Avaliação experimental de estratégias de tolerância a falhas do protocolo SCTP por injeção de falhas

Tórgan Siqueira, Bruno Fiss, Cristina Menegotto, Sergio Cechin, Taisy Weber

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

{torgan,bcfiss,ccmenegotto,cechin,taisy}@inf.ufrgs.br

Abstract*: The use of fault injection to evaluate the fault tolerance strategies of a SCTP implementation is the main focus of this paper. To perform the evaluation we use a tool that injects faults directly into the messages that pass through the kernel protocol stack. We evaluate the fault coverage and the performance degradation of SCTP under link crashes and loss of messages. The experiments show excellent fault coverage with just a small penalty on performance. The experiments also show that fault injection is a powerful experimental technique that allows reaching significant dependability measures with the advantage of an agile fault scenario generation with total control over the fault type and its activation time.*

Resumo*. Esse artigo relata experimentos de injeção de falhas sobre o SCTP conduzidos para avaliar estratégias de tolerância a falhas implementadas neste protocolo. Para avaliação experimental, usou-se um injetor que opera sobre a pilha de protocolos no kernel. Os experimentos mostram uma excelente cobertura de falhas do SCTP para colapso de enlaces e perda de pacotes ao custo de uma pequena queda de desempenho sob falhas. Mostram também que injeção de falhas é uma técnica de avaliação experimental que conduz a métricas significativas de confiabilidade com a vantagem do total controle sobre o tipo de falha e o instante de sua ativação, substituindo a imprevisibilidade da ocorrência de falhas reais de comunicação.*

1 Introdução

Servidores de alta disponibilidade, enlaces replicados, componentes digitais de alta confiabilidade, codificação para detectar e corrigir erros de transmissão são estratégias de tolerância a falhas usuais empregadas para tornar sistemas baseados em comunicação mais robustos. Mas não suficientes. Também os protocolos de rede precisam prover maior confiabilidade para as aplicações empregando estratégias que vão além da simples confirmação de mensagens e verificação de alteração de conteúdo.

Recentemente começaram a ser disponibilizados protocolos de rede, como o SCTP, que prometem garantir níveis mais adequados de confiabilidade do que o TCP. Esses protocolos garantem a correção dos erros provocados por falhas como perdas de mensagens e colapso de enlaces. Descrito inicialmente na RFC 2960 [Stewart et al. 2000] e consolidado na RFC 4960 [Stewart 2007], o SCTP foi originalmente desenvolvido para transmitir mensagens de sinalização da Rede de Telefonia Pública

Comutada sobre IP. SCTP promete conferir alta confiabilidade às aplicações de rede aplicando estratégias eficientes de tolerância a falhas.

As estratégias de tolerância a falhas implementadas devem, entretanto, ser ativadas para avaliar sua correção. Como todo software, os trechos de código que detectam e corrigem falhas podem conter erros de lógica, de má interpretação da especificação e de codificação. Como esse código só entra em operação na ocorrência de falhas, tais falhas devem estar presentes durante a avaliação do protocolo.

Para não ficar dependente da ocorrência natural e esparsa de falhas no ambiente real de operação, uma técnica eficaz para a avaliação experimental [Hsueh et al. 1997] das estratégias de tolerância a falhas é emular falhas reais por software e injetá-las no protocolo sob teste. Através da introdução controlada de falhas [Arlat et al. 1990], é possível determinar se a estratégia tolera ou não as falhas injetadas e qual o custo relacionado. Injeção de falhas é tanto uma ferramenta para teste e determinação de métricas (como cobertura de falhas e queda de desempenho sob falhas), como também para benchmark de dependabilidade [Kanoun and Spainhower 2008] comparando diferentes implementações de um mesmo sistema sujeitas às mesmas falhas.

Um experimento de injeção de falhas é uma rodada de avaliação executada em condições controladas na qual são introduzidas falhas. Seguindo uma carga de falhas, essas falhas são introduzidas em um sistema que executa uma carga de trabalho. Tanto a carga de trabalho como a carga de falhas devem ser representativas do uso do sistema em condições reais de operação. Para avaliação de protocolos, a carga de trabalho são as mensagens trocadas entre os *peers* e a carga de falhas inclui eventos indesejáveis como queda de enlaces e perda de mensagens.

Esse artigo relata experimentos de injeção de falhas de comunicação conduzidos com o objetivo de avaliar estratégias de tolerância a falhas na implementação do protocolo SCTP. É mostrado como é possível avaliar o comportamento de um protocolo na presença de falhas de forma controlada usando um injetor de falhas. Entre os vários injetores de falhas de comunicação mencionados na literatura, um injetor [Drebes 2005] que opera sobre a pilha de protocolos no kernel usando recursos do NetFilter [Russel and Welte 2002] foi aplicado principalmente por estar sendo desenvolvido nos laboratórios onde avaliação foi conduzida. Foi escolhido também por apresentar um alto poder de expressão de carga de falhas e por permitir trabalhar com novos protocolos de rede baseados em IP, características essas necessárias a experimentação proposta e não facilmente encontradas nos demais injetores.

Os experimentos realizados mostram uma excelente cobertura de falhas do SCTP para as cargas de falhas escolhidas, o que comprova a adequação da implementação do protocolo à sua especificação nesses casos. Os experimentos mostram também uma pequena queda de desempenho em caso de falhas comparada a situações ideais, sem falhas. Tal fato indica que o protocolo pode ser uma excelente opção para aplicações que exigem uma maior confiabilidade nas transmissões de dados, mas que não prescindem de um mínimo de desempenho quando operando sob falhas. Os experimentos mostram finalmente que, quando conduzida com as ferramentas adequadas, injeção de falhas é uma atividade de avaliação experimental que traz resultados significativos para o cômputo de métricas relacionadas à confiabilidade sem a necessidade de esperar pela ocorrência natural de falhas.

Nas próximas seções serão apresentadas as características de tolerância a falhas do protocolo alvo SCTP, a metodologia aplicada para a condução da avaliação, as ferramentas usadas, e os experimentos conduzidos sobre o SCTP. O artigo segue com uma discussão sobre os resultados alcançados e trabalhos relacionados. Finalmente, o artigo encerra com o relato das dificuldades encontradas na condução dos experimentos e análise das ferramentas usadas.

2 Tolerância a falhas no SCTP

O SCTP (*Stream Control Transmission Protocol*) é um protocolo da camada de transporte como o UDP e o TCP, especificado na RFC 4960 [Stewart 2007]. De forma semelhante ao TCP, pode fornecer um serviço de transporte confiável. Entretanto, o nível de confiabilidade desejado pode ser ajustado pelo usuário, podendo chegar a oferecer um desempenho comparável ao do UDP: o usuário pode escolher a relação confiabilidade *versus* desempenho mais adequada a sua aplicação.

Como no TCP, é necessário que origem e destino negociem uma forma de conexão, chamada de associação no SCTP. Dessa forma, o protocolo suporta o envio de informações entre dois pontos que, diferentemente do TCP, podem ser representados por múltiplos endereços IP. Uma associação SCTP é unidirecional, de forma que são necessárias duas associações para estabelecer uma comunicação bidirecional. Sobre uma associação SCTP podem ser transportados vários *streams* independentes de dados. Dessa forma, o controle de fluxo e de erros é feito por *stream* e não por conexão, como ocorre com o TCP. A possibilidade de transportar vários *streams* reflete-se no formato do pacote SCTP. Um pacote SCTP possui um *header* comum seguido por uma série de unidades de informação denominadas *chunks*, que fornecem o isolamento necessário aos *streams* de dados e às informações de controle da associação.

O protocolo SCTP, em sua RFC 4960, oferece vários serviços. Entretanto, a avaliação nesse artigo está focada nos serviços que visam tolerância a falhas como os listados a seguir:

Entrega ordenada de mensagens em um stream: Para garantir entrega ordenada, SCTP usa uma numeração das mensagens da aplicação associada a cada *stream* de dados (*Stream Sequence Number* - SSN), que é diferente do TCP. Ao ser aberta uma associação, os dispositivos transmissor e receptor negociam o número de *streams* disponíveis e se as mensagens devem ser ordenadas ou não, em cada um deles. Com isso, a aplicação pode enviar mensagens através de vários *streams* de uma mesma associação de forma a satisfazer os diferentes requisitos de ordenação.

Entrega confiável: O SCTP usa mensagens de confirmação para garantir a entrega das mensagens da aplicação. Faz parte desse serviço a numeração de *chunks* de dados. Essa numeração gera os TSNs (*Transmission Sequence Number)* que são anexados aos *chunks* de dados e na mensagem de confirmação correspondente. Como o TSN é independente da numeração usada para garantir a ordenação em um *stream* (SSNs), os dois mecanismos operam de forma independente. Isso permite que uma aplicação possa escolher a combinação mais adequada de mecanismos.

Validação de pacotes SCTP: Um pacote SCTP carrega dois campos: *verification tag* e *checksum*. O campo *verification tag* é usado pelo receptor para identificar de forma inequívoca os pacotes enviados através de uma associação. O campo *checksum* é usado para verificar a integridade do pacote através de CRC32c. Ao receber um pacote, o

receptor verifica os dois campos e, no caso de não receber o *verification tag* esperado ou detectar erro no valor do *checksum*, o pacote será silenciosamente descartado.

Gerenciamento de rotas: Esse mecanismo, no lado do transmissor, é capaz de manipular um conjunto de endereços da camada de transporte a serem usados como destino para os pacotes SCTP. Para escolher o endereço de destino de cada pacote, o gerenciamento de rotas baseia-se nas instruções do usuário e na alcançabilidade desses endereços. Uma rota principal é estabelecida, através da qual serão enviados os primeiros pacotes da associação.

Mensagens de *Heartbeat*: O SCTP usa mensagens de *heartbeat* (semelhante a *Keep-alive* [Braden 1989] de algumas implementações do TCP) para detectar a alcançabilidade dos endereços de destino da lista obtida no estabelecimento da associação SCTP.

Failover: O mecanismo de *failover*, que no SCTP corresponde ao monitoramento da rota principal e o seu chaveamento para uma rota alternativa, usa as mensagens de *heartbeat* para o monitoramento e o gerenciamento de rotas para o chaveamento. Adicionalmente, a aplicação pode solicitar um chaveamento de rotas. A especificação do SCTP não determina o critério de escolha da rota alternativa. Apenas é indicado o uso da rota com a maior diversidade.

3 Metodologia de avaliação

Para avaliar a implementação das estratégias de tolerância a falhas do SCTP, seguiu-se a metodologia [Menegotto et al. 2007] aplicada na avaliação de OurGrid [Cirne et al. 2006]. Essa metodologia foi desenvolvida com base na proposta de Jain para avaliação de desempenho [Jain 1991], que consiste nos passos detalhados a seguir. Para adequar essa metodologia à avaliação de estratégias de tolerância a falhas, foram incorporadas as noções de injeção de falhas e de carga de falhas.

Objetivos da avaliação, limites do sistema e serviços avaliados

O objetivo neste artigo é avaliar as estratégias de tolerância a falhas da lksctp [Yarroll and Knutson 2001], uma implementação do SCTP para o Linux. Ela é composta de duas partes: o lksctp, e a biblioteca de usuário, lksctp-tools. A biblioteca oferece funções tanto para quem desenvolve o próprio lksctp quanto para os desenvolvedores de aplicações e foi usada para gerar a carga de trabalho e efetuar o monitoramento da comunicação nesta avaliação. A versão usada é a 1.0.6.

Testes preliminares [Cechin et al. 2008] foram conduzidos anteriormente sem a abrangência e aprofundamento apresentados neste artigo. Os testes preliminares avaliaram os serviços de validação de pacotes SCTP e descarte de mensagens duplicadas, mostrando uma cobertura de 100% para as falhas injetadas. Os serviços avaliados aqui neste artigo são entrega confiável de mensagens e recuperação de queda de enlace (*link failure* na RFC 4960). Esses dois serviços, foco dos experimentos aqui relatados, são os mais abrangentes englobando os demais serviços listados na seção 2.

Métricas

A cobertura de falhas é computada pela razão entre as falhas corrigidas e as falhas injetadas. Desempenho sob falhas é computado medindo o tempo total necessário para completar a carga de trabalho de cada experimento sob falha comparando com o tempo total de um experimento de referência, sem falhas. A degradação nos casos de *failover*

não é significativa devido à baixa frequência de ocorrência de quedas de enlaces em situações reais de operação.

Parâmetros do sistema, da carga de trabalho e de falhas e seleção de fatores

Parâmetros do sistema são aqueles que podem afetar as métricas selecionadas, como a quantidade e a configuração dos computadores e suas interconexões. Os experimentos foram executados usando um cliente e um servidor SCTP conectados por três enlaces: um principal e dois auxiliares que atuam como caminhos redundantes (figura 1). O enlace principal interconecta diretamente as duas máquinas, um auxiliar usa um HUB e o outro a rede do laboratório. Para ajustes de confiabilidade do protocolo foram usados o serviço default do lksctp: conexão confiável (*reliable connection-oriented stream*) com entrega ordenada e recuperação de mensagens perdidas. *Failover* é intrínseco ao SCTP e não um parâmetro variável de confiabilidade.

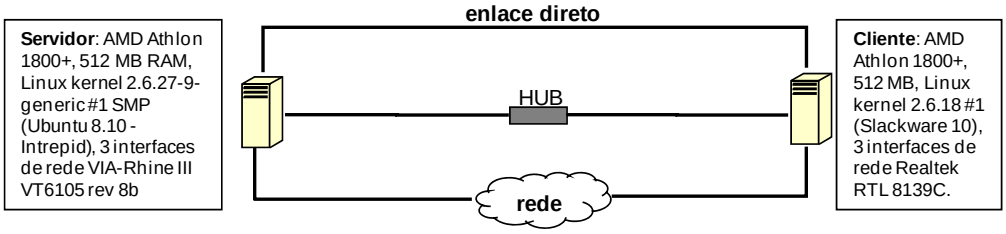

Figura 1 – Configuração do sistema sob avaliação

Parâmetros de carga de trabalho incluem número e tamanho dos pacotes. Finalmente parâmetros de carga de falhas incluem: tipos de falhas, instantes de ativação, frequência e distribuição de probabilidade. Os parâmetros que serão variados durante a avaliação são chamados fatores. Considerando que se deseja avaliar tolerância a falhas, escolheu-se usar os parâmetros da carga de falhas como os fatores na avaliação.

Seleção de técnicas de avaliação

Injeção de falhas, modelagem analítica e avaliação com dados de campo durante a vida de um sistema [Arlat 1990] são algumas das técnicas disponíveis para avaliação de dependabilidade. Considerando que já existem implementações em uso do protocolo, a estratégia mais adequada é injeção de falhas, pois permite acelerar a ocorrência de falhas e não necessita de modelos analíticos complexos. O protocolo também determina o tipo de injetor de falhas mais adequado para a avaliação, que deve operar sobre IP e ser capaz de manipular pacotes diretamente.

Seleção de carga de falhas e de carga de trabalho

Nas metodologias convencionais de avaliação de desempenho, seleção de carga de falhas não existe. Para injeção de falhas, entretanto, é preciso definir além de uma carga de trabalho, uma carga de falhas a ser injetada no protocolo. Tanto a carga de falhas como a de trabalho serão detalhadas nas seções que tratam dos experimentos.

Projeto dos experimentos, análise, interpretação e apresentação dos resultados

Cada experimento compreende a repetição de um mesmo teste um número suficiente de vezes para que sejam obtidas medidas estatisticamente significativas. Dois experimentos foram previstos para avaliar os serviços selecionados e serão apresentados na seção 6.

Medidas obtidas dos experimentos são quantidades randômicas, pois o resultado pode ser diferente a cada vez que o experimento é repetido [Jain 1991]. A análise

somente produz resultados e não conclusões. Os resultados provêem uma base sobre a qual é possível tirar conclusões. Essa etapa da metodologia justifica a quantidade de medidas realizadas. Para garantir a representatividade dos resultados, cada teste em cada experimento foi repetido 200 vezes.

4 Ambiente para os experimentos

O ambiente experimental injeta falhas de comunicação e monitora o comportamento do protocolo enquanto executa uma carga de trabalho. É composto pelo injetor de falhas, FIRMAMENT (*Fault Injection Relocatable Module for Advanced Manipulation and Evaluation of Network Transports*) [Drebes 2005], e demais componentes especialmente desenvolvidos para os experimentos. A figura 2 mostra os componentes do ambiente complementados pelo analisador de protocolo Wireshark [Combs et al. 2006].

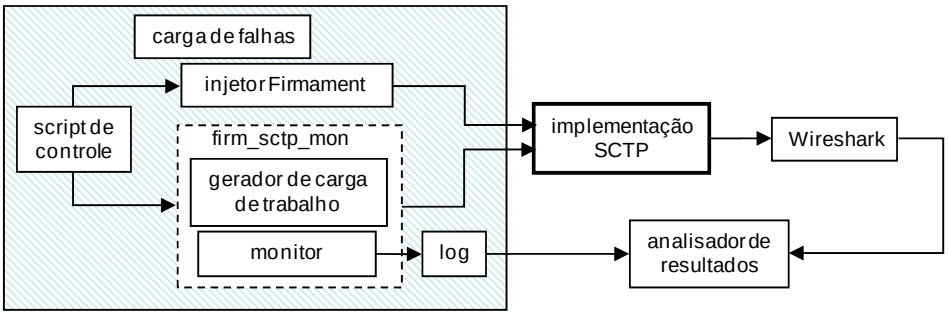

Figura 2 – Ambiente experimental de avaliação do SCTP sob falhas

FIRMAMENT [Firmament 2005], localizado no kernel do Linux, injeta falhas de comunicação em protocolos construídos sobre IP. O injetor associa ao processamento dos pacotes a execução de *scripts* de falhas, chamados *faultlets*, responsáveis por selecionar e atuar sobre as mensagens. O injetor é implementado como módulo do kernel e usa ganchos da interface NetFilter [Russel and Welte 2002] para acessar o fluxo de execução dos protocolos IPv4 e IPv6.

Para usar o injetor, é necessário programar um *faultlet* para cada experimento. Um *faultlet* representa a carga de falhas para um experimento. O *faultlet* é executado sobre cada pacote que cruza um dos fluxos de comunicação, podendo atrasá-lo, alterar seu conteúdo, duplicá-lo, descartá-lo ou aceitá-lo. Além das ações sobre pacotes, um *faultlet* opera sobre variáveis de estado do fluxo, que podem ser usadas para alterar o valor dos dados do pacote. Pode-se também executar operações lógicas e aritméticas, gerando uma maior flexibilidade de descrição de cenários de falhas. São 31 as instruções disponíveis para selecionar e atuar sobre pacotes. O injetor possui também uma opção de cão-de-guarda para evitar que um *faultlet* entre em laço. Ele dispõe ainda de um montador, que verifica o *faultlet* e, se estiver correto, gera uma saída binária. A verificação ajuda a evitar a carga de *faultlets* mal descritos no injetor.

A função de gerador de carga de trabalho e de monitor é realizada por um programa, firm-sctp-mon (figura 2). Este é um programa cliente-servidor que desenvolvemos a partir do sctp_darn, que acompanha lksctp-tools. O mesmo programa atua como cliente, gerando a carga de trabalho, e como servidor, recebendo as mensagens e agindo como monitor dos experimentos, dependendo de como tenha sido

configurado na linha de comando. O firm-sctp-mon é parametrizável em relação às necessidades dos experimentos (tamanho de mensagens, número de mensagens por mensagem de aplicação, localização do arquivo de registro, etc).

No lado cliente, firm-sctp-mon faz o envio, em modo rajada, de 100 ou 500 mensagens com 512 bytes cada uma. Há um pequeno tempo de processamento entre as rajadas. No lado servidor, firm-sctp-mon funciona como monitor controlando se todas as mensagens foram recebidas, se o ordenamento foi mantido, se o protocolo da aplicação atinge somente estados consistentes, e calculando o tempo de recebimento das mensagens. A marcação de tempo é enviada para um arquivo de registro (log na figura 2). A medição de tempo é feita no servidor, pois se constatou que, quando feita no cliente, ela era mascarada pelo efeito dos buffers de transmissão; a aplicação cliente entregava ao protocolo as mensagens e encerrava, antes do servidor recebê-las e encerrar também.

Para análise dos resultados é também necessário registrar os eventos realizados pelo injetor. FIRMAMENT emprega recursos do kernel como o mecanismo de registro klogd, que lê buffers de dados do kernel e os repassa ao utilitário syslogd. Esse recurso, entretanto, não se mostrou adequado, apresentando alta intrusividade. Este problema indica a necessidade de uma nova versão do injetor com um mecanismo de registro mais eficiente. Para esses experimentos, a solução foi usar os dados fornecidos do firm-sctp-mon e um monitor externo (o Wireshark).

O analisador de protocolos Wireshark foi usado para monitorar a correta operação do SCTP, da carga de trabalho gerada e das falhas injetadas. O uso de dois monitores é necessário, pois firm_sctp_mon atua apenas no nível da aplicação (mensagens e *streams* de dados), enquanto que o Wireshark atua nas camadas inferiores da pilha de protocolos (tráfego de pacotes SCTP).

Finalmente, o número de mensagens, seu tamanho e o número de repetições ou rodadas são controlados por um script, que inicia o programa gerador de carga de trabalho. O script também é responsável por inicializar e terminar o servidor, carregar e disparar as rodadas no cliente. Além disso, o script deve descarregar os *faultlets*, acionar e parar o injetor.

5 Condução dos experimentos

Para avaliar as métricas de cobertura de falhas de “perda de mensagens de dados” e de “queda de enlaces”, foram injetados, de forma seletiva, descartes de pacotes SCTP.

5.1 Perda de mensagens

A perda de mensagens de dados foi implementada através do descarte de pacotes de dados enviados do cliente para o servidor. Foram efetuados quatro conjuntos de medições: um conjunto de referência, onde não foram injetadas falhas, e três outros conjuntos com 2%, 5% e 10% de descarte de pacotes. Cada conjunto de medições é composto por 200 repetições de 100 mensagens de dados cada um.

Na figura 3 está reproduzido o *faultlet* usado para efetuar o descarte de pacotes. Na parte inicial, é verificado se o datagrama IP está transportando um pacote SCTP: o byte “protocolo” (byte 9) do datagrama IP terá um valor 132. Se não for SCTP, o pacote será aceito (comando ACP). Caso contrário, será gerado um número pseudo-aleatório

entre -2000 e +18000 e, se o número for negativo (ou seja, 2000 em 20000, 10%), o pacote será descartado (comando DRP).

```
            SET     9       R0      ; Verifica se é pacote do protocolo SCTP
            READB   R0      R1      ; (verifica se o byte 9 do pacote vale 132)
            SET     132     R0
            SUB     R0      R1
            JMPZ    R1      SCTP
            ACP                     ; Aceita pacotes que não são SCTP
SCTP:
            SET     10000   R0      ; Gera um número pseudo-aleatório
            RND     R0      R1      ; Valor entre -10000 e +10000
            SET     8000    R0
            ADD     R0      R1      ; Desloca o valor para -2000 e +18000
            JMPN    R1      DESCARTE
            ACP                     ; Aceita o pacote se valor positivo (90%)
DESCARTE:
            DRP                     ; Descarta o pacote se valor negativo (10%)
```

Figura 3: *Faultlet* de descarte de pacotes

5.2 Queda de enlace

A injeção de falhas de queda de enlaces foi realizada através do descarte de pacotes de entrada e saída do enlace alvo da injeção, no cliente. Dessa forma, o enlace será colocado no estado "operacional" ou "não operacional", quando nenhuma mensagem pode ser transmitida. No caso do enlace estar não operacional, espera-se que o SCTP efetue a comutação para um dos enlaces auxiliares.

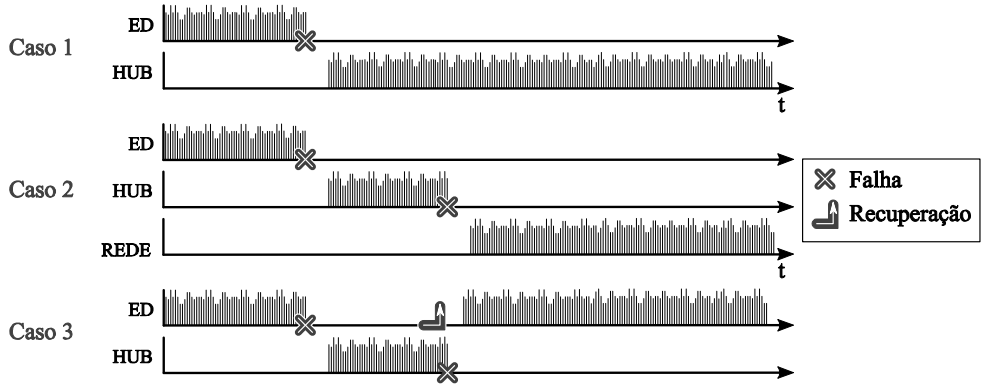

Figura 4: Cenários de falhas de queda de enlace

Para verificar a operação de *failover*, foram estabelecidas três rotas: uma rota principal e duas auxiliares. O lado cliente da aplicação firm_sctp_mon repete 200 vezes o envio de 500 mensagens de dados, para cada caso de injeção de falhas. Os cenários de injeção de falhas estão representados na figura 4, onde **ED** representa o enlace direto; **HUB** representa a rota através de um equipamento do tipo HUB; e **REDE** representa a rota através da rede do laboratório. São três os cenários: no caso 1, ocorre uma queda simples do enlace principal; no caso 2, ocorre a queda do enlace principal seguida da queda do enlace auxiliar HUB (forçando a comutação para o segundo enlace auxiliar); e, no caso 3, ocorre a queda do enlace principal seguido de sua recuperação.

Para injetar as falhas de enlace, foram escritos dois *faultlets*: um que atua sobre os pacotes que chegam ao cliente e o outro que atua sobre os que saem do cliente. Por uma questão de espaço, os *faultlets* são omitidos.

6 Análise dos resultados

6.1 Perda de mensagens

As medidas correspondentes ao tempo gasto para transferir um conjunto de 100 mensagens de aplicação estão apresentadas nas ordenadas dos gráficos onde o eixo das abscissas indica o número da amostra, conforme obtida ao longo do tempo. Nesses gráficos, as medidas estão indicadas por “Ref”, o conjunto de medidas de referência onde não são injetados descartes de pacotes, e por “2%”, “5%” e “10%”, os conjuntos de dados obtidos com descartes de 2%, 5% e 10% das mensagens, respectivamente. Para todos os experimentos com descartes a cobertura de falhas foi de 100%.

Na figura 5 estão apresentados todos os dados coletados, em uma visão geral, na qual se identificam grupos de medidas. Dois grupos salientam-se: um grupo próximo a zero (Grupo 0) e outro próximo a 1 segundo (Grupo 1). Além dessas, foram obtidas medidas próximas aos valores inteiros de tempos de 2 segundos até 8 segundos.

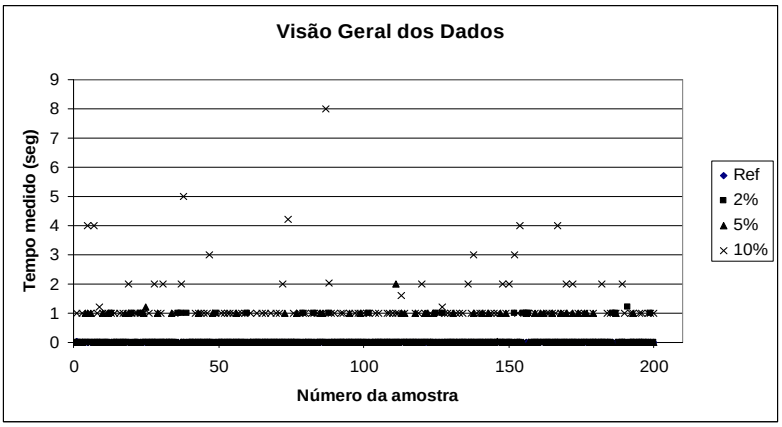

Figura 5: Visão geral das medidas

Para analisar por que as medidas “saltam” de um grupo para outro, foi necessário usar o analisador de protocolos Wireshark e comparar essas observações com a especificação. Em termos gerais, esses saltos correspondem à temporização do transmissor para detectar mensagens perdidas. O SCTP possui dois mecanismos de repetição de mensagens. Um deles, chamado *Fast Retransmit*, repete os pacotes que foram sinalizados, por três vezes, como não entregues no receptor. Esse mecanismo é bastante rápido, pois utiliza, apenas, *chunks* de confirmação (SACK – Selective Acknowledge) para a sinalização. Entretanto, dependendo como ocorreram as perdas de mensagens, esse mecanismo pode não sinalizar adequadamente. Assim, um segundo mecanismo é utilizado. Trata-se de um mecanismo que temporiza a recepção de *chunks* de confirmação dos *chunks* de dados. Apesar de mais lento, o mecanismo não apresenta os problemas do *Fast Retransmit*. A RFC indica um tempo de 1 segundo como o mínimo para essa temporização. Assim, quando a quantidade de mensagens perdidas cresce, a probabilidade de que o mecanismo de temporização entre em ação aumenta.

Quando ocorrem poucos descartes, o tempo total das 100 mensagens aproxima-se muito do tempo gasto quando não existem descartes. O único mecanismo que atua é o *Fast Retransmit*. Quando os descartes aumentam, aumenta a probabilidade de que o mecanismo de temporização atue. Como consequência, a repetição de pacotes ocorrerá em, aproximadamente, 1 segundo após a transmissão das 100 mensagens (notar que a

temporização de 1 s é muito maior do que 0,005 s, que é o tempo gasto para transmitir as 100 mensagens). Finalmente, com o aumento das repetições de pacotes, o mecanismo de temporização atua mais vezes dentro da rajada de 100 mensagens, o que aumenta a probabilidade que ocorram várias temporizações de 1 segundo, podendo levar o tempo para 2, 3 mais segundos.

Dessa forma, as medidas em torno de zero correspondem às situações onde todas as mensagens descartadas foram recuperadas através do *Fast Retransmit*. As outras medidas, em torno dos múltiplos de 1 segundo, são consequências do mecanismo de temporização de detecção de mensagens perdidas.

Medidas de Referência

As estatísticas das medidas de referência apresentadas na tabela 1 e obtidas quando não ocorrem descartes de pacotes, mostram que a média é representativa com um índice de confiança de 95%.

Tabela 1: Medidas de tempo sem descarte de pacotes

Tempo médio medido (s)	Desvio padrão (s)	Intervalo de Confiança (s)	
0,006175	0,001957	0,005904	0,006447

Medidas da atuação do Fast Retransmit

Quando são descartados pacotes, os mecanismos de repetição são ativados.

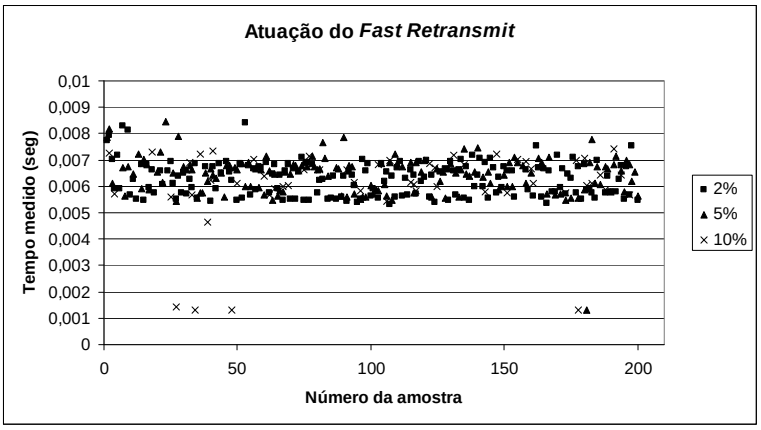

Figura 6: Medidas da atuação do mecanismo *Fast Retransmit*

Entretanto, em algumas rajadas de mensagens atua apenas o *Fast Retransmit*. Nesses casos, o tempo medido está em torno das medidas de referência, conforme pode ser visto no gráfico da figura 6. Além disso, percebe-se que as medidas são relativamente independentes da taxa de descarte, pois todas estão em torno dos mesmos valores.

As estatísticas correspondentes aos tempos da figura 6 estão listadas na tabela 2. A análise dessas estatísticas confirma a representatividade da média calculada com índice de confiança de 95%. Adicionalmente, na tabela 2 estão registrados os percentuais efetivos de descarte, medidos nos experimentos.

Tabela 2: Tempo médio de atuação do mecanismo *Fast Retransmit*

Descarte (especificado)	Descarte (medido)	Tempo médio (s)	Desvio padrão (s)	Intervalo de Confiança (s)	
2%	1,97%	0,006290	0,000635	0,006196	0,006384
5%	5,04%	0,006430	0,000740	0,006312	0,006547
10%	10,25%	0,006050	0,001436	0,005677	0,006423

Medidas de atuação da repetição por temporização

O conjunto de medidas em torno de 1 segundo pode ser separado em dois subconjuntos, conforme aparece na figura 7.

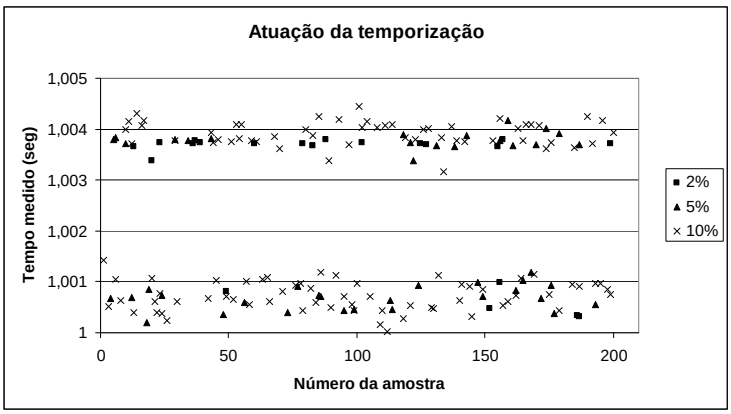

Figura 7: Medidas da atuação do mecanismo de temporização

Na figura 7, o subconjunto em torno de 1,0007 s (grupo inferior) corresponde ao caso onde a temporização terminou após terem sido enviadas todas as mensagens da rajada. Por outro lado, o subconjunto em torno de 1,0038 s (grupo superior) corresponde às rajadas em que os últimos pacotes tiveram que esperar pela repetição de um pacote intermediário, acrescentando 1 segundo ao tempo de transferência da rajada sem repetições por temporização.

Nota-se que a separação em dois subconjuntos, que ocorre em torno de 1 segundo, também ocorre em torno dos outros valores inteiros de tempo. As estatísticas dos dois conjuntos estão apresentadas na tabela 3, onde é usado um coeficiente de confiança de 95%.

Tabela 3: Medidas de tempo com descarte de pacotes

Grupo	Descarte	Tempo médio (s)	Desvio padrão (s)	Intervalo de Confiança (s)	
Inferior	2%	1,000585	0,000299	1,000324	1,000847
	5%	1,000682	0,000243	1,000587	1,000778
	10%	1,000721	0,000288	1,00649	1,000794
Superior	2%	1,003699	0,000097	1,003652	1,003746
	5%	1,003783	0,000163	1,003710	1,003856
	10%	1,003928	0,000238	1,003859	1,003987

6.2 Colapso de enlace

Os cenários de teste foram projetados para medir a cobertura de falhas e os tempos de detecção de colapso de enlace e sua comutação. A cobertura de falhas foi de 100%, uma vez que a aplicação não foi afetada pelas comutações de enlace.

As medidas de desempenho mostraram que os tempos gastos pelo protocolo nas operações de detecção e recuperação do colapso são muito pequenos, não sendo

significativos em um cenário com rajadas de 500 mensagens, período em que foram injetadas as falhas. As estatísticas das medidas obtidas estão apresentadas na tabela 4, onde foi utilizado um coeficiente de confiança de 95%. Nessa tabela aparecem quatro cenários (descritos na figura 4).

Tabela 4: Tempo para transferência de uma rajada com 500 mensagens

Cenários de colapso	Tempo médio (s)	Desvio padrão (s)	Intervalo de Confiança (s)	
Sem colapso	0,024049	0,000038	0,024043	0,024054
No enlace direto	0,024176	0,000431	0,024116	0,024235
No enlace direto e no HUB	0,024060	0,000060	0,024051	0,024058
Colapso e recuperação do enlace direto	0,024045	0,000070	0,024036	0,024055

7 Trabalhos relacionados

Um problema na avaliação baseada em injeção de falhas é a falta de ferramentas adequadas. Apesar dos vários injetores reportados [Aidemark et al. 2001, Stott et al. 2002, Chandra et al. 2004, Hoarau and Tixeuil 2007], ferramentas funcionais e com usabilidade adequada são difíceis de serem encontradas. Como consequência, avaliadores acabam construindo suas próprias ferramentas consumindo tempo que poderia estar sendo melhor aplicado em outras tarefas.

A maioria das ferramentas citadas apresenta problemas de portabilidade que vão desde terem sido desenvolvidas para ambientes muito específicos, a exigirem sistema operacional ou bibliotecas específicas. Esse problema também afeta FIRMAMENT, dependente do kernel do Linux. Grande parte dos injetores não tem código disponível para uso ou não é mais mantido por seus desenvolvedores. Orchestra [Dawson et al. 1996] é uma ferramenta muito referenciada e cujas funcionalidades inspiraram FIRMAMENT, mas não é mais encontrada. O grupo que desenvolve FIRMAMENT tem desenvolvido outras ferramentas de injeção de falhas [Jacques-Silva et al. 2006, Vacaro and Weber 2006] que atuam sobre a máquina virtual Java usando recursos como por exemplo depuração, reflexão computacional e orientação a aspectos. Apesar de serem também ferramentas de injeção de falhas de comunicação, são voltadas apenas para avaliação de aplicações Java e não para avaliação de protocolos.

Na literatura, são encontrados ainda alguns trabalhos que tratam da avaliação das estratégias de tolerância a falhas do SCTP. Um trabalho recente [Hurtig and Brunstrom 2008] avalia, sob perda de pacotes, duas versões do algoritmo *early retransmit*, que é uma melhoria do mecanismo de recuperação de perdas de SCTP. Os testes são realizados para diferentes cenários de tráfego. A métrica usada para avaliar as versões do algoritmo é o tempo de transferência de mensagem, definido como o tempo necessário para um mensagem trafegar da aplicação emissora à aplicação receptora. Outro artigo [Grinnemo and Brunstrom 2005] analisa o desempenho do SCTP em cenários de *failover* com diferentes cargas de trabalho. Ainda outro trabalho [Qiao et al. 2007] analisa, usando simulação, o efeito de atrasos em caminhos no desempenho do SCTP. Esses trabalhos são complementares ao relatado nesse artigo e, como avaliam diferentes estratégias, os resultados não são comparáveis.

8 Conclusões

Apresentou-se uma avaliação de SCTP sob falhas aplicando um injetor de falhas de comunicação que vem sendo desenvolvido no grupo de pesquisa. Protocolos de

comunicação são artefatos de software sofisticados e complexos e é esperado que operem corretamente mesmo com perda de mensagens. Um injetor de falhas permite isolar e validar características de tolerância a falhas destes sistemas. Neste artigo mostrou-se que injeção de falhas é uma técnica de validação viável que permite calcular métricas relacionadas à confiabilidade de protocolos em cenários de falhas facilmente especificados.

Os experimentos realizados mostraram uma cobertura de falhas de 100% para as falhas injetadas, o que comprova a adequação da lkscpt à especificação do protocolo. Os experimentos mostraram também que o protocolo apresenta ligeira queda de desempenho sob falhas. O protocolo pode ser considerado uma excelente opção para aplicações que exigem maior confiabilidade, mas que não podem sofrer quedas acentuadas de desempenho quando operam sob falhas.

A avaliação do grau de intrusividade das ferramentas usadas na avaliação de um sistema é fator chave para o sucesso da análise. Uma análise dos resultados obtidos das medições mostra que estão de acordo com o esperado do protocolo, o que leva a conclusão de que o grau de intrusividade das ferramentas usadas, injetor de falhas, analisador de protocolo e monitor firm_sctp_mon, é baixo.

Identificou-se que certas análises só podem ser efetuadas se o analista tiver um conhecimento detalhado do protocolo. Além disso, a análise requer o uso de um monitor externo (como o Wireshark, usado nos experimentos de recuperação de mensagens perdidas), capaz de monitorar os pacotes que trafegam na rede, de maneira a identificar comportamentos próprios do protocolo.

O estudo do comportamento do protocolo sob falhas verificando sua operação em condições reais seria inviável devido à baixa frequência das falhas e a falta de controle sobre a sua ocorrência. A injeção de falhas permite o controle da ocorrência das mesmas, além de sua freqüência de ativação, o que possibilita um estudo rápido e abrangente do comportamento do sistema.

Cada avaliação de protocolo conduzida permite refinar FIRMAMENT. Os próximos passos prevêem investir em ferramentas mais adequadas de controle e monitoramento, pois o monitoramento provido pela ferramenta se mostrou demasiado intrusivo.

Referências bibliográficas

Aidemark, J., Vinter, J., Folkesson, P., and Karlsson, J. (2001). GOOFI: generic object-oriented fault injection tool. In *Proc. of Int. Conf. on Dependable Systems and Networks, DSN*, pages 83–88, IEEE.

Arlat, J., Aguera, M., Amat, L., Crouzet, Y., Fabre, J.-C., Laprie, J.-C., Martins, E., and Powell, D. (1990). Fault-injection for dependability validation: a methodology and some applications. *IEEE Trans. on Software Engineering*, 16(2):166–182.

Braden, R. (1989). Requirements for internet hosts – Communication Layers: RFC 1122.

Cechin, S., Nedel, W., and Weber, T. (2008). Teste por injeção de falhas da implementação do protocolo de comunicação SCTP. In *Anais do IX Workshop de Testes e Tolerância a Falhas, WTF*, pages. 57–70, SBC.

Chandra, R., Lefever, R. M., Joshi, K. R., Cukier, M., and Sanders, W. H. (2004). A Global-State-Triggered Fault Injector for Distributed System Evaluation. *IEEE Trans. on Parallel and Distributed Systems*, 15(7):593–605.

Cirne, W., Brasileiro, F., Andrade, N., Costa, L., Andrade, A., Novaes, R., and Mowbray, M. (2006). Labs of the world, unite!!! *Journal of Grid Computing*, 4(3):225–246.

Combs, G. et al. (2006) Wireshark. Disponível em <http://www.wireshark.org>. Acesso em: dez. 2008

Dawson, S, Jahanian, F., and Mitton, T. (1996). ORCHESTRA: A probing and fault injection environment for testing protocol implementations. In *Proc. of the Int. Computer Performance and Dependability Symposium, IPDS*, page 56, IEEE.

Drebes, R. J. (2005). FIRMAMENT: Um Módulo de Injeção de Falhas de Comunicação para Linux. Dissertação (Mestrado em Ciência da Computação) – UFRGS.

Firmament (2005). Código fonte e executável do injetor de falhas. Disponível em <http://sourceforge.net/projects/firmament>. Acesso em 8/04/2009.

Grinnemo, K-J., and Brunstrom, A. (2005). Impact of traffic load on SCTP failovers in SIGTRAN. In *Proc. of the 4th Int. Conf. on Networking*, LNCS 3420, pp 774–783.

Hoarau, W., Tixeuil, S., and Vauchelles, F. (2007). FAIL-FCI: versatile fault injection. *Future Generation Computer Systems*, 23(7):913–919.

Hsueh, M.-C., Tsai, T. K., and Iyer, R. K. (1997). Fault injection techniques and tools. *Computer*, 30(4):75–82.

Hurtig, P., and Brunstrom, A. (2008). Enhancing SCTP loss recovery: an experimental evaluation of early retransmit. *Computer Communications*, 31(16):3778–3788.

Jacques-Silva, G., Drebes, R. J., Trindade, J., Weber, T. S., and Jansch-Pôrto, I. (2006). A network-level distributed fault injector for experimental validation of dependable distributed systems. In *Proc. of the 30th Int. Computer Software and Applications Conference, COMPSAC*, v. 1, pages 421–428.

Jain, R. (1991). The Art of Computer Systems Performance Analysis: techniques for experimental design, measurement, simulation and modeling. John Wiley & Sons, 1st ed.

Kanoun, K., and Spainhower, L. (2008). Dependability Benchmarking for Computer Systems. Wiley-IEEE Computer Society.

Menegotto, C. C., Vacaro, J. C., and Weber, T. S. (2007). Injeção de Falhas de Comunicação em Grids com Características de Tolerância a Falhas, in *Anais do VIII Workshop de Testes e Tolerância a Falhas*, WTF, pages 71–84, SBC.

Qiao, Y. et al. (2007). SCTP Performance Issue on Path Delay Differential. In *Proc. of the 5th Int. Conference on Wired/Wireless Internet Communications, WWIC*, LNCS 4517, pp 43–54.

Russel, R. and Welte, H. (2002). Linux net filter hacking HOWTO. Disponível em: <http://www.netfilter.org/documentation/>. Acesso em: dez. 2008.

Stewart, R. (2007) Stream control transmission protocol: RFC 4960.

Stewart, R. et al. (2000). Stream control transmission protocol: RFC 2960.

Stott, D, Jones, P. H, Hamman M, Kalbarczyk, Z., and Iyer, R.K. (2002). NFTAPE: networked fault tolerance and performance evaluator. In *Proc. of Int. Conf. on Dependable Systems and Networks, DSN*, page 542, IEEE.

Vacaro, J. C. and Weber, T. S. (2006). Injeção de falhas na fase de teste de aplicações distribuídas. In *Anais do Simp. Bras. de Eng. de Software, SBES*, pp 161–176, SBC.

Yarroll, L.M. and Knutson, K. (2001). Linux kernel SCTP: The third transport. Disponível em <http://lksctp.sourceforge.net/>. Acesso em: dez. 2008.