Injeção de Falhas de Comunicação por Sondagem Dinâmica*

Roberto Jung Drebes, Taisy Silva Weber

Instituto de Informática – Universidade Federal do Rio Grande do Sul
Caixa Postal 15064 – 90501-970 Porto Alegre, RS, Brasil

{drebes,taisy}@inf.ufrgs.br

Abstract. *This paper presents fault injection as a technique for application and protocol validation. Associating fault injection to network emulation, it shows the techniques used for communication fault injector implementation in the Linux environment. Finally, it suggests the use of dynamic probing (DProbes) for new fault injector building.*

Resumo. *Este trabalho apresenta a injeção de falhas como técnica de validação de protocolos e aplicações. Associando-a à emulação de redes, mostra as técnicas usuais de implementação de injetores de falhas de comunicação em ambiente Linux. Por fim, sugere a utilização de sondagem dinâmica (DProbes) para a construção de novos injetores de falhas.*

1. Injeção de Falhas

A adoção dos sistemas computacionais nas mais diversas tarefas exige a dependabilidade destes sistemas. De forma simplificada, *dependabilidade* abrange duas características principais: *disponibilidade*, a capacidade de um sistema estar pronto para uso; e *confiabilidade*, a capacidade desse sistema continuar em serviço, mesmo na presença de falhas.

É necessário, então, que existam técnicas para a avaliação da dependabilidade. Estas técnicas funcionam através do estudo do comportamento do sistema ou de um modelo dele na presença de falhas. Nas *técnicas experimentais* a idéia é observar o sistema real sobre a presença de falhas e daí levantar conclusões sobre sua dependabilidade. Técnicas relacionadas ao uso, isto é, simplesmente utilizar o sistema aguardando a ocorrência natural de falhas observando como ele responde às mesmas, possuem um inconveniente: falhas naturais podem levar longos períodos para ocorrerem. Estatisticamente, é possível acelerar sua ocorrência ao observarmos um grande número de sistemas idênticos simultaneamente, mas nesse caso troca-se a necessidade de tempo pela de quantidade.

Uma idéia para acelerar a ocorrência de falhas é mudarmos para uma atitude ativa perante o sistema, criando falhas através de um *injetor de falhas*. Trabalhando sobre o sistema real temos uma avaliação *efetiva* da sua dependabilidade, sem estarmos sujeitos à variabilidade temporal da ocorrência de falhas naturais, e, mais importante, podendo observar o sistema sob a ocorrência de falhas específicas, em momentos específicos.

*Desenvolvido em parceria com HP Brasil P&D e Projeto CNPq ACERTE (#472084/2003-8)

Injetores de falhas podem ser implementados em hardware ou software. Injetores por hardware atuam no nível elétrico, sendo efetivamente injetores de falhas; enquanto injetores por software atuam no nível lógico, sendo portanto na realidade injetores de erros. Entretanto, ambas as abordagens costumam ser chamadas de injeção de falhas. Neste artigo a ênfase é nos injetores de falhas por software.

2. Falhas de Comunicação

A comunicação de equipamentos eletrônicos é feita através de dispositivos que atuam sobre o ambiente físico, recebendo e transmitindo sinais eletromagnéticos. A ocorrência de interferência sobre esses sinais, bastante freqüente, se não tratada, pode acarretar na perda de informações. Além disso, a utilização de *buffers* finitos nos equipamentos implica na possível perda de mensagens. Por estas razões é comum o emprego de mecanismos de tolerância a falhas, como cálculos de verificação e correção ou retransmissão na comunicação.

Aplicações adaptativas permitem que mesmo com flutuações no nível de qualidade dos parâmetros de comunicação – taxas de transmissão e perda, latência e variância (*jitter*) – ela se adapte e permaneça fornecendo um serviço aceitável. No seu desenvolvimento, idealmente, deve-se testar essas aplicações sob cada cenário de comunicação. Uma possibilidade é executá-las sobre as diversas tecnologias de acesso, mas isto necessitaria de uma vasta gama de equipamentos, das diversas tecnologias. Outra alternativa é a emulação, através da variação desses parâmetros de comunicação, de várias tecnologias utilizando-se de apenas algumas delas, de forma que para a aplicação seja indistingüível uma rede com parâmetros reais de emulados. Aqui, novamente, a injeção de falhas de comunicação pode ser utilizada, não como mecanismo de validação das técnicas de tolerância a falhas de protocolos e algoritmos distribuídos, mas para esta conformação de tráfego.

3. Falhas de Comunicação em Ambiente Linux

Os experimentos de injeção de falhas só podem ocorrer na plataforma que suporta as aplicações a serem testadas ou validadas. O sistema operacional Linux atingiu maturidade. Além disso, pela sua origem, é imediatamente associado às aplicações de rede e comunicação. E a mesma disponibilidade do código fonte que torna atraente sua escolha como ambiente de experimentação com injeção de falhas também lhe dá portabilidade entre desde equipamentos embarcados de aplicação específica de comunicação, passando por servidores, chegando até *clusters*, *grids*, etc. Projetos como *Carrier Grade Linux* [Open Source Development Labs, 2003], e o *Data Center Linux* [Open Source Development Labs, 2004] tem como objetivo empregar este sistema em ambientes onde a alta disponibilidade é essencial, como justamente serviços de telecomunicações e provimento de dados.

Definida a plataforma alvo ainda é necessário determinar a técnica e localização utilizadas na injeção de falhas. A injeção de falhas por software, além de mais barata, é mais apropriada quando se deseja trabalhar com estruturas de dados de mais alto nível do sistema operacional, como pacotes de comunicação. Quanto à localização, ela pode

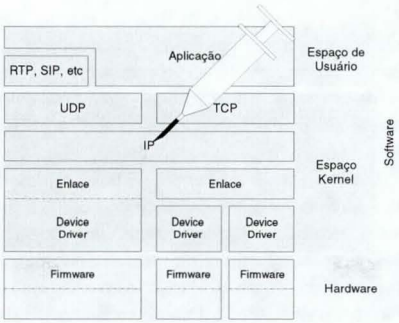

Figura 1: Localização da Injeção de Falha de Comunicações

ocorrer em qualquer ponto do trajeto percorrido pelas mensagens desde a aplicação de origem até a de destino, nas diversas camadas de comunicação.

A localização de um injetor de falhas será determinada pelo local no sistema operacional da implementação dos protocolos associados a cada uma das camadas de comunicação. Aplicações são implementadas como processos de usuário. Assim, para injetarmos falhas associadas ao nível de aplicação, o mais apropriado é através da modificação da aplicação. Há casos onde existem protocolos intermediários entre os níveis de aplicação e transporte, implementando ou funções dos níveis de sessão e apresentação, ou *middleware* com novas funções. Exemplos dessas camadas intermediárias são os protocolos RTP, XDR, SIP, ou ainda *middlewares* de comunicação de grupo ou de suporte a computação em *grid*. No Linux esses protocolos costumam ser implementados como bibliotecas de ligação dinâmica. Os protocolos de nível de transporte (UDP e TCP) são implementados internamente ao *kernel* e utilizados através de chamadas de sistema. Estes, por sua vez, utilizam o protocolo de rede IP, também implementado dentro do *kernel*, que utiliza os *drivers* de dispositivo para ter acesso aos diversos tipos de enlace. Finalmente, o nível físico é implementado através do *firmware* gravado diretamente no controlador de rede.

Se o objetivo de um injetor de falhas é emular as características de desempenho de rede para diversas tecnologias de acesso, sua implementação mais apropriada torna-se no nível IP, pois por suas características (serviço de datagrama, sem retransmissão) este protocolo simplesmente espelha as limitações dos níveis inferiores. A perda ou atraso de um quadro no nível de enlace implica no mesmo comportamento de um pacote IP, com a vantagem de termos a independência de tecnologia de acesso neste protocolo. Portanto, a modificação das dinâmicas de desempenho dos pacotes IP pode emular características dos protocolos inferiores, sendo o que mais se aproxima das falhas de comunicação dos níveis de acesso à rede. A relação entre as camadas e sua implementação no sistema operacional Linux pode ser observada na Figura 1.

Por ser a implementação do protocolo IP do Linux interna ao *kernel*, qualquer modificação que vise alterar o processamento de pacotes deve ter acesso às suas estruturas internas. A arquitetura de sistemas operacionais monolíticos isola processos de usuário

das estruturas do *kernel*: todo o acesso às suas funções é feito através de chamadas de sistema, e a memória a ele associada fica protegida de acessos dos processos. Para atuarmos sobre os pacotes dentro do *kernel*, portanto, devemos estar executando código interno ao mesmo. Uma possibilidade é a modificação do *kernel*, através da sua rescrita e nova compilação. Essa abordagem, utilizada em injetores como ComFIRM [Leite, 2000], apesar de obter excelentes resultados, possui o inconveniente da necessidade de adaptação dessas modificações a cada nova versão do *kernel*. É possível, ainda, utilizar as chamadas de sistema `ptrace()` que param a execução de processos a cada entrada e saída de outras chamadas de sistema. Ao parar o processo em chamadas de comunicação (recebimento e envio de dados) é possível omitir ou atrasar estas funções. Entretanto, essa técnica tem efeito considerável na carga do sistema, pois efetua trocas de contexto a cada chamada de sistema. Mesmo assim, esta técnica, utilizada dentre outras na ferramenta INFIMO [Barcelos et al., 2004], não necessita a modificação de código de sistema. Apresentamos a seguir um novo recurso para depuração do ambiente Linux, as sondas dinâmicas, que possibilita mais uma abordagem para a injeção de falhas.

4. Sondagem Dinâmica como Forma de Injeção de Falhas de Comunicação

Um mecanismo de sondas dinâmicas (DProbes, *dynamic probes*) foi recentemente incorporado ao Linux, estando disponível através de *patches* para a versão estável do *kernel* (2.6), com perspectivas de inclusão à versão oficial. As especificações *Carrier Grade Linux* e *Data Center Linux* requisitam mecanismos de depuração dinâmica e para isto já utilizam este mecanismo. As sondas dinâmicas do Linux descendem diretamente de mecanismo semelhante existente no sistema operacional OS/2 da IBM (*Dynamic Trace*) [Moore, 2000], mas mecanismos de *trap* similares existem também em outros sistemas como z/OS, OS400, AIX e z/VM, embora não de forma tão extensiva e generalizada.

As sondas por software são inspiradas nas ponteiras de sondagem, dispositivos de hardware tipicamente usados para a monitoração e injeção de falhas em circuitos em funcionamento. Apesar de operarem por software, também apresentam baixa intrusão, pois permitem que o fluxo de um programa (tanto em espaço de usuário quanto de sistema) seja rápida e facilmente desviado para uma função de teste ou injeção de falhas. Essa baixa intrusão é interessante ao trabalharmos com protocolos, pois estes obedecem a regras temporais de operação entre os parceiros da comunicação. Além disso, a arquitetura do mecanismo implica em uma baixa dependência das características do sistema operacional hospedeiro, o que lhe garante portabilidade entre variantes de sistemas operacionais UNIX.

DProbes é um habilitador de outras tecnologias de depuração [Moore, 2001]. Seu principal componente é um mecanismo para a interceptação em qualquer ponto de execução (*probepoint*, ponto de sondagem). A cada ponto de sondagem é associada um tratador de sondagem (*probe handler*), que corresponde a uma função com acesso à memória do *kernel*, de usuário e aos registradores do processador.

O mecanismo usado é semelhante a um *watchpoint*, mas utiliza-se o termo *probe* porque a execução não é parada, apenas executa-se uma função pré-determinada. Por não exigir interatividade, sua implementação como um tratador de interrupção torna-se

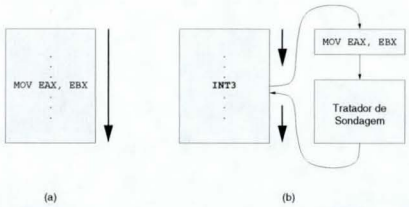

Figura 2: Código não modificado (a) e Implementação de pontos de sondagem na arquitetura IA32, através da substituição de instruções (b)

mais apropriada. Como interrupções chaveiam o processador para modo privilegiado, é possível ainda atuar em uma tarefa, interrupção ou mesmo troca de contexto. Um *probe* permite a observação e modificação do estado de um sistema em qualquer ponto do código sem intrusão significativa.

Em cada ponto de sondagem, DProbes substitui a instrução de máquina original por uma instrução de interrupção, como mostra a Figura 2 (b).Para tratar esta interrupção, ele executa isoladamente a instrução. Se esta executar sem a ocorrência de exceção, o controle é passado ao tratador de sondagem e em seguida retorna ao código original.

Ao colocar pontos de sondagem nas funções de envio de pacotes e no tratamento de interrupção do controlador de rede, um injetor de falhas de comunicação pode ser construído como uma aplicação do DProbes. Esta abordagem está sendo utilizada no desenvolvimento do novo ambiente de experimentação de falhas NEEDLE (*Network Experimentation Environment using DProbes for Link Errors*). Um exemplo de *probe* que intercepta o fluxo de execução sempre que uma mensagem é transmitida é mostrado na Figura 3.

5. Conclusões

As técnicas de injeção de falhas, já comuns para a validação dos mecanismos de detecção e recuperação de falhas, também podem ser aplicadas a emulação de cenários de redes de longa distância ou tecnologias sem fio. Isso facilita o teste de aplicações adaptativas, pois dispensa a necessidade de uma variedade de equipamentos de tecnologias distintas. Entretanto, a construção de um injetor de falhas por software pode ser localizada em diversos pontos de um sistema alvo de testes, e essa localização está relacionada aos níveis de rede onde se deseja injetar falhas.

Este artigo apresenta o mecanismo de depuração por sondas dinâmicas como nova técnica para a injeção de falhas de comunicação em ambiente Linux. A ferramenta NEEDLE implementada utilizando essa proposta pretende facilitar o trabalho de projetistas de protocolos e aplicações de rede, fornecendo uma plataforma útil e de fácil instalação para seus experimentos de validação.

Por utilizar o mecanismo de DProbes, NEEDLE implica em baixa intrusão, tanto em respeito ao desempenho das aplicações ou protocolos a serem validados, quanto à interferência no código do *kernel* utilizado nos testes, pois o injetor de falhas é construído

```
#include <linux/module.h>
#include <linux/kprobes.h>

struct kprobe kp;

void handler_post(struct kprobe *p, struct pt_regs *regs,
                                        unsigned long flags){
    printk("tratador de sondagem, dev_queue_xmit executando.");
}

int init_module(void)
{
    kp.pre_handler=NULL;
    kp.post_handler=handler_post;
    kp.fault_handler=NULL;
    kp.addr = (kprobe_opcode_t *) &dev_queue_xmit;
    register_kprobe(&kp);
    printk("probe instalado\n");
    return 0;
}

void cleanup_module(void)
{
    unregister_kprobe(&kp);
    printk("probe removido\n");
}
```

Figura 3: Examplo de *probe*. Uma mensagem é exibida a cada execução da função `dev_queue_xmit()`

como um *plug-in* de um *kernel* não modificado.

Referências

[Barcelos et al., 2004] Barcelos, P. P. A., Drebes, R. J., Jacques-Silva, G., e Weber, T. S. (2004). A toolkit to test the intrusion of fault injection methods. In *LATW 2004 Digest of Papers*, páginas 152–157, Cartagena de Indias, Colômbia. IEEE.

[Leite, 2000] Leite, F. O. (2000). ComFIRM – injeção de falhas de comunicação através da alteração de recursos do sistema operacional. Dissertação de Mestrado, Instituto de Informática/UFRGS, Porto Alegre, Brasil.

[Moore, 2000] Moore, R. J. (2000). Dynamic probes and generalised kernel hooks interface. In *Proceedings of the 4th Annual Linux Showcase*.

[Moore, 2001] Moore, R. J. (2001). A universal dynamic trace for Linux and other operating systems. In *Proceedings of the 2001 USENIX Annual Technical Conference*. USENIX.

[Open Source Development Labs, 2003] Open Source Development Labs (2003). Carrier grade Linux requirements definition version 2.0. Disponível em: `http://www.osdl.org/docs/carrier_grade_linux_requirements_definition___version_20.pdf`. Acesso em: Jun 2004.

[Open Source Development Labs, 2004] Open Source Development Labs (2004). Data center Linux technical capabilities: Executive summary version 1.0. Disponível em: `http://www.osdl.org/lab_activities/data_center_linux/DCL_ExecSumm_TechCapabilities_1_0.pdf`. Acesso em: Jun 2004.