de espera e aparece naturalmente em processos para os quais os tempos de espera entre eventos com distribuição *Poisson* são relevantes. Ela pertence à família das distribuições exponenciais com parâmetros naturais λ-1 e -1/Θ. A sua função característica φx(t) é $(1 - \Theta it)^{-\lambda}$, em que t é o argumento da função característica e i é a raiz quadrada de menos um.

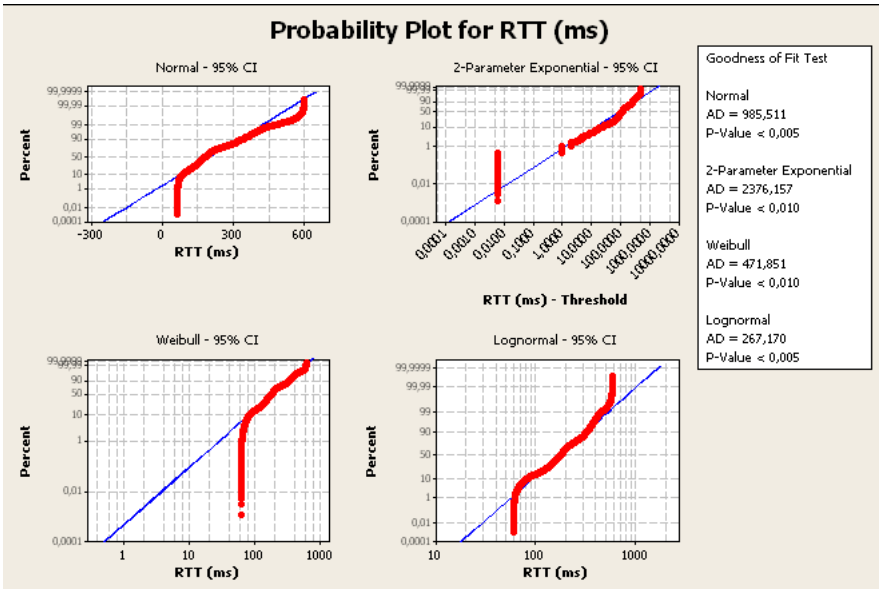


Figura 4.10. Gráfico de Probabilidades das Distribuições II

A figura 4.10 exibe o gráfico de probabilidades de outras distribuições que não obtiveram um ajuste tão bom. Nela, é possível observar que tanto a distribuição normal quanto a exponencial não provêem bons modelos para a distribuição dos atrasos, distribuições estas que são comumente utilizadas para esse fim. Perceba que até mesmo o uso de uma distribuição Weibull seria mais adequado do que a utilização de uma normal ou exponencial. Isso confirma a hipótese levantada na seção 3.3.3 de que a distribuição normal não é um bom modelo para os atrasos de rede.

Tabela 4.3. Estimativas dos Parâmetros da Distribuição dos Atrasos

Distribution	***Localização***	***Forma*** λ	***Escala*** Θ	***Limiar***
3-Parameter Lognormal	5,34111	-	0,41280	-27,12915
Gamma	-	4,63062	43,16537	-
3-Parameter Loglogistic	5,25100	-	0,26255	-9,23031
Lognormal	5,18588	-	0,48332	-
3-Parameter Weibull	-	1,51208	158,33886	56,63291
Weibull	-	2,25873	226,38200	-
Normal	199,88250	-	94,28796	-
2-Parameter Exponential	-	-	138,88860	60,99390

A tabela 4.3 exibe as estimativas dos parâmetros das distribuições para modelagem do atraso de mensagens, ordenadas pelos coeficientes de correlação (AD). Através dela é possível configurar as distribuições para exibirem atrasos de rede compatíveis com o PlanetLab. Portanto, este trabalho fornece tanto uma carga real quanto as dis-

tribuições estatísticas mais apropriadas para a modelagem dos atrasos de rede numa carga sintética. Segundo Silva et al. [2009], a geração de cargas sintéticas mais realistas também possibilitara analisar com mais precisão aplicações em contextos semelhantes ao do *trace* coletado, de maneira a facilitar o projeto, implementação e validação de novos protocolos de rede.

A figura 4.11 exibe as estatísticas descritivas da distribuição Gamma configurada com os parâmetros de melhor ajuste ao RTT do PlanetLab. Sua obliquidade positiva (*skewness*=0,927) faz com que sua massa seja concentrada a esquerda e exiba uma cauda mais longa a direita, demonstrando uma assimetria na dispersão dos dados em relação à curva normal (curva sobre o histograma). A curtose também positiva (*kurtosis*=1,294) caracteriza um menor achatamento da curva em relação à normal, ou seja, exibe um afunilamento com um pico mais alto e uma maior concentração que a normal. Consequentemente, a distribuição Gamma assim configurada possui caudas mais prolongadas, onde é relativamente fácil obter valores que se afastam da média a vários múltiplos do desvio padrão. Observe ainda que a média exibida é de 199 ms, ou seja, ligeiramente diferente da média global exibida pelo PlanetLab de 181 ms (isto é devido a imperfeições no ajuste da modelagem).

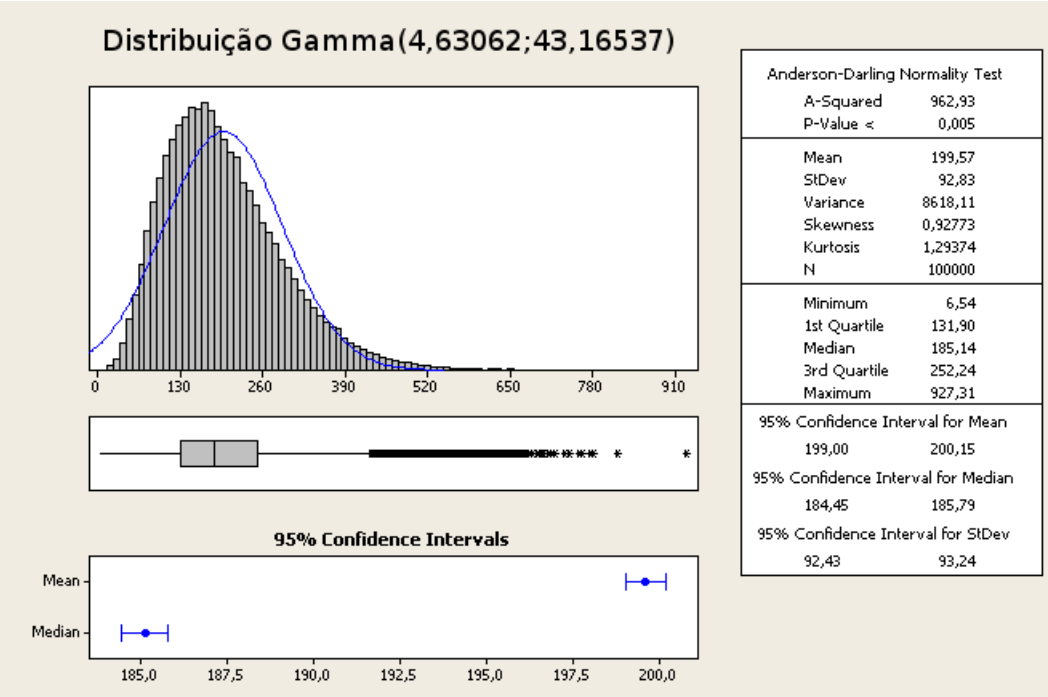


Figura 4.11. Distribuição Gamma para o RTT do PlanetLab

4.3.4 Assimetria dos Atrasos

Numa análise do progresso evolutivo do RTT (*round-trip time*) médio do PlanetLab, Tang et al. [2007] observou uma drástica variação no RTT entre pares de nós (de 150 a 550 ms), mesmo entre curtos períodos de tempo (15 minutos). Isso apresenta um desafio significativo aos modelos topológicos que pressupõem uma latência fim-a-fim estática entre pares de nós. Em recente análise da assimetria dos atrasos na Internet, Pathak et al. [2008] realizaram um compreensivo estudo no qual mediram a severidade da assimetria no atraso entre diversos pares de nós do PlanetLab. Seus achados estão de acordo com os de Tang et al. [2007], sendo observado que a assimetria dos atrasos prevalece na grande maioria dos nós, podendo ser em parte atribuída à assimetria nas rotas e em parte a congestionamentos transientes. Observaram também que a assimetria é dinâmica, ou seja, à medida que o tempo progride, a assimetria dos atrasos varia, e, além disso, foi notado que redes comerciais exibem níveis mais altos de assimetria que redes educacionais e de pesquisa.

Visto que a assimetria dos atrasos na Internet é uma realidade, conduzimos uma análise da assimetria do *trace* coletado. Utilizamos a métrica *assimetria relativa*, que consiste no módulo da diferença entre os atrasos direto e inverso normalizado pelo atraso mínimo, conforme representado na equação 4.1. Nesta equação, quanto mais próximo de zero for o resultado, mais simétrico será e consequentemente, quanto maior for o valor encontrado, maior será a assimetria.

$$assimetria = \frac{|atraso_{AB} - atraso_{BA}|}{min(atraso_{AB}, atraso_{BA})} \tag{4.1}$$

Por exemplo, considere que em determinado momento o RTT observado num par de nós foi de 150 ms e o atraso direto de 60 ms. Portanto, o atraso inverso foi de 90 ms o que representa uma assimetria de 30 ms entre esses dois atrasos. Com 60 ms de atraso direto e 90 ms de atraso inverso, temos uma assimetria relativa de abs(60-90)/60=0,50 . Isso significa que a direção com maior atraso apresentou um acréscimo de 50% em relação ao valor do atraso menor da outra direção.

Os resultados obtidos para métrica estão expressos na figura 4.12 e na tabela 4.4, onde podemos observar que a assimetria média do PlanetLab é alta (0,947), porém com uma alta variabilidade (9,27). Mesmo a média calculada com a exclusão de 5% dos valores discrepantes ainda pode ser considerada alta, com um valor de 0,409. Uma grande parte dos nós apresentou uma assimetria relativa entre 0,08 e 0,56 e 95% deles apresentaram uma assimetria abaixo de 3,08.

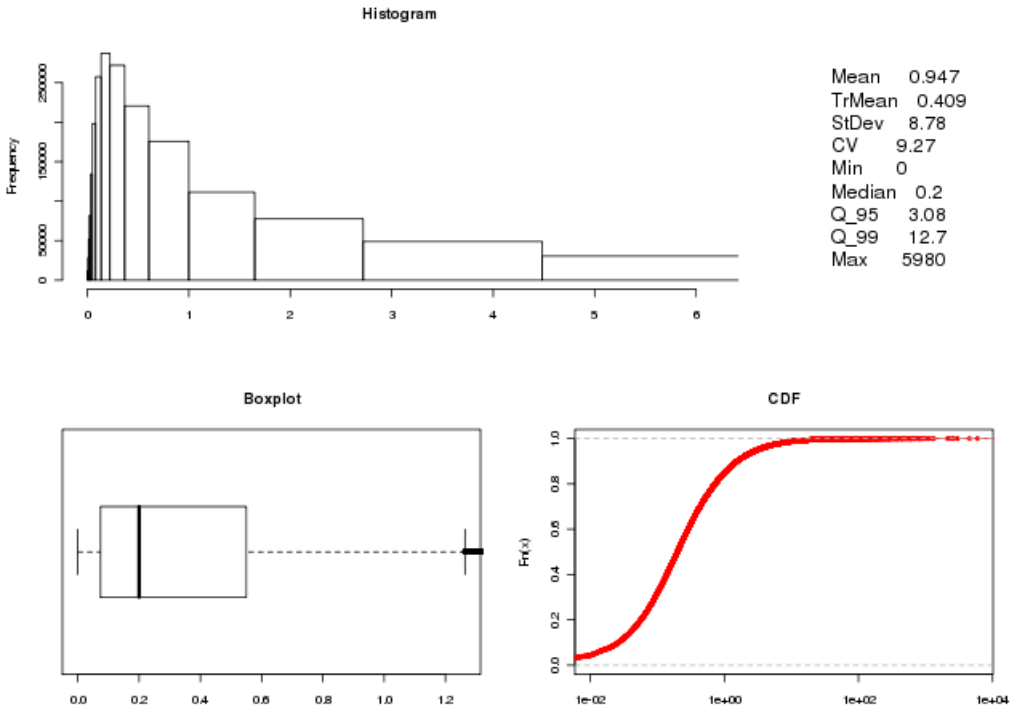


Figura 4.12. Resumo Estatístico da Assimetria Relativa dos Atrasos

$Mean$	$StDev$	CV	Q_{25}	$Median$	Q_{75}	Q_{90}	Q_{95}	Q_{99}
0,947	8,78	9,27	0,08	0,20	0,56	1,62	3,08	12,07

Tabela 4.4. Estatísticas da Assimetria Relativa dos Atrasos

4.3.5 Clusterização dos Enlaces

Realizamos também uma clusterização dos enlaces de acordo com suas estatísticas descritivas (atraso médio, variabilidade do atraso e seus quantis, % de perda), buscando separá-los em grupos que possuam valores aproximados para estatísticas como o atraso médio e taxa de perdas das mensagens, dentre outras. Para tal, utilizamos o software *Weka 3 Data Mining* [Witten & Frank, 2002] com o algoritmo de clusterização Expectation-Maximization (EM), donde chegamos às 5 instâncias de *clusters* listadas na tabela 4.5.

Observamos que o cluster 1 apresentou um baixo RTT médio e baixa perda de mensagens, ao contrário do cluster 5, que apresentou atrasos relativamente bem maiores e uma maior taxa de perda de mensagens. O cluster 2 mostrou uma grande variabilidade nos dados, dado seu alto CV, ao passo que o cluster 3 teve a menor variabilidade. Acerca do cluster 4, podemos dizer que ele apresentou um comportamento

Tabela 4.5. Estatísticas da Clusterização

Cluster	*RTT Mean*	*Loss Rate*	*RTT StDev*	*RTT CV*	*% Enlaces*
c1	49 ms	0,22%	46 ms	1,12	21%
c2	131 ms	0,40%	673 ms	6,37	21%
c3	167 ms	0,30%	55 ms	0,33	24%
c4	269 ms	0,12%	266 ms	0,96	21%
c5	358 ms	1,40%	158 ms	0,44	13%
Global	181 ms	0,41%	240 ms	1,89	100%

próximo da média global do PlanetLab. Os clusters serão utilizados mais adiante para agrupar os resultados da qualidade de detecção apresentada pelos algoritmos na seção de resultados 5.3.1.

Como cada cluster compartilha características próximas de comportamento, possivelmente os enlaces pertencentes a um mesmo cluster apresentarão as mesmas configurações ótimas para os parâmetros do algoritmo de detecção de falhas. De posse destas informações seria possível dividir os nós do PlanetLab por grupos de detecção, onde cada grupo estaria calibrado com o melhor custo-benefício entre tempo de detecção e falso-positivos.

4.3.6 Estatísticas da Utilização de Recursos

A tabela 4.6 exibe as estatísticas globais para os dados coletados da utilização de recursos, onde é possível observar que a maior parte dos nós encontravam-se sobrecarregados.

Tabela 4.6. Estatísticas da Utilização de Recursos

	LoadAvg (1 min)	*% CPU*	*% MEM*	*Throughput (kbps)*
Mean	69,12	76%	75%	4226
StDev	816,39	31	66	6678
CV	11,81	0,42	0,88	1,58
Min	0,01	0%	1%	0
Q_{25}	2,67	57%	62%	1221
Median	5,17	95%	69%	1285
Q_{75}	7,19	100%	77%	4297
Q_{90}	9,61	100%	83%	8625
Q_{95}	11,65	100%	85%	15362
Q_{99}	2399	100%	100%	34464
Max	82458	100%	100%	1146234

Mais de 50% dos nós apresentando uma carga média acima de 5 (1 min *load*

average), uso de CPU acima de 95%, uso de memória acima de 69% e 25% dos nós apresentando uma vazão de rede acima de 0,5 MB/s. Este perfil de utilização dos recursos é devido aos nós se encontrarem constantemente executando diversos *slices*, ou seja, diversas máquinas virtuais para os mais variados experimentos dos usuários do PlanetLab, de maneira que a utilização de recursos se mantém num patamar elevado. Isso pode ocasionar, dentre outras coisas, atrasos locais na entrega de mensagens, seja devido à espera na fila de processos da CPU ou na fila do *buffer* da interface de rede. Estes atrasos causados por sobrecarga das máquinas podem fazer com que o poder de predição dos algoritmos de detecção de falhas seja de certa forma prejudicado, degradando seu desempenho pois terão que lidar com diversos perfis de atrasos e perda de mensagens. A sobrecarga dos nós é inclusive apontada como um problema por muitos pesquisadores, que atribuem a esse fato dificuldade de se reproduzir experimentos naquela rede.

Capítulo 5

Aperfeiçoamento do FD Utilizando Dados de Carga

O fato das máquinas do PlanetLab exibirem um perfil de média a alta utilização de recursos nos motivou a considerar os dados de carga informados pelo CoTop (*daemon* que faz parte do CoMon), de maneira a buscar uma melhora na predição do algoritmo FD-Sensi. A figura 5.1 passa a ideia da influência da utilização de recursos do nó monitorado (em cores) no atraso das mensagens percebido pelo nó monitor (em preto). Ao observá-la, temos a impressão de que os grandes picos de elevação do RTT são acompanhados de elevações nas curvas de utilização de recursos, especialmente a carga média e a vazão da rede.

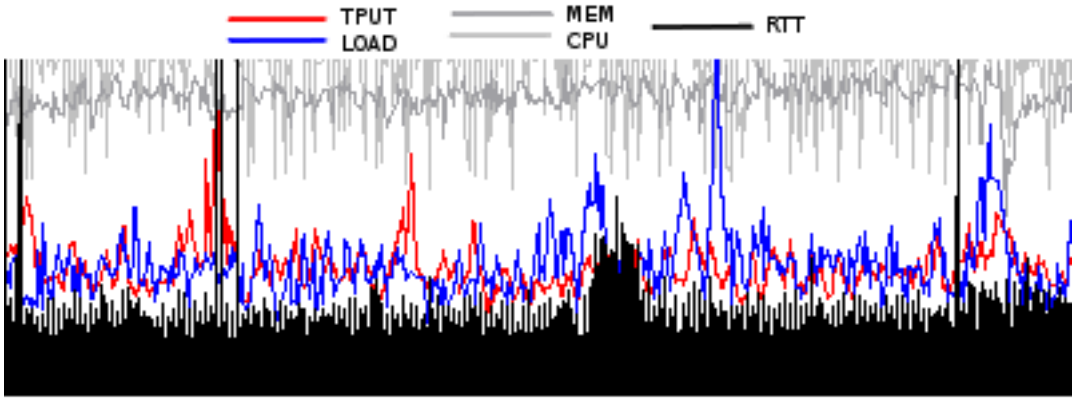


Figura 5.1. Influência da Carga no Atraso das Mensagens

Para utilizar os dados de carga, é necessária uma forma de relacioná-los quantitativamente com os atrasos percebidos. Na seção a seguir propomos uma maneira de quantificar um valor que represente a sobrecarga de utilização de recursos no nó

monitorado em relação aos atrasos observados pelo nó monitor.

5.1 Correlação Entre Carga e Atraso

Para utilizar tais dados de carga devemos primeiro caracterizar a correlação entre a utilização de recursos e os atrasos percebidos nas mensagens. Para tal utilizamos o *software* R-project [R Development Core Team, 2008] para calcular a correlação entre cada um dos dados de utilização de recursos no nó monitorado com o RTT percebido no nó monitor, chegando aos valores exibidos na tabela 5.1. Para o cálculo da correlação foi utilizado o método de Pearson [Croarkin & Tobias, 2002] para checar se as duas variáveis dependem linearmente uma da outra. Após seu cálculo, os valores da correlação foram arredondados para definir o valor do peso de cada métrica no cálculo do fator de sobrecarga da máquina.

Tabela 5.1. Correlação da Utilização de Recursos com o RTT

	LOAD-RTT	***CPU-RTT***	***MEM-RTT***	***TPUT-RTT***
Correlação	0,02040	0,00589	0,00303	0,00895
Pesos Escolhidos	20	6	3	9

Definimos como fator de sobrecarga a média ponderada das métricas de utilização de recursos, de maneira a fornecer um valor numérico que represente o "nível de sobrecarga" que a máquina se encontra. O cálculo é feito através pela equação 5.1, utilizando os dados de carga mais atuais, ou seja, coletado com tempo anterior à última mensagem trocada. No caso do *trace*, a carga foi coletada e armazenada localmente no nó monitorado e ao final da coleta agruparmos os dados de todos os nós num mesmo local. Após, bastou obter o dado de carga do nó monitorado com tempo anterior à última mensagem recebida pelo nó monitor, visto que os relógios estavam sincronizados e seu escorregamento (*clock-drift*) pode ser desprezado em relação ao intervalo de 10 segundos entre mensagens. Já numa implementação real bastaria obter os dados de carga enviados via *piggyback* no último *heartbeat* recebido, ou seja, o nó monitorado adicionaria ao pacote que leva o *heartbeat* os seus dados de carga para aquele instante.

$$S = \frac{(20 * LOAD + 6 * CPU + 3 * MEM + 9 * TPUT)}{(20 + 6 + 3 + 9)} \tag{5.1}$$

Durante os cálculos do fator de sobrecarga, os valores do uso de CPU e memória foram normalizados por 100, a carga foi "normalizada" por 7 e a vazão da rede foi "normalizada" por 4000, valores estes aproximadamente correspondentes ao 75-quantil

da utilização global de recursos do PlanetLab (vide tabela 4.6). Estes valores de normalização foram obtidos após diversos testes com valores máximos e intermediários, visando obter um fator de sobrecarga que melhor prediga os atrasos observados.

O fator de sobrecarga foi adicionado ao algoritmo do FD-Sensi da seguinte maneira: tal qual o histórico dos intervalos de recebimento dos *heartbeats*, é mantido um histórico das sobrecargas com uma janela de amostragem de mesmo tamanho (últimos 1000 valores). Caso o intervalo desde o último *heartbeat* recebido seja maior que o *timeout* (definido pela equação 3.3), antes de suspeitar-se de uma falha no processo verifica-se o valor da sobrecarga naquele momento e, se ela for maior que um certo limiar de sobrecarga, é dado ao nó um "crédito por sobrecarga" de 3 vezes o intervalo de *heartbeat*. Se após este período de crédito não recebermos nenhuma mensagem do nó em questão, aí sim suspeita-se que uma falha tenha ocorrido nele. O limiar de sobrecarga citado acima é por sua vez calculado como a média das sobrecargas anteriores menos 3 desvios padrões, conforme mostra a equação 5.2:

$$S_{max} = \bar{S} - 3\sigma \tag{5.2}$$

Observe que os valores de 3 desvios padrões e 3 intervalos de crédito foram definidos empiricamente por fornecerem um bom custo benefício entre a redução de falso-positivos e a manutenção de um tempo médio de detecção baixo. Foram realizados diversos testes com valores fixos e dinâmicos para os desvios padrões. Intervalos menores de crédito foram utilizados, porém ocasionaram uma redução pouco significativa nos falso-positivos, enquanto que valores maiores apresentaram um impacto muito grande no tempo médio de detecção. Sendo assim, o valor de 3 desvios padrões e intervalos de crédito consistem num bom custo benefício para o cenário distribuído do PlanetLab. Estes valores deverão ser reconfigurados para o uso em outros cenários ou de acordo com as necessidades de outras aplicações.

Esta versão do FD-Sensi otimizada para o PlanetLab foi denominada FD-Sensi+Rsrc, por utilizar informações da utilização de recursos dos nós para melhorar sua predição de falhas, diferenciando-as de atrasos de rede, perdas e atrasos por sobrecargas. Porém, tal técnica de aperfeiçoamento não é restrita ao FD-Sensi, podendo ser aplicada a qualquer algoritmo de detecção de falhas que deseje-se implementar voltado ao PlanetLab. Ela também pode ser utilizada em outros ambientes, desde que seja possível neles obter informações de carga das máquinas e realize-se uma nova análise da correlação entre carga e atraso, específica para o novo cenário.

A correlação entre a utilização de recursos e os atrasos de rede percebidos mostrou-se muito útil na predição de atrasos, possibilitando-nos melhorar consideravel-

mente a velocidade da detecção de falhas como poderá ser visto na seção de resultados (5.3).

5.2 Avaliação do Algoritmo com uma Carga Real

Com o objetivo de avaliar a qualidade da detecção apresentada pelo FD-Sensi , utilizaremos os mesmos parâmetros e métricas utilizados em Satzger et al. [2007], visando comparar o desempenho da detecção de falhas no cenário do Planetlab. Porém, ao invés de utilizarmos uma carga sintética com distribuição normal para amostragem dos atrasos nas mensagens, utilizaremos o *trace* coletado no PlanetLab para melhor refletir o comportamento de um sistema largamente distribuído, possibilitando assim uma comparação de desempenho num cenário real.

Os dados do *trace* foram utilizados para comparar o desempenho do FD-Sensi com um dos melhores detectores de falhas da atualidade, o *Adaptive Accrual*, projetado para ser flexível e adaptativo [Satzger et al., 2007], conforme mencionado na seção 3.2. Nossos resultados mostram que o FD-Sensi obteve um melhor desempenho que o *Adaptive Accrual*, com significativa redução da emissão de falso-positivos e manutenção de um baixo tempo de detecção.

5.2.1 Métricas

Assim como visto na seção 3.2.1, as métricas que selecionamos para avaliar a qualidade da detecção foram:

- *Tempo Médio de Detecção*: mede a **velocidade** em termos do tempo decorrido entre a falha de um processo e a suspeita dessa falha.
- *Quantidade de Falso-positivos*: mede a **precisão** em termos da quantidade de enganos cometidos pelo detector de falhas.

5.2.2 Parâmetros e Fatores

Considerando os resultados da seção 3.2, observamos que o *Adaptive Accrual* diminui a quantidade de falso-positivos quando seu parâmetro de configuração α é diminuído. Isto pode ser explicado devido ao modelo utilizado para predizer se um *heartbeat* está atrasado ou não. No *Adaptive Accrual*, a um *heartbeat* é atribuído um valor normalizado de suspeita, calculado da seguinte maneira: $\frac{|N_{elementos} \leq t_\Delta \cdot \alpha|}{\eta}$, onde t_Δ representa o intervalo de recebimento (*timestamp* do *heartbeat* atual menos o último *heartbeat*

recebido) e η é o tamanho da janela de amostragem. Ou seja, a quantidade de elementos com tempo inferior ao t_Δ inflacionado por α é dividida pelo tamanho da janela de amostragem. Isso nos dá o nível de suspeita, que segundo Satzger et al. [2007], ao utilizar as frequências cumulativas baseia-se na estimativa da função distribuição acumulada (CDF) dos intervalos de recebimento. Pela fórmula acima, percebemos que ao diminuir α fazemos com que se diminua o valor de suspeita e consequentemente diminuímos os falso-positivos (mas aumentando o tempo de detecção).

Já o FD-Sensi mostra comportamento inverso no parâmetro de configuração κ: ao diminuí-lo, o algoritmo apresenta um aumento no número de suspeitas. Isso é também explicado pelo seu modelo de cálculo de suspeitas, baseado em $\kappa\cdot$ desvios padrões, onde obviamente se κ for pequeno demais, o algoritmo causará mais suspeitas, mas obtendo em contrapartida um tempo de detecção menor. Em suma, os parâmetros que influenciam no desempenho dos algoritmos de detecção de falhas são as respectivas calibragens de cada algoritmo:

FD-Sensi: seu *timeout* é calculado como sendo $\bar{T} + \kappa.\sigma$, onde $\bar{T}$ é a média observada nos intervalos de recebimento dos *heartbeats* e σ é o seu desvio padrão. Sendo assim, os parâmetros do nosso algoritmo são:

- κ: define um *fator de confiança* pelo qual o desvio padrão será inflacionado. Este parâmetro variou nos 15 seguintes níveis: 10,0 ; 5,0 ; 3,0 ; 1,5 ; 1,0 ; 0,50 ; 0,25 ; 0,075 ; 0,025 ; 0,0 ; -0,05 ; -0,25 ; -0,5 ; -1,0 ; -3,0
- η: tamanho da janela de amostragem estatística (histórico de *heartbeats*)

Adaptive Accrual: seu *timeout* é calculado utilizando $\alpha.\triangle_i$ (o atraso mais uma margem de segurança) e assim, os parâmetros deste algoritmo são:

- α: define um *fator de escala* para o intervalo de recebimento observado no *heartbeat*. Este parâmetro variou nos 15 seguintes níveis: 0,25 ; 0,50 ; 0,65 ; 0,80 ; 0,90 ; 0,925 ; 0,950 ; 0,975 ; 0,995 ; 1,00 ; 1,033 ; 1,066 ; 1,10 ; 1,30 ; 1,50
- η: tamanho da janela de amostragem estatística (histórico de *heartbeats*)

Tal como explicado na seção 3.2.2, os níveis de cada parâmetro foram escolhidos de maneira a melhor representar as diversas faixas de comportamento dos algoritmos de detecção, provendo tanto uma cobertura dos extremos quanto detalhamentos nas regiões mais críticas. Diferente dos 23 níveis de antes, conseguimos diminuí-los para 15 níveis ao eliminar valores pouco representativos na faixa de baixas velocidades de detecção. Com isso pudemos melhor detalhar áreas de maior interesse como a inflexão da curva de custo-benefício e também altas precisões de detecção de falhas.

Assim como em Satzger et al. [2007], para a avaliação dos algoritmos de detecção de falhas foi fixado o parâmetro η, com uma janela de amostragem com os 1000 últimos *heartbeats* recebidos. Um pequeno teste realizado com uma janela de amostragem menor (últimos 100 *heartbeats*) mostrou que a diferença entre o desempenho dos dois FDs foi amplificada, porém mantendo a tendência das curvas. Com esta janela de 100, o FD-Sensi mostrou um resultado tão bom quanto ou deteriorou pouco em relação a seu resultado com uma janela com os 1000 últimos *heartbeats* recebidos. Já o *Adaptive Accrual* deteriorou muito seu resultado caso utilize-se uma janela de amostragem menor que os 1000 últimos *heartbeats* recebidos. Com uma janela de 1000 os dois algoritmos já mostraram desempenho claramente diferenciável, portanto não foi considerado aumentar tal parâmetro. A utilização de janelas de amostragem muito grandes demanda uma utilização de recursos que pode não ser aceitável para um serviço de infra-estrutura como a detecção de falhas. Além disso pode causar uma menor reatividade pela existência de *heartbeats* muito antigos que já não representam o estado do nó monitorado, confirme discutido na seção 3.1.

5.2.3 Modelo de Sistema Adotado

É assumido um modelo de falhas *fail-stop*, onde os nós falham e não executam nenhuma operação após isso, restando ao nó monitor detectar a falha pela omissão de comunicações posteriores. Nós são considerados como faltosos somente quando realmente tiverem falhado, de maneira que perdas de mensagens e atrasos progressivos (devido a sobrecargas) não caracterizam uma falha. As mensagens enviadas através da rede podem não ser entregues com sucesso ao destinatário, se perdendo por motivos variados tais como congestionamento da rede, estouro de *buffer* no remetente, destinatário ou mesmo na fila dos roteadores intermediários, etc.

No FD-Sensi, não assumimos sincronização dos relógios ao longo dos membros do grupo, somente necessitando que a taxa de escorregamento do relógio individual de cada membro permaneça constante. A sincronização foi necessária apenas para relacionarmos os atrasos e a utilização de recursos dos nós do *trace* coletado. O algoritmo do FD-Sensi por si só não assume nem necessita de tal sincronização.

5.2.4 Carga de Trabalho e Mecânica dos Experimentos

Dentre os dados necessários para avaliar o desempenho dos FDs temos os atrasos e eventuais perdas de mensagem, ambos registrados nos *traces*, necessários para determinar os intervalos de recebimento dos *heartbeats*. Assim sendo, dos 81 *traces* coletados cada

um dos 6480 enlaces de comunicação entre dois nós consiste numa carga de trabalho.

Uma importante medida da confiabilidade é o *time-to-failure* (TTF), ou seja, o intervalo de tempo contínuo enquanto um nó está vivo. Em análise realizada da disponibilidade dos nós do Planetlab, Zhang & Honeyman [2008] caracterizaram o TTF médio (MTTF) do Planetlab em 122,8 horas, modelando a ocorrência de falhas como uma distribuição Weibull. Os parâmetros de escala e forma da distribuição Weibull *best-fit* foram 80.556 e 0,3549, respectivamente. Assim, de maneira a simular a ocorrência de falhas nos nós, para cada carga foi realizada uma injeção de 1% de falhas utilizando uma distribuição Weibull com os parâmetros acima.

Realizamos então um projeto experimental do tipo "projeto simples" [Jain, 1991], onde para cada uma das 6480 cargas de trabalho variamos os parâmetros de ajuste de cada algoritmo em cada um dos 15 níveis citados e para cada uma destas configurações rodamos o simulador com cada um dos FDs: FD-Sensi, FD-Sensi+Rsrc e *Adaptive Accrual*.

Através do *trace* obtemos TWPs, de maneira a ter acesso aos atrasos e perdas necessários para se modelar um cenário real de um sistema distribuído em larga escala. Conforme visto na seção 4.2.1, cada TWP consiste na troca de três mensagens, a saber: PING, PING-ACK e ACK. Para simular os tempos dos *heartbeats* através das mensagens do TWP, bastaria utilizar os tempos de chegada das solicitações iniciais do protocolo, denominadas PING. O temporizador do software TWP é razoavelmente preciso (10 ms), de maneira que cada solicitação inicial PING será recebida a aproximadamente 10 segundos (mais o atraso de sua transmissão). Porém, conforme dito na seção 4.3.2 por uma decisão de projeto que visou viabilizar o armazenamento do *trace*, cada nó envia TWPs a N/2 nós e responde da outra metade. Assim sendo, um nó B receberá solicitações iniciais PING apenas de metade dos outros nós, de maneira que para um nó A (da outra metade) devemos utilizar os tempos das mensagens PING-ACK recebidas, conforme ilustra a figura 5.2.

Observe que o recebimento de um PING-ACK ficaria amarrado à entrega com sucesso do PING que o solicitou. Se o PING enviado por A se perder e consequentemente não for gerado um PING-ACK por parte de B, nem um nem outro receberiam *heartbeats* naqueles momentos, o que poderá ser interpretado como uma simples perda de mensagens. Dito isso, consideramos como uma simplificação razoável a modelagem dos *heartbeats* desta maneira, visto que a alternativa seria a utilização dos RTTs de A para B e vice-versa para simular os heartbeats. Porém, caso assim o fizéssemos, decairíamos numa estratégia de monitoramento *query-response*, confirme explicado na seção 2.3.1.

Ao ler uma carga e encontrar uma injeção de falha, o simulador salva um *check-*

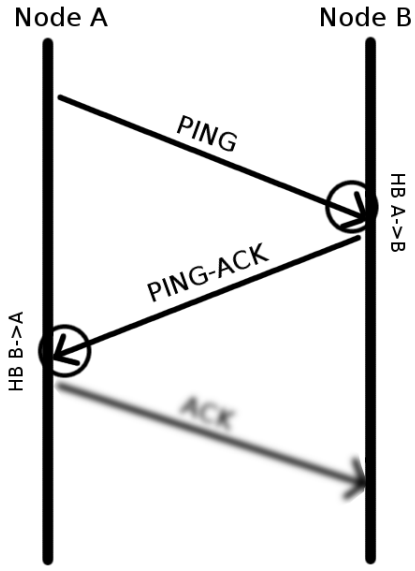


Figura 5.2. Simulação de HB via TWP

point e pára de carregar *heartbeats*, passando a esperar até que o detector de falhas suspeite da ocorrência da falha (*fail-stop*). Após devidamente suspeitada a falha, o simulador restaura o *checkpoint* e continua a ler *heartbeats* da carga a cada iteração, ou seja, volta ao estado anterior à injeção de falha como se ela não tivesse ocorrido. Isso é feito para aproveitar-se o restante da carga, dando continuidade ao processamento da detecção. Observe que uma falha será suspeitada a partir do momento em que um *heartbeat* esperado não chegar. A figura 5.3 ilustra a falha do nó A no instante t5, seguida pela constatação do não recebimento de um *heartbeat* no instante t6. Após t6, acrescido de uma tolerância d, a falha será suspeitada. Um falso-positivo ocorre quando o detector de falhas suspeita de uma falha não injetada, sendo igualmente fácil contabilizar estes eventos.

Os resultados tanto globalizados quanto agrupados de acordo com os clusters identificados são apresentados a seguir. O *trace* do qual foi obtido as cargas de trabalho, dados de utilização de recursos, o código-fonte do coletor e do detector de falhas e respectivos experimentos realizados, bem como uma implementação do FD-Sensi como protocolo de detecção utilizando o *framework* JGroups podem ser encontrados em `http://homepages.dcc.ufmg.br/~evaladao/storage/fdsensi/`.

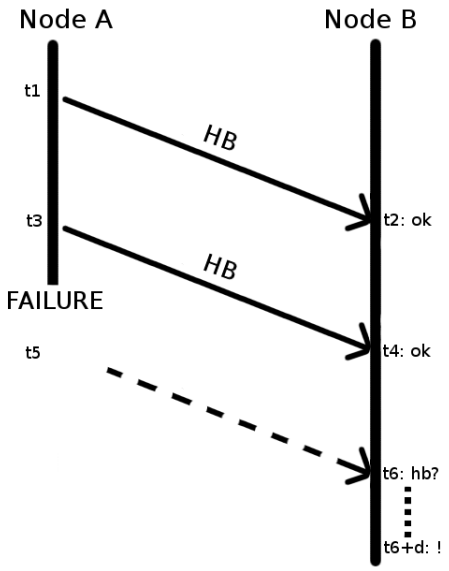
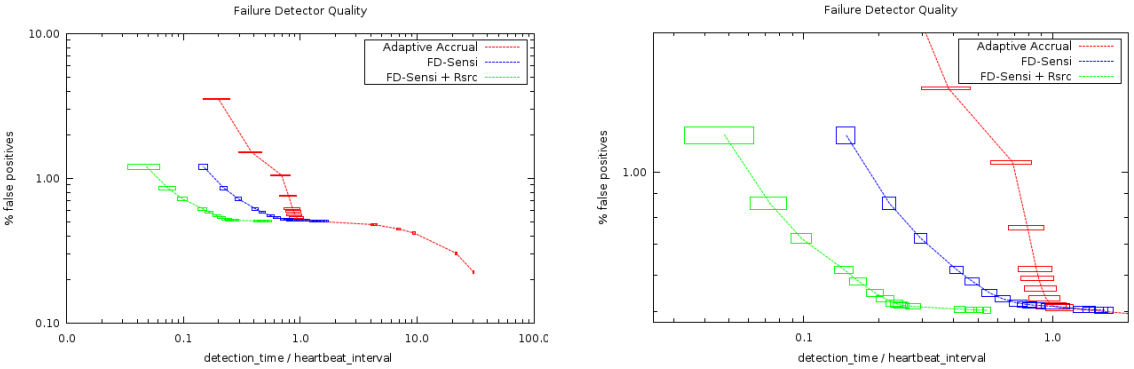


Figura 5.3. Injeção de Falha e sua Detecção

5.3 Resultados da Análise Experimental no PlanetLab

Os resultados da qualidade de detecção de falhas para o *trace* coletado no Planetlab são exibidos na figura 5.4a. Observe que ambos os eixos estão em escala logarítmica e as métricas foram normalizadas: o eixo X representa a *razão* do tempo médio de detecção das falhas normalizado em relação ao intervalo de *heartbeat*. O *detection time* consiste no tempo esperado *após* o não-recebimento de um *heartbeat* para sinalizar a ocorrência de uma falha. Valores iguais ou acima de 1,0 na prática significam que se esperou pelo próximo *heartbeat* e entre 0,0 e 1,0 creditou-se uma porcentagem de atraso ao heartbeat não-recebido antes de sinalizar a falha. Já o eixo Y mostra a *taxa* de falso-positivos normalizada pelo total de *heartbeats* trocados. As caixas representam os intervalos de confiança de 99% para ambas as métricas. A figura 5.4b concentra-se nos valores que interessam para um uso prático dos detectores: aqueles que se encontram à esquerda do ponto de inflexão da curva, ou seja, no canto inferior esquerdo. Esses valores representam um tempo de detecção pequeno com poucos falso-positivos.

Caso a prioridade da aplicação seja obter baixíssimas taxas de falso-positivos com um tempo médio de detecção razoável, considere a faixa de valores exibida em detalhe na figura 5.5. Nela é possível observar tempos médios de detecção de 40% a 180% do intervalo de *heartbeat*, no caso, de 4 a 18 segundos. Nessa faixa, após o não recebimento de um *heartbeat* os detectores identificam a ocorrência de falhas



(a) Visão Geral (b) Faixa de Uso Prático

Figura 5.4. Qualidade de Detecção no PlanetLab

emitindo somente de 0,50% a 0,65% de falso-positivos, ou seja, com uma precisão de mais de 99%. Nessa faixa de valores o desempenho dos detectores começa a diferenciar-se significativamente para tempos médios de detecção abaixo de 0,8 vezes o intervalo de comunicação esperado, ou seja, abaixo de 8 segundos após o não recebimento do *heartbeat*. Acima disso o desempenho do FD-Sensi e do *Adaptive Accrual* não pode ser diferenciado com 99% de confiança (com 95% e 90% também não). Isso pode ser explicado pois acima da razão de 1,0 vez o intervalo de *heartbeat*, na prática o que fizeram foi esperar pela chegada da próxima mensagem, não exibindo efetivamente o seu potencial de predição de atrasos. Abaixo da razão de 0,8 é possível calibrar o FD-Sensi para obter a mesma precisão do *Adaptive Accrual* porém com tempos médios de detecção significativamente menores.

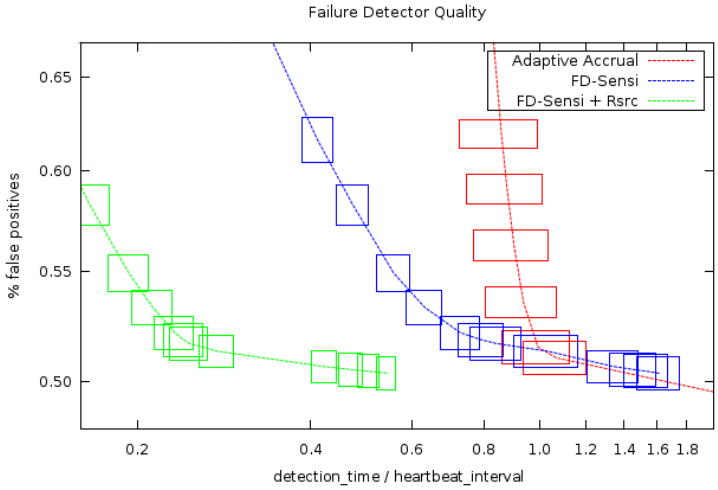


Figura 5.5. Qualidade de Detecção - Baixa Taxa de Falso-positivos

Note ainda na curva verde das figuras 5.4b e 5.5 o significativo ganho de desempenho na velocidade de detecção caso adicione-se ao FD-Sensi dados acerca da carga de recursos dos nós (FD-Sensi+Rsrc). Com isso aumenta-se seu potencial de predição para atrasos no recebimento de mensagens, de maneira que sua melhor velocidade de detecção de falhas fica abaixo de 600 ms (máximo atraso típico de rede) e sua pior velocidade não passa de 6 segundos após o não-recebimento do *heartbeat*.

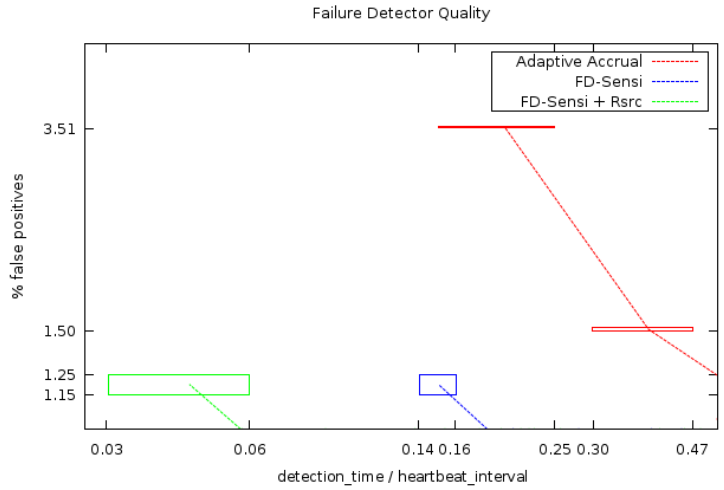


Figura 5.6. Qualidade de Detecção - Baixo Tempo de Detecção

Considerando-se outra possibilidade, caso a prioridade da aplicação seja obter tempos médios de detecção mais baixos ao custo de uma taxa de falso-positivos razoavelmente maior, considere a faixa de valores exibida em detalhe na figura 5.6. Nela observamos tempos médios mínimos de detecção de 3% a 30% do intervalo de *heartbeat* (primeira caixa de confiança de cada curva), ou seja, de 300 ms a 3 segundos. Nessa faixa, à medida que a razão do intervalo de comunicação esperado se aproxima de zero, as suposições acerca da falha de processos se tornariam mais propensas ao erro devido a simples atrasos e/ou perdas na comunicação de mensagens. Nesse universo onde se tem cada vez menos certezas, o FD-Sensi consegue um desempenho claramente superior ao *Adaptive Accrual*, apresentando um tempo médio de detecção de 1,4 a 1,6 segundos ao custo de uma taxa de falso-positivos de 1,15 e 1,25%. O *Adaptive Accrual* conseguiu um tempo médio de detecção variando entre 1,5 a 2,5 segundos, porém ao custo de 3,51% de falso-positivos, ou seja, um tempo de detecção menos homogêneo e com uma precisão menor. Caso consideremos uma configuração na qual o *Adaptive Accrual* obtenha uma taxa de falso-positivos de 1,5%, seu tempo de detecção médio cresce para 3 a 4,5 segundos. Note ainda o também significativo ganho de desempenho do FD-Sensi+Rsrc: mantendo a precisão alcançada pelo FD-Sensi original, o FD-Sensi+Rsrc

reduziu o tempo médio de detecção para 300 a 600 ms. Isso é graças à habilidade de perceber que a grande maioria dos nós estavam apenas sobrecarregados e não faltosos, conforme foi observado na tabela 4.6. O custo para se obter esta melhoria é o envio de 120 bytes extras em cada *heartbeat* (contendo a utilização de recursos) e o armazenamento de um histórico desses dados. Por exemplo, no caso de armazenar-se os últimos 1000 *heartbeats*, seria necessário o armazenamento de cerca de 117 KB extras por nó monitorado.

5.3.1 Resultados por Clusters de Comportamento (RTT & Perda)

Para facilitar a análise dos resultados repetimos aqui a tabela com as estatísticas da clusterização dos enlaces por comportamento de atrasos, variação e perdas de mensagens:

Tabela 5.2. Estatísticas da Clusterização

Cluster	*RTT Mean*	*Loss Rate*	*RTT StDev*	*RTT CV*	*% Enlaces*
c1	49 ms	0,22%	46 ms	1,12	21%
c2	131 ms	0,40%	673 ms	6,37	21%
c3	167 ms	0,30%	55 ms	0,33	24%
c4	269 ms	0,12%	266 ms	0,96	21%
c5	358 ms	1,40%	158 ms	0,44	13%
Global	181 ms	0,41%	240 ms	1,89	100%

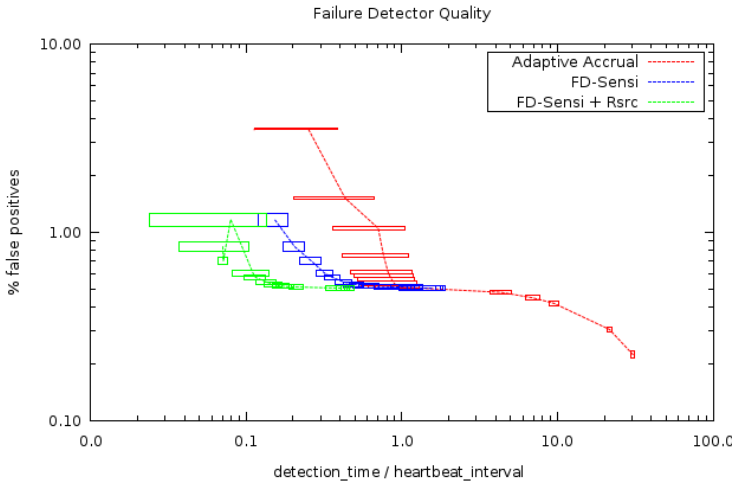
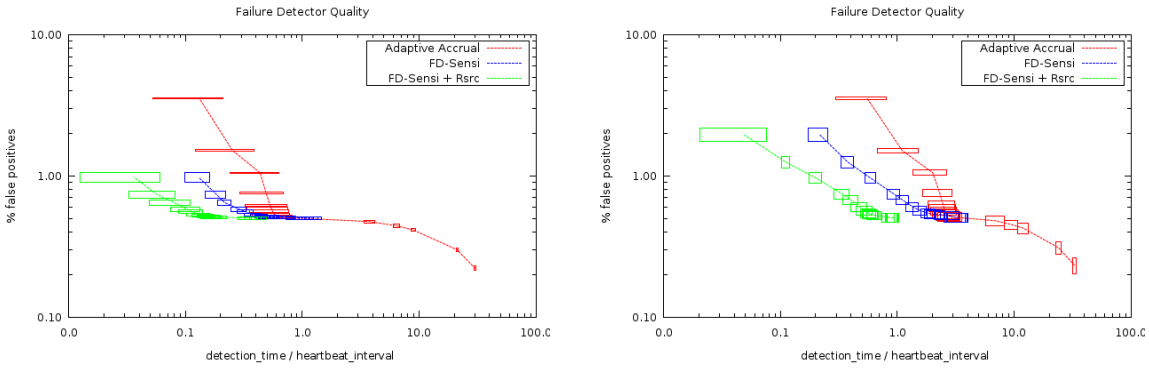


Figura 5.7. Qualidade de Detecção num Cluster Típico

O cluster 4 foi o que mais se assemelhou com o resultado global do PlanetLab no que diz respeito à qualidade de detecção de falhas. A figura 5.7 exibe os resultados observados para o cluster 4 e, tal qual na figura 5.4a, ambos os eixos estão em escala logarítmica. O principal diferencial do cluster 4 em relação ao resultado global observado no PlanetLab são os intervalos de confiança mais largos, que fazem com que os detectores FD-Sensi e *Adaptive Accrual* só possam ser diferenciáveis com tempos de detecção abaixo de 0,33 vezes o intervalo de *heartbeats* (ou o equivalente a 3,3 segundos após o não-recebimento do *heartbeat*).

Conforme podemos verificar na tabela 5.2, os clusters 1 e 5 representam extremos tanto no RTT médio quanto na taxa de perda de mensagens. Através das figuras 5.8a e 5.8b é possível comparar o desempenho dos detectores nestes dois cenários tão diversos. O que mais chama a atenção é o visível deslocamento das curvas no eixo das abscissas, conforme podemos verificar pelos pontos de inflexão das curvas: 0,6 no cluster 1 e 3,0 no cluster 5. Isso indica que no cluster 5, onde os atrasos das mensagens são em média 7 vezes maiores, os detectores apresentam um tempo médio de detecção pior que no cluster 1. Além disso, os intervalos de confiança para a taxa de falso-positivos são significativamente maiores no cluster 5 em relação ao cluster 1, devido a sua taxa de perda de mensagens ser 6 vezes maior. No cluster 1 os detectores FD-Sensi e *Adaptive Accrual* só são diferenciáveis com tempos de detecção abaixo de 0,33 vezes o intervalo de *heartbeats*, mas no cluster 5 este valor sobe para 1,76 vezes o intervalo de *heartbeats* (ou o equivalente a 17,6 segundos).

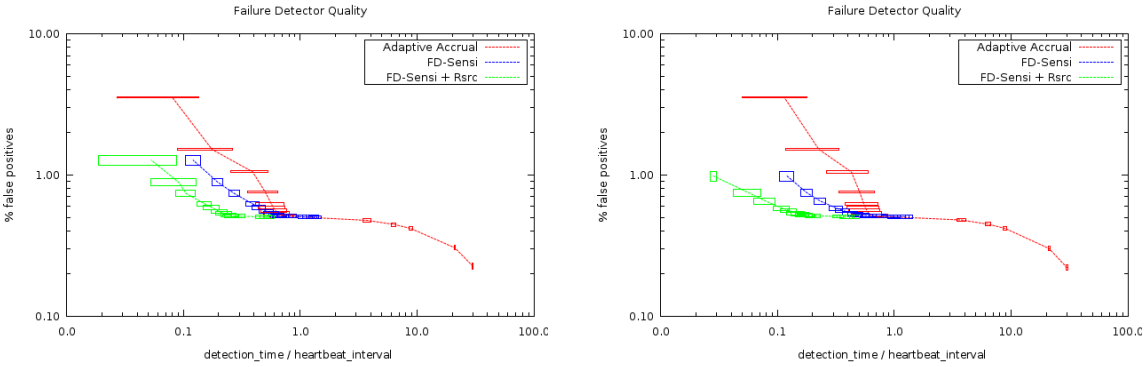


(a) Cluster 1 (b) Cluster 5

Figura 5.8. Qualidade de Detecção - Comparando o Impacto de Atrasos/Perdas

Os clusters 2 e 3 representam extremos de alta e baixa variabilidade nos atrasos das mensagens. Analisando a tabela 5.2 podemos perceber que o cluster 2 apresentou um coeficiente de variação quase 20 vezes maior que o cluster 3. Isso se reflete nos resultados observados nas figuras 5.9a e 5.9b. Nelas, podemos observar uma maior

aproximação das curvas de qualidade de detecção no cluster 2, de maneira que ambos os detectores aproximam mais seu desempenho. No cluster 2 só é possível diferenciar o desempenho dos detectores com tempos de detecção abaixo de 0,33 vezes o intervalo de *heartbeat* enquanto que no cluster 3 este valor sobe para 0,36 vezes o intervalo de *heartbeats* (ou o equivalente a 3,6 segundos).



(a) Cluster 2 (b) Cluster 3

Figura 5.9. Qualidade de Detecção - Comparando o Impacto da Variabilidade do Atraso

Através dos resultados foi possível afirmar com 99% de confiança que, no cenário do PlanetLab, o FD-Sensi apresentou desempenho superior ao *Adaptive Accrual* para baixos tempos de detecção (0,8 vezes o intervalo de comunicação esperado, ou seja, abaixo de 8 segundos) e alta precisão (menos de 1% de falso-positivos). Para tempos de detecção mais elevados (acima de 8 segundos), o desempenho do FD-Sensi e do *Adaptive Accrual* não pode ser diferenciado nem com 90% de confiança. O melhor tempo médio de detecção do FD-Sensi foi 1,5 segundos com a emissão de 1,2% de falso-positivos, ao passo que o *Adaptive Accrual* só conseguiu velocidade próxima com a emissão de três vezes mais falso-positivos e com um tempo de detecção menos homogêneo. A adição de informações de sobrecarga dos nós possibilitou ao FD-Sensi+Rsrc significativas melhorias de desempenho, com a redução do melhor tempo médio de detecção para 300 a 600 ms. Pela clusterização dos dados percebemos que aumentos na taxa de perda de mensagens e na variabilidade dos atrasos intensificam as diferenças de desempenho entre os dois algoritmos, com o FD-Sensi mantendo um bom desempenho tanto na velocidade quanto na precisão da detecção de falhas. O *Adaptive Accrual* seria mais indicado para cenários com distribuição normal dos atrasos de rede. Porém, como identificamos que no PlanetLab os atrasos seguem uma distribuição com cauda mais prolongada e mais dispersa (Gamma), o FD-Sensi seria uma boa solução para cenários distribuídos que experimentem alta variabilidade nos atrasos e sobrecarga dos nós.

Capítulo 6

Conclusões e Trabalhos Futuros

Foi proposto e analisado um novo algoritmo de detecção de falhas adaptativo chamado FD-Sensi, aplicável a cenários de sistemas distribuídos heterogêneos, especialmente aqueles que apresentem sobrecarga e alta variabilidade na comunicação. O algoritmo proposto requer poucos recursos computacionais e é matematicamente bem fundamentado, baseando-se numa predição estatística acerca dos atrasos de comunicação. Uma comparação de desempenho com o *Adaptive Accrual* [Satzger et al., 2007] foi realizada, por este último ter apresentado ótimos resultados em comparação com outros detectores de falhas [Chen et al., 2002; Bertier et al., 2002; Hayashibara et al., 2004]. Tal avaliação foi efetuada com dados reais coletados no Planetlab, na qual comparações de desempenho foram realizadas tanto a nível global como em clusters de nós que apresentaram características semelhantes de perdas de mensagens, atrasos e variação nos atrasos das mensagens.

Através de injeções de falhas com distribuição Weibull, os resultados mostraram que o FD-Sensi superou o *Adaptive Accrual*, especialmente em termos de uma significativa redução do tempo médio de detecção com a manutenção de uma taxa de falso-positivos menor ou compatível. Através dos experimentos realizados, foi possível afirmar com 99% de confiança que os algoritmos apresentaram desempenho diferenciável para tempos médios de detecção abaixo de 0,8 vezes o intervalo de comunicação esperado e precisão de 99%. Acima disso não é possível diferenciá-los nem com 90% de confiança. Em sistemas distribuídos de tempo real é importante detectar as falhas tão rápido quanto possível, de maneira a dar início ao processo de recuperação. Já em outros sistemas como comércio eletrônico é necessária uma forte precisão para evitar operações duplicadas [Sotoma & Madeira, 2001; Yang et al., 2006]. Uma maior precisão ao custo de uma menor velocidade na detecção também é também necessária a esquemas de distribuição de processamento do tipo mestre-escravo, onde o custo da

duplicação de esforços devidos à falso-positivos é alto. Portanto, a aplicação do detector de falhas é que irá determinar quais características da qualidade do serviço de detecção que deve-se priorizar.

A partir de dados acerca da utilização de recursos nos nós do PlanetLab, propomos uma técnica de aperfeiçoamento do FD-Sensi que, baseando-se na correlação entre a carga do nó monitorado e os atrasos percebidos, identifica sobrecargas e evita falso-positivos que delas proviriam. Com isso foi possível melhorar significativamente a precisão e velocidade do FD-Sensi no cenário do PlanetLab, atingindo velocidades abaixo de 6% do intervalo de comunicação e precisão acima de 98,7%. Tal técnica de aperfeiçoamento não é restrita ao FD-Sensi, podendo ser aplicada a qualquer algoritmo de detecção de falhas que deseje-se implementar voltado ao PlanetLab. Tal técnica poderia ser utilizada para melhorar o desempenho do *Adaptive Accrual*, porém, observe que o FD-Sensi original já foi suficiente para superar o desempenho do *Adaptive Accrual* no cenário do PlanetLab, sem a necessidade de tal aperfeiçoamento.

As modelagens de carga utilizadas nas avaliações de detectores de falhas comumente utilizam a distribuição normal para modelar os atrasos de rede. Porém, conforme observado na análise do *trace* coletado, a utilização da distribuição Gamma é mais apropriada para a modelagem dos atrasos de rede em cenários globalmente distribuídos. Isso é devido a ela apresentar uma cauda mais longa, ajustando-se melhor aos valores extremos de atrasos percebidos ao longo da Internet. Ao contrário do *Adaptive Accrual*, o FD-Sensi não pressupõe que os intervalos de chegada das mensagens sigam uma distribuição normal, o que lhe conferiu uma maior precisão e velocidade na detecção de falhas no cenário do Planetlab. Observe que através do *trace* coletado e sua respectiva análise, este trabalho fornece tanto uma carga real quanto as distribuições estatísticas mais apropriadas para a modelagem de uma carga sintética deste cenário. Estes dados facilitarão o projeto e análise de detectores de falhas voltados para cenários reais de utilização, especialmente ambientes globalmente distribuídos como o PlanetLab.

Um protótipo utilizando o algoritmo FD-Sensi foi implementado como um protocolo de detecção de falhas para o *framework* JGroups [Montresor, 2006], no qual os membros de um grupo de detecção realizam um protocolo de consenso através de uma votação. Porém, ele ainda não provê a escalabilidade necessária ao Planetlab, que atualmente possui cerca de 1006 nós globalmente distribuídos. Apesar de possuir difusão epidêmica de informações [van Renesse et al., 1998; Das et al., 2002] constitui-se num grupo único de detecção que suportaria eficientemente apenas algumas dezenas de nós.

Como trabalhos futuros, realizaremos uma caracterização mais extensa e cuidadosa do *trace* coletado, visto que ele poderá prover informações essenciais para o

projeto de detectores de falhas para sistemas distribuídos em larga escala (WANs). Uma versão com grupos de detecção hierárquicos será desenvolvida para prover a escalabilidade necessária [Bertier et al., 2003]. Ao dividir a tarefa de detecção de centenas de nós entre diversos detectores espalhados na rede, haverá um aumento na escalabilidade e possivelmente na precisão e velocidade de detecção das falhas. Também visamos desenvolver um *overlay* par-a-par (P2P) que gere uma topologia que melhor explore a localidade de rede. Os nós serão escolhidos de acordo com o comportamento de seus *heartbeats*, buscando associar-se com nós mais estáveis e de menores latências.

Referências Bibliográficas

Aguilera, M. K.; Chen, W. & Toueg, S. (1997). Heartbeat: A timeout-free failure detector for quiescent reliable communication. In *Distributed Algorithms*, volume 1320 of *Lecture Notes in Computer Science*, pp. 126–140, Saarbrucken, Alemanha. Springer.

Aguilera, M. K.; Chen, W. & Toueg, S. (2000). On quiescent reliable communication. *SIAM Journal on Computing (SICOMP)*, 29(6):2040–2073.

Avizienis, A.; Laprie, J.-C.; Randell, B. & Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33.

Bertier, M.; Marin, O. & Sens, P. (2002). Implementation and performance evaluation of an adaptable failure detector. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks (DSN '02)*, pp. 354–363, Washington, DC, EUA. IEEE Computer Society.

Bertier, M.; Marin, O. & Sens, P. (2003). Performance analysis of a hierarchical failure detector. In *Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN '03)*, pp. 635–644, San Francisco, CA, EUA. IEEE Computer Society.

Brasileiro, F. & Sampaio, L. (2001). A Pratical Approach to Validate the Design of Fault-Tolerant Distributed Protocols for Asynchronous Systems. In *Proceedings of the IX Simpósio de Computadores Tolerantes a Falhas*, volume 9, pp. 164–176.

Chandra, T. D. & Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267.

Chen, W.; Toueg, S. & Aguilera, M. (2002). On the quality of service of failure detectors. *IEEE Transactions on Computers*, 51(5):561–580.

Chockler, G. V.; Keidar, I. & Vitenberg, R. (2001). Group communication specifications: a comprehensive study. *ACM Computing Surveys*, 33(4):427–469.

Chu, F. C. (1998). Reducing omega to diamond w. *Information Processing Letters*, 67(6):289–293.

Chun, B.; Culler, D.; Roscoe, T.; Bavier, A.; Peterson, L.; Wawrzoniak, M. & Bowman, M. (2003). Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Computer Communication Review*, 33(3):3–12.

Cristian, F. & Fetzer, C. (1999). The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):642–657.

Croarkin, C. & Tobias, P. (2002). NIST/SEMATECH e-Handbook of Statistical Methods. Disponível em `http://www.itl.nist.gov/div898/handbook/`. Visitado em 2009-08-05.

Cui, J.-H.; Faloutsos, M.; Maggiorini, D.; Gerla, M. & Boussetta, K. (2003). Measuring and modelling the group mmbership in the internet. In *Proceedings of the* 3^{rd} *ACM SIGCOMM Conference on Internet Measurement (IMC '03)*, pp. 65–77, New York, NY, EUA. ACM.

Das, A.; Gupta, I. & Motivala, A. (2002). SWIM: Scalable weakly-consistent infection-style process group membership protocol. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks (DSN '02)*, pp. 303–312, Washington, DC, EUA. IEEE Computer Society.

Dutta, P.; Guerraoui, R. & Pochon, B. (2004). Fast non-blocking atomic commit: an inherent trade-off. *Information Processing Letters*, 91(4):195–200.

Défago, X.; Urban, P.; Hayashibara, N. & Katayama, T. (2005). Definition and specification of accrual failure detectors. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN '05)*, pp. 206–215, Los Alamitos, CA, EUA. IEEE Computer Society.

Felber, P.; Défago, X.; Guerraoui, R. & Oser, P. (1999). Failure detectors as first class objects. In *Proceedings of the International Symposium on Distributed Objects and Applications (DOA '09)*, p. 132, Los Alamitos, CA, EUA. IEEE Computer Society.

Fetzer, C.; Reynal, M. & Tronel, F. (2001). An adaptive failure detection protocol. In *Proceedings of the 2001 IEEE Pacific Rim International Symposium on Dependable Computing*, pp. 146–156, Los Alamitos, CA, EUA. IEEE Computer Society.

Fischer, M. J.; Lynch, N. A. & Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382.

Friedman, R. (2003). Fuzzy group membership. In *Future Directions in Distributed Computing*, volume 2584 of *Lecture Notes in Computer Science*, pp. 114–118. Springer.

Friedman, R.; Tcharny, G. & LTD, I. (2005). Evaluating failure detection in mobile ad-hoc networks. *International Journal of Wireless and Mobile Computing*, 1(8).

Greve, F. G. P. & Narzul, J.-P. L. (2005). Generating fast atomic commit from hyperfast consensus. In *Dependable Computing*, volume 3747 of *Lecture Notes in Computer Science*, pp. 226–244. Springer.

Guerraoui, R. (2000). Indulgent algorithms (preliminary version). In *Proceedings of the* 19^{th} *annual ACM symposium on Principles of Distributed Computing (PODC '00)*, pp. 289–297, New York, NY, EUA. ACM.

Guerraoui, R. (2002). Non-blocking atomic commit in asynchronous distributed systems with failure detectors. *Distributed Computing*, 15(1):17–25.

Guerraoui, R. & Kouznetsov, P. (2002). On the weakest failure detector for non-blocking atomic commit. In *Proceedings of the 2002 IFIP International Conference on Theoretical Computer Science (IFIP TCS '02)*, volume 223 of *IFIP Conference Proceedings*, pp. 461–473, Montréal, Québec, Canadá. Kluwer.

Gupta, I.; Chandra, T. D. & Goldszmidt, G. S. (2001). On scalable and efficient distributed failure detectors. In *Proceedings of the* 20^{th} *annual ACM Symposium on Principles of Distributed Computing (PODC '01)*, pp. 170–179, New York, NY, EUA. ACM.

Hayashibara, N. (2004). *Accrual Failure Detectors*. PhD thesis, Japan Advanced Institute of Science and Technology.

Hayashibara, N.; Défago, X.; Yared, R. & Katayama, T. (2004). The ϕ accrual failure detector. In *Proceedings of the 2004 IEEE Symposium on Reliable Distributed Systems*, pp. 66–78, Los Alamitos, CA, EUA. IEEE Computer Society.

Huang, M.; Bavier, A. & Peterson, L. (2006). Planetflow: maintaining accountability for network services. *Operating Systems Review*, 40(1):89–94.

Jacobson, V. (1995). Congestion avoidance and control. *SIGCOMM Computer Communication Review*, 25(1):157–187.

Jain, R. (1991). *The art of computer systems performance analysis*. John Wiley & Sons, New York, EUA.

Liang, J.; Ko, S. Y.; Gupta, I. & Nahrstedt, K. (2005). Mon: management overlay networks for distributed systems. In *Proceedings of the* 20^{th} *ACM symposium on Operating systems principles (SOSP '05)*, pp. 1–2, New York, NY, EUA. ACM.

Lynch, N. (1989). A hundred impossibility proofs for distributed computing. In *Proceedings of the* 8^{th} *annual ACM Symposium on Principles of Distributed Computing (PODC '89)*, pp. 1–28, New York, NY, EUA. ACM.

Mo, T. (1984). Chebyshev inequality with estimated mean and variance. *American Statistician*, pp. 130–132.

Montresor, A. (1999). The jgroup distributed object model. In *Proceedings of the* 2^{nd} *International Working Conference on Distributed Applications and Interoperable Systems (DAIS '99)*, volume 143 of *IFIP Conference Proceedings*, Helsinki, Finlândia. Kluwer.

Montresor, A. (2006). Jgroups - a toolkit for reliable multicast communication. Disponível em `http://www.jgroups.org/javagroupsnew/docs/overview.html`. Visitado em 2007-11-07.

Mostéfaoui, A. & Reynal, M. (1999). Solving consensus using chandra-toueg's unreliable failure detectors: A general quorum-based approach. In *Proceedings of the* 13^{th} *International Symposium on Distributed Computing (DISC'99)*, volume 1693 of *Lecture Notes in Computer Science (LNCS)*, pp. 49–63, Bratislava, Eslováquia. Springer.

Mota, V. F. S.; Silva, T. H. & Nogueira, J. M. (2009). Introduzindo Tolerância a Interrupção em Redes AdHoc Móveis para Cenários de Emergência. In *Proceedings of XXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC '09)*, pp. 671–684, Recife, Brasil.

Nunes, R. & Jansch-Pôrto, I. (2002). Non-stationary communication delays in failure detectors. In *IEEE Latin-American Test Workshop (LATW '02)*, pp. 16–21, Montevideo, Uruguay. IEEE Computer Society.

Nunes, R. & Jansch-Porto, I. (2004). Qos of timeout-based self-tuned failure detectors: the effects of the communication delay predictor and the safety margin. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN '04)*, pp. 753–761, Washington, DC, EUA. IEEE Computer Society.

Oppenheimer, D.; Albrecht, J.; Patterson, D. & Vahdat, A. (2005). Design and implementation tradeoffs for wide-area resource discovery. In *Proceedings of* 14^{th} *IEEE International Symposium on High-Performance Distributed Computing (HPDC-14)*, pp. 113–124, Los Alamitos, CA, EUA. IEEE Computer Society.

Park, K. & Pai, V. S. (2006). Comon: a mostly-scalable monitoring system for planetlab. *SIGOPS Operating Systems Review*, 40(1):65–74.

Pathak, A.; Pucha, H.; Zhang, Y.; Hu, Y. C. & Mao, Z. M. (2008). A measurement study of internet delay asymmetry. In *Proceedings of the 9th International Conference on Passive and Active Network Measurement (PAM '08)*, volume 4979 of *Lecture Notes in Computer Science*, pp. 182–191, Cleveland, OH, EUA. Springer.

Pease, M.; Shostak, R. & Lamport, L. (1980). Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234.

Peterson, L. & Roscoe, T. (2006). The design principles of planetlab. *SIGOPS Operating Systems Review*, 40(1):11–16.

PlanetLab (2008). Planetlab - an open platform for developing, deploying, and accessing planetary-scale services. Disponível em `https://www.planet-lab.org/`. Visitado em 2009-02-05.

R Development Core Team (2008). R: A language and environment for statistical computing. Disponível em `http://www.R-project.org`. Visitado em 2009-02-05.

Redígolo, F.; de Lima, M.; Silva, M.; Souza, J. & de Brito Carvalho, T. (2007). PlanetLab: Ambiente para Desenvolvimento e Pesquisa de Aplicações Distribuídas. In *Minicursos do XXV Simpósio Brasileiro de Redes de Computadores (SBRC '07)*, pp. 163–202. SBC.

Reynal, M. (2005). A short introduction to failure detectors for asynchronous distributed systems. *SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 36(1):53–70.

Ryan, B.; Cryer, J. & Joiner, B. (2005). *Minitab Handbook: Updated for Release 14*. Brooks/Cole - Thomson Learning.

Sampaio, L. & Brasileiro, F. (2001). Deploying Fault-Tolerant Processing Services for Asynchronous Distributed Systems. In *Proceedings of the* 2^{nd} *IEEE Latin American Test Workshop*, Cancun, México.

Sampaio, L. & Brasileiro, F. (2005). Adaptive indulgent consensus. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN '05)*, pp. 422–431, Yokohama, Japão.

Sampaio, L.; Brasileiro, F.; Cirne, W. & Figueiredo, J. (2003). How bad are wrong suspicions? towards adaptive distributed protocols. In *Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN '03)*, pp. 551–560, San Francisco, EUA.

Satzger, B.; Pietzowski, A.; Trumler, W. & Ungerer, T. (2007). A new adaptive accrual failure detector for dependable distributed systems. In *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC '07)*, pp. 551–555, New York, NY, EUA. ACM.

SBC (2006). Grandes desafios da pesquisa em computação no brasil: 2006 - 2016. In *Relatório sobre o Seminário realizado em 8 e 9 de maio de 2006*, São Paulo, Brasil. Sociedade Brasileira de Computação.

Silva, T. H.; Mota, V. F. S.; Valadão, E.; Almeida, J. & Guedes, D. (2009). Caracterização do Comportamento dos Espectadores em Transmissões de Vídeo ao Vivo Geradas por Usuários. In *Proceedings of XXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC '09)*, pp. 613–626, Recife, Brasil.

Smith, A. J. (2007). Workloads (creation and use). *Communications of the ACM*, 50(11):45–50.

So, K. C. W. & Sirer, E. G. (2007). Latency and bandwidth-minimizing failure detectors. In *Proceedings of the* 2^{nd} *ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys '07)*, pp. 89–99, New York, NY, EUA. ACM.

Sotoma, I. (2006). *Qualidade de serviço de detectores de defeitos na presença de rajadas de perdas de mensagens*. PhD thesis, Universidade Estadual de Campinas.

Sotoma, I. & Madeira, E. (2001). Adaptation-algorithms to adaptive fault monitoring and their implementation on CORBA. In *Proceedings of the* 3^{rd} *International Symposium on Distributed-Objects and Applications (DOA '01)*, pp. 219–228, Los Alamitos, CA, EUA. IEEE Computer Society.

Stelling, P.; DeMatteis, C.; Foster, I.; Kesselman, C.; Lee, C. & von Laszewski, G. (1999a). A fault detection service for wide area distributed computations. *Cluster Computing*, 2(2):117–128.

Stelling, P.; DeMatteis, C.; Foster, I. T.; Kesselman, C.; Lee, C. A. & von Laszewski, G. (1999b). A fault detection service for wide area distributed computations. *Cluster Computing*, 2(2):117–128.

Stephens, M. (1974). EDF statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, pp. 730–737.

Stribling, J. (2005). All-sites-pings (app). Disponível em `http://pdos.csail.mit.edu/~strib/pl_app/`. Visitado em 2009-02-05.

Tang, L.; Chen, Y.; Li, F.; Zhang, H. & Li, J. (2007). Empirical study on the evolution of planetlab. In *Proceedings of the 2007 International Conference on Networking (ICN '07)*, p. 64, Los Alamitos, CA, EUA. IEEE Computer Society.
ao

Turchetti, R. & Nunes, R. (2006). Otimização de Algoritmos Para Detecção de Defeitos: um mecanismo para a produção de Softwares confiáveis em ambientes de larga escala. In *Proceedings of the XXVI Encontro Nacional de Engenharia de Produção (ENEGEP '06)*, Fortaleza, Brasil.

Valadão, E. (2007). Protocolo para detecção de falhas no jgroups com sensibilidade adaptativa. Technical report, Universidade Federal de Ouro Preto, Ouro Preto, Brasil.

van Renesse, R.; Minsky, Y. & Hayden, M. (1998). A gossip-style failure detection service. In *Proceedings of Middleware*, volume 98, pp. 55–70.

Verissimo, P. & Rodrigues, L. (2004). *Distributed Systems for System Architects*. Kluwer Academic Publishers, Massachussets,EUA, 2ª edição.

Wang, C.-Y. & Buehrer, D. J. (2008). A ring-based decentralized collaborative non-blocking atomic commit protocol. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, pp. 395–398, Los Alamitos, CA, EUA. IEEE Computer Society.

Witten, I. H. & Frank, E. (2002). Data mining: practical machine learning tools and techniques with java implementations. *SIGMOD Record*, 31(1):76–77.

Yang, J.; Cao, J.; Wu, W. & Travers, C. (2006). The notification based approach to implementing failure detectors in distributed systems. In *Proceedings of the 1st International Conference on Scalable Information Systems (InfoScale '06)*, p. 14, New York, NY, USA. ACM.

Yoshikawa, C. (2006). Planetlab - all sites pings. Disponível em `http://ping.ececs.uc.edu/ping/`. Visitado em 2009-02-05.

Zhang, J. & Honeyman, P. (2008). Performance and availability tradeoffs in replicated file systems. In *Proceedings of the 2008 IEEE International Symposium on Cluster Computing and the Grid (CCGRID '08)*, pp. 771–776, Los Alamitos, CA, EUA. IEEE Computer Society.

Zhuang, S. Q.; Geels, D.; Stoica, I. & Katz, R. H. (2005). On failure detection algorithms in overlay networks. In *Proceedings of the* 24^{th} *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '05)*, volume 3, pp. 2112–2123.

Apêndice A

Glossário

7Zip : software utilizado para alta compressão de dados.

ack (acknowledgement) : sinal ou mensagem utilizada para reconhecimento, acusando o recebimento de algo.

broadcast : estilo de transmissão onde cada mensagem enviada será recebida por todo dispositivo na rede.

caching : consiste no armazenamento temporário de arquivos frequentemente requisitados, visando melhorias de desempenho.

CDN : é o acrônimo de *content distribution network*, consistindo numa rede de distribuição de conteúdo.

clustering : em análise de dados é a atribuição de um conjunto de observações em subconjuntos (chamados de *clusters*) de maneira que as observações em um mesmo *cluster* sejam similares segundo algum critério.

cluster : um grupo de computadores interconectados que trabalham cooperativamente, tipicamente sendo utilizados para melhorar o desempenho e/ou disponibilidade em relação à um único computador.

daemon : um programa que é executado em segundo plano e recebe instruções/pedidos de outros programas para a execução de determinada ação.

datagrama : pacote transmitido via um serviço não-confiável (como o UDP).

DHT : é o acrônimo de *distributed hash table*, consistindo numa tabela *hash* distribuída.

downstream : refere-se à direção na qual os dados são recebidos de um outro computador, também conhecido como *downloading*.

FD : acrônimo de *failure detector*, ou seja, detector de falhas.

framework : uma coleção de bibliotecas de software reutilizável, consistindo no esqueleto através do qual pode-se basear a solução de problemas correlatos.

fuzzy : difuso. Ao contrário de um valor booleano do tipo Falso(0) ou Verdadeiro(1), um valor difuso admite valores intermediários (0<Talvez<1). No contexto de grupos de processos, representam o grau no qual um processo pertence ao grupo.

grid : uma grade computacional pode ser vista como um *cluster* de computadores fracamente acoplados e distribuído em larga escala geográfica, visando desempenhar tarefas muito grandes e/ou melhorar a disponibilidade de um serviço.

GZip : software utilizado para compressão de dados, muito comum em instalações UNIX.

hardware : componente físico de um sistema computacional.

hash table : uma estrutura de dados que associa chaves a valores utilizando uma função de dispersão.

heartbeat : mensagem periódica enviada com o significando de que o emissor está "vivo".

histerese : consiste na tolerância à pequenas variações onde o estado atual depende das propriedades dos estados anteriores e é intencionalmente adicionada para prevenir chaveamentos (troca de estados) rápidos.

jitter : consiste na na variação do atraso de rede.

LAN : é o acrônimo de *local area network*, consistindo numa rede local de computadores cobrindo uma pequena área física como uma casa, escritório ou mesmo um pequeno grupo de prédios como uma escola ou aeroporto.

liveness : propriedade de uma aplicação de continuar seu propósito de existência (de acordo com sua especificação).

log : um registro sequencial de dados, ordenado cronologicamente.

middleware : uma camada de intermediação entre softwares ou aplicações, provendo uma série de serviços que permite múltiplos processos interagirem entre si.

multicast : estilo de transmissão onde cada mensagem será enviada simultaneamente à um grupo de destinatários interessados.

overlay : uma rede de computadores construída em cima de outra rede de maneira a agregar algum valor, tendo seus nós conectados através de enlaces virtuais que podem passar por um ou mais enlaces físicos.

P2P : é o acrônimo de *peer-to-peer*, consistindo numa rede distribuída do tipo par-a-par, onde os nós participantes oferecem uma porção de seus recursos (processamento, armazenamento ou largura de banda) aos demais. Assim, cada par se comporta tanto como fornecedor quanto como consumidor de recursos, em contraste com o modelo tradicional de cliente-servidor.

piggyback : técnica de transmissão de mensagens na qual adiciona-se informações extras à um pacote que já seria enviado, economizando-se pela não necessidade de enviar um pacote extra.

ping : mensagem enviada com o propósito de checar se uma determinada máquina está acessível, de maneira que este tipo de solicitação requer o envio de uma mensagem de "eco" como resposta.

slice : conjunto de nós nos quais o usuário do PlanetLab recebe uma porção dos recursos de cada nó, formando assim uma coleção de máquinas virtuais a qual ele tem acesso.

software : programa de computador que desempenha uma determinada tarefa para um usuário.

testbed : ambiente de testes onde experimentos podem ser executados.

trace : arquivo contendo uma sequência de mensagens capturadas durante seu envio ou recebimento, ordenadas por seus respectivos *timestamps*.

upstream : refere-se à direção na qual os dados são transferidos para um outro computador, também conhecido como *uploading*.

uptime : o período de tempo no qual um computador está operativo ou pode ser operado.

WAN : WAN é o acrônimo de *wide-area network*, consistindo numa rede de longa distância, ou seja, abrangendo uma grande área geográfica como um estado, país ou continente(s). Um exemplo prático é a própria Internet.

Apêndice B

Lista de Nós do *Trace*

ID	***Nó***	**País**
1	gschembra4.diit.unict.it	Itália
2	planetlab2.cis.upenn.edu	Estados Unidos
5	server1.planetlab.iit-tech.net	Estados Unidos
10	planetlab-1.dis.uniroma1.it	Itália
14	onelab03.inria.fr	França
22	planet1.cs.huji.ac.il	Israel
23	planetslug4.cse.ucsc.edu	Estados Unidos
36	planet1.ku.edu.tr	Turquia
41	planetlab2.dtc.umn.edu	Estados Unidos
52	planetlab-2.cs.ucy.ac.cy	Chipre
60	planetlab-2.usask.ca	Canadá
66	planetlab2.postel.org	Estados Unidos
79	node2pl.planet-lab.telecom-lille1.eu	França
81	planetlab1.nrl.dcs.qmul.ac.uk	Inglaterra
83	planetlab1.cs.unibo.it	Itália
88	planetlab3.gdansk.rd.tp.pl	Polônia
89	planetlab3.upc.es	Espanha
95	iraplab1.iralab.uni-karlsruhe.de	Alemanha
96	planetlab2.ics.forth.gr	Grécia

Tabela B.1. Lista dos Nós Desconsiderados na Análise

ID	*Nó*	País
3	planetlab2.ifi.uio.no	Noruega
4	planetlab2.informatik.uni-goettingen.de	Alemanha
6	planetlab2.di.unito.it	Itália
7	node1.planetlab.mathcs.emory.edu	Estados Unidos
8	stella.planetlab.ntua.gr	Grécia
9	planetlab-2.cs.colostate.edu	Estados Unidos
11	plab2-c703.uibk.ac.at	Áustria
12	planetlab2.tmit.bme.hu	Hungria
13	pl2.csl.utoronto.ca	Canadá
15	planetlab2.cs.duke.edu	Estados Unidos
16	plab-2.sinp.msu.ru	Rússia
17	pl2.ucs.indiana.edu	Estados Unidos
18	planetlab-3.cmcl.cs.cmu.edu	Estados Unidos
19	itchy.cs.uga.edu	Estados Unidos
20	planetlab2.csres.utexas.edu	Estados Unidos
21	planetlab3.comp.nus.edu.sg	Singapura
24	planetlab3.williams.edu	Estados Unidos
25	plab2.larc.usp.br	Brasil
26	planetlab-3.imperial.ac.uk	Inglaterra
27	planetlab2.elet.polimi.it	Itália
28	planetlab1.netmedia.gist.ac.kr	Coréia do Sul
29	netapp6.cs.kookmin.ac.kr	Coréia do Sul
30	planetlab2.pop-rs.rnp.br	Brasil
31	merkur.planetlab.haw-hamburg.de	Alemanha
32	planetlab2.ie.cuhk.edu.hk	Hong Kong (China)
33	planetlab1.cs.uoi.gr	Grécia
34	planetlab3.mini.pw.edu.pl	Polônia
35	vicky.planetlab.ntua.gr	Grécia
37	csplanetlab2.kaist.ac.kr	Coréia do Sul
38	planetlab02.cs.washington.edu	Estados Unidos
39	plab1-itec.uni-klu.ac.at	Áustria
40	pl2.bit.uoit.ca	Canadá
42	pl1-higashi.ics.es.osaka-u.ac.jp	Japão
43	planetlab-1.ssvl.kth.se	Suécia
44	planetlab2.comp.nus.edu.sg	Singapura
45	planetlab4.cse.nd.edu	Estados Unidos
46	planetlab1.cs.wisc.edu	Estados Unidos
47	planetlab2.cnds.jhu.edu	Estados Unidos
48	planetlab4.inf.ethz.ch	Suíça
49	planetlab02.cnds.unibe.ch	Suíça

Tabela B.2. Lista dos Nós Utilizados - Parte I

ID	*Nó*	País
50	pads21.cs.nthu.edu.tw	Formosa (Taiwan)
51	planet1.cs.ucsb.edu	Estados Unidos
53	planetlab-02.naist.jp	Japão
54	planetlab1.cs.purdue.edu	Estados Unidos
55	planetlab1.ls.fi.upm.es	Espanha
56	planetlab-2.ece.iastate.edu	Estados Unidos
57	planetlabnode-2.docomolabs-usa.com	Estados Unidos
58	planetlab-04.naist.jp	Japão
59	planetlab2.simula.no	Noruega
61	planetlab1.fri.uni-lj.si	Eslovénia
62	planetlab-2.tagus.ist.utl.pt	Portugal
63	planetlab2.utdallas.edu	Estados Unidos
64	planetlab2.ceid.upatras.gr	Grécia
65	pl2.eecs.utk.edu	Estados Unidos
67	planetlab2.cesnet.cz	República Checa
68	planetlab3.ani.univie.ac.at	Áustria
69	pl3.planetlab.uvic.ca	Canadá
70	pl2.snu.ac.kr	Coréia do Sul
71	planetlab1.cs.uit.no	Noruega
72	pli1-pa-5.hpl.hp.com	República Checa
73	lsirextpc02.epfl.ch	Suíça
74	planetlab-2.cs.uh.edu	República Checa
75	planetlab5.millennium.berkeley.edu	República Checa
76	planet-lab2.ufabc.edu.br	Brasil
77	planetlab2.hiit.fi	Finlândia
78	planetlab-2.iscte.pt	Portugal
80	planet2.l3s.uni-hannover.de	Alemanha
82	plab-1.sinp.msu.ru	Rússia
84	planetlab04.mpi-sws.mpg.de	Alemanha
85	planetlab2.mnlab.cti.depaul.edu	Estados Unidos
86	planetlab2.iii.u-tokyo.ac.jp	Japão
87	planetlab2.cs.ubc.ca	Canadá
90	planetlab1.informatik.uni-kl.de	Alemanha
91	node-1.mcgillplanetlab.org	Canadá
92	planetlab1.pop-mg.rnp.br	Brasil
93	planetlab2.tlm.unavarra.es	Espanha
94	ricepl-2.cs.rice.edu	Estados Unidos
97	planetlab2.fct.ualg.pt	Portugal
98	planetlab3.cs.columbia.edu	Estados Unidos
99	planetlab2.georgetown.edu	Estados Unidos
100	node1.lbnl.nodes.planet-lab.org	Estados Unidos

Tabela B.3. Lista dos Nós Utilizados - Parte II