# Testes por injeção de falhas sobre o protocolo de transporte SCTP

**Bruno Coswig Fiss**[1], **Sérgio Luis Cechin**[1], **Taisy Silva 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

bcfiss,cechin,taisy@inf.ufrgs.br

***Resumo.*** *Neste trabalho apresenta-se a avaliação experimental sob falhas da implementação do protocolo SCTP. Para a injeção de falhas foi utilizado o FIRMAMENT, ferramenta recente com boa capacidade de especificação de cenários de falhas e fácil adaptação a novos protocolos. Os motivos principais para essa avaliação são: analisar a tolerância a falhas característica do SCTP, estudar a capacidade desse de tornar-se um dos principais protocolos de transporte sobre IP, e prosseguir com a experimentação do FIRMAMENT, injetor de falhas de comunicação por software que, quando consolidado, será útil na avaliação sob falhas de protocolos de rede em geral, auxiliando tanto no seu desenvolvimento quanto na análise das implementações existentes.*

## 1. Introdução

A injeção de falhas [Hsueh et al. 1997] utilizada neste artigo consiste na emulação de falhas reais através de software. Testes que utilizam essa técnica são ideais para a verificação do comportamento de aplicações tolerantes a falhas, pois tais aplicativos, quando desenvolvidos, são planejados de modo a se comportarem corretamente em cenários de falhas. Como algumas dessas situações raramente ocorrem durante o uso normal da aplicação, não é viável validar um sistema tolerante a falhas apenas utilizando-o por tempo prolongado . Através da injeção de falhas, pode-se forçar tais situações, criando-se meios para que sejam realizados os testes e se faça a validação dos sistemas de tolerância a falhas.

SCTP é um protocolo de comunicação, definido na RFC 2960 [Stewart et al. 2000], atualizado na RFC 3309 [Stone et al. 2002] e redefinido na RFC 4960 [Stewart et al. 2006]. O protocolo, no modelo de camadas TCP/IP, faz a comunicação entre a camada de rede e a de aplicação. Possuindo características que o diferenciam de protocolos mais conhecidos, como TCP, SCTP promete confiabilidade através da chegada confirmada de dados, não duplicação de pacotes e redundância de fluxos. Por essas razões, aplicações podem se beneficiar ao utilizá-lo, desde que os serviços prometidos por ele sejam confiáveis. Dessa forma, como um aplicativo utilizará uma implementação em software do protocolo, é essencial que essa possua dependabilidade. Assim SCTP mostra-se bom alvo para validação utilizando injeção de falhas.

A ferramenta de injeção utilizada é o FIRMAMENT [Drebes 2005], que funciona inserindo falhas dentro da pilha de protocolos. Seguindo exemplo de outros injetores, como ORCHESTRA [Dawson et al. 1996] e ComFIRM [Drebes et al. 2006], a ferramenta utiliza ganchos (*hooks*). No caso do FIRMAMENT, os ganchos estão na pilha

de protocolos. Dessa forma, todas as mensagens transmitidas na rede passam por ele. Contando ainda com ajuda dos faultlets, FIRMAMENT facilita a criação de cenários de falhas para diferentes protocolos, e tem baixo nível de instrusão, pois funciona como módulo do kernel. Possui também código aberto, diferente do que acontece com o injetor ORCHESTRA, o que possibilita o estudo do seu funcionamento e permite a sua alteração de acordo com a GPL. Além disso, FIRMAMENT é recente, fato que incentiva sua utilização para avaliação de corretude e desempenho.

As próximas seções tratam sobre FIRMAMENT (2), o protocolo SCTP (3), os testes realizados (4), em que se descreve o ambiente de testes e as dificuldades encontradas, além das soluções descobertas para a maioria das questões. Por último há uma seção que lista futuros trabalhos relacionados com este artigo, e as conclusões (5).

## 2. O FIRMAMENT

Essa ferramenta foi criada com o objetivo de realizar injeção de falhas sobre protocolos de comunicação e sistemas distribuídos. Tem poder de especificação de cenários de falhas complexos e baixa intrusividade [Drebes 2005]. Funciona com injeção por software, e utiliza faultlets para criação dos cenários de falhas.

FIRMAMENT foi baseado principalmente no injetor ComFIRM, que funciona por meio da modificação do cógido-fonte do núcleo Linux. Essa ferramenta disponibiliza os mesmos tipos de falhas que FIRMAMENT: omissão de envio e recebimento, colapso de nó e canal, temporarização e falhas bizantinas, porém é mais intrusiva, tem menos poder de expressão, e tornou-se incompatível com novos núcleos, motivando a criação do FIRMAMENT. Como forma de interceptação de pacotes, FIRMAMENT utiliza a interface NetFilter [Russel and Welte 2002], *framework* do Linux presente desde a versão 2.4 do seu núcleo, tornando-se independente da arquitetura da máquina, e reduzindo sua intrusividade, não dependendo diretamente do núcleo do sistema, como o ComFIRM.

O alto poder de expressão da ferramenta se dá pela implementação do conceito de faultlets. Esses emulam um cenário de falhas, através de um código interpretado pela máquina virtual do FIRMAMENT, FIRMVM. Os cenários são previamente descritos em linguagem semelhante à linguagem de máquina, com mnemônicos, e então transformados em arquivos binários. Essas etapas buscam melhor desempenho na execução do injetor, evitando o processamento, durante a injeção, de código mais elaborado, ou até com erros.

Com o surgimento de núcleos novos, FIRMAMENT passou a necessitar de adaptações. Atualmente a ferramenta está em processo de desenvolvimento, e as dificuldades encontradas até agora são oriundas de atualizações do Netfilter das novas versões[1], restando analisar quais foram exatamente as alterações que causaram tais problemas. Como o injetor passou a apresentar incompatibilidades nas novas versões do núcleo e o objetivo deste artigo é testar a implementação do protocolo SCTP, que pode ser instalada em qualquer kernel a partir do 2.4, desde que devidamente atualizado, o núcleo utilizado na realização dos testes foi o 2.6.18.

## 3. SCTP

SCTP, ou *Stream Control Transmission Protocol*, é um protocolo criado com a principal função de permitir transmissão de sinais RTPC, rede de telefonia pública comutada, por

[1] A versão estável mais recente do núcleo Linux é a 2.6.26.

IP. Apesar desse objetivo inicial, após o seu desenvolvimento, percebeu-se sua utilidade para um grande número de tarefas. O protocolo apresenta funções semelhantes às do TCP ou do UDP, e permite escolha entre ativar ou não o envio de mensagens confiável, com aviso de recebimento, e o recebimento das mensagens em ordem de envio. Essa capacidade de escolha já permitiria a substituição desses dois protocolos por SCTP. Além disso, ele possui características importantes, como suporte a tolerância a falhas em nível de rede através de redundância de caminhos para um mesmo endereço, tipo de conexão diferente da do TCP, chamada associação, baseada em múltiplos endereços de origem e de destino, e vários fluxos.

Com esses atributos, SCTP pode ser usado para subtituir UDP e TCP, porém apresenta desvantagens em relação a esses. Como possui mais opções, necessita de um cabeçalho mais elaborado e de mais tempo para execução de suas funções. Assim, num cenário em que é necessária somente uma conexão de envio ordenado e com transmissão confiável de dados, TCP é mais eficiente que SCTP [Coene 2002]. Em vista dessa desvantagem, é importante que uma aplicação ou protocolo possa utilizar as características de opção de estilo de envio e/ou múltiplos fluxos para fazer proveito do SCTP, e que possa confiar na dependabilidade oferecida pelo protocolo. Um exemplo de aplicação seria o protocolo http, que pode fazer uso de fluxos diferentes numa mesma associação SCTP para envio de cada elemento de uma página [Cechin et al. 2008].

O desenvolvimento do protocolo para o Linux foi realizado pela OpenSS7 Project (disponível em: http://www.openss7.org/), desenvolvedora de sistemas de código aberto, principalmente de rede, para Linux, entre eles SS7, ISDN e VoIP. Como o protocolo é desenvolvido em software, e todo software possui erros residuais, qualquer implementação deve ser testada em situações de operação extremas, como sob falhas. Isso é essencial tanto para o desenvolvimento da implementação do protocolo quanto para benchmarking de dependabilidade e certificação.

A biblioteca utilizada para disponibilização de rotinas SCTP para o teste é a lksctp (disponível em: http://lksctp.sourceforge.net/). Essa também tem por objetivo implementar o protocolo no núcleo Linux e disponibilizar às aplicações soluções SCTP através das características apresentadas nas definições do protocolo.

Assim, é importante verificar a implementação do SCTP, encontrando erros possíveis e aumentando a sua confiabilidade, permitindo que aplicações possam utilizá-lo com mais segurança [Yarroll and Knutson 2001].

## 4. Procedimento

O objetivo inicial da experimentação realizada foi retomar os trabalhos descritos no artigo de Cechin, S., Nedel, W. e Weber, T. [Cechin et al. 2008], e por Nedel, W. [Nedel 2007]. Portanto, para realização dos testes, estudou-se o método utilizado nesses trabalhos e verificou-se em que aspectos ainda era possível corrigi-lo e melhorá-lo. Haviam sido usados o aplicativo *sctp_test*, pertencente à biblioteca lksctp , o kernel 2.6.21 do Linux, o FIRMAMENT 0.26, e os dados dos testes precisavam ser validados. Com isso, iniciou-se a realização dos testes propostos no trabalho citado. Posteriormente, várias modificações foram implementadas.

Os faultlets publicados foram os de descarte e duplicação de pacotes, mudança de tag de verificação e alteração de checksum. Ao se estudar em particular o faultlet de

descarte, que realiza a injeção da falha de acordo com um número aleatório, usado no descarte de 10%, 20% e 40% dos pacotes SCTP que passam pela camada IP, descobriu-se que o número utilizado nos faultlets, entre -100 e 100, é menos ”justo”do que um de maior módulo, e.g. -10000 e 10000. Isso acontece pois o algoritmo de geração de números aleatórios baseado nos geradores de Tausworthe (1965), utilizado no FIRMAMENT, funciona mais homogeneamente com números maiores. Criou-se também um contador do número de pacotes que passam pela ação do faultlet, e da quantidade dos que sofrem injeção, visando melhor acompanhamento das ações do injetor, e das razões pelas quais possíveis atrasos ocorreriam.

Após a criação de faultlets com devidos melhoramentos, configurou-se o ambiente de testes. O módulo FIRMAMENT e a biblioteca lksctp foram instalados, e o programa exemplo que acompanha lksctp, chamado *sctp_test*, foi utilizado como carga de trabalho.

Esse programa, *sctp_test*, envia um número escolhido de mensagens de um tamanho selecionado, em um número de fluxos também determinado. Após alguns testes realizados com essa aplicação, constatou-se grande variação no tempo de envio de qualquer número de mensagens, a qualquer momento. Isso foi causado, aparentemente, pela natureza da aplicação, criada simplesmente para testar a operação do protocolo entre máquinas, não utilizando o sistema de mensagem de resposta, que verifica se cada mensagem enviada chegou ao destino, e estabelece as associações (o programa permitia o envio de mensagens para um servidor inexistente). Dessa forma, detectou-se que, quando o envio ocorria em tempos muito baixos, a associação não tinha sido estabelecida corretamente, de forma que os pacotes foram enviados sem espera por resposta. Nas demais ocasiões, a associação foi realizada com sucesso, fazendo com que o tempo de envio de mensagens fosse maior. Os detalhes desse processo estão sendo investigados mais profundamente, ficando sua análise detalhada como trabalho posterior.

Para solucionar esse problema, foi utilizado um programa que acompanha a biblioteca, similar ao *sctp_test*, chamado *sctp_darn*. Esse programa havia sido utilizado nos testes do trabalho de conclusão citado [Nedel 2007], e respeita o ordenamento e o sistema de mensagens de resposta. Ele foi configurado para envio de mensagens para um ou mais receptores, porém sem opção para envio de mensagens pré-configuradas, e escolha de seus tamanhos, pois o objetivo da aplicação era permitir ao usuário a escrita e envio das mesmas. Como era importante a comparação dos resultados desse experimento com os citados anteriormente [Cechin et al. 2008], também devia ser possível o teste cronometrado do envio de um conjunto de mensagens, pois os experimentos realizados naquele trabalho justificaram através desse recurso o funcionamento da duplicação de pacotes, e usaram-no para estudar o atraso causado pelo descarte de pacotes.

Assim, modificações simples, que não comprometem a estrutura de envio de mensagens, foram adicionadas ao *sctp_darn* com o objetivo de realizar testes sobre SCTP. O programa oferecia opções de envio interativo e outras que aumentavam o tempo de processamento do programa, tirando a exatidão dos testes temporizados. Essa parte foi retirada do código. Com o intuito de enviar várias mensagens de determinado tamanho, foram criados trechos que substituíram o recebimento de uma mensagem do teclado por uma pré-configurada, declarada no código, e colocou-se um laço na parte de envio, que repetia essa função um número selecionado de vezes. A mensagem enviada podia ser alterada, para que se pudesse escolher seu tamanho, e continha um contador indicando

qual era a sua posição dentre todas as enviadas, para testar o sistema de numeramento da própria implementação do SCTP.

## 5. Dados dos experimentos

Os testes de validação do mecanismo de checksum, alteração do tag de verificação para teste do sistema de negociação de associação, e o teste de duplicação de pacotes forneceram dados que resultam em conclusões similares às relatadas no artigo supracitado [Cechin et al. 2008], agora com mais dados e confiabilidade.

Ao se modificar o campo de tag do pacote SCTP, utilizada para identificação do fluxo ao qual pertence o pacote, foi constatado que as mensagens não chegavam à aplicação servidor, demonstrando que a implementação do protocolo testada se encarrega de descartar mensagens que vêm de origens com as quais não foi negociada uma associação, o que é previsto na especificação do protocolo.

Ao fazer uma alteração no campo de checksum do pacote SCTP, mecanismo para verificação de integridade da mensagem, as mensagens foram descartadas silenciosamente antes de chegarem à aplicação servidor, similarmente ao teste de alteração do tag de verificação, mostrando que a implementação do protocolo se encarregou de retirar qualquer mensagem que possa ter sido corrompida.

No caso da duplicação, notou-se diferença em relação ao teste mencionado [Cechin et al. 2008], mas isso foi causado pela mudança de carga de trabalho (*sctp_darn*), o que alterou o cenário. Os dados levaram às mesmas conclusões que haviam sido obtidas. As mensagens duplicadas possuem as mesmas informações, inclusive o número de seqüência. Esse número será igual ao da mensagem original, e através desse mecanismo, o protocolo percebe mensagens duplicadas e as descarta silenciosamente. Como pôde ser visto nos experimentos, nenhuma mensagem chegou duplicada, mesmo com a ação do injetor. Para comprovar a ação do injetor, foi medido o tempo de envio e recebimento das mensagens como forma de verificar o aumento da carga na rede causado pela duplicação de 100% dos pacotes SCTP enviados. Essa diferença temporal é pequena, mas foi comprovada em 120 medições, com um intervalo de confiabilidade de 90%.

Na tabela 1 os dados de envio de 100 mensagens de 1500 bytes, em milissegundos:

**Tabela 1. Tempo do envio de dados**

| Falha | Tempo médio | Valor mínimo | Valor máximo |
|---|---|---|---|
| Nenhuma | 3208 | 3201 | 3214 |
| Duplicação | 4457 | 4448 | 4467 |

O último teste, de descarte de pacotes, tem grande importância, pois gerou novas questões, relativas aos atrasos. Essas questões são oriundas de dados relevantes encontrados, que atualmente exigem mais investigação para se tornarem conclusivos. Descobriu-se que, apesar de os tempos de envio de mensagens com maior percentual de descartes serem maiores, a quantidade de descartes não é diretamente proporcional ao tempo de envio das mensagens. Para identificar as causas dessas variações temporais, busca-se isolar cada elemento que possa causar interferência nos experimento e verificar se outros protocolos, como UDP e TCP, apresentam tal comportamento em cenários similares. Essas

causas podem estar ligadas ao protocolo, à ferramenta de injeção FIRMAMENT, a outros elementos relacionados, como NetFilter, ou ainda à própria implementação da pilha de protocolos do Linux.

## 6. Conclusão e trabalhos posteriores

Até o presente momento, pode-se concluir que a implementação do protocolo segue a especificação quanto aos mecanismos de tolerância a falhas que oferece. Entretanto é necessário o término dos testes para que se façam avaliações mais completas. As medições realizadas neste trabalho revelam questões que poderão auxiliar no desenvolvimento de benchmarking de dependabilidade do protocolo, em suas futuras implementações e principalmente no aprimoramento do FIRMAMENT. As justificativas das variações temporais serão úteis para o desenvolvimento de ambos.

Fica evidente a necessidade de continuar os experimentos com descarte de pacotes, possivelmente utilizando como referência testes sobre outros protocolos de transporte, que poderão explicitar se há relação direta da implementação do SCTP com as variações temporais. Deve-se ainda estudar as variações dos resultados obtidos com o *sctp_darn* em comparação com os do *sctp_test*, utilizado originalmente nos testes publicados [Cechin et al. 2008].

A ferramenta FIRMAMENT deve ser atualizada, trabalho que já está em andamento no grupo de tolerância a falhas da UFRGS.

Testes diferentes, como os de duplicação e descarte mas com múltiplos fluxos associados a diferentes rotas, perda de *chunks* de dados e atraso de pacotes, são importantes para uma validação mais apurada da implementação do SCTP.

É importante a criação de um cliente/servidor sobre SCTP mais robusto, que permita escolha quanto às opções da associação, tornando possível o teste do protocolo em situações em que nem todas as características de tolerância a falhas são utilizadas, ou naquelas em que opções de menor RTO[2] são determinadas.

Espera-se, finalmente, fornecer subsídios para futuras implementações do protocolo SCTP e também para desenvolvimento de novas técnicas de injeção de falhas sobre protocolos de comunicação.

## Referências

Cechin, S., Nedel, W., and Weber, T. (2008). Teste por injeção de falhas da implementação do protocolo de comunicação sctp. *SBRC - Workshop de Testes e Tolerância a Falhas*.

Coene, L. (2002). Stream Control Transmission Protocol Applicability Statement. RFC 3257 (Informational).

Dawson, S., Jahanian, F., and Mitton, T. (1996). ORCHESTRA: A fault injection environment for distributed systems. Technical Report CSE-TR-318-96.

Drebes, R. J. (2005). Firmament: Um módulo de injeção de falhas de comunicação para linux, tese de mestrado.

[2] *Retransmission timeout*, definido como o tempo de espera de recebimento de mensagem de resposta no SCTP. É o tempo que o protocolo espera para considerar que uma mensagem não atingiu seu destino.

Drebes, R. J., Jacques-Silva, G., da Trindade, J. M. F., and Weber, T. S. (2006). A kernel-based communication fault injector for dependability testing of distributed systems. In Ur, S., Bin, E., and Wolfsthal, Y., editors, *Haifa Verification Conference*, volume 3875 of *Lecture Notes in Computer Science*, pages 177–190. Springer.

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

Nedel, W. (2007). Avaliação do protocolo sctp através do injetor de falhas firmament.

Russel, R. and Welte, H. (2002). Linux net filter hacking howto.

Stewart, R., Arias-Rodriguez, I., Poon, K., Caro, A., and Tuexen, M. (2006). Stream Control Transmission Protocol (SCTP) Specification Errata and Issues. RFC 4460 (Informational).

Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and Paxson, V. (2000). Stream Control Transmission Protocol. RFC 2960 (Proposed Standard). Obsoleted by RFC 4960, updated by RFC 3309.

Stone, J., Stewart, R., and Otis, D. (2002). Stream Control Transmission Protocol (SCTP) Checksum Change. RFC 3309 (Proposed Standard). Obsoleted by RFC 4960.

Yarroll, L. M. H. P. and Knutson, K. (2001). Linux kernel sctp : The third transport.