FLAME: Uma Plataforma para Prototipagem Rápida de Ferramentas de Medição Ativa

Marcos L. Kirszenblatt, Thiago B. Cardozo, Antônio Tadeu A. Gomes, Artur Ziviani

[1]Laboratório Nacional de Computação Científica – LNCC
Av. Getúlio Vargas, 333 – 25651-075 – Petrópolis, RJ

{marcoslk,thiagoc,atagomes,ziviani}@lncc.br

Abstract. *In this paper, we present a platform for rapid prototyping of active measurement tools. The FLAME platform provides its users with basic active measurement primitives upon which sophisticated active measurement tools can be prototyped quickly, practically, and efficiently. We illustrate the platform flexibility through experiments conducted on Planet-lab and a local testbed.*

Resumo. *Neste artigo apresentamos uma plataforma para a prototipagem rápida de ferramentas de medição ativa. A plataforma FLAME oferece a seus usuários primitivas básicas de medição ativa sobre as quais ferramentas sofisticadas de medição ativa podem ser prototipadas de forma rápida, prática e eficiente. Nós ilustramos a flexibilidade da plataforma proposta por meio de experimentos realizados no Planet-lab e em um testbed.*

1. Introdução

Ferramentas de medição *ativa* de redes são baseadas no envio de *sondas* (pacotes com propósito único de efetuar medições) entre nós de origem e de destino na rede, permitindo a medição de propriedades dos caminhos entre esses nós. As ferramentas de medição ativa atualmente disponíveis e utilizadas nas principais plataformas de medição são primordialmente codificadas em alguma linguagem de programação de baixo nível, geralmente em C, de modo a reduzir a influência de sua execução na precisão dos resultados das medições [Michaut e Lepage 2005]. Como consequência, essas ferramentas apresentam: (i) um tempo de construção potencialmente alto, pelo fato de grande parte do código ter que lidar com questões transversais à funcionalidade de medição em si, como gerência de memória. tratamento de erros, saída formatada de dados, entre outras; e (ii) um nível de reuso particularmente baixo, o que é observável a partir do código-fonte das principais ferramentas de medição com código aberto disponíveis na literatura.

Em [Ziviani et al. 2010] apresentamos uma discussão detalhada sobre as principais plataformas de medição ativa existentes. Essas plataformas são em geral baseadas na implantação distribuída, em alguns nós da rede, de agentes de medição que emitem e recebem sondas em resposta a comandos de um gerente central. Em algumas plataformas esses agentes publicam os dados obtidos com as medições de forma padronizada em um repositório centralizado, simplificando a gerência e análise dos mesmos. Contudo, essas plataformas usam ferramentas de medição pré-existentes para efetuar experimentos de medição. Assim, não é coberta por nenhuma dessas plataformas a questão da *prototipagem rápida* de ferramentas de medição ativa, o que inviabiliza, por exemplo, o teste e a depuração práticos de técnicas inovadoras de medição.

Uma exceção ao contexto acima e que, portanto, merece menção à parte, é o sistema Scriptroute [Spring et al. 2003]. Esse sistema permite a prototipagem de ferramentas de medição ativa usando scripts Ruby. No sistema Scriptroute os scripts que descrevem os experimentos de medição são submetidos a agentes de medição implementados em C, que interpretam esses scripts e executam as medições. Contudo, a flexibilidade do sistema Scriptroute é limitada pela impossibilidade de prototipagem de ferramentas que dependam de nós de destino *cooperativos*, isto é, nós de destino que hospedem parte da ferramenta de medição (caso de ferramentas como `owping` [Boote 2010] e `pathload` [Jain e Dovrolis 2003], por exemplo). Além disso, o sistema Scriptroute não dispõe, como as plataformas de medição em geral, de um repositório centralizado em que os resultados das medições são armazenados. Desse modo, a formatação de dados de saída deve ser explicitamente codificada nos scripts Ruby que implementam as ferramentas. Esta abordagem torna os scripts um emaranhado de instruções de sondagem e de formatação de dados de saída, o que inibe maiores níveis de reuso e manutenção de código.

Neste trabalho apresentamos uma plataforma, denominada FLAME (*Flexible Lightweight Active Measurement Environment*),[1] para a prototipagem rápida de ferramentas de medição ativa. A plataforma FLAME utiliza uma arquitetura gerente-agente, similar à comumente empregada em outras plataformas de medições ativas. De modo similar ao sistema Scriproute, a plataforma FLAME oferece a seus usuários primitivas de medição ativa implementadas em C nos agentes, sobre as quais ferramentas de medição ativa podem ser prototipadas de forma rápida, prática e eficiente por meio de scripts Lua [Ierusalimschy et al. 1996]. Contudo, ao contrário do sistema Scriptroute, essas primitivas no FLAME permitem a prototipagem de ferramentas para caracterização de métricas que dependam ou independam de nós de destino cooperativos.

O restante do artigo está estruturado como se segue. A Seção 2 apresenta a arquitetura da plataforma FLAME. A Seção 3 apresenta as primitivas de medição ativa oferecidas pela plataforma. A Seção 4 apresenta os experimentos de demonstração da plataforma FLAME. Por fim, a Seção 5 apresenta considerações finais e trabalhos futuros.

2. Arquitetura da plataforma FLAME

A arquitetura da plataforma FLAME é apresentada na Figura 1. Um servidor XMPP (*Extensible Messaging and Presence Protocol*) [Saint-Andre 2004] atua como um barramento de mensagens entre os principais componentes da plataforma FLAME[2] – o *console*, o *gerente de medição*, e os *agentes de medição*.

O console é usado pelos usuários da plataforma para solicitar a execução de sessões de medição – denominadas na plataforma de *experimentos* – ao gerente de medição. O gerente de medição é responsável por iniciar os experimentos solicitados nos agentes de medição e armazenar os resultados obtidos em um repositório centralizado (um banco de dados relacional na versão atual da plataforma). O mecanismo de controle de presença oferecido pelo protocolo XMPP permite ao gerente de medição colocar os agentes para realizar um experimento específico em modo *exclusivo*, o que significa que o servidor

[1] O código fonte e instruções de instalação estão disponíveis em `http://martin.lncc.br/main-software-flame/`

[2] [Kirszenblatt et al. 2009] apresentou uma versão preliminar da plataforma baseada em web services.

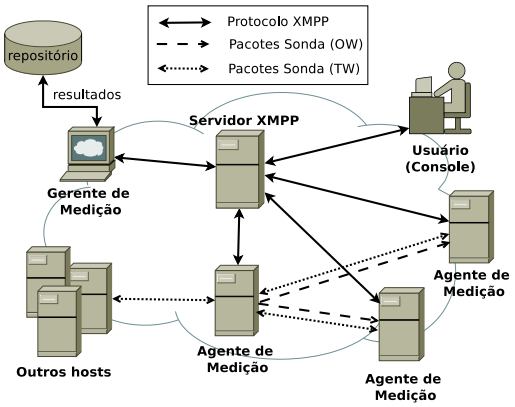

Figura 1. Arquitetura da plataforma FLAME.

XMPP fará com que tais agentes estejam temporariamente indisponíveis para outros experimentos. Essa funcionalidade permite efetuar experimentos que de outro modo seriam influenciados por experimentos simultâneos no mesmo grupo de agentes. Os agentes de medição são responsáveis por executar os experimentos, armazenar temporariamente os resultados colhidos e enviar esses resultados para o gerente ao final dos experimentos – esse armazenamento temporário nos agentes é necessário para evitar que atualizações remotas do repositório central durante os experimentos também influenciem nos resultados.

Um experimento é especificado como um script que descreve os protótipos das ferramentas utilizadas no experimento e os comandos que invocam essas ferramentas. Tal script tem acesso a uma API de primitivas básicas de medição implementadas pelo agente (ver Seção 3). Tanto o script quanto os respectivos resultados coletados pelas suas medições são publicados no repositório central. Isso confere à plataforma um primeiro suporte à proveniência de dados, permitindo que os protótipos das ferramentas e os comandos que foram invocados possam ser posteriormente analisados (por exemplo, com relação à sua acurácia) com base nos resultados obtidos.

Note que as medições podem ser realizadas entre um grupo de agentes de medição (nós cooperativos), ou entre agentes de medição e nós que não hospedam agentes de medição (por exemplo, nós que respondam a sondas do tipo ICMP Echo Request), como ilustrado na Figura 1. No caso das medições entre nós cooperativos, a plataforma dá suporte a medições bidirecionais (Two Way – TW) e unidirecionais (One Way – OW).

3. Prototipagem rápida de ferramentas de medição ativa

Na plataforma FLAME, os scripts que descrevem as ferramentas prototipadas e os comandos de invocação dessas ferramentas são especificados usando a linguagem Lua [Ierusalimschy et al. 1996]. Essa linguagem foi escolhida por oferecer: (i) uma sintaxe simples (procedural) e com nível alto de abstração; (ii) uma semântica extensível (p.ex. permitindo o uso de *threads* em diferentes níveis de colaboração); e (iii) um *footprint* de memória bastante reduzido,[3] o que torna a plataforma proposta igualmente aplicável em nós com diferentes níveis de disponibilidade de recursos computacionais.

[3]Um *benchmarking* comparativo com Ruby, Python e Perl, disponível em `http://shootout.alioth.debian.org`, aponta um consumo médio de memória, respectivamente, 530%, 260% e 190% maior nessas linguagens do que em Lua. Isso confere à plataforma FLAME um outro diferencial, de caráter tecnológico, em comparação com o sistema Scriptroute.

Cada agente de medição na plataforma FLAME hospeda um interpretador Lua adaptado, que provê um ambiente de *sandboxing* para a execução controlada de scripts Lua. As primitivas de medição são disponibilizadas pelos agentes aos scripts Lua por meio de uma API em C exportada para o interpretador Lua com o nome `lamp` (*Lua Active Measurements Primitives*). Os nomes das operações de medição oferecidas pela API `lamp` obedecem ao padrão `send[protocol][type](...)`, onde `protocol` indica o uso do protocolo utilizado para o envio de sondas (na implementação atual, UDP, TCP ou ICMP), e `type` indica sondas bidirecionais (TW – *Two Way*), unidirecionais (OW – *One Way*) ou em rajadas (PT – *Packet Train*). Todas as operações de medição ativa recebem como conjunto de parâmetros de entrada e devolvem como conjunto de valores de retorno uma tabela Lua.[4]

As operações do tipo TW independem da existência de um nó de destino cooperativo, e os resultados obtidos com essas operações são coletados no nó de origem. Nesse caso, o nó de origem é responsável pelo envio dos resultados ao gerente de medição, a fim de serem armazenados no repositório centralizado. As operações do tipo OW e PT dependem da presença de um nó de destino cooperativo, que é responsável por coletar os resultados e enviá-los ao gerente de medição para armazenamento no repositório. Nesse caso, porém, o nó de destino também é responsável por enviar os resultados coletados ao nó de origem, o que é importante no caso de ferramentas de medição ativa que dependam de iterações sucessivas baseadas no retorno dos nós de destino cooperativos (como no caso da ferramenta `pathload` [Jain e Dovrolis 2003]).

Além das operações de medição na API `lamp`, a plataforma FLAME oferece também operações auxiliares em uma API `lamu` (*Lua Active Measurements Utilities*), como suspensão da execução do script durante um período de tempo (`lamu.sleep()`), por exemplo. Devido a restrições de espaço, omitimos neste artigo detalhes das operações dessas APIs. Contudo, o uso de várias dessas operações é exemplificado na seção a seguir – em [Ziviani et al. 2010] apresentamos mais detalhes sobre os tipos de parâmetros de entrada e valores de retorno de cada uma dessas operações.

4. Demonstração da plataforma

Nesta seção são apresentados alguns experimentos de demonstração para ilustrar como as ferramentas de medição ativa podem ser prototipadas e usadas na plataforma FLAME. Esses mesmos experimentos, bem como outros apresentados em [Ziviani et al. 2010] (não descritos aqui por limitações de espaço), serão conduzidos durante o Salão de Ferramentas do SBRC 2010.

Os experimentos de demonstração envolvem dois cenários: o Planet-Lab e um testbed local. Destacamos que esses experimentos de demonstração têm como objetivo ilustrar o uso da plataforma, e não prover uma avaliação de desempenho da mesma.

As subseções a seguir descrevem os cenários de experimentação e comparam os resultados dos experimentos executados nesses cenários com ferramentas disponí-

[4] Lua oferece um único tipo estruturado de dados, a *tabela*, que implementa arranjos associativos do tipo $\{k_1 = v_1, \ldots, k_n = v_n\}$, ou seja, cada valor v_i armazenado na estrutura é associado a uma chave k_i. Chaves e valores podem ser de qualquer tipo, sendo números e strings os tipos mais comuns de chaves. Valores são indexados por chaves numéricas na forma `tabela[chave]` e por chaves alfanuméricas na forma `tabela["chave"]` ou simplesmente `tabela.chave`.

veis publicamente e protótipos funcionalmente equivalentes implementados na plataforma FLAME. No caso desses protótipos, optou-se por uma abordagem de coleta das medições por meio de consultas pós-experimentos ao repositório central da plataforma. Essa abordagem encoraja o desacoplamento entre as funcionalidades de sondagem e de formatação de dados de saída, o que pode ser observado pela forma (no nosso entendimento) compacta e limpa dos códigos dos protótipos apresentados nas subseções a seguir.

4.1. Traçado de rota

Para nossos experimentos de traçado de rota usamos 6 nós do Planet-Lab espalhados pelos 5 continentes, sendo 1 nó (S1) como origem e outros 5 nós (T1, ..., T5) como destino. Usamos no nó S1 a ferramenta `traceroute` e prototipamos uma ferramenta equivalente (`flameTrace`) usando a API `lamp`, implantando-a no agente de medição executando nesse mesmo nó. Para essa demonstração, é usado um servidor XMPP e um gerente de medição hospedados em nosso laboratório. O código da ferramenta `flameTrace` é apresentado na Listagem 1.

```
1  function flameTrace(params)
2    local _target = params.target or "127.0.0.1"; local _size = params.size or 60
3    local _interval = params.interval or 250000;  local _npackets = params.npackets or 3
4    local _protocol = params.protocol or "udp";   local _timeout = params.timeout or 5
5    local _maxhops = params.maxhops or 30
6
7    for h = 1, _maxhops do
8      local _response
9      for i = 1,_npackets do
10       -- Choose probing protocol (for UDP and TCP primitives,
11       --  since port is not indicated it is randomly chosen above 1024)
12       if _protocol == "icmp" then
13         _response=lamp.sendICMPTW{ip=_target, size=_size, timeout=_timeout, ttl=h}
14       elseif _protocol == "udp" then
15         _response=lamp.sendUDPTW{ip=_target, size=_size, timeout=_timeout, ttl=h}
16       elseif _protocol == "tcp" then
17         _response=lamp.sendTCPTW{ip=_target, size=_size, timeout=_timeout, ttl=h}
18       else
19         print("INVALID PROTOCOL:", _protocol); return
20       end
21       -- Check host/net unreachability
22       if _response and _response.loss == lamu.err.ICMP_HOST_UNREACH then
23         print("DESTINATION UNREACHABLE: ", _target); return
24       end
25       -- Wait time between probes
26       lamu.sleep(_interval)
27     end --for i
28     -- Is current node the target?
29     if (_response.dstIP == _target) then break end
30   end --for h
31 end
```

Listagem 1. Ferramenta de traçado de rota.

Para disparar os experimentos de traçado de rota no Planet-Lab usando o protótipo `flameTrace`, os seguintes comandos foram executados em um console FLAME:

```
local targets = {T1="130.83.166.198", T2="128.112.139.80",
                 T3="132.65.240.100", T4="130.216.1.22", T5="131.112.243.102"}

for k,v in ipairs(targets) do flameTrace{target=v} end
```

Os experimentos de traçado de rota foram conduzidos no Planet-Lab usando as ferramentas `traceroute` e `flameTrace` de forma intercalada por um total de 18

horas. A Figura 2 representa graficamente os caminhos identificados pelas duas ferramentas entre os nós do Planet-Lab usados nos experimentos. Os resultados de ambas as ferramentas foram muito similares, não apontando mudanças de rotas durante os experimentos. A única diferença percebida foi no processo de traçado de rota entre os nós S1 e T1. Na Figura 2, os loops no nó T1 representados por linhas tracejadas indicam que na maioria dos experimentos a ferramenta `traceroute` não conseguiu perceber que chegou ao nó de destino, continuando a incrementar o campo TTL das sondas até seu valor máximo (30 saltos). Essa situação não ocorreu com a ferramenta `flameTrace` em experimento algum. Atribuímos esse fenômeno a uma situação inesperada que a ferramenta `traceroute` não soube tratar adequadamente. Com as facilidades de prototipagem rápida oferecidas pela plataforma FLAME esperamos que esse tipo de situação inesperada possa ser mais facilmente identificada e corrigida pelo desenvolvedor do protótipo.

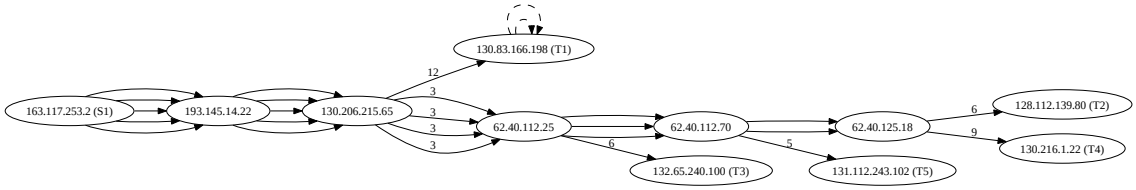

Figura 2. Representação gráfica dos resultados das ferramentas `traceroute` e `flameTrace`. Os arcos numerados indicam o número de saltos omitidos entre os nós correspondentes.

4.2. Medição do atraso unidirecional

Ao contrário das ferramentas `traceroute` e `flameTrace`, as ferramentas usadas na medição do atraso unidirecional dependem de nós de destino cooperativos. Para minimizar erros de sincronização de relógio entre os nós de origem e destino (algo não trivial de ser conseguido no Planet-Lab), esses experimentos foram conduzidos em nosso testbed local com os nós de origem e destino sincronizando seus relógios com um servidor NTP local. Esse testbed está disponível para uso remoto através de um console FLAME Web acessível a partir de `http://martin.lncc.br/main-software-flame/`, permitindo a familiarização com a plataforma FLAME em ambiente controlado. Instruções de uso do console Web podem ser obtidos diretamente a partir dessa URL, bem como exemplos de protótipos para medição de RTT e de estimação de capacidade de enlace, além dos de traçado de rota e de medição de atraso unidirecional ilustrados neste artigo.

Para os experimentos de medição de atraso unidirecional, instalamos a ferramenta `owping` em dois nós do testbed (uma origem e um destino) e prototipamos uma ferramenta equivalente (`flameOWD`) na plataforma FLAME usando a API `lamp`, implantando-a no agente de medição executando no nó de origem (nesses experimentos precisamos também de um agente executando no nó de destino, mas nenhum script Lua precisa ser implantado explicitamente no mesmo). O código na Listagem 2 ilustra o protótipo da ferramenta `flameOWD`.

O comando `flameOWD{target="10.0.2.2", npackets=200}` foi executado em um console FLAME para disparar os experimentos de medição de atraso unidirecional a partir do nó de origem em nosso testbed local. O nó de destino nesse testbed está localizado a 3 saltos do nó de origem.

```
function flameOWD(params)
  local _target = params.target or "127.0.0.1";
  local _size = params.size or 56;           local _interval = params.interval or 750000
  local _npackets = params.npackets or 5;    local _port = params.port or nil;
  local _timeout = params.timeout or 900000; local _protocol = params.protocol or "udp"

  for i = 1,_npackets do
    local _response
    -- Choose probing protocol (if port is not indicated the destination node
    --  chooses a port above 1024. In any case, for OW operations,
    --  the source and destination nodes agree on the used port through XMPP messages)
    if _protocol == "udp" then
      _response=lamp.sendUDPOW{ip=_target, size=_size, timeout=_timeout, port=_port}
    elseif _protocol == "tcp" then
      _response=lamp.sendTCPOW{ip=_target, size=_size, timeout=_timeout, port=_port}
    else
      print("INVALID PROTOCOL:", _protocol)
      return
    end
    -- Check port unavailability and host/net unreachability
    ...
    -- Wait time between probes
    ...
  end --for i
end
```

Listagem 2. Ferramenta de medição de atraso unidirecional.

A Tabela 1 ilustra algumas estatísticas de atraso unidirecional coletadas em nosso testbed local usando as ferramentas `owping` e `flameOWD` e duas configurações de enlace distintas: (i) com os 3 enlaces entre os nós de origem e destino trabalhando a 10 Mbps (“configuração 10-10-10”); e (ii) com o enlace no meio do caminho entre os nós de origem e destino trabalhando a 100 Mbps (“configuração 10-100-10”). Essas estatísticas foram obtidas com 200 sondas de 56 bytes para cada uma das configurações de enlace descritas. A tabela mostra para cada ferramenta e configuração de enlace a mediana e os percentis 05 e 95 entre as 200 amostras. É importante notar que em ambas as configurações de enlace o protótipo `flameOWD` obteve menores atrasos unidirecionais, embora o erro de sincronização estimado pelo NTP (o mesmo para ambas as ferrramentas) indique resultados estatisticamente equivalentes entre as duas ferramentas.

Tabela 1. Atraso unidirecional medido (em ms). O erro de sincronização estimado pelo NTP é de $\pm$ 0.41 ms.

	Configuração 10-10-10			**Configuração 10-100-10**		
	P05	Mediana	P95	P05	Mediana	P95
`owping`	1.28	1.39	1.50	1.10	1.21	1.32
`flameOWD`	0.87	0.91	0.98	0.46	0.51	0.58

5. Conclusões

Neste documento apresentamos uma plataforma para prototipagem rápida de ferramentas de medições ativas. No geral, em comparação com as principais características dos trabalhos anteriores, o FLAME oferece as seguintes contribuições: (i) um ambiente para a prototipagem rápida de ferramentas de medição ativa baseado em um conjunto pequeno mas abrangente de primitivas de sondagem, permitindo a implementação de ferramentas que dependem ou não de nós de destino cooperativos; (ii) um repositório centralizado para o armazenamento de resultados de medições usando um formato de dados único, o que encoraja o desenvolvedor de ferramentas a desacoplar as funcionalidades

de sondagem e formatação de dados de saída e facilita o processo de análise e comparação entre diferentes conjuntos de dados de medições. Também é interessante destacar o objetivo futuro declarado pela nova infraestrutura de medição ativa da CAIDA, o Ark [CAIDA 2010], de fornecer uma API de alto nível para ajudar os desenvolvedores de ferramentas de medição a lidar com os desafios de codificação desse tipo de ferramenta. Nesse contexto, acreditamos que a nossa plataforma FLAME atinge este objetivo.

Como trabalho futuro, pretendemos implementar outras ferramentas de medição ativa bem conhecidas para continuar validando a abrangência da nossa API de medição. Além disso, pretendemos avaliar o desempenho da plataforma FLAME e seus protótipos em comparação com outras plataformas como Scriptroute e ferramentas desenvolvidas com linguagens de baixo nível para medidas ativas. Pretendemos também portar nossa plataforma FLAME para dispositivos com recursos limitados como PDAs e sensores, tirando vantagem do baixo footprint de memória do interpretador Lua, para que possamos coletar dados de medição em redes celulares, redes ad-hoc e redes de sensores.

Agradecimentos

Este trabalho foi parcialmente financiado pelo MCT, CNPq e FAPERJ.

Referências

[Boote 2010] Boote, J. (2010). One–way ping (OWAMP). `http://e2epi.internet2.edu/owamp`.

[CAIDA 2010] CAIDA (2010). Archipelago measurement infrastructure. `http://www.caida.org/projects/ark/`.

[Ierusalimschy et al. 1996] Ierusalimschy, R., Henrique, L., Waldemar, F., e Filho, C. (1996). Lua – an extensible extension language. *Software: Practice and Experience*, 26:635–652.

[Jain e Dovrolis 2003] Jain, M. e Dovrolis, C. (2003). End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput. *IEEE/ACM Trans. Netw.*, 11(4):537–549.

[Kirszenblatt et al. 2009] Kirszenblatt, M., Correa, B., Ziviani, A., e Gomes, A. T. A. (2009). Prototipagem rápida de ferramentas de medição ativa. In *Anais do XIV Workshop de Gerência e Operação de Redes e Serviços (WGRS)*, Recife, PE. SBC.

[Michaut e Lepage 2005] Michaut, F. e Lepage, F. (2005). Application-oriented Network Metrology: Metrics and Active Measurement Tools. *IEEE Comm. Surv. & Tut.*, 7(2):24.

[Saint-Andre 2004] Saint-Andre, P. (2004). Extensible Messaging and Presence Protocol (XMPP): Core. *RFC 3920*.

[Spring et al. 2003] Spring, N., Wetherall, D., e Anderson, T. (2003). Scriptroute: a public internet measurement facility. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*.

[Ziviani et al. 2010] Ziviani, A., Gomes, A. T. A., Kirszenblatt, M. L., e Cardozo, T. B. (2010). FLAME: Flexible lightweight active measurement environment. In *Proceedings of the 6th International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities - TridentCom 2010*. Springer.