colloquium exactarum

COLETA, TRATAMENTO E ARMAZENAMENTO DE FLUXOS NO PADRÃO NETFLOW

COLLECTION, PROCESSING AND STORAGE OF FLOWS IN NETFLOW STANDARD

Vitor Ruiz[1], Helton Sapia[1]

[1]Faculdade de Informática – FIPP, Universidade do Oeste Paulista – UNOESTE
E-mail: vitorasr@hotmail.com, helton.sapia@gmail.com

RESUMO - Este trabalho apresenta uma ferramenta que coleta, armazena e permite uma análise posterior dos fluxos, no padrão *NetFlow*. Com o crescente aumento do número de dispositivos conectados e do acesso à Internet, a densidade do tráfego nas redes também cresceu muito. Para garantir a segurança, é preciso que seja possível identificar eventos prejudiciais ao ambiente de rede. Os métodos tradicionais de análise utilizam a coleta de pacotes, que é computacionalmente inviável em alguns casos (quando a quantidade de tráfego é muito grande, por exemplo), sendo a coleta de fluxo mais indicada nestes casos. Este trabalho tem como objetivo o desenvolvimento de uma aplicação que colete, trate e armazene fluxos, no padrão *NetFlow*. Os fluxos são coletados utilizando um socket UDP, com auxílio de um Gateway (OpenBSD) e salvos em um banco de dados MySQL.
Palavras-chave: *Netflow*; Coletor; Fluxos.

ABSTRACT - This work presents a tool that collect, store and allows a posterior analysis of the flows, in the NetFlow standard. Due to increasing number of connected devices and access to the Internet, the traffic density over networks have also increased greatly. To ensure safety, it must be possible to identify adverse events to the network environment. Traditional analytical methods used to capture packets, which is computationally infeasible in some cases (when the amount of traffic is very large, for example), the collection using flows is most suitable in these cases. This work aims at the development of an application that collects, treats and stores flows, in the Netflow standard. The flows are collected using a UDP socket, with a Gateway (OpenBSD) assistance and stored in a MySQL database.
Keywords: Netflow; Colector; Flows.

Recebido em: 05/05/2015
Revisado em: 10/09/2015
Aprovado em: 08/11/2015

1 INTRODUÇÃO

Diariamente uma diversidade de eventos ocorre em um ambiente de rede. Esses eventos podem ser normais para o ambiente ou não (ex. eventos maliciosos), portanto é muito importante garantir a segurança e a qualidade dos serviços dentro da infraestrutura de rede e para isso é necessário identificar esses eventos.

Desempenho e exatidão são requisitos necessários para que um evento normal não seja erroneamente considerado como malicioso ou vice-versa. A coleta de pacotes deve ocorrer de uma forma rápida o suficiente para que a taxa de perda de não seja alta e posteriormente não afete na análise do evento (os pacotes perdidos não podem afetar no resultado final).

Empresas como a Cisco possuem softwares para detecção de eventos em redes como por exemplo o Cisco SMARTnet. Entretanto, tais softwares são proprietários, ou seja, é preciso licenciar uma cópia se quiser utilizá-lo.

Um fluxo de rede é definido como uma sequência de pacotes unidirecionais que possuem algumas propriedades comuns que passam através de um dispositivo da rede. O tráfego numa rede de dados pode ser visto como fluxos que passam através de elementos de rede. Muitas vezes é interessante, útil ou mesmo necessário ter acesso a informações sobre esses fluxos. Um processo de coleta deve ser capaz de

receber essas informações, o que exige uniformidade no método de representá-las partir dos elementos de rede para o ponto de coleta, afirma Claise et al. (2013).

O *Netflow* é um protocolo que inicialmente foi implementado como um recurso apenas em dispositivos da Cisco e de acordo com Claise (2004) os serviços *NetFlow* da *Cisco Systems* fornecem aos administradores acesso as informações de fluxo de IP a partir de suas redes de dados. Os elementos da rede (roteadores e *switches*) reúnem dados de fluxo e os exportam para coletores. Os dados coletados fornecem medição de grão fino para recursos de uso de contabilidade altamente flexível e detalhado.

O objetivo deste trabalho é o desenvolvimento de uma ferramenta acadêmica que colete fluxos no padrão *Netflow*, trate os dados extraídos destes fluxos e os armazene em um banco de dados relacional, para que seja possível analisar esses dados posteriormente utilizando apenas ferramentas gratuitas. Apesar do padrão *Netflow* ser proprietário da Cisco Systems, este trabalho pôde ser desenvolvido devido à criação do padrão IPFIX (*IP Flow Information eXport*).

O IPFIX foi criado pela IETF (*Internet Engineering Task Force*), uma organização de padrões abertos que desenvolve e promove padrões de Internet devido à necessidade de um padrão comum (universal) de exportação para informações de fluxo IP de roteadores (equipamentos intermediários em uma rede que encaminham pacotes de dados entre redes de computadores), sondas e outros dispositivos que são utilizados por sistemas de mediação, sistemas de contabilidade/faturamento e sistemas de gerenciamento de rede para facilitar serviços como a medição, contabilidade e faturamento. O padrão IPFIX (assim como o padrão *Netflow*) define como as informações de fluxo de IP devem ser formatadas e transferidas de um exportador a um coletor. A versão 9 do *Netflow* foi a base para a criação do IPFIX, motivo pelo qual também é conhecido por versão 10 do *Netflow*.

De acordo com Corrêa, Proto e Cansian (2008), esses padrões fornecem uma série de especificações para a sumarização de informações da rede, possibilitando de forma ampla a análise do tráfego, incluindo a detecção de eventos de segurança. Entretanto, esses padrões não preveem um modelo de armazenamento dessas informações, ou seja, a forma de armazenamento fica a cargo dos desenvolvedores de aplicações. Por esse motivo, problemas como a impossibilidade de uma aplicação utilizar dados armazenados por outra aplicação e o baixo desempenho da solução de armazenamento escolhida por um desenvolvedor interferem na eficiência da análise de tráfego e detecção de eventos de segurança, fator que contribuiu na escolha de um banco de dados relacional para o armazenamento dos dados dos fluxos de rede.

O restante deste trabalho está organizado segundo a seguinte estrutura: a seção 2 descreve sobre os trabalhos relacionados, os fatores que influenciaram na escolha das ferramentas e do padrão utilizado considerando o diferencial da atual proposta. A seção 3 lista os materiais usados neste trabalho e explica detalhadamente o método utilizado. Já a seção 4 demonstra possíveis utilizações com os dados obtidos pela aplicação aqui descrita e os problemas encontrados durante o desenvolvimento. E por fim, a seção 5 contém as conclusões e sugestões para possíveis trabalhos futuros.

2 TRABALHOS RELACIONADOS

Coleta e análise de informações de pacotes são largamente realizadas por diversas empresas (e até mesmo por pessoas, como os *hackers*, por exemplo). Tanto a coleta quanto a análise podem ser feitas de diversas maneiras.

O trabalho de Corrêa (2009) explica detalhadamente o que é um fluxo e os motivos que determinaram o uso de fluxos (ao invés de pacotes) para se fazer a coleta e a análise dos dados. Segundo Corrêa (2009), quando um fluxo *Netflow* é coletado de um roteador *stub* (equipamento que liga a rede de uma instituição com outra rede ou com a Internet), permite a visão completa de todas as conexões e sessões do ambiente, não inserindo nenhum tipo de latência devido ao desencapsulamento já que não houve análise do conteúdo dos pacotes. Correa afirma que a exportação de fluxos *Netflow* pode diminuir a carga de um roteador central ativando a exportação em *switches* (equipamentos que interligam computadores em uma rede) ou com softwares ligados à rede e ainda pontua que o *Netflow* possibilita obter respostas sobre quem, o que, de onde, como e quando um tráfego ocorreu (em uma análise de dados), porém reforça que essa é uma tecnologia que não define nenhuma maneira de utilização dessas informações. Devido a essas e outras vantagens

descritas em Corrêa (2009), o padrão *Netflow* foi escolhido para este trabalho.

Já o trabalho de Navarro, Nickless e Winkler (2000) demonstra que utilizar banco de dados relacionais facilita a manipulação dos dados, propõe um modelo de tabela para o banco relacional e consultas que identificam eventos nos dados extraídos dos registros *NetFlow*. De acordo com Corrêa, Proto e Cansian (2008), O MySQL proporciona vários recursos que auxiliam a consulta aos dados. Um exemplo disso são as funções *inet_ntoa*() e *inet_aton*() que convertem números decimais em uma expressão alfa-numérica e vice-versa. Devido a essas vantagens e aos comparativos descritos em Navarro, Nickless e Winkler (2000) o banco de dados relacional MySQL foi escolhido para este trabalho.

Corrêa, Proto e Cansian (2008) reforçam que a tecnologia de banco de dados relacional é uma boa solução para o problema de armazenamento e propõem melhorias no trabalho de Navarro, Nickless e Winkler (2000), como otimização no espaço utilizado para armazenamento, modelo de várias tabelas para buscas otimizadas e consultas detalhadas para detectar diversos tipos de eventos na rede.

O diferencial deste trabalho é a utilização de ferramentas e linguagens que possuem código aberto (não gerando custo de aquisição) e a flexibilidade. A aplicação é totalmente configurável, ou seja, pode-se alterar a porta de escuta, a quantidade de exportadores de dados, a quantidade de coletores, a localização do banco de dados relacional (local ou remoto), o método de tratamento e extração das informações dos fluxos ou até mesmo a versão *Netflow na* qual a aplicação irá atuar (desde que sejam feitos os devidos ajustes). Diferente do modelo proposto por Corrêa, Proto e Cansian (2008), este trabalho utiliza apenas 2 tabelas simples, onde cada campo representa um respectivo dado do fluxo *Netflow.*

3 MATERIAIS E MÉTODO

Para alcançar o objetivo deste trabalho, o método foi composto por três etapas: preparação do ambiente para exportar fluxos no padrão *Netflow*, coleta e tratamento e armazenamento dos dados dos fluxos coletados.

Conforme evidenciado pelos trabalhos relacionados o banco de dados relacional foi utilizado neste trabalho devido às vantagens citadas na seção 2 e também pelo fato de ser uma ferramenta gratuita. O mesmo vale para o *Netflow*, graças ao IPFIX foi possível utilizar este padrão sem custo adicional, agregando todas as vantagens também citadas na seção 2 a este trabalho.

O ambiente utilizado neste trabalho é ilustrado pela Figura 1.

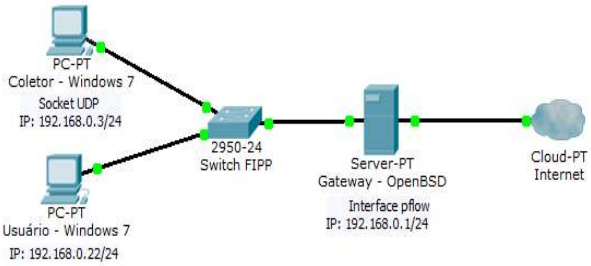

Figura 1. Ambiente utilizado.

Ao navegar na Internet a partir da máquina Usuário utilizando um navegador web para acessar o site da universidade cuja *URL* é www.unoeste.br, irá gerar diversos pacotes na rede. Esses pacotes devem obrigatoriamente passar pela máquina *Gateway – OpenBSD* (2014) que contém uma interface que contabiliza os fluxos, os transforma em padrão *Netflow* e os envia para a máquina Coletor onde existe a aplicação desenvolvida que estará pronta para receber, tratar e armazenar esses fluxos.

Um fluxo é definido como um conjunto de pacotes IP de passagem em um ponto de observação na rede durante um determinado intervalo de tempo. Todos os pacotes pertencentes a um fluxo particular têm um conjunto de propriedades comuns. Cada propriedade é definida como o resultado da aplicação de uma função para os valores de:

- Um ou mais campos de cabeçalho do pacote (ex. endereço IP destino), campo de cabeçalho de transporte (ex. o número da porta de destino) ou aplicação de campo de cabeçalho (ex. campos de cabeçalho de aplicações em tempo real);
- Uma ou mais características do próprio pacote (ex. número de etiquetas *MPLS* [*Multi Protocol Label Switching*]);
- Um ou mais campos derivados do tratamento de pacotes (ex. endereço IP do próximo salto, a interface de saída etc.).

Um pacote é definido pertencente a um fluxo desde que preencha completamente todas as propriedades definidas do fluxo, segundo Quitekk et al. (2004).

3.1 COLETA DOS FLUXOS

A coleta dos fluxos foi realizada com o auxílio de um *socket UDP* (*User Datagram Protocol*), implementado em uma aplicação escrita em linguagem Java. A aplicação acessa o banco de dados através de uma conexão local, entretanto, este acesso poderia ser feito de modo remoto, com o banco de dados localizado em outro host. O *socket*, que está sendo executado em máquina denominada Coletor permanece em um estado chamado *listening* (pronto para receber) na porta 9876 (no caso deste trabalho, com a possibilidade de alteração da porta caso necessário) e ao receber os fluxos a aplicação armazena todos que são entregues na porta com seus respectivos cabeçalhos nas tabelas definidas no banco de dados relacional.

Os dados que serão armazenados no banco de dados são os dados extraídos do fluxo coletado. Para que seja possível extrair os dados, é necessário converter os bytes de cada posição de memória do fluxo coletado. Para isso, foi necessário o uso de algumas funcionalidades, representadas pela Figura 2.

```
1  public static int unsignedByteToInt(byte b) {
2      return (int) b & 0xFF;
3  }
4
5  public static int unsignedByteToInt(byte[] b, int offset, int len) {
6      int r = 0;
7      for (int i = 0; i < len; i++) {
8          r += unsignedByteToInt(b[offset + i]);
9      }
10     return r;
11 }
12
13 public static String byteToHex(byte b) {
14     int i = b & 0xFF;
15     return Integer.toHexString(i);
16 }
17
18 public static String byteToHex(byte[] b, int offset, int len) {
19     String r = "";
20     for (int i = 0; i < len; i++) {
21         r += byteToHex(b[offset + i]);
22     }
23     return r;
24 }
```

Figura 2. Funcionalidades.

A função *unsignedByteToInt(byte b)* que retorna um "E" binário (*AND*) entre o byte enviado por parâmetro e o valor 0xFF (255, em hexadecimal). A variável buffer, utilizada no coletor, é um vetor (variável que armazena várias variáveis do mesmo tipo) de bytes que recebe o valor da funcionalidade Java *DatagramPacket.getData()*, a qual retorna um vetor de bytes contendo todos os dados do fluxo capturado. Cada "posição" do vetor buffer equivale a um byte. Por exemplo, para obter o valor das variáveis do cabeçalho (especificadas na Tabela 1) como a variável *version*, deve-se enviar a primeira posição (primeiro byte) da variável buffer, ou seja, buffer[1] (em termos de programação, [1] seria a "posição" que se deseja acessar) como parâmetro da função *unsignedByteToInt*. Para a variável *count*, deve-se enviar o terceiro byte da variável buffer (buffer[3]) e assim sucessivamente, para cada uma das informações. Para obter o valor de *sys_uptime*, por exemplo, deve-se utilizar o seguinte comando: *Integer.parseInt(byteToHex(buffer, 4, 4), 16)*, onde *byteToHex* irá concatenar a variável enviada como primeiro parâmetro (buffer), a partir da posição do segundo parâmetro (4), na quantidade de vezes indicada pelo terceiro parâmetro (4), ou seja, os valores de buffer[4] até buffer[7] serão concatenados e retornados em uma *String (cadeia de caracteres)* que será convertida em um número inteiro pela funcionalidade *Integer.parseInt* do Java, onde o primeiro parâmetro é a *String* a ser convertida e o segundo é a base numérica desejada, que neste caso é 16 por ser um valor hexadecimal. O cabeçalho possui 24 bytes (0-23), após ele estão localizados os registros de fluxo. Após o cabeçalho pode-se ter entre 1 e 30 registros (cada registro contém todas as informações listadas na Tabela 2), onde o número exato de registros é indicado pela variável *count*. Para que seja possível extrair os dados de todos os fluxos, deve-se fazer uma repetição de "*count*" vezes e obter o valor de cada variável em suas respectivas posições. A primeira variável do registro é a *srcaddr*, que contém um valor IP e possui 4 bytes, portanto deve-se obter e juntar todos octetos para compor o valor real. Por exemplo, o IP 192.168.0.1 será obtido da seguinte maneira: buffer[24+i] contém o valor 192, buffer[25+i] contém 168, buffer[26+i] contém 000 e, por fim, buffer[27+i] contém 001. Ao juntar os octetos obtém-se o endereço IP real. A variável i é inicializada com o valor 0 e deve ser somada com o valor da posição de memória que é enviada por parâmetro nas funcionalidades devido à repetição que está inserida. Esta variável i é incrementada de 48 em 48 (tamanho em bytes de um registro) para que seja possível obter o

valor de todos os seguintes registros. A extração do valor IP 192.168.0.1 exemplificada acima pode ser representada pela Figura 3.

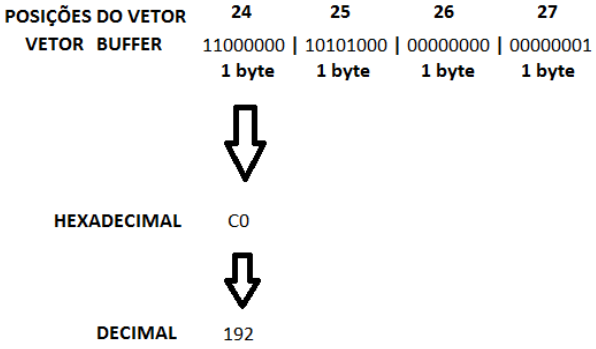

Figura 3. Extração de um endereço IP.

Para produzir pacotes no padrão *Netflow*, um contabilizador *Netflow* é necessário. Para isso, um *gateway* é utilizado para que os fluxos sejam contabilizados no padrão *Netflow*, ou seja, são contabilizados os pacotes que passam pelo *gateway*, o qual foi implementado em uma máquina com o sistema operacional *OpenBSD*. O processo de contabilização é realizado por uma interface virtual, que no caso do ambiente utilizado neste trabalho, trata-se de uma interface *pflow*, onde os dados serão contabilizados e depois enviados para a máquina coletora. A interface *pflow* é um pseudo-dispositivo que exporta dados contábeis *pflow* a partir do *kernel* (parte principal do sistema operacional do computador) usando pacotes *UDP*. *Pflow* é compatível com a versão 5 do *Netflow* e com o IPFIX.

3.2 ESTRUTURA DE ARMAZENAMENTO

A estrutura de armazenamento proposto por este trabalho utiliza duas tabelas no banco de dados relacional: tabela_cabecalho e tabela_fluxos. As tabelas foram criadas com base no trabalho de Corrêa, Proto e Cansian (2008) e no cabeçalho da versão 5 do *Netflow*, com algumas alterações. O cabeçalho de cada fluxo é armazenado na primeira tabela e, através de uma chave estrangeira, tem-se uma referência da tabela_cabecalho na tabela_fluxos, para que seja possível identificar os pacotes com seus respectivos cabeçalhos. A Tabela 1 e a Tabela 2 representam, respectivamente, a tabela_cabecalho e a tabela_fluxos no banco de dados relacional.

Tabela 1. Modelo proposto para a tabela_cabecalho

Campos	**Tipo de dados**	**Representação**
codigo	*INT*	*Primary key*, *not null*, auto incremento
version	*TINYINT*	*not null*
count	*TINYINT*	*not null*
sys_uptime	*INT*	*not null*
unix_secs	*BIGINT*	*not null*
unix_nsecs	*BIGINT*	*not null*
flow_sequence	*INT*	*not null*
engine_type	*INT*	*not null*
engine_id	*INT*	*not null*
ip	*VARCHAR*(15)	*not null*
porta	*INT*	*not null*
tamanho	*INT*	*not null*
timestamp	*TIMESTAMP*	*not null*

Legenda:
codigo: chave primária da tabela, é incrementado automaticamente em uma unidade a cada cabeçalho inserido.
version: número da versão do formato de exportação *Netflow*.
count: número de fluxos exportados nesse registro.
sys_uptime: tempo em milissegundos desde que o dispositivo está ligado.
unix_secs: contagem atual de segundos desde 0000 UTC 1970.
unix_nsecs: nanossegundos residuais desde 0000 *UTC* 1970.
flow_sequence: contador de sequência dos fluxos totais visto.
engine_type: tipo de mecanismo de comutação de fluxo.
engine_id: número do slot do mecanismo de comutação de fluxo.
ip: IP origem do fluxo.
porta: porta origem pela qual o fluxo passou.
tamanho: 24 (bytes do cabeçalho) + *count* * 48 (bytes de cada registro de fluxo).
timestamp: carimbo de tempo do momento em que o cabeçalho foi armazenado.

Tabela 2. Modelo proposto para a tabela_fluxos

Campos	Tipo de dados	Representação
codigo	*INT*	*Primary key*, *not null*, auto incremento
cod_cabecalho	*INT*	*Foreign key* (referencia código da tabela cabeçalho), *not null*
srcaddr	*VARCHAR*(15)	*not null*
dstaddr	*VARCHAR*(15)	*not null*
nexthop	*VARCHAR*(15)	*not null*
input	*SMALLINT*	*not null*
output	*SMALLINT*	*not null*
dPkts	*INT*	*not null*
dOctets	*INT*	*not null*
first	*INT*	*not null*
last	*INT*	*not null*
srcport	*SMALLINT*	*not null*
dstport	*SMALLINT*	*not null*
tcp_flags	*TINYINT*	*not null*
prot	*TINYINT*	*not null*
tos	*TINYINT*	*not null*
src_as	*SMALLINT*	*not null*
dst_as	*SMALLINT*	*not null*
src_mask	*TINYINT*	*not null*
dst_mask	*TINYINT*	*not null*

Legenda:
codigo: chave primária da tabela, é incrementado automaticamente em uma unidade a cada cabeçalho inserido.
cod_cabecalho: chave estrangeira que referencia o campo "código" da tabela cabeçalho.
srcaddr: endereço IP origem.
dstaddr: endereço IP destino.
nexthop: endereço IP do próximo roteador.
input: índice SNMP de interface de entrada.
output: índice SNMP de interface de saída.
dPkts: quantidade de pacotes no fluxo.
dOctets: número total de bytes de camada 3 nos pacotes do fluxo.
first: tempo, em centésimos de segundo, no início do fluxo.
last: tempo, em centésimos de segundo, do momento em que o último pacote do fluxo foi recebido.
srcport: número da porta de origem.
dstport: número da porta de destino.
tcp_flags: byte cumulativo ou usado como *flags* TCP (resultado da operação lógica OU com todas as *flags* que aparecem no fluxo). Em fluxos bem formados, este campo significa que as *flags SYN*, *FYN* e *ACK* estarão sempre ativas.
prot: tipo de protocolo IP (TCP = 6, UDP = 17).
tos: tipo de serviço IP, utilizado para diferenciar o tipo do pacote a ser transportado (para que possa ter prioridade em sua transmissão, por exemplo).
src_as: número de sistema autônomo de origem. Número único, disponível globalmente para identificar um sistema autônomo. Permite que o sistema troque informações exteriores de roteamento com outros sistemas autônomos vizinhos.
dst_as: número de sistema autônomo de destino. Número único, disponível globalmente para identificar um sistema autônomo. Permite que o sistema troque informações exteriores de roteamento com outros sistemas autônomos vizinhos.
src_mask: máscara de bits do prefixo do endereço de origem.
dst_mask: máscara de bits do prefixo do endereço de destino.

4 DISCUSSÃO

A ferramenta, com o auxílio do banco de dados relacional, consegue coletar, tratar e armazenar as informações dos fluxos.

Uma vez coletados e armazenados, os dados dos fluxos estão prontos para análise, a qual pode ser feita de diversas maneiras. O conteúdo a ser analisado e a forma da análise dependerá do objetivo do usuário, que pode utilizar desde instruções SQL e até realizar mineração de dados.

Separar o coletor permite que seja possível especificar uma máquina para fazer apenas a coleta dos fluxos, obtendo um melhor desempenho já que a mesma não será utilizada por outros processos enquanto faz a coleta. Os dados também podem ser salvos localmente ao invés de salvar diretamente no servidor da empresa, evitando a sobrecarga da rede. Também é possível a coleta de um ou mais contabilizadores simultaneamente, ou seja, caso existam duas ou mais interfaces que exportam dados *Netflow*, basta que os dados contabilizados sejam direcionados ao coletor para que estes sejam armazenados.

A partir dos dados armazenados, pode-se utilizar diversos comandos SQL para fazer diversos tipos de análises diferentes. Para calcular a taxa de perda de pacotes, por exemplo,

pode-se utilizar a seguinte instrução SQL, representada pela Figura 4:]

```
SELECT c.* FROM tabela_cabecalho c
LEFT JOIN tabela_cabecalho c2
ON (c.flow_sequence+c.count) = c2.flow_sequence
WHERE c.porta = 60324
AND c2.flow_sequence IS null;
```

Figura 4. Comando SQL para calcular a taxa de perda de pacotes.

Este SQL irá retornar todos os fluxos que não possuem o valor *flow_sequence+count*, onde *flow_sequence* é o identificador do fluxo coletado e, ao somar *count* (quantidade de fluxos inseridos), tem-se o valor *flow_sequence* do próximo fluxo. Por exemplo, se o *flow_sequence* atual está em 1000 e o valor de *count* do fluxo coletado for 30, o "próximo" deveria ser 1030, entretanto, se não houver um fluxo cujo *flow_sequence* é 1030, o fluxo com o valor 1000 será uma das linhas retornadas no SQL da Figura 5.

codigo	version	count	sys_uptime	unix_secs	unix_nsecs	flow_sequence
25504	5	30	408000	88417894	31302216	240
25551	5	30	484000	1414686290	500879056	2280
25557	5	30	485000	1414686291	500879615	2490
25565	5	30	495000	1414686301	31305361	2790
25567	5	30	496000	1414686302	500886180	3000
25569	5	30	497000	1414686303	500887018	3420
25573	5	30	507000	1414686313	31306685	3630
25578	5	30	508000	1414686314	31307104	3810
25580	5	30	509000	1414686315	31307803	4080
25620	5	30	562000	1414686368	500923824	5580
25632	5	30	575000	1414686381	500930669	6540

Figura 5. Fluxos sem o "próximo".

Para identificar os maiores consumidores em uma determinada data, pode-se usar o SQL representado pela Figura 6:

```
SELECT sum(dOctets) AS soma, srcaddr
FROM projeto.tabela_fluxos
WHERE cod_cabecalho
IN (SELECT codigo FROM projeto.tabela_cabecalho
WHERE date(timestamp) = '2014-10-28')
GROUP BY srcaddr
ORDER BY soma DESC
```

Figura 6. Comando SQL para identificar maiores consumidores em uma determinada data.

Os maiores consumidores da data 28/10/2014 utilizada no SQL anterior estarão listados em ordem decrescente, como representa a Figura 7.

soma	srcaddr
11300958302	177.131.35.1
686833942	149.5.0.70
519525517	62.210.72.23
504868735	95.211.227.171
450474198	200.174.107.16
293192390	31.13.73.40
287170169	31.216.144.65
268591870	149.5.0.73
251053646	192.16.70.71
247654200	31.13.73.7
247324621	31.13.73.23

Figura 7. Maiores consumidores do dia 28/10/2014.

Para saber quantos fluxos que foram inseridos em cada dia, o seguinte comando representado pela Figura 8 pode ser

utilizado:

```
SELECT min(timestamp), max(timestamp), sum(count)
FROM projeto.tabela_cabecalho
GROUP BY date(timestamp)
```

Figura 8. Comando SQL para identificar a quantidade de fluxos inseridos por dia.

O resultado do SQL anterior irá representar a data e a hora em que o primeiro e o último fluxo foram inseridos e o somatório dos fluxos inseridos durante este período, como mostra a Figura 9.

min(timestamp)	max(timestamp)	sum(count)
2014-10-28 11:39:29	2014-10-28 23:59:52	158375
2014-10-29 00:00:34	2014-10-29 23:59:43	432492
2014-10-30 00:00:19	2014-10-30 23:59:49	566650
2014-10-31 00:00:28	2014-10-31 23:59:48	893710
2014-11-01 00:00:37	2014-11-01 23:52:25	6859
2014-11-02 00:27:13	2014-11-02 23:37:32	13259
2014-11-03 00:31:40	2014-11-03 23:57:39	786955
2014-11-04 00:15:01	2014-11-04 23:59:48	814150
2014-11-05 00:00:22	2014-11-05 21:18:28	826646
2014-11-06 16:11:55	2014-11-06 23:15:08	62279
2014-11-07 17:40:40	2014-11-07 20:37:18	118770

Figura 9. Período e quantidade de fluxos inseridos no mesmo.

Se desejar saber a quantidade de fluxos inseridos por minuto e quantos fluxos foram inseridos por segundo, deve-se utilizar o comando SQL representado pela Figura 10:

```
SELECT date_sub(timestamp, interval second(timestamp) second)
AS Date,
count(*) AS Flows,
count(*) AS Flows_per_second
FROM tabela_cabecalho
WHERE porta = 60324
GROUP BY hour(timestamp), minute(timestamp)
ORDER BY timestamp
```

Figura 10. Comando SQL para identificar a quantidade de fluxos por minuto e a quantidade de fluxos inseridos por segundo.

O resultado do SQL representado pela Figura 10 será o período, separado de minuto a minuto, a quantidade total de fluxos inseridos neste período e quantos fluxos foram inseridos por segundo no mesmo, como pode ser visto na Figura 11.

Date	Flows	Flows_per_second
2014-10-28 19:37:00	128	2.1333
2014-10-28 19:38:00	133	2.2167
2014-10-28 19:39:00	125	2.0833
2014-10-28 19:40:00	128	2.1333
2014-10-28 19:41:00	121	2.0167
2014-10-28 19:42:00	96	1.6000
2014-10-28 19:43:00	96	1.6000
2014-10-28 19:44:00	115	1.9167
2014-10-28 19:45:00	118	1.9667
2014-10-28 19:46:00	119	1.9833
2014-10-28 19:47:00	126	2.1000

Figura 11. Período, quantidade de fluxos inseridos neste período e quantidade de fluxos inseridos por segundo.

Como é demonstrado acima, são inúmeras as consultas possíveis para análise dos dados. Nos comandos utilizados neste trabalho, a porta 60324 foi utilizada na cláusula *WHERE* para que se analise os fluxos de apenas uma das interfaces que estão exportando dados *pflow* (a porta de origem do cabeçalho capturado de uma interface exportadora é sempre a mesma).

4.1 PROBLEMAS ENCONTRADOS

Extrair dados dos pacotes, o passo mais importante deste trabalho, não foi algo trivial. Os dados dos pacotes dos fluxos coletados pela máquina Coletor estavam em binário, o que dificultou a obtenção das informações. Foi necessária a criação de uma funcionalidade para converter bytes de posições específicas em um valor inteiro. Algumas variáveis como '*input*' e '*first*' precisaram ser convertidas de binário para hexadecimal antes de serem convertidas para um valor inteiro.

Outro problema foi a captura de pacotes na versão 9 do *Netflow*, a segunda mais utilizada, após a versão 5. Uma vez capturado, todos os cabeçalhos dos fluxos possuíam o valor 1 na variável '*count*', que representa a quantidade de fluxos exportados no pacote capturado (esta variável deve ter valor entre 1 e 30), ou seja, não contém dados após o cabeçalho, portanto ao mudar da versão 5 para a 9 no

ambiente utilizado, perdeu-se o conteúdo dos fluxos, o que inviabilizou o uso da mesma.

5 CONCLUSÕES E TRABALHOS FUTUROS

A coleta, o tratamento e o armazenamento dos fluxos no banco de dados relacional realizados neste trabalho prepararam os dados para que possam agora ser analisados.

A análise destes dados irá permitir que os dados sejam transformados em informações que possam ser utilizadas de alguma forma pertinente de acordo com os objetivos de quem utilizou a ferramenta.

A partir das instruções SQL listadas acima, uma série de informações foram obtidas, como a taxa de perda de pacotes, maior consumidor, fluxos inseridos por dia, por minuto, por segundo, dentre outras. Essas informações podem auxiliar na tomada de decisão em uma empresa, por exemplo: caso a taxa de perda de pacotes seja alta, pode-se verificar a rede e sua infraestrutura para identificar o motivo pelo qual a perda está ocorrendo; verificar quem são os maiores consumidores e o motivo. Enfim, uma série de informações podem ser obtidas e de acordo com elas atitudes devem ser tomadas.

5.2 TRABALHOS FUTUROS

Para que a ferramenta seja mais flexível, pretende-se:

- Implementar a coleta de fluxos compatível com as outras versões do *Netflow*, principalmente a versão 9;
- Otimizar os campos e os tipos dos dados nas tabelas do banco de dados relacional (eliminar campos não utilizados e diminuir o tamanho das variáveis como *int* para *smallint* ou *tinyint*, por exemplo);
- Otimizar o código fonte da aplicação que coleta os fluxos, para que esta seja mais rápida em tempo de execução e diminua possíveis perdas de pacotes.
-

REFERÊNCIAS

CLAISE, B. et al. **Specification of the IP Flow Information Export (IPFIX) protocol for the exchange of ip traffic flow information**. 2013. Disponível em: <http://www.ietf.org/rfc/rfc7011.txt>.

CLAISE, B. **RFC 3954**: Cisco Systems NetFlow Services Export Version 9. 2004. Disponível em: <http://www.ietf.org/rfc/rfc3954.txt>.

CORRÊA, J.; PROTO, A.; CANSIAN, A. **Modelo de armazenamento de fluxos de rede para análises de tráfego e de segurança**. 2008. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/sbseg/2008/006.pdf>.

CORRÊA, J. **Um modelo de detecção de eventos em redes baseado no rastreamento de fluxos.** 2009. 98 f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual Paulista, São Paulo - SP.

NAVARRO, J.; NICKLESS, B.; WINKLER, L. **Combining Cisco NetFlow Exports with Relational Database Technology for Usage Statistics, Intrusion Detection, and Network Forensics**. 2000. Disponível em: <https://www.usenix.org/legacy/event/lisa2000/full_papers/navarro/navarro_html>

OpenBSD. **OpenBSD Programmer's Manual PFLOW**, n.4, 2014. Disponível em: <http://www.openbsd.org/cgi-bin/man.cgi?query=pflow&apropos=0&sektion=0&manpath=OpenBSD+Current&arch=i386&format=html>.

QUITTEK, J. et al. **Requirements for IP Flow Information Export (IPFIX).** 2004. Disponível em: <http://tools.ietf.org/html/rfc3917>.