Teste por injeção de falhas da implementação do protocolo de comunicação SCTP

Sergio Cechin[1], Werner Nedel[1], Taisy Weber[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil
{cechin,wmnedel,taisy}@inf.ufrgs.br

Abstract*: The advantages of using a fault injector to test the implementation of communication protocols are discussed. SCTP is a new protocol over IP that offers services to network applications that are similar to UDP and TCP. SCTP promises to these applications high reliability guarantees. Injecting communication faults into SCTP, we show the feasibility of the evaluation of protocol behavior under faults. FIRMAMENT is the injector chosen for the test experiments because its availability in our laboratories, but mainly on account of its ability to operate directly along the kernel protocol stack and the easiness to be configured to handle several communication protocols.*

Resumo*. São apresentadas as vantagens do uso de um injetor de falhas de comunicação para o teste da implementação de protocolos de rede. SCTP é um protocolo novo, que oferece serviços semelhantes aos do UDP e do TCP e promete garantir confiabilidade às aplicações de rede. Através de experimentos de injeção de falhas sobre o protocolo SCTP mostra-se como é possível avaliar o comportamento do protocolo na presença de falhas controladas. O injetor FIRMAMENT foi escolhido para os experimentos de teste por estar disponível para uso, por possuir um alto poder de expressão de carga de falhas, por operar diretamente na pilha de protocolos do sistema operacional e pela facilidade com que pode ser configurado para operar com diferentes protocolos de rede.*

1 Teste sob falhas da implementação de protocolos de comunicação

Qualquer artefato de software que deva atender requisitos mínimos de confiabilidade e disponibilidade deve ser desenvolvido prevendo a ocorrência de falhas e o tratamento ou sinalização dos erros por elas ocasionados. Os procedimentos implementados para a detecção e tratamento de erros devem ser testados em um ambiente sujeito a falhas. Para não depender da ocorrência natural de falhas, cuja freqüência pode ser muito baixa e a controlabilidade nula, uma técnica adequada é emular falhas por software e aplicá-las à implementação sob teste.

Protocolos de comunicação são desenvolvidos em software seguindo uma dada especificação. Por atuarem nos níveis mais baixos de abstração, uma implementação incorreta destes protocolos apresenta um enorme potencial de interferir negativamente no comportamento dos níveis superiores, prejudicando as aplicações que dela dependam e, portanto, qualquer serviço de rede fornecido por software.

A execução de um protocolo de comunicação é tipicamente determinada pelo estado de cada participante e pelas mensagens que os participantes trocam entre si. É muitas vezes impossível ou inconveniente alterar diretamente o estado do protocolo, mas para levar o protocolo a um determinado comportamento observável é possível atuar diretamente sobre as mensagens. Por exemplo, em um protocolo especificado para tolerar perdas de mensagens ou colapso de um fluxo, é possível testar seu comportamento a esses tipos de falha simplesmente suprimindo algumas ou todas as mensagens que um participante recebe.

Uma ferramenta atuando sob o protocolo alvo na pilha de protocolos é extremamente útil para o teste. Essa ferramenta pode observar e atuar sobre qualquer mensagem enviada ou recebida. Desta forma o desenvolvedor ou testador do protocolo pode suprimir, alterar valores de campos e atrasar cada uma das mensagens recebidas ou enviadas, controlando o momento preciso da ocorrência de uma falha, e assim verificar o comportamento da sua implementação em situações de maior risco. Para o testador ou certificador da implementação, uma ferramenta na pilha de protocolos permite também emular parâmetros como freqüência de omissão de mensagens e atrasos crescentes por sobrecarga de rede, úteis para obter medidas estatísticas sobre o comportamento do protocolo sob determinada taxa de falhas de comunicação.

SCTP é um protocolo relativamente novo, descrito inicialmente na RFC2960 [Stewart 2000] e consolidado na RFC4960 [Stewart 2006], originalmente desenvolvido para transmitir mensagens de sinalização da Rede de Telefonia Pública Comutada sobre IP. SCTP fornece associações entre clientes e servidores, podendo oferecer múltiplos fluxos entre pontos finais de conexão, cada um com sua própria entrega confiável de mensagens. O protocolo SCTP promete conferir alta confiabilidade às aplicações de rede.

Neste artigo relatamos um experimento de injeção de falhas sobre o SCTP conduzido com o objetivo de ilustrar as vantagens do uso de injetores de falhas no nível da pilha de protocolos para o teste da implementação de protocolos de comunicação. É mostrado como é possível avaliar o comportamento do SCTP em presença de falhas controladas. Entre os injetores de falhas mencionados na literatura para experimentação com protocolos de comunicação, FIRMAMENT foi escolhido principalmente por ter sido desenvolvido e estar disponível nos laboratórios onde o trabalho foi conduzido, mas também por possuir um alto poder de expressão de carga de falhas, por operar diretamente na pilha de protocolos do sistema operacional e por permitir ser facilmente configurado para trabalhar com diferentes protocolos de rede.

Nas próximas seções serão apresentados os conceitos básicos de condução de testes de injeção de falhas, a ferramenta usada, o protocolo alvo SCTP e os experimentos conduzidos sobre o SCTP.

2 Injeção de falhas

Um problema é saber se a estratégia empregada para detectar falhas e corrigir os erros provocados por essas falhas resulta realmente em aumento de confiabilidade. Como na maior parte das redes de comunicação as taxas de falhas são relativamente baixas e as falhas acontecem de forma aleatória e incontrolável, o problema consiste em avaliar se a estratégia empregada está realmente tolerando as falhas para as quais foi planejada, sem

necessidade de esperar indefinidamente para que as falhas significativas para uma dada situação realmente ocorram durante o teste. Uma solução é a injeção de falhas.

A injeção de falhas é um procedimento de teste das estratégias de tolerância a falhas empregadas. Através da introdução controlada de falhas pode se determinar se a estratégia permite ao sistema tolerar as falhas injetadas e qual o custo em desempenho relacionado à cobertura de falhas alcançada. Na fundamentação teórica, essa abordagem é explorada há tempo suficiente para ser considerada madura ([Arlat 1990], [Hsueh 1997]). Injetores de falhas têm sido construídos e aplicados ([Han 1995], [Carreira 1998], [Chandra 2004], [Locker and Xu 2003], [Martins 2002], [Some 2001], [Stoot 2000]). Raras, entretanto, são as ferramentas realmente disponíveis e, mesmo essas, são específicas para alguns domínios restritos de aplicações. Entre elas, mais raras ainda são os injetores de falhas que permitam emular falhas de rede e assim testar protocolos de comunicação. A complexidade dos sistemas de conexão e comunicação facilita a propagação de falhas e dificulta sua detecção. Sem controle sobre o tipo e origem das falhas torna-se extremamente difícil validar a correção das estratégias de tolerância a falhas nesses sistemas.

Apesar de madura na teoria, a experiência acumulada com teste sob falhas de sistemas baseados em troca de mensagens ainda não é suficiente. A experimentação com alvos reais, como implementações de protocolos de rede, traz como benefícios uma maior familiaridade com a área de testes por injeção de falhas e a possibilidade de avaliar também a própria ferramenta usada visando sugestões para a continuidade de seu desenvolvimento.

3 O protocolo SCTP

O SCTP – *Stream Control Transmission Protocol* – é um protocolo de nível de transporte, confiável, capaz de operar sobre um serviço de pacotes não confiável e sem conexão, como é o caso do IP. A confiabilidade oferecida por este protocolo é alcançada através do uso de mensagens de confirmação (*acknowledge messages*) com eventual retransmissão, em caso de erro na recepção das mensagens. A detecção de erros nas mensagens é obtida com o uso de CRC e a duplicação de mensagens é evitada através de uma numeração seqüencial das mesmas.

O protocolo SCTP está definido na RFC4960 [Stewart 2006], que torna obsoletos os dois documentos de definição anteriores: a RFC2960 [Stewart 2000] e a RFC3309 [Stewart 2004]. Esse protocolo foi originalmente projetado para a transmissão, sobre IP, de mensagens de sinalização da RTPC – Rede de Telefonia Pública Comutada. Entretanto, devido às suas características, outros tipos de aplicação podem obter vantagens em utilizá-lo. De uma forma geral, o protocolo SCTP fornece os seguintes serviços:

- transferência de dados sem erro e sem duplicação de mensagens;
- fragmentação de pacotes para ajustar ao MTU da rota selecionada;
- entrega seqüencial de mensagens, com possibilidade de ordenação por usuário, mesmo que transmitidas em múltiplos fluxos;
- possibilidade de empacotamento de múltiplas mensagens de usuário em um único pacote SCTP;

- suporte a tolerância a falhas pelo uso de múltiplos encaminhamentos para o destino.

3.1 Aplicação do SCTP

Apesar do TCP ser o principal protocolo para a transmissão confiável de dados sobre redes IP, um número crescente de aplicações têm sido desenvolvidas sobre UDP, de maneira a utilizarem mecanismos próprios para prover confiabilidade de transmissão. Dois são os fatores que levam a essa decisão de projeto: a inflexibilidade com que o protocolo TCP efetua o controle de erros, não levando em consideração as peculiaridades das aplicações, e o (baixo) desempenho apresentado, quando se compara uma conexão TCP com uma solução especificamente projetada para a aplicação (usando UDP, por exemplo).

O TCP oferece a transferência confiável e a ordenação estrita na entrega de pacotes. Entretanto, as aplicações podem necessitar a transferência confiável, porém sem a ordenação. Outras, ainda, podem requerer um ordenamento parcial das mensagens sem a necessidade da transferência confiável. Note-se que essas aplicações, se utilizassem o TCP, apresentariam um baixo desempenho relativo, causado por um mecanismo que não lhes é útil. Dessa forma, o SCTP possibilita que a aplicação configure, de forma independente, o uso da confirmação de entrega de mensagens e ordenamento das mesmas, permitindo a escolha da operação de forma semelhante ao do UDP (sem confirmação e sem ordenamento) ou, no outro extremo, com ambas: confirmação e ordenamento total.

Diferentemente do protocolo TCP, o protocolo SCTP é orientado a mensagens ao invés de um fluxo contínuo de bytes. Dessa forma, cada pacote é recebido atomicamente, como um bloco indivisível, exatamente da forma que foi transmitido. Devido a essa característica, o serviço de entrega de pacotes do SCTP pode ser descrito como um serviço de datagramas (pois são entregues na forma como foram transmitidos) com confirmação (devido ao mecanismo de confirmação de entrega das mensagens). Apesar de o SCTP oferecer um serviço de transmissão orientado a mensagens, as aplicações não necessitam fragmentar seus dados de acordo com um tamanho máximo de pacote. O próprio protocolo SCTP encarrega-se da fragmentação e remontagem dos fragmentos no destino.

Note-se que, no caso do TCP, a aplicação não tem a informação de início ou término das mensagens, cabendo às aplicações incorporarem delimitadores de registros aos dados transmitidos, de maneira a possibilitar a sua separação no destino. Além disso, no caso de uso do TCP e quando a aplicação encerra o envio dos dados, esta deve limpar o buffer de transmissão de maneira que todos os bytes sejam transmitidos. Isso não é necessário com o SCTP, uma vez que o protocolo incorpora um mecanismo que permite o envio e a recepção dos pacotes, mesmo após o encerramento da "associação" (forma mais geral de conexão, usada pelo SCTP).

Ainda, o TCP é capaz de associar um único fluxo de dados a uma conexão. Assim, a transmissão de múltiplos fluxos deve ser feita através de múltiplas conexões. De forma diferente, um pacote SCTP é capaz de transportar vários fluxos, uma vez que oferece serviços separados para cada um deles.

3.2 Associações SCTP

Os pacotes SCTP, diferentemente do TCP, não são transmitidos em uma conexão, mas em uma associação. Uma conexão utiliza apenas dois terminais e um caminho entre eles enquanto que uma associação é mais abrangente, pois possibilita que vários fluxos de dados sejam negociados entre os terminais.

Uma associação também permite que os terminais de origem e destino tenham associados vários endereços de transporte (combinação de endereço de rede, tipicamente IP, e de uma porta SCTP), possibilitando que um fluxo possa ser transportado através de várias rotas, o que oferece redundância de encaminhamento como forma de tolerância a falhas. Além disso, cada fluxo de uma associação possui um controle de erros e de ordenação de entrega de mensagens independentes, evitando a interferência entre fluxos. Essa característica é especialmente interessante quando ocorre a perda ou o atraso de mensagens em algum dos fluxos. Um exemplo de interferência, que é solucionado por esse mecanismo, é o bloqueio de transmissão conhecido como *head-of-line*, ao qual o TCP está sujeito.

3.3 Pacote SCTP

As PDU – Protocol Data Unit –, ou pacotes SCTP, são transportados como *payload*s dos datagramas IP. Estes, por sua vez, para permitirem alcançar as funcionalidades citadas anteriormente, devem apresentar uma estrutura hierárquica que permita, por exemplo, encapsular mais de um fluxo de dados em um único pacote.

O pacote SCTP corresponde ao primeiro nível da hierarquia de estruturas de dados e o cabeçalho comum desses pacotes contém informações relativas ao pacote como um todo. Fazem parte desse cabeçalho as portas SCTP de origem e destino, uma identificação da associação (*tag*) e dígitos de verificação para controle de erros de transmissão.

A área de dados dos pacotes SCTP é dividida em *chunks*, cada um deles correspondendo a um fluxo de dados: esses *chunks* formam o segundo nível da hierarquia e são compostos, também, de um cabeçalho e uma área de dados. No cabeçalho estão identificados o tipo de dados contidos no *chunk*, um conjunto de flags de controle e o tamanho da área de dados do *chunk*.

A estrutura da área de dados de cada *chunk* depende de seu tipo. Os *chunks* do tipo DATA (que são usados para transportar dados) carregam a informação do TSN – *Transmission Sequence Number* –, usado para determinar eventuais falhas na recepção de *chunks*, a identificação do fluxo, usada para separar os vários fluxos de dados de uma determinada associação, e o SSN – *Stream Sequence Number* –, usado para ordenar os dados recebidos em um determinado fluxo.

Existem 15 tipos pré-definidos de *chunks* que são usados de maneira a obter a comunicação confiável e com ordenamento de entrega de mensagens. Assim, além dos *chunks* usados para a transferência de dados, existem *chunks* para inicializar e encerrar uma associação, para verificar a conectividade entre transmissor e receptor (mecanismo de *heartbeat*), para informar que uma mensagem foi recebida com erro (*acknowledge*), para sinalização de erros, etc.

3.4 Tolerância a falhas com SCTP

O formato do pacote SCTP permite que vários fluxos sejam encapsulados em um mesmo pacote. Dessa forma, pode-se manter várias conexões entre dois nodos quaisquer, sem que haja interferência entre eles. Além disso, a informação de uma mensagem foi entregue corretamente é enviada na forma de um *chunk* especial de reconhecimento. Além disso, uma vez que cada fluxo possui identificação própria e uma seqüência de numeração independente, a confiabilidade de entrega assim como a ordenação são, efetivamente, realizados no nível de fluxo de informação e não no nível de pacote (tal como acontece no TCP).

O SCTP é capaz de fragmentar e remontar, automaticamente, pacotes maiores do que o MTU da rota utilizada. Também é capaz de receber pacotes por várias rotas e processá-los juntos, discriminando os vários fluxos existentes nos pacotes recebidos. Essas duas funcionalidades são possíveis graças à numeração seqüencial existente no *chunks*. Dessa forma, os dados de um fluxo podem ser enviados através de várias rotas e, ao chegarem ao destino, serão corretamente identificados e remontados para entrega à aplicação. Essa funcionalidade possibilita a diversidade de rotas, o que torna o protocolo robusto diante de falhas de comunicação.

O controle de erros através de mensagens de reconhecimento está sempre ativo no SCTP. Entretanto, a ordenação dos *chunks* pode ser ativada ou desativada, sendo que, quando ativada, a ordenação é parcial, ou seja, os dados são ordenados dentro de cada fluxo de dados. O receptor é informado se deve aplicar o algoritmo de ordenação ou não através de um flag no conjunto de flags do cabeçalho do *chunk*.

4 O ambiente de teste

O ambiente de teste emula situações reais de funcionamento do sistema alvo sobre uma rede Internet injetando falhas através de um injetor e observando o comportamento da aplicação. Esse teste experimental permite determinar medidas como cobertura de falhas e queda de desempenho em caso de falhas, entre outras.

O ambiente usado no teste sob falhas do SCTP é composto pelo injetor de falhas, uma carga de trabalho e uma carga de falhas a ser injetada. A carga de falhas é construída de acordo com o modelo de falhas relacionado ao protocolo alvo e previsto na sua especificação. Cada experimento é um teste executado em condições controladas no qual são introduzidas as falhas descritas na carga de falhas. Os dados coletados durante um experimento são posteriormente analisados e as medidas desejadas calculadas.

A injeção de falhas em um protocolo de comunicação consiste basicamente em interpretar o conteúdo da mensagem, analisar a carga de falhas, selecionar as mensagens de interesse e atuar de alguma maneira sobre a mensagem [Dawson 96]. Essa atuação pode ocorrer de diversas maneiras, como duplicando ou descartando mensagens e/ou modificando do seu conteúdo.

4.1 O injetor de falhas

FIRMAMENT (*Fault Injection Relocatable Module for Advanced Manipulation and Evaluation of Network Transports*) [Drebes 2005], desenvolvido nos laboratórios do Grupo de Tolerância a Falhas da UFRGS, é um injetor localizado no kernel do sistema

operacional Linux, que permite injeção por software de falhas de comunicação no protocolo IP ou qualquer outro protocolo construído sobre IP. FIRMAMENT é a evolução de outro injetor do Grupo, ComFIRM [Drebes 2006], provendo um melhor mecanismo para a descrição de cargas de falhas. A funcionalidade desses injetores foi influenciada por ORCHESTRA [Dawson 96]. Os tipos de falhas injetáveis por FIRMAMENT são: omissão de envio e recebimento de mensagens, colapso de nodo e canal, atraso de mensagens e falhas bizantinas. Duas idéias básicas norteiam o injetor: possibilitar a modificação da configuração e da descrição dos cenários de falhas de um experimento durante sua execução e representar cargas de falhas através de regras simples, que não sobrecarreguem o processamento de mensagens.

As primeiras versões de ComFIRM exigiam a alteração do código do kernel do sistema operacional hospedeiro. Apesar de eficiente, essa estratégia é intrusiva no kernel e compromete a portabilidade do injetor. A sua última versão já adotava o conceito de módulo carregável no kernel. Esse conceito foi aproveitado em FIRMAMENT.

O injetor associa ao processamento dos pacotes de comunicação a execução de *scripts* de processamento eficiente, chamados *faultlets*, responsáveis por selecionar e atuar sobre as mensagens. FIRMAMENT provê portabilidade via módulo do kernel e através do uso de ganchos da interface NetFilter [Russel and Welte 2002] para acessar o fluxo de execução dos protocolos IPv4 e IPv6. NetFilter está disponível a partir da versão 2.4 do Linux. Assim, qualquer versão do kernel do Linux posterior suporta o injetor, sem a necessidade de recompilação do kernel.

4.2 Programação de cargas de falhas

Faultlets, definidos por FIRMAMENT, permitem um alto poder de expressão na descrição de cargas de falhas. Para usar o injetor, o testador programa um *faultlet* específico para cada experimento. Um *faultlet* vai ser tanto mais difícil de ser programado quanto mais complexo for o protocolo sendo testado e a cobertura desejada para o teste. Alguns injetores de falhas como FIONA [Jacques-Silva 2006] e ComFIRM possuem uma séria limitação na descrição de cargas de falhas, pois testes de valores em determinados campos da mensagem são feitos a partir de deslocamentos fixos na mensagem. Esse empecilho dificulta testes como nos cabeçalhos do protocolo IPV4, que possuem tamanho variável. FIRMAMENT não apresenta esse inconveniente.

Um *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 para qualquer protocolo.

Um *faultlet* é interpretado por uma máquina virtual no injetor, que associa o pacote ao *faultlet* e às variáveis de estado. Essa máquina virtual opera sobre IPV4 e IPV6, possibilitando a descrição de cargas de falhas para quatro fluxos de dados. As variáveis de estado da máquina virtual são um conjunto de 16 registradores de uso geral de 32 bits, disponíveis para cada fluxo. Os registradores operam com representação de números inteiros com sinal no formato de rede, sendo que o injetor faz a sua conversão automática para o formato nativo do hardware onde está sendo executado. São 31 as

instruções disponíveis divididas em sete classes: entrada e saída, lógicas e aritméticas, ação sobre o pacote, desvio, manipulação de auto-incremento, manipulação de seqüência de caracteres e propósitos gerais. O injetor possui também uma opção de "cão-de-guarda" para evitar que um *faultlet* entre em laço eterno.

FIRMAMENT não interpreta diretamente a forma textual de um *faultlet*. A ferramenta dispõe de um montador, que verifica o tipo dos parâmetros das instruções e, se estiverem corretos, gera uma saída binária. A verificação ajuda a evitar o carregamento de *faultlets* mal descritos no injetor.

4.3 Arquivos de controle e configuração do injetor

Por ser um módulo do kernel, o injetor FIRMAMENT não dispõe de uma interface de alto nível para sua operação. A ferramenta é controlada através da interface de arquivos virtuais do sistema operacional, no diretório /proc/net/firmament. Encontram-se, neste diretório, os arquivos de regras, que permitem a leitura e escrita dos *faultlets* já em formato binário, para cada fluxo de pacotes, e o arquivo de controle, para o qual são passados os comandos de controle do injetor. Esses comandos permitem iniciar e parar o processamento do *faultlet*, mostrar registradores da máquina virtual do injetor e reiniciar a ferramenta, parando todos os fluxos e eliminando todos os *faultlets* configurados.

Para análise dos resultados de um experimento de injeção de falhas é necessário registrar os eventos realizados pelo injetor, o que permite relacionar os resultados com as alterações de estado do protocolo sob teste e a determinação da cobertura de falhas de seus mecanismos de tolerância a falhas. No FIRMAMENT, são empregados recursos do sistema como o mecanismo de captura de mensagens klogd, que lê buffers de mensagens do kernel as repassa ao utilitário syslogd.

5 Experimentos de injeção de falhas

Os testes envolvendo a injeção de falhas no protocolo SCTP foram aplicados de maneira a exercitar os mecanismos de tolerância a falhas oferecidos pelo protocolo. Foram exercitadas as funcionalidades relacionadas com o controle de fluxo, verificação de associação, entrega seqüencial e controle de duplicação de mensagens.

Para cada experimento foi desenvolvido um *faultlet*, a ser usado na injeção de falhas. Foram escritos *faultlets* para obter os seguintes cenários de falha:

- descarte de pacotes.
- duplicação de pacotes;
- alteração dos *tags* de verificação.

Devido à limitação de espaço só mostraremos o *fautlet* escrito para o primeiro cenário.

5.1 Configuração de teste

Os experimentos foram realizados em máquinas idênticas com processador AMD Athlon XP 2000+, 512Mbytes de memória principal e controladores de rede modelo VIA Rhine II padrão IEEE 802.3u (Ethernet, 100 Mbit/s). Todas as máquinas utilizam o

kernel 2.6.20.3 do Linux, sendo que o FIRMAMENT foi carregado em apenas uma delas.

Para que fosse suportado o SCTP, foi necessário instalar a biblioteca lksctp (disponível em: < http://lksctp.sourceforge.net/>). Essa biblioteca permite o acesso dos usuários aos mecanismos do protocolo, que devem estar habilitados e compilados no kernel do Linux. O suporte ao SCTP foi incluído ao kernel como um módulo. Acompanha o código da biblioteca alguns programas de teste, que foram utilizados na geração e monitoramento do tráfego SCTP, sobre o qual as falhas foram injetadas.

5.2 Descarte de pacotes

Conforme apresentado anteriormente, o protocolo SCTP é capaz de detectar a perda de pacotes e reenviá-los de maneira a recuperar essa falha. Esse procedimento, entretanto, impõe uma queda de desempenho devido ao tempo que o cliente deve esperar até decidir por reenviar o pacote. Assim, para verificar o impacto desse mecanismo no desempenho da aplicação, escreveu-se um *faultlet* (figura 1) que descarta pacotes em percentuais pré-definidos de 10%, 20% e 40%. Esses percentuais foram aplicados a pacotes que continham um fluxo de dados, dois fluxos de dados e oito fluxos de dados.

```
; localizar o protocolo de transporte no cabeçalho IP
       SET    9     R0          ; deslocamento no pacote capturado
       READB  R0    R1
       SET    132   R0          ; seleciona os pacotes SCTP
       SUB    R0    R1
       JMPZ   R1    ACPSCTP     ; se o valor lido do cabeçalho for 132
                                ; desvia para aceitar ou descartar SCTP
       ACP                      ; aceita qualquer outro pacote
ACPSCTP:
       SET    100   R0          ; inicializa R0 para gerar número aleatório
       RND    R0    R1          ; gera valor aleat. entre -100 e 100 em R1
       SET    20    R0          ; define percentagem de descarte
       ADD    R0    R1          ;
       JMPN   R1    DESCARTE    ; se negativo então descarta pacote
       ACP                      ; se positivo então aceita pacote
DESCARTE:
       DRP
```

Figura 1 – Faultlet para descarte aleatório de pacotes SCTP

A figura 1 mostra o faultlet que permite testar se cada pacote capturado pertence ao protocolo SCTP, deixando passar pacotes de todos os demais protocolos. Quando encontra um pacote SCTP, o faultlet determina de acordo com uma taxa uniforme de distribuição de probabilidade se o pacote deve ser aceito ou descartado. Para cada percentual de descarte desejado deve ser escrito um novo faultlet.

Os vários *faultlets* e número de fluxos de dados foram combinados em nove configurações distintas. A estas, foram acrescentadas três configurações de referência, onde não foram injetadas falhas, resultando em um total de doze configurações.

Para avaliar o desempenho de cada configuração, foram efetuadas medições do tempo de processamento para enviar um total de 100 mensagens com 1500 bytes cada uma. Foram efetuadas 200 medições para cada configuração, para um índice de confiança de 95%. Na tabela 1 estão listados os tempos medidos nas configurações de referência (sem descartes), com um, dois e oito fluxos. A análise desses resultados mostra que o número de fluxos não afeta significativamente os tempos. A razão disso é

que, na maioria das transmissões, os vários fluxos estão sendo encapsulados no mesmo pacote. Para que obtivéssemos um ganho com o aumento do número de fluxos, seria necessário que houvesse vários rotas por onde os fluxos pudessem ser transportados em paralelo.

Tabela 1: tempo de processamento das configurações sem injeção de falhas

Número de fluxos	Tempo (seg)
1	2,805
2	2,786
8	2,923

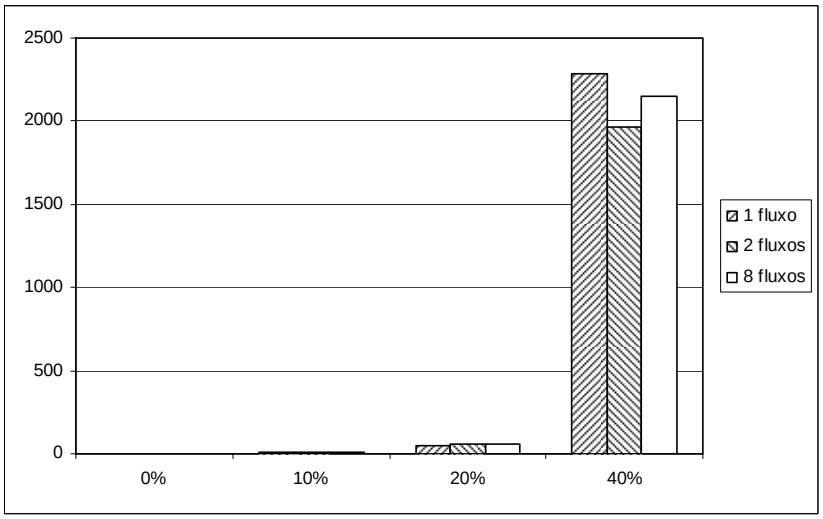
Na tabela 2 estão listadas as medidas efetuadas nas configurações onde foram aplicados os *faultlets* que descartam pacotes. Da mesma forma que no caso sem descarte, a análise dessas medidas mostra que os tempos permanecem, aproximadamente, os mesmos, independentemente do número de fluxos de dados usados para transportar os dados. O fato de poder transportar dados em fluxos independentes é apresentado como vantagem do protocolo SCTP. Essa característica deveria levar a um desempenho tanto maior quanto mais fluxos fossem usados. Entretanto, as medidas apresentadas na tabela 2 não confirmaram esse fato. Uma análise detalhada da forma como os *faultlets* foram projetados explica esse resultado: os *faultlets* foram projetados para descartar pacotes inteiros e não *chunks* dentro dos pacotes. Assim, sempre que um pacote é descartado, todos os fluxos sofrem a mesma perda. Note-se que a incapacidade em fornecer um melhor desempenho quando o número de fluxos é maior não é uma limitação do SCTP, mas sim da inexistência de rotas alternativas que pudessem ser usadas pelos diferentes pacotes.

Tabela 2: tempo de processamento das configurações com injeção de falhas

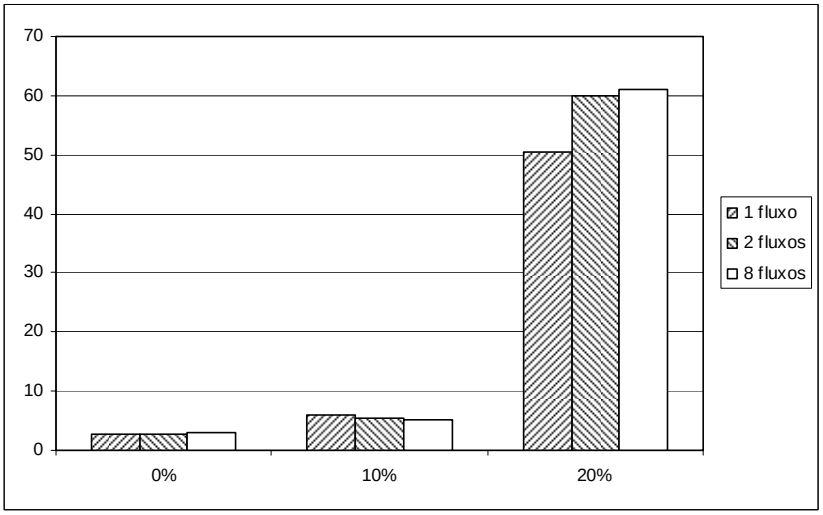
Número de fluxos	Tempo (seg) descarta 10%	Tempo (seg) descarta 20%	Tempo (seg) descarta 40%
1	6,095	50,501	2.287,204
2	5,313	60,011	1.961,677
8	5,212	61,083	2.152,372

O protocolo SCTP detecta a perda de pacotes através da temporização (*timeout*). O valor padrão para esse tempo é de 1 segundo. Esse tempo foi definido em função da aplicação para a qual o protocolo foi, originalmente, projetado (sinalização telefônica). Caso o protocolo SCTP seja usado para transportar informações de aplicações de dados, esse valor de temporização pode não ser adequado.

Os dados da tabela 1 e 2 estão representados na figura 2, onde pode-se verificar o aumento exponencial do tempo de processamento com o aumento do percentual de descarte de pacotes. No gráfico (a) da figura, pode-se verificar que o valor do tempo de processamento para um percentual de 40% de descarte é muito maior do que os outros, dificultando a visualização dessas outras medidas. No gráfico (b) da figura as medidas para 40% de descarte foram removidas, de maneira a evidenciar o comportamento das medidas restantes. Nesse gráfico, identifica-se, ainda, a tendência exponencial de crescimento com o aumento do percentual de descarte de mensagens.



(a)



(b)

Figura 2 – Tempo de processamento das configurações

5.3 Duplicação de pacotes

O protocolo prevê o envio de mensagens de reconhecimento da recepção de cada mensagem de dados. Assim, caso um pacote seja perdido, o transmissor identificará esse fato através do esgotamento do tempo de espera pela mensagem de reconhecimento e reenviará o pacote perdido. Entretanto, pode ocorrer da mensagem de reconhecimento chegar ao destino após ter esgotado o tempo de espera e, portanto, ter sido enviada uma repetição da mensagem. Com isso, o destinatário da mensagem de dados receberá dois pacotes idênticos, o que, segundo a especificação do protocolo SCTP, será detectado e corrigido (a instância do SCTP no receptor não entregará a duplicata para a aplicação).

Para verificar a robustez da implementação do SCTP, escreveu-se um *faultlet* capaz de duplicar todos os pacotes de dados enviados pelo cliente SCTP. Observou-se que a aplicação não percebia a duplicação de pacotes, uma vez que recebia um único exemplar de cada: a instância do SCTP no receptor filtrava as repetições do pacote. O

teste mostra que na ocorrência de duplicação de pacotes, a implementação do protocolo está em conformidade com a especificação.

5.4 Alteração dos *tags* de verificação

Durante o processo de estabelecimento de uma associação SCTP, os terminais envolvidos trocam informações que permitem a identificação da mesma. Essa identificação é feita através de *tags* de verificação, um para cada direção de fluxo (cliente para servidor e servidor para cliente), e identificam a associação de maneira inequívoca, de forma a impedir a recepção de dados de outras associações. Essa verificação ocorre na instância de implementação do SCTP que está sendo executada no receptor dos pacotes.

Para verificar a implementação do SCTP, escreveu-se um *faultlet* que injeta falhas no *tag* de verificação. Com isso, o pacote alterado não deverá ser reconhecido no receptor como sendo da associação corrente e, portanto, não será entregue para a aplicação. O procedimento para a aplicação do *faultlet* foi o seguinte: estabeleceu-se uma associação entre cliente e servidor; em seguida, o *faultlet* foi ativado; então, foram enviadas mensagens de dados.

Usando um dos programas monitores fornecidos com a biblioteca lksctp, observou-se que os pacotes SCTP chegavam ao destinatário. Entretanto, em conformidade com a especificação, as mensagens desses pacotes não eram entregues à aplicação, uma vez que o *tag* desses pacotes não correspondia ao *tag* registrado na associação estabelecida.

6 Conclusão e trabalhos futuros

Esse artigo mostrou que um injetor de falhas de comunicação operando no kernel do sistema operacional Linux é extremamente útil para o teste da implementação de protocolos de rede construídos sobre IP. Ganchos na pilha de protocolo permitem selecionar e atuar sobre todas as mensagens que fluem pela pilha. Uma ferramenta como FIRMAMENT, aplicada nos experimentos aqui reportados, pode tanto ser usada pelo desenvolvedor para o teste sob falhas de sua implementação como por um testador independente para teste de conformidade da implementação à especificação nas situações de falhas previstas. Injeção de falhas não substitui outros testes convencionais necessários, apenas facilita criar cenários de falhas usuais ao ambiente onde o protocolo vai operar. Tais cenários, uma combinação de carga de falhas e carga de trabalho, permitem o máximo de flexibilidade e controlabilidade para o teste sob falhas de comunicação.

A análise da implementação do protocolo SCTP através do uso de injetores de falhas não está encerrada. Os resultados obtidos possibilitaram verificar algumas das características de tratamento de erros da implementação do SCTP para Linux. Adicionalmente, no caso da perda de mensagens, foi possível avaliar o desempenho da implementação, o que levou ao questionamento do valor do tempo de espera para repetição da mensagem.

Em continuidade ao trabalho apresentado, alguns novos experimentos estão sendo propostos. O primeiro deles é a transmissão de dados através de rotas alternativas, que possibilitará avaliar a implementação no que diz respeito a essa

característica do protocolo. Além disso, será possível avaliar a eficiência com que o protocolo utiliza essa redundância.

Um segundo experimento será utilizar endereços alternativos para o estabelecimento dos fluxos de dados em uma associação SCTP. Dessa forma, serão possíveis, por exemplo, múltiplas conexões *http* independentes. O experimento consistirá em injetar falhas em alguns desses fluxos e observar o efeito nos outros fluxos. Segundo a especificação do protocolo, não deve haver interferência.

Um experimento semelhante ao dos endereços alternativos é o de perda de *chunks* de dados. Novamente, a perda de um *chunk* de dados de um fluxo não poderá interferir nos demais fluxos. A injeção de falhas permitirá verificar essa independência e, além disso, avaliar a queda de desempenho devido a essas falhas.

Como conseqüência das conclusões apresentadas nesse trabalho de que o desempenho piora significativamente quando ocorrem perdas de mensagens, um experimento natural é o de reduzir-se esse tempo e avaliar o impacto no desempenho. Como resultado desses experimentos espera-se poder sugerir uma temporização mais adequada ao uso do protocolo SCTP para o transporte de dados.

Finalmente, experimentos de injeção de falhas desta complexidade, conduzidos sobre protocolos reais implementados por terceiros permitirão um aprofundamento do conhecimento sobre as reais necessidades de testadores de implementações de protocolos. Com base no conhecimento prático adquirido durante os experimentos, ferramentas de injeção de falhas podem ser melhor definidas e projetadas para aumentar sua flexibilidade, funcionalidade, usabilidade e portabilidade.

7 Referências bibliográficas

Arlat, J.; Aguera, M.; Amat, L.; Crouzet, Y.; Laprie, J.; Fabre, J.; Martins, E.; Powell, D. (1990). Fault-injection for dependability validation: a methodology and some applications. IEEE Trans. on Software Eng., Special Issue on Experimental Computer Science, New York, vol. 16, n.2, p. 166-82, Feb.

Carreira, J.; Madeira, H.; Silva, J. G. (1998). Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers. IEEE Trans. on Software Eng., p. 125-135.

Chandra, R,; Lefever, R. M.; Joshi, K. R.; Cukier, M.; Sanders, W. H. (2004). A Global-State-Triggered Fault Injector for Distributed System Evaluation. IEEE Transactions On Parallel And Distributed Systems, v. 15, n. 7, July, p. 593-605

Dawson, S; Jahanian, F.; Mitton, T. (1996). ORCHESTRA: A Probing and Fault Injection Environment for Testing Protocol Implementations. Proceedings of IPDS'96. Urbana-Champaign, USA.

Drebes, R. J. (2005). FIRMAMENT: Um Modulo de Injeção de Falhas de Comunicação para Linux. Dissertação (Ciências da Computação) - Universidade Federal do Rio Grande do Sul

Drebes, R. J.; Silva, G. Jacques; Trindade, J. M. F.; Weber, Taisy Silva. (2006). A Kernel-based Communication Fault Injector for Dependability Testing of Distributed

Systems. Hardware and Software, Verification and Testing, First International Haifa Verification Conference, Revised Selected Papers. Lecture Notes in Computer Science. Berlin- Heidelberg: Springer-Verlag,. v. 3875, p. 177-190.

Han, S.; Shin, K. G.; Rosenberg, H. A. (1995). Doctor: An Integrated Software Fault-Injection Environment for Distributed Real-Time Systems. Second Annual IEEE Int'l Computer Performance and Dependability Symp., 1995, Erlangen. Los Alamitos. p. 204-13.

Hsueh, Mei-Chen; Tsai, T. K.; Iyer, R. K. (1997). Fault Injection Techniques and Tools. Computer, april, pp. 75-82

Jacques-Silva, G.; Drebes, R. J.; Trindade, J.; Weber, Taisy Silva; Pôrto, I. (2006). A Network-level Distributed Fault Injector for Experimental Validation of Dependable Distributed Systems. COMPSAC 2006. 30th Annual International Computer Software and Applications Conference. Chicago. IEEE Computer Society Press, 2006, v. 1. p. 421-428.

Looker, N.; XU, J. (2003). Assessing the dependability of OGSA middleware by fault injection. Proc. of the 22nd SRDS, p. 293–302, Florence, Italy.

Martins, E.; Rubira, C.M. F.; Leme, N. G. M. (2002). Jaca: A reflective fault injection tool based on patterns. Proceedings of the International Conference on Dependable Systems and Networks (DSN'02), pages 483–487.

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

Some, R. R. et al. (2001) A Software-Implemented Fault Injection Methodology for Design and Validation of System Fault Tolerance. IEEE Int'l Symposium on Dependable Systems and Networks, 2001, Göteborg (Sweden). p. 501-6.

Stewart, R. et al. (2000). Stream Control Transmission Protocol: RFC 2960. [S.1.]: Internet Engineering Task Force, Network Working Group.

Stewart, R. et al. (2004) Stream Control Transmission Protocol (SCTP) partial reability extension: RFC 4460. [S.1.]: Internet Engineering Task Force, Network Working Group.

Stewart, R. et al. (2006) Stream Control Transmission Protocol (SCTP) specification errata and issues: RFC 4460. [S.1.]: Internet Engineering Task Force, Network Working Group.

Stott, D. T.; Floering, B.,; Burke, D.,; Kalbarcyzk, Z.; Iyer, R. K. (2000). NFTAPE: a Framework for Assessing Dependability in Distributed Systems with Lightweight Fault Injectors. IEEE International Computer Performance and Dependability Symposium, pp. 91-100.