S-Trace: Construindo Caminhos Causais em Redes Definidas por Software

Fabrício B. de Carvalho[1], **Ronaldo A. Ferreira**[1]

[1]Faculdade de Computação (FACOM)
Universidade Federal de Mato Grosso do Sul (UFMS)
CEP – 79070-900 – Campo Grande – MS – Brasil

{fabricio,raf}@facom.ufms.br

Abstract. *Determining hardware and software elements used to process an application request and group them in a set, called causal path, that exposes relevant parameters, such as processing time and delays, and that may explain the behavior of the application is a challenging task. Several tools for building causal paths have been proposed for the current Internet architecture, but none so far explores features offered by Software Defined Networks (SDN). This paper proposes S-Trace, a tool for building causal paths that does not modify applications and that uses specific features of SDN to build precise causal paths. To build a causal path, S-Trace intercepts library function calls to correlate communication events between processes and uses record and replay techniques in SDN to correlate network events. S-Trace was evaluated using a benchmark (TPC-W) and an application that emulates the behavior of multi-tier applications that was instrumented to validate the causal paths constructed by S-Trace. The experimental results show that S-Trace builds correct causal paths at the cost of a small overhead associated with some library function calls.*

Resumo. *Determinar corretamente elementos de hardware e software utilizados no processamento de requisições de uma aplicação e agrupá-los em um conjunto, denominado caminho causal, que exponha parâmetros relevantes, como tempo de processamento e atrasos, e que possam explicar o comportamento da aplicação é uma tarefa extremamente desafiadora. Diversas ferramentas de construção de caminhos causais foram concebidas para a arquitetura atual da Internet, mas nenhuma delas até então explora funcionalidades oferecidas pelas redes definidas por software (SDN* - Software Defined Networks*). Este trabalho propõe S-Trace, uma ferramenta de construção de caminhos causais que não exige instrumentação de aplicações e que utiliza funcionalidades específicas de SDN para construir caminhos causais precisos. Para construir um caminho causal, S-Trace intercepta chamadas de função de bibliotecas para correlacionar eventos de comunicação entre processos e utiliza técnicas de gravação e reprodução de tráfego de SDN para correlacionar eventos de rede. S-Trace foi avaliada utilizando um benchmark (TPC-W) e uma aplicação que emula o comportamento de aplicações multicamadas e que foi totalmente instrumentada para validar os caminhos causais gerados. Os resultados experimentais mostram que S-Trace gera caminhos causais corretos ao custo de um pequeno overhead em algumas chamadas de função de biblioteca.*

1. Introdução

Centros de processamento de dados modernos hospedam milhares de servidores para prover serviços de Internet de larga escala, sendo que cada servidor, normalmente, executa um conjunto diverso de aplicações de rede. Grandes empresas, como a HP (*Hewlett-Packard*), possuem mais de 6000 diferentes aplicações em uso em um único centro de dados [Scheck 2008]. As interações nesse ambiente — sejam entre aplicações ou entre dispositivos ou entre aplicações e dispositivos — se tornam extremamente complexas à medida que novos dispositivos ou aplicações são adicionadas ao ambiente. Balanceadores de carga, servidores proxy e dispositivos de NAT são fatores complicadores para se determinar e compreender as interações entre as aplicações, pois modificam o que seria o fluxo normal de uma conexão segundo o princípio fim-a-fim [Saltzer et al. 1984].

As interações entre as aplicações podem revelar dependências tanto diretas quanto indiretas. Um exemplo simples é uma requisição de um navegador a um servidor *Web*. Existe uma dependência direta entre a requisição do cliente e a resposta do servidor, mas podem haver também várias dependências indiretas, como: resolução de nomes via DNS, tradução de endereço por dispositivos NAT ou servidores de *proxy*, acesso a um servidor de banco de dados ou de autenticação pelo servidor *Web*, etc. Além das dependências de serviços e aplicações, há várias dependências de elementos de hardware, como *switches*, roteadores e enlaces.

Os elementos envolvidos no processamento de uma requisição, tanto de hardware como de software, podem ser agrupados em um conjunto denominado caminho causal. Em situações de falhas ou instabilidades, um caminho causal bem definido auxilia na identificação dos componentes que estão apresentando problemas. Várias ferramentas, entre elas Sherlock [Bahl et al. 2007], X-Trace [Fonseca et al. 2007] e Nemo [Júnior et al. 2013], utilizam o conceito de caminho causal para determinar qual elemento do conjunto está afetando o processamento da requisição. Essas ferramentas são divididas em dois grandes grupos: as intrusivas e as não intrusivas. A principal diferença entre os grupos é que nas intrusivas as aplicações precisam ser modificadas para se inserir informações de controle, enquanto nas não intrusivas não há essa necessidade.

A maioria das ferramentas de construção de caminhos causais até então conhecidas foram concebidas para a arquitetura atual da Internet. Essa arquitetura tem sido fortemente criticada por ser de difícil alteração e evolução, sendo inclusive rotulada de ossificada. O novo paradigma de *Rede Definida por Software* (SDN – *Software Defined Network*) foi proposto para contornar os problemas da arquitetura atual da Internet. SDN proporciona a dissociação entre o plano de controle e o plano de dados, permitindo que elementos externos de software exerçam funções do plano de controle e alterem o comportamento do plano de dados. A partir dessa dissociação, a arquitetura das SDNs permite que a inteligência da rede seja concentrada lógicamente em um único ponto. Esse ponto possui uma visão global da rede e simplifica significativamente o gerenciamento da rede.

Este trabalho propõe S-Trace, uma ferramenta não intrusiva de construção de caminhos causais que explora aspectos positivos das ferramentas de construção de caminhos causais intrusivas e não intrusivas, além da separação de planos fornecida por SDN. Para construir um caminho causal, S-Trace intercepta

chamadas de função de bibliotecas para correlacionar os eventos de comunicação inter-processo (IPC – *Inter-Process Communication*) utilizados ao processar uma requisição. S-Trace também utiliza técnicas de gravação e reprodução de tráfego de SDN para correlacionar eventos de rede. Os pacotes de dados são capturados e enviados para um sistema de armazenamento (*data store*) para reprodução posterior em um ambiente virtualizado. A reprodução do tráfego de rede é utilizada para aprimorar caminhos causais construídos com informações apenas dos extremos das conexões, pois permite correlacionar eventos de rede que possam alterar informações dos pacotes, como, por exemplo, tradução de endereços por dispostivos balanceadores de carga ou NAT.

Para a avaliação de S-Trace, foram utilizados um *benchmark* e uma aplicação representativa dos principais modelos de implementação de aplicações distribuídas. O *benchmark* TPC-W [Menasce 2002] foi utilizado para emular aplicações de comércio eletrônico (*e-Commerce*). A aplicação emula o comportamento de uma aplicação multicamadas (*multi-tier*) que realiza comunicações entre múltiplos processos e *threads* em diversos níveis de profundidade. Ela utiliza diferentes mecanismos de comunicação entre processos e foi instrumentada para mostrar o caminho causal exato de uma requisição. O objetivo principal dessa aplicação é comparar os resultados gerados por S-Trace com os caminhos causais exatos gerados pela instrumentação da aplicação. Os resultados da avaliação mostram que S-Trace é capaz de construir caminhos causais precisos, mesmo na presença de falhas e de dispositivos NAT que modificam os endereços IP das conexões utilizadas nas requisições, fatores que dificultam significativamente a construção de caminhos causais.

O restante deste trabalho está organizado como segue. A Seção 2 apresenta os trabalhos relacionados. S-Trace é descrita em detalhes na Seção 3. A Seção 4 demonstra os resultados dos experimentos e o impacto da utilização de S-Trace, e a Seção 5 conclui este trabalho.

2. Trabalhos Relacionados

Várias ferramentas foram propostas na literatura para construção de caminhos causais como mecanismo auxiliar de detecção de falhas e sobrecargas, mas todas elas foram concebidas para a arquitetura atual da Internet e não exploram possíveis funcionalidades fornecidas pela separação dos planos de dados e de controle em SDN. Uma das principais características de S-Trace é a utilização de serviços de gravação e reprodução de tráfego em SDN para correlação de eventos de rede. Recentemente, a ferramenta OFRewind [Wundsam et al. 2011] utilizou esses serviços em uma implementação relativamente simples devido à presença de um controlador logicamente centralizado. Entretanto, eles são de difícil implementação na arquitetura atual da Internet, principalmente pela natureza distribuída dos protocolos de roteamento.

Ferramentas intrusivas, como Pinpoint [Chen et al. 2004] e X-Trace [Fonseca et al. 2007], modificam as aplicações para que identificadores únicos sejam transportados em todo o processamento de uma requisição e para garantir a identificação de todos eventos que sejam provenientes de uma mesma requisição. X-Trace se destaca por ser a ferramenta mais intrusiva da literatura, pois, além de instrumentar aplicações, instrumenta toda a pilha de protocolo das estações de trabalho e servidores para prover visibilidade total no processamente de requisições. PIP [Reynolds et al. 2006], por outro lado, é uma ferramenta que, ao invés de modificar

aplicações, instrumenta *frameworks* para observar recursos utilizados pelas aplicações e suportar aplicações legadas sem a necessidade de modificações. vPath [Tak et al. 2009] expandiu o conceito de instrumentação para ambientes virtualizados para capturar comunicações entre *threads* de uma mesma máquina virtual.

Outro grupo importante de ferramentas são as não intrusivas que não modificam aplicações, protocolos ou *frameworks* para construir caminhos causais. BorderPatrol [Koskinen and Jannotti 2008], por exemplo, monitora as mensagens de aplicação trocadas entre processos e utiliza processadores de protocolo específicos para analisar as mensagens e correlacionar eventos que permitam a construção de caminhos causais. Sherlock [Bahl et al. 2007], por outro lado, modela as interações em um ambiente distribuído utilizando Redes Bayesianas e aplica tempo de coocorrência para determinar a relação entre eventos e construir os caminhos. Nemo [Júnior et al. 2013] reduz o tempo de inferência de Sherlock explorando conhecimento específico do domínio do problema e uma propriedade teórica de Redes Bayesianas.

Magpie [Barham et al. 2004] é a ferramenta que possui maior similaridade com S-Trace. Ambas monitoram as mensagens de aplicação e, a partir de esquemas de eventos, realizam as combinações necessárias para correlacionar os eventos e construir os caminhos causais. Contudo, o esquema de eventos de S-Trace é realizado de forma automática, diferentemente de Magpie em que administradores devem informar manualmente o esquema de cada aplicação. Além disso, S-Trace associa cabeçalhos dos pacotes com eventos de E/S para obter caminhos causais com granularidade fina, semelhante à X-Trace, mas sem realizar instrumentação na pilha de protocolo e nem nas aplicações.

3. S-Trace

A ferramenta S-Trace foi desenvolvida com intuito de construir e visualizar caminhos causais de requisições de aplicações distribuídas. Ela explora informações provenientes de uma SDN para identificar os componentes de hardware e software utilizados no processamento de uma requisição e para corrigir e aprimorar caminhos causais construídos com as informações dos extremos das conexões. S-Trace utiliza os conceitos de agente e gerente e seu fluxo geral está ilustrado na Figura 1.

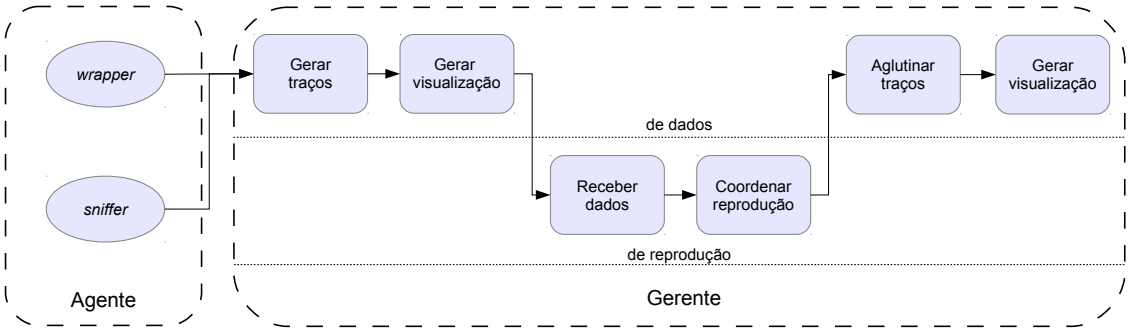

Figura 1. Fluxo geral da ferramenta S-Trace.

Os agentes, que são instalados em máquinas a serem monitoradas (estações de trabalho ou servidores), são divididos em dois componentes: *wrapper* e *sniffer*. O *wrapper* monitora as chamadas de função utilizadas durante a execução das aplicações e

o *sniffer* captura pacotes enviados e recebidos pelas estações monitoradas. Ambos enviam informações do monitoramento para o gerente de dados por meio de canais confiáveis e exclusivos, utilizando o protocolo TCP.

O gerente da ferramenta S-Trace é dividido em gerente de dados e gerente de reprodução. O gerente de dados é responsável por receber informações enviadas pelos agentes, construir — em um estágio inicial — os caminhos causais e gerar um arquivo com os dados para a visualização dos caminhos. O gerente de reprodução é responsável por armazenar e reproduzir o comportamento da rede original e fornecer informações ao gerente de dados para refinamento dos caminhos causais. A reprodução do comportamento da rede utiliza as mesmas técnicas de gravação e reprodução do tráfego de rede da ferramenta OFRewind [Wundsam et al. 2011].

3.1. Componente *wrapper*

O componente *wrapper* é encapsulado como uma biblioteca compartilhada, denominada *libwrapper*, para monitorar chamadas de função que as aplicações realizam durante a execução, capturar informações relevantes desse monitoramento e enviar essas informações para o gerente de dados. As principais informações são: LWPID, PID, PPID, argumentos da função, caminho absoluto de arquivos e carimbo de tempo. O *wrapper*, na realidade, realiza uma interposição de biblioteca e intercepta as chamadas para que funções da *libwrapper* sejam invocadas ao invés das funções da biblioteca original, sem que haja necessidade de instrumentação das aplicações. A interposição de biblioteca faz com que a biblioteca compartilhada seja carregada antes de qualquer outra. Para que isso aconteça, o carregador do sistema operacional deve ser informado da existência da *libwrapper* e deve carregá-la junto com a aplicação que será monitorada. Este último passo é facil de ser realizado, pois basta que a variável de ambiente `LD_PRELOAD` seja inicializada com o caminho completo da *libwrapper*.

As funções da *libwrapper* são apenas invólucros que realizam pequenos processamentos e invocam as funções originais. Com isso, o comportamento das funções e, consequentemente, das aplicações, não é alterado pelo componente *wrapper*. As funções da *libwrapper* que foram implementadas neste trabalho estão listadas na Tabela 1.

Tabela 1. Funções da biblioteca *libwrapper*.

`read`	`recv`	`write`	`send`
`socket`	`bind`	`listen`	`accept`
`connect`	`close`	`fopen`	`fclose`
`dup`	`dup2`	`fork`	`msgget`
`msgsnd`	`msgrcv`	`shmget`	`shmat`
`semget`	`semop`	`pipe`	`pipe2`
`mkfifo`	`readv`	`writev`	`sendfile`

As funções implementadas pelo componente *wrapper* possuem fluxo predefinido, que é dividido em três fases. A primeira fase consiste em invocar a função original da biblioteca correta com os mesmos parâmetros, armazenar o valor de retorno da função original e retorná-lo pela função da *libwrapper*. Dessa forma, o comportamento da aplicação não é modificado. A segunda fase consiste em extrair informações referentes à aplicação – como identificadores de *threads*, de processos e de descritores de mecanismos

de IPC – que auxiliam S-Trace a construir os caminhos causais das requisições. A última fase consiste em enviar essas informações para o gerente de dados, que será descrito na Seção 3.3.

O componente *wrapper* utiliza processadores de protocolo para interpretar as mensagens trocadas entre as aplicações e extrair informações como delimitadores de mensagens de aplicação e atributos específicos de protocolo. O conhecimento necessário para o desenvolvimento de processadores é específico de protocolo e não de aplicação. Neste trabalho, foram utilizados processadores dos protocolos HTTP e MySQL. Como exemplo de atributo específico de protocolo, pode-se citar o caso em que cliente e servidor decidem manter conexões persistentes. O *wrapper* também extrai informações da função que está sendo executada como: valor de retorno; LWPID, PID e PPID da *thread* que invocou a função; descritor de arquivo; etc. Com essas informações, o gerente de dados produz caminhos causais iniciais que serão aprimorados posteriormente com informações fornecidas pelo gerente de reprodução.

A utilização do componente *wrapper* não modifica as aplicações e nem as funções originais. O componente modifica apenas o fluxo de dados das aplicações com um novo estágio adicionado ao caminho do fluxo. Esse estágio captura e envia informações sobre a aplicação e a função para o gerente de dados e em seguida o fluxo prossegue normalmente. A adição desse novo estágio não afeta significativamente o desempenho da aplicação, pois são operações rápidas e curtas. A Seção 4 avalia o *overhead* introduzido pelo *wrapper* em diferentes chamadas de função.

3.2. Componente *sniffer*

O componente *sniffer* monitora todos os pacotes enviados e recebidos pela estação de trabalho ou servidor e envia os cabeçalhos dos pacotes para o gerente de dados. Esse componente é inicializado como um *daemon* e não exige interação com os usuários. Apesar de capturar os pacotes, o *sniffer* não influencia no tráfego de rede, muito menos no comportamento das aplicações, pois ele realiza uma cópia dos pacotes para não alterar o fluxo normal da conexão.

Para realizar o monitoramento, o componente *sniffer* captura pacotes utilizando a biblioteca *libpcap*, disponível para os principais sistemas operacionais atuais. Para cada pacote recebido ou enviado pela estação monitorada, uma função de *callback* do *sniffer* é invocada pela *libpcap*.

As informações de tamanho dos cabeçalhos do pacote, o carimbo de tempo do momento em que o pacote trafegou pela estação de trabalho ou servidor e os cabeçalhos dos pacotes são encapsulados em uma estrutura de dados. O *sniffer* envia essas informações encapsuladas por meio de um canal exclusivo e confiável para o gerente de dados, utilizando o protocolo TCP. Contudo, esse envio é realizado, normalmente, por meio da mesma interface de rede que é monitorada pelo componente *sniffer*. Dessa forma, há necessidade de um filtro especial no monitoramento dos pacotes para que os pacotes com destino e origem ao gerente de dados não sejam capturados. Se esse filtro não fosse utilizado, haveria um laço infinito no monitoramento dos pacotes.

3.3. Gerente de Dados

O gerente de dados é um dos principais componentes de S-Trace. Ele recebe e organiza os dados enviados pelos agentes, gera caminhos causais iniciais, produz uma visualização

prévia dos caminhos, troca informações com o gerente de reprodução, aperfeiçoa os caminhos causais e, por fim, constrói os caminhos causais finais.

O gerente de dados cria duas listas, uma para os *wrappers* e outra para os *sniffers*. Os dados recebidos são adicionados em suas respectivas listas ordenados pelos carimbos de tempo. Periodicamente, o gerente de dados inicia o processo de construção de caminhos causais a partir da lista de nós *wrappers*. Inicialmente, esse processo busca por nós que representem início e fim de uma conexão, pois eles representam delimitadores em um caminho causal. O final da conexão pode ser obtido por meio de funções que representem término de um fluxo ou de nós que utilizaram o processador de protocolo que interpretou e delimitou as mensagens de aplicação. Os nós recebidos entre os nós inicial e final são analisados para se verificar se possuem alguma ligação com o caminho. Em caso positivo, eles são adicionados ao caminho.

As mensagens trocadas pelos agentes e gerente são do tipo assíncronas e diversas mensagens podem ser encaminhadas a qualquer momento, inclusive durante o processo de construção dos caminhos causais. Para que a construção dos caminhos não interfira no recebimento de dados, o gerente de dados possui dois ponteiros extras: um para a lista dos *wrappers* e outro para os *sniffers*. Quando o processo é iniciado, há uma troca de ponteiros para que as listas não sejam modificadas durante o processo de construção e para que os recebimentos não sejam afetados.

A função que verifica se um nó n possui algum tipo de ligação com um caminho causal C é definida da seguinte maneira. Primeiramente, o conjunto S de LWPID baseado nos nós de C é definido. Caso o valor do campo LWPID de n pertença a esse conjunto, n possui ligação com C. Caso contrário, a função de ligação verifica se n possui algum mecanismo de IPC com algum nó de C. Os tipos de IPC verificados são: arquivo, *pipe*, *socket*, semáforo, fila de mensagens e memória compartilhada. Basicamente, a verificação desses mecanismos busca por identificadores comuns entre os nós para realizar a correlação e, assim, adicioná-los ao caminho causal. A função é utilizada pelo processo de construção de caminhos causais e é invocada para todos os nós *wrappers* que não foram adicionados a nenhum caminho causal. S-Trace assume que *threads* que compartilham blocos de memória ou dados estão processando a mesma requisição, o que nem sempre é verdadeiro e pode gerar falsos positivos. Falsos positivos podem ser minimizados ou até eliminados por análise de correlação temporal dos eventos. Entretanto, esse assunto não será abordado neste trabalho e será objeto de trabalho futuro.

Após as uniões dos caminhos causais, o gerente de dados inicia o processo de associação entre as operações realizadas pelas aplicações (*wrapper*) e os cabeçalhos dos pacotes trafegados pelas estações de trabalho ou servidores (*sniffer*), realizando a união entre os nós dos caminhos causais e os cabeçalhos dos pacotes. Essa operação é realizada por meio dos números de sequência utilizados pelo protocolo TCP. A associação é dividida em duas partes, a primeira coleta todas operações de entrada e todos os cabeçalhos de entrada e a outra parte coleta os dados de saída. Essa divisão é realizada pois os nós *wrappers* estão ordenados pelo carimbo de tempo, garantindo a ordem de invocação das funções. Entretanto, o carimbo de tempo dos nós *sniffers* não é um forte parâmetro para realizar a associação, pois não há garantia da ordem de saída ou de entrada dos pacotes por meio do tempo.

Após o processo de construção dos caminhos causais, enriquecidos com as informações dos cabeçalhos capturados, o gerente de dados gera um arquivo de visualização do tipo `dot` que contém grafos que representam todos os caminhos causais construídos. Cada nó representa alguma chamada de função que a aplicação invocou. Caso a chamada produza pacotes de rede, novos nós são criados para representar os pacotes. Após a construção do arquivo `.dot`, o gerente percorre cada caminho procurando por inícios de conexão que não foram unidos durante o processo de construção dos caminhos causais. Se existirem nós de conexão que não foram unidos, esses nós são informados ao gerente de reprodução, para que, posteriormente, caminhos causais que aparentemente não possuem relação possam ser unidos. O intuito dessa notificação é aprimorar os caminhos causais construídos pelo gerente de dados com informações provenientes da reprodução do tráfego de rede, por exemplo.

Em um primeiro momento, a notificação dos inícios de conexão para o gerente de reprodução não possui nenhum motivo, pois se as informações fornecidas pelos agentes não forem suficientes, aparentemente não há ligação entre os caminhos. Entretanto, caso alguma técnica de modificação de campos do pacote seja utilizada pela rede, por exemplo, NAT, o gerente de dados e os agentes da ferramenta não saberiam dessa utilização. Porém, o gerente de reprodução possui essa informação que é fornecida pela centralização lógica da inteligência da rede fornecida por SDN. Essa informação é essencial para que o gerente de dados realize a ligação entre os caminhos causais. A Figura 2 ilustra um exemplo de dois caminhos causais aparentemente sem nenhum tipo de ligação, porém, há uma regra em um comutador que realiza a modificação dos endereços IP dos pacotes, fazendo com que os caminhos sejam ligados.

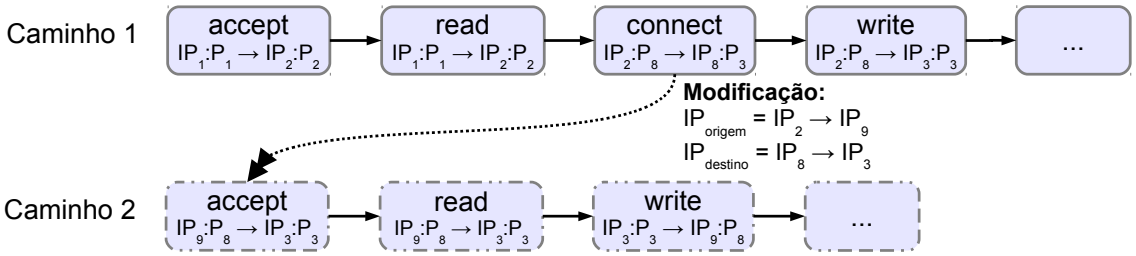

Figura 2. Exemplo de dois caminhos causais sem ligação aparente entre os caminhos. Durante o tráfego, modificações nos endereços IP dos pacotes são realizadas, apontando ligação entre os caminhos que não é observada pelos extremos das conexões.

A simples utilização da técnica NAT faz com que um único caminho causal seja particionado, erroneamente, em duas requisições diferentes. Esse fato acontece pois não há nenhum tipo de união, nesse estágio, entre os caminhos. Entretanto, com a utilização do gerente de reprodução, essa situação é contornada, e os caminhos causais são unidos. Observando a Figura 2, sem a informação das modificações, os Caminhos 1 e 2 são totalmente distintos, o que é uma falsa dedução. As informações de modificação não são visíveis para os agentes e para o gerente de dados, apenas o gerente de reprodução possui essa informação, pois ele tem acesso às regras de fluxo instaladas nos comutadores de rede.

O gerente de reprodução, após receber os inícios de conexão, realiza diversos

procedimentos (que são explicados detalhadamente na próxima seção) e envia ao gerente de dados informações de instalações de regras nos comutadores de rede referentes aos dados recebidos. Essas informações são utilizadas pelo gerente de dados para realizar uniões entre os caminhos causais que, aparentemente, não possuiam nenhum tipo de ligação. Dessa forma, o gerente de dados novamente invoca o processo de união entre os caminhos causais juntamente com os dados recebidos pelo gerente de reprodução. Feito isso, o processo de visualização também é invocado novamente, gerando um novo arquivo `.dot` final, com os caminhos causais unidos e bem definidos.

3.3.1. Gerente de Reprodução

O gerente de reprodução possui o objetivo de gravar o trafégo de rede de modo escalável, consistente e controlado, receber informações sobre os caminhos causais fornecidos pelo gerente de dados, coordenar a reprodução do tráfego armazenado e informar ao gerente de dados as regras de fluxo que influenciam nos caminhos causais e que foram instaladas nos comutadores OpenFlow. O gerente intercepta as regras de fluxo, observando todas as ações tomadas pelos comutadores em determinados fluxos.

O núcleo do gerente de reprodução é extraído da ferramenta OFRewind [Wundsam et al. 2011] que consolida as técnicas de gravação e reprodução de tráfego de rede em um ambiente programável utilizando o protocolo OpenFlow. O gerente de reprodução explora essas técnicas para notificar qual foi o comportamento da rede durante o processo de gravação, ou seja, quais foram as ações tomadas pelos comutadores de rede.

O gerente de reprodução conta com dois módulos: um é responsável por receber e enviar mensagens para o gerente de dados e o outro é responsável por interceptar regras de fluxo a serem instaladas nos comutadores. O módulo que recebe e envia dados para o gerente de dados deve receber os dados e os encapsular em estruturas de dados de tal forma que, o módulo de interceptação seja capaz de verificar se alguma regra de fluxo influencia nos dados recebidos. De forma semelhante ao gerente de dados, esse módulo possui uma lista para receber os dados e ordená-los pelo carimbo de tempo.

A atuação do módulo que intercepta as regras de fluxo não afeta o comportamento da rede, pois as regras não são alteradas, apenas informações de modificação de campos de pacotes são extraídas das regras. As regras observadas são apenas aquelas que possuem alguma ligação com os nós *wrappers* que indicam inícios de conexão que não foram unidos durante o processo de construção de caminhos causais do gerente de dados. Essa ligação é fornecida pelos endereços IP e portas de comunicação da regra e do nó *wrapper*.

O gerente de reprodução é um componente fundamental para que o gerente de dados produza caminhos causais confiáveis e bem definidos, pois a reprodução do comportamento da rede revela relacionamentos que não podem ser capturados nos extremos de uma conexão. Essa contribuição é realizada, principalmente, pela captura das ações das regras de fluxo instaladas nos comutadores, fazendo que modificações nos pacotes de redes sejam observadas e notificadas para o gerente de dados por meio da interceptação das mensagens do plano de controle. Além disso, como todas as regras são instaladas em um momento pós-gravação, o comportamento do tráfego de rede não é alterado. Essa é uma característica importante da ferramenta OFRewind que é mantida pelo gerente de reprodução.

A Figura 3 ilustra o caminho causal construído pela combinação dos dois caminhos ilustrados na Figura 2. O gerente de reprodução observa as regras de fluxo que indicam as modificações ilustradas e informa ao gerente de dados sobre as modificações. Ao receber essa informação, o gerente de dados invoca novamente o processo de construção e os dois caminhos são unidos, formando um único caminho causal.

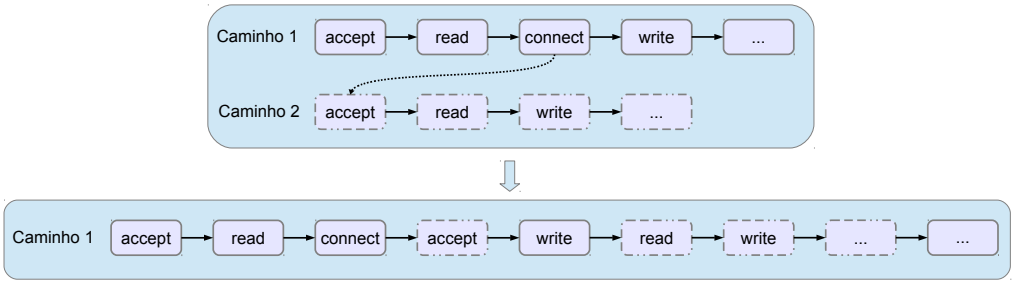

Figura 3. Exemplo de união entre caminhos causais. A linha tracejada indica que os dois nós possuem uma ligação que só foi possível identificá-la com a reprodução do tráfego de rede, observando as regras de fluxo.

4. Avaliação Experimental

S-Trace foi avaliada utilizando o *benchmark* (TPC-W) e uma aplicação (n-Tier) que emula o comportamento de aplicações multicamadas e que foi totalmente instrumentada para validar os caminhos causais gerados. A ferramenta Mininet [Lantz et al. 2010] foi utilizada para emular a rede, pois ela possibilita a criação de diferentes redes virtuais em um único computador, além de fornecer a rápida prototipação de diferentes redes virtuais baseadas no protocolo OpenFlow [McKeown et al. 2008]. A topologia de rede utilizada nesta avaliação foi composta por 13 *hosts* e 4 comutadores. Dentre os *hosts*, 8 são instâncias de n-Tier, 3 realizam requisições e 2 executam TPC-W.

O objetivo da avaliação experimental é mostrar que S-Trace captura de maneira confiável os elementos utilizados no processamento de uma requisição. Algumas falhas foram produzidas para demonstrar que os caminhos causais construídos são capazes de, além de auxiliar a compreender como as aplicações se comportam ao processar uma requisição, contribuir para detecção de falhas.

4.1. Aplicação n-Tier

A aplicação n-Tier foi desenvolvida para emular o comportamento de aplicações que utilizam múltiplas camadas, aspecto comum em grande parte das aplicações distribuídas usuais. n-Tier foi totalmente instrumentada para fornecer todos os eventos realizados pelas *threads* no processamento das requisições. Com isso, é possível comparar os caminhos causais construídos por S-Trace e os eventos fornecidos pela instrumentação. Cada instância de n-Tier é denominada réplica que possui um *pool* de *thread* e uma *thread* principal que recebe as requisições e as encaminha para uma das *threads* do *pool*, ou seja, a primeira camada de processamento.

A primeira camada de processamento, ao receber uma requisição, deve decidir se processa a requisição ou se realiza um encaminhamento. Caso a decisão seja não processar a requisição, então essa camada deve decidir se encaminha a requisição para

uma nova camada ou para uma outra instância da aplicação. Ambas decisões são determinadas aleatoriamente. A cada requisição, a *thread* decide, com base em uma variável aleatória com distribuição uniforme, se a requisição deve ser repassada ou não. A probabilidade de repasse é um parâmetro de execução da aplicação n-Tier.

O mecanismo de IPC `socket` é utilizado para o encaminhamento entre as réplicas, visto que, cada *host* produzido por Mininet possui um endereço IP próprio e executa uma instância da aplicação n-Tier. Os encaminhamentos entre as camadas de processamento podem ser realizados utilizando arquivo, *pipe*, fila de mensagens ou memória compartilhada.

Para mostrar a interação de S-Trace em uma SDN, a técnica de NAT foi implementada nos comutadores de tal forma que quando réplicas da aplicação n-Tier que estão na mesma sub-rede trocam informações entre si, as conexões são realizadas por meio dos endereços externos não pertencentes ao escopo de endereçamento da sub-rede. Dessa forma, quando os comutadores recebem pacotes de rede com esses endereços externos de destino, eles são instruídos a modificar o campo destino IP para o endereço da réplica correta. Além disso, o campo origem IP também é modificado para o endereço externo correto nos pacotes com as respostas das réplicas.

A Tabela 2 ilustra a quantidade de cada um dos mecanismos de IPC utilizado pela aplicação n-Tier em diferentes quantidades de requisições. O valor da somatória da quantidade de cada mecanismo de IPC é superior ao valor da quantidade de requisições porque vários mecanismos de IPC podem ser utilizados no processamento de uma única requisição.

Tabela 2. Quantidade de cada mecanismo de IPC utilizado pela aplicação n-Tier para processar diversas requisições.

	Mecanismos de IPC				
Requisições	Arquivo	*Pipe*	*Socket*	Fila de mensagens	Memória compartilhada
25	25	17	17	11	9
100	39	35	24	26	19
200	63	59	43	48	49
300	88	83	59	72	70
400	111	99	83	88	92
500	127	126	104	111	106
1000	218	239	228	243	199
10000	2012	2017	1634	2054	1964

4.2. TPC-W

TPC-W [Menasce 2002] é um *benchmark* para transações *Web* do tipo *e-Commerce* que utiliza o servidor *Web* Tomcat e o servidor de banco de dados MySQL. TPC-W realiza diversas requisições ao servidor *Web* para comparar o desempenho do servidor ao processar as requisições. As requisições são comuns a vários tipos de sistema *e-Commerce*, como: navegação pela lista de produtos, compra, venda, etc. Várias requisições são disparadas para simular a navegação em um sistema de comércio eletrônico.

Para ilustrar um dos benefícios de caminhos causais bem definidos, durante o experimento com TPC-W, o servidor de banco de dados foi desativado. Com o caminho

causal, foi possível observar que a *thread* que processa a requisição tenta iniciar conexão com o banco por diversas vezes sem sucesso. Com isso, também foi possível observar que o Tomcat não utilizou uma conexão persistente com o banco e, também, que uma única *thread* foi responsável por receber as conexões e repassar as requisições para as demais *threads*. Essas características são determinadas em arquivos de configuração na execução do servidor *Web*. Entretanto, com os caminhos construídos por S-Trace, é possível revelar esses aspectos sem examinar esses arquivos.

4.3. Resultados

Os caminhos causais construídos por S-Trace são ilustrados em forma de grafos em que cada nó representa uma função que foi invocada por uma *thread* de uma aplicação e também um pacote que trafegou na rede proveniente da chamada da função. Os nós em nível horizontal representam as chamadas de função e possuem as seguintes informações: endereço IP do agente que enviou os dados para o gerente, carimbo de tempo, identificação da função invocada, LWPID e PID. Os nós em nível vertical representam os pacotes de rede que trafegaram devido à função invocada. Estes nós possuem as seguintes informações: carimbo de tempo, *flags* do protocolo TCP, número de sequência do pacote e indicação de erro, caso exista.

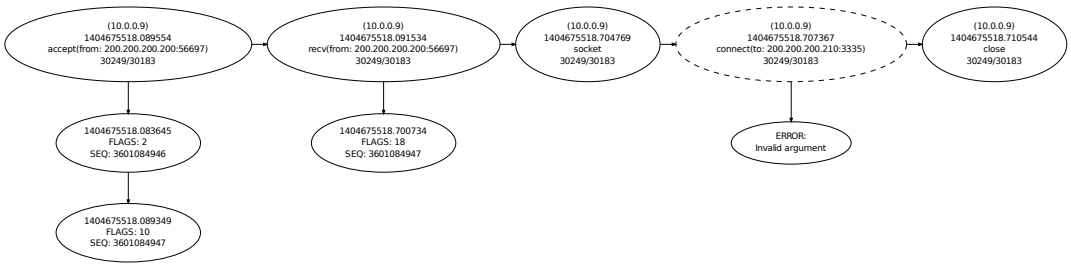

Figura 4. Caminho causal de uma requisição de n-Tier construído por S-Trace. Durante a execução, a conexão com destino `200.200.200.210:3335` não foi estabelecida e por isso, o nó foi contornado com linha tracejada indicando algum erro. Alguns nós do caminho foram omitidos para facilitar a visualização.

A Figura 4 ilustra um caminho causal construído por S-Trace. O tamanho do caminho foi reduzido para facilitar a visualização. Esse caminho causal é construído quando uma instância da aplicação n-Tier recebe a requisição e tenta conexão com a réplica com o endereço `200.200.200.210:3335`. Entretanto, essa conexão não foi estabecida retornando o erro `Invalid Argument`, neste caso, S-Trace faz com que este nó seja contornado com linha tracejada indicando o erro.

Foram realizadas 100 requisições para instâncias da aplicação n-Tier e em todas elas, S-Trace identificou todos os relacionamentos entre as *threads* e os mecanismos de IPC utilizados. Com base em uma variável aleatória com distribuição uniforme, uma réplica de n-Tier é escolhida como destino inicial para cada requisição. A comparação entre os caminhos causais construídos por S-Trace e os relacionamentos de n-Tier foi possível pois n-Tier foi instrumentada para fornecer os eventos realizados pelas *threads* no processamento das requisições.

O componente *wrapper* do agente da ferramenta S-Trace também foi avaliado para

demonstrar o impacto, em tempo de execução, da utilização do agente nas aplicações, mais especificamente, a utilização da biblioteca *libwrapper*. Para isso, uma aplicação cliente/servidor simples foi desenvolvida em que o cliente realiza uma conexão com o servidor, envia um *buffer* de mensagem e encerra a conexão. Além disso, durante a execução dessa aplicação, as funções `fork` e `fopen` foram invocadas.

Na avaliação de tempo de execução, é capturado o carimbo de tempo antes e depois da chamada das funções `send`, `fopen` e `fork`. O tamanho do *buffer* da mensagem a ser enviada variou entre 100, 1500 e 10000 bytes. Para cada tamanho de *buffer* utilizado na função `send` e para as funções `fork` e `fopen`, foram executadas 500 instâncias do cliente, produzindo 500 requisições. A Tabela 3 ilustra os resultados obtidos e os intervalos de confiança de 95% do tempo médio. Os resultados numéricos mostram que os atrasos inseridos pelo *wrapper* são desprezíveis.

Tabela 3. Tempo de execução médio de chamadas de função (ms).

	sem *wrapper* (I.C. 95%)	com *wrapper* (I.C. 95%)	*overhead*
send (100 bytes)	0,068±0,005	0,144±0,012	∼0,076
send (1500 bytes)	0,103±0,009	0,174±0,015	∼0,071
send (10000 bytes)	0,150±0,013	0,221±0,019	∼0,071
fopen	0,326±0,028	0,401±0,035	∼0,075
fork	0,477±0,041	0,553±0,048	∼0,076

Por limitações de espaço, esta seção apresenta um conjunto reduzido dos resultados obtidos com S-Trace. Outros experimentos, como: injeção de atrasos, réplica inexistente, encerramento do servidor de banco de dados, etc., foram realizados e podem ser consultados em [Carvalho 2014]. A dissertação [Carvalho 2014] também lista em seu apêndice a implementação das funções *wrappers* de S-Trace.

5. Conclusão

S-Trace é uma ferramenta de construção de caminhos causais que não exige instrumentação de aplicações, protocolos ou *frameworks* e que utiliza informações de fluxos de dados em um controlador SDN e de chamadas de funções de IPC da `libc` para construir caminhos causais. Os resultados experimentais são promissores, pois a ferramenta foi capaz de reconstruir caminhos causais precisos em execuções controladas de uma aplicação multicamadas que foi totalmente instrumentada. Entretanto, a ferramenta ainda está em um estágio inicial e mostra algumas direções para investigações e trabalhos futuros, como: análise de falsos positivos; implementação de novos processadores de protocolo de aplicação; avaliação do melhor local (*switch* ou *host*) para captura de pacotes; interface gráfica para visualização dos resultados; e informações que o controlador SDN pode fornecer para melhorar a precisão dos caminhos causais.

Referências

Bahl, P., Chandra, R., Greenberg, A., Kandula, S., Maltz, D. A., and Zhang, M. (2007). Towards Highly Reliable Enterprise Network Services via Inference of Multi-level Dependencies. In *Proceedings of the 2007 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'07)*.

Barham, P., Donnelly, A., Isaacs, R., and Mortier, R. (2004). Using Magpie for Request Extraction and Workload Modelling. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation (OSDI'04)*.

Carvalho, F. B. (2014). Construção de Caminhos Causais em Redes Definidas por Software. Faculdade de Computação (FACOM), UFMS. `http://ndsg.facom.ufms.br/S-Trace.pdf`.

Chen, M. Y., Accardi, A., Kiciman, E., Lloyd, J., Patterson, D., Fox, A., and Brewer, E. (2004). Path-Based Failure and Evolution Management. In *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI'04)*.

Fonseca, R., Porter, G., Katz, R. H., Shenker, S., and Stoica, I. (2007). X-Trace: A Pervasive Network Tracing Framework. In *Proceedings of the 4th Symposium on Networked Systems Design and Implementation (NSDI'07)*.

Júnior, B. A. S., Carvalho, F. B., and Ferreira, R. A. (2013). Nemo: Procurando e Encontrando Anomalias em Aplicações Distribuídas. In *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC'13)*.

Koskinen, E. and Jannotti, J. (2008). BorderPatrol: Isolating Events for Black-box Tracing. In *Proceedings of the 3rd EuroSys European Conference on Computer Systems (Eurosys'08)*.

Lantz, B., Heller, B., and McKeown, N. (2010). A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets-IX)*.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review*, pages 69–74.

Menasce, D. (2002). TPC-W: a Benchmark for *e-Commerce*. *Internet Computing, IEEE*, pages 83–87.

Reynolds, P., Edwin K., C., Wiener, J. L., Mogul, J. C., Shah, M. A., and Vahdat, A. (2006). PIP: Detecting the Unexpected in Distributed Systems. In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI'06)*.

Saltzer, J. H., Reed, D. P., and Clark, D. D. (1984). End-to-end Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277–288.

Scheck, J. (2008). Taming Technology Sprawl. The Wall Street Journal. `http://online.wsj.com/article/SB120156419453723637.html`. Acessado em 13/12/2014.

Tak, B. C., Tang, C., Zhang, C., Govindan, S., Urgaonkar, B., and Chang, R. N. (2009). vPath: Precise Discovery of Request Processing Paths from Black-Box Observations of Thread and Network Activities. In *Proceedings of The 34th USENIX Annual Technical Conference (USENIX'09)*.

Wundsam, A., Levin, D., Seetharaman, S., and Feldmann, A. (2011). OFRewind: Enabling Record and Replay Troubleshooting for Networks. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference (USENIXATC'11)*.