PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

WESLEY DAVISON BRAGA MELO

AVALIAÇÃO E OTIMIZAÇÃO DE SISTEMAS DPI COM BASE EM DFA E PRIORIZAÇÃO DE ASSINATURAS

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE

2014

Wesley Davison Braga Melo

Avaliação e otimização de sistemas DPI com base em DFA e priorização de assinaturas

Este trabalho foi apresentado à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de mestre em Ciência da Computação.

ORIENTADOR: **Prof .Dr. Djamel Fawzi Hadj Sadok**

CO-ORIENTADOR: **Prof. Dr. Stênio Flávio De Lacerda Fernandes**

RECIFE

2014

Catalogação na fonte
Bibliotecário Jefferson Luiz Alves Nazareno CRB4-1758

M528a Melo, Wesley Davison Braga.
Avaliação e otimização de sistemas DPI com base em DFA e priorização de assinaturas/ Wesley Davison Braga Melo – Recife: O Autor, 2014.
73 f.: fig., tab.

Orientador: Djamel Fawzi Hadj Sadok.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da computação, 2014.
Inclui referências e anexos.

1. Redes de computadores. 2. Classificação de tráfego. 3. Gerenciamento de redes de computadores I. Sadok, Djamel Fawzi Hadj. (Orientador). II. Titulo.

004.6 CDD (22. ed.) UFPE-MEI 2018-05

Wesley Davison Braga Melo

Avaliação e otimização de sistemas DPI com base em DFA e priorização de assinaturas

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 08/09/2014

BANCA EXAMINADORA

Prof. Dr. Djamel Fawzi Hadj Sadok
Centro de Informática / UFPE

Prof. Dr. Arthur de Castro Callado
Mestrado e Doutorado em Ciência da Computação / UFC

Prof. Dr. Glauco Estácio Gonçalves
Departamento de Estatística e Informática / UFRPE

Ao meu Deus, meus pais e meu amor.

Agradecimentos

Ao vencermos, vemos quão importantes foram todos aqueles que estiveram ao nosso lado nos apoiando na difícil jornada percorrida. E é a vocês que dedico este agradecimento.

A vocês, meus colegas de trabalho, agradeço por terem me mostrado o "caminho das pedras", me ajudando nos momentos de pouca experiência!

A vocês, meus chefes, agradeço pelas imensas oportunidades que me deram, depositando em mim uma confiança jamais imaginada!

A vocês, meus amigos, agradeço por terem escutado minhas declarações de cansaço e termos compartilhado dores comuns de longas batalhas!

A vocês, minha família, agradeço por terem sempre me visto como o bom representante do nosso comum sangue!

A você, meu Sol, agradeço por ter sempre me ajudado, com teu grande amor, a ver a glória da vitória durante o caminho tão espinhoso que andei!

A você, minha Mãe, agradeço por, mesmo longe, ter sempre orado ao Nosso Deus pedindo ânimo para o teu filho!

A você, meu Pai, as palavras são incapazes de descrever a tua importância nessa minha grande conquista! Você sempre foi e sempre será meu grande herói!

A ti, meu Onisciente e Amoroso Deus, que soubeste me guiar pelo Teu caminho me mostrando o real significado da palavra Kairos: o Teu Tempo Perfeito!

Resumo

Identificação e caracterização de tráfego são atividades relevantes no campo de gerência de redes. Objetivando uma melhor administração dos recursos disponíveis, gestores de conteúdo podem inserir componentes no núcleo da rede a fim de identificar os diferentes tipos de fluxos de pacotes que por ela trafegam. Tais objetivos variam, desde uma simples análise do perfil do tráfego até a priorização de conexões baseada em conteúdo. A fim de realizar tais atividades, a inspeção profunda de pacotes (Deep Packet Inspection, ou DPI) se apresenta como uma das técnicas mais difundidas, devido a sua precisão na classificação dos fluxos da rede. Dentre as diversas maneiras de se construir tal mecanismo, a utilização de Expressões Regulares (RE) tem uma forte aceitação, aliada a Autômatos Finitos Determinísticos (DFA). Contudo, em cenários onde seja exigido alto desempenho computacional, esses dois componentes podem se tornar um possível gargalo, quando não são corretamente utilizados. Ainda mais, a incorreta utilização desses componentes pode provocar até a perda de pacotes pelo sistema DPI, acarretando no total fracasso da classificação do tráfego. Dessa forma, este trabalho apresenta uma avaliação da combinação dos principais componentes de um DPI e o impacto de escolhas inadequadas no desempenho do sistema. Além disso, é proposta uma modificação em um componente importante, e negligenciado, do sistema DPI: a ordenação da lista de assinaturas. Por fim, serão fornecidos indicativos de como a utilização das técnicas propostas em conjunto permitem que os ganhos sejam maximizados.

Palavras-chave: Identificação de tráfego. Inspeção Profunda de Pacotes. Prioridade de Assinaturas. Autômatos Finitos Determinísticos. Layout de Memória

Abstract

Traffic Identification and characterization are relevant activities in the field of network management. Aiming a better administration of the available resources, content managers can insert some probes in the core of a network in order to identify the different packet flows passing through. These goals vary from a simple analysis of the traffic profile by prioritizing connections based on content. In order to perform such activities, Deep Packet Inspection (DPI) is one of the most used techniques, due to its precision when classifying network flows. Among the various ways to build such a mechanism, the utilization of Regular Expressions (RE) has a strong acceptance, coupled with Deterministic Finite Automata (DFA). However, in scenarios where high computational performance is required, these two components can become a potential bottleneck, when not properly used. Further, the improper use of these components can lead to packet loss by DPI system, resulting in total failure of the traffic classification. Thus, this work presents an evaluation of the combination of the main components of a DPI and the impact of poor choices in system performance. Further, we propose a modification on an important component, and neglected in the DPI system: the ordering on the list of signatures. Finally, it is provided indicatives of how the use of proposed techniques in conjunction allows the maximization of the gains.

Keywords: Traffic Identification. Deep Packet Inspection. Priority of Signatures. Deterministic Finite Automata. Memory Layout

Lista de Figuras

Lista de Tabelas

Abreviações e Acrônimos

BE	Bitmapped Encoding
CPU	Central Processing Unit
CS	Char-State Encoding
D^2FA	Delayed Input DFA
DFA	Deterministic Finite Automata
DPI	Deep Packet Inspection
FTBI	Flow-based Traffic Identification
GPU	Graphical Processing Unit
GT	Ground-Truth
HTML	HyperText Markup Language
IP	Internet Protocol
LE	Linear Encoding
ME	Matricial Encoding
N-NGEN	New Network traffic workload Generator
NFA	Non-deterministic Finite Automata
PAPI	Performance API
PC	Personal Computer
RCDFA	Ranged Compressed Deterministic Finite Automaton
RE	Regular Expression
SFA	Semi-deterministic Finite Automata

Sumário

1 Introdução

"A cultura é o melhor conforto para a velhice."

Aristóteles

1.1 Motivação e Contextualização

Existem diversas técnicas para caracterizar tráfego da Internet, entretanto, é possível destacar a Inspeção Profunda de Pacotes (DPI) devido ao seu alto nível de precisão [1]. Baseado nos benefícios de Expressões Regulares (RE), mecanismos de DPI podem identificar um vasto número de aplicações. Com o objetivo de classificar tráfego de redes de alta velocidade, sondas DPI podem ser inseridas no núcleo das redes a serem analisadas, o que implica em coletar e analisar pacotes a taxas *multigigabit*. Diante disso, esforços em pesquisas atuais focam em melhorar o desempenho da identificação de tráfego investigando os mecanismos internos dos sistemas DPI, tais como máquinas abstratas [2][3][4][5][6][7][8], complexidade de listas de assinaturas [9], prioridades de assinaturas [10] e utilização de processamento paralelo [11][12].

Estes sistemas DPI podem ser desenvolvidos em *hardware* e/ou *software*. A implementação desses sistemas em hardware possui a vantagem de proporcionar uma maior velocidade de processamento e, consequentemente, aumentar a vazão de análise de dados inspecionados [13]. Apesar destes benefícios, o projeto e desenvolvimento de um analisador de pacotes em *hardware* possui algumas desvantagens. Uma delas é o alto custo para construção, pois a construção dos componentes físicos exige conhecimento técnico especializado. Além disso, visto que sua arquitetura se baseia fortemente em programação de baixo nível, o custo de desenvolvimento é maior. Os programadores devem ser especializados em linguagens mais específicas e o ciclo de testes deve envolver a depuração dos circuitos lógicos [14].

Sistemas de identificação de tráfego baseados em DPI são mais facilmente implementados em plataformas de computadores convencionais (x86). Alguns dos motivos são as facilidades de programação em linguagens mais populares (e.g. C, C++), ciclos de desenvolvimento e testes mais baratos, além de reduzir o custo da solução completa [14][13]. Em tais casos, esses sistemas podem ser vistos como aplicativos convencionais que são executados no topo de algum Sistema Operacional (SO) e de algum *hardware* de prateleira.

O desenvolvimento de sistemas DPI em *software* ainda pode ser realizado em conjunto com a implementação em *hardware*. Devido à facilidade de desenvolvimento de programas em uma arquitetura x86, o processo de construção de um DPI em *software* pode ser utilizado como uma fase de validação de teorias em uma arquitetura real e mais simplificada do que em *hardware* específico. Modificações de um projeto de *hardware* nesta fase podem diminuir bastante o custo de ajustes em uma fase posterior. Em resumo, os conceitos teóricos e práticos existentes no desenvolvimento de sistemas DPI baseados em *software* contribuem para a área de análise de tráfego de forma abrangente.

Nas duas abordagens apresentadas, melhorias de desempenho devem ser cruciais no projeto da arquitetura do sistema. Um sistema de inspeção de pacotes necessita de alto desempenho devido à sua característica principal de analisar o conteúdo do tráfego da rede, sem perder informação relevante à classificação. Em detalhes, tal sistema realiza os seguintes passos:

1. Capturar o pacote da rede;
2. Identificar o fluxo associado ao pacote e recuperar suas informações;
3. Atualizar as estatísticas do fluxo;
4. Manipular o pacote para extrair a sua carga útil (dados);
5. Analisar os dados *byte* a *byte*, a fim de encontrar um padrão conhecido.

Se for considerado, em um cenário hipotético, um fluxo de dados distribuído uniformemente em uma rede de 10Gbps, cada pacote atravessaria o enlace num intervalo entre 10^{-6} e 10^{-8} segundos. Em outras palavras, os cinco passos descritos acima devem ser realizados num intervalo de tempo não superior a alguns microssegundos. O desafio para este problema se mostra maior quando se percebe que o passo cinco é o mais custoso de todos, podendo ser detalhado em várias outras etapas.

O passo cinco é a inspeção de pacotes propriamente dita. Nesta etapa, o objetivo é descobrir se a cadeia de caracteres presente na carga útil do pacote contém algum tipo de padrão conhecido. Tais padrões, chamados de assinaturas, são criados em um momento anterior à análise e representam o comportamento de uma aplicação em gerar conteúdo na rede. Esse comportamento se expressa pela inserção de palavras conhecidas ou caracteres em posições específicas do conteúdo do pacote. Cada aplicação insere padrões diversos em posições diferentes. Dessa forma, cada aplicação possui a sua respectiva assinatura. Se analisarmos algum pacote ou fluxo de rede e identificarmos que a cadeia de caracteres segue o mesmo padrão de uma assinatura, podemos classificar o fluxo como pertencente à mesma aplicação da assinatura utilizada.

A Figura 1-1 mostra um exemplo de um pacote, tendo, na sua área destacada, a carga útil do mesmo. Pode-se perceber que é possível identificar um padrão de dados, composto de palavras como "menu", "script", entre outras. Ao mesmo tempo, também é possível verificar a presença de informações mais específicas, como extensões ("js") e linguagens ("javascript"), além de *tags* HTML ("</body>" e "</html>"). Aproveitando o exemplo, é possível ver que a cadeia de caracteres possui sequências de *bytes* específicas. Com base na descrição da linguagem HTML [15], as *tags* </body> e </html> estão fortemente associadas, de forma que a cadeia "</body></html>" indica o fim de um arquivo html. Consequentemente, adicionando mais elementos a essa expressão, é possível criar uma assinatura que possa representar a transmissão de um arquivo HTML pela rede.

```
0000  74 69 63 48 6f 76 65 72 48 79 70 65 72 4c 69 6e  ticHoverHyperLin
0010  6b 43 6c 61 73 73 20 3d 20 27 63 74 6c 30 30 5f  kClass = 
0020  4d 65 6e 75 34 5f 39 27 3b 0d 0a 2f 2f 20 2d 2d  Menu4_9';../// --
0030  3e 0d 0a 3c 2f 73 63 72 69 70 74 3e 0d 0a 3c 2f  >..</script>..</
0040  66 6f 72 6d 3e 0d 0a 0d 0a 3c 73 63 72 69 70 74  form>....<script
0050  20 73 72 63 3d 22 6a 73 2f 4e 6f 49 45 41 63 74   src="js/NoIEAct
0060  69 76 61 74 65 2e 6a 73 22 20 74 79 70 65 3d 22  ivate.js" type="
0070  74 65 78 74 2f 6a 61 76 61 73 63 72 69 70 74 22  text/javascript"
0080  3e 3c 2f 73 63 72 69 70 74 3e 0d 0a 0d 0a 3c 2f  ></script>....</
0090  62 6f 64 79 3e 0d 0a 3c 2f 68 74 6d 6c 3e 0d 0a  body>..</html>..
```

Figura 1-1. Conteúdo útil do pacote

Fonte: autoria própria

Uma assinatura pode ser representada através do uso de expressões regulares. Estas expressões possuem um alto poder de representatividade, sendo possível representar mais de uma cadeia de caracteres por expressão. A Tabela 1-1 mostra os principais símbolos e seus respectivos significados na representação das expressões. Utilizando a simbologia da Tabela 1-1, é possível identificar várias cadeias de *bytes* similares apenas em uma única expressão regular. Como exemplo, podemos considerar a RE seguinte: "http/(0\.9|1\.0|1\.1)".

Observando a Tabela 1-1, podemos ver que as seguintes cadeias são reconhecidas pela assinatura: "http 0.9", "http 1.0" e "http 1.1". Desta forma, apenas uma assinatura se mostra capaz de reconhecer 3 cadeias de caracteres distintas. Tal poder de representação das REs se destaca quando se deseja representar infinitas cadeias de caracteres. Por exemplo, a assinatura "http/(0\.9|1\.0|1\.1)/ .*\.com" pode reconhecer qualquer cadeia que contenha os três casos citados anteriormente ("http 0.9", "http 1.0" e "http 1.1"), seguida de qualquer expressão que contenha o final ".com", como "http 1.0 google.com".

Tabela 1-1. Principais símbolos e significados utilizados em expressões regulares

Símbolo	Significado
Ancoradores	
^	Início de linha
\A	Início de cadeia de caracteres
$	Fim de linha
\Z	Fim de cadeia de caracteres
\b	Limite de palavra
\<	Começo de palavra
Classes de Caracteres	
\c	Caractere de controle
\s	Espaço em branco
\d	Dígito
\xhh	Caractere hexadecimal hh
Quantificadores	
*	0 ou mais
+	1 ou mais
?	0 ou 1
{3}	Exatamente 3
{3,}	3 ou mais
Intervalos	
.	Qualquer caractere, exceto linha em branco (\n)
(a|b)	a ou b
[abc]	a ou b ou c
[^abc]	Nem a nem b nem c
[a-q]	Entre a e q

Fonte: autoria própria

Com o objetivo de uma padronização das assinaturas visando à identificação de tráfego, o projeto L7-Filter [16] concentra 123 assinaturas de diversas aplicações e protocolos. A lista completa

pode ser vista na Tabela 7-1. Também é possível verificar a aplicabilidade dos operadores da Tabela 1-1, de forma a aumentar o poder de expressividade dos padrões.

Uma das maneiras de representar as expressões regulares se faz com o uso de autômatos finitos determinísticos. Em poucas palavras, um DFA é uma máquina abstrata, composta de estados e transições. Como definição formal [17], um autômato finito determinístico é uma 5-tupla (Q, Σ, δ, q_0, F), sendo:

- Um conjunto de estados finitos (Q);
- Um conjunto finito de símbolos de entrada (Σ);
- Uma função de transição ($\delta : Q \times \Sigma \rightarrow Q$);
- Um estado inicial ($q_0 \in Q$);
- Um conjunto de estados de aceitação ($F \subseteq Q$).

A Figura 1-2 mostra um exemplo de um DFA. Podemos verificar um conjunto de estados finitos Q = {q0, q1, q2} , um conjunto de símbolos de entrada Σ = {0,1} , uma função de transição δ = { (q0 × 0 → q1), (q0 × 1 → q0), (q1 × 0 → q1), (q1 × 1 → q2), (q2 × 0 → q1), (q2 × 1 → q0) }, um estado inicial q0 e um conjunto de estados de aceitação F = { q2 }.

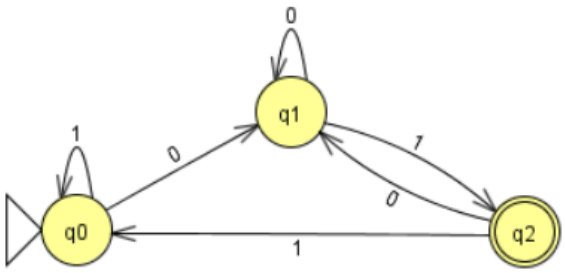

Figura 1-2. Exemplo de um DFA

Fonte: teste de doutorado [35]

Um DFA analisa uma cadeia de caracteres percorrendo cada *byte* da cadeia. Dado que todo DFA tem um estado inicial, para cada caractere, o DFA dispara uma transição, considerando o estado atual e o caractere de entrada. Este passo leva o DFA a um novo estado. Por fim, um DFA aceita uma cadeia quando o último caractere da cadeia faz o DFA chegar a um estado de aceitação.

Ainda de acordo com a Figura 1-2, podemos verificar que o DFA aceita as seguintes cadeias: 01, 101, 1101, 11001. Generalizando, o DFA citado aceita toda cadeia de comprimento maior ou igual a 2 que termine com a sequência 01. Segundo a própria teoria dos autômatos finitos

determinísticos, todo DFA possui uma RE que o represente [18]. Por exemplo, o autômato da Figura 1-3 poderia ser representando pela RE ^\x01[\x08\x09][\x03\x04], sem perda de significado ou generalidade.

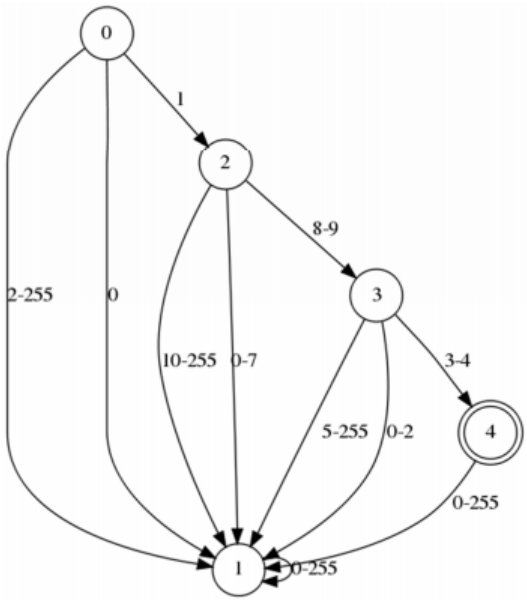

Figura 1-3. DFA que representa a RE ^\x01[\x08\x09][\x03\x04]

Fonte: teste de doutorado [35]

Os desafios computacionais envolvendo essas assinaturas/DFAs são notados quando se converte as REs em DFAs. Em [4], os autores mostram que determinadas condições, como os "ponto-estrela", podem favorecer uma "explosão de estados" (i.e. uma quantidade elevada de estados para representar apenas um símbolo da RE), quando se converte uma RE em um DFA. Tais conceitos teóricos se mostram mais do que essenciais quando se analisa a implementação dessas máquinas abstratas em uma arquitetura real de computador, como em [19]. Se for considerado que os conceitos abstratos de estados e transições são traduzidos para estruturas de dados reais, que alocam memória para sua concreta manipulação, uma explosão de estados pode chegar a ponto de consumir todos os recursos de memória disponíveis para um programa que utilize tal estrutura.

Além disso, deve ser também destacada a velocidade computacional que tais estruturas apresentam quando são utilizadas na análise de tráfego. Os autômatos citados anteriormente reconhecem os padrões analisando caractere a caractere do conteúdo do pacote. De forma sucinta, o casamento de um padrão com uma assinatura ocorre da seguinte forma:

1. A função de transição do autômato recebe um estado e um caractere de entrada;
2. É verificado se a saída do passo 1 (próximo estado) é um estado de aceitação;

3. Caso positivo, a verificação reporta um casamento positivo;
4. Caso negativo, o algoritmo volta para o passo 1, com o estado atual e o próximo caractere;
5. Caso o algoritmo consuma todos os caracteres da pacote sem reportar um casamento positivo, o algoritmo retorna sem sucesso.

Como este evento deve ocorrer da forma mais rápida possível, é necessário que a função de transição retorne o próximo estado no menor número de ciclos de processamento possível. Contudo, veremos no capítulo 2 que nem sempre isso é alcançado, o que pode provocar uma diminuição significativa de desempenho neste componente crítico do DPI.

Diante disso, pesquisas recentes têm apresentado constantes melhorias nos mecanismos internos que compõem um sistema DPI. Alguns exemplos incluem a otimização dos autômatos finitos utilizados, uso de placas gráficas como unidade de processamento paralelo e utilização de técnicas de aprendizagem de máquina para acelerar o processo de identificação dos fluxos na rede.

1.2 Trabalhos Relacionados

Objetivando minimizar o uso de memória de DFAs tradicionais, alguns estudos têm mostrado como comprimi-los de forma a reduzir o número de transições e salvar espaço em memória.

Até onde se sabe em relação à explosão de espaço de memória e tempo de processamento, Yang e Prasanna [25] propuseram o SFA, que representa um balanceamento entre redução de espaço de Autômatos Finitos Não-determinísticos (NFA) e o pequeno custo de tempo dos DFAs. Para os modelos de DFA, Kumar et. al. [8] apresentaram o D^2FA. Eles demonstraram que muitos estados compartilham alguns conjuntos de transições, estando aptos a substituí-los por apenas uma, chamada transição padrão. Tais transições especiais ligam estados e são disparadas quando não há transição correspondente para o símbolo de entrada. Em [3] e [7], Becchi et. al. propuseram o FastFA (A-DFA). Eles modificaram a criação destas transições padrão, o que tornou o tempo de travessia independente do tamanho do máximo caminho. Considerando que a travessia começa em um único estado, eles alcançaram um limite do pior-caso para o tempo de processamento. Quando comparado ao proposto em D^2FA [8], o FastFA alcança a mesma taxa de compressão com menos consumo de memória. Além disso, os autores realizaram um estudo de viabilidade da combinação do algoritmo proposto com redução de alfabeto e DFA *multistride*. Por fim, também foram discutidas por [3] diferentes codificações de memória. Em uma abordagem diferente, Antonello et. al [2] introduziram o RCDFA, apresentando uma nova maneira de representar as transições. Eles

reduziram a compressão das transições do DFA de tal forma que uma transição representa não apenas um caractere, mas uma faixa deles. Isto reduziu a quantidade de memória necessária para representar as transições. Cerca de 97% de redução em espaço em memória utilizado foi alcançado em comparação ao DFA original e aproximadamente 93% em relação a técnicas existentes. . Em todas as pesquisas apresentadas até este ponto, é possível observar que o foco principal está na economia de memória.

Para o caso de aplicação real dos modelos de DFA em arquiteturas de sistemas reais, a Codificação Linear (LE) [20] é apresentada como a forma de codificar transições de um DFA linearmente (uma após a outra), espalhadas pela memória. Com o LE, um estado com l transições é codificado usando l+1 palavras (de memória), sendo a primeira transição a transição padrão. A travessia pelos estados começa a partir da primeira palavra e continua sequencialmente até a última. Codificação Bitmap (BE) [20] foca em melhorar a travessia sequencial do LE, codificando cada estado com um array de bitmap e l+1 palavras. A travessia analisa primeiro o array de bitmap e determina se existe uma transição para o símbolo de entrada analisado. Então, ela conta os bits precedentes e calcula o ponteiro do endereço da transição a ser disparada. Char-State [5][22] é um layout teórico que foi modificado em [2] para arquiteturas orientadas a endereçamento por byte. Baseado na ideia de que, em muitos casos, transições chegando a um estado são rotuladas com o mesmo caractere, eles usaram o conceito de "identificador relativo", o que implica em um menor número de bits para representar as transições e menor espaço de memória para armazená-las.

Sobre as formas de avaliação dos DFAs e dos *layouts* de memória, em [4], é sugerida uma forma diferente de avaliar o desempenho dos DFAs, separando a análise em dois níveis: o nível do modelo, para representar o comportamento teórico de um autômato finito, e o nível do *layout* de memória, para representar o modelo em termos de estruturas de dados. Também são mostrados os benefícios de se analisar o par modelo/*layout* de forma desacoplada, destacando a utilização de memória dentre as várias combinações entre os dois níveis descritos.

Ao longo da última década, vários estudos foram realizados a fim de avaliar o desempenho computacional de um sistema DPI. Entretanto, as pesquisas abordam diferentes aspectos do sistema para alcançar seus resultados. É importante destacar que a maioria dessas técnicas são complementares e podem ser aplicadas em conjunto a fim de alcançar melhores resultados.

Diversas pesquisas nessa área estão relacionadas à compressão de autômatos finitos determinísticos. Em [6], foi proposto o agrupamento de diferentes autômatos. Em outras palavras, um único autômato é utilizado para representar um conjunto de autômatos (ou um subconjunto dos autômatos contidos em um sistema DPI). Em comparação com outros trabalhos da literatura, os

resultados obtidos pelos autores demonstraram ganhos de até 700 vezes a velocidade de processamento original

Outras partes do sistema DPI foram modificadas por diversos trabalhos a fim de alcançar melhorias de desempenho. Considerando a velocidade de casamento, o trabalho apresentado por [12] apresentou a arquitetura FBTI (Flow-based Traffic Identification). Essa abordagem propõe a utilização de uma Unidade de Processamento Gráfico (GPU) para aumentar a capacidade de processamento e, assim, acelerar a velocidade de casamento de um sistema DPI. Os autores desse trabalho utilizaram apenas os *bytes* iniciais de cada fluxo na classificação para otimização ainda maior do desempenho do sistema. No cenário de melhor caso, a arquitetura proposta alcançou uma vazão de aproximadamente 120 Gbps.

Outra técnica bastante interessante é proposta em [20], em que os autores propuseram que a inspeção apenas dos 7 primeiros pacotes de cada fluxo seria suficiente para classificar de maneira satisfatória o tráfego e, em contrapartida, reduziria o tempo de processamento. Os resultados demonstraram um decréscimo de aproximadamente 75% no tempo de processamento em comparação à abordagem em que todos os pacotes do fluxo são inspecionados.

Por fim, o arcabouço proposto por [9] analisa o impacto do uso de diferentes bases de assinaturas, avaliando e comparando suas complexidades. Os resultados expostos se baseiam na execução de um mesmo DPI com conjuntos de assinaturas diferentes. Nesse contexto, os autores concluíram que não é justa a comparação do desempenho de um DPI utilizando bases de assinaturas de diferentes complexidades.

Em resumo, é possível perceber que os trabalhos da literatura nessa área, de maneira geral, estão relacionados à compressão de autômatos, à plataforma utilizada pelo sistema DPI e à complexidade de diferentes listas de assinaturas. Será visto ao longo deste trabalho que ainda existem pontos em aberto na literatura a serem estudados, como a combinação de DFAs e *layouts*, como também a ordenação das assinaturas durante o processo de identificação de tráfego. Estes dois temas, quando bem compreendidos, podem proporcionar benefícios significativos na velocidade de classificação de analisadores de tráfego.

1.3 Objetivos

De maneira geral, as melhorias em sistemas DPI podem ser agrupadas em dois grandes grupos: as que otimizam o consumo de memória[2][21][5][7][8] e as que aumentam a vazão de análise do sistema DPI [11][10]. Com o objetivo de aumentar o desempenho computacional em termos de velocidade de processamento dos pacotes de entrada num sistema DPI, esta dissertação

possui como objetivo principal avaliar e otimizar os componentes internos de um sistema DPI, a fim de melhorar a velocidade de processamento e, consequentemente, a inspeção de pacotes.

Além deste objetivo principal são apresentados alguns objetivos específicos:

1) Analisar o desempenho computacional dos diferentes tipos de DFAs em arquiteturas de computadores reais;
2) Propor uma solução para otimizar o processamento de um sistema DPI, inserindo custo de processamento em assinaturas de aplicações de rede.

1.4 Estrutura da Dissertação

A dissertação se estrutura em torno de dois temas complementares para otimizações de mecanismos DPI. Um deles é a revelação dos problemas associados aos autômatos e suas implementações em memória. O outro é a análise do impacto da lista de assinaturas no desempenho de um analisador DPI.

Considerando que os temas são complementares e ortogonais em suas análises, a estrutura desta dissertação concentra em cima de dois grandes capítulos. A metodologia de avaliação, resultados e lições aprendidas de cada grande área estão incluídos no próprio capítulo.

O capítulo 2 trata dos problemas de desempenho em sistemas DPI. Em poucas palavras, ele trará uma avaliação dos principais autômatos otimizados combinados com alguns *layout*s de memória. Os resultados mostram como más combinações entre esses dois componentes podem prejudicar um DPI, além de sugerir quais as melhores combinações. O capítulo 3 faz uma análise profunda sobre a lista de assinaturas e mostrando o impacto da sua ordenação na velocidade de análise do DPI. Além disso, serão apresentados mecanismos de ordenação dinâmica capazes de otimizar o processo em tempo de execução, beneficiando o DPI como um todo. Por fim, o capítulo 4 trará a conclusão desta dissertação. Nele, serão apresentadas as contribuições alcançadas com este trabalho, além das 3 publicações em artigos científicos originadas a partir desta dissertação e outras duas publicações em temas correlatos onde o autor também utilizou os conhecimentos adquiridos sobre este assunto.

2 Revelando Problemas de Desempenho em Sistemas DPI

"A verdade da questão é que sempre sabemos a coisa certa a ser feita. A parte difícil é fazê-la."

General H. Norman Schwarzkopf

Autômatos finitos determinísticos comprimidos prometem o mesmo poder de representação dos DFAs tradicionais, enquanto usam menos memória para representar expressões regulares. Avaliações experimentais de sistemas de DPI baseados em DFA focam principalmente em consumo de memória sem observar outros aspectos importantes relacionados, tais como a velocidade de casamento das assinaturas. Um design apropriado dos sistemas de DPI requer a avaliação de várias métricas de desempenho (e.g. instruções de CPU executadas), de forma a ter certeza que a implementação não comprometerá o desempenho geral. Este capítulo propõe uma avaliação nova e sistemática dos DPIs, revela o impacto das estruturas de dados dos DFAs e a implementação dos *layouts* de memória correspondentes em métricas em nível de *hardware*. Resultados experimentais mostram que alguns modelos de DFA e de *layout* de memória combinados são quase 100 vezes mais rápidos que outros. Além disso, eles também mostram que escolher o par modelo-*layout* de forma incorreta pode levar a problemas de desempenho significativos. A metodologia apresentada e os resultados certamente contribuem com o projeto eficiente de mecanismos de DPI, através da seleção das melhores combinações de modelos de DFAs e *layouts* de memória a fim de alcançar o desempenho geral objetivado.

De forma resumida, este capítulo avalia o impacto de escolhas de projeto em DPI baseados em DFA, analisado tanto as métricas relacionadas ao DFA como à CPU. Levando em consideração fatores de desempenho, tais como modelos de DFA, layouts de memória, conjuntos de assinaturas e a probabilidade de casamento, este capítulo foca em revelar problemas de desempenho em sistemas DPI.

O resto deste capítulo se organiza da seguinte maneira: seção 2.1 introduz uma contextualização sobre o assunto. Seção 2.2 mostra a metodologia de avaliação, enquanto que seção 2.3 apresenta e discute as métricas de baixo nível e as dinâmicas. Seção 2.4 destaca algumas descobertas importantes. Finalmente, seção 2.5 apresenta as lições aprendidas com o conteúdo apresentado.

2.1 Contextualização

Nos anos recentes, pesquisadores e desenvolvedores têm desenvolvido ferramentas de identificação de tráfego de rede baseadas em técnicas de DPI de forma a ter um conhecimento profundo da interconexão entre redes de acesso e redes *backbone* [14][16][1][22]. De forma abrangente, ferramentas de DPI reconhecem padrões dentro do conteúdo dos pacotes em um enlace de rede. Muitas delas são baseadas em máquinas abstratas conhecidas como DFA, pois elas reconhecem padrões expressos por expressões regulares. Embora poderosas, REs expressas como DFAs podem resultar em representações extremamente grandes[4]. A fim de diminuir o tamanho dos DFAs, estudos recentes apresentam como comprimi-los, enquanto mantém intacto o seu poder de representação[2][5][23][8].

Argumenta-se neste trabalho que uma análise de desempenho das suas representações teóricas dá apenas uma visão parcial do desempenho geral. É importante codificar e entender seus desempenhos em arquiteturas de hardware reais. *Layouts* de memória são as estruturas de dados que representam os autômatos finitos. Estudar estes *layouts* desacoplados dos modelos de DFA é o primeiro passo para entender o impacto de combinar estas duas camadas. Uma recente pesquisa [19] mostra o impacto do par "modelo de DFA" e "*layout* de memória" a partir da quantidade de memória utilizada. Embora os resultados que foram apresentados indiquem direções para projetar analisadores de tráfego de rede mais eficientes (e.g. desenvolver em um sistema embarcado), a análise de desempenho ali realizada não considerou a velocidade de processamento das duas camadas combinadas (camada de DFA e camada do layout de memória). Na verdade, para todos os DFAs comprimidos para sistemas DPI na literatura [2][3][4][5][23][7][6], não há análise de desempenho de processamento em redes de alta velocidade. Se o par modelo-*layout* escolhido para um certo DPI não é rápido o suficiente para manipular os fluxos de pacotes recebidos (e.g. em um enlace de rede de 10Gbps), uma substancial perda de pacotes irá ocorrer, o que, consequentemente, terá um impacto na precisão do sistema. Além disso, existem diferenças de desempenho significantes quando comparados apenas os aspectos de desempenho teórico dos modelos de DFA para o desempenho de hardware do par modelo-*layout*. Pode-se tomar como exemplo a análise do FastFA[7] e D^2FA[5]. Analisando apenas a camada de modelo do DFA, o FastFA dispara mais transições do que o D^2FA. Este fato poderia induzir a se pensar que o FastFA desempenhará pior do que D^2FA. Contudo, quando analisado o FastFA com o layout de memória Codificação Linear [21], nota-se que esta combinação utiliza menos instruções de CPU do que o D^2FA com o layout de memória Codificação Bitmap. Portanto, escolher erroneamente o layout de memória pode impactar significativamente nas melhorias feitas nos modelos teóricos.

Até onde se sabe, este trabalho, juntamente com o apresentado em [19], são os primeiros a apresentar uma análise rigorosa das combinações entre os diversos modelos de DFA e de *layouts* de memória, quando se observa sistemas DPI. Aqui, se objetiva o entendimento do impacto real dessas duas partes (autômatos e *layouts*) em projetos que requeiram alto desempenho em termos de velocidade de processamento. As contribuições deste capítulo são as seguintes:

i. Analisam-se métricas dinâmicas (i.e. relacionadas a modelos de DFA) e mostra-se o impacto dos mais importantes conjuntos de assinaturas (i.e. conjuntos de padrões) e o perfil de tráfego de rede no desempenho dos modelos de DFA;

ii. Analisam-se métricas de baixo nível (i.e. relacionadas a métricas de CPU) e verifica-se o impacto de diferentes combinações de modelos de DFA e *layouts* de memória no desempenho geral do sistema;

iii. Discutem-se os impactos de escolhas erradas de *layouts* de memória ao projetar analisadores de tráfego.

Neste capítulo serão analisados os modelos DFA padrão (RawDFA), D^2FA[5], FastFA[7], e RCDFA[2]. Para os layouts de memória, foram escolhidas as codificações Matricial (ME), Bitmap (BE), Linear (LE) e Char-State [5]. Foram usados conjuntos de assinaturas bem conhecidos na literatura (i.e. L7-Filter [16] e SnortWeb [22]) e o N-NGEN [24] como gerador de tráfego. A ferramenta PAPI [25] foi utilizada como ferramenta de análise de perfil de código.

Resultados experimentais apresentados neste trabalho confirmam que alguns *layouts* de memória, tais como CS e BE, degradam o desempenho em quase 100 vezes se comparado com o tradicional ME. Também é revelado que o LE varia o seu desempenho em até 10 vezes, dependendo do conjunto de padrões escolhido. Além disso, o conjunto de assinaturas influencia significativamente o desempenho dos layouts de memória. Também é notado que alguns *layouts* de memória ineficientemente projetados têm consequências negativas nas métricas de *hardware*, tais como ciclos de CPU e L1 *cache miss*. Argumenta-se e mostra-se que a escolha do par DFA-*layout* é crucial para o projeto de um analisador de tráfego de rede rápido. Além disso, conclui-se que *layouts* de memória escolhidos de forma inadequada podem causar perdas de desempenho significativas para os sistemas DPI.

2.2 Avaliação Metodológica

Nesta sessão são analisados os DFAs considerando dois grupos de métricas, denominadas *métricas dinâmicas* e *métricas de baixo nível*. A primeira representa o comportamento dos modelos de DFA de

acordo com o perfil de tráfego analisado. Métricas dinâmicas não variam para todos os modelos de DFA, de forma que elas são apresentadas como uma única série na seção de "Resultados e Discussão". O segundo grupo de métricas está relacionado à arquitetura de CPU, representando o desempenho do *hardware* para as combinações de modelos e *layouts* estudados. Em resumo, ambos os grupos de métricas medem como o par modelo-*layout* afeta o desempenho geral de um sistema DPI. Observa-se que foi utilizada a mesma metodologia proposta em [19].

O impacto do perfil de tráfego da rede nas características do par para ambos os grupos de métricas também foi avaliado. *Traces* de pacotes sintéticos foram gerados com as seguintes características: pacotes de 710 *bytes*, 10 pacotes por fluxo e 100 fluxos por conjunto de assinaturas. Tais parâmetros são baseados no *framework* de avaliação de desempenho para sistema DPI baseados em DFA [9]. A ferramenta N-NGEN foi utilizada para gerar tais pacotes [24]. Em resumo, a ferramenta gera fluxos de pacotes de rede com padrões, baseado em dois fatores: o conjunto de assinaturas e a probabilidade de avanço dentro do DFA. Para gerar um novo padrão, o N-NGEN pega uma expressão regular de um conjunto de assinaturas (passado como entrada) e constrói o DFA correspondente. Após isso, ele gera uma sequência de caracteres de forma a alcançar os estados mais profundos do DFA, disparando transições baseado na probabilidade de avanço. Se a probabilidade é alta (i.e. p=0,95), o padrão gerado terá altas chances de alcançar os estados mais profundos do DFA, aumentando a possibilidade de casamento. Nos cenários que serão aqui apresentados, a probabilidade de casamento é configurada para quatro valores: 0,35 , 0,55 , 0,75 , 0,95 .

Foram escolhidos dois conjuntos de assinaturas como entrada, denominados L7-Filter e SnortWeb. Como descrito em [9], os conjuntos de assinaturas L7-Filter e SnortWeb diferem em tamanho, complexidade e no número de sub-padrões. O conjunto L7-Filter é menos complexo e menor do que o SnortWeb. Escolhendo esses dois conjuntos, é garantido que os resultados cobrem uma grande variedade de características dos principais conjuntos de assinaturas.

As métricas de desempenho são descritas na Tabela 2-1.

Tabela 2-1. Métricas Dinâmicas

Métrica	Significado
Estados Visitados	Número de estados visitados
Transições Disparadas	Número de transições disparadas, relacionadas a qualquer DFA
Máxima Profundidade Alcançada	Máxima profundidade alcançada enquanto o autômato está em execução
Transições DFA disparadas	Número de todas as transições disparadas (padrão e outras), relacionadas a um DFA específico
Transições padrão disparadas	Número apenas das transições padrão disparadas enquanto o autômato está em execução

Fonte: autoria própria

Para as métricas de baixo nível, foi usada uma ferramenta de análise de código chamada PAPI[25], onde o mecanismo DPI foi isolado e medido. Isso significa que não foi permitida a interferência de outros componentes na avaliação do modelo de DFA junto com o *layout* de memória. As métricas de baixo nível analisadas foram *Instruções Executadas, Ciclos de CPU* e *L1 Cache Miss*. Os DFAs analisados foram o RawDFA (referencial), RCDFA (sem otimizações), D^2FA e FastFA. Para os *layouts* de memória foram considerados as codificações Matriz (ME) (referencial), Linear (LE), Bitmap (BE) e Char-State (CS). Tabela 2-2 resume os fatores e níveis dos experimentos realizados.

Tabela 2-2. Fatores e Níveis

Fatores	Níveis
Probabilidade de Casamento	0,35 ; 0,55 ; 0,75 ; 0,95
Conjunto de Assinaturas	L7-Filter ; SnortWeb
Modelos DFA	RawDFA ; D^2FA ; FastFA ; RCDFA
Layouts de Memória	ME ; LE ; BE ; CS

2.3 Resultados e Discussão

Para facilitar o entendimento serão analisadas primeiramente as métricas dinâmicas. Depois, serão explicadas as métricas de baixo nível, correlacionando ambos os grupos de métricas.

2.3.1 Métricas Dinâmicas

A Figura 2-1, Figura 2-2 e Figura 2-3 mostram a profundidade máxima alcançada, estados visitados e o número de transições disparadas respectivamente. São mostrados os resultados experimentais para os modelos de DFA comprimidos (FastFA, D^2FA, RCDFA) na mesma série do gráfico devido ao fato de tal métrica se comportar igualmente ao RawDFA.

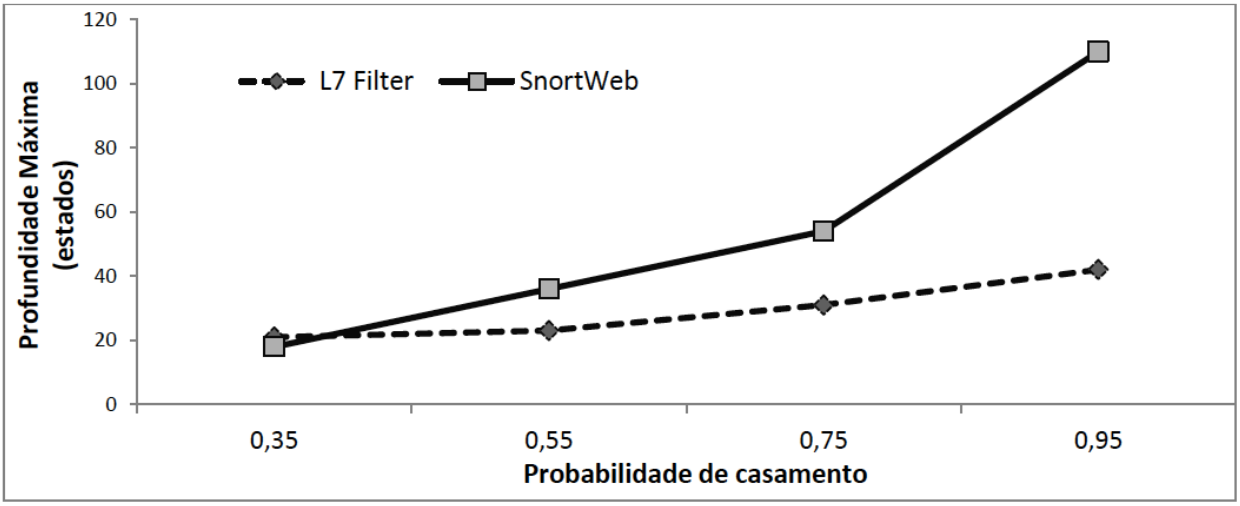

Figura 2-1. Máxima Profundidade Alcançada por um DFA

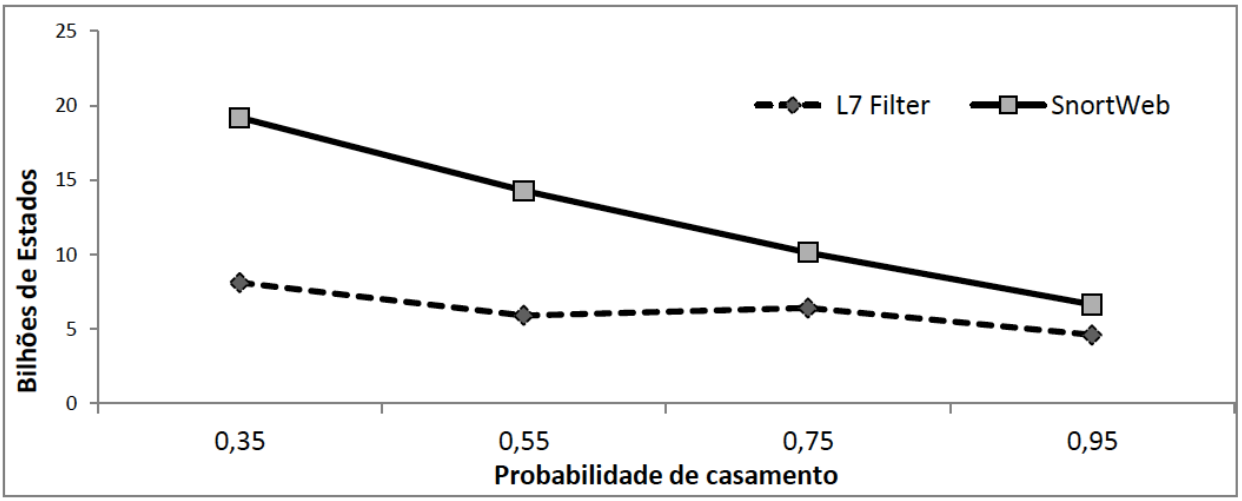

Figura 2-2. Estados Visitados por um DFA

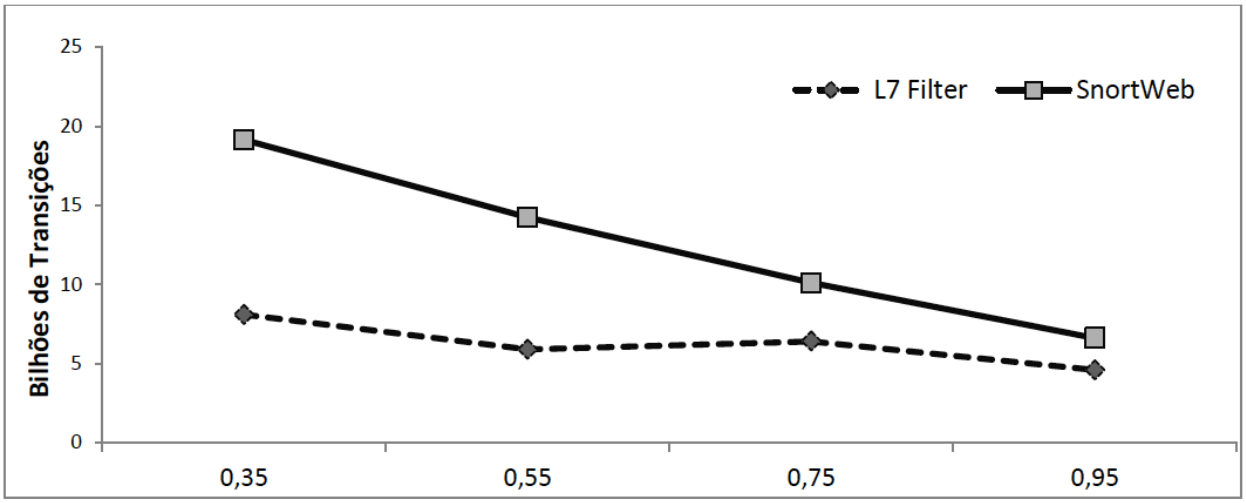

Figura 2-3. Transições Disparadas por um DFA

Para o valor de probabilidade de casamento igual a 0,35 (p=0,35), o número de estados visitados e transições disparadas (Figura 2-2 e Figura 2-3, respectivamente) para o SnortWeb é cerca de 75% maior do que para o L7-Filter. Contudo, essa diferença tende a diminuir com o aumento da probabilidade. Para p=0,95, SnortWeb é cerca de 45% maior para as transições disparadas. Figura 2-1 mostra a máxima profundidade alcançada para estes cenários. Como se pode notar, SnortWeb e L7-Filter alcançam praticamente a mesma profundidade (18 e 21, respectivamente), para p=0,35, embora o conjunto de assinaturas SnortWeb produza assinaturas maiores do que o conjunto L7-Filter, como descrito em [9].

Para a probabilidade p=0,95, SnortWeb mostra um valor mais alto para a máxima profundidade alcançada. Tais fatos podem ser explicados da seguinte forma: para baixas probabilidades de casamento (e.g. p=0,35), a carga útil dos pacotes possui maior probabilidade de parar durante a travessia do DFA logo nos primeiros estados. Isso leva a valores menores de máxima profundidade alcançada, independente do conjunto de assinaturas escolhido. Além disso, o sistema baseado no DFA precisará comparar mais assinaturas com os fluxos sem casamento. Se o conjunto de assinaturas é grande (como o do SnortWeb), o sistema irá analisar muitas assinaturas, tentando encontrar um casamento positivo. Considerando o conjunto de assinaturas completo, tal processo dispara mais transições e visita mais estados. Isso explica o motivo da diferença entre a Figura 2-2 e a Figura 2-3 ser maior para p=0,35 do que para p=0,95.

Para analisar os efeitos dos *traces* nos modelos de DFA, foram consideradas o número de transições DFA disparadas (Figura 2-4 e Figura 2-5) e o número de transições padrão disparadas (Figura 2-6 e Figura 2-7). A partir deste ponto, começam a ser considerados os diferentes modelos de DFA e suas características e comparadas suas dinâmicas de acordo com os cenários escolhidos.

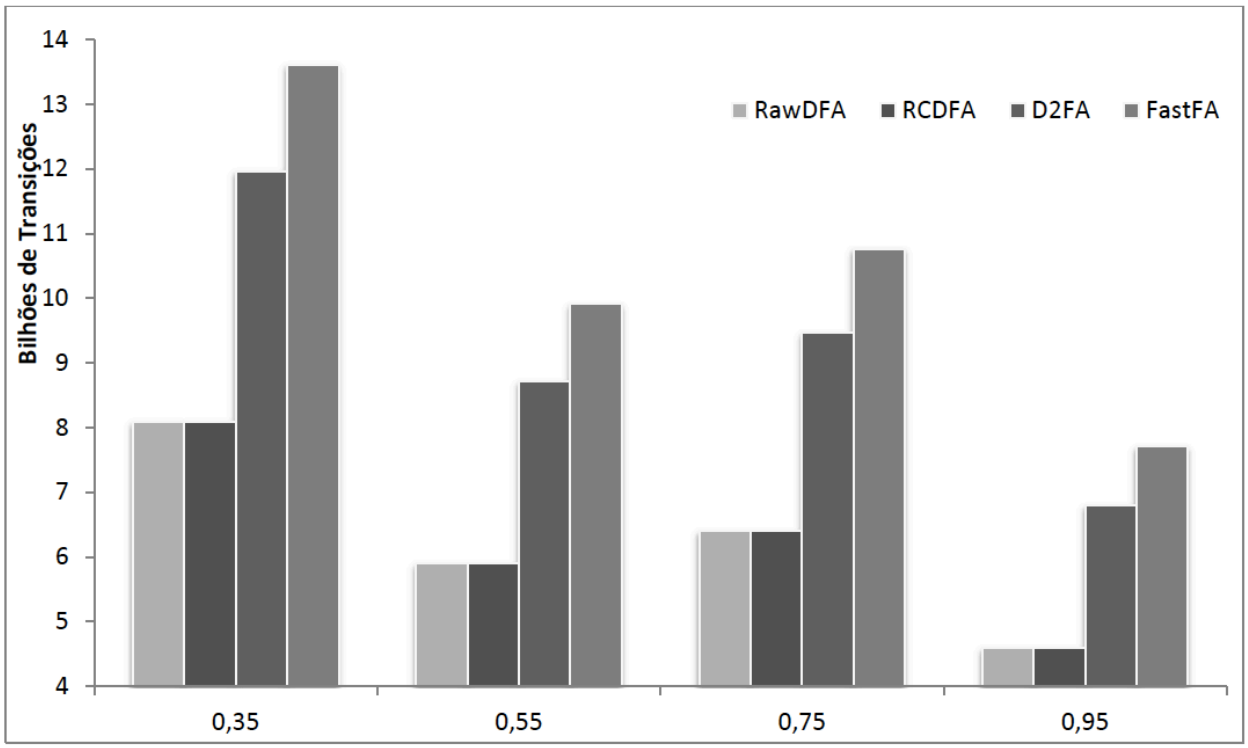

Figura 2-4. Transições DFA disparadas para o L7-Filter

A Figura 2-4 apresenta os resultados para o conjunto de assinaturas L7-Filter. O referencial é o RawDFA (i.e. o DFA original e sem compressão). É possível notar que o RCDFA é o DFA comprimido que melhor se comportou em todos os cenários. Ele disparou o mesmo número de transições que o referencial (i.e. 8,09 * 10^9 transições para p=0,35). D^2FA ativa perto de 12,1 * 10^9 transições para p=0,35. FastFA é o pior modelo, disparando cerca de 5,5 * 10^9 mais transições do que o referencial (p=0,35). Isto representa quase 70% mais transições do que o referencial. Para p=0,95, a diferença é similar: cerca de 48% mais transições para o D^2FA e cerca de 68% para o FastFA. Um comportamento similar ocorre para o conjunto SnortWeb (Figura 2-5). Neste cenário, a diferença é menor, mas ainda se mantém significante. Para p=0,35, o referencial mostrou 1,92 * 10^{10} transições, o mesmo que o RCDFA. O FastFA tem 2,4 * 10^{10} transições, enquanto que o D^2FA tem 2,45 * 10^{10} transições. Isto representa cerca de 28% mais transições. Para poder explicar estes fatos é preciso considerar que ambos D^2FA e FastFA se utilizam de transições padrão para alcançar a compressão objetivada. Contudo, autômatos comprimidos com transições padrão apresentam uma desvantagem de desempenho: eles disparam mais transições durante o processo de casamento de padrões. Como o RCDFA não se utiliza desta técnica, ele dispara a mesma quantidade de transições que o RawDFA.

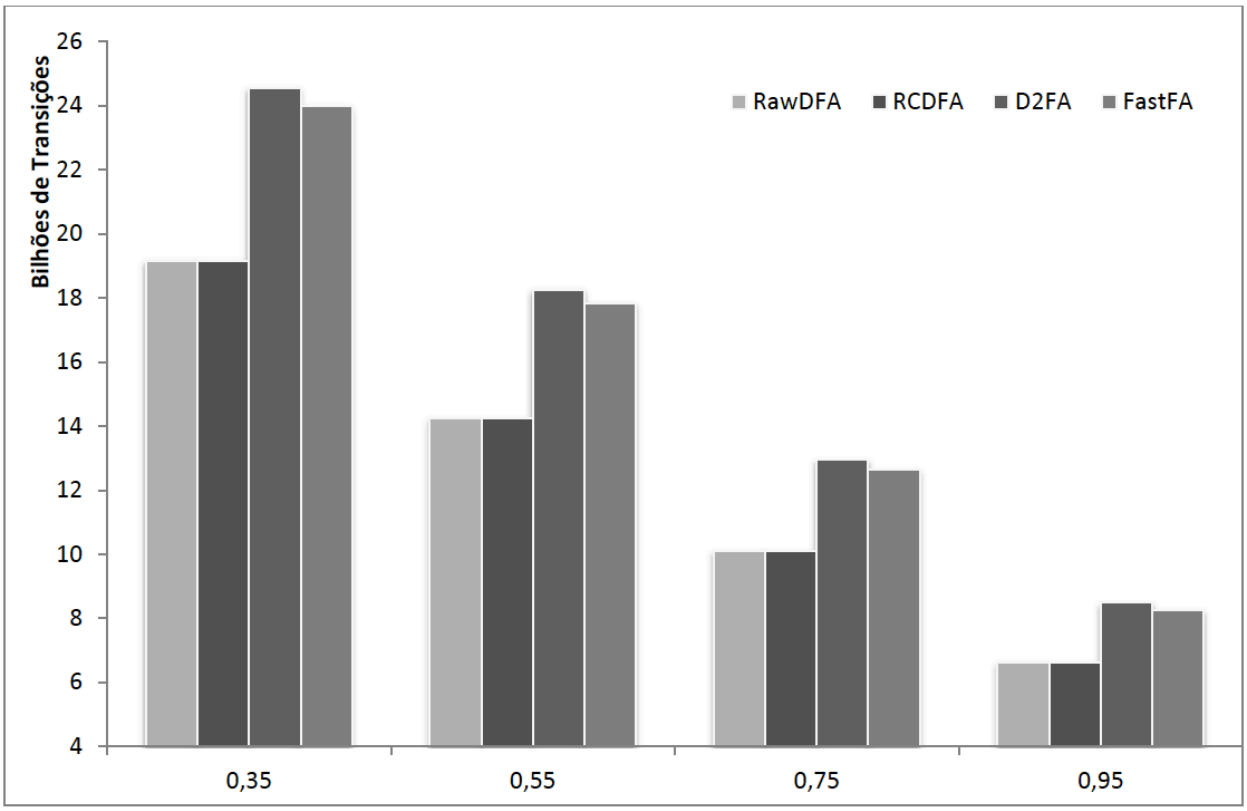

Figura 2-5. Transições DFA disparadas para o SnortWeb

Analisando a Figura 2-5, pode-se perceber uma tendência declinante iniciando em p=0,35 e terminando em p=0,95 que pode ser explicada da seguinte forma: quanto maior a probabilidade de avanço, maior é a probabilidade de se conseguir um casamento com o conteúdo do pacote. Consequentemente, mais rápido o mecanismo DPI encontrará a assinatura correspondente ao padrão analisado. Este fato reduz a quantidade de pesquisas desnecessárias à base de assinaturas e travessias dentro dos DFAs, resultando em menos transições disparadas. Figura 2-6 e Figura 2-7 apresentam os resultados para o número de transições padrão disparadas com a base do L7-Filter e do SnortWeb, respectivamente. Novamente, fica claro o impacto do conjunto de assinaturas nas transições padrão.

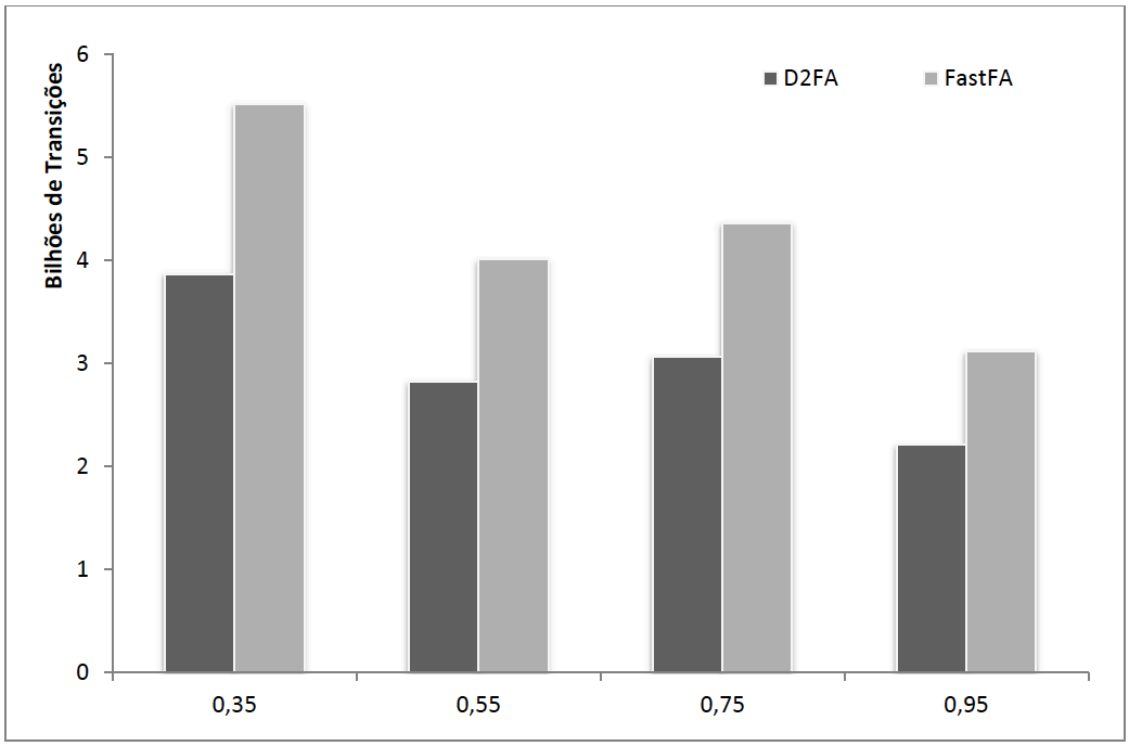

Figura 2-6. Transições Padrão disparadas para o L7-Filter

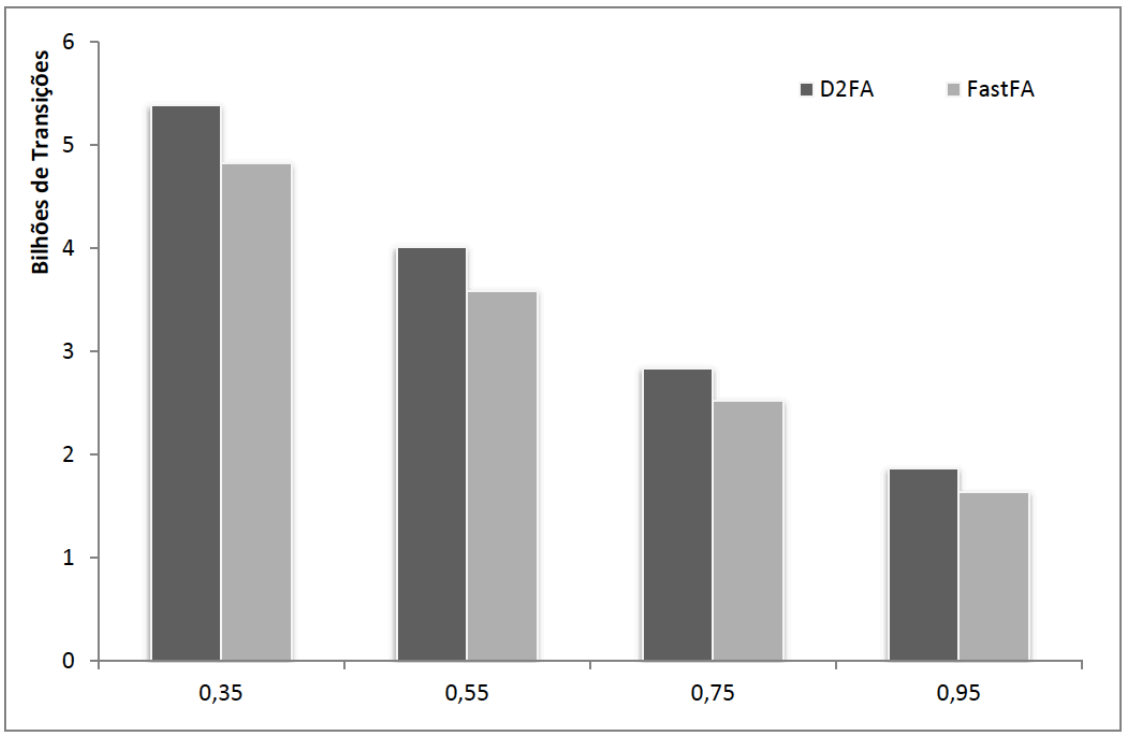

Figura 2-7. Transições Padrão disparadas para o SnortWeb

2.3.2Métricas de Baixo Nível

A fim de avaliar o desempenho dos *layouts* de memória foram medidos o número de instruções executadas, o número de ciclos de CPU, e L1 *Cache Miss*. As comparações realizadas não incluem o

RCDFA. A razão é que o RCDFA não pode ser implementado com os *layouts* de memória conhecidos, devido ao seu conceito de utilizar uma transição para representar um conjunto de caracteres. Dessa forma, esta análise foi deixada para um trabalho futuro. Os resultados para as instruções executadas podem ser analisados na Tabela 2-3. Novamente, o referencial considerado é o RawDFA com a layout de memória Matriz (ME). Não é possível encontrar o *layout* referencial em combinação com os autômatos D^2FA e FastFA devido ao fato de que não é possível representar transições padrão no *layout* matricial.

Tabela 2-3. Instruções CPU Executadas (10^{11})

	L7-Filter				**SnortWeb**			
	0.35	***0.55***	***0.75***	***0.95***	***0.35***	***0.55***	***0.75***	***0.95***
D^2FA/BE	396,3	297,8	315,9	235,5	682,8	510,9	393,5	287,2
D^2FA /LE	48,2	29,2	36,7	20,4	276,2	203,0	123,5	61,7
D^2FA /CS	275,0	206,9	219,3	164,0	482,0	360,5	277,8	203,0
RawDFA/ME	**2,4**	**1,7**	**1,9**	**1,3**	**5,7**	**4,3**	**3,0**	**2,0**
FastFA/BE	396,2	297,7	315,8	235,4	682,5	510,7	393,4	287,0
FastFA/LE	51,5	31,6	39,6	22,5	275,8	202,5	123,2	61,7
FasFAt/CS	266,2	200,4	212,3	159,0	482,5	360,8	278,1	203,2

Se forem analisados os resultados do D^2FA para o cenário L7-Filter e p=0,35, veremos que o *layout* Linear (LE) é 5,7 vezes mais rápido que o *layout* Char-State (CS), e 8,2 vezes mais rápido que o *layout* Bitmap (BE). Este comportamento similar também é percebido em outras probabilidades, assim como para o autômato FastFA, no cenário com o L7-Filter. Para o conjunto do SnortWeb, esta vantagem é um pouco menor, mas ainda significante. Para o D^2FA e p=0,95, o LE é 4,6 vezes mais rápido que o BE e quase 3,3 vezes mais rápido que o CS. Esta explicação vem da arquitetura dos *layouts*. Este fato acontece devido à quantidade de processamento que ambos CS e BE precisam realizar para recuperar o ponteiro do próximo estado. Levando isso em consideração, podemos verificar o alto impacto que um *layout* de memória tem no desempenho do par modelo-*layout*.

Também podem ser notadas as diferenças entre os *layouts* de memória não tradicionais (BE, LE e CS) e o referencial (ME). Se for comparado apenas o LE com o referencial (ME), será notado que o menor valor de LE é mais de 15 vezes superior ao ME (para p=0,95). Considerando o CS, as diferenças vão desde 84,5 vezes (para SnortWeb, p=0,55) a 126,5 vezes (para L7-Filter, p=0,95). O BE apresenta os piores resultados, tendo uma diferença de 118,76 vezes (SnortWeb, p=0,55) a 181,15 vezes (L7-Filter, p=0,95).

Os resultados citados acima podem ser explicados da seguinte maneira: o RawDFA foi desenvolvido considerando que cada símbolo do alfabeto do autômato (∑) tem uma transição para um próximo estado. O layout ME, dessa forma, tem uma transição para cada símbolo de todos os estados, tornando rápido o processo de procura da próxima transição. De forma diferente, os outros *layouts* de memória (LE, BE e CS) foram desenvolvidos para salvar espaço de memória, comprometendo o processo de procura das transições. Para o caso do layout LE, durante a procura da próxima transição, o algoritmo precisa buscar as transições do estado uma após uma, até se encontrar a correspondente ao símbolo de entrada. Para o caso do BE, a estrutura em forma de *bitmap* implica que o algoritmo de busca das transições conte os *bits* precedentes correspondentes ao símbolo de entrada, gastando, assim, tempo demais neste processo. Finalmente, o CS precisa utilizar uma tabela de endereçamento indireto a fim de recuperar o ponteiro do próximo estado, demandando mais ciclos de CPU neste processo.

A fim de avaliar as diferenças entre os resultados teóricos dos modelos de DFA com o desempenho deles em arquiteturas de *hardware* reais, foram comparados o número de transições disparadas (Figura 2-4 e Figura 2-5) com o número de instruções executadas (Tabela 2-3). Tomando como exemplo o cenário com L7-Filter e p=0,35 (Figura 2-4), pode-se notar que o D^2FA dispara cerca de $1,6 * 10^9$ menos transições que o FastFA. Isto representa cerca de 14% menos transições. Poderia se pensar que esta vantagem do D^2FA sobre o FastFA seria mantida quando fossem analisadas as instruções executadas. Contudo, considerando o mesmo *layout* de memória, para o conjunto do L7-Filter, o D^2FA utiliza mais instruções do que o FastFA (para o layout CS).

Tal problema de desempenho se agrava quando são consideradas as diferentes combinações entre modelos e layouts. Modelos DFA podem ser codificados usando vários tipos de *layouts* de memória diferentes. Dessa forma, é bem possível realizar uma comparação entre o D^2FA codificado com o BE e o FastFA codificado com o LE. Desta forma, para o conjunto L7-Filter com p=0,95, o D^2FA/BE utilizou $253,5 * 10^{11}$ instruções, enquanto que o FastFA/LE utilizou $22,5 * 10^{11}$ instruções. Portanto, o D^2FA/BE utilizou cerca de 7,7 vezes mais instruções que o FastFA/LE, tornando irrelevantes os 14% de ganho obtidos em disparar menos transições na camada de DFA. A partir desses resultados, conclui-se que a escolha errada do *layout* de memória pode reduzir (ou até mesmo eliminar) os benefícios alcançados na otimização do modelo DFA.

A Tabela 2-4 mostra os resultados para a métrica Ciclos de CPU. Embora alguns resultados sejam similares ao número de instruções executadas (Tabela 2-3), é possível ainda analisar um ponto interessante. Considerando as instruções executadas, o layout BE tem valores mais altos que CS. Contudo, analisando os ciclos de CPU (Tabela 2-4), o comportamento é o oposto: CS é mais lento que BE. Isso pode ser explicado analisando o L1 *Cache Miss* na Tabela 2-5. Como se pode ver, para

todos os cenários, o *layout* BE tem valores melhores de L1 *Cache Miss* do que o CS. As consequências de um maior L1 *Cache miss* são que a CPU precisa buscar informações fora da cache L1. Isso implica em mais ciclos de CPU gastos, o que degrada o desempenho geral, similar no cenário do layout CS. Ainda fazendo comparações entre a Tabela 2-4 e a Tabela 2-5, pode-se perceber que o layout LE é o mais rápido (em ciclos de CPU) para o DFA comprimido, mas ainda assim mais lento do que o referencial (ME).

Tabela 2-4. Ciclos de CPU gastos (10^{11})

	L7-Filter				**SnortWeb**			
	0.35	***0.55***	***0.75***	***0.95***	***0.35***	***0.55***	***0.75***	***0.95***
D²FA/BE	179,7	135,0	142,8	106,3	322,8	240,5	182,8	131,3
D²FA /LE	30,1	18,1	22,8	12,5	178,8	130,9	79,1	39,1
D²FA /CS	186,4	140,2	148,6	111,1	326,5	244,3	188,2	137,3
RawDFA/ME	**1,6**	**1,1**	**1,2**	**0,9**	**3,8**	**2,8**	**2,0**	**1,3**
FastFA/ BE	179,7	135,0	142,8	106,3	322,5	240,3	182,6	131,1
FastFA / LE	32,4	19,7	24,7	13,9	178,5	130,6	78,9	39,1
FastFA / CS	181,0	136,2	144,3	108,0	329,2	246,2	189,4	138,2

Tabela 2-5. L1 Cache Miss (10^{8})

	L7-Filter				**SnortWeb**			
	0.35	***0.55***	***0.75***	***0.95***	***0.35***	***0.55***	***0.75***	***0.95***
D²FA /BE	1,28	0,93	0,96	0,70	2,79	2,18	1,54	0,90
D²FA /LE	1,71	1,44	1,55	1,05	4,79	3,71	2,69	1,49
D²FA /CS	3,17	2,34	2,34	1,68	7,50	5,99	4,23	2,40
RawDFA/ME	**0,92**	**0,68**	**0,68**	**0,50**	**2,48**	**1,96**	**1,38**	**0,79**
FastFA / BE	1,12	0,82	0,85	0,61	2,56	2,01	1,42	0,85
FastFA / LE	1,69	1,42	1,53	1,05	4,53	3,47	2,51	1,40
FastFA / CS	2,89	2,14	2,13	1,53	7,13	5,69	4,03	2,29

2.4 Lições Aprendidas

Os resultados experimentais mostram que o perfil de tráfego tem um impacto significante no desempenho das métricas analisadas. Como se pode ver através das métricas *máxima profundidade alcançada* e *transições disparadas* (Figura 2-4), as probabilidades de avanço e o conjunto de assinaturas mudam o comportamento dos DFAs completamente. A diferença no desempenho entre os cenários pode alcançar até 100% em alguns casos. Para o número de transições DFA disparadas (Figura 2-4), o RCDFA é o melhor entre os autômatos comprimidos, enquanto que o FastFA é o pior. Os

resultados mostraram que as transições padrão (Figura 2-6 e Figura 2-7) têm um impacto considerável no comportamento geral dos autômatos comprimidos. Como sugerido, estas diferenças também têm impacto nas métricas de baixo nível. Analisando os autômatos comprimidos em conjunto com os layouts de memória, pode-se concluir que o D^2FA com o LE utiliza menos instruções de CPU (Tabela 2-3), sendo também o mais rápido em ciclos de CPU (Tabela 2-4). Contudo, todos os autômatos comprimidos são piores do que o referencial RawDFA, em termos de velocidade de processamento.

Os resultados revelam dois pontos importantes: 1) para os modelos de DFA, transições padrão degradam substancialmente o desempenho dinâmico dos autômatos comprimidos; 2) para os *layouts* de memória, os *layouts* analisados para codificar DFAs comprimidos possuem um baixo desempenho, quando comparados com o referencial. De acordo com os resultados experimentais, a velocidade de processamento é influenciada tanto pela combinação do modelo de DFA como com o *layout* de memória. Assim, pesquisadores devem também considerar a velocidade de processamento quando projetarem DFAs comprimidos para alcançarem, pelo menos, desempenho comparável ao referencial apresentado aqui neste trabalho.

De maneira geral, este capítulo avaliou sistemas DPI olhando para os DFAs comprimidos e os layouts de memória utilizados para codificá-los em memória e os investigou de forma desacoplada. Analisando métricas dinâmicas e de baixo nível, foi demonstrado como o perfil de tráfego pode afetar o desempenho da combinação modelo-*layout*. Considerando os modelos de DFA, mostrou-se que alguns autômatos comprimidos têm um desempenho inferior inesperado devido às transições padrão. Ainda mais, para os *layouts* de memória, nenhum dos *layouts* para autômatos comprimidos obteve um desempenho melhor que o referencial. Isso implica que grandes esforços devem ser feitos de forma a melhorar a combinação modelo-*layout* a ser utilizado em sistemas DPI de alto desempenho.

3 Reordenando Assinaturas em Mecanismos de Inspeção de Pacotes Baseado em Prioridade Dinâmica

"A experiência nunca falha, apenas as nossas opiniões falham ao esperar da experiência aquilo que ela não é capaz de oferecer."

Leonardo da Vinci

Como visto e apresentado nos capítulos anteriores, um sistema DPI apresenta um alto custo computacional. Tendo em vista a melhoria da velocidade de análise, vários componentes da arquitetura de um DPI são constantemente estudados do ponto de vista do seu impacto no desempenho computacional do sistema, em especial a base de assinaturas. Neste capítulo, o impacto da ordem das assinaturas é atestado através de uma ordenação estática da lista de assinaturas. Adicionalmente, foi proposto um método de ordenação dinâmica com o intuito de reduzir o tempo de processamento da inspeção de pacotes. Os resultados demonstraram a influência da ordem das assinaturas no desempenho do DPI, bem como o ganho de desempenho, através da ordenação dinâmica, em torno de 60% do tempo de processamento. Por fim, o método proposto neste trabalho pode ser combinado com outras técnicas do estado da arte para alcançar um melhor desempenho do DPI. O restante deste capítulo é organizado da seguinte forma: A introdução ao tema é discutida na seção 3.1. Um novo método de ordenação é proposto na seção 3.2. Em seguida, na seção 3.3 a metodologia é descrita e os resultados são apresentados na seção 3.4. Por fim, as conclusões e trabalhos futuros são expostos.

3.1 Contextualização

Diversas áreas relacionadas a redes de computadores ratificam a importância da identificação de tráfego passante na rede. Como exemplo, é possível citar áreas como gerenciamento de redes [26] e detecção de anomalias [27]. De maneira geral, existem três métodos para classificação de tráfego apresentados na literatura [1]: classificação baseada em portas, classificação baseada em fluxos e classificação baseada em pacotes. Existem diversos trabalhos utilizando as técnicas apresentadas na

literatura[14][1][11][10][12][22], no entanto, é possível destacar a DPI – que é baseada em pacotes – como a técnica mais precisa do ponto de vista da classificação. Em outras palavras, esse método alcança as maiores taxas de pacotes corretamente classificados, inspecionando o conteúdo dos pacotes que passam na rede monitorada [1]. Nesse caso, o método inspeciona a carga útil do pacote à procura de um determinado padrão, denominado assinatura. Para que a classificação seja efetiva, é necessário que várias assinaturas sejam procuradas na carga útil do pacote até que algum padrão seja encontrado (casamento). Embora seja um método preciso, o DPI enfrenta alguns problemas de desempenho relacionados ao seu alto custo computacional[12]. Alguns estudos propõem formas de melhorar o desempenho do DPI, como a utilização de GPUs para incrementar sua capacidade de processamento [12] ou a inspeção apenas dos pacotes iniciais de um fluxo [20]. Melhorar o desempenho computacional de um sistema DPI permanece sendo uma questão em aberto na comunidade científica.

Diante da revisão de literatura realizada, é possível perceber que nenhuma pesquisa anterior analisa a ordem das assinaturas como um importante elemento que possa interferir no desempenho de um DPI. Os autores em [9] avaliaram o impacto de diferentes listas de assinaturas em um DPI, mas não foi analisada a ordem das assinaturas em uma mesma lista. Ordenar as assinaturas dinamicamente para melhorar o desempenho de um DPI não é uma tarefa trivial. Pesquisas em outras áreas de conhecimento propuseram abordagens similares para os problemas estudados, como algoritmos de cache [28] e algoritmos de *placement* de servidores [29]. Embora o cenário de aplicação seja distinto, a ideia é bastante similar. No caso da ordenação dinâmica das assinaturas, a ideia consiste basicamente na priorização das assinaturas mais relevantes para a classificação. Isto é, considerando que as assinaturas estão em uma lista que é percorrida sequencialmente, as assinaturas mais relevantes são posicionadas no início da lista. Essa técnica permite que os casamentos ocorram mais rápido e o sistema DPI economize tempo de processamento. Caso toda a lista precise ser percorrida, um maior tempo de processamento será gasto. Para uma única troca de posicionamento das assinaturas, a diferença entre os dois cenários citados será irrelevante. No entanto, em um cenário de redes multigigabits, esse evento ocorre milhões de vezes em um dia, tornando significativo o tempo de processamento poupado.

Este capítulo apresenta um estudo minucioso do impacto que a ordenação das assinaturas utilizadas por um sistema DPI causa ao seu desempenho. Evidências quantitativas desse impacto são apresentadas. Adicionalmente, um novo mecanismo foi proposto e desenvolvido para alterar dinamicamente a ordem das assinaturas na lista a fim de melhorar o desempenho do sistema. Em resumo, este capítulo apresenta duas contribuições: (i) demonstração do impacto da ordem das

assinaturas no desempenho do DPI e (ii) desenvolvimento de um novo método de ordenação dinâmica das assinaturas.

3.2 Prioridade no Conjunto de Assinaturas

Diante do objetivo de sistemas DPI de encontrar assinaturas de aplicações na carga útil de pacotes, existem várias formas de representar esses padrões. Uma maneira bastante utilizada é o formato de expressões regulares. Neste formato, podem ser observadas algumas listas de assinaturas que são disponibilizadas pela comunidade científica. Uma das mais populares é a lista de assinaturas do L7-Filter [16].

A importância da lista de assinaturas fica evidente quando se estuda minuciosamente o processo de classificação de um DPI. No cenário proposto neste trabalho, os pacotes chegam ao sistema em elevadas taxas, podendo chegar à ordem de Gigabits por segundo. O DPI armazena o seu conjunto de assinaturas previamente processadas (expressões regulares representadas como DFAs) na memória principal utilizando uma lista ligada. O processo de classificação de um pacote pode ser sintetizado da seguinte forma: uma vez que o pacote chega à máquina de medição onde a análise é executada, o DPI examina a carga útil do pacote buscando sequencialmente um padrão correspondente na lista de assinaturas. Caso esse padrão seja identificado, ocorre o casamento, significando que o pacote (e consequentemente o fluxo associado) foi classificado. Quando um casamento ocorre, o sistema não realiza mais comparações para o fluxo associado ao pacote, encerrando seu processamento para o fluxo em questão. Nesse cenário, quanto antes ocorrer o casamento do pacote com a assinatura, mais cedo a análise cessará e o tempo de processamento será reduzido. Dessa forma, o cenário ideal se configura quando a assinatura do pacote analisado está posicionada nas primeiras posições da lista.

Inicialmente, uma solução interessante seria realizar a ordenação prévia da lista de assinaturas antes mesmo que o sistema DPI inicie sua execução. No entanto, para que isso fosse possível, seria necessário conhecer previamente quais aplicações (ou mesmo classes de aplicações) são responsáveis pela maior parcela do tráfego da rede. O benefício dessa abordagem fica evidenciado quando as características da lista de assinaturas são observadas com mais detalhes. São 123 assinaturas presentes na lista do L7-Filter, que é a lista utilizada neste trabalho. Alguns agrupamentos possíveis podem ser observados nas assinaturas na Tabela 3-1.

Pode-se notar que cerca de 15% do total de assinaturas pertence ao grupo "Jogos". Dessa forma, se o sistema conhece a frequência com que as aplicações aparecem no tráfego, é possível posicionar o grupo "Jogos" no início da lista, no final ou mesmo retirá-lo do conjunto de assinaturas

de acordo com o interesse do administrador do sistema no resultado do DPI. Entretanto, esse tipo de informação dificilmente está disponível de maneira prévia para o sistema DPI. Além disso, a inspeção de pacotes costuma ocorrer em tempo real e uma eventual mudança do perfil do tráfego inspecionado não seria percebida. De acordo com [30], o perfil de tráfego da Internet muda à medida que o tempo passa. Essas mudanças ocorrem em diferentes escalas e dificilmente são previsíveis.

Sob a ótica da lista de assinaturas, a capacidade de adaptação às mudanças de tráfego é indispensável para sistemas DPI de alto desempenho. Por exemplo, a adaptação da lista de assinaturas pode melhorar o desempenho computacional de um DPI em um cenário de *flash crowd*. De maneira geral, um *flash crowd* pode ser definido como um fenômeno social traduzido na Internet que aumenta o volume de uma parcela específica do tráfego da rede rapidamente e é bastante difícil prevê-lo [31][32]. Se o novo comportamento da rede for percebido pelo DPI e as assinaturas relevantes para o evento forem posicionadas no topo da lista de assinaturas, um grande volume de processamento será economizado.

A solução aqui proposta é usar a informação sobre os casamentos positivos como um mecanismo de realimentação para o mecanismo DPI. Tal informação pode relatar quais aplicações são as mais comuns em uma janela de tempo específica. Com o conhecimento da popularidade das assinaturas, é possível usar algum tipo de priorização na lista. As assinaturas mais utilizadas podem ser posicionadas, durante a execução, nas melhores posições da lista. Esta ideia é ilustrada na Figura 3-1.

Tabela 3-1. Assinaturas e possíveis grupos para o conjunto de assinaturas do L7-Filter

Nome do grupo	Assinaturas
HTTP	http; http-rdsp; http-dap; http-freshdownload; http-itunes; httpaudio; httpcachehit; httpcachemiss; httpvideo; quicktime
P2P	100bao; applejuice; ares; bittorrent; directconnect; edonkey; fasttrack; gnucleuslan; gnutella; goboogy; hotline; imesh; kugoo; mute; napster; openft; poco; pplive; skypetoskype; soribada; soulseek; tesla; thecircle; xunlei
SNMP	snmp; snmp-mon; snmp-trap
Mensagem instantânea	aim; aimwebcontent; gtalk; irc; jabber; msnmessenger; qq; yahoo;
Jogos	armagetron; battlefield2; battlefield1942; battlefield2142; counterstrike-source; dayofdefeat-source; doom3; guildwars; halflife2-deathmatch; liveforspeed; mohaa; quake-halflife; quake1; runesofmagic; subspace; teamfortress2; worldofwarcraft; xboxlive

Fonte: autoria própria

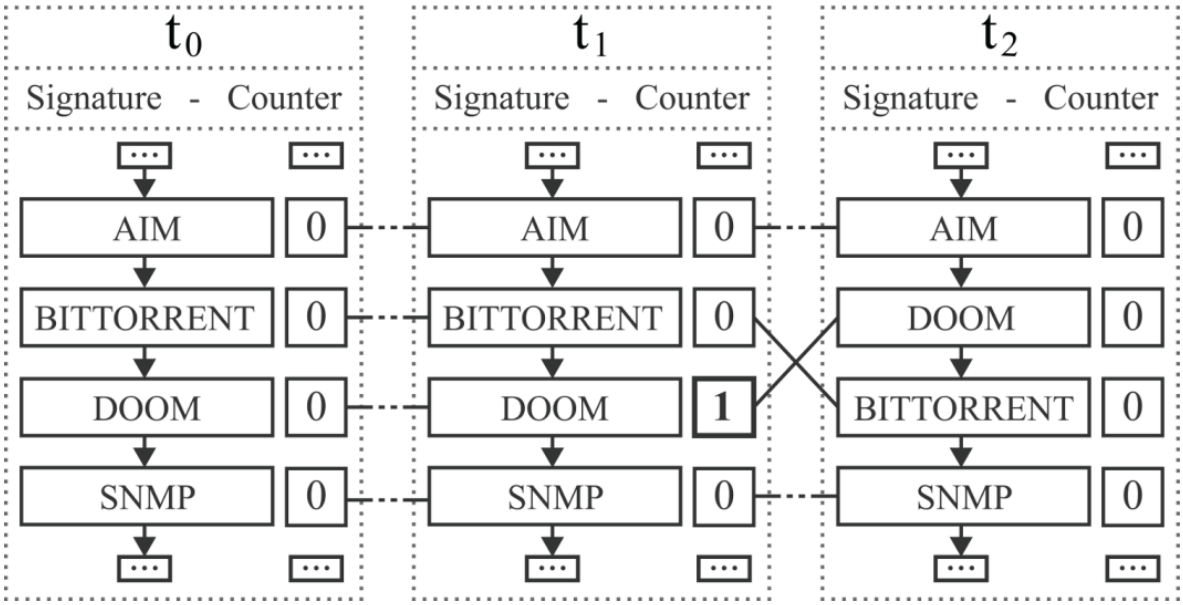

Figura 3-1. Troca de posições na lista de assinatura

Fonte: artigo de conferência [10]

A Figura 3-1 mostra a lista de assinaturas em três momentos distintos (ou fotografias) de um possível cenário. Cada fotografia contém a lista de assinaturas e um contador para cada assinatura. O contador representa a quantidade de casamentos positivos que uma assinatura tem durante um período de tempo específico. A fotografia t0 mostra a lista de assinaturas em algum momento da classificação de tráfego. Se uma assinatura apresentar um casamento positivo para algum pacote analisado, o contador de casamento positivo (e.g. contador da assinatura DOOM),

será incrementado em 1 e a nova fotografia será como t1. Neste momento, a assinatura do casamento positivo se movimentará para mais perto do início da lista, trocando de posição com a assinatura imediatamente acima dela, como mostrado na fotografia t2. Finalmente, o último contador incrementado é reiniciado e o mecanismo pode inspecionar um novo pacote.

O mecanismo de troca é simples, mas ele precisa ser o mais leve possível, de forma a superar a sobrecarga de processamento imposta pelas trocas. Mecanismos DPI precisam ser rápidos o suficiente para lidar com uma alta taxa de entrada de pacotes. Desta forma, os benefícios do mecanismo de prioridade precisam superar a sobrecarga, devido às mudanças na ordem das assinaturas. Mecanismos avançados de ordenamento podem gastar muito tempo mudando o posicionamento das assinaturas para cada *matching* positivo [31]. Tempo é um recurso escasso quando se trata de altas taxas de entrada. Em enlaces *multigigabit*, processamento DPI deve ser realizado em escala nanométrica.

3.2.1 Limiar de Prioridade

O processo de troca de assinaturas baseado no contador de casamento positivo pode ser estendido. Em vez de trocar as assinaturas de posição a cada vez que um casamento positivo ocorrer, pode ser adicionado um limiar de prioridade (i.e. 'P', no restante deste capítulo). Dessa forma, as assinaturas são trocadas apenas quando o contador alcançar o limiar previamente estabelecido. O mesmo cenário, visto na Figura 3-1, pode ser visualizado com P = 10, na Figura 3-2.

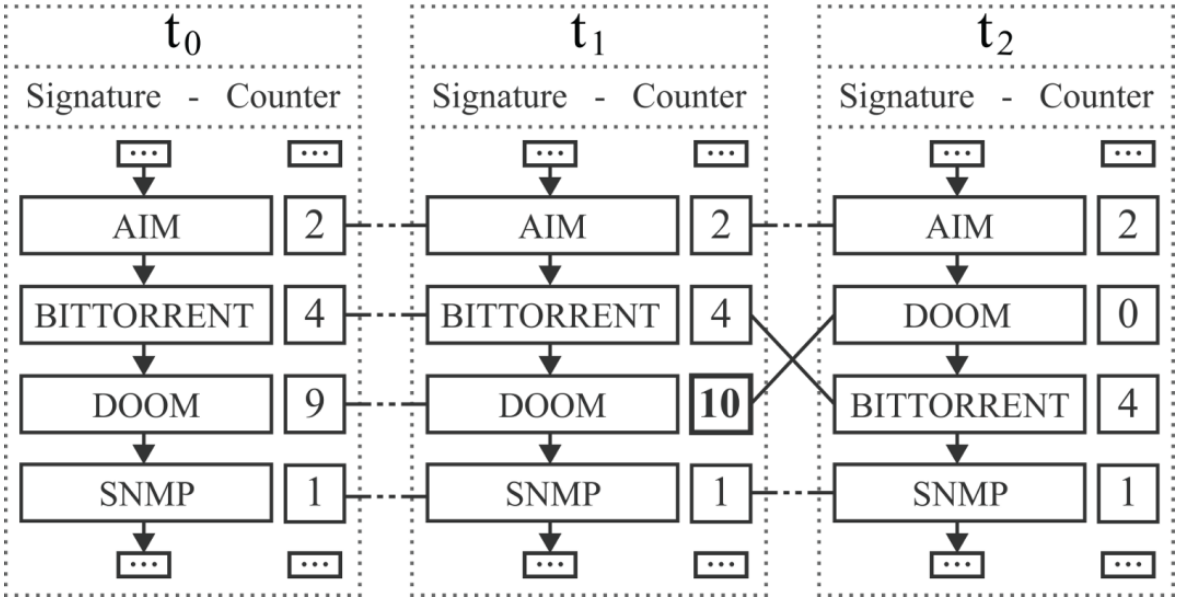

Figura 3-2. Troca de Assinaturas baseado em um limiar de prioridade P = 10

Fonte: artigo de conferência [10]

3.2.2 Função de Prioridade

Outra solução proposta neste capítulo está relacionada às estatísticas de casamento de assinaturas e aos custos definidos para cada assinatura. Essa informação associada de maneira específica fornece subsídios para que existam critérios de priorização na lista de assinaturas. Esse critério é definido de acordo com a Equação 1:

Equação 1. Função de prioridade da assinatura sig

$$\mathcal{F}_{\mathrm{sig}} = \frac{\mathrm{P_{sig}}}{\mathrm{C_{sig}}}$$

Nessa equação, a função de prioridade ($\mathrm{F_{sig}}$) de uma assinatura sig é definida pela razão entre sua popularidade ($\mathrm{P_{sig}}$) e o seu custo ($\mathrm{C_{sig}}$). A popularidade de uma assinatura é a parcela mutável de sua função de prioridade. Cada casamento proporcionado por uma assinatura específica incrementa uma unidade à sua popularidade. Já o custo é um valor fixo e particular de cada assinatura. A definição do custo de cada assinatura será detalhada na subseção a seguir. Em linhas gerais, a ideia da priorização das assinaturas está exposta na Figura 3-3, que mostra em dois momentos distintos um possível cenário da lista de assinaturas e como a função de prioridade realiza a reordenação da lista.

No primeiro momento (t1), cada assinatura tem um valor de função associado. O próximo pacote que chega ao sistema DPI pertence a uma aplicação (e.g. DOOM) e promove um casamento com a respectiva assinatura. Nesse cenário, o valor da função f2 é atualizado. Em seguida, o valor de f2 é comparado ao valor da função da assinatura acima na lista (f1). Dado que o resultado da função de prioridade da aplicação DOOM é maior que o da função do BitTorrent, ocorre a inversão na ordem das assinaturas envolvidas na comparação. A partir dessa troca, o cenário observado passa a ser o de t2. Essa operação ocorre porque, de acordo com a informação fornecida pela função de prioridade, espera-se que o tempo de processamento seja menor com a nova ordem das assinaturas.

Alguns outros elementos poderiam ser inseridos nessa função e, com maior eficácia, as assinaturas poderiam ser ordenadas. Um exemplo disso é a utilização de uma janela de tempo para controle da popularidade, corrigindo, assim, possíveis desvios na ordenação. Entretanto, é necessário que o mecanismo de ordenação cause a menor sobrecarga possível ao processamento do DPI. No nosso cenário, o sistema DPI opera em redes de alta velocidade e, portanto, deve ser suficientemente rápido para tratar os pacotes que chegam à interface de rede. Além disso, o benefício da reordenação das assinaturas deve superar a sobrecarga gerada pela função de prioridade. Algoritmos avançados de ordenação apresentam alta complexidade [33] e não se adequam à

realidade de um DPI. Como já foi dito anteriormente, em redes de alta velocidade, o DPI deve operar na escala de microssegundos.

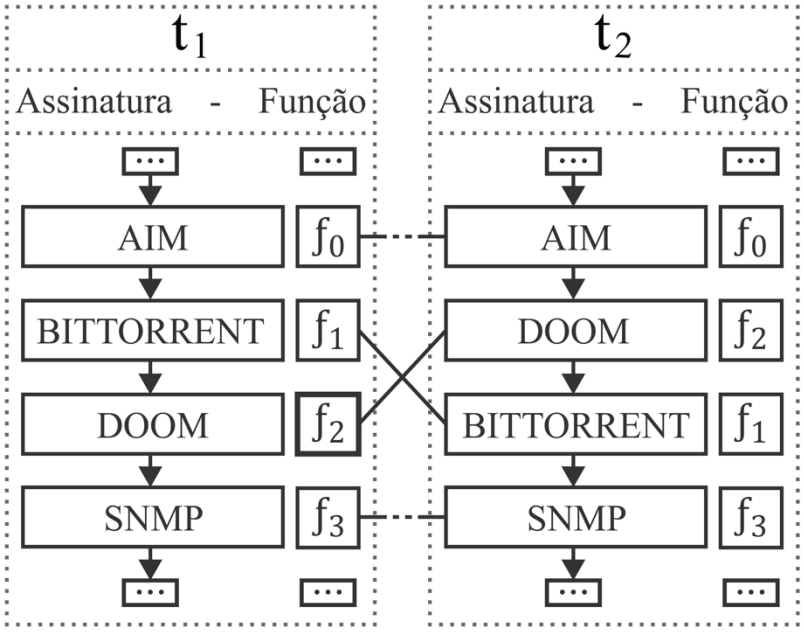

Figura 3-3. Funcionamento com função de prioridade

Fonte: artigo de conferência [10]

3.2.2.1 Custo da Assinatura

Como citado anteriormente, cada assinatura, utilizando o conceito de função de prioridade, tem um custo associado. O cálculo desse custo é previamente realizado (antes do início da operação do DPI) e o custo obtido se mantém fixo durante todo o funcionamento do sistema e para suas próximas execuções.

Uma vez que cada assinatura é uma expressão regular e é representada por um DFA, o custo foi estabelecido como sendo o menor caminho desde o estado inicial do autômato até um estado de aceitação. Em outras palavras, o custo é definido como o número de saltos desde o estado inicial até o estado de aceitação do DFA que representa a assinatura. Para a obtenção do custo foram gerados os autômatos que representam cada assinatura e um algoritmo de busca em profundidade foi implementado para identificar esse valor. Os custos de cada uma das assinaturas podem ser verificados na Tabela 6-1.

Já que o custo é um valor definido antes da execução do DPI, esse cálculo pode ser tão complexo quanto necessário, sem adicionar prejuízo ao tempo de processamento do DPI. Além

disso, outras métricas podem ser utilizadas para definir o custo. Por exemplo, o maior caminho até o estado de aceitação, o número de estados do autômato ou o seu número de transições.

É interessante notar que o mecanismo de prioridade pode ser implementado com alguma das otimizações descritas na seção 3.2, tais como compressão de DFAs e agrupamento de DFAs. Se DFAs agrupados podem ser armazenados em uma estrutura do tipo lista ligada, o mecanismo de prioridade pode ser usado para os DFAs agrupados, obtendo as vantagens destes dois grupos de otimizações, sem perda de generalidade para outras associações.

3.3 Metodologia de Avaliação

Neste trabalho, o impacto da ordenação das assinaturas sobre o processo de classificação de um DPI foi analisado. Nesse caso, o objetivo é reduzir o tempo necessário para classificar todos os fluxos de um *trace*. Portanto, a análise realizada neste trabalho avalia o tempo de processamento e não a precisão do DPI. Dessa forma, o número de fluxos que são analisados é mais relevante que o tipo de tráfego dos *traces* utilizados.

Para a avaliação da abordagem proposta poderiam ser utilizados *traces* reais ou sintéticos. No entanto, um tráfego real possui características específicas, apresentando variações que um tráfego sintético não consegue reproduzir. Nesse caso, foram utilizados três *traces* reais.

Dois dos *traces* (Trace1 e Trace2) foram coletados em um *backbone* do Brasil (não especificados neste trabalho por acordos de confidencialidade dos dados). Esses *traces* apresentam um grande volume de pacotes e fluxos. A terceira coleta de tráfego (Trace3) foi realizada através da ferramenta GT e está disponível publicamente [34]. O *trace* coletado pelo GT consiste na concatenação da coleta de dois dias consecutivos.

Já que o tipo de tráfego especificamente não interfere diretamente no desempenho da abordagem proposta, os experimentos realizados podem ser reproduzidos considerando as informações contidas na Tabela 3-2.

Tabela 3-2. Traces e Características

Trace	# de pacotes	# de fluxos	Pacotes inspecionados	Número de aplicações
Trace1	118308001	838975	3429920	31
Trace2	114168550	871055	3519427	33
Trace3	7512639	74447	778829	16

Para avaliar o impacto da ordem das assinaturas, duas abordagens foram utilizadas. Na primeira, são realizadas mudanças estáticas na ordem das assinaturas do L7-Filter antes da execução do DPI. Nesse caso, foram definidos cinco arranjos diferentes para a avaliação das mudanças estáticas: Ordenada (assinaturas em ordem alfabética), Inversa (ordem inversa à lista Ordenada), Aleatória 1, Aleatória 2 e Aleatória 3. O conjunto de assinaturas 'Ordenada' é considerado como o referencial da lista de assinaturas. A segunda abordagem trata da ordenação dinâmica, realizada através do limiar de prioridade e da função de prioridade. É interessante notar que a ordenação dinâmica foi realizada partindo das 5 diferentes organizações realizadas estaticamente. Essa avaliação é importante porque um ponto de partida (ordenação inicial) diferente poderia impactar no tempo de convergência da ordenação dinâmica.

Para analisar a efetividade da ordenação dinâmica, a avaliação utiliza como base de comparação o desempenho do DPI sobre a lista estática de assinaturas. Em outras palavras, cada uma das listas de assinaturas (Ordenada, Inversa, Aleatória 1, Aleatória 2, Aleatória 3) é analisada com e sem a utilização do limiar de prioridade / função de prioridade.

A ideia deste trabalho é avaliar e apresentar ganhos de desempenho computacional de um sistema DPI. Dessa forma, os valores absolutos do desempenho do DPI serão apresentados apenas para a lista Ordenada, que é a base de comparação. Nesse cenário, a métrica utilizada é o tempo total necessário para analisar todo o "caminho" percorrido pelo pacote dentro do sistema, denominada "tempo de processamento do pacote". Essa métrica considera também o tempo gasto para a ordenação das assinaturas. A fim de ter uma visão geral do sistema, o tempo de processamento de cada pacote é agregado em um nível menor de granularidade, no nível de fluxos. Essa métrica será tratada apenas como tempo de processamento e será o tempo total de processamento dos fluxos por cenário. A avaliação foi realizada de maneira *offline* para que sejam evitadas perdas de pacotes e, consequentemente, medições injustas entre diferentes cenários.

A Tabela 3-3 mostra um resumo dos fatores e métricas analisados neste capítulo.

Tabela 3-3. Fatores e níveis dos experimentos

Fatores	Níveis
Lista de Assinaturas	Ordenada, Inversa, Aleatória 1, Aleatória 2, Aleatória 3
Limiar de Prioridade	0 (sem prioridade); 1; 2 ; 10 ; 100; 1.000; 10.000; 100.000 ; 500.000; 1.000.000
Função de Prioridade	Sem função e com função

Todos os testes foram executados em uma máquina Linux (Ubuntu 12.04 - x86), kernel 3.11, Intel (R) Core (TM) i7-2600 3.4 GHz, 8 GB RAM. O DFA utilizado foi o RawDFA com o *layout* de memória matricial (ME).

3.4 Resultados e Discussão

Nesta seção, os resultados relacionados às mudanças estáticas e dinâmicas na ordem das assinaturas serão apresentados. Todas as comparações são mostradas como uma porcentagem da base de comparação. Na Tabela 3-4, os valores absolutos para a lista Ordenada são apresentados para que a comparação com outros trabalhos seja possível.

Tabela 3-4. Tempo de processamento absolutos para o limiar de prioridade

Mudanças Estáticas		**Mudanças Dinâmicas (Limiar)**	
Referencial	***Tempo (ns)***	***Referencial***	***Tempo (ns)***
Ordenada – Trace 1	7,509E+11	Ordenada - Trace 1 P = 1	3,040E+11
Ordenada – Trace 2	7,857E+11	Ordenada - Trace 2 P = 1	3,253E+11
Ordenada – Trace 3	1,812E+11	Ordenada - Trace 3 P = 1	1,793E+11

Com relação aos experimentos relacionados à função de prioridade, os valores de referência podem ser vistos na Tabela 3-5:

Tabela 3-5. Tempo de processamento absolutos para a função de prioridade

Referencial	**Sem função de prioridade (ns)**	**Com função de prioridade (ns)**
Ordenada – Trace 1	7,509E+11	4,001E+11
Ordenada – Trace 2	7,857E+11	5,292E+11
Ordenada – Trace 3	1,812E+11	3,053E+11

Nesse caso, é possível perceber que os dois primeiros *traces* apresentam um tempo de processamento menor quando a função de prioridade é utilizada. No entanto, para o Trace3, ocorre um aumento nesse tempo de processamento, que pode ser justificado pelo menor número de fluxos presente nesse tráfego coletado. Nesse caso, a sobrecarga gerada pela função de prioridade não é superada pelos benefícios da ordenação.

Na Tabela 3-6, são apresentados os percentuais de diferença do processamento com e sem função de prioridade, que expõem a relação entre os valores absolutos que foram apresentados na tabela anterior.

Tabela 3-6. Ganho percentual do tempo de processamento com o uso da função de prioridade

Trace utilizado	Tempo de processamento economizado (%)
Trace1	43
Trace2	33
Trace3	-68

É possível perceber que, para o Trace1 e para o Trace2, o tempo de processamento apresenta ganho de desempenho quando a função de prioridade é utilizada, sendo cerca de 43% no melhor caso (Trace1). Entretanto, existe uma queda de desempenho quando a função é aplicada ao cenário do Trace3. Como dito anteriormente, isso ocorre por conta do baixo número de fluxos presentes nesse trace. Nesse caso, menos interações ocorrem, sendo insuficiente para que a lista de assinaturas seja ordenada de maneira ótima. Não há fluxos suficientes para que a função de prioridade exerça seus benefícios, superando a sobrecarga de sua operação. No entanto, em um cenário real de redes de altas velocidades, a quantidade de fluxos será consideravelmente maior que o observado no Trace3. Dessa forma, os resultados a partir deste ponto serão limitados às análises envolvendo o Trace1 e o Trace2, já que, por conta do seu número de fluxos, permitem que a função de prioridade seja efetiva.

3.4.1 Ordenação Estática

A Figura 3-4 mostra o tempo proporcional gasto para inspecionar os *traces* pelo mecanismo DPI.

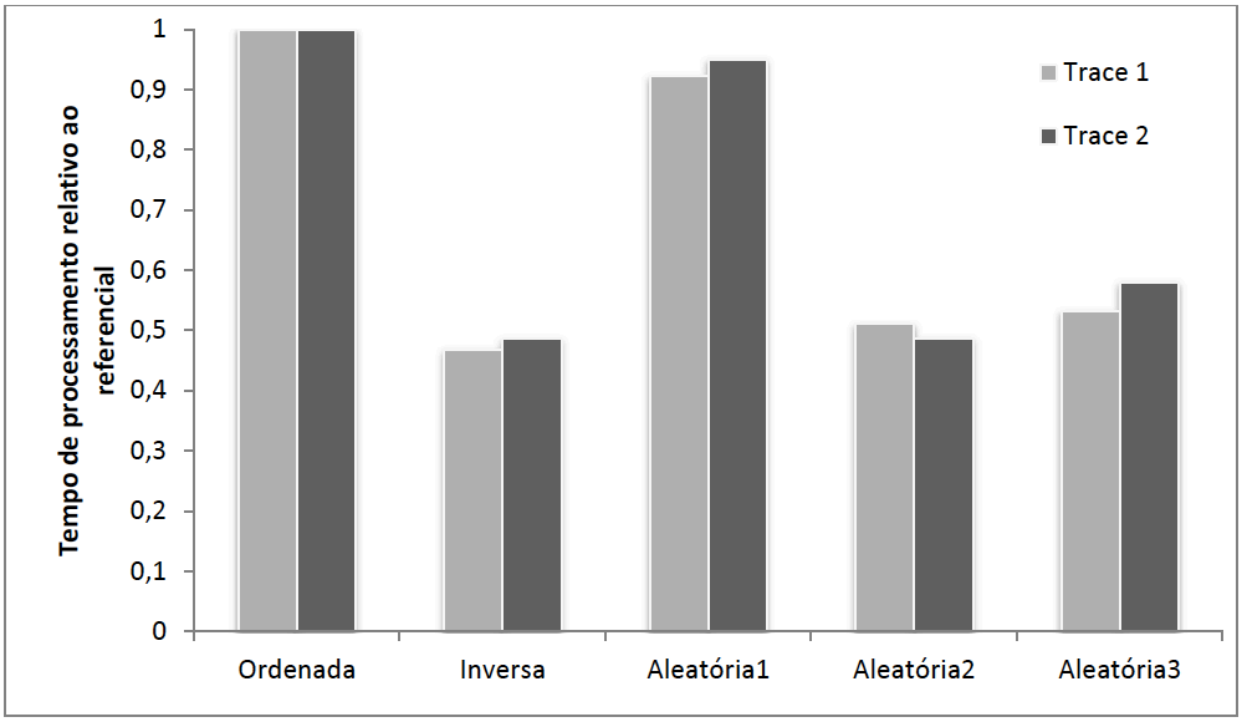

Figura 3-4. Comparações para as mudanças estáticas na lista de assinaturas

Como mencionado anteriormente, o valor de referência é a lista de assinaturas "Ordenada", apresentando, assim, o valor de 1. Em ambos os casos (*traces* 1 e 2), a lista de assinaturas "Ordenada" apresentou o pior resultado. A lista "Aleatória1" apresentou resultados similares ao referencial. A lista "Inversa" apresentou os melhores resultados, sendo 0,467 do referencial para o Trace1 e 0,485 do referencial para o Trace2. Para a lista "Aleatória1", o Trace1 apresentou 0,921 do referencial e o Trace2 apresentou 0,95 do referencial.

Estes fatos destacam pontos importantes: o primeiro é que a ordem das assinaturas dentro da lista tem um grande impacto no desempenho de um mecanismo DPI. Este fato é provado quando são analisados os resultados das listas aleatórias ("Aleatória1", "Aleatória2" e "Aleatória3"), as quais geraram resultados tão bons quando o melhor caso ("Aleatória2") e tão ruins quanto o pior caso ("Aleatória1"). O segundo ponto é que a aleatoriedade pode gerar listas melhores. Embora esteja fora do escopo deste trabalho, os resultados mostram que é possível encontrar alguma lista de assinaturas melhor do que a lista reversa ("Inversa") com um processo aleatório. Contudo, como uma lista dessas pode ter um número bastante elevado de assinaturas, é sugerido que seja usada alguma heurística no processo de geração da ordem, o que seria mais confiável do que a geração aleatória. Tal conhecimento prévio poderia ser a complexidade de cada assinatura e as suas relações com outras do mesmo grupo. Além disso, correlações entre *traces* poderiam ser investigadas de forma a manter assinaturas altamente correlacionadas próximas umas das outras.

3.4.2 Mudanças Dinâmicas

A partir deste ponto, serão analisadas as mudanças dinâmicas utilizando duas técnicas diferentes: limiar de prioridade e função de prioridade.

3.4.2.1 Limiar de Prioridade

Os seguintes resultados também são apresentados como uma proporção do referencial. Neste caso, o valor do referencial é a lista de assinaturas ordenada ("Ordenada"), sem as mudanças dinâmicas (P=0).

Figura 3-5 e Figura 3-6 mostram os resultados da mudança dinâmica na ordem de assinaturas durante a execução do DPI (P = 1), em comparação com a manutenção das assinaturas em suas posições iniciais (P = 0). Estes testes foram feitos com todas as listas, de forma a mostrar o impacto das mudanças dinâmicas em todos os cenários.

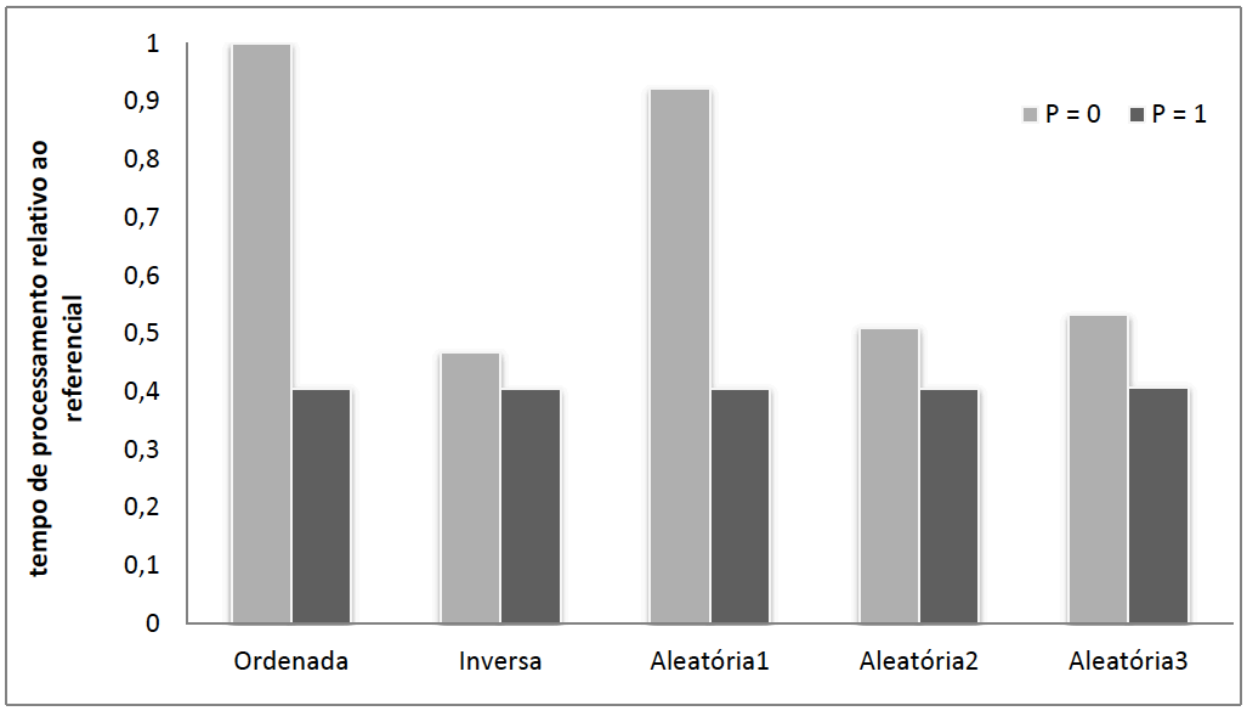

Figura 3-5. Trace1 com limiar de prioridade P = 1

Para os cenários sem mudanças na ordem das assinaturas, a proporção é a mesma da Figura 3-4. Os novos resultados estão relacionados com as mudanças dinâmicas na ordem de assinaturas. Neste caso, para todas as listas, os resultados são cerca de 60% menores do que o referencial (Ordenada = 0,414; Inversa = 0,414; Aleatória1 = 0,413; Aleatória2 = 0,413; Aleatória3 = 0,414). Os resultados mostram dois fatos importantes: o primeiro é que, dependendo do cenário, a prioridade dinâmica (P = 1) pode acelerar o processo de casamento no DPI em quase 60% (cenário "Ordenada"). O segundo ponto é que a prioridade dinâmica pode otimizar a lista de assinaturas para

uma configuração ótima (pelo menos, um ótimo local, e não um global). Isto pode ser confirmado analisando as relações, em todas as listas de assinaturas, entre a prioridade dinâmica desativada (P = 0) e a ativada (P = 1). Em todos os casos, a prioridade dinâmica reduziu o tempo de casamento.

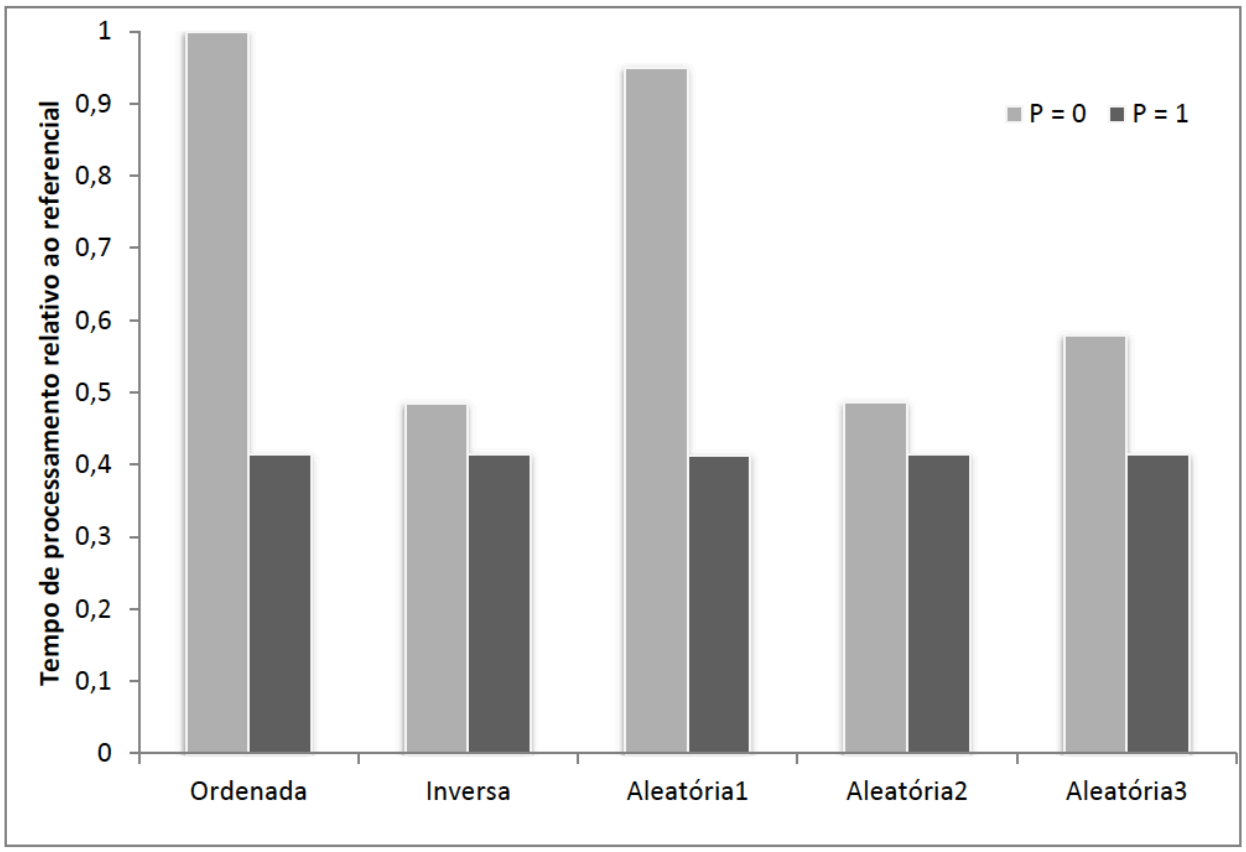

Figura 3-6. Trace2 com limiar de prioridade = 1

Considerando apenas os cenários com mudanças dinâmicas, foi introduzido o conceito de "limiar de prioridade". Como mencionado anteriormente, o limiar de prioridade apenas troca as assinaturas quando o contador de casamentos positivos alcança o valor do limiar. Figura 3-7 e Figura 3-8 apresentam os resultados para lista de assinaturas "Ordenada", para o Trace1 e o Trace2. O valor de referência é quando a prioridade dinâmica é desativada (P = 0, no gráfico).

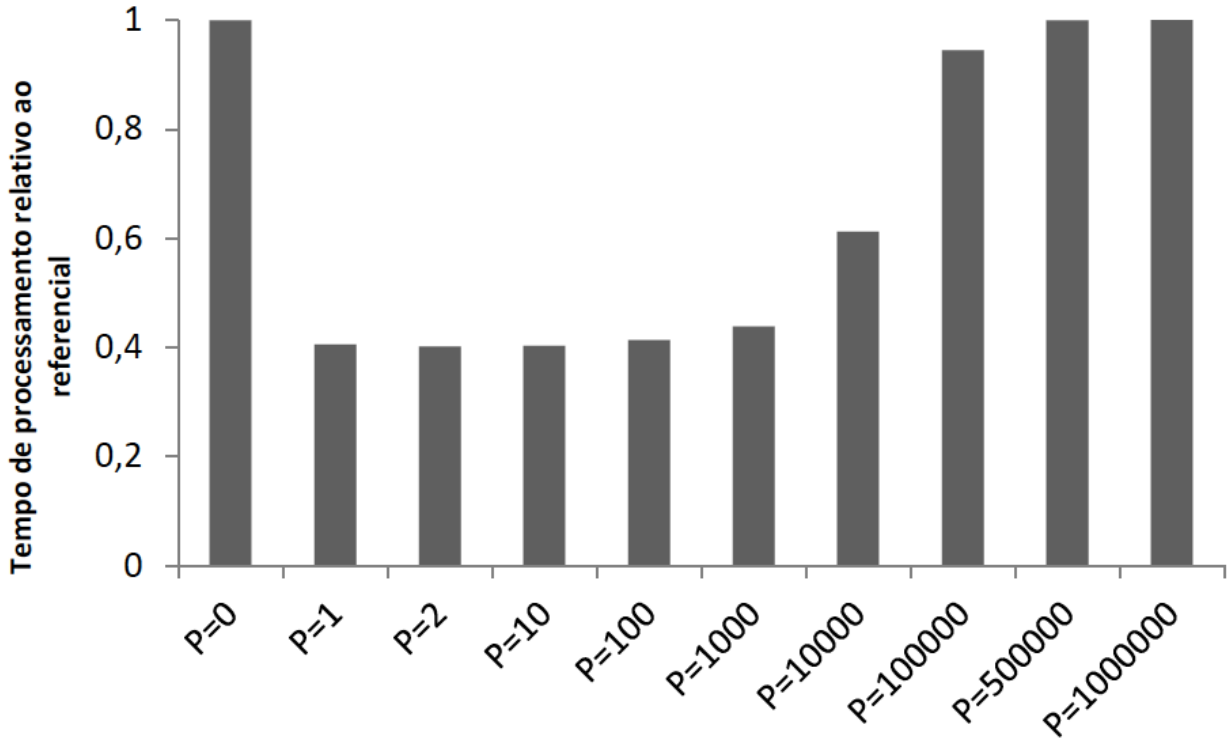

Figura 3-7. Trace1: Lista de assinaturas "Ordenada" com todos os limiares de prioridade

O resultado quando o limiar de prioridade é 1 não muda em relação aos resultados anteriores. Os outros limiares apresentados na Figura 3-7 foram analisados para entender o balanceamento entre trocar as assinaturas imediatamente após o casamento ou mantê-las em suas posições durante períodos maiores. Para os valores de P igual ou superior a 2, o tempo de casamento apresenta piores resultados do que com P = 1.

Tal piora no tempo de casamento em relação ao aumento do limiar de casamento tem uma possível explicação. O bom desempenho quando P = 1 ocorreria devido à sua configuração ser mais reativa do que as outras, i.e., a ordem das assinaturas converge mais rapidamente para a configuração ótima. De outra maneira, com P = 1, o método aqui apresentado realiza muitas trocas do que em outras configurações (P = 2, P = 10, ...). Portanto, pode-se concluir que a sobrecarga da troca de assinaturas é irrelevante, de forma que a configuração mais reativa (P = 1) apresenta o melhor tempo de processamento.

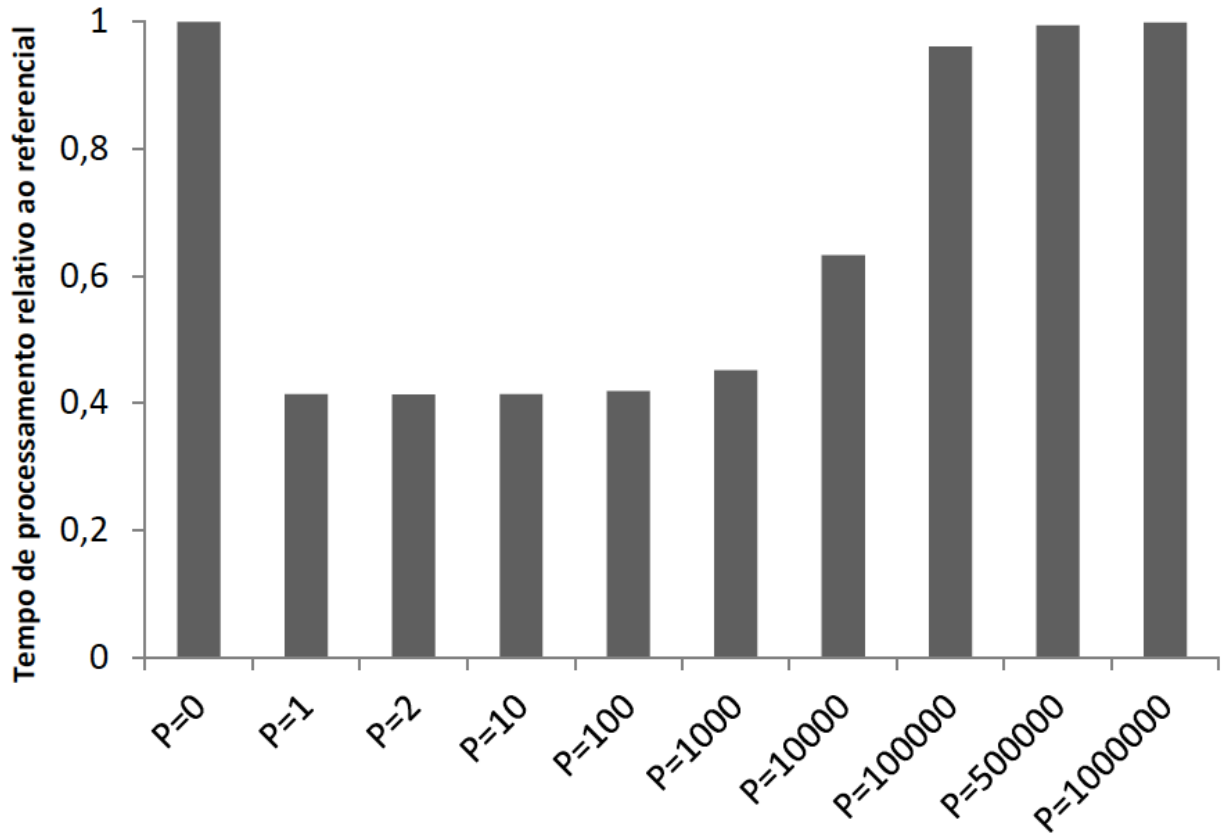

Figura 3-8. Trace2: Lista de assinaturas "Ordenada" com todos os limiares de prioridade

Outro ponto interessante a se destacar é quando P = 500.000. Como se pode ver, neste caso, o tempo de casamento é o mesmo que o do referencial (i.e. mudanças dinâmicas desabilitadas). Isto é explicado quando se compara estes resultados com os da Tabela 3-2. Como descrito, os *traces* contém cerca de 800.000 fluxos. Portanto, no caso em que P = 500.000, apenas uma assinatura poderia ter a sua prioridade aumentada. Além disso, a troca de apenas uma assinatura em toda a lista não melhora de forma significativa o tempo de casamento. Estas conclusões destacam que o limiar de prioridade se torna ineficiente para valores próximos do número de fluxos em um período de tempo. Considerando que muitos estudos são realizados off-line (i.e. com *traces* de pacotes e não online), a efetividade do limiar depende do tamanho do *trace* de pacotes, em termos de número de fluxos.

Este fato explica os resultados em experimentos preliminares quando a prioridade é analisada para o Trace3, como mostrada na Tabela 3-5. O tempo de processamento em todos os cenários é muito similar ao referencial ("Ordenada", com P = 0). Nestes resultados, é notado que a baixa quantidade de pacotes e fluxos torna inviável a convergência da lista de assinaturas usando a abordagem aqui apresentada. Na Tabela 3-7, são apresentados os valores absolutos em nanosegundos obtidos na avaliação do Trace3.

Tabela 3-7. Tempo de análise para o Trace3

Limiar	Tempo de análise (ns)
P=0	1,81E+11
P=1	1,79E+11
P=2	1,79E+11
P=10	1,80E+11
P=100	1,81E+11
P=1000	1,81E+11
P=10000	1,81E+11
P=50000	1,81E+11
P=100000	1,81E+11

Considerando a implementação desta técnica em um cenário real, o limiar de prioridade não apresenta resultados "bons" ou "ruins". Ele apenas representa a estabilidade da lista de assinaturas ao lidar com um tráfego de rede altamente dinâmico. Projetistas de mecanismos DPI podem ajustar esta estabilidade de acordo com requisitos específicos, e.g., baseados em teoria de controle.

3.4.2.2 Função de Prioridade

Para ratificar a relevância da ordenação dinâmica, os resultados expostos a seguir comparam os tempos de processamento com e sem o uso da função de prioridade. Para todos os casos, como dito anteriormente, o tempo de processamento base é o da lista Ordenada sem utilizar a função de prioridade. Todos os outros valores são relativos a essa base. Na Figura 3-9, são apresentados os tempos de processamento relativos obtidos quando o Trace1 foi inspecionado.

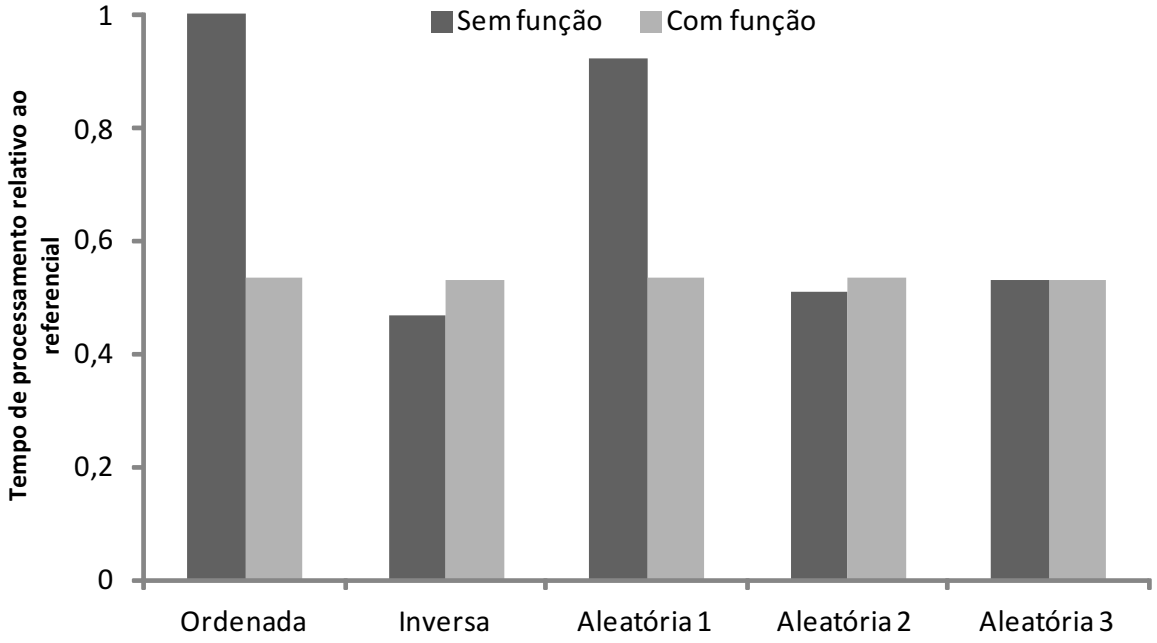

Figura 3-9. Tempos de processamento relativos para o Trace1

É possível observar que os resultados obtidos com o uso da função de prioridade superam amplamente os obtidos com as listas "Ordenada" e "Aleatória1" sem a função. Nesses casos, contata-se que, apesar da sobrecarga gerada pela função de prioridade, a ordenação das assinaturas na lista permite que o tempo de processamento seja reduzido. Entretanto, quando o comportamento das outras listas é observado, percebe-se que não há melhora no tempo total de processamento. Diante desse cenário, o tempo de processamento utilizando a função de prioridade foi analisado em duas partes: (i) o tempo de classificação propriamente dito e (ii) a sobrecarga gerada pela operação da função. A sobrecarga gerada pela função de prioridade e pelas trocas de posição das assinaturas representa em média aproximadamente 42,5% do tempo de processamento para o Trace1. Nesse caso, a reordenação proporcionada pela função de prioridade demonstra um desempenho bastante superior ao alcançado sem a função, como pode ser observado na Figura 3-10.

Observando a Figura 3-11 e Figura 3-12 , percebe-se que o cenário propiciado pelo Trace2 é bastante similar ao descrito acima para o Trace1. Na Figura 3-11, existe um ganho considerável de desempenho quando são utilizadas as listas "Ordenada" e "Aleatória 1". Nesse cenário, o tempo de processamento para as listas "Inversa", "Aleatória 2" e "Aleatória 3" apresenta uma perda significativa de desempenho quando a função é utilizada.

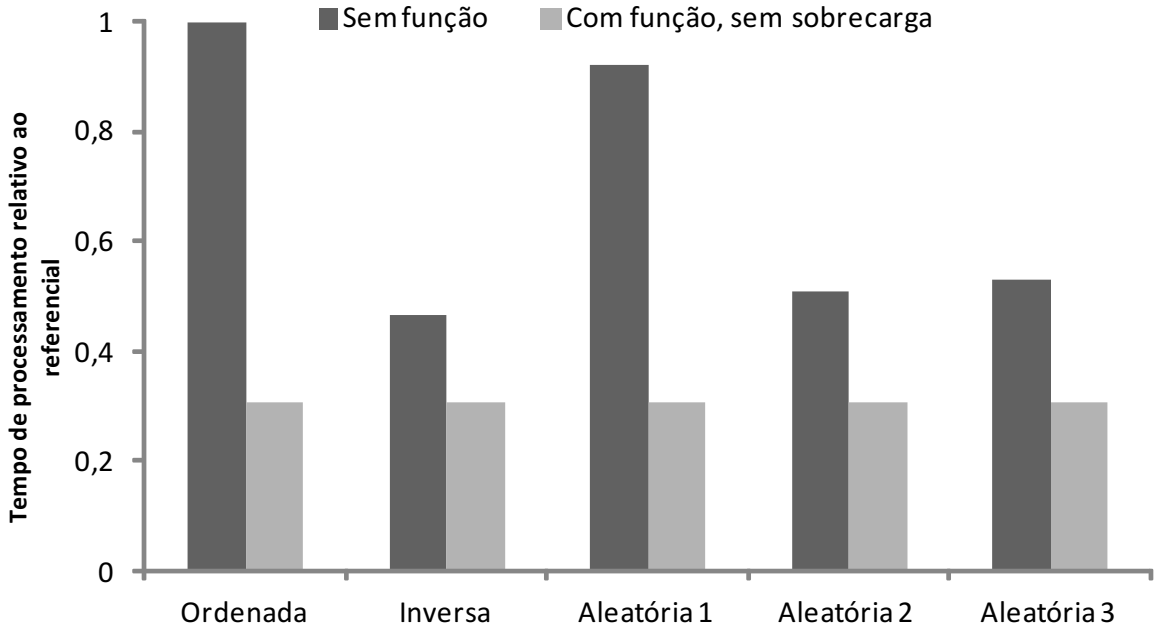

Figura 3-10. Tempos de processamento relativos para o Trace1, desconsiderando a sobrecarga

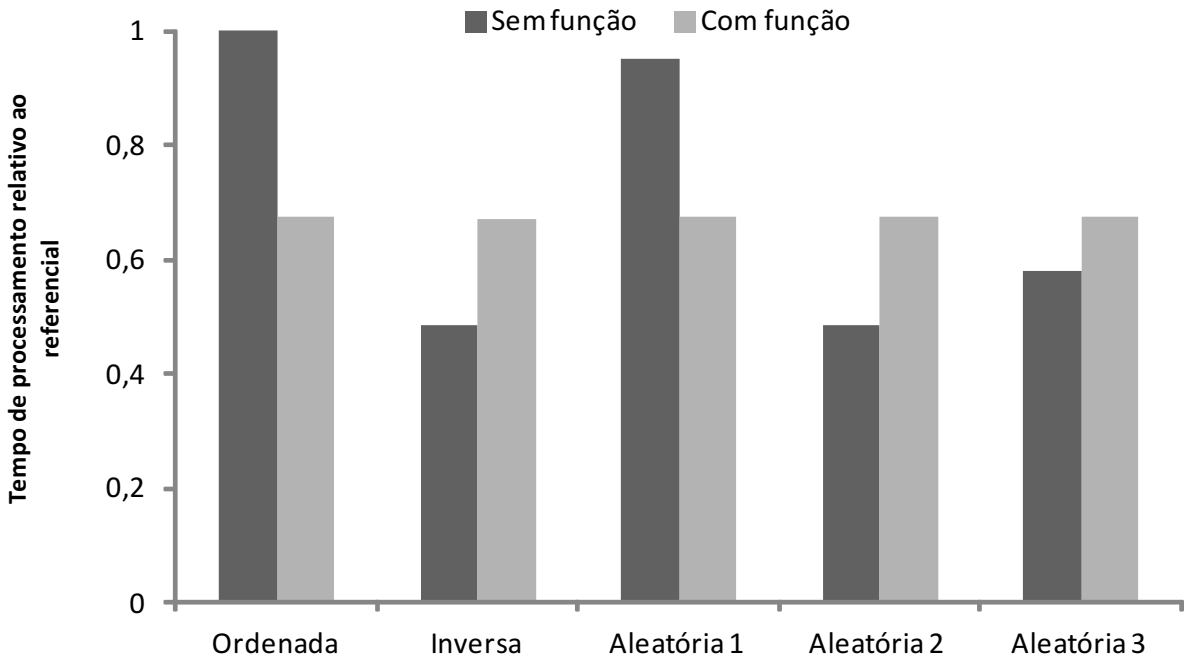

Figura 3-11. Tempos de processamento relativos para o Trace2

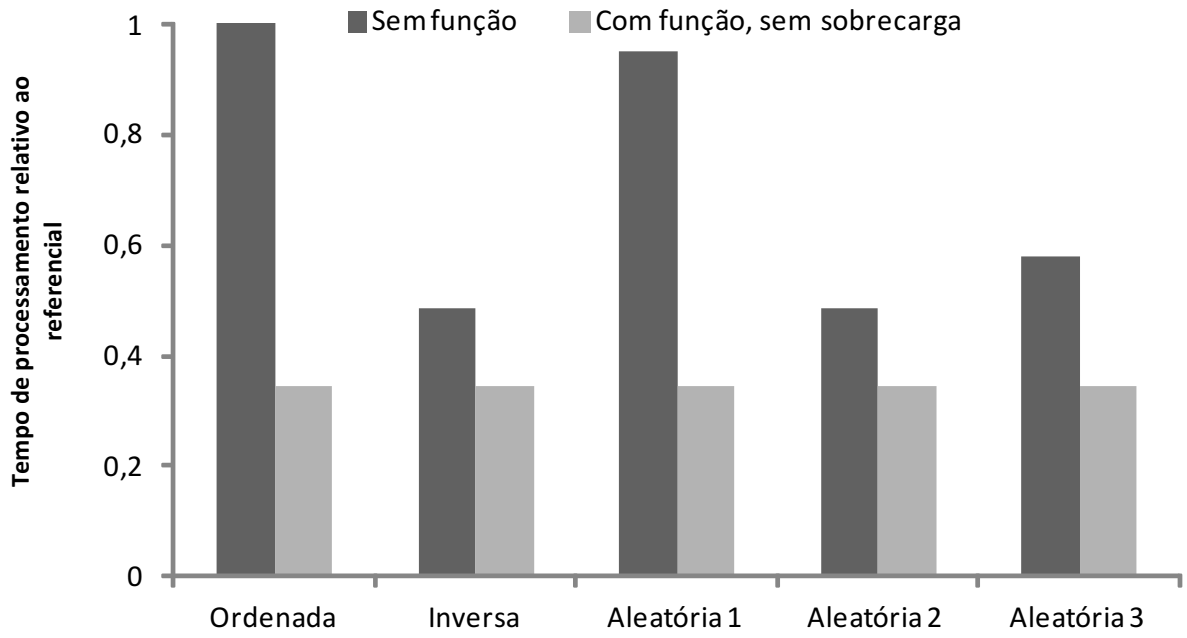

Figura 3-12. Tempos de processamento relativos para o Trace1, desconsiderando a sobrecarga

É interessante destacar que um trabalho voltado à otimização da função de ordenação ou da estrutura de dados pode permitir que a sobrecarga seja minimizada e, com isso, o tempo de processamento total seja reduzido. Sem a sobrecarga, o desempenho da ordenação proporcionada pela função é significativamente superior ao alcançado sem a função de prioridade. Essa sobrecarga pode ser reduzida, por exemplo, com a utilização de uma estrutura de dados otimizada, própria para executar trocas de posições entre as assinaturas da lista.

Por fim, o uso da função de prioridade apresenta uma característica bastante interessante e desejável para o cenário investigado neste trabalho. Independente da ordem inicial da lista de assinaturas, o tempo de processamento utilizando a função converge para o mesmo desempenho. Isso pode ser observado na Figura 3-13, em que a variação do tempo de processamento para cada *trace* é exposta.

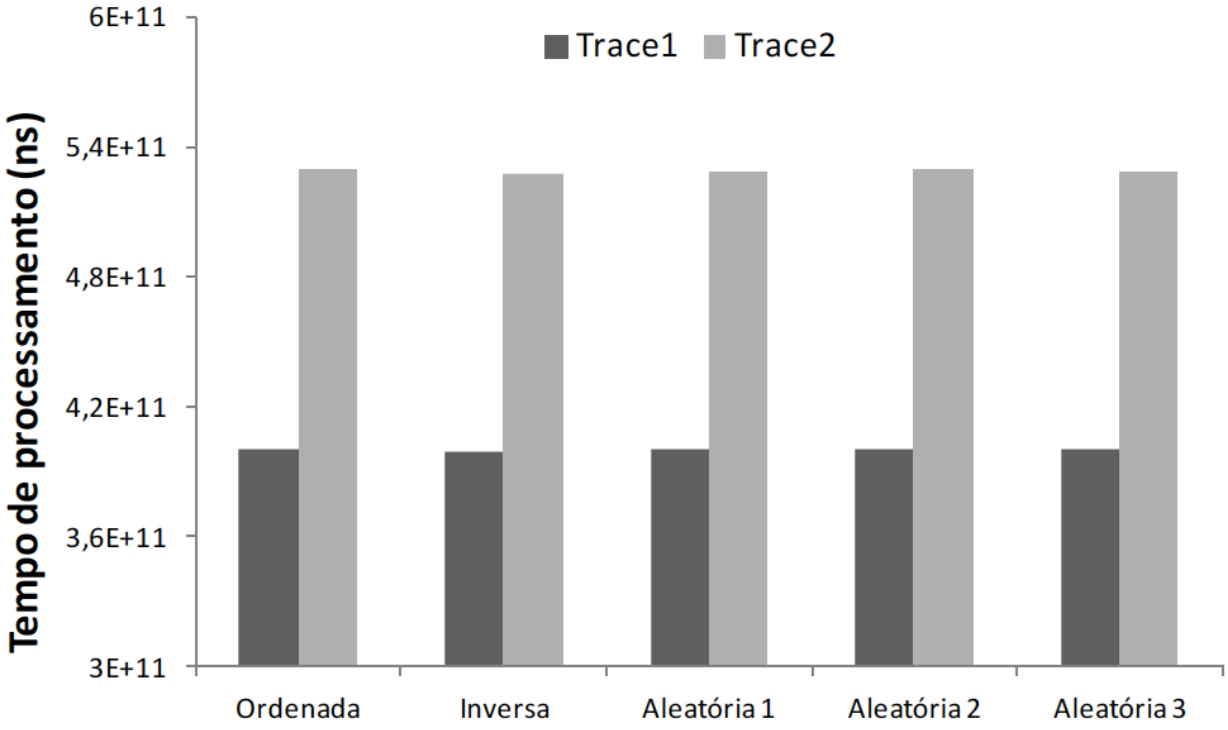

Figura 3-13. Os tempos de processamento utilizando a função de prioridade se mantém constantes

Essa característica é bastante interessante já que, mesmo que não apresente o melhor desempenho para qualquer classificação, o tempo de processamento será baixo e estável. Nesse caso, percebe-se que o desempenho se mantém constante para as diferentes listas de assinaturas utilizadas.

3.5 Lições Aprendidas

De forma geral, foi possível constatar, ao longo deste capítulo , como a lista de assinaturas afeta o desempenho geral de um mecanismo DPI. Foi demonstrado como uma simples inserção de aleatoriedade na ordem pode criar listas próximas tanto ao pior caso como ao melhor caso. Em outras palavras, os resultados são fortemente influenciados pela ordem das assinaturas.

Além disso, foi introduzido o conceito de prioridade na lista de assinaturas. Adicionalmente, foi provado que a troca das assinaturas pode ser feita de forma dinâmica, durante a inspeção dos pacotes. Para isto, foi mostrado um mecanismo de troca rápida, o qual insere sobrecarga pequena ao processo. Também foi introduzido o conceito de estabilidade no processo de troca e testado o seu impacto em traces pequenos e grandes. Esses fatores foram variados de forma a entender melhor o comportamento do método proposto.

Foram discutidas possíveis otimizações para listas estáticas de assinaturas. Alguns delas são relacionadas a agrupamentos de características comuns, tais como P2P, Instant Messenger e Jogos,

ou a remoção destes grupos da lista (quando não são mais interessantes para a análise do DPI). Além disso, a análise sobre as mudanças dinâmicas mostrou que muitas otimizações ainda podem ser realizadas, em relação aos limiares, custos de cada assinatura para a troca e correlações entre assinaturas.

Ainda mais, outras técnicas podem ser providas, considerando que a sobrecarga da troca de posições seja superada pelos benefícios obtidos.

Finalmente, é muito importante destacar a possibilidade de combinar a solução proposta com outros métodos do estado-da-arte. Não existem, até onde se sabe, obstáculos para utilizar soluções combinadas de forma a melhorar o desempenho do DPI.

Acredita-se que as contribuições propostas neste capítulo sejam as primeiras em associar os impactos de mudar a ordem de assinaturas para melhoria do desempenho do mecanismo DPI. Nenhum outro trabalho destrinchou a lista de assinaturas como um fator de desempenho.

4 Conclusão

"Ideias são matéria-prima para a construção de ideias"

Jason Zebehazy

Ao longo de todo este trabalho, foi possível perceber a importância de se conhecer profundamente os componentes de um sistema DPI, a fim de se evitar perdas significativas de desempenho.

No capítulo 2, foi revelado como DFAs e layouts de memória afetam a velocidade de processamento do mecanismo DPI. Primeiramente, alguns DFAs otimizados para compressão de memória (D^2FA, FastFA, RCDFA, RawDFA) foram analisados em termos de seus comportamentos dinâmicos. Depois, foram inseridos alguns dos principais layouts de memória (ME, LE, BE e CS) e as combinações entre estes e os DFAs foram investigadas, considerando o desempenho em uma arquitetura PC real.

Já no capítulo 3, o foco dado na ordenação da lista de assinaturas trouxe uma melhor interpretação da importância deste componente no projeto do DPI. Foi apresentado o conceito de prioridade no conjunto de assinaturas, além de destacar como essa característica poderia beneficiar num incremento de desempenho. Ainda mais, foram apresentadas duas propostas de ordenação dinâmica, objetivando flexibilizar a ordenação para cada perfil de tráfego. Em muitos dos casos apresentados, os benefícios da técnica se mostraram eficazes, diminuindo consideravelmente o tempo de processamento.

De forma geral, os dois capítulos apresentaram conclusões significativas para a classificação de tráfego utilizando DPI, do ponto de vista do desempenho, de forma que as principais são listadas a seguir:

1. O perfil de tráfego analisado causa um forte impacto no número de transições disparadas e estados visitados;
2. A complexidade do conjunto de assinaturas, analisando o número de estados analisados, só mostra ser relevante com uma probabilidade de casamento baixa (e.g. 0,35). Para probabilidades altas (e.g. 0,95), a importância da complexidade é diminuída;

3. Sem considerar as implementações em memória, o RCDFA é o DFA comprimido que melhor se comportou, disparando o menor número de transições, comparado ao FastFA e ao D^2FA;
4. Sem considerar as implementações em memória, o FastFA foi o pior modelo analisado, disparando cerca de 70% mais transições que o referencial (RawDFA), em alguns cenários;
5. Transições padrão são eficientes para consumo de memória, mas diminuem a velocidade de processamento consideravelmente;
6. A análise de métricas de baixo nível (e.g. instruções executadas e ciclos de CPU) é crucial para se medir o real desempenho de um DFA em uma arquitetura real, através de um layout de memória;
7. Combinações errôneas entre DFAs e layouts de memória podem ser altamente prejudiciais ao desempenho do sistema DPI, a ponto de anularem ganhos de desempenho previstos em análises teóricas;
8. Nenhum dos DFAs comprimidos, junto com os layouts de memória analisados, foi capaz de superar o referencial (DFA padrão e layout de memória matricial). Isso mostra o quão aberto esse campo ainda se encontra, no ponto de vista de velocidade de processamento;
9. A simples mudança da ordem das assinaturas contribuiu com um ganho de desempenho de mais de 50%, em alguns cenários, reforçando, assim, a importância deste componente no desempenho geral do DPI;
10. Um sistema de ordenação dinâmica para a lista de assinaturas é capaz de reconfigurar a ordem para um estado de otimização mais adequado ao perfil momentâneo do tráfego analisado;
11. Desconsiderando a sobrecarga, uma função de prioridade (como a apresentada no capítulo 3) é capaz de capturar as particularidades das assinaturas (e.g. custo), ajustando de forma mais precisa a ordem das assinaturas;
12. Funções de prioridade mais complexas devem considerar sempre o custo da sobrecarga sobre o desempenho geral. Cálculos complexos podem diminuir (ou até mesmo anular) os ganhos obtidos com a reordenação das assinaturas.

4.1 Contribuições

Ao longo de todo o trabalho, várias contribuições na evolução dos mecanismos DPI foram apresentadas, que são sumarizadas abaixo:

1) Avaliação da dinâmica dos DFAs comprimidos;
2) Avaliação da combinação entre DFAs comprimidos e layouts de memória;
3) Proposta e avaliação de mudança na ordenação das assinaturas;
4) Proposta e avaliação de uma ordenação dinâmica na ordem das assinaturas utilizando um limiar de prioridade;
5) Proposta e avaliação de uma ordenação dinâmica na ordem das assinaturas utilizando uma função de prioridade.

Além dessas contribuições, é possível destacar ainda as publicações resultantes deste trabalho, listadas abaixo:

1) *A look under the hood: Revealing performance issues in the DPI engine*, em: IEEE International Conference on Communications, (IEEE ICC 2013), Budapeste, Hungria. Julho 2013. Qualis A2.
2) *On the Performance of DPI Signature Matching with Dynamic Priority*, em: IEEE Symposium on Computers and Communications (IEEE ISCC 2014), Madeira, Portugal. Junho 2014. Qualis A2;
3) *Reordenando Assinaturas em Mecanismos de Inspeção de Pacotes Baseado em Prioridade Dinâmica*, em: WPerformance - XIII Workshop em Desempenho de Sistemas Computacionais e de Comunicação, 2014, Brasília. Anais do CSBC 2014. Qualis B5

Além dos três trabalhos citados, durante o período do mestrado, foram publicados outros dois artigos, listados a seguir:

1) *Benchmarking of compressed DFAs for traffic identification: Decoupling data structures from models*. Em: IEEE Global Communications Conference (IEEE GLOBECOM 2012), Anaheim, Califórnia, 2012. Qualis A1
2) *GPU-Oriented Stream Data Mining Traffic Classification*. Em: IEEE Symposium on Computers and Communications (IEEE ISCC 2014), Madeira, Portugal. Junho 2014. Qualis A2;

4.2 Trabalhos Futuros

Os capítulos 2 e 3 trouxeram, cada um, contribuições dentro de cada tema tratado. Complementando tais informações, ainda podemos destacar os seguintes trabalhos futuros:

1) Realizar avaliações do RCDFA com um *layout* de memória que o represente;
2) Desenvolver modelos analíticos que capturem os principais comportamentos do sistema DPI e suas respectivas configurações, a fim de mensurar estimativas de desempenho dos modelos em velocidades multi-gigabit (e.g. 40Gbps);
3) Analisar outros métodos de troca de posições das assinaturas, assim como outros critérios para aumentar a prioridade e outras estruturas de dados;
4) Realizar uma avaliação detalhada sobre a sobrecarga do algoritmo de ordenação das assinaturas, considerando a complexidade do algoritmo e diferentes formas de ordenar as assinaturas;
5) Implementar as técnicas dos capítulos 2 e 3 de forma conjunta, e avaliar os ganhos/perdas em um ambiente simulado. Como a ordenação das assinaturas foi analisada com apenas um par modelo de DFA/layout de memória, será possível verificar o impacto da ordem da lista utilizando combinações menos eficientes, como FastFA+BE;
6) Avaliar a proposta acima descrita considerando funções de prioridade mais eficientes, do ponto de vista do desempenho geral do sistema DPI. Tais funções podem conter, ainda, elementos de custo relacionados com a estrutura dos DFAs comprimidos, junto com o impacto das combinações entre modelos e layouts.

4.3 Dificuldades encontradas

No decorrer do trabalho, vários obstáculos foram superados para que esta etapa fosse concluída. Tais dificuldades apareceram em diferentes pontos, desde a compreensão dos modelos teóricos de DFAs, passando pela implementação dos modelos, chegando até ao mecanismo de prioridade testado.

Primeiramente, foi preciso entender profundamente a dinâmica de um mecanismo DPI, observando os principais gargalos de desempenho quando utilizado em altas velocidades. Tal estudo, por si só, já revela diferentes formas de avaliação dos mecanismos, além uma ampla gama de aplicações, desde antivírus a classificadores de anomalias. Definir o escopo neste campo se mostrou difícil pela grande similaridade do método de análise, mas ampla dissemelhança no objetivo final.

Essas sutis diferenças possuem um grande impacto quando se avalia o mecanismo DPI completo, sendo crucial a correta definição do escopo deste trabalho.

Após isso, foi preciso fazer uma identificação dos principais componentes do sistema DPI, observando o comportamento de tais partes quando submetidos a cargas excessivas de trabalho. Os módulos de estruturas de dados para os DFAs comprimidos e os layouts de memória são complexos. A implementação deles exigiu conhecimento avançado de C/C++ para o desenvolvimento/testes, sendo um dos principais desafios em todo o trabalho.

Outra dificuldade encontrada foi a forma de avaliação das métricas de baixo nível. Várias ferramentas de análise de código (*code profiling*) foram testadas, não conseguindo atender aos objetivos desejados. Muitas delas não davam suporte para algumas das métricas que eram pretendidas. Outras delas não faziam a análise em alguns trechos específicos de código. Algumas não forneciam maneiras práticas de análise dos dados coletados. Em resumo, coletar e analisar foi tão difícil quanto implementar e testar.

Por fim, a implementação da ordenação na lista de assinaturas apresentou uma necessidade de escolhas corretas em estruturas de dados voltadas para tal fim, para que a implementação não prejudicasse a ideia em si. Além disso, foi complexo o desenvolvimento de uma função de prioridade que não provocasse grande sobrecarga, devido às restrições de execução do DPI em altas velocidades.

Referências

[1] CALLADO, A. et al. A Survey on Internet Traffic Identification. **IEEE Communications Surveys Tutorials**, , v. 11, n. 3, p. 37-52, August 2009.

[2] ANTONELLO, R. et al. **Deterministic Finite Automaton for scalable traffic identification: The power of compressing by range**. Network Operations and Management Symposium. Maui, HI, USA: IEEE. 2012. p. 155-162.

[3] BECCHI, M.; CROWLEY, P. A-DFA: A Time- and Space-Efficient DFA Compression Algorithm for Fast Regular Expression Evaluation. **ACM Transactions on Architecture and Code Optimization (TACO)**, New York, NY, USA, v. 10, n. 4, p. 1-26, april 2013. ISSN: 1544-3566.

[4] BECCHI, M.; CROWLEY, P. **A Hybrid Finite Automaton for Practical Deep Packet Inspection**. Proceedings of the 2007 ACM CoNEXT Conference. New York, New York: [s.n.]. 2007. p. 1-12.

[5] FICARA, D. et al. Differential Encoding of DFAs for Fast Regular Expression Matching. **IEEE/ACM Transactions on Networking (TON)**, Piscataway, NJ, USA, v. 19, n. 3, p. 683-694, June 2011. ISSN 1063-6692

[6] YU, F. et al. **Fast and Memory-efficient Regular Expression Matching for Deep Packet Inspection**. ACM/IEEE Symposium on Architecture for Networking and Communications Systems. New York, NY, USA: ACM. 2006. p. 10.

[7] BECCHI, M.; CROWLEY, P. **An Improved Algorithm to Accelerate Regular Expression Evaluation**. Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems. Orlando, Florida, USA: ACM. 2007. p. 145-154.

[8] KUMAR, S. et al. Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection. **ACM SIGCOMM Computer Communication Review**, New York, NY, USA, v. 36, n. 4, p. 339-350, October 2006. ISBN:1-59593-308-5.

[9] ANTONELLO, R. et al. **Characterizing signature sets for testing DPI systems**. IEEE GLOBECOM Workshops (GC Wkshps). Houston, TX, USA: IEEE. 2011. p. 678-683.

[10] MELO, W. et al. **On the performance of DPI signature matching with dynamic priority**. IEEE Symposium on Computers and Communications (ISCC). Funchal, Madeira, Portugal, Portugal: IEEE. 2014. p. 1-6

[11] LOPES, P. et al. **GPU-oriented stream data mining traffic classification**. IEEE Symposium on Computers and Communications (ISCC). Funchal, Madeira, Portugal, Portugal: IEEE. 2014. p. 1-7

[12] SANTOS, A. F. et al. **Multi-gigabit traffic identification on GPU**. Proceedings of the first edition workshop on High performance and programmable networking (HPPN'13). New York, New York, USA: ACM. 2013. p. 39-44

[13] JIANG, W.; PRASANNA, V. K. **A FPGA-based Parallel Architecture for Scalable High-Speed Packet Classification**. Proceedings of the 2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP'09). Washington, DC, USA: ACM. 2009. p. 24-31.

[14] ANTONELLO, R. et al. Deep packet inspection tools and techniques in commodity platforms: Challenges and trends. **Journal of Network and Computer Applications**, London, UK, v. 35, n. 6, p. 1863-1878, november 2012. ISSN: 1084-8045.

[15] HYPERTEXT Markup Language. Disponivel em: <http://www.w3schools.com/tags>. Acesso em: 20 julho 2014.

[16] Application Layer Packet Classfier for Linux. Disponivel em: <http://l7-filter.sourceforge.net>. Acesso em: 15 julho 2014.

[17] HOPCROFT, J.; ULLMAN, J. **Introduction to Automata Theory, Languages, and Computation**. 1ª. ed. USA: Addison-Wesley, v. único, 1979. ISBN 0-201-02988-X..

[18] SIPSER, M. **Introdução à Teoria da Computação**. Tradução de Ruy Queiroz. 1ª. ed. : CTP, Cengage, v. único, 2005.

[19] MELO, W. et al. **Benchmarking of compressed DFAs for traffic identification:** Decoupling data structures from models. IEEE Global Communications Conference (GLOBECOM). Anaheim, CA, USA: IEEE. 2012. p. 1763-1769.

[20] FERNANDES, S. et al. **Slimming down Deep packet inspection systems**. Proceedings of the 28th IEEE international conference on Computer Communications Workshops (INFOCOM'09). Rio de Janeiro, Brazil: IEEE Press. 2009. p. 61-66.

[21] BECCHI, M.; FRANKLIN, M.; CROWLEY, P. **A workload for evaluating deep packet inspection architectures**. IEEE International Symposium on Workload Characterization. Seattle, WA, USA: IEEE. 2008. p. 79-89.

[22] Snort. Disponivel em: <https://www.snort.org>. Acesso em: 15 julho 2014..

[23] FICARA, D. et al. An improved DFA for fast regular expression matching. **SIGCOMM Computer Communication Review**, New York, NY, USA, v. 38, n. 5, p. 29-40, October 2008. ISSN: 0146-4833

[24] SANTOS, A. et al. **High-Performance Traffic Workload Architecture for Testing DPI Systems**. IEEE Global Telecommunications Conference - GLOBECOM 2011. Kathmandu, Nepal: IEEE Press. 2011. p. 1-5.

[25] Performance Application Programming Interface. Disponivel em: <http://icl.cs.utk.edu/papi/>. Acesso em: 15 julho 2014..

[26] SONG, T.; ZHOU, Z. File-aware P2P traffic classification: An aid to network management. **Peer-to-Peer Networking and Applications**, v. 6, n. 3, p. 325-339, september 2013. ISSN ISSN: 1936-6450.

[27] CHANDOLA, V.; BANERJEE, A.; KUMAR, V. Anomaly detection: A survey. **ACM Computing Surveys (CSUR)**, New York, NY, USA, v. 41, n. 3, p. 15, july 2009. ISSN: 0360-0300.

[28] GAO, Y.; ZHANG, Y.; ZHOU, Y. **A Cache Management Strategy for Transparent Computing Storage System (ISCTCS'12)**. Beijing, China: Yuan, Yuyu and Wu, Xu and Lu, Yueming. 2012. p. 651-658.

[29] HUANG, X.; PENG, Y. **A Novel Replica Placement Strategy for Data Center Network**. International Conference on Information Technology and Management Science(ICITMS 2012) Proceedings. Berlin, Heidelberg: Xu, Bing. 2012. p. 599-609

[30] CHO, K. et al. **The impact and implications of the growth in residential user-to-user traffic**. Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications. Pisa, Italy: ACM. 2006. p. 207-218.

[31] HIREMAGALORE, S. et al. Improving network response times using social information. **Social Network Analysis and Mining**, , v. 3, n. 2, p. 209–220, june 2013. ISSN: 1869-5469.

[32] ZHANIKEEV, M.; TANAKA, Y. A graphical method for detection of Flash Crowds in traffic. **Telecommunication Systems**, , v. 57, n. 1, p. 91-105, september 2014. ISSN: 1572-9451.

[33] CORMEN, T. et al. **Introduction to Algorithms**. 2ª edição. ed. : McGraw-Hill Higher Education, v. único, 2001. ISBN 0070131511

[34] GRINGOLI, F. et al. GT: Picking Up the Truth from the Ground for Internet Traffic. **SIGCOMM Comput. Commun. Rev**, New York, NY, USA, v. 39, n. 5, p. 12-18, october 2009. ISSN ISSN: 0146-4833

[35] ANTONELLO, Rafael Thyago. Optimizing finite automata for DPI engines. 2012. 83 f. Tese (Doutorado em Ciência da Computação) – Centro de Informática. Universidade Federal de Pernambuco, Pernambuco

Anexo A – Custo das Assinaturas do L7-Filter

Tabela 6-1. Custos das assinaturas do L7-Filter

Assinatura	Custo da Assinatura
100bao	4
aimwebcontent	15
aim	3
applejuice	8
ares	3
armagetron	6
audiogalaxy	9
battlefield1942	9
battlefield2142	3
bgp	20
biff	5
bittorrent	46
chikka	17
cimd	6
ciscovpn	4
citrix	5
counterstrike_source	25
cvs	27
dayofdefeat_source	20
dazhihui	10
dhcp	7
directconnect	8
dns	9
doom3	11
edonkey	2
fasttrack	10
finger	2
freenet	3
ftp	6
gkrellm	14
gnucleuslan	42
gnutella	4
goboogy	12

gopher	8
gtalk	29
guildwars	5
h323	6
halflife2_deathmatch	19
hddtemp	16
hotline	30
httpaudio	31
httpcachehit	24
httpcachemiss	25
http_dap	18
http_freshdownload	27
http	24
http_itunes	26
http_rtsp	30
httpvideo	31
ident	7
imap	7
imesh	6
ipp	6
irc	9
jabber	43
kugoo	2
live365	23
liveforspeed	7
lpd	3
mohaa	14
msn_filetransfer	27
msnmessenger	19
mute	25
napster	10
nbns	7
ncp	6
netbios	42
nntp	7
ntp	1
openft	16
pcanywhere	2
poco	10
pop3	5
pplive	8
pressplay	20
qq	5
quake1	9
quake_halflife	11
quicktime	53
radmin	4
rdp	18

replaytv_ivs	24
rlogin	7
rtp	11
rtsp	15
runesofmagic	21
shoutcast	19
sip	20
skypetoskype	16
smb	5
smtp	38
snmp_mon	14
snmp_trap	16
snmp	15
socks	7
soribada	4
soulseek	2
ssdp	28
ssh	7
ssl	5
stun	16
subspace	16
subversion	17
teamfortress2	23
teamspeak	12
telnet	8
tesla	18
tftp	7
thecircle	20
tonghuashun	9
tor	14
tsp	3
uucp	6
validcertssl	27
ventrilo	7
vnc	12
whois	3
worldofwarcraft	3
x11	25
xboxlive	11
xunlei	8
yahoo	14
zmaap	6

Anexo B – Assinaturas do L7-Filter

Tabela 7-1. Assinaturas do L7-Filter

Aplicação	Padrão
100bao	^\x01\x01\x05\x0a
aimwebcontent	user-agent:aim/
aim	^(*[\x01\x02].*\x03\x0b|*\x01.?.?.?.?\x01)|flapon|toc_signon.*0x
applejuice	^ajprot\x0d\x0a
ares	^\x03[\]Z].?.?\x05
armagetron	YCLC_E|CYEL
audiogalaxy	^(\x45\x5f\xd0\xd5|\x45\x5f.*0.60(6|8)W)
battlefield1942	^\x01\x11\x10|\xf8\x02\x10\x40\x06
battlefield2142	^(\x11\x20\x01\x90\x50\x64\x10|\xfe\xfd.?.?.?\x18|[\x01\\].?battlefield2)
bgp	^\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff..?\x01[\x03\x04]
biff	^[a-z][a-z0-9]+@[1-9][0-9]+
bittorrent	^(\x13bittorrent protocol|azver\x01|get /scrape\?info_hash=get /announce\?info_hash=|get /client/bitcomet/|GET /data\?fid=)|d1:ad2:id20:|\x08'7P\)[RP]
chikka	^CTPv1\.[123] Kamusta.*\x0d\x0a
cimd	\x02[0-4][0-9]:[0-9]+.*\x03
ciscovpn	^\x01\xf4\x01\xf4
citrix	\x32\x26\x85\x92\x58
counterstrike_source	^\xff\xff\xff\xff.*cstrikeCounter-Strike
cvs	^BEGIN (AUTH|VERIFICATION|GSSAPI) REQUEST\x0a
dayofdefeat_source	^\xff\xff\xff\xff.*dodDay of Defeat
dazhihui	^(longaccoun|qsver2auth|\x35[57]\x30|\+\x10*)
dhcp	^[\x01\x02][\x01-]\x06.*c\x82sc
directconnect	^(\$mynick |\$lock |\$key)
dns	^.?.?.?.?[\x01\x02].?.?.?.?.?.?[\x01-?][a-z0-9][\x01-?a-z]*[\x02-\x06][a-z][a-z][fglmoprstuvz]?[aeop]?(um)?[\x01-\x10\x1c][\x01\x03\x04\xFF]
doom3	^\xff\xffchallenge
edonkey	^[\xc5\xd4\xe3-\xe5].?.?.?.?([\x01\x02\x05\x14\x15\x16\x18\x19\x1a\x1b\x1c\x20\x21\x32\x33\x34\x35\x36\x38\x40\x41\x42\x43\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58[\x60\x81\x82\x90\x91\x93\x96\x97\x98\x99\x9a\x9b\x9c\x9e\xa0\xa1\xa2\xa3\xa4]|\x59.....?[-~]|\x96....)
fasttrack	^get (/.download/[-~]*|/.supernode[-~]|/.status[-~]|/.network[-~]*|/.files|/.hash=[0-9a-f]*/[-~]*) http/1.1|user-agent: kazaa|x-kazaa(-username|-network|-ip|-supernodeip|-xferid|-xferuid|tag)|give [0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]?[0-9]?[0-9]?

finger	^[a-z][a-z0-9\-_]+|login: [\x09-\x0d -~]* name: [\x09-\x0d -~]* Directory:
freenet	^\x01[\x08\x09][\x03\x04]
ftp	^220[\x09-\x0d -~]*ftp
gkrellm	^gkrellm [10].[0-9].[0-9]\x0a
gnucleuslan	gnuclear connect/[\x09-\x0d -~]*user-agent: gnucleus [\x09-\x0d -~]*lan:
gnutella	^(gnd[\x01\x02]?.?.?\x01|gnutella connect/[012]\.[0-9]\x0d\x0a|get /uri-res/n2r\?urn:sha1:|get /.*user-agent: (gtk-gnutella|bearshare|mactella|gnucleus|gnotella|limewire|imesh)|get /.*content-type: application/x-gnutella-packets|giv [0-9]*:[0-9a-f]*/|queue [0-9a-f]* [1-9][0-9]?[0-9]?\.[1-9][0-9]?[0-9]?\.[1-9][0-9]?[0-9]?\.[1-9][0-9]?[0-9]?:[1-9][0-9]?[0-9]?[0-9]?|gnutella.*content-type: application/x-gnutella|.....?lime)
goboogy	<peerplat>|get /getfilebyhash\.cgi\?|get /queue_register\.cgi\?|get /getupdowninfo\.cgi\?
gopher	^[\x09-\x0d]*[1-9,+tgi][\x09-\x0d -~]*\x09[\x09-\x0d -~]*\x09[a-z0-9.]*\.[a-z][a-z].?.?\x09[1-9]
gtalk	^<stream:stream to="gmail\.com"
guildwars	^[\x04\x05]\x0c.i\x01
h323	^\x03..?\x08...?.?.?.?.?.?.?.?.?.?.?.?.?.?\x05
halflife2_deathmatch	^\xff\xff\xff\xff.*hl2mpDeathmatch
hddtemp	^|/dev/[a-z][a-z][a-z]|[0-9a-z]*|[0-9][0-9]|[cfk]|
hotline	^.....TRTPHOTL\x01\x02
httpaudio	http/(0\.9|1\.0|1\.1)[\x09-\x0d][1-5][0-9][0-9][\x09-\x0d -~]*(content-type: audio)
httpcachehit	http/(0\.9|1\.0|1\.1)[\x09-\x0d][1-5][0-9][0-9][\x09-\x0d -~]*(x-cache: hit)
httpcachemiss	http/(0\.9|1\.0|1\.1)[\x09-\x0d][1-5][0-9][0-9][\x09-\x0d -~]*(x-cache: miss)
http_dap	User-Agent: DA [678]\.[0-9]
http_freshdownload	User-Agent: FreshDownload/[456](\.[0-9][0-9]?)?
http	http/(0\.9|1\.0|1\.1) [1-5][0-9][0-9] [\x09-\x0d -~]*(connection:|content-type:|content-length:|date:)|post [\x09-\x0d -~]* http/[01]\.[019]
http_itunes	http/(0\.9|1\.0|1\.1).*(user-agent: itunes)
http_rtsp	^(get[\x09-\x0d -~]* Accept: application/x-rtsp-tunnelled|http/(0\.9|1\.0|1\.1) [1-5][0-9][0-9] [\x09-\x0d -~]*a=control:rtsp://)
httpvideo	http/(0\.9|1\.0|1\.1)[\x09-\x0d][1-5][0-9][0-9][\x09-\x0d -~]*(content-type: video)
ident	^[1-9][0-9]?[0-9]?[0-9]?[0-9]?[0-9]?[\x09-\x0d]*,[\x09-\x0d]*[1-9][0-9]?[0-9]?[0-9]?[0-9]?[0-9]?(\x0d\x0a|[\x0d\x0a])?
imap	^(* ok|a[0-9]+ noop)
imesh	^(post[\x09-\x0d -~]*<PasswordHash>.....</PasswordHash><ClientVer>|\x34\x80?\x0d?\xfc\xff\x04|get[\x09-\x0d -~]*Host: imsh\.download-prod\.musicnet\.com|\x02[\x01\x02]\x83.*\x02[\x01\x02]\x83)
ipp	ipp://
irc	^(nick[\x09-\x0d -~]*user[\x09-\x0d -~]*:|user[\x09-\x0d -~]*:[\x02-\x0d -~]*nick[\x09-\x0d -~]*\x0d\x0a)
jabber	<stream:stream[\x09-\x0d][-~]*[\x09-\x0d]xmlns=['"]jabber
kugoo	^(\x64.....\x70....\x50\x37|\x65.+)
live365	membername.*session.*player
liveforspeed	^..\x05\x58\x0a\x1d\x03
lpd	^(\x01[!-~]+|\x02[!-~]+\x0a.[\x01\x02\x03][\x01-\x0a -~]*|[\x03\x04][!-~]+[\x09-\x0d]+[a-z][\x09-\x0d -~]*|\x05[!-~]+[\x09-\x0d]+([a-z][!-~]*[\x09-\x0d]+[1-9][0-9]?[0-9]?|root[\x09-\x0d]+[!-~]+).*)\x0a

mohaa	^\xff\xff\xff\xffgetstatus\x0a
msn_filetransfer	^(ver [-~]*msnftp\x0d\x0aver msnftp\x0d\x0ausr|method msnmsgr:)
msnmessenger	ver [0-9]+ msnp[1-9][0-9]? [\x09-\x0d -~]*cvr0\x0d\x0a|usr 1 [!-~]+ [0-9.]+\x0d\x0a|ans 1 [!-~]+ [0-9.]+\x0d\x0a
mute	^(Public|AES)Key: [0-9a-f]*\x0aEnd(Public|AES)Key\x0a
napster	^(.[\x02\x06][!-~]+ [!-~]+ [0-9][0-9]?[0-9]?[0-9]?[0-9]? "[\x09-\x0d -~]+" ([0-9]|10)|1(send|get)[!-~]+ "[\x09-\x0d -~]+")
nbns	\x01\x10\x01|\)\x10\x01\x01|0\x10\x01
ncp	^(dmdt.*\x01.*(""|\x11\x11|uu)|tncp.*33)
netbios	\x81.?.?.[A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P][A-P]
nntp	^(20[01][\x09-\x0d -~]*AUTHINFO USER|20[01][\x09-\x0d -~]*news)
ntp	^([\x13\x1b\x23\xd3\xdb\xe3]|[\x14\x1c].......?.?.?.?.?.?.?.?[\xc6-\xff])
openft	x-openftalias: [-)(0-9a-z ~.]
pcanywhere	^(nq|st)
poco	^\x80\x94\x0a\x01....\x1f\x9e
pop3	^(\+ok |-err)
pplive	\x01...\xd3.+\x0c.
pressplay	user-agent: nsplayer
qq	^.?.?\x02.+\x03
quake1	^\x80\x0c\x01quake\x03
quake_halflife	^\xff\xff\xff\xffget(info|challenge)
quicktime	user-agent: quicktime \(qtver=[0-9].[0-9].[0-9];os=[\x09-\x0d -~]+\)\x0d\x0a
radmin	^\x01\x01(\x08\x08|\x1b\x1b)
rdp	rdpdr.*cliprdr.*rdpsnd
replaytv_ivs	^(get /ivs-IVSGetFileChunk|http/(0\.9|1\.0|1\.1) [1-5][0-9][0-9] [\x09-\x0d -~]*\x23\x23\x23\x23\x23REPLAY_CHUNK_START\x23\x23\x23\x23\x23)
rlogin	^[a-z][a-z0-9][a-z0-9]+/[1-9][0-9]?[0-9]?[0-9]?00
rtp	^\x80[\x01-"`-\x7f\x80-\xa2\xe0-\xff]?.........*\x80
rtsp	rtsp/1.0 200 ok
runesofmagic	^\x10\x03.........\x0a\x02.....\x0e
shoutcast	^get /.*icy-metadata:1|icy [1-5][0-9][0-9] [\x09-\x0d -~]*(content-type:audio|icy-)
sip	^(invite|register|cancel|message|subscribe|notify) sip[\x09-\x0d -~]*sip/[0-2]\.[0-9]
skypetoskype	^..\x02.....
smb	\xffsmb[\x72\x25]
smtp	userspace flags=REG_NOSUB REG_EXTENDED
snmp_mon	^\x02\x01\x04.+[\xa0-\xa3]\x02[\x01-\x04].?.?.?.?\x02\x01.?\x02\x01.?\x30
snmp_trap	^\x02\x01\x04.+\xa4\x06.+\x40\x04.?.?.?.?\x02\x01.?\x02\x01.?\x43
snmp	^\x02\x01\x04.+([\xa0-\xa3]\x02[\x01-\x04].?.?.?.?\x02\x01.?\x02\x01.?\x30|\xa4\x06.+\x40\x04.?.?.?.?\x02\x01.?\x02\x01.?\x43)
socks	\x05[\x01-\x08]*\x05[\x01-\x08]?.*\x05[\x01-\x03][\x01\x03].*\x05[\x01-\x08]?[\x01\x03]
soribada	^GETMP3\x0d\x0aFilename|\x01.?.?.?(\x51\x3a\+|\x51\x32\x3a)|\x10[\x14-\x16]\x10[\x15-\x17].?.?.?.?
soulseek	^(\x05..?|.\x01.[-~]+\x01F..?.?.?.?.?.?.?)
ssdp	^notify[\x09-\x0d]*[\x09-\x0d]http/1\.1[\x09-\x0d -~]*ssdp:(alive|byebye)|m-search[\x09-\x0d]*[\x09-\x0d]http/1\.1[\x09-\x0d -

	~]*ssdp:discover
ssh	^ssh-[5]\.[0-9]
ssl	^(.?.?\x16\x03.*\x16\x03|.?.?\x01\x03\x01?.*\x0b)
stun	^[\x01\x02].....
subspace	^\x01....\x11\x10.....\x01
subversion	^\(success \(1 2 \(
teamfortress2	^\xff\xff\xff\xff.....*tfTeam Fortress
teamspeak	^\xf4\xbe\x03.*teamspeak
telnet	^\xff[\xfb-\xfe].\xff[\xfb-\xfe].\xff[\xfb-\xfe]
tesla	\x03\x9a\x89\x22\x31\x31\x31\.\x30\x30\x20\x42\x65\x74\x61\x20|\xe2\x3c\x69\x1e\x1c\xe9
tftp	^(\x01|\x02)[-~]*(netascii|octet|mail)
thecircle	^t\x03ni.?[\x01-\x06]?t[\x01-\x05]s[\x0a\x0b](glob|who are you|query data)
tonghuashun	^(GET /docookie\.php\?uname=|\xfd\xfd\xfd\xfd\x30\x30\x30\x30\x30)
tor	TOR1.*<identity>
tsp	^[\x01-\x13\x16-]\x01.?.?.?.?.?.?.?.?.?[-~]+
uucp	^\x10here=
validcertssl	^(.?.?\x16\x03.*\x16\x03|.?.?\x01\x03\x01?.*\x0b).*(thawte|equifax secure|rsa data security, inc|verisign, inc|gte cybertrust root|entrust\.net limited)
ventrilo	^..?v\\xcf
vnc	^rfb 00[1-9]\.00[0-9]\x0a
whois	^[!-~]+\x0d\x0a
worldofwarcraft	^\x06\xec\x01
x11	userspace flags=REG_NOSUB
xboxlive	^\x58\x80.....\xf3|\x06\x58\x4e
xunlei	^([()]|get)(...?.?.?(reg|get|query)|.+User-Agent: (Mozilla/4\.0 \(compatible; (MSIE 6\.0; Windows NT 5\.1;? ?\)|MSIE 5\.00; Windows 98\)))|Keep-Alive\x0d\x0a\x0d\x0a[7]
yahoo	^(ymsg|ypns|yhoo).?.?.?.?.?.?[lwt].*\xc0\x80
zmaap	^\x1b\xd7\x3b\x48[\x01\x02]\x01?\x01