MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE MESTRADO EM SISTEMAS E COMPUTAÇÃO

DIOGO SOARES DOS SANTOS

UMA PLATAFORMA PARA SIMULAÇÃO DE ATAQUES *DDOS* UTILIZANDO O *NS-3*

Rio de Janeiro
2014

INSTITUTO MILITAR DE ENGENHARIA

DIOGO SOARES DOS SANTOS

UMA PLATAFORMA PARA SIMULAÇÃO DE ATAQUES *DDOS* UTILIZANDO O *NS-3*

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para obtenção do título de Mestre em Sistemas e Computação.

Orientador: Anderson Fernandes Pereira dos Santos - D.Sc.

Rio de Janeiro
2014

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80-Praia Vermelha
Rio de Janeiro-RJ CEP 22290-270

XXXXXSantos, D.S.
Uma Plataforma para Simulação de Ataques *DDoS* Utilizando o *ns-3*/ Diogo Soares dos Santos.
– Rio de Janeiro: Instituto Militar de Engenharia, 2014.
xxx p.: il., tab.

Dissertação (mestrado) – Instituto Militar de Engenharia – Rio de Janeiro, 2014.

1. Robôs móveis autônomos. 2. Robôs cooperativos. I. Título. II. Instituto Militar de Engenharia.

CDD 629.892

INSTITUTO MILITAR DE ENGENHARIA

DIOGO SOARES DOS SANTOS

UMA PLATAFORMA PARA SIMULAÇÃO DE ATAQUES *DDOS* UTILIZANDO O *NS-3*

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para obtenção do título de Mestre em Sistemas e Computação.

Orientador: Anderson Fernandes Pereira dos Santos - D.Sc.

Aprovada em 07 de agosto de 2014 pela seguinte Banca Examinadora:

Anderson Fernandes Pereira dos Santos - D.Sc. do IME - Presidente

---

Prof. Ronaldo Moreira Salles - Ph.D, do IME

Prof. Sidney da Cunha Lucena - DSc, da UNIRIO

Rio de Janeiro
2014

Dedico esta à...

AGRADECIMENTOS

Agradeço a todas as pessoas que contribuíram com o desenvolvimento desta dissertação de mestrado, tenha sido por meio de críticas, idéias, apoio, incentivo ou qualquer outra forma de auxílio. Em especial, desejo agradecer às pesssoas citadas a seguir.

Por fim, a todos os professores e funcionários do Departamento de Engenharia de Sistemas (SE/8) do Instituto Militar de Engenharia.

Nome do aluno

Epígrafe

AUTOR

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SÍMBOLOS

ABREVIATURAS

AS	-	*Sistema Autônomo*
BGP	-	*Border Gateway Protocol*
BRITE	-	*Boston university Representative Internet Topology gEnerator*
CPU	-	*Unidade Central de Processamento*
DDoS	-	*Distributed Denial of Service*
DNS	-	*Domain Name System*
DoS	-	*Denial of Service*
FC	-	*Flash Crowd*
HTTP	-	*Hypertext Transfer Protocol*
ICMP	-	*Internet Control Message Protocol*
IGP	-	*Interior Gateway Protocol*
IP	-	*Internet Protocol*
IRC	-	*Internet Relay Chat*
I/O	-	*Entrada e Saída*
IME	-	*Instituto Militar de Engenharia*
PoP	-	*Ponto de presença*
OSI	-	*Open Systems Interconnection*
RAM	-	*Random Access Memory*
SIP	-	*Session Iniciation Protocol*
TcL	-	*Tool Command Language*
TCP	-	*Transmission Control Protocol*
TTL	-	*Time to Live*
UDP	-	*User Datagram Protocol*

RESUMO

Nos últimos anos houve um gradativo aumento na quantidade e qualidade dos ataques distribuídos de negação de serviço (*DDoS*). Além dos alvos tradicionais como portais de bancos e *e-commerce* existe hoje uma grande quantidade de ataques direcionados a órgãos governamentais. Tal fato aumenta ainda mais a importância na pesquisa nesta área. Uma ferramenta importante para apoiar os pesquisadores é a simulação destes ataques. Apesar disso, as ferramentas tradicionalmente disponíveis não apresentam facilidades necessárias para a modelagem das características deste tipo de ataque. Neste trabalho de pesquisa é proposto um ambiente de simulação de ataques *DDoS* utilizando como base o simulador de redes *ns-3*. A plataforma permite que o pesquisador realize experimentos com flexibilidade e escalabilidade.

ABSTRACT

In recent years there has been a gradual increase in the quantity and quality of distributed denial of service (DDoS) attacks. Beyond the traditional targets like banking sites and e-commerce there is a large amount of targeted attacks on government agencies. This fact adds to the importance of research in this area. An important tool to support researchers is a simulation environment where such attacks occur. Nevertheless, the tools have not traditionally available facilities necessary for a good modeling of the characteristics of this type of attack. In this research proposes a simulation environment DDoS attacks based on the network simulator ns-3.The platform allows researchers to perform experiments with flexibility and scalability.

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

O uso da Internet tornou-se indispensável na vida moderna. Empresas e órgãos governamentais buscam cada vez mais disponibilizar seus serviços e informações na rede mundial de forma a aumentar o alcance e eficiência no atendimento às necessidades de seus clientes e usuários. Uma das maiores ameaças à disponibilidade destes serviços são os ataques *DDoS*. *DoS*(*Denial of Service*) foi definido como o ataque que reduz, ou elimina, ilegalmente a disponibilidade de um serviço para um usuário legítimo (NOURELDIEN, 2002). Quando este é proveniente de várias fontes é chamado de *DDoS*(*Distributed Denial of Service*).

A defesa contra este tipo de ataque é um grande desafio. Inclui formas de prevenir, detectar, identificar responsáveis e mitigar o ataque. Uma das principais formas de defesa é a detecção do ataque e conseguir realizar a separação entre tráfego malicioso e tráfego legítimo.

Existem várias pesquisas nesta área (AKELLA, 2003), (FEINSTEIN, 2003), (JIN, 2004), (KARIMAZAD, 2011), (KULKARNI, 2006), (LI, 2005), (MIRKOVIC, 2006), (SANTOS, 2007), (XIANG, 2011), (YU, 2008), (ZHANG, 2012) e utilizam, para a validação de suas técnicas, *traces* de redes que estão recebendo tráfegos maliciosos e legítimos. Os *traces* utilizados podem ser tanto reais, isto é, obtidos através de ataques monitorados em redes reais ou então sintéticos, àqueles produzidos de forma artificial mas que possuem as mesmas características encontradas em ataques reais. Pode ainda ser utilizada a simulação de redes.

Os ataques reais são de difícil obtenção, tanto pelo fato de nem sempre as redes estarem sendo monitoradas de forma a produzir *traces* quanto pelo fato de serem dados estratégicos tanto para pesquisadores quanto em relação a segurança de empresas e de governos. Os ataques sintéticos podem ser produzidos utilizando-se um ambiente controlado onde os responsáveis pelo tráfego malicioso e legítimo estejam presentes e direcionem seus fluxos a um determinado *host*. Este tipo de obtenção porém possui pouca escalabilidade pois existem ataques com milhares de nós e torna-se difícil possuir esta quantidade de *host*s

em um ambiente de laboratório.

Outra forma de produzir *traces* sintéticos é através de emulação e simulação. Ambas são abordagens próximas e buscam criar modelos que representem a realidade. A principal diferença entre as duas é que na emulação ainda existe alguma parte do sistema real enquanto na simulação todo o sistema encontra-se modelado.

As grandes vantagens da simulação são a escalabilidade e o custo. A desvantagem em relação aos outros métodos é a distância da realidade e a dificuldade de implementação. Cabe relembrar que o ambiente a ser simulado é de uma rede especialmente complexa: a Internet. Os alvos dos ataques *DDoS* são em sua maioria serviços disponibilizados na Internet.

A dificuldade na obtenção de *traces* representa um grande obstáculo nas áreas de pesquisa citadas. O objetivo com esse trabalho é iniciar o desenvolvimento de uma plataforma que simule ataques *DDoS* juntamente com todo o tráfego lícito existente em uma situação real.

1.2 DEFINIÇÃO DO PROBLEMA

Os ataques *DDoS* são complexos e a cada dia surgem novas formas de ataque. (MIRKOVIC, 2004b) apresenta a taxonomia de ataques, que representa uma considerável variedade dos ataques que estão ocorrendo hoje em dia. O objetivo não é esgotar todas as formas de ataques existentes. Entretanto foi proposta uma plataforma que buscasse independência da técnica de ataque utilizada.

1.3 OBJETIVO

Este trabalho tem por objetivo propor uma plataforma que permita a modelagem de um ataque *DDoS* a um *host* através de simulação.

Na plataforma proposta é necessário apenas que se tenham amostras reais de *traces* de tráfego lícito e de ataque. A obtenção destas amostras individualmente é bem mais simples quando comparadas às amostras de um ataque *DDoS* e podem ser geradas facilmente através do uso de ferramentas mais simples.

Para atingir o objetivo de se gerar os *traces* a plataforma é composta de um simulador de redes, um gerador de topologias e um gerador de tráfego.

A geração de topologia deve buscar representar o ambiente onde ocorrem os ataques

DDoS — a Internet. Além de ser necessario possuir a capacidade de gerar uma topologia com centenas e até milhares de nós, o processo de gerar a topologia deve respeitar propriedades intrínsecas existentes na Internet (FALOUTSOS, 1999).

A geração de tráfego deve permitir pelo menos a existência de dois tipos de tráfegos: tráfego de ataque (TA) e tráfego de fundo (TF). O tráfego de ataque é aquele realizado pelos integrantes da *botnet*. O tráfego de fundo é aquele produzido pelos usuários legítimos do serviço disponibilizado pelo servidor web. No mundo real existem tráfegos além dos que serão modelados no presente trabalho. Exemplos destes tráfegos são as mensagens enviadas pelo *botmaster* aos *bots* e a comunicação entre os estes. O foco da simulação desenvolvida é aquele que possui como destino o alvo do ataque.

Após a avaliação de várias ferramentas foi concluido que as que atenderiam da melhor forma o objetivo proposto seriam o *NS-3*(HENDERSON, 2006), *BRITE*(MEDINA, 2001) e *Tmix*(WEIGLE, 2006). O funcionamento e comparação com outras ferramentas serão abordados nas seções posteriores.

Ao fim da construção da plataforma esta será avaliada em termos de sua aderência com *traces* reais e em relação a escalabilidade em relação ao número de nós e tempo de simulação.

1.4 CONTRIBUIÇÕES

A ferramenta otimizará tempo e permitirá que pesquisadores validem suas técnicas e algoritmos com dados produzidos por ferramentas confiáveis e testadas pela comunidade acadêmica.

A avaliação dos *traces* gerados pela plataforma serve tanto para corroborar com a eficácia das ferramentas utilizadas quanto para a avaliação das capacidades das mesmas, principalmente do simulador *NS-3* quando utilizado com um grande número de nós.

1.5 ORGANIZAÇÃO DO TRABALHO

Esta dissertação é organizada da forma que se segue. No capítulo 2 é apresentada a fundamentação teória necessária para o desenvolvimento do presente trabalho. Isto inclui revisão de diversas técnicas de ataque *DDoS*, descrição detalhada das ferramentas utilizadas assim como a razão das suas escolhas. No capítulo 3 é apresentada a plataforma proposta e a comparação com outros simuladores disponíveis. No capítulo 4 são apresen-

tados os experimentos realizados e seus resultados. No capítulo 5 é realizada a conclusão da dissertação assim como propostas de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 *DDOS*

DoS(*Denial of Service*) foi definido como o ataque que reduz ou elimina ilegalmente a disponibilidade de um serviço para um usuário legítimo (NOURELDIEN, 2002). Isto pode ocorrer em uma série de contextos, desde sistemas operacionais(GLIGOR, 1983) até serviços de rede(NEEDHAM, 1994). Em qualquer cenário os atacantes são aqueles que possuem como objetivo que suas vítimas não consigam prestar os serviços a que foram destinadas. No presente estudo o foco é a negação dos serviços que são disponibilizados na Internet.

A indisponibilidade dos serviços podem ser atingidas de diversas formas como por exemplo pela interrupção do fornecimento de energia elétrica, incêndios e ataques físicos aos equipamentos responsáveis pelos serviços. Os serviços também podem ser interrompidos através da utilização de técnicas que utilizam o envio de mensagens aos alvos buscando interferir na sua operação fazendo com que esta pare, reinicie ou torne-se indisponível.

Desta maneira a vítima não consigue comunicar-se de forma adequada com usuários legítimos. Os alvos podem ser os próprios *hosts* vítimas assim como outros elementos da infraestrutura da Internet (roteadores, servidores *DNS*, etc...) que prestam suporte a estes *hosts*.

A negação de serviço pode ser ocasionada por duas formas principais:

- ***Exploits*:** através da utilização de erros no tratamento de alguma informação pelo *software* responsável por prover determinado serviço; e
- **Esgotamento de recursos:** quando os recursos necessários para prover o serviço são esgotados pela ação do usuário malicioso. Durante toda a pesquisa serão considerados ataques de negação de serviço (*DoS*) aqueles que referem-se a esta segunda forma de ataque.

O esgotamento dos recursos utiliza-se do envio de um tráfego maior (inundação, tradução livre de *flooding*) que o suportado pelo servidor ou pelos recursos que ele necessita para conectar-se à Internet e normalmente se refere a dois casos (ZARGAR, 2013):

- impedir que um ususário legítimo conecte-se através do esgotamento das capacidades de largura de banda do enlace, capacidade de processamento de roteadores ou recursos de rede. Estes são essencialmente ataques de inundação que utilizam as camadas de rede e transporte do modelo *OSI* (MIRKOVIC, 2004b); e

- impedir que um usuário legítimo utilize determinado serviço pelo esgotamento dos recursos do servidor (exemplo, *sockets*, *CPU*, memória, largura de banda do disco ou banco de dados, barramento de *I/O*). Estes normalmente incluem ataques de inundação que utilizam a camada de aplicação do modelo *OSI* (RANJAN, 2006).

Para gerar-se um tráfego que sobrecarregue as capacidades da vítima é necessário que o atacante possua sob seu comando um computador com capacidades de processamento e largura de banda suficientes para produzir o efeito desejado.

Assumindo que a vítima possua recursos abundantes de forma que o ataque realizado por apenas um computador torne-se inviável. Entretanto, suponha-se que o atacante possua controle sobre milhares de computadores e tem a capacidade de fazer com que estes gerem tráfego para vítima simultaneamente. Este conjunto de computadores mesmo possuindo poucos recursos (processadores lentos e pouca largura de banda disponível), em conjunto, conseguem formar uma grande rede de ataque, e, utilizados de forma adequada, conseguem sobrecarregar os recursos da vítima.

Ataque distribuído de negação de serviço (*DDoS*) é definido como a realização de vários ataques de negação de serviço, organizados em um ataque coordenado para um ou mais alvos predeterminados. Tanto o *DoS* quanto o *DDoS* são grandes ameaças, todavia o *DDoS* é um problema muito mais complexo e de difícil solução.

Ao utilizar um grande número de computadores para realizar o ataque, permite que a maioria das vítimas, por mais que possua recursos, possa ter seus serviços prejudicados. Controlar e enganjar um grande número de computadores em um ataque pode tornar-se uma tarefa simples, devido ao grande número de ferramentas atomáticas destinadas a ataques *DDoS* disponíveis na Internet. A utilização de um grande número de computadores propicia ao atacante outra grande vantagem. Mesmo que a vítima seja capaz de identificar os computadores que estão realizando o ataque, poucas ações podem ser tomada contra uma rede formada por milhares de computadores.

Outra característica que torna os ataques *DDoS* de difícil solução é o fato de alguns ataques utilizarem tráfego similar ao lícito. Quando as mensagens responsáveis pelo

ataque são comparadas com mensagens de usuários lícitos, frequentemente não existem características reveladoras que as distinguam. Desta forma, responder a um ataque acaba por atingir tamém a usuários legítimos.

Para a realização de um ataque *DDoS* é necessário que o atacante forme uma rede que é definida como *botnet*. *Botnets* são redes formadas por *hosts* infectados por um *malware* e comandada pelo atacante (neste caso denominado *botmaster*). Os *bots* individualmente são programas executados em *hosts* e que permitem aos *botmasters* controlar ações destes *hosts* remotamente (SAHA, 2005; ABU RAJAB, 2006). Um exemplo da arquitetura básica de um ataque é exemplificado na Figura 2.1.

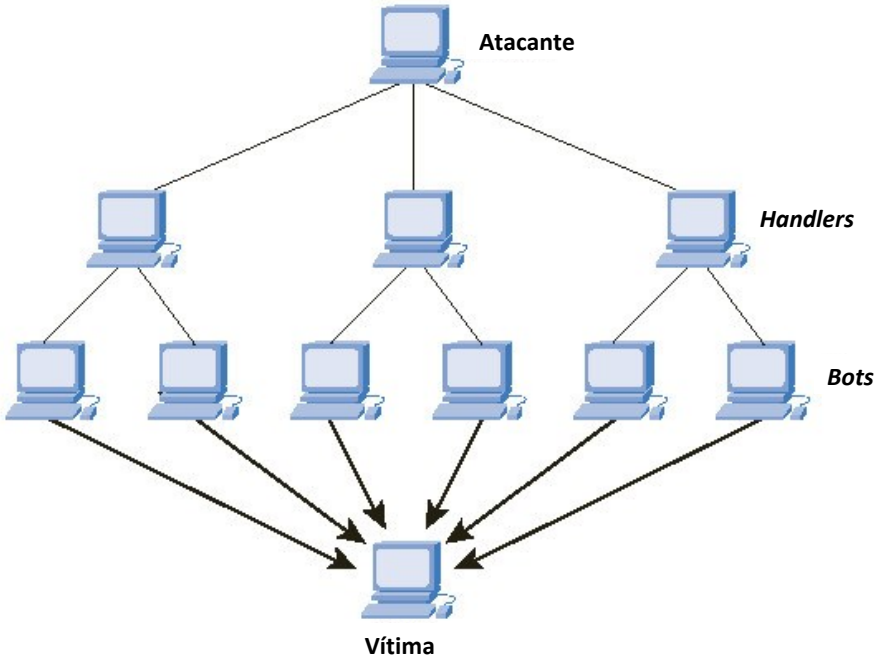

FIG. 2.1: Ataque *DDoS*.

A preparação e condução de um ataque *DDoS* pode ser descrita em fases (MIRKOVIĆ, 2003):

a) **Recrutamento**: O atacante seleciona, dentre *hosts* localizados na Internet, aqueles que realizarão o ataque. Estes equipamentos, usualmente são chamados de *bots*, e normalmente são: (a) externas à rede da vítima, visando evitar uma resposta eficiente da vítima, e (b) externas à rede do atacante, com o objetivo de evitar a responsabilização caso seja identificada a fonte do ataque. Para que os *hosts* transformem-se em *bots* é necessário que existam falhas de segurança para que

códigos maliciosos possam ser instalados. Também é desejável que os *bots* possuam recursos abundantes para serem capazes de realizar ataques poderosos. Este processo pode ser realizado manualmente, entretanto, é comum que seja feito de forma automática com a utilização de ferramentas que produzem listas de *hosts* que sejam potencialmente vulneráveis.

b) **Comprometimento**: O atacante ganha acesso (normalmente *root*) nos *hosts* através de brechas na segurança e implanta o código de ataque. Ele adota medidas para impedir que o código seja descoberto (renomeando arquivos, tornando-os ocultos ou colocando-os no diretório do sistema) e desativado (fazendo com que agendadores do sistema, como o *cron* do *Linux*, o reinicie periodicamente). Esta fase também pode ser automática. As mesmas ferramentas utilizadas para escanear a Internet podem obter acesso, inserir o código de ataque e enviar ao atacante a lista de *hosts* comprometidos.

c) **Comunicação**: *bots* reportam sua prontidão para o ataque através de *handlers* - *hosts* comprometidos que serão usados para controlar o ataque. Nos primeiros ataques *DDoS*, os endereços *IP* dos *handlers* encontravam-se diretamente no código do ataque e os *handlers* guardavam os *bots* disponíveis para o ataque em um arquivo encriptado. Logo a descoberta de um simples *bot* participante do ataque revelava todos os outros participantes. Com a utilização de outros canais de comunicação como o *Internet Relay Chat* (*IRC*), o servidor *IRC* rastreia o endereço dos *bots* conectados e dos *handlers* e facilita a comunicação entre eles. A descoberta de um participante do ataque leva à descoberta do canal de comunicação, mas a identidade dos outros participantes permanece protegida.

d) **Ataque**: Os atacantes normalmente comandam o ataque atavés dos *handlers* e dos canais de comunicação para os *bots*. O alvo, duração e características dos pacote como tipo, tamanho, TTL[1], número das portas, dentre outros, podem ser customizados. Atualmente as ferramentas de ataque *DDoS* utilizam-se de várias técnicas para evitar sua detecção:

- IP spoofing: ocorre quando um código malicioso cria seu próprio pacotee não

[1] *Time to Live* - número de saltos entre máquinas que os pacotes podem demorar numa rede de computadores antes de serem descartados.

seta o endereço *IP* real no cabeçalho do pacote (MIRKOVIC, 2004a).

- Tamanho variável de pacotes: a utilização de variados tamanhos de pacotes pelos *bots* dificulta a criação de uma assinatura para o ataque.
- Diferentes canais de comunicação: os *bots* podem comunicar-se entre si e com o *botmaster* utilizando vários canais e protocolos como *IRC*, *HTTP*, email, redes *P2P*, dentre outros.

2.1.1 CLASSIFICAÇÃO DOS ATAQUES *DDOS*

Com relação a fase de ataque, como foi citado anteriormente, pode-se classificar os ataques *DDoS* naqueles que atuam nas camadas de rede e transporte e aqueles que atuam sobre a camada de aplicação.

- Ataques de *flooding* nas camadas de rede e transporte: estes ataques são executados em sua maioria utilizando-se pacotes dos protocolos *TCP, UDP, ICMP* e *DNS*. Existem quatro tipos de ataques nesta categoria:
  - Ataques de *flooding*: atacantes focam em impedir usuários legítimos de conectarem-se pela exaustão da largura de banda da rede da vítima (*UDP flood, ICMP flood, DNS flood, por exemplo*);
  - Ataques de *flooding* de *exploit* de protocolo: Atacantes exploram características específicas ou erros de implementação dos protocolos da vítima com o objetivo de consumir em quantidades excessivas os recursos da vítima (exemplos: *TCP SYN flood, TCP SYN-ACK flood, ACK & PUSH ACK flood, RST/FIN flood*);
  - Ataques de *flooding* com refletores: os atacantes usualmente enviam requisições (ICMP echo request) para os refletores forjadas com o endereço de origem sendo o da vítima. Estes refletores enviam então suas respostas para a vítima e esgotam seus recursos (Ataques Smurf e Fraggle);
  - Ataques de *flooding* com amplificadores: Atacantes exploram serviços que possuem como característica gerar respostas muito maiores que as requisições realizadas ou várias mensagens de resposta para cada requisição. Reflexão e amplificação são normalmente aplicadas em conjunto como no caso do ataque *Smurf* onde o atacante envia requisições com o endereço *IP* forjado (Reflexão)

para um grande número de refletores explorando a característica do endereço *IP* de *broadcast* dos pacotes (Amplificação).

- Ataques de *flooding* na camada de aplicação: Estes ataques focam em impedir que usuários legítimos usem serviços exaurindo os recursos do servidor (*CPU*, memória e etc.). Ataques *DDoS* na camada de aplicação geralmente consomem menos largura de banda e são difíceis de serem detectados por natureza quando comparados aos ataques na camada de rede e transporte. Entretanto, ataques de *flooding* na camada de aplicação usualmente possuem o mesmo impacto aos serviços pois eles atuam em características específicas de aplicações como *HTTP, DNS* ou *SIP* [2].

  - Ataques de *flooding* com refletores/amplificadores: utilizam as mesmas técnicas dos ataques das camadas de rede e transporte (ex: enviando requisições da camada de aplicação forjadas a um grande número de refletores). Por exemplo, o ataque de amplificação de *DNS* emprega técnicas tanto de reflexão quanto de amplificação. Os *bots* geram pequenas consultas *DNS* com o endereço *IP* de origem forjado e consegue gerar um grande volume de tráfego de rede pois as respostas do *DNS* podem ser substancialmente maiores que as mensagens de consulta *DNS*.
  - Ataques de *flooding HTTP*: Existem três tipos de ataques nesta categoria:
    - Ataques de *flooding* de sessão: Neste tipo de ataque, a quantidade de requisições de conexão de sessões feitas pelos atacantes é maior que as realizadas pelos usuários lícitos.
    - Ataques de *flooding* de requisição: atacantes enviam sessões que contém um número de requisções maior que o usual.
    - Ataques assimétricos: Neste ataque são enviadas sessões que produzem uma alta carga de trabalho no servidor.
      - *Flooding* de múltiplos *HTTP get/post*: o atacante envia um pacote contendo múltiplas requisições *HTTP* sem possuir nenhuma delas em uma única sessão *HTTP*.
      - Slowloris: o atacante envia requisições *HTTP* incompletas que continuamente e rapidamente crescem, fazem lentas atualizações e nunca

[2] *Session Iniciation Protocol.*

são fechadas. O ataque é efetuado até que todos os *sockets* disponíveis são ocupados por essas requisições e o servidor torna-se inacessível.

- Fragmentação de *HTTP*: os atacantes estabelecem uma conexão *HTTP* válida com o servidor. Ele então fragmenta os pacotes *HTTP* legítimos em pequenos fragmentos e envia cada um deles o mais lento que o *time out* do servidor permitir.
- *Slowpost*: O atacante envia um cabeçalho *HTTP* que define o campo "*content-lenght*" do corpo da mensagem *post* para iniciar o tráfego. Envia os dados para preencher o corpo da mensagem com a taxa de um *byte* a cada dois minutos. O servidor aguarda cada corpo da mensagem ser completado.
- *Slowreading*: este ataque funciona setando um valor pequeno para o tamanho da janela e o enviando para o servidor. O protocolo *TCP* mantém abertas conexões mesmo que não exista comunicação de dados. Desta forma o atacante pode forçar o servidor a manter um grande número de conexões abertas.

Um importante observação é a predominância dos ataques que utilizam protocolos da camada de aplicação como *DNS* e *HTTP*, conforme é reportado no relatório *DDoS Threat Report 2013* (NSFOCUS, 2013) do *NSFOCUS*, figura 2.2.

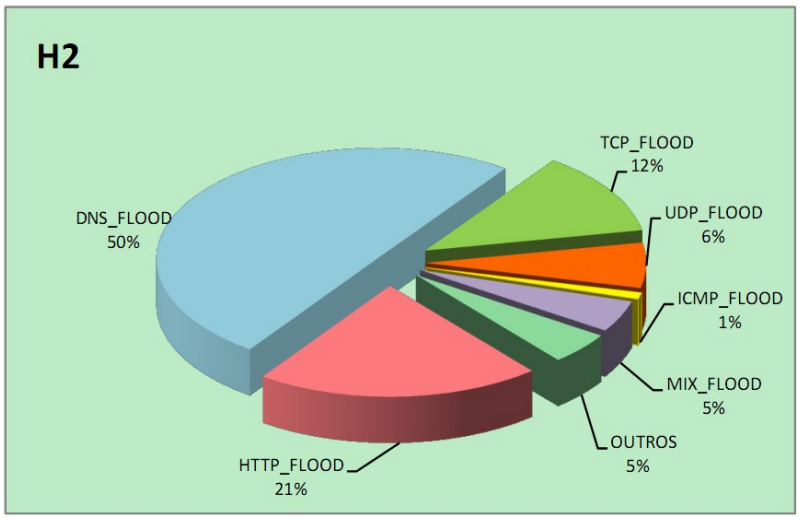

FIG. 2.2: Estatística dos tipos de ataques *DDoS* no segundo semestre de 2013 (NSFOCUS, 2013).

2.1.2 FERRAMENTAS *DDOS*

Serão apresentados abaixo alguns exemplos e suas características.

- *Trinoo*(DITTRICH, 1999a): utilizado para proceder um ataque de *flood UDP* contra um ou mais endereços *IP*. O ataque utiliza pacotes *UDP* de tamanho constante contra portas aleatórias da vítima. O *handler* utiliza *UDP* ou *TCP* para comunicar-se com os *bots*. O canal de comunicação pode ser criptografado e protegido por senha.

- *Tribe Flood Network (TFN)*(DITTRICH, 1999c): pode gerar *flooding* de *UDP* e *ICMP echo request*, *flooding* de *TCP SYN* e *broadacast* de *ICMP* direcionado (*Smurf*). Pode realizar *spoofing* de *IP* e atacar portas aleatórias da vítima. A comunicação entre os *handlers* e os *bots* é feita exclusivamente através de pacotes *ICMP_ECHO_REPLY*.

- *Stacheldraht*(DITTRICH, 1999b): combina características do Trinoo (arquitetura *handler/bot*) com as do *TFN* (*flooding ICMP/TCP/UDP* e ataques *Smurf*) . Adiciona criptografia ao canal de comunicação entre o atacante e os handlers. A comunicação é realizada através de pacotes *TCP* e *ICMP*. Permite atualização realizando uma nova cópia de si mesmo através de comandos *Berkely "rcp"* em algum sistema que funcione como servidor.

- *Shaft*(DIETRICH, 2000): Utiliza-se de *flooding* de *TCP*, *UDP* e *ICMP* para realizar o ataque, podendo utilizar os três ao mesmo tempo. O protocolo *UDP* é utilizado como forma de comunicação entre o *bot* e o *handler* sem encriptação. Utiliza portas e endereços *IP* de origem aleatórios. O tamanho dos pacotes é fixo durante o ataque.

- *LOIC*(MANSFIELD-DEVINE, 2011): desenvolvido inicialmente como uma ferramenta de teste de *stress* em redes. Realiza ataques utilizando protocolos *UDP, TCP* e *HTTP*. Possui como característica a possibilidade de recrutamento e o comprometimento de usuários voluntários usando a ferramenta para ceder acesso do seu equipamento para o atacante. Utiliza canais *IRC* para enviar comandos de ataque.

Um evento com comportamento semelhante aos ataques *DDoS* mas que merece especial distinção são os *flash crowds* (JUNG, 2002). Estes ocorrem quando um número grande de usuários legítimos acessam simultaneamente determinado portal, normalmente de notícias. Este aumento anormal de usuários é relacionado a acontecimentos específicos como promoções, eventos esportivos, dentre outros. Os acontecimentos citados são previsíveis e passíveis de preparação para atender ao aumento de demanda de usuários. Em termos gerais o que difere um *flash crowd* de um ataque *DDoS* é principalmente a existência de um acontecimento que o provoque e a inexistência de usuários maliciosos.

Clientes que acessam um *site* podem ser agregados em grupos que encontram-se sob o mesmo domínio administrativo. Estes grupos são chamados de *network-aware clusters*(KRISHNAMURTHY, 2000). Em um *flash crowd* são encontrados um grande número de *clusters* identificados anteriormente ao evento quando comparados a um ataque *DDoS*. Ocorre uma sobreposição de *clusters*. Em um *flash crowd* a quantidade de *hosts* que possuem acessos anteriores ao evento é bem superior do que em ataques *DDoS*.

Outra diferença é que em ataques *DDoS* a distribuição dos *hosts* entre os *clusters* é mais uniforme em comparação a *flash crowds*. Isso se deve à grande diferença entre o tamanho dos *clusters* (KRISHNAMURTHY, 2000). Um ataque entretanto não reflete essa diferença de tamanho entre os *clusters*.

A terceira diferença é a regularidade das requisições dos *hosts* enganjados em um ataque. Normalmente as ferramentas que realizam o ataque fazem requisições em tempos fixos ou aleatórios. No caso de *flash crowds*, conforme o servidor é sobrecarregado e tem seu tempo de resposta às requisições aumentado, o tempo entre as requisições dos *hosts* também aumenta.

A quarta diferença é o padrão da requisição de arquivos. Em um *flash crowd*, ao contrário do que acontece em ataques *DDoS*, a distribuição das requisições de arquivos respeitam a lei de Zipf.

A lei de Zipf é uma lei empírica a qual rege a dimensão, importância ou frequência dos elementos de uma lista ordenada. Trata-se de uma lei de potências sobre a distribuição de valores de acordo com o número de ordem numa lista. Numa lista, o membro n teria uma relação de valor com o primeiro da lista segundo 1/n. Por exemplo, em um portal, a lista da frequência de acessos aos arquivos armazenados no portal segue uma distribuição

que aproxima-se de $P_n \sim 1/n^a$, onde P_n representa a frequência de um arquivo ordenado na n-ésima posição e o expoente a é próximo da unidade. Isto significa que o segundo elemento se repetirá aproximadamente com uma frequência que é metade da do primeiro, e o terceiro elemento com uma frequência de 1/3 e assim sucessivamente.

Estas características estão sumarizadas a Tabela 2.1

Outro fenômeno que pode ser confundido com um ataque *DDoS* é o efeito *slashdot* (HALAVAIS, 2001). Este ocorre quando um *site* muito popular possui um *link* para um de menor capacidade causando a este um aumento no seu tráfego. Este aumento e carga no *site* de menor capacidade pode até mesmo causar a interrupção do serviço prestado por este.

2.3 SIMULADORES DE REDE

O *OMNeT++* (VARGA, 2010) não é um simulador de redes em sua definição, mas um *framework* para simulações de eventos discretos de propósito geral baseado em *C++* (WEINGARTNER, 2009). É utilizado para a modelagem de redes de comunicações, multiprocessadores e outros sistemas paralelos ou distribuídos. Ao invés de implementar diretamente componentes para redes de computadores provê os meios necessários para que a construção e simulação destes elementos possa ser executada(VARGA, 2008).

O *ns-2* é um simulador de redes que utiliza *C++* para modelar o comportamento dos nós e *scripts Tcl* que controlam a simulação e outros aspectos como a topologia da rede(WEINGARTNER, 2009). Apesar de ainda existirem muitos trabalhos que o utilizem seu desenvolvimento e manutenção foram descontinuados.

2.3.1 *NS-3*

O *NS-3* (HENDERSON, 2006) é um simulador de redes de eventos discretos, com ênfase nas camadas 2, 3 e 4 do modelo *OSI*. Seus escopos de utilização são principalmente os voltados a pesquisa e educacionais. As simulações são realizadas utilizado-se C++ ou *Python*.Apesar das características similares ao seu antecessor *ns-2*, não é uma atualização mas sim um novo projeto. Algumas das principais entidades implementadas no *ns-3* são descritas a seguir:

- **Nó**: responsável por representar os dispositivos computacionais da simulação. Deve ser pensado como um computador no qual se adicionam funcionalidades, tal como

Característica	*Flash Crowd*	*DDoS*
Volume de tráfego	Ambos possuem um aumento significativo no número de requisições.	
Distribuição dos clientes	É esperado que siga a mesma distribuição populacional existente nos *clusters*.	Não respeita necessariamente a distribuição populacional existente nos *clusters*.
Requisições por cliente	Como o servidor responde mais lentamente durante um *FC*, a tendência é que a quantidade de requisições por cliente diminuam conforme a resposta do servidor torna-se mais lenta.	A quantidade de requisições por cliente permanece constante durante o ataque e é significantemente diferente do normal.
Quantidade de clientes novos	A quantidade de clientes que já acessaram o servidor anteriormente é considerável durante um *FC*.	Existe uma grande quantidade de clientes que não acessam o servidor com frequência ou que nunca acessaram. Ocorre sobreposição de *clusters*.
Arquivos requisitados	A popularidade dos arquivos segue a lei de Zipf.	A popularidade dos arquivos acessados pelos *bots* não segue a lei de Zipf. Tanto é possível que um grande número *bots* acessem um pequeno conjunto de arquivos quanto um *bot* em particular faça repetidamente a requisição de um mesmo conjunto maior de arquivos.

TAB. 2.1: Tabela comparativa *flash crowd* e *DDoS*

aplicativos e pilhas de protocolos. Esta abstração é representada pela classe ***node***. Elementos de *hardware* como memória e processador não fazem parte do modelo.

- **Pacote**: representa os pacotes que trafegam na rede. Possuem tanto os cabeçalhos quanto os *payloads* de cada protocolo.

- **Aplicação**: representa um programa de usuário que gera alguma atividade a ser simulada. É representado pela classe Application. Um exemplo é a classe ***OnOffApplication*** que permite estabelecer tempos em que a aplicação deve estar ativa criando tráfego e os tempos em que esta não deve gerar qualquer tráfego.

- **Canal de comunicação**: representa o canal de comunicação ao qual um nó é conectado. Representado pela classe ***Channel***, fornece os métodos para gerenciar objetos de comunicação de sub-redes e nós conectados a eles.

- **Dispositivos de rede**: representa tanto o *hardware* (placa de rede, por exemplo) quanto o *software* (*driver*) relacionado a este hardware. Um dispositivo de rede é "instalado" em um nó para que este possa comunicar-se com outros na simulação, através dos canais de comunicação. Desta forma um nó pode ser conectado a mais de um canal via múltiplos dispositivos de rede. É representado pela classe ***NetDevice***.

- **Protocolo**: representa os principais protocolos como *TCP, UDP, IP* e *Ethernet* por exemplo.

Além de possuir uma crescente quantidade de trabalhos científicos utilizando o *ns-3* como plataforma de simulação, sua escolha também deveu-se ao desempenho em comparação com outros simuladores. Em duas importantes métricas de performance como o tempo de simulação e utilização de memória, o *ns-3* obteve um desempenho superior às duas plataformas citadas anteriormente(WEINGARTNER, 2009).

O *ns-3* foi desenvolvido para que pudesse possuir grande escalabilidade desde o seu projeto (HENDERSON, 2006). A escalabilidade depende quase que exclusivamente do *hardware* que é utilizado na simulação como foi demonstrado em (RENARD, 2012). Ainda com relação a adequação do *ns-3* a experimentos que modelem ataques *DDoS* cabe salientar que mesmo que se tenham *botnets* com centenas de milhares de *bots*, apenas os participantes do ataque serão simulados. Por exemplo, em (WANG, 2009), foi estimado que a *botnet Storm* possuía em fevereiro de 2008 por volta de 400.000 *bots*. Entretanto, foram contados entre 16.500 e 23.000 *bots online* (em condições de participarem de um ataque).

2.4 GERADORES DE TOPOLOGIA

2.4.1 *IGEN*

O *IGen*(QUOITIN, 2009) é uma ferramenta que propõe a geração de topologias baseando-se em heurísticas de projetos de rede ao invés de uma abordagem baseada em redes "*pure degree-based*" como o modelo *Barabási-Albert* por exemplo.

O funcionamento da ferramenta pode ser resumido da seguinte forma:

- Um conjunto inicial de nós com coordenadas geográficas é passado como entrada. Este conjunto inicial pode ser gerado de forma aleatória ou baseado em uma topologia pré-existente.
- Os nós são agrupados em pontos de presença (*PoPs*)(GREENE, 2002).
- São geradas as topologias de cada *PoP*. São selecionados n nós centrais (geograficamente) como nós *backbone*. Estes nós são densamente conectados de forma a garantir a criação de uma clique. Os outros nós restantes do *PoP* são então conectados aos nós *backbone* utilizando k vértices $(n, k \geq 2)$.
- É gerada a topologia de interconexão dos *PoPs*. As heurísticas implementadas são: *MENTOR, MENTour, Two-Trees, Delaunay* e *Multi-Tours*, descritas por Quoitin em (QUOITIN, 2004).
- São gerados pesos *IGP* e as capacidades dos enlaces.
- Finalmente é gerada a topologia lógica *BGP* da rede.

A grande desvantagem encontrada no uso do *IGen* é o fato deste não possuir integração com o *ns-3*.

2.4.2 *BRITE*

Uma das ferramentas de geração de topologias da Internet mais populares no meio acadêmico é o *BRITE (Boston University Representative Internet Topology Generator)* (MEDINA, 2001). Foi desenvolvido com o objetivo de permitir a geração de diversos modelos de topologias, adição de novos modelos, interoperabilidade com outros geradores e visualizadores de grafos. A geração de topologias pelo *BRITE* é altamente customizável

e é controlada por *scripts* de configuração. Possui uma interface gráfica que faciliata a criação destes *scripts*.

As topologias são geradas utilizando dois modelos principais.Estes modelos são os que mais aproximam-se da topologia existente na Internet. O modelo Waxman baseia-se no modelo *Erdös-Renyi* (CSORGO, 1979) de grafos aleatórios. A difereça é que todos os nós gerados são colocados em um plano e a conectividade entre nós é baseada na distância euclidiana. A probabilidade de que dois nós x e y sejam conectados é dada pela fórmula:

$$P(x, y) = \alpha \mathrm{e}^{-d/\beta L} \tag{2.1}$$

onde d é a distância do nó x ao nó y, L é o diâmetro euclidiano da rede e α e β são parâmetros.

O modelo *Barabási-Albert* pode ser usado para criar redes livre de escala. Este modelo utiliza crescimento incremental e ligação preferencial para criar topologias que estejam em conformidade com a lei de potência. Neste modelo os nós são conectados à topologia de forma incremental. Assim que cada nó é adicionado, este possui mais chances de estar conectado à um nó que seja altamente conectado. A probabilidade de um dado nó x, que esteja sendo adicionado à rede, estar conectado à um nó y, que já perteça à rede é dado pela fórmula:

$$P(x, y) = \frac{d_y}{\sum k \epsilon V d_k} \tag{2.2}$$

onde d_i é o grau do nó i e V é o conjunto de nós ja presentes no grafo. O denominador é a soma dos graus externos de todos os nós que já estão presentes no grafo.

Existem três tipos de topologias disponíveis no *BRITE*: Roteador, Sistema Autônomo *(AS)* e Hierárquica que é a combinação da topologias *AS* e Roteador. As topologias no nível roteador utilizam tanto o modelo *Waxman* quanto *Barabási-Albert*. Cada modelo possui diferentes parâmetros que afetam a criação da topologia e cada um destes estão especificados no arquivo de configuração.

Topologias hierárquicas possuem dois níveis. Primeiro há um nível superior que representa a topologia de *AS*, a qual pode ser contruída utilizando tanto o modelo *AS-Waxman* quanto o modelo *AS-Barabási-Albert*. Estes modelos são equivalentes aos do nível roteador. Após o nível dos *AS* ser construído, para cada nó existente no grafo que representa os *AS*, um nível de roteador é construído. Este nível pode ser criado com modelos e parâmetros diferentes dos utilizados na criação dos *AS*.

Após a topologia ser criada, são atribuídos valores de largura de banda aos enlaces. É possível realizar esta atribuição utilizando quatro distribuições: constante, uniforme, exponencial e *heavy-duty*.

2.4.2.1 BRITE E NS-3

A classe *BriteTopologyHelper* é utilizada para o funcionamento do *BRITE* no *ns-3*. O usuário passa como parâmentro para o construtor da classe uma *string* que representa o caminho para o arquivo de configuração do *BRITE*. Os usuários podem passar também arquivos semente para serem utilizados na construção da topologia. O *BRITE* utiliza geradores de números pseudo-aleatórios e sementes são necessárias para criar topologias diferentes que contenham as mesmas características.

Após o objeto BriteTopologyHelper ser criado a topologia pode então ser criada. Isto é realizado pelo método *BuildBriteTopology*. Este método aceita como parâmetro um objeto da classe *InternetStackHelper*. Esta pilha é instalada em todos os nós pertencentes à topologia. Caso seja criado mais de um *AS* cada um desses recebe um valor inteiro que o identifica.

Ao criar a topologia, o *BRITE* classifica os nós baseado na conectividade destes na rede. O nós no nivel de roteador são classificados como : nenhum, folha, borda, *backbone* e *stub*. Os nós classificados como folha podem ser utilizados para conectar outras topologias criadas no *ns-3* e até mesmo outras topologias criadas pelo *BRITE*. O método *GetLeafNodeForAs* da classe *BriteTopologyHelper* aceita como parâmetro o número do *AS* e um índice do nó folha.

Os endereços *IP* são atribuídos aos nós da topologia criada através dos métodos *AssignIpv4Adresses* ou *AssignIpv6Adress*, pertencentes ao *helper*. Quando os endereços *IP* são atribuídos às interfaces, cada enlace ponto-a-ponto é tratado como uma rede separada.

2.5 GERADORES DE TRÁFEGO

2.5.1 HARPOON

O *Harpoon* (SOMMERS, 2004) utiliza uma série de parâmetros que podem ser extraídos de *traces NetFlow* para gerar fluxos *TCP* e *UDP* que apresentem as mesmas características estatísticas existentes em *traces* de Internet. O *Harpoon* possui uma abordagem estatística, onde após analizar os *traces* de entrada, geram as distribuições que represen-

tam o tamanho e o tempo entre pacotes. Após esta análise gera um novo trace utilizando as distribuições geradas. O *Harpoon* pode ser utulizado para gerar tráfegos de fundo representativos para aplicações que testem protocolos ou para testar *switch* de rede por exemplo. Não possui suporte para o *ns-3*.

2.5.2 *TMIX*

Um gerador de tráfego sintético que pode ser utilizado como um módulo do *ns-3*. Seu funcionamento é calcado na observação de que mesmo que existam vários protocolos de aplicação trocando mensagens através de uma rede, todo esse tráfego pode ser visto como trocas de pacotes entre *hosts* da rede que seguem algum padrão lógico. Através deste fato é possível converter *traces* em um modelo de tráfego independente de aplicações.

O *Tmix* (WEIGLE, 2006) trabalha com o protocolo *TCP*, não importando quais protocolos das camadas superiores estão sendo utilizados. Esta é um característica importante pois permite a geração do tráfego de ataque e de fundo com facilidade e ao mesmo tempo riqueza de detalhes. Um exemplo é um *host* que ao mesmo tempo que esteja infectado por um *bot* esteja realizando um acesso lícito à vítima. Este *host* produz tanto tráfego de ataque quanto de fundo. Com o uso do *Tmix*, para modelar este comportamento, basta possuir o *trace* de um nó executando esta mesma atividade.

2.5.2.1 VETORES DE CONEXÃO

O modelo utilizado pelo *Tmix* para a geração de tráfego é construído pela análise dos cabeçalhos dos pacotes existentes em um *trace*. Este *trace* é compilado de forma "reversa" em uma representação abstrata e de alto nível que captura a dinâmica dos protocolos da camada de aplicação utilizados por dois *hosts*. Cada conexão *TCP/IP* da rede é representada como um simples vetor de conexão a-b-t. O vetor de conexão modela como uma aplicação usa esta conexão *TCP* para uma série de trocas de unidades de dados entre o iniciador da conexão ("a") e o receptor ("b"). As unidades de dados modeladas não são pacotes ou segmentos *TCP*, estas unidades correspondem aos objetos (arquivos, por exemplo) ou elementos dos protocolos da camada de aplicação (requisições *HTTP GET* ou mensagens *SMTP HELLO*, por exemplo). O tamanho das unidades de dados (*application-data units - ADU*) depende apenas do protocolo da camada de aplicação e são independentes dos tamanhos as unidades de da da camada de transporte e inferiores (HERNÀNDEZ-CAMPOS, 2004).

Por exemplo, requisições e respostas *HTTP* dependem do tamanho dos cabeçalhos definidos pelo protocolo *HTTP* e o tamanho dos arquivos referenciados, mas não pelos tamanhos dos segmentos *TCP* usados na camada de transporte. As unidades de dados que são trocadas são separadas mor intervalos de tempo (t) que representam os tempos de processamento ou o tempo de interação do usuário com as páginas. Uma sequência destas trocas constitui um vetor de conexão.

Por exemplo, em uma conexão *TCP* entre um servidor *web* e um navegador, o comportamento em relação ao tempo pode ser representado pela figura 2.3 . Um navegador faz uma requisição a um servidor que responde com o objeto requisitado.

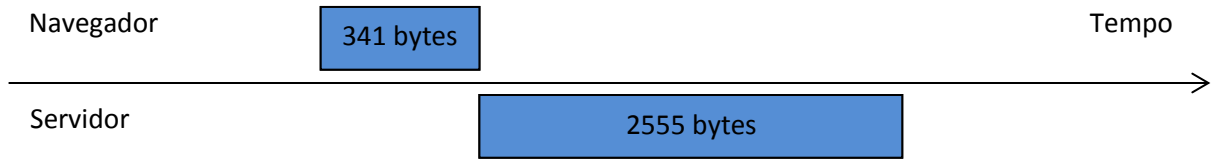

FIG. 2.3: Padrão de troca de *ADU* em uma conexão *HTTP* 1.0. Fonte: (HERNÀNDEZ-CAMPOS, 2004)

Outro padrão comum de conexões *TCP* existe quando protocolos de aplicação trocam múltiplas *ADU* entre os *hosts* de uma conexão lógica (ex: *HTTP/1.1, SMTP, NNTP,* etc.).Este padrão é mostrado na figura 2.4. Em adição ao tamanho das *ADU* representados por a_i e b_i, são introduzidos também uma variável de tempo, t_i, e representa o tempo entre as trocas de *ADU*. Este tempo representa tanto o tempo do usuário interagir com a página quanto o tempo de processamento do protocolo da camada de transporte.

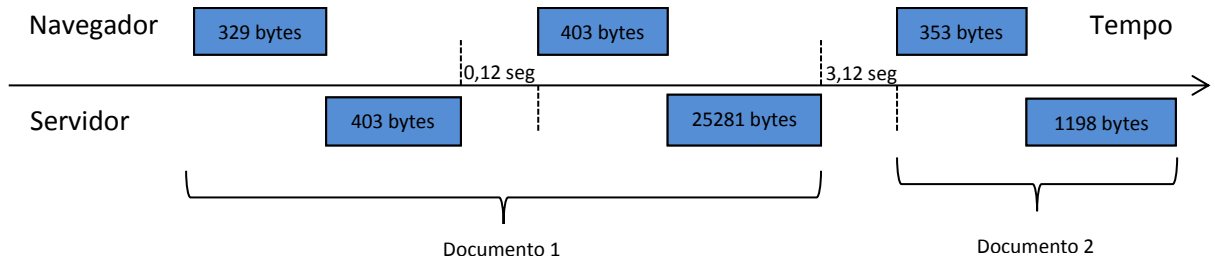

FIG. 2.4: Padrão de troca de *ADU* em uma conexão *HTTP* 1.1. Fonte: (HERNÀNDEZ-CAMPOS, 2004)

Mais formalmente, um padrão de troca de *ADU* é representado como um vetor de conexão $C_i =< E_1, E_2, ..., E_k >$ consistindo de um conjunto de *epochs* $E_i = (a_i, b_i, t_i)$

onde a_i é o tamanho do i-ésimo ADU enviado do iniciador para conectar o receptor, b_i é o tamanho do i-ésimo ADU enviado do receptor para o iniciador e t_i é o tempo de interação com a página' ou de processamento da recepção do i-ésimo ADU de resposta e a transmissão da (i+1)-ésima requisição.

Se esta análise for aplicada ao *trace* representado pela figura 2.4, seria produzido o vetor de conexão: $C_i =< (329, 403, 0, 12), (403, 25821, 3, 12)e(356, 1198, \perp)$

Um vetor de conexão pode ser síncrono ou assíncrono. Um *cvec* síncrono é aquele onde uma nova solicitação é feita somente após a chegada de uma resposta. O modelo de *cvec* assíncrono não necessita deste sincronismo e os nós enviam suas mensagens de forma independente. A figura 2.5 é um exemplo de um arquivo contendo vetores de conexão.

```
S 6851 1 21217 555382      # inicia no tempo 6.851 ms e contém 1 troca (epoch)
w 64800 6432               # tamanho da janela (bytes)
r 1176194                  # min RTT
l 0.000000 0.000000        # taxa de perda
I 0 0 245                  # Iniciador envia 245 bytes
A 0 51257 510              # receptor aguarda 51.257 ms depois de receber e envia 510 bytes
A 6304943 0 0              # receptor espera 6.3 sec depois de enviar e então envia um FIN
C 1429381 2 2 26876 793318 # inicia no tempo 1.4 s, iniciador envia 2 ADUs e receptor envia 2 ADUs
w 65535 5840               # tamanho da janela (bytes)
r 36556                    # min RTT
l 0.000000 0.000000        # taxa de perda
I 0 0 222                  # Iniciador envia 222 bytes
A 0 0 726                  # Receptor envia 726 bytes
I 62436302 0 16            # Iniciador espera 62 sec e envia 16 bytes
A 62400173 0 84            # Receptor aguarda 62 sec e envia 84 bytes
I 725 0 0                  # Iniciador aguarda 725 us e envia um FIN
A 130 0 0                  # Receptor aguarda 130 us e envia um FIN
```

FIG. 2.5: Exemplo de vetor de conexão.

Vetores de conexão sequenciais e assíncronos são diferenciadas pela *string* que começa na primeira linha: "S" para uma conexão sequencial e "C" para uma conexão assíncrona. O segundo campo na primeira linha dá o tempo inicial para a conexão em microssegundos do tempo "0" (o início da simulação). Para conexões seqüenciais, o terceiro campo indica o número de *epochs*, presentes na conexão. Para ligações assíncronas, o terceiro campo indica o número de *ADUs* enviadas pelo iniciador, e o quarto campo indica o número de *ADUs* que o receptor envia. Os dois campos finais na primeira linha são os números de identificação que não são utilizados na implementação do Tmix para o ns-3.

A segunda linha, começando com "w", dá os tamanhos de janela do iniciador e do receptor, respectivamente, em bytes. A terceira linha, começando com "r", dá o RTT [3] mínimo em microssegundos entre o iniciador e o receptor. A quarta linha, começando

[3]Round Trip Time

Característica	*Tmix*	*Harpoon*
TCP/UDP	*TCP*	*TCP/UDP.*
Modelo	modelo (a-b-t).	Distribuição estatística do tempo entre os pacotes e tamanho.
Trace	*tcpdump*	*NetFlow*
Suporte	*Linux*, *ns-2* e *ns-3*	Linux

TAB. 2.2: Tabela comparativa *Tmix* e *Harpoon*

com "l", proporciona as taxas de perda envolvidos em cada sentido da conexão. As linhas restantes no vetor de conexão mostram as trocas de *ADU*.

No caso de vetores de conexão sequenciais, existem dois tempos associados com o iniciador ou receptor, enquanto enviam *ADUs* um para os outros:

a) A quantidade de tempo que o iniciador/receptor tem que esperar antes de enviar o próximo *ADU* após o envio do *ADU* anterior.

b) A quantidade de tempo que o iniciador/receptor tem que esperar antes de enviar o próximo *ADU* após receber um *ADU*.

As linhas que começam com"I" denotam ações para o iniciador, e as linhas que começam com um "A" são de ações para o receptor.

A tabela 2.2 resume as características do *Harpoon* e do *Tmix*.

2.5.2.2 FUNCIONAMENTO

O *Tmix* funciona gerando tráfego entre um par de nós. Arquivos contendo vetores de conexão são passados como parâmetros de entrada e o tráfego contido nos *cvecs* é então gerado entre este par de nós dentro da simulação. A topologia existente entre esse par de nós é transparente para o *Tmix*. É criado um *socket* entre os nós do par e os pacotes são transmitidos entre eles. Para o *ns-3* o *Tmix* apresenta-se como uma *Application*. A Figura 2.6 (WEIGLE, 2006) nos apresenta este funcionamento.

Um fluxo *TCP* modelado por determinado *cvec* segue do nó n0 para o nó n1. Outro fluxo representado por outro *cvec* segue do nó n3 para o nó n2. Os nós n0 e n1 formam um par *Tmix* assim como os nós n3 e n2. Os nós n0 e n3 são chamados de iniciadores da conexão enquanto os nós n2 e n3 receptores da conexão. O par *Tmix* é formalmente representado como $< A, B >$ onde A é o iniciador do par e B é o receptor do par. Na figura 2.6 tem-se os pares $< n0, n1 >$ e $< n3, n2 >$. O *Tmix* implementa nós chamados *DelayBox*. Estes nós são necessários pois em sua arquitetura original não são previstas

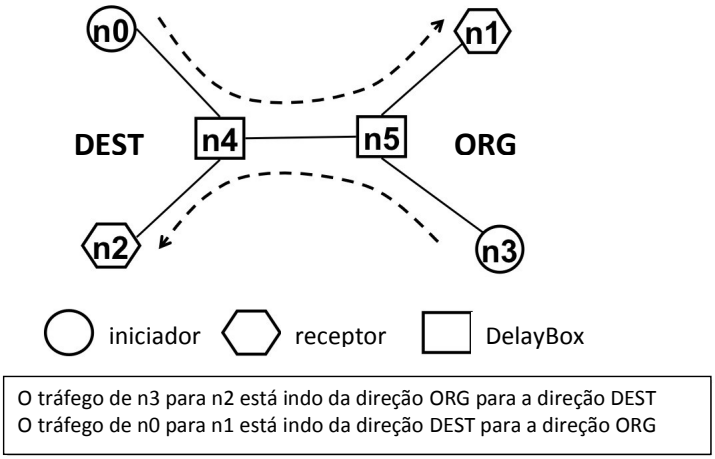

FIG. 2.6: Funcionamento do Tmix.

topologias complexas e estes representam os retardos aleatórios decorrentes de vários enlaces.

2.5.2.3 *TMIX* E *NS-3*

A interação do *Tmix* com o *ns-3* é realizada inicialmente através da criação de um par de nós do *Tmix* e de uma instância de um *DelayBox*. A instância do *DelayBox* é criada através de um *smart pointer* e do template *CreateObject*.

O responsável pela criação do par é estrutura *TmixNodePair* da classe *TmixTopology*. São passados então os nós pertencentes ao par, um como iniciador e outro como receptor. Além disso devem ser passados o *Device* do nó, o endereço deste *Device*, e o *Device* do roteador ao qual o nó está conectado. É então criado um *smart pointer TmixHelper* tendo como parâmetros um *DelayBox*, os nós e os respectivos endereços.

O vetor de conexão é então carregado em uma estrutura *ConnectionVector* por intermédio do método *ParseConnectionVector* ambos da classe *Tmix*. O tráfego é gerado utilizando-se o método *AddConnectionVector* da estrutra *TmixTopology*.

3 PLATAFORMA PROPOSTA

Este trabalho tem por objetivo permitir a modelagem de um ataque *DDoS* a um servidor *web* através de simulação. Para tanto é proposto um fluxo composto de quatro etapas representadas pela Figura 3.1. Nele são representados os macro processos necessários para que se possa realizar a geração da topologia onde ocorre o ataque *DDoS*, o tráfego existente durante o ataque, simulação propriamente dita e o seu resultado, um trace que represente um ataque.

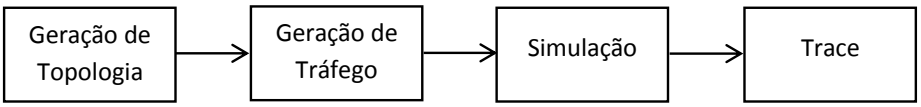

FIG. 3.1: Fluxo da Simulação

A geração de topologia deve buscar representar o ambiente onde ocorrem os ataques *DDoS* — a Internet. Além de ser necessário possuir a capacidade de gerar uma topologia com centenas e até milhares de nós, o processo de gerar a topologia deve respeitar propriedades intrínsecas existentes na Internet (FALOUTSOS, 1999).

A geração de tráfego deve permitir pelo menos a existência de dois tipos de tráfegos: tráfego de ataque (TA) e tráfego de fundo (TF). O tráfego de ataque é aquele realizado pelos integrantes da *botnet*. O tráfego de fundo é aquele produzido pelos usuários legítimos do serviço disponibilizado pelo servidor web. No mundo real existem tráfegos além dos que serão modelado no presente trabalho. Exemplos destes tráfegos são as mensagens enviadas pelo *botmaster* aos *bots* e a comunicação entre os *bots*. O foco da simulação desenvolvida é aquele que possui como destino o alvo do ataque.

3.1 DESCRIÇÃO DA PLATAFORMA SUGERIDA

O *ns-3* (HENDERSON, 2006) foi escolhido como simulador, para o gerador de topologia o *BRITE* (MEDINA, 2001) e para o gerador de tráfego o *Tmix* (WEIGLE, 2006). A figura 3.2 ilustra a plataforma utilizada para a simulação.

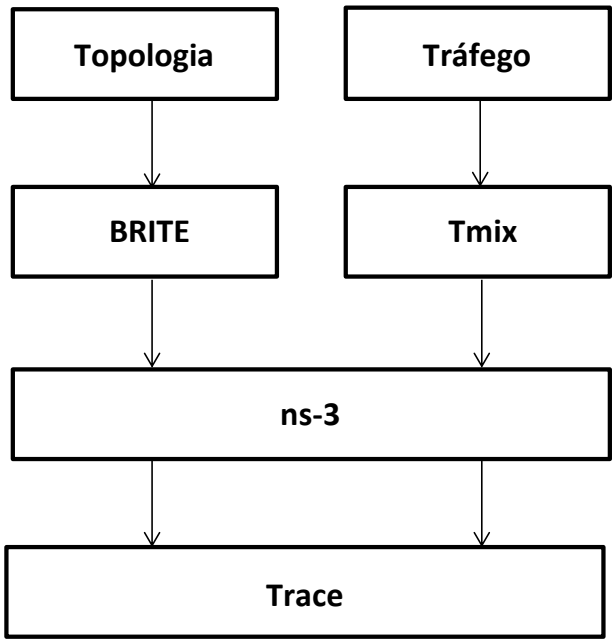

FIG. 3.2: Plataforma da simulação

3.2 IMPLEMENTAÇÃO

A plataforma proposta no presente trabalho não necessita de um *hardware* específico para seu funcionamento. Durante seu desenvolvimento foi utilizado um ambiente virtualizado através do *VirtualBox* (ORACLE CORPORATION, 2013). O sistema operacional *host* utilizado foi o *Windows 7* e o *guest*, onde ocorreu todo o desenvolvimento do trabalho, *Ubuntu Linux* 13.10. A distribuição do ns-3 utilizada foi a 3.19. Esta versão já possui o gerador de topologias *BRITE* integrado. Este ambiente foi usado apenas para a codificação e experimentos em pequena escala (poucos nós) que visam apenas a validação da plataforma.

3.2.1 *BRITE*

Como o *BRITE* apresenta-se de forma integrada ao *ns-3* não foi necessária nenhuma alteração na forma como este funciona. No trecho de código abaixo é mostrado como a topologia é construída pelo *BRITE* dentro da simulação.

```
//Path do arquivo de configuração do BRITE
std::string confFile =
    "src/brite/examples/conf_files/TD_ASBarabasi_RTWaxman.conf";
```

```
//Criação do BriteTopologyHelper recebendo como parâmetro o arquivo de
    configuração.
 BriteTopologyHelper bth (confFile);
//Número de streams utilizadas para gerar a semente utilizada pelo BRITE
 bth.AssignStreams (3);
//Aloca endereços para os nós pertencentes à topologia
 Ipv4AddressHelper address;
 address.SetBase ("10.0.0.0", "255.255.255.252");

 InternetStackHelper internet;

//Criação da Topologia
 bth.BuildBriteTopology (internet);
 bth.AssignIpv4Addresses (address);
```

3.2.2 *TMIX*

Inicialmente foi procedido um experimento visando verificar a adequação do *Tmix* a uma simulação de ataque *DDoS*. Como citado na seção 2.5.2.2, o *Tmix* gera tráfego entre um par de nós existente em uma topologia. O tráfego é gerado em ambas as direções do par. Um ataque DDoS caracteriza-se por vários nós atacantes enviando tráfego a um nó vítima. Para verificar o funcionamento do Tmix nesta situação foi realizado o experimento ilustrado na figura 3.3.

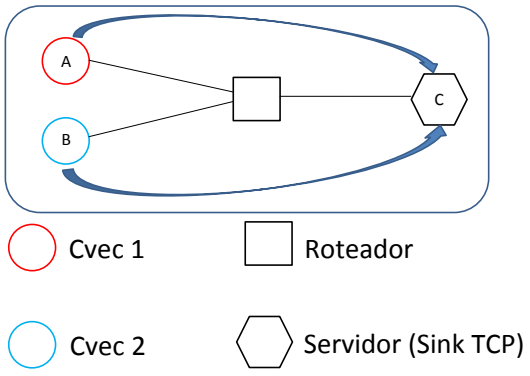

FIG. 3.3: Experimento para verificar funcionamento do *Tmix*

Na primeira versão o *Tmix* foi utilizado com sua implementação padrão em todos os

nós *hosts* da simulação (A, B e C). Os pares *Tmix* neste caso são $< A, C >$ e $< B, C >$. Esta configuração produziu erros e a simulação foi abortada antes de sua conclusão. Foi possível observar que o *Tmix* não consegue criar *sockets* entre os pares quando estes tem pelo menos um nó em comum. A simulação produzia erro na criação dos *sockets* e era abortada.

Para contornar este fato, na segunda versão, foram realizadas alterações no funcionamento do *Tmix*. A aplicação instalada no nó receptor passou a ser sempre um *PacketSink*. Um *PacketSink* é um *application* existente no *ns-3* que não cria tráfego, apenas aceita conexões e responde aos pacotes enviados por essas conexões.

Com esta modificação, o tráfego existente nos vetores de conexão é reproduzido apenas no sentido do iniciador para o receptor. Todo o tráfego na direção do receptor para o iniciador presente nos vetores de conexão é ignorado. O receptor funcionando como *PacketSink* reponde apenas aceitando as conexões, respondendo aos pacotes enviados com *ACK* e finanlizando as conexões caso ocorra *timeout*.

Esta abordagem permitiu ainda criar pares *Tmix* opostos aos originais, isto é, no caso da figura 3.3, $< C, A >$ e $< C, B >$. Neste caso C é o iniciador e possui a aplicação *Tmix* instalada e os nós A e B são receptores e possuem a aplicação *PacketSink* instalada.

O *ns-3* permite que várias aplicações sejam instaladas no mesmo nó. Então quando dados são enviados dos nós A e B para o nó C, A e B comportam-se como nós *Tmix* e C como um *PacketSink*. Quando dados são enviados de C para A e B, C é um nó *Tmix* e A e B nós *PacketSink*.

No trecho de código abaixo é exemplificado como funciona a criação de um par $< A, C >$. O mesmo processo deve ser utilizado para gerar o tráfego entre o par $< C, A >$.

```
//Criação dos nós da simulação. O nó 0 é o roteador.
NodeContainer c;
c.Create (4);

//Criação dos NodeContainer que possuem o nó e o roteador ao qual ele
   encontra-se conectado.
NodeContainer nAn0 = NodeContainer (c.Get (1), c.Get (0));
NodeContainer nCn0 = NodeContainer (c.Get (2), c.Get (0));

InternetStackHelper internet;
```

```
internet.Install (c);

//Criação dos Devices
PointToPointHelper p2p;
p2p.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
p2p.SetChannelAttribute ("Delay", StringValue ("2ms"));
NetDeviceContainer dAd0 = p2p.Install (nAn0);
NetDeviceContainer dCd0 = p2p.Install (nCn0);

//Endereçamento dos nós
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer iAi0 = ipv4.Assign (dAd0);
ipv4.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer iCi0 = ipv4.Assign (dCd0);

//Criação do DelayBox. Apesar de não ser utilizado deve ser passado como
   parâmetro.
Ptr<DelayBox> m_delayBox;
m_delayBox = CreateObject<DelayBox> ();

//Criação do par Tmix
TmixTopology::TmixNodePair pair;
pair.initiator = c.Get (1);
pair.acceptor = c.Get (2);
pair.initiatorDevice = StaticCast<PointToPointNetDevice> (dAd0.Get (0));
pair.acceptorDevice = StaticCast<PointToPointNetDevice> (dCd0.Get (0));
pair.initiatorAddress = iAi0.GetAddress (0);
pair.acceptorAddress = iCi0.GetAddress (0);
pair.routerInitiatorDevice = StaticCast<PointToPointNetDevice> (dAd0.Get (1));
pair.routerAcceptorDevice = StaticCast<PointToPointNetDevice> (dCd0.Get (1));

//Criação do helper
 pair.helper = Create<TmixHelper> (m_delayBox, pair.initiator,
     pair.initiatorAddress, pair.acceptor, pair.acceptorAddress);
```

```
//Passagem do vetor de conexão e geração do tráfego entre o par <A, C>
double chance = 1.0;
Tmix::ConnectionVector cvec;
while (Tmix::ParseConnectionVector(cvecfileA, cvec))
{
   if (UniformVariable().GetValue() < chance)
     {
       pair.helper->AddConnectionVector(cvec);
     }
 }
```

3.2.3 VETORES DE CONEXÃO

A transformação de *traces* em vetores de conexão realizada com a utilização do *a-b-t model* também não atendeu a modelagem de um ataque *DDoS*. Esta transformação baseia-se na reconstrução de *ADU's* e muitos dos tráfegos de ataque por exemplo não permitem a remontagem destas estruturas. Além disso, o *a-b-t model* cria vetores de conexão que representam dados enviados tanto pelos iniciadores quanto pelos receptores.

Com as alterações realizadas no funcionamento do *Tmix* descritas na seção 3.2.2, os vetores de conexão gerados devem apenas apresentar os dados enviados pelo iniciador para o receptor. Desta forma, o *trace* que dará origem aos vetores de conexão deve ser dividido em duas partes.

Por exemplo, na figura 3.3, de posse do *trace* que representa o tráfego entre os nós A e C, deve-se dividi-lo de forma que tenha-se um *trace* que possua apenas pacotes como origem o endereço *IP* de A e destino o endereço *IP* de C. Estes vetores de conexão serão utilizados para gerar o tráfego no par $< A, C >$. O segundo *trace* deverá possuir apenas os pacotes que tenham como endereço *IP* de origem igual a C e endereço *IP* de destino igual a A. Este *trace* será utilizado para gerar o tráfego do par $< C, A >$.

Foi desenvolvido um programa que cria os vetores de conexão a partir de um *trace* mas sem remontar os *ADU*. Isto é, em cada *epoch* (seção 2.5.2.1) $E_i = (a_i, b_i, t_i)$ teremos a_i com o valor do tamanho do dado enviado em determinado pacote em *bytes*. O valor de b_i será sempre igual a zero. O valor de t_i é igual a diferença entre o *timestamp* do pacote em relação ao *timestamp* primero pacote do vetor de conexão em microssegundos.

O programa desenvolvido para criar os vetores de conexão foi implementado em *Java* e a biblioteca utilizada foi a *jNetPcap* (SLY TECHNOLOGIES, 2013). Esta biblioteca possui a função de ao receber um arquivo *pcap* realizar a separação em fluxos *TCP* e percorrer cada fluxo pacote por pacote. O bloco de código abaixo apresenta o programa implementado.

```
try {
    //Realiza o mapeamento do arquivo .pcap e o separa em fluxos TCP
    JFlowMap map = new JFlowMap();
    pcap.loop(Pcap.LOOP_INFINITE, map, null);
    System.out.println(map.toString());
    Set<JFlowKey> setKey = map.keySet();
    Iterator<JFlowKey> it = setKey.iterator();
    File file = new File("cvec.txt");
    if (!file.exists()) {
        file.createNewFile();
    }
    FileWriter fw = new FileWriter(file.getAbsoluteFile());
    BufferedWriter bw = new BufferedWriter(fw);
    //Realiza iteração para cada fluxo
    while (it.hasNext()){
        //Variáveis utilizadas para controle
        i = 0;
        j = 0;
        k = 0;
        JFlowKey key = it.next();
        JFlow flow = map.get(key);
        List<JPacket> forward = flow.getForward();
        // Percorre cada pacote presente no fluxo
        for (JPacket p : forward) {
            i++;
            if (j==1){
                break;
            }
            Tcp tcp = new Tcp();
```

```
    if (p.hasHeader(Tcp.ID)){
        p.getHeader(tcp);

        long timeStamp = (p.getCaptureHeader().timestampInMicros());
        //Calcula o tempo de início do vetor de conexão em relação ao
            tempo de início da simulação
        long timeRelative = Math.abs(timeStamp-antTempoCvec);
        //Verifica se é o primeiro pacote do fluxo
        if (i == 1){
            bw.write("C "); bw.write(Long.toString(timeRelative) + " ");
            antTempo = p.getCaptureHeader().timestampInMicros();
            bw.write(Integer.toString(flow.size()) + " ");
            bw.write("0 ");
            //Os valores do tamanho da janela, RTT e taxa de perda não
                são utilizados pois não é
            //feito uso do DelayBox
            bw.write(Integer.toString(numSeq++) + "
                ");bw.write(Integer.toString(numSeq++) + "\n");
            bw.write("w 64800 6432" + "\n");
            bw.write("r 1118156" + "\n");
            bw.write("l 0.000000 0.000000" + "\n");
        }
    }
//Caso seja o primeiro pacote este inicia no mesmo tempo de início do
    vetor de conexão.
    if(k==0){
        //Verifica se o pacote contém dados a serem enviados.
        if(tcp.getPayloadLength() != 0){
            bw.write("I ");
            bw.write("0 ");
            bw.write("0 ");
            bw.write(Integer.toString(tcp.getPayloadLength()) + "\n");
            //Salva o timestamp do pacote
            antTempo = p.getCaptureHeader().timestampInMicros();
            k++;
```

```
  }
}else{
    //Verifica se o pacote possui dados
    if(tcp.getPayloadLength() != 0){
        bw.write("I ");
        //Escreve no arquivo o tempo de envio do pacote em relação
            ao anterior
        bw.write(Long.toString((p.getCaptureHeader().timestampInMicros())-
            antTempo) + " ");
        bw.write("0 ");
        bw.write(Integer.toString(tcp.getPayloadLength()) + "\n");
        //Salva o tempo do pacote
        antTempo = p.getCaptureHeader().timestampInMicros();
        k++;
    }
}
//Caso seja um pacote com a flag FIN, cria um epoch sem envio de
    dados.
if (tcp.flags_FIN()){
    //Verifica se é o primeiro pacote do vetor de conexão
    if(k==0){
        bw.write("I ");
        //Como é o primeiro pacote do vetor de conexão é enviado
            logo após o início do mesmo.
        bw.write("0 ");
        bw.write("0 ");
        bw.write("0 " + "\n");
        //salva o tempo do pacote.
        antTempo = p.getCaptureHeader().timestampInMicros();
        j=1;
        break;
    }else{
        bw.write("I ");
        //Escreve no arquivo a diferença do tempo em relação ao
            pacote anterior.
```

```
                    bw.write(Long.toString((p.getCaptureHeader().timestampInMicros())-
                        antTempo) + " ");
                    bw.write("0 ");
                    bw.write("0 " + "\n");
                    //Salva o tempo do pacote.
                    antTempo = p.getCaptureHeader().timestampInMicros();
                    j=1;
                    break;
                }
            }
        }
    } bw.close();
}catch (IOException e) {
    e.printStackTrace();
} finally {
    pcap.close();
}
```

3.2.4 SIMULAÇÃO

Em resumo, para que a simulação proposta seja realizada, o usuário deve primeiramente utilizar um *script* de configuração do *BRITE* ou então gerar um novo. Deve então selecionar os *traces* responsáveis pelos tráfegos de ataque de fundo. Os *traces* de tráfego de fundo são produzidos através de *hosts* realizando acesso normal ao site cujo ataque será simulado. Os *traces* responsáveis pelo ataque são obtidos através de *hosts* com ferramentas de ataque instaladas. Pode-se ter um ou vários *traces* tanto de tráfego de fundo quanto de ataque. Todos os *traces* precisam apenas ser coletados nos *hosts* responsável por gerar os *traces*.

Após a coleta os *traces* devem ser separados como explicado na seção 3.2.3. Os *traces* divididos são então utilizados para gerar o tráfego entre os vários pares da simulação. O produto da simulação são arquivos pcap de qualquer nó que pertença a simulação, inclusive aquele alvo do ataque.

A simulação obtida é mostrada na figura 3.4. Tem-se nesta figura nós ou topologias ligadas a roteadores. Estes por sua vez estão ligados a roteadores folha de uma topologia

criada pelo *BRITE*. Cada um dos nós forma pares *Tmix* com o nó vítima do ataque.

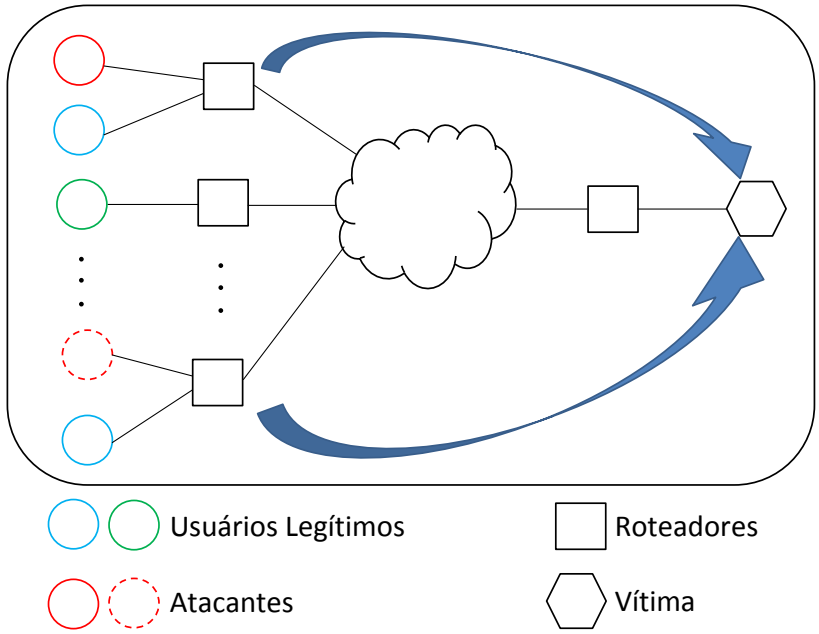

FIG. 3.4: Simulação

4 EXPERIMENTOS REALIZADOS

Com o objetivo de validar as alterações realizadas no funcionamento do *Tmix* foi realizado um experimento real e o seu resultado (*trace* no nó vítima) foi comparado com o mesmo experimento realizado em ambiente simulado.

4.1 DESCRIÇÃO DO EXPERIMENTO

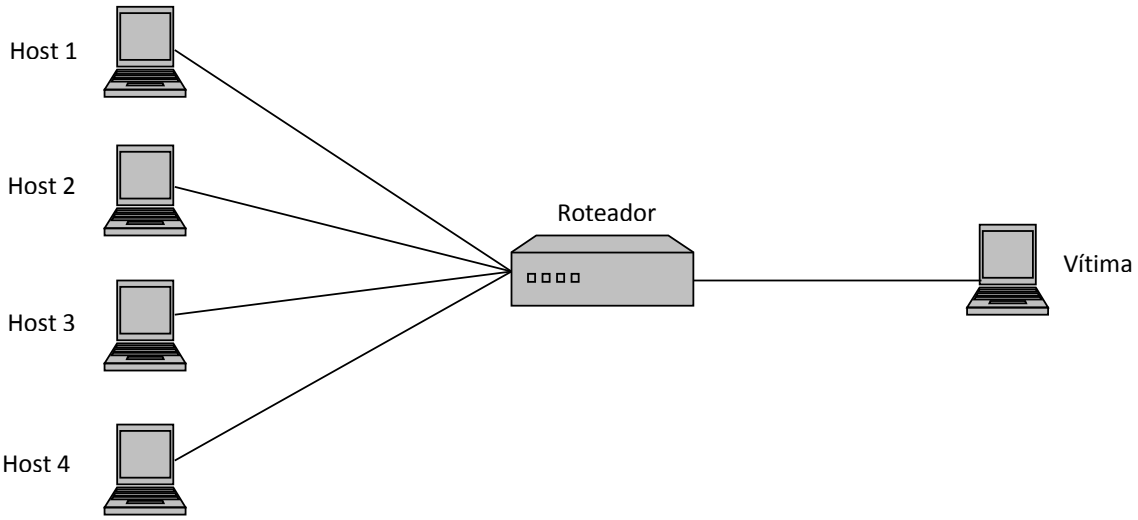

FIG. 4.1: Experimento de validação

A topologia utilizada para realizar o experimento é a mesma da figura 4.1. O experimento foi realizado utilizando-se as seguintes configurações de *hardware* e *software*:

- *Host* 1: processador intel core i3, com 4 Gb de memória *RAM* e sistema operacional *windows* 7.
- *Host* 2: processador intel core i3, com 4 Gb de memória *RAM* e sistema operacional *windows* 7.
- *Host* 3: processador intel core i5, com 4 Gb de memória *RAM*, sistema operacional *windows* 7 e com a ferramenta de ataque *LOIC* instalada.

- *Host* 4: processador intel core i5, com 4 Gb de memória *RAM*, sistema operacional *windows* 8 e com a ferramenta de ataque *LOIC* instalada.

- Vítima: processador intel core i5, com 4 Gb de memória *RAM*, sistema operacional *windows* 7 e Servidor *web Apache HTTP Server Version* 2.2.

Na vítima foi espelhado o site http://www.comp.ime.eb.br. O tráfego de fundo foi gerado realizando-se acessos através da interação de pessoas com o *site* espelhado. O tráfego de ataque foi gerado utilizando-se a ferramenta *LOIC* e a forma de ataque *HTTP*.

Os *hosts* 1 e 2 geraram apenas tráfego de fundo. O *host* 3 gerou tráfego de fundo e de ataque. O host 4 gerou apenas tráfego de ataque. Ambos ataques foram inciados 11 segundos após o início do experimento. Todos os *traces* foram coletados utilizando-se a ferramenta *Wireshark* (WIRESHARK, 2014). Foram coletados cinco traces, cada um pertencente à interface de rede de cada *host* do experimento. Os traces foram filtrados de forma que contivessem apenas tráfego direcionado para a porta 80 *TCP* da vítima.

Após isso a mesma topologia foi reproduzida no ambiente da plataforma de simulação. Os *traces* pertencentes aos *hosts* de 1 a 4 foram então divididos da forma citada na seção 3.2.3. Foram gerados os vetores de conexão de cada *trace* e simulação executada. O *trace* do nó vítima simulado foi então comparado com o mesmo trace do experimento real.

4.2 AVALIAÇÃO DOS RESULTADOS

Para verificar o desempenho da simulação foram utilizadas três características dos *traces*:

- Throughput: quantidade total de dados que passa pela interface de rede em determinado período de tempo.

- Tempo entre pacotes: Intervalo de tempo entre a transmissão de dois pacotes.

- Pacotes em trânsito: Pacotes que foram transmitidos mas ainda não receberam confirmação de recebimento (*ACK*).

Os dados foram extraídos dos traces com a utilização da ferramenta *captcp* disponível no site http://research.protocollabs.com/captcp. Os valores do experimento real foram subtraídos dos valores da simulação e o resultado foi analisado utilizando o técnica *4-plot* (SEMATECH, 2012).

A análise *4-plot* utiliza-se de quatro gráficos visando avaliar suposições existentes sobre determinado grupo de dados. Os gráficos utilizados são:

- *Run sequence*: é um gráfico que apresenta os dados observados em uma sequência de tempo. Avalia a variação do conjunto de dados.
- *Lag*: avalia a aleatoriedade do conjunto de dados. Dados aleatórios não devem exibir qualquer estrutura identificável no gráfico.
- *Histogram*: sumariza graficamente a distribuição de um conjunto de dados.
- *Normal probability*: avalia se os dados são distribuídos de forma normal. Confirma a avaliação realizada pelo gráfico *Histogram* caso este assemelhe-se a uma distribuição normal.

Os gráficos do *4-plot* foram construídos através da ferramenta *Matlab R2013a* (MATLAB, 2013).

4.2.1 *THROUGHPUT*

A análise do *throughput* demonstrou existir uma um *gap* entre o experimento real e o simulado. Tal diferença já era esperada pois a mesma é descrita no próprio trabalho original do *Tmix*. Apesar deste *gap* é possível observar no gráfico da figura 4.2 que a simulação acompanha as variações do *throughput* do experimento real.

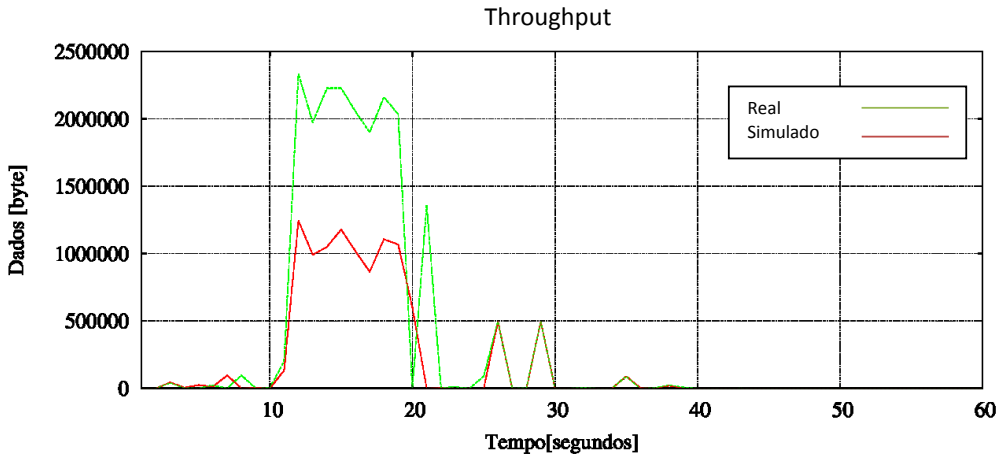

FIG. 4.2: Throughput

4.2.1.1 *4-PLOT*

O gráfico da figura 4.3 apresenta a análise do throughput através da técnica *4-plot*.

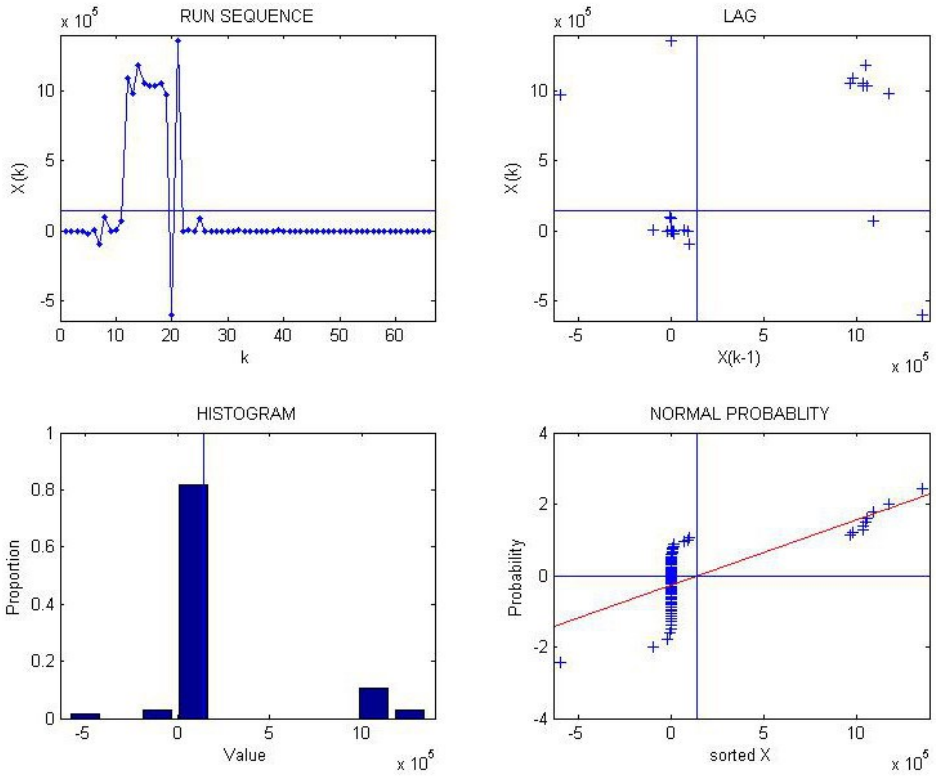

FIG. 4.3: Análise 4-plot do gráfico de Throughput

- *Run sequence*: reproduziu o *gap* existente entre o experimento real e simulado. Confirmou o fato de que apesar da diferença das medições a simulação possui a variação do throughput bem próxima ao tráfego real.
- *Lag*: mostra duas áreas aleatórias separadas, uma referente ao trecho de *gap* e outra referente ao restante do gráfico.
- *Histogram*: A diferença de throughput não aparenta ser uma distribuição normal. Resultado esperado pela existência do *gap*.
- *Normal probability*: Confirma que não se trata de uma distribuição normal.

Pela análise *4-plot* do *throughput* foi confirmado que apesar da existência de um *gap* entre os dados reais e simulados estes possuem as mesma variações e reproduz de forma aceitável o comportamento do experimento real.

4.2.2 TEMPO ENTRE PACOTES

A análise do gráfico de tempo entre pacotes demonstra que a simulação possui um comportamento próximo ao experimento real. O gráfico da figura 4.4 apresenta grande quantidade de ruído e esta hipótese deve-se confirmar com o uso do *4-plot*.

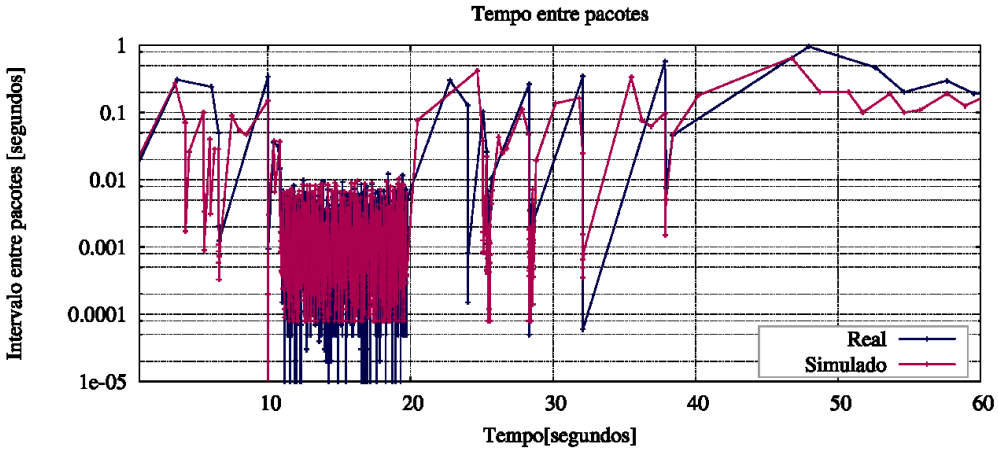

FIG. 4.4: Tempo entre pacotes

4.2.2.1 *4-PLOT*

O gráfico da figura 4.5 apresenta a análise do tempo entre pacotes através da técnica *4-plot*.

- *Run sequence*: indica que não há nenhuma mudança significante na localização ou na escala dos dados durante o tempo.
- *Lag*: indica que os dados são aleatórios.
- *Histogram*: os dados são simétricos e é possível assumir que aproxima-se de uma distribuição normal.
- *Normal probability*: confirma que a aproximação de uma distribuição normal é válida.

Pela análise *4-plot* do tempo entre pacotes podemos concluir que a diferença entre a simulação e o experimento pode ser modelada na forma $Y_i = C + E_i$ onde C é uma constante e E_i é um erro aleatório existente para cada amostra da diferença entre o

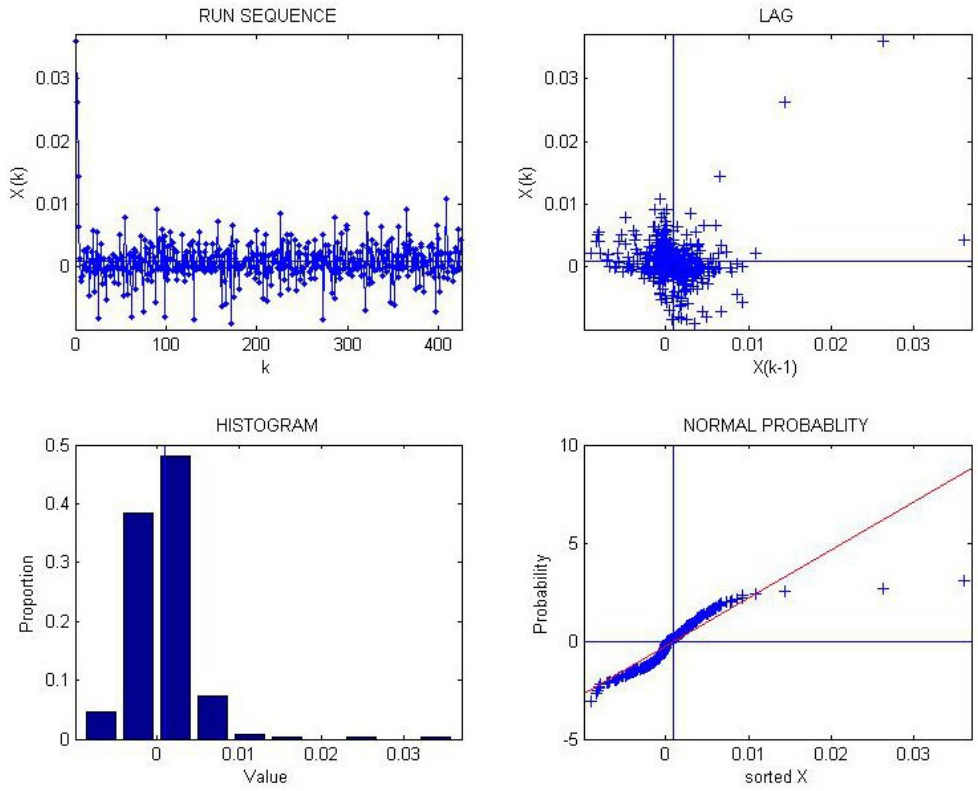

FIG. 4.5: Análise 4-plot do gráfico de tempo entre pacotes

experimento e simulação. Pode-se então caracterizar a diferença com sendo um "ruído branco".

4.2.3 PACOTES EM TRÂNSITO

A análise do gráfico de pacotes em trânsito demonstra que a simulação possui um comportamento próximo ao experimento real. O gráfico da figura 4.6 apresenta grande quantidade de ruído e esta hipótese deve-se confirmar com o uso do *4-plot*.

4.2.3.1 *4-PLOT*

O gáfico da figura 4.7 apresenta a análise dos pacotes em trânsito através da técnica *4-plot*.

- *Run sequence*: indica que não há nenhuma mudança significante na localização ou na escala dos dados durante o tempo.

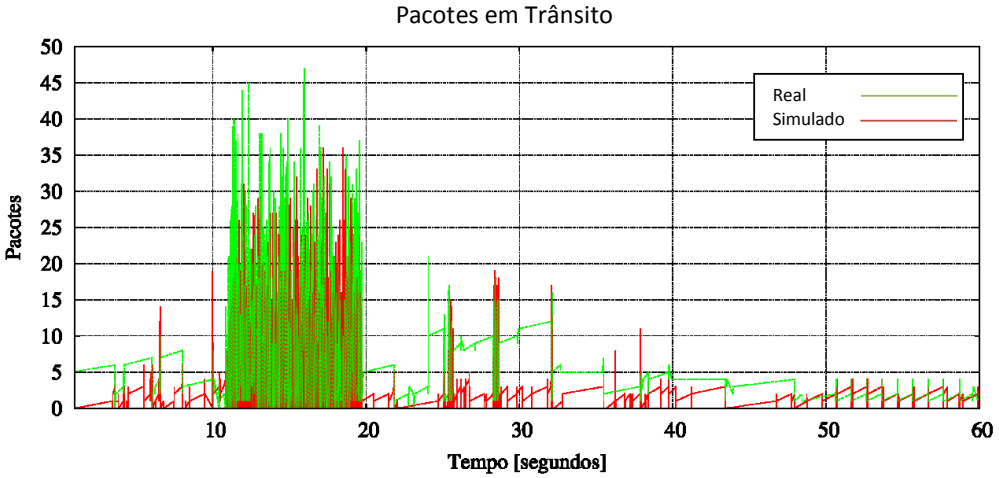

FIG. 4.6: Pacotes em trânsito

- *Lag*: Demostra existir forte linearidade. Este resultado era esperado pois a quantidade de pacotes em trânsito em determinado tempo t_i depende fortemente da quantidade de pocaotes em trânsito em t_{i-1}.

- *Histogram*: como não existe aleatoriedade na amostra, não é avaliada sua normalidade e o gráfico é desconsiderado.

- *Normal probability*: Pelo mesmo motivo citado anteriormente, a normalidade da amostra não é avaliada.

Pela análise *4-plot* dos pacotes em trânsito podemos concluir que a diferença entre a simulação e o experimento pode ser modelada de forma linear $Y_i = A_0 + A_1 * Y_{i-1} + E_i$. Esta função exprime a forte correlação existente entre o número de pacotes em trânsito atual, Y_i com os pacotes em trânsito existentes anteriormente Y_{i-1}.

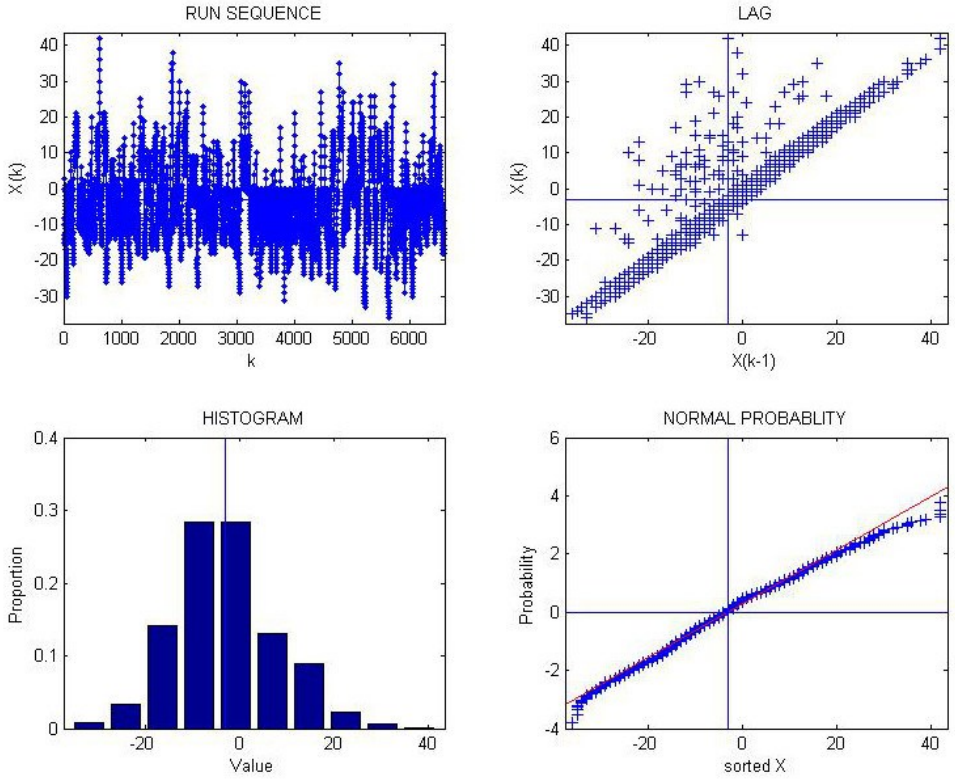

FIG. 4.7: Análise 4-plot do gráfico de pacotes em trânsito

5 CONSIDERAÇÕES FINAIS

Com o crescimento da quantidade e qualidade de ataques *DDoS* a capacidade se reproduzir este fenômeno é essencial para o desenvolvimento na área de sua mitigação. Apesar de uma grande gama de simuladores de redes, a maioria destes não permite a geração de grandes topologias e tráfegos diferenciados.

Na proposta apresentada, foi buscado tornar a tarefa de simular um *DDoS* mais realista. A utilização do *BRITE* em conjunto com o *ns-3* permite escalabilidade aos experimentos. O *BRITE* permite que topologias póximas do ambiente onde os ataques normalmente acontecem, a Internet. O uso do *Tmix* possibilita a simulação dos mais variados tipos de ataques e tráfegos de fundo, sendo necessário para isso apenas o *trace* de um *host* executando a característica desejada.

O experimento realizado validou o funcionamento do *Tmix* com as alterações realizadas em seu funcionamento original. A análise da diferença do throughput, tempo entre pacotes e pacotes em trânsito através da técnica *4-plot* foi realizada e seus resultados satisfatórios.

5.1 TRABALHOS RELACIONADOS

O *ReaSE* (GAMER, 2008) é um dos projetos cuja estrutura mais se aproxima da plataforma proposta. Ele utiliza um gerador de topologias desenvolvido pelos próprios autores. Para a geração do tráfego foram utilizadas distrubuições *heavy-tail* tanto para o tamanho dos pacotes quanto para o intervalo entre os mesmos.

Em (KOTENKO, 2006) foi desenvolvido o ambiente de simulação *DDoSSim*. O objetivo é a simulação de ataques DDoS e teste de técnicas de defesa na própria ferramenta. A ferramenta de simulação utilizada foi o *OMNET++*, também um simulador de eventos discretos como o *ns-3*. Uma abordagem relacionada a agentes foi utilizada onde estes foram separados em dois grupos: agentes responsáveis pelo ataque e agentes responsáveis por identificar o ataque. O *Inet framework* (JIN, 2000) foi utilizado para gerar as topologias da simulação. A abordagem para a geração do tráfego dos nós da simulação baseou-se em um período para coleta de informações de *traces* para criação de modelos estatísticos.

Em (BALA, 2013) é proposto realizar a simulação de ataques DDoS utilizando o *SimEvents* do *MATLAB*. Apesar do mesmo objetivo, não foi utilizado um simulador

desenvolvido com a finalidade da simulação de redes. Também não foi feito uso de nenhum gerador de topologias. Não foi feito uso de nenhum gerador de tráfego. Os pacotes são criado em certos momentos de forma aleatória e em outros utilizando uma distribuição exponencial.

5.2 TRABALHOS FUTUROS

Atualmente o presente trabalho não contempla algumas características relevantes que podem estar presentes em futuras implementações e pesquisas. A distribuição dos *bots* é de responsabilidade do usuário. Não existe a figura do *botmaster* representada na simulação e consequentemente não há reprensetação das mensagens entre este e os *bots*. Também não foi previsto nenhum mecanismo de troca de mensagem entre os *bots*.

O *Tmix* funciona apenas com o protocolo *TCP*. Como consequência, o programa que transforma *traces* no formato pcap em vetores de conexão trabalha apenas com pacotes *TCP*. O funcionamente do *Tmix* para o protocolo *UDP* e o consequente programa que trabalhe com este protocolo devem ser estudados.

A comparação da plataforma proposta com os travalhos citados em 5.1 também é desejável em futuras pesquisas.

6 REFERÊNCIAS BIBLIOGRÁFICAS

ABU RAJAB, M., ZARFOSS, J., MONROSE, F. e TERZIS, A. A multifaceted approach to understanding the botnet phenomenon. págs. 41–52, 2006.

AKELLA, A., BHARAMBE, A., REITER, M. e SESHAN, S. Detecting ddos attacks on isp networks. Em *Proceedings of the Twenty-Second ACM SIGMOD/PODS Workshop on Management and Processing of Data Streams*, págs. 1–3, 2003.

BALA, A. e OSAIS, Y. Modelling and simulation of ddos attack using simevents. *International Journal of Scientific Research in Network Security and Communication*, 1 (02):5–14, 2013.

CSORGO, S. Erdos-renyi laws. *The Annals of Statistics*, págs. 772–787, 1979.

DIETRICH, S., LONG, N. e DITTRICH, D. Analyzing distributed denial of service tools: The shaft case. Em *LISA*, págs. 329–339, 2000.

DITTRICH, D. The dos project's 'trinoo' distributed denial of service attack tool. 1999a.

DITTRICH, D. The stacheldraht distributed denial of service attack tool, 1999b.

DITTRICH, D. The "tribe flood network" distributed denial of service attack tool. *University of Washington*, 10, 1999c.

FALOUTSOS, M., FALOUTSOS, P. e FALOUTSOS, C. On power-law relationships of the internet topology. Em *ACM SIGCOMM Computer Communication Review*, volume 29, págs. 251–262. ACM, 1999.

FEINSTEIN, L., SCHNACKENBERG, D., BALUPARI, R. e KINDRED, D. Statistical approaches to ddos attack detection and response. Em *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*, volume 1, págs. 303–314. IEEE, 2003.

GAMER, T. e SCHARF, M. Realistic simulation environments for ip-based networks. Em *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, pág. 83. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

GLIGOR, V. A note on the denial-of-service problem. Em *Proceedings of the 1983 IEEE Symposium on Security and Privacy*, págs. 5101–5111, 1983.

GREENE, B. R. e SMITH, P. *Cisco ISP Essentials*. Cisco Press, 2002. ISBN 1587050412.

HALAVAIS, A. M. C. *The Slashdot effect: analysis of a large-scale public conversation on the World Wide Web*. Tese de Doutorado, 2001.

HENDERSON, T., ROY, S., FLOYD, S. e RILEY, G. ns-3 project goals. Em *Proceeding from the 2006 workshop on ns-2: the IP network simulator*, pág. 13. ACM, 2006.

HERNÀNDEZ-CAMPOS, F., SMITH, F. D. e JEFFAY, K. Generating realistic tcp workloads. Em *Int. CMG Conference*, págs. 273–284, 2004.

JIN, C., CHEN, Q. e JAMIN, S. Inet: Internet topology generator. 2000.

JIN, S. e YEUNG, D. S. A covariance analysis model for ddos attack detection. Em *Communications, 2004 IEEE International Conference on*, volume 4, págs. 1882–1886. IEEE, 2004.

JUNG, J., KRISHNAMURTHY, B. e RABINOVICH, M. Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites. Em *Proceedings of the 11th international conference on World Wide Web*, págs. 293–304. ACM, 2002.

KARIMAZAD, R. e FARAAHI, A. An anomaly-based method for ddos attacks detection using rbf neural networks, 2011.

KOTENKO, I. e ULANOV, A. Simulation of internet ddos attacks and defense. Em *Information Security*, págs. 327–342. Springer, 2006.

KRISHNAMURTHY, B. e WANG, J. On network-aware clustering of web clients. Em *ACM SIGCOMM Computer Communication Review*, volume 30, págs. 97–110. ACM, 2000.

KULKARNI, A. e BUSH, S. Detecting distributed denial-of-service attacks using kolmogorov complexity metrics. *Journal of Network and Systems Management*, 14(1): 69–80, 2006.

LI, L. e LEE, G. Ddos attack detection and wavelets. *Telecommunication Systems*, 28 (3-4):435–451, 2005.

MANSFIELD-DEVINE, S. Anonymous: serious threat or mere annoyance? *Network Security*, 2011(1):4–10, 2011.

MATLAB. *version 8.1.0.604 (R2013a)*. The MathWorks Inc., Natick, Massachusetts, 2013.

MEDINA, A., LAKHINA, A., MATTA, I. e BYERS, J. Brite: An approach to universal topology generation. Em *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on*, págs. 346–353. IEEE, 2001.

MIRKOVIĆ, J. *D-WARD: DDoS network attack recognition and defence*. Tese de Doutorado, Ph. D. thesis, Computer Science Department, University of California, Los Angeles, 2003.

MIRKOVIC, J., ARIKAN, E., WEI, S., FAHMY, S., THOMAS, R. e REIHER, P. Benchmarks for ddos defense evaluation. Em *Military Communications Conference, 2006. MILCOM 2006. IEEE*, págs. 1–10. IEEE, 2006.

MIRKOVIC, J., DIETRICH, S., DITTRICH, D. e REIHER, P. *Internet Denial of Service: Attack and Defense Mechanisms (Radia Perlman Computer Networking and Security)*. Prentice Hall PTR, 2004a.

MIRKOVIC, J. e REIHER, P. A taxonomy of ddos attack and ddos defense mechanisms. *SIGCOMM Comput. Commun. Rev.*, 34:39–53, April 2004b. ISSN 0146-4833. URL `http://doi.acm.org/10.1145/997150.997156`.

NEEDHAM, R. Denial of service: an example. *Communications of the ACM*, 37(11): 42–46, 1994.

NOURELDIEN, A. Protecting web servers from dos/ddos flooding attacks. a technical overview. Em *International Conference on Web-Management for International Organisations. Proceedings. Geneva*, 2002.

NSFOCUS. Ddos attacks in h2 2013, 2013. URL `http://en.nsfocus.com/SecurityReport/NSFOCUSDDoSThreatReport2013.pdf`.

ORACLE CORPORATION. Oracle VM VirtualBox User Manual, dec 2013. `https://www.virtualbox.org/manual/UserManual.html`.

QUOITIN, B. Towards more representative internet topologies. *Computer Networks*, 44: 737–755, 2004.

QUOITIN, B., DEN SCHRIECK, V. V., FRANÇOIS, P. e BONAVENTURE, O. Igen: Generation of router-level internet topologies through network design heuristics. Em *Proceedings of the 21st International Teletraffic Congress*, págs. 1 – 8, September 2009.

RANJAN, S., SWAMINATHAN, R., UYSAL, M. e KNIGHTLY, E. W. Ddos-resilient scheduling to counter application layer attacks under imperfect detection. Em *INFOCOM*. Citeseer, 2006.

RENARD, K., PERI, C. e CLARKE, J. A performance and scalability evaluation of the ns-3 distributed scheduler. Em *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, págs. 378–382. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2012.

SAHA, B. e GAIROLA, A. Botnet: an overview. *CERT-In White Paper, CIWP-2005-05*, 240, 2005.

SANTOS, A. F. P. e SILVA, R. S. Detecting bandwidth ddos attack with control charts. Em *Networks, 2007. ICON 2007. 15th IEEE International Conference on*, págs. 519–524. IEEE, 2007.

SEMATECH, N. Engineering statistics handbook, disponível em http://www.itl.nist.gov/div898/handbook/, 2012.

SLY TECHNOLOGIES. User Guide, dec 2013. `http://jnetpcap.com/userguide`.

SOMMERS, J., KIM, H. e BARFORD, P. Harpoon: a flow-level traffic generator for router and network tests. *SIGMETRICS Perform. Eval. Rev.*, 32:392–392, June 2004. ISSN 0163-5999. URL `http://doi.acm.org/10.1145/1012888.1005733`.

VARGA, A. Omnet++. Em *Modeling and Tools for Network Simulation*, págs. 35–59. Springer, 2010.

VARGA, A. e HORNIG, R. An overview of the omnet++ simulation environment. Em *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, pág. 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

WANG, B., LI, Z., TU, H. e MA, J. Measuring peer-to-peer botnets using control flow stability. Em *Availability, Reliability and Security, 2009. ARES'09. International Conference on*, págs. 663–669. IEEE, 2009.

WEIGLE, M. C., ADURTHI, P., HERNÁNDEZ-CAMPOS, F., JEFFAY, K. e SMITH, F. D. Tmix: a tool for generating realistic tcp application workloads in ns-2. *ACM SIGCOMM Computer Communication Review*, 36(3):65–76, 2006.

WEINGARTNER, E., VOM LEHN, H. e WEHRLE, K. A performance comparison of recent network simulators. Em *Communications, 2009. ICC'09. IEEE International Conference on*, págs. 1–5. IEEE, 2009.

WIRESHARK, N. P. A. Disponível em:¡http://www.wireshark.org¿. *Acesso em Junho*, 2014.

XIANG, Y., LI, K. e ZHOU, W. Low-rate ddos attacks detection and traceback by using new information metrics. *Information Forensics and Security, IEEE Transactions on*, 6(2):426–437, 2011.

YU, S., ZHOU, W. e DOSS, R. Information theory based detection against network behavior mimicking ddos attacks. *Communications Letters, IEEE*, 12(4):318–321, 2008.

ZARGAR, S., JOSHI, J. e TIPPER, D. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. 2013.

ZHANG, C., CAI, Z., CHEN, W., LUO, X. e YIN, J. Flow level detection and filtering of low-rate ddos. *Computer Networks*, 56(15):3417–3431, 2012.